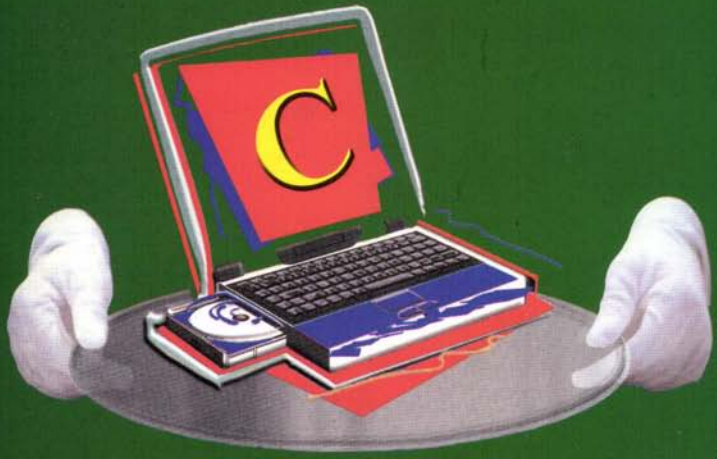
ویراست سوم

تألیف:
مهندس عین‌الله جعفرنژادقمی

برنامه‌نویسی به زبان C



مرجع کامل

فهرست مطالب

پیشگفتار

مدت‌ها است که زبان C به عنوان یک زبان برنامه‌سازی قدرتمند، در دانشگاه‌ها، صنایع و تجارت مورد بهره‌برداری قرار می‌گیرد. وجود منابع آموزشی غنی C ، کمک شایانی به پیشرفت علم انفورماتیک در این مراکز است. ویرایش اول این کتاب که ۱۶ بار به چاپ رسیده است، مرجع کاملی در زبان C بود که مورد اقبال اساتید محترم، دانشجویان و سایر علاقه‌مندان قرار گرفت. ویرایش دوم این کتاب به چاپ بیست و ششم رسیده است. اکنون در ویرایش سوم کتاب به سر می‌بریم. در این ویرایش، تغییرات اساسی به وجود آمده است تا تدریس آن توسط اساتید و فراگیری آن توسط دانشجویان آسانتر شود.

نگارنده معتقد است هنر این کتاب این‌است‌که در فصل‌های اولیه، با ارائه مثال‌های ساده و روان، مفاهیم زبان C را آموزش می‌دهد. ذکر مثال‌های پیچیده، همزمان با آموزش ساختارهای زبان، از میزان بهره گیری از درس می‌کاهد، زیرا دانشجو بیش از آن‌که به ساختارهای زبان توجه کند، به حل مسئلهٔ پیچیده معطوف خواهد شد. پس از بررسی ساختارهای زبان، کاربردهای عمده زبان C در طراحی و برنامه‌نویسی مورد بررسی قرار می‌گیرد. در کتاب، موضوعات بسیار گسترده‌ای بحث شده‌اند، و الگوریتم‌ها، توابع و روش‌های بسیار ارزشمندی برای برنامه‌نویسی در C ارائه گردیدند. مهندسی نرم‌افزار در C ، طراحی به کمک کامپیوتر (گرافیک)، طراحی منوها، فشرده‌سازی و رمزگذاری داده‌ها، مدیریت صفحه کلید، مدل‌های حافظه و ارتباط با دستگاه‌های جانبی از جمله موضوعات مهمی هستند که در کتاب مطرح شده، برنامه‌های مفیدی نوشته شده‌اند.

فصل‌های اول تا هشتم کتاب باید به طور سلسله مراتب و به همین ترتیب مطالعه شوند. یعنی هر یک از این فصل‌ها پیشنیاز یکدیگرند. پس از مطالعه این فصل‌ها، خواننده می‌تواند هر فصل دلخواهی را از این کتاب مطالعه کند و از مطالب آن بهره‌برداری نماید.

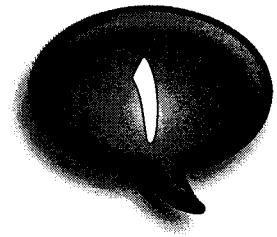
در ویرایش‌های دوم و سوم کتاب، از نظرات اساتید محترمی که چندین سال این کتاب را تدریس نموده‌اند، استفاده شده است. دانشجویان نیز از نقاط مختلف کشور، طی مکاتباتی، نظرات خود را منعکس کردند که سعی شد حتی‌الامکان نظرات آنها نیز منظور شود.

تمام برنامه‌های کتاب، در محیط بورلند C و توربو C قابل اجرایند. همهٔ برنامه‌ها با کامپیوتر تست شده‌اند تا از اشتباهات احتمالی در آنها جلوگیری به عمل آید. با تمام تلاشی که انجام شد تا کتاب عاری از اشتباه باشد، باز هم ممکن‌است، اشتباهاتی در آن وجود داشته باشد. از اساتید محترم و دانشجویان انتظار می‌رود، همچون گذشته، نظرات خود را منعکس نمایند تا در ویرایش‌های بعدی مورد توجه قرار گیرد. فرصت را غنیمت شمرده، پیشاپیش، از کسانی که نویسنده را از نظرات خود بهره‌مند می‌کنند، تشکر به‌عمل می‌آورم.

برنامه‌های کتاب بر روی دیسکتی موجود است که علاقه‌مندان می‌توانند از طریق تماس با نویسنده، برنامه‌ها را به طور رایگان دریافت نمایند. تلفن تماس ۳۲۶۰۷۷۲ - ۰۱۱۱

عین اله جعفرنژاد قمی

بابل - ۱۳۸۱



مقدمات زبان C

زبان C در سال ۱۹۷۲ توسط دنیس ریچی طراحی شد. این زبان تکامل‌یافتهٔ زبان BCPL می‌باشد که طراح آن مارتین ریچاردز است. زبان BCPL از زبان B که طراح آن کِن تامپسون می‌باشد، نتیجه شده است. علت نامگذاری C این‌است‌که بعد از B طراحی شد.

کسانی که تا حدودی با زبانهای برنامه‌سازی آشنایی دارند، می‌دانند که زبان دیگری به نام زبان C++ وجود دارد و آن از C ناشی شده است. C++ علاوه بر ویژگیهای C، ویژگیهای جدیدی دارد که در C موجود نیست. در کتاب حاضر، زبان برنامه‌نویسی C مورد بررسی قرار می‌گیرد. در این فصل، بعضی از عناصر زبان C را مورد بحث قرار می‌دهیم. بعضی از ویژگیهای زبان C عبارت‌انداز:

- زبان C یک زبان میانی است. زبانهای برنامه‌سازی را می‌توان به سه دسته تقسیم کرد: **زبانهای سطح بالا**، **زبانهای میانی**، **زبانهای سطح پایین** (جدول ۱-۱). علت میانی بودن زبان C این‌است‌که، از طرفی همانند زبان سطح پایینی مثل اسمبلی قادر است مستقیماً به حافظه دستیابی داشته باشد و با مفاهیم بیت، بایت و آدرس کار کند و از طرف دیگر، برنامه‌های این زبان، همچون زبانهای سطح بالایی مثل پاسکال، از قابلیت خوانایی بالایی برخوردارند. به عبارت دیگر، دستورالعملهای این زبان، به زبان محاوره‌ای انسان نزدیک است، که این ویژگی، مربوط به زبانهای سطح بالا است.

- زبان C، یک زبان ساخت یافته است. در این زبان با استفاده از حلقه‌های تکراری مثل while، for و do while می‌توان برنامه‌هایی نوشت که قابلیت خوانایی و درک آنها بالا باشد. بعضی از زبانهای ساخت‌یافته در جدول ۲-۱ آمده‌اند.

جدول ۱-۱ سطوح زبانهای برنامه‌سازی.

زبانهای سطح بالا	زبانهای میانی	زبانهای سطح پایین
پاسکال اِدا ماجولا-۲ کوبول بیسیک	جاوا فورث C، (C++)	ماکرواسمبلر اسمبلر

جدول ۲-۱ زبانها از نظر ساخت‌یافتگی.

زبانهای ساخت‌یافته	زبانهای غیرساخت‌یافته
پاسکال اِدا C، (C++) ماجولا-۲ جاوا	فرترن بیسیک کوبول

جدول ۳-۱ کلمات کلیدی زبان C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- زبان C، قابل انعطاف و بسیار قدرتمند است. در این زبان، هیچ محدودیتی برای برنامه‌نویس وجود ندارد. هر آنچه را که فکر می‌کنید، می‌توانید در این زبان پیاده‌سازی کنید.

- C، زبان برنامه‌نویسی سیستم است. برنامه‌های سیستم، برنامه‌هایی هستند که امکان بهره‌برداری از سخت‌افزار و سایر نرم‌افزارها را فراهم می‌کنند. بعضی از برنامه‌های سیستم عبارت‌انداز: سیستم عامل، مفسر[1]، کامپایلر، ویراستارها، واژه‌پردازها، مدیریت بانکهای اطلاعاتی و اسمبلر.

- ارتباط تنگاتنگی بین زبان C و اسمبلی وجود دارد و به‌این‌ترتیب می‌توان از تمام قابلیتهای اسمبلی در زبان C استفاده کرد. چگونگی برقراری ارتباط بین این دو زبان، در فصل ۲۱ به طور مفصل مورد بحث قرار می‌گیرد.

- C، زبان قابل حمل است. معنای قابلیت حمل این‌است‌که برنامه‌هایی که به زبان C، در یک نوع کامپیوتر (مثل آی.بی.ام) نوشته شدند، بدون انجام تغییرات یا انجام تغییراتِ اندک، در کامپیوترهای دیگر (مثل VAX و DEC) قابل استفاده‌اند.

- C، زبان کوچکی است. تعداد **کلمات کلیدی**[2] این زبان انگشت‌شمار است (۳۰کلمه کلیدی - جدول ۳-۱). تصور نشود که هرچه تعداد کلمات کلیدی زبان بیشتر باشد، آن زبان قدرتمند است. به عنوان مثال، زبان بیسیک در حدود ۱۵۰ کلمه کلیدی دارد ولی قدرت زبان C به مراتب بیشتر از زبان بیسیک است. توجه داشته باشید که بعضی از کامپایلرهای C، علاوه براین ۳۲کلمه کلیدی، کلمات دیگری را به زبان اضافه کرده‌اند (جدول ۴-۱).

- C نسبت به حروف حساس است[3]. یعنی در این زبان، بین حروف کوچک و بزرگ تفاوت است و تمام کلمات کلیدی این زبان با حروف کوچک نوشته می‌شوند. به عنوان مثال، while یک کلمه کلیدی است ولی WHILE اینطور نیست. توصیه می‌شود که تمام برنامه‌های C با حروف کوچک نوشته شوند.

جدول ۴-۱ کلمات کلیدی C که بعضی از کامپایلرها اضافه کردند.

asm	_cs	_ds	_es
_ss	cdecl	far	huge
interrupt	near	pascal	

1. interpreter 2. key words 3. case sensitive

■ دستورالعملهای برنامهٔ C دارای ویژگیهای زیر هستند:

۱. هر دستور زبان C به ; ختم می‌شود.
۲. حداکثر طول یک دستور، ۲۵۵ کاراکتر است.
۳. هر دستور می‌تواند در یک یا چند سطر ادامه داشته باشد.
۴. در هر سطر می‌توان چند دستور را تایپ کرد (این کار، توصیه نمی‌شود).
۵. توضیحات می‌توانند در بین */ و /* قرار گیرند و یا بعد از // ظاهر شوند:

```
/* This is a sample comment */
// This is another sample comment
```

انواع داده‌ها

هدف از برنامه‌نویسی، ورود داده‌ها به کامپیوتر، پردازش داده‌ها و استخراج نتایج است. لذا، داده‌ها نقش مهمی را در برنامه‌نویسی ایفا می‌کنند. یکی از جنبه‌های زبانهای برنامه‌سازی که باید دقیقاً مورد بررسی قرار گیرد، انواع داده‌هایی است که آن زبان با آنها سروکار دارد. در زبان C، پنج نوع داده وجود دارند که عبارتند از: **char**، **int**، **float**، **double** و **void**. نوع char برای ذخیره داده‌های کاراکتری مثل 'a'، 'b'، 'x' به کار می‌رود. نوع int برای ذخیره اعداد صحیح مثل 125، 430، 1650 به کار می‌رود. نوع float برای ذخیره اعداد اعشاری مثل 15.5، 175.5 و 1250.25 به کار می‌رود و نوع double برای ذخیره اعداد اعشاری که بزرگتر از float باشند مورد استفاده واقع می‌شود. نوع void را در جای مناسبی تشریح خواهیم کرد. هر یک از انواع داده‌های char، int، float و double مقادیری را می‌پذیرند که ممکن است از پردازنده‌ای (CPU) به پردازنده دیگر متفاوت باشد. به عنوان مثال، طول نوع int در محیطهای ۱۶ بیتی مثل DOS یا ویندوز ۳/۱، شانزده بیت و در محیطهای ۳۲ بیتی مثل ویندوز NT، سی و دو بیت است. بنابراین، اگر برنامه‌هایی می‌نویسید که باید در محیطهای مختلف اجرا شوند، سعی کنید از کوچکترین مقدار انواع در C استفاده نمایید.

جدول ۵-۱ انواع داده‌ها و مقادیر قابل قبول آنها.

نوع	اندازه به بیت	بازه قابل قبول
char	۸	۱۲۷- تا ۱۲۷
unsigned char	۸	۰ تا ۲۵۵
signed char	۸	۱۲۷- تا ۱۲۷
int	۱۶ یا ۳۲	۳۲۷۶۷- تا ۳۲۷۶۷
unsigned int	۱۶ یا ۳۲	۰ تا ۶۵۵۳۵
signed int	۱۶ یا ۳۲	۳۲۷۶۷- تا ۳۲۷۶۷
short int	۱۶	۳۲۷۶۷- تا ۳۲۷۶۷
unsigned short int	۱۶	۰ تا ۶۵۵۳۵
signed short int	۱۶	۳۲۷۶۷- تا ۳۲۷۶۷
long int	۳۲	۲۱۴۷۴۸۳۶۴۷- تا ۲۱۴۷۴۸۳۶۴۷
signed long int	۳۲	۲۱۴۷۴۸۳۶۴۷- تا ۲۱۴۷۴۸۳۶۴۷
unsigned int	۳۲	۰ تا ۴۲۹۴۹۶۷۲۹۵
float	۳۲	۷ رقم دقت (ارقام بعد از اعشار) (تقریباً 10^{-38} تا 10^{38})
double	۶۴	۱۵ رقم دقت (تقریباً 10^{-308} تا 10^{308})
long double	۸۰	۱۹ رقم دقت (تقریباً 10^{-4932} تا 10^{4932})

با استفاده از کلماتی مثل signed (با علامت)، unsigned (بدون علامت)، long و short می‌توان انواع جدیدی را ایجاد کرد. کلمات signed ، short ، long و unsigned را می‌توان با انواع int به کار برد. نوع char را می‌توان با signed و unsigned به کار برد. long به همراه double نیز قابل استفاده است. چون داده‌های نوع int با علامت هستند، کاربرد signed با آنها، بی‌مورد است. انواع مختلف داده‌ها و مقادیری که هر یک از انواع پشتیبانی می‌کنند، در جدول ۵-۱ آمده است.

متغیرها

متغیر نامی برای کلمات حافظه است که داده‌ها در آنها قرار می‌گیرند و ممکن‌است در طول اجرای برنامه تغییر کنند. برای مراجعه به متغیرها از نامشان استفاده می‌شود. لذا متغیرها امکان نامگذاری برای کلمات حافظه را فراهم می‌کنند. برای نامگذاری متغیرها می‌توان از ترکیبی از حروف a تا z یا A تا Z، ارقام و خط ربط (_) استفاده کرد، به طوری که اولین کاراکتر آنها رقم نباشد. نام متغیر می‌تواند با هر طولی باشد ولی ۳۱کاراکتر اولِ آن مورد استفاده قرار می‌گیرد. بعضی از اسامی مجاز و غیرمجاز برای متغیرها در جدول ۶-۱ آمده‌اند.

تعریف متغیرها

همانطور که گفته شد،متغیرها محل ذخیره داده‌ها هستند و چون داده‌ها دارای نوع‌اند، متغیرها نیز باید دارای نوع باشند. به عبارت دیگر، متغیرهای فاقد نوع، در C شناخته شده نیستند. قبل از به کارگرفتن متغیرها، باید نوع آنها را مشخص کرد. نوع متغیر، مقادیری راکه متغیر می‌تواند بپذیرد و اعمالی راکه می‌توانند بر روی آن مقادیر انجام شوند، مشخص می‌کند. تعیین نوع متغیر را **تعریف متغیر** گویند. برای تعیین نوع متغیر، به صورت زیر عمل می‌شود:

نام متغیر ; نوع داده

در این شکل کلی، نوع داده، یکی از انواع موجود در جدول ۵-۱ است. برای تعیینِ نوع بیش از یک متغیر، باید آنها را باکاما از هم جدا کرد.

مثال ۱-۱

تعریف متغیرهای x و y از نوع int، متغیرهای m و n از نوع float، متغیرهای ch1 و ch2 از نوع char، متغیر d1 از نوع double و متغیر p1 از نوع long int.

```
int       x, y ;
float     m, n ;
char      ch1, ch2 ;
double    d1 ;
long int  p1 ;
```

جدول ۶-۱ بعضی از اسامی مجاز و غیرمجاز برای متغیرها.

اسامی مجاز	اسامی غیرمجاز
count	1test
test23	high!there
sum	grade.1
S_1	.pcx

مقداردادن به متغیرها

برای مقداردادن به متغیرها به سه روش می‌توان عمل کرد:

۱. هنگام تعریف (تعیین نوع) متغیر
۲. پس از تعریف نوع متغیر و با دستور انتساب (=)
۳. دستورات ورودی

مثال ۲-۱

مقداردادن به متغیرها در هنگام تعریف آنها.

```
int       x, y = 5 ;
char      ch1 = 'a' , ch2 = 'm';
```

دستور اول، دو متغیر x و y را از نوع int تعریف می‌کند و مقدار متغیر y را برابر با ۵ قرار می‌دهد. دستور دوم متغیرهای ch1 و ch2 را از نوع char تعریف می‌کند، مقدار ch1 را برابر با 'a'، و مقدار ch2 را برابر با 'm' تعیین می‌کند. کاراکترها در داخل کوتیشن یکانی(') قرار می‌گیرند.

مثال ۳-۱

مقدار دادن به متغیرها با دستور انتساب.

```
int       x, y, m ;
float     f1, f2 ;
char      ch1, ch2 ;
f1 = 15.5 ;
f2 = 20.25 ;
x = y = m = 0 ;
ch1 = ch2 = 'a';
```

سه دستور اول، متغیرها را تعریف می‌کنند. دستور چهارم، مقدار 15.5 را در f1 و دستور پنجم مقدار 20.25 را در متغیر f2 قرار می‌دهد. دستور ششم سه متغیر x و y و m را برابر با صفر قرار می‌دهد (انتساب چندگانه در C امکان‌پذیر است). دستور هفتم، حرف 'a' را در متغیرهای ch1 و ch2 قرار می‌دهد.

مثال ۴-۱

مقدار دادن به متغیرها با دستورات ورودی.

```
int       x, y ;
scanf ("%d%d",&x,&y) ;
```

توجه داشته باشید که در اینجا، متغیرهای x و y از نوع int تعریف شدند و دستور scanf مقادیر آنها را از طریق صفحه‌کلید دریافت می‌کند. فعلاً به چگونگی عملکرد آن کاری نداشته باشید، بلکه فقط این مطلب را یاد بگیرید که، متغیرها با دستورات ورودی، مقدار می‌گیرند.

در مورد تعیین نوع متغیرها این نکته را به خاطر داشته باشید: نوع متغیرها را برحسب نیاز تعریف کنید. به عنوان مثال، اگر تعریف متغیری مثل x از نوع int، جوابگوی نیاز شما است، آن را از نوع float، double یا long int تعریف نکنید.

تعریف ثوابت

ثوابت مقادیری هستند که در برنامه وجود دارند ولی قابل تغییر نیستند. برای تعریف ثوابت به دو روش عمل می‌شود: ۱. استفاده از دستور define# و ۲. استفاده از دستور const. برای تعریف ثوابت از طریق دستور define# به صورت زیر عمل می‌شود:

#define <نام ثابت> <مقدار>

نامگذاری برای ثوابت از قانون نامگذاری برای متغیرها تبعیت می‌کند. مقداری که برای ثابت تعیین می‌شود، نوع ثابت را نیز مشخص می‌کند. دقت داشته باشید که در انتهای دستور define# علامت ; قرار نمی‌گیرد. علتش این‌است‌که این دستور، از دستورات پیش پردازنده[1] است، نه دستور زبان C. **پیش پردازنده** یک برنامه سیستم است که قبل از ترجمهٔ برنامه توسط کامپایلر، تغییراتی در آن ایجاد می‌کند. پیش پردازنده مقدار ثابت را که در دستور define# آمده است، به جای نام ثابت در برنامه قرار می‌دهد و این دستور در زمان اجرا وجود ندارد. نام دیگر ثوابتی که به‌این‌صورت تعریف می‌شوند، ماکرو[2] است که در ادامه کتاب مورد بحث قرار می‌گیرد. برای تفکیک اینگونه ثوابت از متغیرهای برنامه، بهتر است نام آنها با حروف بزرگ انتخاب شود.

مثال ۵-۱

تعریف ثوابت PI و M با دستور define# . دستور اول، مقدار M را برابر با ۱۰۰ و دستور دوم مقدار PI را برابر با ۳/۱۴ تعیین می‌کند.

```
#define  M   100
#define  PI  3.14
```

برای تعریف ثوابت با دستور const به صورت زیر عمل می‌شود:

const <نوع داده> <نام ثابت> = <مقدار>

در این شکل کلی، نوع داده، یکی از انواع موجود در جدول ۵-۱ است. نام ثابت مثل نام متغیرها انتخاب می‌شود که مقدار ثابت با علامت = در آن قرار می‌گیرد.

مثال ۶-۱

تعیین ثوابت n و count از نوع int و با مقادیر 100 و 50، و ثابت x از نوع signed char و با مقدار 'x'.

```
const   int  n = 100 , int    count = 50 ;
const signed  char    x = 'a' ;
```

اگر پس از تعریف ثوابت، در ادامه برنامه سعی کنید مقادیر آنها را عوض کنید، کامپایلر خطایی را به شما اعلان خواهد کرد. لذا با توجه به تعاریف مثال ۶-۱، دستورات زیر نادرست‌اند:

```
n = 150 ;
x = 'b' ;
count = 200 ;
```

1. preprocessor 2. macro

کاربرد ثوابت در برنامه، از امتیازات خاصی برخوردار است. به عنوان مثال، اگر بخواهید در هر بار اجرای برنامه، مقدار n را که در مثال ۶-۱ آمده است عوض کنید، لازم نیست، کلیه دستورات برنامه را که با n سروکار دارند تغییر دهید، بلکه کافی است، مقدار n را در دستور const به مقدار مورد نظر تغییر دهید.

عملگرها

عملگرها[1] نمادهایی هستند که اعمال خاصی را انجام می‌دهند. به عنوان مثال، نماد '+' عملگری است که دو مقدار را با هم جمع می‌کند (عمل جمع را انجام می‌دهد). پس از تعریف متغیرها و مقداردادن به آنها باید بتوان عملیاتی را روی آنها انجام داد. برای انجام این عملیات باید از عملگرها استفاده کرد. عملگرها در زبان C به چند دسته تقسیم می‌شوند: ۱. عملگرهای محاسباتی ۲. عملگرهای رابطه‌ای ۳. عملگرهای منطقی ۴. عملگرهای بیتی. عملگرها بر روی یک یا دو مقدار عمل می‌کنند. مقادیری را که عملگرها بر روی آنها عمل می‌کنند، **عملوند**[2] گویند. به عنوان مثال، در a + 5، متغیر a و مقدار 5 را عملوندهای عملگر + گویند.

عملگرهای محاسباتی

عملگرهای محاسباتی، عملگرهایی هستند که اعمال محاسباتی را روی عملوندها انجام می‌دهند. این عملگرها در جدول ۷-۱ مشاهده می‌شوند. هر یک از عملگرهای -، +، *، / تقریباً در همه زبانها وجود دارد. عملگر % برای محاسبه باقیمانده تقسیم به کار می‌رود. این عملگر، عملوند اول را بر عملوند دوم تقسیم می‌کند (تقسیم صحیح) و باقیمانده را برمی‌گرداند. عملگر **کاهش** (--) یک واحد از عملوندش کم می‌کند و نتیجه را در آن عملوند قرار می‌دهد و عملگر **افزایش** (++) یک واحد به عملوندش اضافه کرده نتیجه را در آن عملوند قرار می‌دهد. دستورات زیر را در نظر بگیرید:

```
int  x = 10,  m = 10 ;
x ++ ;
++ m ;
```

متغیرهای x و m با مقدار اولیه ۱۰ تعریف می‌شوند. دستور ; ++ x یک واحد به x اضافه می‌کند و نتیجه را در x قرار می‌دهد و دستور ; m ++ یک واحد به m اضافه کرده نتیجه را در m قرار می‌دهد. بنابراین در این نوع دستورات، عملگر افزایش (یا کاهش) چه قبل از عملوند باشند و چه بعد از آن، عملکرد آنها یکسان است. اما عملکرد آنها در عبارات محاسباتی با هم متفاوت است.

جدول ۷-۱ عملگرهای محاسباتی.

عملگر	نام	مثال
-	تفریق و منهای یکانی	x - y یا -x
+	جمع	x + y
*	ضرب	x * y
/	تقسیم	x / y
%	باقیمانده تقسیم	x % y
--	کاهش (decrement)	x -- یا -- x
++	افزایش (increment)	x ++ یا ++ x

1. operators 2. operand

عبارات محاسباتی

عبارات[1] ترکیبی از متغیرها، ثوابت و عملگرها هستند. به عنوان مثال، x + y، 4 + x / 5، 6 * 2 / m + 4 همگی عبارات محاسباتی‌اند. اگر عملگرهای ++ و -- در عبارات محاسباتی، قبل از عملوند قرار گیرند، ابتدا این عملگرها عمل کرده، نتیجه آن در محاسبات شرکت می‌کند ولی اگر بعد از عملوند ظاهر شوند، مقدار فعلی عملوند مورد استفاده قرار گرفته سپس عملگر بر روی عملوند عمل می‌کند. دستورات زیر را در نظر بگیرید:

```
int   x, y ;
x = 10 ;
y = ++ x ;
```

با اجرای دستور دوم مقدار ۱۰ در x قرار می‌گیرد و با اجرای دستور سوم، ابتدا یک واحد به x اضافه می‌شود و سپس مقدار حاصل در متغیر y قرار می‌گیرد. لذا، y برابر با ۱۱ خواهد شد. اکنون دستورات زیر را در نظر بگیرید:

```
int   x, y ;
x = 10 ;
y = x ++ ;
```

با اجرای دستور دوم، مقدار ۱۰ در x قرار می‌گیرد و با اجرای دستور سوم، مقدار فعلی x در y قرار می‌گیرد و سپس یک واحد به x اضافه می‌شود. لذا، مقدار y برابر با ۱۰ ولی مقدار x برابر با ۱۱ خواهد شد. برای آشنایی بیشتر با این عملگرها، دستورات زیر را در نظر بگیرید:

```
int  x, y, m ;
x = 10 ;
y = 15 ;
m = ++x + y++ ;
```

در اجرای دستور چهارم، ابتدا به مقدار فعلی x (۱۰) یک واحد اضافه می‌شود تا به ۱۱ تبدیل شود. مقدار جدید x با مقدار فعلی y (۱۵) جمع می‌شود و نتیجه آن، یعنی ۲۶ در m قرار می‌گیرد. در پایان، مقدار y برابر با ۱۶ خواهد شد.

تقدم عملگرها

وقتی در عبارتی، چندین عملگر وجود داشته باشند، ترتیب اجرای عملگرها چگونه خواهد بود؟ برای پاسخ به این سئوال، باید **تقدم عملگرها** را مشخص کرد. برای پی بردن به این مسئله، دستورات زیر را در نظر بگیرید:

```
int   m, x = 6 , y = 10 ;
m = x + y / 2 * 3 ;
```

به نظر شما، حاصل این عبارت چه خواهد بود؟ اگر ابتدا x با y جمع شود و حاصل آن بر ۲ تقسیم شود و نتیجه آن در ۳ ضرب شود، حاصل عبارت **۲۴** خواهد بود. اگر ابتدا y بر ۲ تقسیم شود و حاصل آن با x جمع شود و نتیجه آن در ۳ ضرب شود، حاصل آن **۳۳** خواهد بود. اگر ابتدا y بر ۲ تقسیم شود و نتیجه آن در ۳ ضرب شود، حاصل آن با x جمع

1. expressions

شود، حاصل عبارت، ۲۱ خواهد بود. کدام مقدار درست است؟ برای یافتن پاسخ درست، باید تقدم عملگرها را بدانیم. تقدم عملگرهای محاسباتی در جدول ۸-۱ آمده است. همانطور که در این جدول مشاهده می‌کنید، بالاترین تقدم مربوط به عملگرهای افزایش و کاهش و کمترین تقدم مربوط به عملگرهای + و -است. عملگرهایی که تقدم آنها یکسان است، مثل + ، - یا % ، * و / نسبت به هم **تقدم مکانی** دارند. یعنی، هر کدام که زودتر ظاهر شد، همان عملگر زودتر انجام می‌شود.

با توجه به مطالب گفته شده، در عبارت m = x + y / 2 * 3 ابتدا عملگر /و سپس عملگر * و در انتها عملگر +انجام می‌شود. به‌این‌ترتیب حاصل این عبارت برابر با ۲۱ است.

عملگرهای رابطه‌ای

عملگرهای رابطه‌ای، ارتباط بین عملوندها را مشخص می‌کنند. اعمالی مثل تساوی دو مقدار، کوچکتر یا بزرگتر بودن، مقایسه با صفر، و غیره، توسط عملگرهای رابطه‌ای مشخص می‌شود. عملگرهای رابطه‌ای در جدول ۹-۱ آمده‌اند. در مورد عملگرهای رابطه‌ای، شاید با عملگر == آشنایی نداشته باشید. این عملگر در دستورات شرطی برای مقایسه دو مقدار مورد استفاده قرار می‌گیرد. به عنوان مثال، دستور مقایسه دو مقدار x و y، باید به صورت آیا x==y است نوشته شود.

جدول ۸-۱ تقدم عملگرهای محاسباتی.

بالاترین تقدم	++ --
	-(منهای یکانی)
	* / %
پایین‌ترین تقدم	+ -

جدول ۹-۱ عملگرهای رابطه‌ای.

عملگر	نام	مثال
>	بزرگتر	x > y
>=	بزرگتر یا مساوی	x >= y
<	کوچکتر	x < y
<=	کوچکتر یا مساوی	x <= y
==	متساوی	x == y
!=	نامساوی	x != y

جدول ۱۰-۱ عملگرهای منطقی به ترتیب تقدم.

عملگر	نام	مثال
!	نقیض (not)	!x
&&	و (and)	x > y && m <p
||	یا (or)	x > y || m < p

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند. عبارات منطقی دارای دو ارزش **درستی** و **نادرستی** اند. در زبان C، ارزش **نادرستی** با مقدار صفر و ارزش **درستی** با مقادیر غیر صفر مشخص می‌شود. عملگرهای منطقی در جدول ۱۰-۱ آمده‌اند. ترتیب قرار گرفتن آنها در این جدول از تقدم بالا به پایین است.

نتیجه عملگر ! وقتی درست است که عملوند آن دارای ارزش نادرستی باشد. نتیجه عملگر && وقتی درست است که هر دو عملوند ارزش درستی داشته باشند و نتیجه عملگر || وقتی نادرست است که هر دو عملوند ارزش نادرستی داشته باشند (در بقیه موارد نتیجه آن ارزش درستی دارد). به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
int  x, y, m, p, q ;
x = 0 ;
y = 1 ;
m = x && y ;
p = x || y ;
q = ! x ;
```

با اجرای دستور چهارم، ارزش دارای m برابر با نادرستی خواهد بود، زیرا x دارای ارزش نادرستی و y دارای ارزش درستی است و حاصل بر && بر روی آنها، نادرستی است. با اجرای دستور پنجم، p ارزش درستی خواهد شد، زیرا قرار x دارای ارزش نادرستی است و y دارای ارزش درستی است و عملگر || روی آنها عمل می‌کند و ارزش درستی را برمی‌گرداند. چون x ارزش نادرستی دارد، x! دارای ارزش درستی خواهد بود که نتیجه آن در q قرار می‌گیرد.

چون اغلب، عملگرهای منطقی و رابطه‌ای با هم مورد استفاده قرار می‌گیرند درک تقدم آنها از اهمیت ویژه‌ای برخوردار است. تقدم آنها را در جدول ۱۱-۱ مشاهده کنید.

عملگرهای ترکیبی

از ترکیب عملگرهای محاسباتی و علامت =، مجموعه دیگری از عملگرها ایجاد می‌شود که عمل محاسباتی و انتساب را انجام دهند. این عملگرها در جدول ۱۲-۱ آمده‌اند. تقدم این عملگرها پایین‌تر از سایر عملگرها است.

جدول ۱۱-۱ تقدم عملگرهای منطقی و رابطه‌ای.

بالاترین	!
	> >= < <=
	== !=
	&&
پایین‌ترین	||

جدول ۱۲-۱ عملگرهای ترکیبی.

عملگر	نام	مثال	معادل
+=	انتساب جمع	x += y	x = x + y
-=	انتساب تفریق	x -= y	x = x - y
*=	انتساب ضرب	x *= y	x = x * y
/=	انتساب تقسیم	x /= y	x = x / y
%=	انتساب باقیمانده تقسیم	x %= y	x = x % y

عملگرهای بیتی

وجود عملگرهای بیتی در C موجب شد تا بسیاری از کارهای زبان اسمبلی در C انجام شود. عملگرهای بیتی برای تست کردن، مقدار دادن یا شیفت دادن و سایر اعمال بر روی مقادیری که در یک بایت (char) یا کلمه (int) ذخیره شده‌اند به کار می‌روند. عملگرهای بیتی را نمی‌توان با انواع float، double، long double، و void یا سایر انواع پیچیده به کار برد. عملگرهای بیتی در جدول ۱۳-۱ آمده‌اند.

با فرض اینکه p و q دو بیت از داده‌ها باشند، عملکرد هر یک از عملگرهای &، |، ^ و ~ در جدول ۱۴-۱ آمده است. نتیجه عملگر & وقتی یک است که هر دو بیت یک باشند. نتیجه عملگر | وقتی صفر است که هر دو بیت صفر باشند. نتیجهٔ عملگر ^ وقتی یک است که یکی از بیتها صفر و دیگری یک باشد.

عملگرهای شیفت، بر روی یک عملوند عمل می‌کند و بیتهای آن را به سمت راست یا چپ شیفت می‌دهند. این عملگرها به صورت زیر به کار می‌روند:

تعداد شیفت >> متغیر

تعداد شیفت << متغیر

جدول ۱۳-۱ عملگرهای بیتی.

عملگر	نام	
&	و	(AND)
|	یا	(OR)
^	یای انحصاری	(XOR)
~	نقیض	(Not)
>>	شیفت به راست	(right shift)
<<	شیفت به چپ	(left shift)

جدول ۱۴-۱ عملکرد عملگرهای بیتی.

x	y	x & y	x | y	x ^ y	~ x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

جدول ۱۵-۱ عملکرد عملگرهای شیفت.

unsigned char x ;	مقدار دودویی x	مقدار دهدهی x
x = 7 ;	00000111	7
x = x << 1	00001110	14
x = x << 3	01110000	112
x = x << 2	11000000	192
x = x >> 1	01100000	96
x = x >> 2	00011000	24

در این عملگرها، متغیر، یک بایت یا کلمه‌ای از حافظه است که باید به تعداد معین شده، به سمت چپ یا راست شیفت داده شوند. هنگام شیفت دادن به راست، بیتها از سمت راست خارج می‌شوند و از سمت چپ به تعداد موردنظر، صفر وارد می‌شود. در شیفت به چپ، بیتها از سمت چپ خارج شده، به تعداد لازم، صفر از سمت راست وارد می‌شود.

برای آشنایی با عملگرهای شیفت، جدول ۱۵-۱ را ببینید. در این جدول، مقداردهدهی و دودویی x، قبل و بعد از شیفت دادن نشان داده شده است. همانطور که در این جدول مشاهده می‌شود، هر شیفت به چپ معادل ضرب کردن در ۲ است، مثل، اطلاعات x = x << 1 که عدد ۷ را به ۱۴ تبدیل می‌کند و دستور x << 3 نیز عدد ۱۴ را به ۱۱۲=۱۴x۸ تبدیل می‌کند. توجه داشته باشید که پس از دستور x << 2 اطلاعات واقعی از بین می‌روند و در نتیجه روند ضرب در ۲ شدن در اینجا صدق نمی‌کند. هر شیفت به راست نیز معادل تقسیم بر ۲ است. دستور x = x >> 1 ، عدد ۱۹۲ را بر ۲ تقسیم کرده، ۹۶ به دست می‌آید و دستور x = x >> 2 عدد ۹۶ را بر ۴ تقسیم کرده، عدد ۲۴ به‌دست می‌آید.

مثال ۷-۱ نمونه‌ای از عملکرد عملگرهای بیتی.

10000000 |	11000001 &
00000011	01111111
10000011	01000001
01111111	01111111 ^
~	01111000
10000000	00000111

همانطور که دیدید عملگرهای بیتی برای دستکاری بیتها مورد استفاده قرار می‌گیرند. خواندن بایتهای وضعیت دستگاههایی مثل چاپگر و مودم و تست وضعیت آنها، از جمله موارد کاربرد عملگرهای بیتی است. در ادامه کتاب، سعی می‌شود با ارائه مثالهای کاربردی، شما را با عملکرد این عملگرها آشنا کنیم.

عملگرهای & و *

همانطور که گفته شد، متغیرها نامی برای کلمات حافظه‌اند و کلمات حافظه نیز دارای شماره ردیف می‌باشند که ما آنها را **آدرس** می‌نامیم. با استفاده از عملگر & می‌توانیم به آدرس متغیرها دسترسی داشته باشیم. عملگر * نیز برای دسترسی غیر مستقیم به حافظه مورد استفاده قرار می‌گیرد. به عنوان مثال، اگر فرض کنیم آدرس متغیر x در p قرار داشته باشد، از طریق آدرس x و با استفاده از عملگر * می‌توانیم به محتویات x دسترسی پیدا کنیم. با فرض اینکه p می‌تواند حاوی آدرس x باشد، دستورات زیر را در نظر بگیرید:

p = & x ;	آدرس x در p قرار می‌گیرد.
* p = 5 ;	جایی که آدرس آن در p است (همان x)، برابر با ۵ می‌شود.
m = * p ;	محتویات جایی که آدرسش در p است (۵) در m قرار می‌گیرد.

این عملگرها، عملگرهای مربوط به اشاره‌گرها هستند که در فصل ۶ با جزئیات بیشتری شرح داده خواهد شد.

عملگر ؟

این عملگر، عبارتی را ارزیابی کرده، براساس ارزش آن عبارت (درستی یا نادرستی)، نتیجه عبارت دیگر را در متغیری قرار می‌دهد:

<عبارت ۳> : <عبارت ۲> ? <عبارت ۱> = متغیر

اگر <عبارت ۱> دارای ارزش درستی باشد، مقدار ارزیابی شدهٔ <عبارت ۲> در متغیر قرار می‌گیرد وگرنه مقدار ارزیابی شدهٔ <عبارت ۳> در متغیر قرار خواهد گرفت. دستورات زیر را در نظر بگیرید:

```
int  x , y ;
x = 5 ;
y = x > 5   ?   x * 2 : x * 5 ;
```

در این دستورات، موارد زیر را داریم:

<عبارت ۱> : x > 5
<عبارت ۲> : x * 2
<عبارت ۳> : x * 5

<عبارت ۱> دارای ارزش نادرستی است، زیرا x از ۵ بزرگتر نیست. بنابراین مقدار <عبارت ۳> که برابر با ۲۵ = ۵ × ۵ است، در متغیر y قرار می‌گیرد.

عملگر کاما (،)

این عملگر برای انجام چند عمل در یک دستور به‌کار می‌رود و روش کاربرد آن به صورت زیر است:

متغیر = (<عبارت ۱> , <عبارت ۲>);

<عبارت ۱> به نحوی با <عبارت ۲> در ارتباط است. به‌طوری‌که، ابتدا <عبارت ۱> ارزیابی می‌شود و <عبارت ۲> می‌تواند از نتیجه آن استفاده کند و حاصل <عبارت ۲> در متغیر قرار می‌گیرد. دستورات زیر را در نظر بگیرید:

```
int  x , y ;
y = (x = 2 , x * 4 / 2) ;
```

در این دستورات، موارد ذیل را داریم:

<عبارت ۱> : x = 2
<عبارت ۲> : x * 4 / 2

در <عبارت ۱> مقدار ۲ در x قرار می‌گیرد و این مقدار در محاسبه عبارت x * 4 / 2 شرکت کرده، حاصل آن، یعنی ۴ در y قرار می‌گیرد.

عملگر sizeof

این عملگر، یک عملگر زمان ترجمه است و می‌تواند طول یک متغیر یا نوع داده را برحسب بایت تعیین کند. اگر با کامپیوتری کار می‌کنید و نمی‌دانید انواع آن، مثلاً نوع int چند بایتی است، با این عملگر می‌توانید به آن پی ببرید. این عملگر به صورتهای زیر به کار می‌رود:

```
sizeof    متغیر ;
sizeof    (نوع داده) ;
```

همانطور که در نحوه کاربرد این عملگر می‌بینید، وقتی برای تعیین طول نوع داده به کار می‌رود، نوع داده باید در داخل پرانتز قرار گیرد. دستورات زیر را در نظر بگیرید:

```
int  x , y , m ;
x = sizeof  y ;
m = sizeof (float) ;
```

دستور اول، سه متغیر x و y و m را از نوع صحیح تعریف می‌کند. دستور دوم طول متغیر y را محاسبه کرده در x قرار می‌دهد و دستور سوم، طول نوع float را محاسبه کرده در m قرار می‌دهد. بعداً که روش چاپ متغیرها را یاد گرفتید، می‌توانید با چاپ متغیرهای x و m، به طول متغیر y و نوع float پی ببرید.

عملگر ()

پرانتزها عملگرهایی هستند که تقدم عملگرهای داخل خود را بالا می‌برند. به عنوان مثال، عبارت زیر را در نظر بگیرید:

```
y = 4 * 2 / (3+1) + (6 + (7-2))
```

جدول ۱۶-۱ تقدم عملگرها در حالت کلی.

()	بالا ترین تقدم
! ~ ++ -- sizeof	
* / %	
+ -	
<< >>	
< <= > >=	
== !=	
&	
^	
|	
&&	
||	
?	
= += -= *= /= %=	پایین‌ترین تقدم

برای ارزیابی این عبارت، باید ابتدا عبارت موجود در داخلی‌ترین پرانتز را ارزیابی کرد. در این عبارت، ترتیب انجام عملیات به‌این صورت است: ابتدا عدد ۴ در ۲ ضرب می‌شود که حاصل آن ۸ است. بعد از ۲، عملگر تقسیم قرار دارد که عملوند بعدی آن در داخل پرانتز است. لذا تقدم عملگر + موجود در آن، از تقدم عملگر تقسیم بیشتر می‌شود و در نتیجه ۳ با یک جمع شده، حاصل آن ۴ است. اکنون، مقدار ۸ (که قبلاً محاسبه شد) بر ۴ تقسیم می‌شود و حاصل آن ۲ است. عملگر بعدی، + است ولی بعد از آن پرانتز آمده است. چون تقدم عملگرهای موجود در پرانتز بالاتر است، ابتدا ۲ از ۷ کم می‌شود که مقدار آن ۵ است و ۵ با ۶ جمع می‌شود که حاصل آن ۱۱ است و با مقدار ۲ که از قبل محاسبه شد جمع می‌شود و حاصل عبارت ۱۳ است که در y قرار می‌گیرد.

تقدم عملگرها در حالت کلی

تاکنون عملگرهای موجود در C را مطالعه کردیم. اگر انواع مختلفی از عملگرها با هم در عبارتی به کار گرفته شوند، برای ارزیابی عبارت باید ترتیب اجرای آنها را بدانیم. به همین منظور، تقدم عملگرها در حالت کلی، در جدول ۱۶-۱ آمده است.

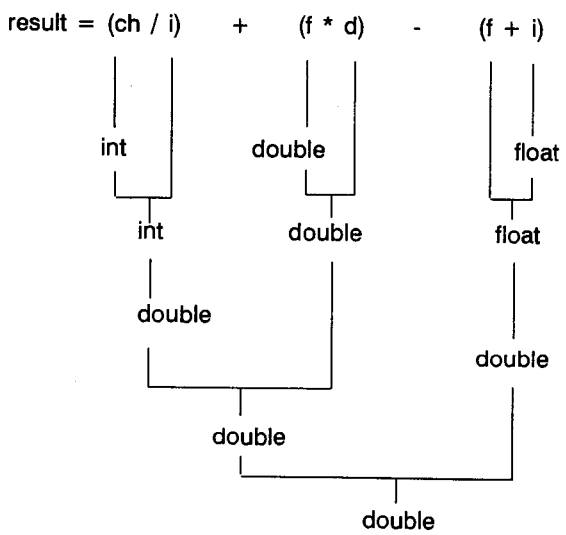
تبدیل انواع

در زبان C انواع مختلف داده‌ها می‌توانند به یکدیگر تبدیل شوند. تبدیل انواع از دو جهت قابل بررسی است: ۱. تبدیل انواع در عبارات محاسباتی و ۲. تبدیل انواع در احکام انتساب. در مورد تبدیل انواع در عبارات محاسباتی، این قانون حاکم است که انواع کوچکتر به انواع بزرگتر تبدیل می‌شوند:

- اگر یکی از عملوندها long double باشد، عملوند دیگر به long double تبدیل می‌شود.
- وگرنه، اگر یکی از عملوندها double باشد، عملوند دیگر به double تبدیل می‌شود.
- وگرنه، اگر یکی از عملوندها float باشد، عملوند دیگر به float تبدیل می‌شود.
- وگرنه، اگر یکی از عملوندها unsigned long باشد، عملوند دیگر به unsigned long تبدیل می‌شود.
- وگرنه، اگر یکی از عملوندها long باشد، عملوند دیگر به long تبدیل می‌شود.
- وگرنه، اگر یکی از عملوندها unsigned int باشد، عملوند دیگر به unsigned int تبدیل می‌شود.

توجه داشته باشید که اگر یکی از عملوندها از نوع long و دیگری از نوع unsigned int باشد، ولی مقدار unsigned int نتواند توسط long نمایش داده شود، هر دو عملوند به unsigned long تبدیل می‌شوند. دستورات زیر چگونگی ارزیابی عبارت و تبدیل انواع در عبارات محاسباتی را تشریح می‌کنند. همان‌طور که در ارزیابی این عبارت مشاهده می‌کنید، نوع نتیجه عبارت، double است.

```
char    ch ;
int     i ;
float   f ;
double  d ;
```



تبدیل انواع در احکام انتساب وقتی اتفاق می‌افتد که دو متغیر از انواع مختلف به یکدیگر نسبت داده شوند. به عنوان مثال، اگر متغیری از نوع char به متغیری از نوع int نسبت داده شوند، با تبدیل انواع در احکام انتساب سروکار داریم. دستورات زیر را در نظر بگیرید:

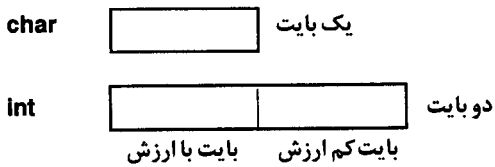
```
int x ;
char ch ;
float f ;
...
ch = x ;
x = f ;
f = ch ;
f = x ;
```

همان‌طوری که در این احکام انتساب می‌بینید، انواع کاراکتری، اعشاری و دقت مضاعف به یکدیگر نسبت داده می‌شوند. هر کدام از این انواع به انواع دیگری قابل تبدیل است ولی باید دقت داشته باشید که در تبدیل انواع، بخشی از اطلاعات ممکن است از بین برود. به عنوان مثال، در انتساب یک مقدار اعشاری به یک مقدار صحیح، حداقل مقداری که از بین می‌رود، بخش اعشاری عدد است. تبدیل انواع مختلف به یکدیگر و اطلاعاتی که ممکن است از بین برود، در جدول ۱۷-۱ آمده‌اند.

جدول ۱۷-۱ تبدیل انواع در احکام انتساب.

نوع منبع	نوع مقصد	اطلاعاتی که ممکن است از بین برود
char	signed char	اگر مقدار بیش از ۱۲۷ باشد، مقصد منفی خواهد شد
short int	char	۸ بیت با ارزش
int (۱۶ بیتی)	char	۸ بیت با ارزش
int (۳۲ بیتی)	char	۲۴ بیت با ارزش
long int	char	۲۴ بیت با ارزش
int (۱۶ بیتی)	short int	اطلاعات از بین نمی‌رود
int (۳۲ بیتی)	short int	۱۶ بیت با ارزش
long int	int (۱۶ بیتی)	۱۶ بیت با ارزش
long int	int (۳۲ بیتی)	اطلاعات از بین نمی‌رود
float	int	بخش کسری و یا بیشتر
double	float	دقت علائم کم می‌شود و نتیجه گرد می‌شود
long double	double	دقت علائم کم می‌شود و نتیجه گرد می‌شود

همان‌طور که در این جدول مشاهده می‌کنید، تبدیل نوع double به int وجود ندارد. برای چنین تبدیلی، ابتدا باید double را به float و سپس float را به int تبدیل کنید. برای مشاهده چگونگی از بین رفتن اطلاعات در تبدیل انواع، انتساب int به char را در نظر می‌گیریم:



داده‌های کاراکتری در یک بایت و داده‌های صحیح در دو بایت ذخیره می‌شوند. اگر داده‌های ذخیره شده در متغیر صحیح، به اندازه‌ای کوچک باشد که در بایت کم ارزش قرار گیرد، در انتقال مقادیر صحیح به کاراکتری، اطلاعاتی را از دست نخواهیم داد. ولی اگر مقدار موجود در متغیر صحیح، در دو بایت ذخیره شده باشد، بخشی از عدد که در بایت باارزش آن قرار دارد، از بین می‌رود. لذا برنامه‌نویس باید در تبدیل انواع دقت کافی به خرج دهد تا اطلاعات مفید و ارزشمند خود را از دست ندهد.

روش ایجاد برنامه

تاکنون با مقدمات زبان C آشنا شدید و آماده می‌شوید که در فصل ۲ برنامه‌های ساده ای را بنویسید. اما قبل از آن، خوب است که با روش ایجاد برنامه در C آشناشوید. برنامه نویسی، نوعی حل مسئله است. اگر مسئله‌ها را به‌راحتی حل می‌کنید، برنامه‌نویس موفقی می‌شوید. یکی از اهداف کتاب این‌است‌که توانایی شما را در حل مسئله بالا ببرد. در C می‌توان برنامه‌های **ساخت‌یافته** نوشت. برنامه‌های ساخت‌یافته برنامه‌هایی هستند که قابلیت خوانایی آنها بالا است، درک آنها آسان است، و نگهداری آنها مشکل نیست. برنامه‌نویسی ساخت‌یافته، برنامه‌نویسی استانداردی است. برای نوشتن برنامه در C، باید مواردِ زیر را در نظر بگیرید:

۱. تعیین نیازمندیهای مسئله
۲. تحلیل مسئله
۳. طراحی الگوریتم حل مسئله
۴. پیاده‌سازی الگوریتم
۵. تست و کنترل برنامه
۶. نگهداری و نوسازی برنامه

تعیین نیازمندیهای مسئله

تعیین نیازمندیهای مسئله موجب می‌شود تا مسئله را به وضوح و بدون هیچ ابهامی بشناسید و چیزهایی را که برای حل مسئله لازم است، درک کنید. هدف این‌است‌که جنبه‌های بی‌اهمیت مسئله را نادیده بگیرید و به موارد اصلی بپردازید. این کار، چندان ساده نیست. ممکن‌است لازم باشد با کسی که مسئله را طرح کرد، مذاکراتی داشته باشید.

تحلیل مسئله

تحلیل مسئله، شامل تعیین **ورودیها** و **خروجیها** و سایر نیازمندیها یا محدودیتهای حل مسئله است. ورودیها، داده‌هایی هستند که مسئله باید بر روی آنها کار کند و خروجیها، نتایج مورد انتظارِ مسئله‌اند. در این مرحله، فرمت خروجی اطلاعات باید مشخص شود، متغیرها باید تعریف شوند و رابطه بین آنها باید مشخص گردد. ارتباط بین متغیرها ممکن‌است با فرمول خاصی بیان شود.

اگر مراحل ۱ و ۲ به خوبی انجام نشوند، مسئله به درستی حل نخواهد شد. صورت مسئله را به‌دقت بخوانید تا اولاً ایدهٔ روشنی از مسئله پیدا کنید و ثانیاً ورودیها و خروجیها را تعیین نمایید. ممکن‌است با خواندن دقیقِ صورت مسئله، به این مطلب پی ببرید که بسیاری از ورودیها و خروجیها، در صورتِ مسئله وجود دارند. جملات زیر را که مسئله‌ای را بیان می‌کنند در نظر بگیرید:

دانشجویی می‌خواهد با دانستن **طول و عرض** مستطیل، **مساحت** آن را حساب کند.

ورودی‌ها و خروجی‌های مسئله با استفاده از کلماتی که پررنگ شده‌اند مشخص می‌گردد:

ورودی‌های مسئله

- طول مستطیل.
- عرض مستطیل.

خروجی مسئله

- مساحت مستطیل.

وقتی ورودی‌ها و خروجی‌های مسئله مشخص شد، باید فرمول‌هایی تدوین شود که رابطه بین آن‌ها را مشخص کند. فرمول محاسبه مساحت مستطیل به صورت زیر است:

عرض مستطیل × طول مستطیل = مساحت مستطیل

در بعضی از موارد ممکن‌است رابطه بین ورودیها و خروجیها به این سادگی مشخص نشود و نیاز به فرضها و تسهیلات خاصی باشد. فرایند استخراج متغیرهای مسئله و تعیین روابط بین آنها از طریق صورت مسئله، **انتزاع**[1] نام دارد.

طراحی الگوریتم

در طراحی الگوریتم برای حل مسئله، لازم است قدم‌به‌قدم **رویه‌هایی**[2] نوشته شوند-**الگوریتم**- و سپس بررسی شود که آیا الگوریتم، مسئله را به درستی حل می‌کند یا خیر. نوشتن الگوریتم، مشکل‌ترین بخش حل مسئله است. سعی نکنید تمام جزئیات مسئله را حل کنید، بلکه سعی کنید شیوه طراحی **بالا به پایین**[3] را به کار ببرید. در روش طراحی بالا به پایین، ابتدا مراحل اصلی مسئله که باید حل شوند، مشخص می‌گردند و سپس با حل هر مرحله اصلی، کل مسئله حل می‌شود. اغلب الگوریتمها معمولاً مراحل زیر را دارا هستند:

۱. خواندن داده‌ها ۲. انجام محاسبات ۳. چاپ نتایج

وقتی مراحل مهم مسئله مشخص شدند، می‌توانید هر مرحله را به‌طور جداگانه حل کنید. به عنوان مثال، مرحله انجام محاسبات ممکن‌است به بخشهای کوچکتری تقسیم شود. این عمل را **بهینه‌سازی الگوریتم** گویند.

اگر هنگام نوشتن مقاله‌ای، از عناوین مهم استفاده کرده باشید، ممکن‌است با موضوع طراحی بالابه‌پایین آشنایی داشته باشید. ابتدا باید لیستی از عناوین مهم تهیه کنید و سپس با نوشتن شرحی برای هر عنوان، آنها را بهینه‌سازی کنید. پس از نوشتن این شرح، متن هر موضوع را می‌نویسید.

یکی از مراحل طراحی الگوریتم، **کنترل** الگوریتم است. برای کنترل الگوریتم، هر مرحله از الگوریتم را مانند کامپیوتر اجرا کنید و ببینید که آیا الگوریتم مطابق خواستهٔ شما عمل می‌کند یا خیر. اگر خطاهای الگوریتم در این مرحله (مرحله حل مسئله) رفع شوند، موجب صرفه‌جویی در وقت می‌شود.

پیاده‌سازی الگوریتم

در پیاده‌سازی الگوریتم، باید الگوریتم را به برنامه تبدیل کرد. هر مرحله از الگوریتم باید به یک یا چند دستور زبان برنامه‌سازی تبدیل شود. یکی از کارهای مهم این مرحله مشخص کردن فایل‌های سرآیند[4] است که باید به برنامه اضافه شوند. با این فایل‌ها در فصل ۲ آشنا می‌شوید.

---

1. abstraction 2. procedures 3. top down design 4. header

تست برنامه

در تست و بررسی برنامه، باید کل برنامه را تست کنید تا مشخص شود که آیا خواسته شما را برآورده می‌کند یا خیر. در تست برنامه، باید آن را برای داده‌های مختلفی چند بار اجرا کنید و خروجی‌های برنامه را بررسی کنید.

نگهداری برنامه

نگهداری و **نوسازی**[1] برنامه شامل اصلاح برنامه جهت حذف خطاهای قبلی و نوسازی آن جهت پاسخگویی به نیازهای فعلی است. بعضی از سازمانها بعد از اینکه نویسنده برنامه‌ای به جایی دیگر منتقل شد، برنامه‌های آن را تا ۵ سال یا بیشتر نگهداری می‌کنند، ولی به تدریج آن را از دور خارج می‌کنند.

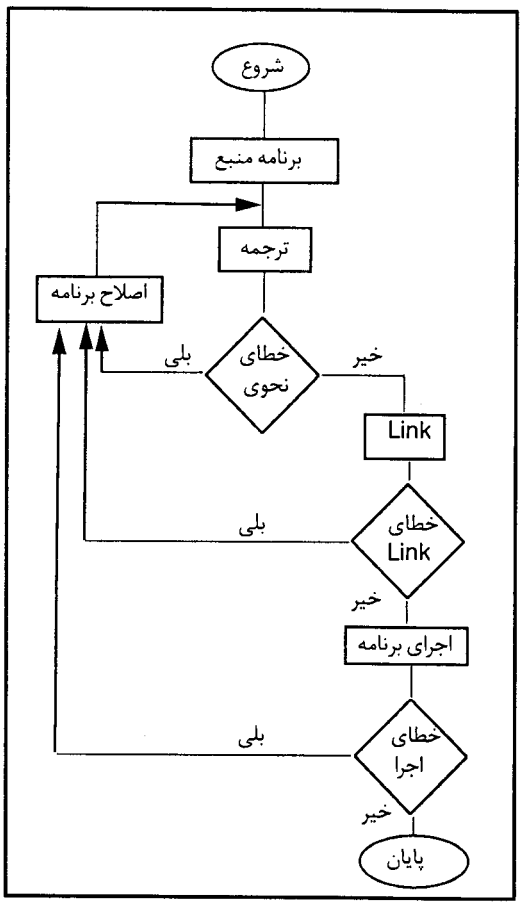
فرآیند آماده‌سازی و اجرای برنامه

پس از اینکه برنامه‌تان را نوشتید، باید آن را وارد کامپیوتر کرده، ترجمه، پیوند و اجرا نمایید. لذا فرآیند آماده‌سازی و اجرای برنامه شامل موارد زیر است:

۱. واردکردن برنامه در یک محیط ویراستاری و ذخیره کردن بر روی دیسک. فایل برنامه بر روی دیسک، در کامپایلرهای C++ با پسوند cpp. و در کامپایلرهای توربو C با پسوند c. ذخیره می‌شود.

۲. ترجمه برنامه جهت اشکالزدایی آن. اگر برنامه **اشکالات نحوی**[2] داشته باشد، باید آنها را برطرف کرده و برنامه را دوباره ترجمه کنید. این مرحله را آنقدر انجام دهید تا کلیه اشکالات برنامه برطرف شوند.

۳. پس از ترجمه برنامه، فایلی با پسوند obj ایجاد می‌گردد که به زبان ماشین است ولی قابل اجرا نیست. علتش این است که هنوز بخش‌های مختلف برنامه به هم پیوند نخورده‌اند و آدرس‌دهی آنها کامل نیست.



شکل ۱-۱ روند آماده‌سازی و اجرای برنامه

1. update 2. syntax error

۴. پس از ترجمه برنامه، باید عمل **پیوند**[1] را انجام دهید. در این مرحله، فایلی با پسوند exe تشکیل می‌شود که قابل اجرا است.

با توجه به مطالب گفته شده، مراحل آماده‌سازی و اجرای برنامه را می‌توان مانند فلوچارت شکل ۱-۱ رسم کرد. دقت داشته باشید که در محیط‌های برنامه‌نویسی C++، تمام این موارد مهیا است و هنگامی که برنامه را ترجمه می‌کنید، در صورت عدم وجود خطا، مرحله پیوند زدن و اجرا نیز قابل انجام است.

تمرینات

۱. ویژگیهای مهم زبان C را توصیف کنید.

۲. ضرورت وجود توضیحات را در برنامه بیان کرده، روش اعلان توضیحات را در C تشریح کنید.

۳. هدف از انواع داده چیست؟ انواع داده‌ها را در C تشریح کنید.

۴. تعریف متغیرها و چگونگی مقداردهی به آنها را بیان کنید.

۵. ثوابت را تعریف کرده، چگونگی تعیین آنها را در C بیان کنید. به نظر شما استفاده از ثوابت در برنامه چه امتیازاتی دارد؟

۶. دستوراتی بنویسید که متغیرهای x و y را از نوع int، ch و d را از نوع double و ثابت k را از نوع unsigned int و با مقدار اولیه ۲۰ تعریف کند.

۷. منظور از عملگر چیست؟ به نظر شما کاربرد عملگرهای ++ و -- چه امتیازاتی دارد؟

۸. منظور از تقدم عملگرها چیست؟ مثال بزنید.

۹. مثالی بزنید که در آن از عملگرهای بیتی استفاده شده باشد؟

۱۰. نقش عملگر ? را بیان کنید.

۱۱. مثالهایی از تبدیل انواع در احکام انتساب و تبدیل انواع در عبارات محاسباتی ارائه دهید.

۱۲. قوانین حاکم بر تبدیل انواع در عبارات محاسباتی را بیان کنید.

۱۳. تشریح کنید که در چه مواقعی ممکن است در تبدیل انواع در احکام انتساب، اطلاعات از بین برود؟

۱۴. فرمولهای زیر را به صورت عباراتی در C بنویسید.

$$1.\ x = \frac{y * 2}{m + p} \qquad 2.\ y = x + m^2 - \frac{k}{r + 2}$$

۱۵. با توجه به مقادیر تعیین شده، هر یک از عبارات زیر را ارزیابی کنید:

```
x = 8  ,  y = 10  ,  m = 6
k = x / 4 * (y / 2) * m
k = x / y ++ + y / -- m
```

۱۶. با توجه به مقادیر زیر مقدار y چند است؟

```
x = 8 , m = 6
y = x * 2 < m + 4   ?   4 * m : 8 *  m
```

۱۷. مراحل ایجاد برنامه را تشریح کنید.

۱۸. برای مسئلهٔ زیر مراحل ایجاد برنامه را طی کنید:

دانش آموزی می‌خواهد **معدل ۵ نمره** خود را محاسبه کند.

---

1. link

ساختار برنامهٔ C و ورودی - خروجی

پس از اینکه مقدمات زبان C را در فصل اول آموختید، آمادگی دارید تا در این فصل با ساختار برنامه در C آشنا شده، با یادگیری شیوهٔ ورودی - خروجی داده‌ها، برنامه‌های ساده‌ای بنویسید. اما قبل از پرداختن به امکانات ورودی - خروجی زبان C، ساختار یک برنامه را در این زبان تشریح می‌کنیم. ساختاری که در این فصل بررسی می‌شود، ساختار ساده‌ای است که به تدریج با آموختن امکانات دیگری در C، آن را تکمیل خواهیم کرد.

برنامه‌های زبان C، از مجموعه‌ای از دستورات و تعدادی تابع تشکیل می‌شود (شکل ۱-۲). هر تابع، برای حل بخشی از مسئله نوشته می‌شود و دارای نام است. نامگذاری برای توابع، از قانون نامگذاری برای متغیرها تبعیت می‌کند. در این فصل نمی‌خواهیم راجع‌به توابع بحث کنیم بلکه می‌خواهیم متذکر شویم که بدنه اصلی برنامه، تابعی به نام ()main است. به عبارت دیگر، هیچ برنامه‌ای وجود ندارد که فاقد این تابع باشد. بنابراین، تابع ()main یکی از اجزای مهم برنامه زبان C است.

علاوه بر تابع ()main، توابع دیگری از قبل نوشته شده همراه کامپایلر زبان C ارائه می‌شوند و به وفور در برنامه‌ها مورد استفاده قرار می‌گیرند. در واقع، بسیاری از اعمال در برنامه‌نویسی با C، توسط این توابع از پیش نوشته شده انجام می‌شوند. به عنوان مثال، ورود داده‌ها از صفحه کلید و نوشتن آنها در صفحه‌نمایش، در C توسط توابعی انجام می‌شود که از قبل نوشته شدند و همراه کامپایلر وجود دارند. بنابراین قبل از پرداختن به برنامه‌نویسی، باید اطلاع داشته باشیم که این توابع در کجا وجود دارند و برنامه، چگونه به آنها دسترسی پیدا می‌کند. این توابع و سایر اطلاعاتی که کامپایلر برای ترجمه برنامه به آنها نیاز دارد، در تعدادی از فایل‌ها به نام فایل‌های سرآیند[1] قرار دارند.

پسوند این فایل‌ها h. است و معمولاً بر روی فهرستی از دیسک به نام INCLUDE قرار دارند (آنها را در نرم‌افزار C خود مشاهده کنید). بنابراین باید راهی برای اتصال این فایل‌ها به برنامه وجود داشته باشد. برای این کار، باید بدانیم هر تابع مورد استفاده در برنامه، در کدام فایل سرآیند قرار دارد و همان فایل را به برنامه اضافه کنیم. به عنوان مثال، تابع ()scanf که برای ورود داده‌ها از صفحه کلید مورد استفاده قرار می‌گیرد، در فایلی به نام stdio.h قرار دارد.

```
#include <فایل سرآیند>
int main()
{
    اعلان متغیرها
    دستورات اجرایی
    return 0;
}
```

شکل ۱-۲ ساختار ساده‌ای از برنامه C.

1. header

با توجه به توضیحاتی که ارائه شد، مشخص گردید که بخشی از برنامه C باید کار **اتصال** فایل‌های سرآیند را به برنامه انجام دهد. برای این منظور از دستوری به نام **#include** استفاده می‌شود. این دستور که از دستورات **پیش پردازنده** است، معمولاً قبل از تابع ()main قرار می‌گیرد. به طور کلی، باید به این نکته توجه داشته باشید که، پیش‌پردازنده، مترجمی است که با مشاهده دستوراتی که با # شروع می‌شوند، اجرا می‌شود و آنها را به دستورات زبان C تبدیل می‌کند. توجه داشته باشید که این دستور به ; ختم نمی‌شود. نحوه کاربرد این دستور به صورت زیر است:

#include < نام فایل سرآیند >

به عنوان مثال، برای اضافه کردن فایل stdio.h به برنامه، به صورت زیر عمل می‌شود:

#include <stdio.h>

در مورد دستور #include به نکات زیر توجه کنید:

- بین # و include نباید فاصله‌ای وجود داشته باشد.
- بین نام فایل و علائم > و < نباید فاصله‌ای وجود داشته باشد.
- ذکر علائم > و < ضروری است.

در مورد دستورات پیش پردازنده، در فصل ۲۲ به طور کامل بحث خواهد شد. ولی در اینجا به دلیل ضرورت، مختصری راجع به #include بحث شده است. نکته دیگری که در مورد برنامه C باید بدانید این است که، سیستم عامل که اجرا کننده تابع ()main است، می‌خواهد بداند که این تابع با موفقیت به پایان رسیده است یا خیر. برای این منظور بهتر است در پایان اجرای برنامه، مقداری به سیستم عامل برگردانده شود تا سیستم عامل متوجه گردد که برنامه با موفقیت به پایان رسیده است. به همین دلیل، آخرین دستور برنامه C، دستور **return 0;** است. این دستور مقدار صفر را به سیستم عامل برمی‌گرداند و سیستم‌عامل متوجه می‌شود که برنامه با موفقیت اجرا شده است. البته وجود این دستور در بعضی از کامپایلرها ضروری، نیست ولی بعضی دیگر از کامپایلرها آن را ضروری می‌دانند. لذا عادت کنید که این دستور را در انتهای برنامه به کار ببرید.

مقداری که به سیستم عامل برگردانده می‌شود، باید از نوع int باشد. لذا تابع ()main که می‌خواهد مقداری از نوع int را برگرداند، خودش باید از نوع int باشد، به همین دلیل تابع ()main را به صورت زیر در برنامه مورد استفاده قرار می‌دهیم:

int main()

توجه به این نکته نیز ضروری است که در بعضی از گونه‌های زبان C، برنامه می‌تواند مقداری را به سیستم عامل برنگرداند. در این صورت باید نوع تابع ()main را void تعریف کرد:

void main()

اگر به‌این صورت عمل کنید،دستور **return 0;** را نباید به‌کار ببرید.

چند نکته راجع به شکل ۱-۲ باید مورد توجه قرارگیرد:

۱. دستورالعملهای برنامه، با { شروع شده به } ختم می‌شوند.

۲. برای اضافه کردن چند فایل سرآیند به برنامه، برای هر کدام از آنها باید یک دستور #include گنجانده شود.

۳. متغیرهای موردنیاز برنامه، در ابتدای برنامه و بلافاصله پس از { تعریف می‌شوند.
۴. به تدریج که مطالب جدیدی از C می‌آموزید، ساختار برنامه در C کاملتر می‌شود.

اکنون که با ساختار ساده برنامه در C آشنا شدید، باید دستورالعملهای اجرایی C را بیاموزید تا بتوانید برنامه‌های ساده‌ای بنویسید.

ورودی - خروجی داده‌ها

اصولاً، امکانات ورودی - خروجی هر زبان برنامه‌سازی، نمایانگر میزان قابلیت آن زبان در طراحی فرمهای ورود داده‌ها و اخذ گزارش از برنامه است. در C روشهای گوناگونی برای ورودی - خروجی داده‌ها وجود دارد. بعضی از امکانات ورودی - خروجی این زبان را در این فصل و بعضی دیگر را در فصلهای آینده بررسی می‌کنیم. چون در نظر است آموزش C در این کتاب به صورت گام به گام و خودآموز باشد، سعی می‌کنیم از ذکر مفاهیم پیچیده در فصول اولیه کتاب جلوگیری کنیم.

چاپ اطلاعات با تابع printf()

تابع printf() که در فایل stdio.h قرار دارد، برای چاپ اطلاعات در صفحه‌نمایش به کار می‌رود. اگر این تابع با موفقیت اجرا شود، تعداد کاراکترهایی را که به خروجی منتقل شده‌اند برمی‌گرداند و در صورت بروز خطا، یک عدد منفی را برمی‌گرداند. نحوه کاربرد این تابع به صورت زیر است:

```
printf(" < عبارت ۱ > " , < عبارت ۲ >)
```

در این تابع، <عبارت ۲> اطلاعاتی است که باید به خروجی منتقل شوند و <عبارت ۱> می‌تواند شامل موارد زیر باشد:

۱. اطلاعاتی که باید عیناً در خروجی چاپ شوند.

۲. کاراکترهای تعیین کننده فرمت خروجی. این کاراکترها نوع اطلاعاتی را که در <عبارت ۲> ذکر شده‌اند و باید به خروجی بروند، مشخص می‌کنند. کاراکترهای فرمت با علامت % شروع می‌شوند. به عنوان مثال، %f برای چاپ اعداد اعشاری به کار می‌رود. کاراکترهای فرمت در جدول ۱-۲ آمده‌اند.

۳. کاراکترهای کنترلی. این کاراکترها شکل خروجی اطلاعات را مشخص می‌کنند. اینکه آیا تمام اطلاعات در یک سطر باشند یا در چند سطر چاپ شوند، آیا اطلاعات با فاصله خاصی از یکدیگر چاپ شوند یا خیر، و مواردی از این قبیل، توسط کاراکترهای کنترلی مشخص می‌شود. کاراکترهای کنترلی با \ شروع می‌شوند. به عنوان مثال \n موجب می‌شود تا سطر جاری (سطری که فعلاً در حال نوشتن در آن سطر هستیم) خرد شود و چاپ اطلاعات از سطر جدیدی آغاز گردد. کاراکترهای کنترلی در جدول ۲-۲ آمده‌اند.

توجه داشته باشید که در تابع printf()، <عبارت ۲> می‌تواند وجود نداشته باشد. به عبارت دیگر، تابع printf() به صورت زیر نیز قابل استفاده است:

```
printf (" <عبارت ۱> ")
```

جدول ۱-۲ کاراکترهای‌فرمت در دستور ()printf .

کاراکتر	نوع اطلاعاتی که باید به خروجی برود
%c	یک کاراکتر
%d	اعداد صحیح دهدهی مثبت و منفی
%i	اعداد صحیح دهدهی مثبت و منفی
%e	نمایش علمی عدد همراه با حرف e
%E	نمایش علمی عدد همراه با حرف E
%f	عدد اعشاری ممیز شناور
%g	اعداد اعشاری ممیز شناور
%G	اعداد اعشاری ممیز شناور
%o	اعداد مبنای ۸ مثبت
%s	رشته‌ای از کاراکترها (عبارت رشته‌ای)
%u	اعداد صحیح بدون علامت (مثبت)
%x	اعداد مبنای ۱۶ مثبت با حروف کوچک
%X	اعداد مبنای ۱۶ مثبت با حروف بزرگ
%p	Pointer (اشاره‌گر)
%n	موجب می‌شود تا تعداد کاراکترهایی که تا قبل از این کاراکتر به خروجی منتقل شده‌اند شمارش شده در پارامتر متناظر با آن قرار گیرد
%%	علامت ٪

جدول ۲-۲ کاراکترهای‌کنترلی در دستور ()printf .

کاراکتر	عملی که باید انجام شود
\f	موجب انتقال کنترل به صفحه جدید می‌شود
\n	موجب انتقال کنترل به خط جدید می‌شود
\t	انتقال به ۸ محل بعدی صفحه‌نمایش
\"	چاپ کوتیشن (")
\'	چاپ کوتیشن (')
\°	NULL (رشته تهی)
\\	back slash
\v	انتقال کنترل به ۸ سطر بعدی
\N	ثابت‌های مبنای ۸ (N عدد مبنای ۸ است)
\xN	ثابت‌های مبنای ۱۶ (N عدد مبنای ۱۶ است)

مثال ۱-۲

برنامه‌ای که چگونگی استفاده از تابع printf() را برای چاپ رشته نشان می‌دهد (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
int main()
{
  printf("C is a language.");
  return 0;
}
```

خروجی

C is a language.

تحلیل مثال ۱-۲

در این برنامه، دستور printf() طوری به کار برده شد که فاقد <عبارت ۲> است. در <عبارت ۱> نه کاراکتر ٪ وجود دارد و نه کاراکتر \. لذا هیچ کاراکتر فرمت و کنترلی در این عبارت وجود ندارد. بنابراین، آنچه که در داخل کوتیشن آمده است، عیناً در خروجی چاپ می‌شود.

مثال ۲-۲

برنامه‌ای که با تابع printf() دو رشته را به خروجی می‌برد (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
int main()
{
  printf("C is a language ");
  printf("and it is my favourite.");
  return 0;
}
```

خروجی

C is a language and it is my favourite.

تحلیل مثال ۲-۲

همان‌طور که در خروجی برنامه می‌بینید، هر دو دستور printf()، خروجی خود را در یک سطر تولید کرده‌اند. علتش این‌است‌که دستور printf() اول، سطری را که اطلاعات خود را نوشت، رد نمی‌کند و دستور printf() دوم بر روی همان سطر عمل می‌کند.

مثال ۳-۲

برنامه‌ای که با استفاده از کاراکتر کنترلی n\ سطر جاری را رد می‌کند.

```
#include <stdio.h>
int main()
{
  printf("C is a language\n ");
  printf("and it is my favourite.");
  return 0;
}
```

خروجی

```
C is a language
and it is my favourite.
```

تحلیل مثال ۳-۲

در این برنامه، وجود \n در انتهای دستور printf() اول، موجب شد تا سطری که اطلاعات در آن نوشته شد رد شود و دستور printf() دوم، اطلاعات را از ابتدای سطر جدید چاپ کند.

مثال ۴-۲

برنامه‌ای که اطلاعات کاراکتری، عددی صحیح و اعشاری و آدرس متغیر x را در خروجی چاپ می‌کند. هدف از این برنامه، آشنایی با کاراکترهای فرمت است (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
int main()
{
  int x = 10;
  float y = 15.5;
  char ch = 'a';
  printf("\n x=%d, y=%f, ch=%c", x, y, ch);
  printf("\n address of x is : %p", &x);
  return 0;
}
```

خروجی

```
x = 10 , y = 15.500000 , ch = a
address of x is : FFF4
```

تحلیل مثال ۴-۲

در این برنامه، در دستور printf() کاراکتر \n سطر جاری را رد می‌کند. کاراکترهای =x عیناً در خروجی چاپ می‌شوند و با فرمت %d مقدار x (۱۰) به خروجی می‌رود. سپس کاما (,) و کاراکترهای =y عیناً در خروجی چاپ می‌شوند و با فرمت %f مقدار y (15.5) در خروجی چاپ می‌شود. سپس کاما (,) و کاراکترهای =ch عیناً در خروجی چاپ می‌شود و با فرمت %c کاراکتر ch (حرف 'a') به خروجی می‌رود. در آخرین دستور printf()، آدرس متغیر x در خروجی چاپ شد که این آدرس ممکن است در کامپیوتر شما متفاوت باشد.

مثال ۵-۲

برنامه‌ای که با استفاده از کاراکترهای فرمت و کاراکترهای کنترلی، اطلاعاتی را به خروجی می‌برد.

توضیح:

در این برنامه، برای چاپ علامت ٪ از دو علامت ٪٪ استفاده شد و برای ایجاد فاصله بین x و y از کاراکتر کنترلی \t استفاده گردید. \t موجب شد تا y در محل نهم (ابتدای ۸ محل بعدی) چاپ شود.

```
#include <stdio.h>
int main()
{
  int x = 15, y = 20;
```

```
    printf("\nx=%%%d\ty=%%%d", x, y);
    return 0;
}
```

خروجی

x = %15 y = %20

مثال ۶-۲

برنامه‌ای که اعدادی را به صورت نماد علمی نمایش می‌دهد. همانطور که در این برنامه می‌بینید، برای چاپ اعداد با نماد علمی از کاراکترهای فرمت %e، %E و %g استفاده می‌شود.

```
#include <stdio.h>
int main()
{
  double d;
  d = 2e+007;
  printf("\nThe value of d is : %e", d);
  printf("\nThe value of d is : %E", d);
  printf("\nThe value of d is : %g", d);
  return 0;
}
```

خروجی

The value of d is : 2.000000e + 07
The value of d is : 2.000000E + 07
The value of d is : 2e + 07

مثال ۷-۲

برنامه‌ای برای نمایش چگونگی کاربرد کاراکتر فرمت %n در دستور printf().

توضیح:
کاراکتر فرمت %n مشخص می‌کند که تا قبل از رسیدن به این کاراکتر، چند کاراکتر در خروجی چاپ شده‌اند. این تعداد، در آدرس متغیری که در بخش <عبارت ۲> در دستور printf() آمده است قرار می‌گیرد. در این مثال، تعداد ۱۰ کاراکتر (با احتساب فضای خالی) به خروجی رفته‌اند. به همین دلیل مقدار ۱۰ در count قرار گرفت و چاپ شد.

```
#include <stdio.h>
int main()
{
   int count;
   printf("This is a %n test statement", &count);
   printf("\n count = %d", count);
   return 0;
}
```

خروجی

This is a test statement
count = 10

مثال ۸-۲

برنامه‌ای که با استفاده از کاراکترهای کنترلی، کوتیشن دوتایی را در خروجی چاپ می‌کند و به کمک کاراکتر کنترلی \t فاصله‌ای بین اطلاعات خروجی قرار می‌دهد.

```
#include <stdio.h>
int main()
{
    printf("\n\"each\"\t\"word\"\t\"is\"\n");
    printf("\n\"tabed\"\t\"over\"\t\"once\"");
   return 0;
}
```

خروجی

```
"each"  "word"  "is"
"tabed"  "over"  "once"
```

مشاهده صفحه خروجی برنامه

اگر برنامه‌هایی راکه تاکنون در این فصل نوشته شد، در کامپیوتر تان اجرا کرده باشید، دیدید که پس از تولید خروجی برنامه، کامپیوتر سریعاً به برنامه برمی‌گردد. لذا خروجی برنامه را نمی‌توانید مشاهده کنید. خروجی برنامه در صفحه‌ای غیر از صفحه‌ای که برنامه‌تان را تایپ می‌کنید تولید می‌شود. آن صفحه را **صفحه خروجی** می‌نامیم. اما خوب است که برنامه پس از تولید خروجی، در صفحه خروجی باقی بماند تا پس از مشاهدهٔ خروجی، برای برگشتن به برنامه، کلیدی فشار داده شود. برای این منظور می‌توانید از تابع getch() استفاده کنید (به مثال ۹-۲ مراجعه شود). این تابع منتظر فشردن کلیدی از صفحه کلید می‌ماند. الگوی این تابع در فایل conio.h قرار دارد. این تابع کاربرد دیگری نیز دارد که در ادامه بحث خواهد شد. توجه داشته باشید که اگر از تابع getch() استفاده نمی‌کنید، پس از اجرای برنامه می‌توانید با کلید ALT+F5 به صفحه خروجی بروید و پس از مشاهدهٔ اطلاعات، با فشردن کلید Enter، به صفحه برنامه برگردید.

پاک کردن صفحه خروجی

اگر چند برنامه را اجرا کنید و هر برنامه خروجی خاص خودش را تولید کند، صفحه خروجی حاوی اطلاعات متعددی خواهد شد. به طوری که به راحتی نمی‌توانید خروجی برنامه را مورد مطالعه و بررسی قرار دهید. بنابراین، بهتر است در هر بار اجرای برنامه، صفحه خروجی پاک شود. برای این منظور از تابع clrscr() به همین صورت استفاده می‌شود. این تابع در فایل conio.h قرار دارد (به مثال ۹-۲ مراجعه شود).

انتقال مکان‌نما در صفحه خروجی

گاهی ممکن‌است بخواهید مکان‌نما را در صفحه‌خروجی به محل خاصی منتقل کنید و اطلاعات را از آنجا دریافت و یا در آنجا چاپ کنید. به عنوان مثال، ممکن است بخواهید عدد x را از سطر ۵ و ستون ۱۰ بخوانید و در سطر ۶ و ستون ۱۰ چاپ کنید. برای این منظور از تابع gotoxy() استفاده می‌شود. الگوی این تابع در فایل conio.h قرار دارد و به صورت زیر است:

```
gotoxy(int x , int y) ;
```

x شماره ستون و y شماره سطری است که مکان‌نما به آنجا منتقل می‌شود. به عنوان مثال، دستور gotoxy(40, 10); مکان‌نما را به ستون ۴۰ و سطر ۱۰ منتقل می‌کند. (به مثال ۹-۲ مراجعه شود).

مثال ۹-۲

برنامه‌ای که دو مقدار عددی صحیح را تعریف کرده حاصل جمع آنها را به خروجی می‌برد. برنامه، قبل از تولید خروجی، صفحه خروجی را پاک می‌کند و پس از تولید خروجی در ستون ۲۰ و سطر ۱۰، در این صفحه منتظر می‌ماند تا کلیدی فشار داده شود. پس از مشاهده خروجی این برنامه، باید کلیدی را فشار دهید تا به صفحه برنامه برگردید.

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int m, x = 5, y = 10;
   clrscr();
   m = x + y;
   gotoxy(20, 10);
   printf("x=%d, y=%d, sum=%d", x, y, m);
   getch();
   return 0;
}
```

خروجی

```
x = 5 , y = 10 , sum = 15
```

چاپ اعداد نوع short و long

برای چاپ اطلاعات عددی از نوع short و long ، از کاراکترهای خاصی استفاده می‌شود. کاراکتر l (اِل) به همراه d برای چاپ مقادیر long و کاراکتر h به همراه d برای چاپ مقادیر short به کار می‌رود. ضمناً، کاراکترهای h و l را می‌توان با کاراکترهای فرمت d، i، o، و u نیز به کار برد.

مثال ۱۰-۲

برنامه‌ای که مقادیر long و short int را در خروجی نمایش می‌دهد.

```
#include <conio.h>
#include <stdio.h>
int main()
{
   short int x = 15;
   long  int m = 35789;
   clrscr();
   printf("\n x=%hd, m=%ld", x, m);
   getch();
   return 0;
}
```

خروجی

```
x = 15 , m = 35789
```

تعیین طول میدان در تابع ()printf

با استفاده از امکانات دیگری که در تابع ()printf وجود دارد، می‌توان مشخص کرد که هر کدام از اطلاعاتی که به خروجی می‌روند، چند بایت از فضای خروجی را اشغال کنند. فضایی را که هر قلم اطلاعات اشغال می‌کند، **طول میدان** خروجی گویند. طول میدان خروجی، معمولاً برای اعداد تعیین می‌شود و در تولید منظم خروجی و جدول‌بندی اطلاعات بسیار مفید است.

طول میدان مقادیر صحیح به صورت %wd بیان می‌شود که w طول میدان را مشخص می‌کنند. اگر طول میدان از تعداد ارقام عدد صحیح بیشتر باشد، عدد در سمت راست میدان قرار می‌گیرد و سمت چپ خالی می‌ماند. اما اگر طول میدان کمتر از تعداد ارقام عدد باشد، طول میدان نادیده گرفته شده تمام ارقام عدد در خروجی چاپ می‌شود.

طول میدان مقادیر اعشاری به صورت %w.df بیان می‌شود که در آن، w کل طول میدان و d تعداد ارقام اعشار است. به عنوان مثال، اگر برای چاپ عدد اعشاری، میدان %5.2f را اعلام کنیم، ۲ رقم برای قسمت اعشار (d=2)، دو رقم برای قسمت صحیح و یک محل به نقطه اعشار اختصاص می‌یابد. اگر طول میدان قسمت صحیح بیشتر از بخش صحیح عدد باشد، سمت چپ قسمت صحیح خالی باقی می‌ماند. ولی اگر طول میدان قسمت صحیح، کمتر از بخش صحیح عدد باشد، کل بخش صحیح در خروجی چاپ و طول میدان نادیده گرفته می‌شود. اگر طول میدان قسمت اعشار، از تعداد ارقام اعشار بیشتر باشد، ارقام اعشاری در سمت چپ میدان قرار گرفته سمت راست میدان خالی می‌ماند. اما اگر طول میدان قسمت اعشاری، از تعداد ارقام اعشاری کمتر باشد، قسمت اعشاری عدد، گرد می‌شود. هنگام گرد کردن، چنانچه رقم حذف‌شده، بزرگتر یا مساوی ۵ باشد، یک واحد به رقم سمت چپ آن اضافه می‌شود.

مثال ۱۱-۲

برنامه‌ای که چگونگی نمایش اعداد صحیح و اعشاری را با استفاده از طول میدان نشان می‌دهد (به تحلیل مثال توجه کنید).

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int x = 125, y = 1468;
   float m = 327.348, n = 4351.32;
   clrscr();
   printf(" x=%5d, y=%3d, sum=%d", x, y);
   printf("\n m=%7.2f, n=%6.2f", m, n);
   getch();
   return 0;
}
```

خروجی

```
x = □□ 125 , y = 1467

m =  |3|2|7|.|3|5| ,  n = |4|3|5|1|.|3|2|
```

تحلیل مثال ۱۱-۲

در سطر اولِ خروجی، x سه رقمی است ولی ۵ محل برای آن جا در نظر گرفته شده است. لذا دو محل در سمت چپ عدد خالی باقی می‌ماند. y چهاررقمی است ولی برای آن ۳ محل در نظر گرفته شد. چون طول میدان کمتر از تعداد ارقام عدد است، طول میدان نادیده گرفته می‌شود و چهار رقم آن به خروجی می‌رود. در سطر دومِ خروجی، برای قسمت اعشاری m دو رقم در نظر گرفته شد که این عدد دارای سه رقم اعشار است، قسمت اعشار گرد می‌شود و در نتیجه رقم ۴ به ۵ تبدیل می‌گردد و ۸ حذف می‌شود. قسمت صحیح m سه رقمی است ولی ۴ محل برای آن در نظر گرفته شده است و لذا یک محل خالی در سمت چپ عدد باقی می‌ماند. قسمت صحیح عدد n، چهار رقم است ولی طول میدان آن سه رقم منظور شد. به همین دلیل طول میدان قسمت صحیح نادیده گرفته می‌شود، هر چهار رقمِ صحیح، به خروجی می‌رود.

اگر برای چاپ یک عدد صحیح، از طول میدانی به شکل %w.dd استفاده شود، d حداقل طول میدان را مشخص می‌کند. اگر طول میدان رشته‌ها به صورت %w.ds ذکر شود، w حداقل طول میدان و d حداکثر کاراکترهای قابل چاپ را مشخص می‌کند.

مثال ۱۲-۲

برنامه‌ای که کاربرد طول میدان به شکل w.d را برای اعداد صحیح و مقادیر رشته‌ای نشان می‌دهد (به تحلیل مثال توجه کنید).

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int x = 1234;
   clrscr();
   printf("x=%4.6d, x=%8.4d", x, x);
   printf("\n%5.10s","this is a sample output.");
   getch();
   return 0;
}
```

خروجی

```
x = 001234 , x =     1234
this is a □
```

تحلیل مثال ۱۲-۲

در فرمت اول، حداقل طول میدان برابر با ۶ در نظر گرفته شد و دو رقم صفر در سمت چپ عدد چاپ شده است، در فرمت دوم، حداقل طول میدان ۴ حداکثر طول میدان ۸ در نظر گرفته شد و چهار محل سمت چپ خالی است. در فرمت سوم (برای رشته)، حداکثر کاراکترهای قابل چاپ برابر با ۱۰ است و در نتیجه ۱۰ کاراکتر چاپ شده است.

ورود اطلاعات توسط تابع scanf()

این تابع برای ورود اطلاعات از صفحه کلید مورد استفاده قرار می‌گیرد و یک تابع همه منظوره در ورود داده‌ها است.

الگوی این تابع در فایل stdio.h قرار دارد. این تابع، تمام انواع داده‌ها را می‌تواند از ورودی بخواند و آنها را در حافظه ذخیره نماید. اگر این تابع با موفقیت اجرا شود، تعداد متغیرهایی را که از ورودی خوانده است برمی‌گرداند و در صورت بروز خطا، EOF توسط تابع برگردانده می‌شود. EOF مقداری است که بیانگر عدم اجرای صحیح تابع scanf() است (EOF با حروف بزرگ در نظر است). تابع scanf() به صورت زیر به کار می‌رود:

scanf (<عبارت ۲> , "<عبارت ۱>")

<عبارت ۲> آدرس متغیرهایی است که باید از ورودی خوانده شوند و <عبارت ۱> مشخص می‌کند که مقادیر ورودی چگونه باید خوانده شوند و در متغیرهایی که آدرس آنها در <عبارت ۲> مشخص شده است قرار گیرند. <عبارت ۱> شامل سه نوع کاراکتر است:

۱. کاراکترهای فرمت. این کاراکترها تعیین می‌کنند که چه نوع اطلاعاتی باید از ورودی خوانده شوند و با ٪ شروع می‌شوند. مثل d% که برای خواندن اعداد صحیح از ورودی به کار می‌رود. کاراکترهای فرمت که در تابع scanf() مورد استفاده قرار می‌گیرند در جدول ۳-۲ آمده‌اند.

۲. کاراکترهای فضای خالی[1]. وجود فضای خالی در <عبارت ۱> موجب می‌شود تا تابع scanf() از فضای خالی موجود در ابتدای اطلاعات ورودی صرفنظر کند. کاراکترهای جدول‌بندی[2]، خط جدید[3] و ردکننده صفحه[4] نیز به عنوان فضای خالی محسوب می‌شوند. حتی یک فضای خالی در <عبارت ۱> موجب می‌شود تا scanf() هر تعداد از فضای خالی (و حتی صفرها) را که در ابتدای اولین کاراکتر غیر فضای خالی (یا صفر) وجود دارد بخواند و از آنها رد شود (آنها را ذخیره نمی‌کند).

۳. کاراکترهایی غیر از فضای خالی و فرمت. وجود چنین کاراکتری موجب می‌شود تا چنانچه همان کاراکتر در رشته ورودی وجود داشته باشد، آن را خوانده از آن صرفنظر کند. به عنوان مثال، "%d,%d" موجب می‌شود تا یک عدد صحیح خوانده شود، سپس یک کاما (,) خوانده و از آن صرفنظر شود و سپس عدد صحیح دیگری خوانده شود. اگر کاراکتر مشخص شده در <عبارت ۱> در رشته ورودی وجود نداشته باشد، تابع scanf() خاتمه می‌یابد. برای خواندن و سپس صرفنظر از علامت ٪ باید ٪٪ را در <عبارت ۱> به کار ببرید.

وقتی دستور scanf() اجرا می‌شود، منتظر می‌ماند تا داده‌ها را از صفحه کلید دریافت نماید. هنگام واردکردن داده‌ها، هر یک از اقلام داده را با یک فاصله یا کاما (,) از هم جدا کنید و پس از ورود داده‌ها، کلید Enter را فشار دهید.

جدول ۳-۲ کاراکترهای فرمت در تابع scanf().

کاراکتر	اطلاعاتی که خوانده می‌شوند	کاراکتر	اطلاعاتی که خوانده می‌شوند
%c	یک کاراکتر	%o	عدد مبنای ۸
%d	یک عدد صحیح دهدهی	%x	عدد مبنای ۱۶
%i	یک عدد صحیح دهدهی	%p	یک اشاره‌گر
%e	عدد اعشاری ممیز شناور	%f	عدد اعشاری ممیز شناور
%g	عدد اعشاری ممیز شناور	%u	عدد صحیح مثبت
%s	رشته‌ها		
%n	مشخص کننده تعداد کاراکترهایی است که تا n% از ورودی خوانده شده‌اند		

1. space 2. tab 3. new line 4. form feed

مثال ۱۳-۲

برنامه‌ای که طول و عرض مستطیلی را از ورودی خوانده مساحت و محیط آن را محاسبه و در صفحه‌نمایش چاپ می‌کند.

توضیح

ورودی برنامه، طول و عرض مستطیل‌اند که آنها را به ترتیب x و y نامگذاری می‌کنیم. خروجیهای برنامه مساحت و محیط مستطیل‌اند که آنها را به ترتیب area و p می‌نامیم. روابط بین ورودیها و خروجیها به صورت زیر است:

عرض × طول = مساحت مستطیل
۲ × (عرض + طول) = محیط مستطیل

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int x, y, area, p;
   clrscr();
   printf(" Enter length and width:");
   scanf("%d%d", &x, &y);
   area = x * y;
   p = (x + y) * 2;
   printf(" Area = %d, p = %d", area, p);
   getch();
   return 0;
}
```

خروجی

```
Enter length and width: 12 10
Area = 120, p = 44
```

مثال ۱۴-۲

برنامه‌ای که سه عدد صحیح را از ورودی خوانده میانگین آنها را محاسبه می‌کند و به خروجی می‌برد.

توضیح

چون هر سه عدد ورودی از نوع صحیح هستند، مجموع آنها نیز صحیح خواهد بود. پس از تقسیم شدن مجموع بر عدد ۳ حاصل آن نیز مقدار صحیح خواهد شد. یعنی تقسیم به صورت صحیح انجام می‌شود. در حالی که انتظار داریم تقسیم به صورت اعشاری انجام شده، میانگین اعشاری در خروجی چاپ شود. برای این منظور باید تبدیل نوع صورت گیرد. چون می‌خواهیم حاصل تقسیم به صورت اعشاری باشد، قبل از تقسیم، نوع اعشاری، یعنی float را در داخل پرانتز قرار می‌دهیم. این روش تبدیل نوع را **type casting** گویند. در این برنامه، x، y و n سه متغیر صحیح هستند که از ورودی خوانده می‌شوند و ave میانگین آن سه عدد است.

```
#include <conio.h>
#include <stdio.h>
int main()
{
```

```
    int x, y, n;
    float ave;
    clrscr();
    printf("\nEnter three integers:");
    scanf("%d%d%d", &x, &y, &n);
    ave = (float)(x + y + n) / 3;
    printf("\nave=%6.2f", ave);
    getch();
    return 0;
}
```

خروجی

```
Enter three integers : 10 15 19
ave = 14.67
```

مثال ۱۵-۲

برنامه‌ای که شعاع دایره‌ای را که یک عدد صحیح است از ورودی خوانده، مساحت و محیط آن را محاسبه می‌کند و به خروجی می‌برد. در این برنامه، متغیر r به عنوان شعاع، area به عنوان مساحت و p به عنوان محیط دایره در نظر گرفته شده است.

توضیح

اگر r شعاع دایره باشد، مساحت آن πr^2 و محیط آن $2\pi r$ است که در آن $\pi = 3.14$ است.

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int r;
   float area, p;
   clrscr();
   printf("Enter the radius:");
   scanf("%d",&r);
   area = 3.14 * r * r;
   p = 2 * 3.14 * r;
   printf("Area = %6.2f, p = %6.2f", area, p);
   getch();
   return 0;
}
```

خروجی

```
Enter the radius:10
Area = 314.00, p =  62.80
```

مثال ۱۶-۲

برنامه‌ای که دو متغیر صحیح x و y را از ورودی خوانده، محتویات آنها را بدون استفاده از متغیر کمکی، عوض کرده، نتیجه را در خروجی چاپ می‌کند.

توضیح

برای عوض کردن محتویات دو متغیر بدون استفاده از متغیر کمکی، آن دو متغیر را جمع کرده، در یکی از آنها قرار می‌دهیم. سپس با عمل تفریق، این جابجایی صورت می‌گیرد. به عنوان مثال، اگر x = 12 و y = 15 باشد، x + y که برابر با ۲۷ است در x قرار می‌گیرد. از این x مقدار y را کم می‌کنیم(۱۵-۲۷) و حاصل آن را در y قرار می‌دهیم(۱۲). سپس y را از x کم می‌کنیم(۱۲-۲۷) و حاصل آن را در x قرار می‌دهیم(۱۵). آیا به نظر شما این روش همواره درست است ؟

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int x, y;
   clrscr();
   printf("\n Enter two integers:");
   scanf("%d%d", &x, &y);
   printf("\n before change:x=%d, y=%d", x, y);
   x += y;
   y = x - y;
   x -= y;
   printf("\n after change:x=%d, y=%d", x, y);
   getch();
   return 0;
}
```

خروجی

```
Enter two integers : 12 15
before change : x = 12 , y = 15
after change : x = 15 , y = 12
```

ورودی و خروجی کاراکترها

همانطور که دیدید، با استفاده از توابع scanf() و printf() می‌توان ورودی و خروجی کاراکترها را انجام داد. ولی در C، توابع خاصی برای ورودی و خروجی کاراکترها منظور شد که کارکردن با آنها راحت‌تر از توابع scanf() و printf() است. در این بخش، تعدادی از آنها را مورد بررسی قرار می‌دهیم.

خواندن کاراکتر با توابع getch() و getche()

این توابع، کاراکتری را از ورودی خوانده در متغیری قرار می‌دهند. این توابع در فایل conio.h قرار دارند و نحوه کاربرد آنها به صورت زیر است:

```
متغیر = getch() ;
متغیر = getche() ;
```

وقتی برنامه به این دستورات می‌رسد، منتظر می‌ماند تا کلیدی از صفحه کلید فشار داده شود. در این صورت، کاراکتر معادل آن کلید، در متغیر قرار می‌گیرد. این توابع به صورت زیر نیز قابل استفاده‌اند.

```
getch() ;
getche()
```

در این صورت، برنامه منتظر می‌ماند تا کلیدی از صفحه کلید فشار داده شود. پس از فشردن کلید، اجرای بقیه دستورات برنامه ادامه می‌یابد.

تابع getch() عکس‌العملی در صفحه‌نمایش ندارد. یعنی وقتی کلیدی فشار داده شد، کاراکتر معادل آن در صفحه‌نمایش ظاهر نمی‌شود. در حالی که تابع getche() پس از خواندن کاراکتر آن را در صفحه‌نمایش نیز ظاهر می‌کند. ضمناً در این توابع نیاز به فشردن کلید Enter نیست. در حالی که هنگام خواندن کاراکتر از طریق تابع scanf()، پس از واردکردن کاراکتر، کلید Enter نیز باید فشار داده شود.

مثال ۱۷-۲

برنامه‌ای که کاراکتری را از ورودی خوانده، در صفحه‌نمایش چاپ می‌کند.

توضیح

در این برنامه، کاراکتری را که خوانده شد، یک بار با فرمت %c و بار دیگر با فرمت %d در خروجی چاپ کردیم. وقتی کاراکتر با فرمت %d چاپ می‌شود، کد اسکی آن نمایش داده خواهد شد.

```
#include <conio.h>
#include <stdio.h>
int main()
{
   char ch;
   clrscr();
   printf("\n Enter a char:");
   ch = getche();
   printf("\n ch=%c, ch=%d", ch, ch);
   getch();
   return 0;
}
```

خروجی

```
Enter a character : s
ch = s , ch = 115
```

خواندن کاراکتر با تابع getchar()

این تابع نیز همانند توابع getch() و getche() برای خواندن کاراکتر از صفحه کلید به کار می‌رود. این تابع در فایل stdio.h قرار دارد و نحوه کاربرد آن به صورت زیر است:

```
متغیر = getchar() ;
```

توجه داشته باشید که در این تابع، پس از ورود کاراکتر، باید کلید enter نیز فشار داده شود. این تابع کاراکتر خوانده شده را در صفحه‌نمایش نیز ظاهر می‌کند.

نوشتن کاراکتر با توابع ()putch و ()putchar

این توابع می‌توانند یک کاراکتر یا یک متغیر کاراکتری را در صفحه‌نمایش چاپ کند. تابع ()putch در فایل conio.h و تابع ()putchar در فایل stdio.h قرار دارد و به صورت زیر به کار می‌روند:

```
putch (متغیر) ;
putch ('کاراکتر') ;
putchar (متغیر) ;
putchar ('کاراکتر') ;
```

به عنوان مثال، دستورات زیر، کاراکتر 'a' را در خروجی چاپ می‌کند:

```
putch ('a') ;
putchar ('a') ;
```

مثال ۱۸-۲

برنامه‌ای که کاراکتری را از ورودی خوانده، با صدور پیامی آن را در خروجی چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
int main()
{
   char ch;
   clrscr();
   printf("\n Enter a character:");
   ch = getchar();
   printf(" You typed the character:");
   putch(ch);
   getch();
   return 0;
}
```

خروجی

```
Enter a character : x
You typed the character : x
```

مثال ۱۹-۲

فرض کنید دستگاه اندازه گیری درجه هوایی که در شهرستان نصب است، دمای هوا را بر حسب درجه فارنهایت اندازه گیری می‌کند. دمای هوا باید برحسب درجه سانتیگراد به اطلاع شهروندان برسد. برنامه‌ای می‌نویسیم که درجه فارنهایت را به درجه سلسیوس تبدیل کند.

تحلیل مثال ۱۹-۲

اولین مرحله در حل مسئله این‌است‌که مشخص شود مسئله چه چیزی را خواسته است. باید یک سیستم اندازه گیری دمای هوا را به سیستم اندازه گیری دیگری تبدیل کنید. **سیستم فارنهایت** باید به **سیستم سلسیوس** تبدیل شود. لذا ورودی برنامه، درجه حرارت برحسب فارنهایت است. شهروندان باید دما را **برحسب درجه سلسیوس** بدانند. لذا خروجی برنامه، دمای هوا برحسب درجه سلسیوس است. متغیر farendeg ، محلی از حافظه را مشخص می‌کند که ورودی برنامه در آن قرار می‌گیرد (درجه فارنهایت) و متغیر centdeg ، محلی از حافظه است که نتیجه محاسبه برنامه یا خروجی را نگه می‌دارد (درجه سلسیوس).

ورودی برنامه

ورودی برنامه، میزان درجه حرارت برحسب فارنهایت است که در متغیر farendeg قرار می‌گیرد.

خروجی برنامه

خروجی برنامه، درجه حرارت برحسب سانتیگراد است که در متغیر centdeg قرار می‌گیرد.

فرمول محاسبه

فرمول محاسبه درجه حرارت برحسب سانتیگراد از درجه حرارت برحسب فارنهایت، به صورت زیر است :

$$\text{سلسیوس} = (\text{فارنهایت} - ۳۲) * \frac{۵}{۹}$$

طراحی

اکنون الگوریتمی طراحی می‌کنیم که مسئله را حل کند :

الگوریتم

۱. درجه حرارت را برحسب فارنهایت بخوان.
۲. درجه حرارت را به سانتیگراد تبدیل کن.
۳. درجه حرارت برحسب سانتیگراد را نمایش بده.

اکنون باید مشخص کنید که هر مرحله از الگوریتم نیاز به اصلاحاتی دارد یا خیر. مرحله اول و مرحله سوم نیاز به اصلاح ندارند. مرحله دوم نیز واضح است، ولی اگر به جزئیات پرداخته شود، روشن‌تر می‌گردد. با توجه به رابطه بین درجه فارنهایت و درجه سلسیوس، الگوریتم را می‌توان به صورت زیر اصلاح کرد:

۱. درجه حرارت را برحسب فارنهایت بخوان.
۲. درجه حرارت را به سلسیوس تبدیل کن.
   ۱-۲. رابطه بین فارنهایت و سلسیوس عبارت‌انداز: $\text{سلسیوس} = (\text{فارنهایت} - ۳۲) * \frac{۵}{۹}$
۳. درجه حرارت برحسب سلسیوس را نمایش بده.

اکنون الگوریتم را تست می‌کنیم. اگر درجه حرارت برحسب فارنهایت برابر با ۵۰ باشد، با توجه به مرحله ۱-۲ الگوریتم، درجه حرارت برحسب سلسیوس برابر است با :

$$\text{سلسیوس} = (۵۰ - ۳۲) * \frac{۵}{۹} = ۱۰$$

در نتیجه، در مرحله سوم، عدد ۱۰ در خروجی چاپ می‌شود.

پیاده‌سازی

برای پیاده‌سازی الگوریتم، باید آن را به برنامه C تبدیل کنید. ابتدا باید به کامپایلر C بگویید که از چه ثوابت و متغیرهایی استفاده می‌کنید و سپس هر قدم الگوریتم را به یک یا چند دستور C تبدیل نمایید. برنامه و نمونه‌ای از اجرای آن در زیر آمده است (نام این برنامه در دیسک، ۱۹-۲ است).

```
#include <conio.h>
#include <stdio.h>
int main()
{
   const int farconst = 32;
   float farendeg, centdeg;
   clrscr();
   printf("Enter farenheite:");
   scanf("%f", &farendeg);
   centdeg =((float)5 / 9) * (farendeg - farconst);
   printf("Celsius = %5.2f", centdeg);
   getch();
   return 0;
}
```

خروجی

```
Enter farenheite : 70
Celsius = 21.11
```

تمرینات

۱. فایل‌های سرآیند به چه منظور مورد استفاده قرار می‌گیرند و امتیازات استفاده از آنها را بیان کنید.

۲. شکل کلی یک برنامه در C را بیان کرده و هر بخش را توصیف نمایید.

۳. برنامهٔ زیر چه اشکالی دارد؟ سعی کنید آن را تایپ و اجرا کنید تا به اشکال آن پی ببرید.

```
#include  <stdio.h>
main() ;
{
      printf ("keep looking!") ;
      printf ("you\'ll find it") ;
      return 0 ;
}
```

۴. برنامه‌ای بنویسید که خروجی زیر را تولید کند:

```
The answer to the question of
Life, the Universe, and Everything is 42.
```

۵. برنامه‌ای بنویسید که مقدار x را از ورودی خوانده، عبارت زیر را محاسبه کند:

$$y = \frac{1}{x^2+x+3}$$

۶. خروجی دستورات زیر چیست ؟

```
number = (1/3) * 3 ;
printf("(1/3) * 3 is equal to %5.2f", number);
```

۷. برنامه‌ای بنویسید که وزن کالایی را بر حسب کیلوگرم دریافت کرده، وزن آن را بر حسب گرم در خروجی چاپ کند.

۸. شرکتی به هر یک از دو نفر از متخصصین خود، ماهانه ۷۵۰۰۰۰ ریال پرداخت می‌کند. او می‌خواهد بداند که اگر ۱۳/۵ درصد به حقوق هر کدام اضافه کند، سالانه چقدر به هزینه شرکت اضافه می‌شود. برنامه‌ای بنویسید که این کار را برای شرکت انجام دهد.

۹. در شرکتی، سالانه ۱۵۰ خودکار و ۵۰ بسته کاغذ A4 مصرف می‌شود. در پایان سال، این شرکت می‌خواهد بداند که در سال آینده چقدر باید برای این بخش از تجهیزات اداری، هزینه کند. برنامه‌ای بنویسید که قیمت این اقلام را در امسال از ورودی خوانده، با خواندن نرخ تورم در سال آینده، هزینه شرکت را در این بخش محاسبه نماید و به خروجی ببرد. تورم به صورت درصد وارد می‌شود که برنامه باید آن را به یک مقدار اعشاری تبدیل کند. مثلاً اگر تورم را به صورت ۵/۶ از ورودی بخواند باید آن را به صورت ۰/۰۵۶ به کار ببرد.

۱۰. برنامه‌ای بنویسید که ارتفاع و قاعده مثلثی را از ورودی خوانده مساحت آن را محاسبه کرده به خروجی ببرد. روش ایجاد برنامه را بر اساس مثال ۱۹-۲ دنبال کنید.

۱۱. وزن یک مولکول آب $۳/۰ \times ۱۰^{-۲۳}$ گرم، و وزن یک لیتر آب در حدود ۹۵۰ گرم است. برنامه‌ای بنویسید که وزن آب را برحسب لیتر از ورودی خوانده، تعداد مولکولهای آن را محاسبه کند.

۱۲. هر سال برابر با $۳/۱۵۶ \times ۱۰^{۷}$ ثانیه است. برنامه‌ای بنویسید که سن شما را دریافت کرده، به ثانیه تبدیل کند.

تمرین حل شده

برنامه‌ای که سن شما را برحسب روز از ورودی خوانده، مشخص می‌کند که سن شما چند سال، چند ماه، چند هفته و چند روز است.

توضیح

برای انجام این تمرین، از تقسیم صحیح و باقیمانده تقسیم استفاده می‌شود. برای به دست آوردن سال، باید تعداد روزها را بر ۳۶۰ تقسیم کرد. برای به دست آوردن روزهای باقیمانده (پس از تبدیل به سال)، باید باقیمانده تعداد کل روزها را نسبت به ۳۶۰ به دست آورد. این روند برای محاسبه تعداد ماه، هفته و روز نیز به کار می‌رود. در این برنامه، days سن شما برحسب روز، y تعداد سال، و w تعداد هفته است. سرانجام روزهای باقیمانده نیز در days قرار می‌گیرد (بر روی دیسک به نام exam.cpp ذخیره شده است).

حل

متغیرهای برنامه

نام	هدف
days	سن برحسب روز
y	سن برحسب سال
m	باقیمانده سن برحسب ماه
w	باقیمانده سن برحسب هفته

```
#include <conio.h>
#include <stdio.h>
int main() {
   int days, y, m, w;
   clrscr();
   printf("\n Enter your age in days:");
   scanf("%d", &days);
   y = days / 360;
   days %= 360;
   m = days / 30;
   days %= 30;
   w = days / 7;
   days %= 7;
   printf("\nyour age is:%d years, %d month, %d weeks, %d days.",y,m,w,days);
   getch();
   return 0;
}
```

خروجی

```
Enter your age in days : 3432
your age is : 9 years, 6 month, 1 weeks, 5 days.
```

حلقه‌های تکرار و ساختارهای تصمیم

در حالت عادی، دستورات برنامه، از اولین دستور به آخرین دستور اجرا می‌شوند. اگر بخواهیم بعضی از دستورات چندین بار اجرا شوند و بعضی از دستورات تحت شرایط خاصی اجرا شده و بعضی دیگر اجرا نشوند، از ساختارهای **تکرار و تصمیم** استفاده می‌کنیم. ساختارهای تکرار برای تکرار اجرای دستورات و ساختارهای تصمیم برای بررسی شرایطی در برنامه و تصمیم‌گیری براساس آن شرایط مورد استفاده قرار می‌گیرند.

ساختارهای تکرار

ساختارهای تکرار، تحت شرایط خاصی، یک یا چند دستور را چندین بار اجرا می‌کنند. به عنوان مثال، اگر بخواهیم تعداد ۱۰۰ عدد را از ورودی بخوانیم و آنها را با هم جمع کنیم. باید عمل **خواندن عدد** را ۱۰۰ بار تکرار کنیم. ساختارهای تکرار در زبانهای برنامه‌سازی مختلف به شکلهای گوناگونی مورد استفاده قرار می‌گیرند. این ساختارها را در زبان C مورد بررسی قرار می‌دهیم.

ساختار تکرار for

ساختار تکرار for یکی از امکانات ایجاد حلقه است و معمولاً در حالتی که تعداد دفعات تکرار حلقه از قبل مشخص باشد، به کار می‌رود. در این ساختار، متغیری وجود دارد که تعداد دفعات تکرار حلقه را کنترل می‌کند. این متغیر را **شمارنده** یا **اندیس حلقه تکرار** می‌نامیم. اندیس حلقه دارای یک مقدار اولیه است و در هر بار اجرای دستورات حلقه، مقداری به آن اضافه می‌شود. این مقدار را که پس از هر بار اجرای حلقه به شمارنده اضافه می‌شود، **گام حرکت** گویند. گام حرکت می‌تواند عددی صحیح و اعشاری، مثبت یا منفی و یا کاراکتری باشد. یکی دیگر از اجزای حلقه for، شرط حلقه است. **شرط حلقه** مشخص می‌کند که دستورات داخل حلقه تا کی باید اجرا شوند. اگر این شرط دارای ارزش درستی باشد، دستورات داخل حلقه اجرا می‌شوند وگرنه کنترل برنامه از حلقه تکرار خارج می‌شود. اندیس حلقه تکرار می‌تواند عددی منفی، مثبت، صحیح و یا اعشاری و کاراکتری باشد. دستور for را به دو شکل می‌توان به کار برد:

روش اول :

```
for  ( گام حرکت ; شرط حلقه ; مقدار اولیه اندیس حلقه ) {
            دستور ۱
            دستور ۲
            دستور n
}
```

روش دوم :

```
for  (;;) {
            دستور ۱
            دستور ۲
            دستور n
}
```

در هر یک از دو شیوه کاربرد، } می‌تواند در سطر بعدی (زیر for) قرار داشته باشد. ولی برای صرفه‌جویی در طول برنامه، در این کتاب به همین شکل که بیان شد استفاده می‌شود. در هر یک از روشهای کاربرد دستور for ، چنانچه فقط یک دستور در حلقه وجود داشته باشد، نیازی به } و { نیست. در این حالت، این دستورات به صورت زیر قابل استفاده‌اند:

روش اول :

```
for  ( گام حرکت ; شرط حلقه ; مقدار اولیه اندیس حلقه )
        ; دستور
```

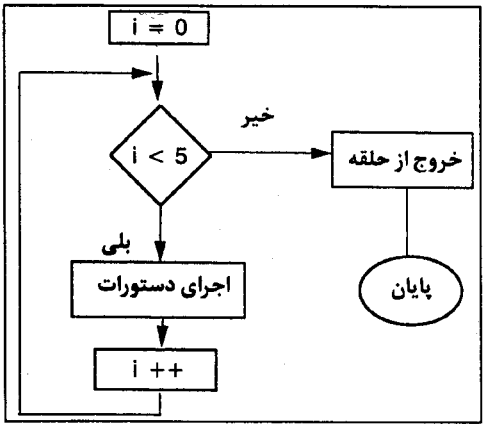
روش دوم :

```
for  (;;)
        ; دستور
```

همان‌طور که ملاحظه می‌شود، در روش کاربرد دوم، for فاقد مقدار اولیه اندیس حلقه، شرط حلقه و گام حرکت است. این دستور برای ایجاد حلقه تکرار **بی‌نهایت** (حلقه تکراری که شرط پایان ندارد) مورد استفاده قرار می‌گیرد. برای خاتمه دادن به اجرای حلقه تکرار بی‌نهایت، باید کلید CTRL+BREAK را از صفحه‌کلید فشار داد. برای آشنایی با مفاهیم مطرح شده، دستور ساده زیر را در نظر بگیرید:

```
for   (i = 0 ; i < 5 ; i ++)
      printf("%3d\n", i) ;
```

در این دستور سادهٔ for ، اندیس حلقه تکرار (شمارنده) i و مقدار اولیه اندیس حلقه برابر با صفر است. شرط حلقه $i < 5$ است. یعنی تا زمانی که $i < 5$ باشد، اجرای دستور موجود در این حلقه ادامه می‌یابد وگرنه کنترل از حلقه تکرار خارج می‌شود. گام حرکت نیز یک است. یعنی به ازای هر بار اجرای دستور printf() که در داخل حلقه قرار دارد، یک واحد به i اضافه می‌شود. شیوه اجرای این دستور for را می‌توان به صورت شکل ۱-۳ رسم کرد.



شکل ۱-۳ شیوه اجرای حلقه for.

مثال ۱-۳

برنامه‌ای که تعداد ۵ عدد صحیح را از ورودی خوانده، میانگین آنها را محاسبه می‌کند و به خروجی می‌برد. توجه داشته باشید که در این برنامه، برای به‌دست آوردن خارج قسمت اعشاری، از type casting استفاده شده است (به مثال ۱۴ از فصل ۲ مراجعه شود).

توضیح

ثابت n با مقدار ۵ تعریف شد و سپس یک حلقه تکرار براساس n ایجاد گردید. در این حلقه، با صدور پیامی، شماره عددی که باید خوانده شود اعلام می‌گردد و سپس با دستور scanf این عدد خوانده می‌شود. عدد خوانده شده با sum جمع می‌شود و در sum قرار می‌گیرد. در خارج از حلقه تکرار، sum بر n تقسیم می‌شود تا میانگین محاسبه شود. سپس میانگین به خروجی می‌رود.

متغیرهای برنامه

نام	هدف
i	شمارنده حلقه
sum	مجموع اعداد
ave	میانگین
num	عددی که خوانده می‌شود
n	ثابتی به طول ۵ که تعداد اعداد است

```
#include <stdio.h>
#include <conio.h>
int main()
 {
      int i, sum=0, num ;
      float ave ;
      const int n = 5;
      clrscr();
      for(i = 0 ; i < n; i++) {
          printf("enter number %d :",i+1) ;
          scanf("%d",&num) ;
          sum += num ;
      }
      ave = (float) sum / n ;
      printf("\n the average is :%6.2f ",ave) ;
      getch();
      return 0;
 }
```

خروجی

```
enter number 1 : 12
enter number 2 : 13
enter number 3 : 15
enter number 4 : 16
enter number 5 : 13
the average is : 13.80
```

مثال ۲-۳

برنامه‌ای که اعداد 0.5 تا 3.5 را با فاصله 0.5 در خروجی چاپ می‌کند. هدف از این برنامه، آشنایی با حلقه تکرار با اندیس اعشاری است. چون فقط یک دستور در حلقه تکرار می‌شود، نیاز به { و } نیست.

```
#include <stdio.h>
#include <conio.h>
int main()
 {
      float i ;
      clrscr();
      for(i = 0.5; i <= 3.5; i += 0.5)
          printf("%5.2f ", i) ;
      getch();
      return 0;
 }
```

خروجی

```
0.50   1.00   1.50   2.00   2.50   3.00   3.50
```

مثال ۳-۳

برنامه‌ای که کاراکترهای 'a' تا 'f' را به همراه کداسکی آنها به خروجی می‌برد.

توضیح

در این برنامه، از یک حلقه تکرار استفاده شد که اندیس آن کاراکتری است. وقتی کاراکتر با فرمت c٪ نوشته می‌شود، خودِ کاراکتر و وقتی با فرمت d٪ نوشته می‌شود، کداسکی آن چاپ می‌شود.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char ch;
   clrscr();
   for(ch = 'a'; ch <= 'f'; ch ++)
       printf("ch = %c, code = %d\n", ch, ch);
   getch();
   return 0;
}
```

خروجی

```
ch = a, code = 97
ch = b, code = 98
ch = c, code = 99
ch = d, code = 100
ch = e, code = 101
ch = f, code = 102
```

مثال ۴-۳

برنامه‌ای که تعداد حروف جمله‌ای راکه از ورودی می‌خواند، شمارش می‌کند، انتهای جمله به نقطه ختم می‌شود. در این برنامه، ch کاراکتری است که از ورودی خوانده می‌شود و count تعداد کاراکترهای خوانده شده است.

توضیح

حلقه تکراری که در این برنامه آمده است، به ; ختم شده است. به همین دلیل، هیچ دستوری در داخل حلقه تکرار وجود ندارد. در نتیجه، هر کاراکتری که از ورودی خوانده می‌شود، یک واحد به count اضافه می‌گردد و در پایان حلقه تکرار، مقدار آن چاپ می‌شود.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char ch;
   int count;
   clrscr();
   printf("Enter a statement with . in end:\n");
   for(count = 0; (ch = getche())!='.' ; count++) ;
   printf("\n length of statement is:%d",count) ;
   getch();
   return 0;
}
```

خروجی

```
Enter a statement with . in end :
I like C very much.
Length of statment is : 18
```

حلقه‌های تکرار تودرتو

وقتی حلقه تکراری در داخل حلقه تکرار دیگر قرار داشته باشد، می‌گوییم که حلقه‌های **تودر تو** ایجاد شده‌اند. قانونی که بر حلقه‌های تکرار تو در تو حاکم است این‌است‌که، به ازای هر بار اجرای حلقه تکرار خارجی، حلقه تکرار داخلی به طور کامل اجرا می‌شود. ضمناً، انتهای حلقه تکرار داخلی، زودتر از حلقه تکرار خارجی مشخص می‌شود. به عنوان مثال، در دستورات زیر، حلقه تکرار با اندیس i، حلقه خارجی و حلقه تکرار بااندیس j، حلقه تکرار داخلی است:

```
for   (i = 0; i < 5; i++) {
        ...
        for   (j = 0; j < 6; j++) {
           ...
        }
        ...
}
```

اگر حلقه‌های تکرار تو در تو، از یکجا شروع و به یکجا ختم شوند، حلقه خارجی نیاز به آکولاد ندارد. نمونه‌ای از این نوع حلقه در زیر مشاهده می‌شود:

```
for   (I = 0; I < 5; I++)
      for   (j = 0; j <= 6; j++) {
          ...
      }
```

مثال ۵-۳

برنامه‌ای که جدول ضرب اعداد ۱ تا ۱۰ را تولید کرده در خروجی چاپ می‌کند. هدف از این برنامه، آشنایی با کاربرد دو حلقه تکرارِ تودرتو است.

توضیح

در این برنامه، دو حلقه تکرار تودرتو وجود دارد. به ازای هر مقدار i، مقدار jاز ۱ تا ۱۰ تغییر می‌کند و در نتیجه یک سطر از جدول تولید می‌شود و در خروجی چاپ می‌گردد و پس از پایان حلقه j، دستور printf("\n") سطر جاری را رد می‌کند تا دفعه بعد، سطر دیگری از جدول تولید شود. این روند آنقدر ادامه می‌یابد تا iبه ۱۱ برسد. به‌این‌ترتیب، جدول ضرب ایجاد می‌شود و برنامه در صفحه خروجی منتظر فشردن کلیدی از صفحه‌کلید است.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      int i,j ;
      clrscr();
      for(i=1 ; i<=10 ; i++) {
          for(j=1 ; j<=10 ; j++)
              printf("%3d ",i*j) ;
          printf("\n") ;
      }
      getch();
      return 0;
}
```

خروجی

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

مثال ۶-۳

برنامه‌ای که تعدادی عدد را از ورودی خوانده، فاکتوریل آنها را محاسبه می‌کند. تعداد اعداد نامعلوم است و برای خاتمه اجرای برنامه باید کلید CTRL+BREAK را فشار داد. هدف از این برنامه، آشنایی با ایجاد حلقه‌های تکرار بی‌نهایت با استفاده از for است.

توضیح

در این برنامه، یک حلقه تکرار برای خواندن اعداد و حلقه تکرار داخلی برای محاسبه فاکتوریل هر عدد به‌کار می‌رود. num عددی است که فاکتوریل آن باید محاسبه شود و fact فاکتوریل آن عدد است. توجه داشته باشید که فاکتوریل برای اعداد صحیح مثبت به صورت زیر تعریف می‌شود:

0! = 1

n! = 1 × 2 × 3 × 4 × ... × (n-1) × n ;

متغیرهای برنامه

نام	هدف
i	شمارندهٔ حلقه تکرار
num	عدد خوانده شده
fact	فاکتوریل عدد

```
#include <stdio.h>
#include <conio.h>
int main()
{
   long int fact ;
   int i, num ;
   clrscr();
   for(;;) {
     printf("\nType a number :") ;
     scanf("%d",&num) ;
     fact = 1 ;
     for(i = 1; i<= num; i++)
         fact *= i ;
     printf("Factorial is :%ld", fact);
   }
 }
```

نمونه‌هایی از خروجی برنامه

```
Type a number : 5
Factorial is : 120
Type a number : 6
Factorial is : 720
```

مثال ۷-۳

برنامه‌ای که یک سکه ۱۰۰ ریالی را به سکه‌های ۲، ۵، ۱۰، ۲۰ و ۵۰ ریالی خرد می‌کند.

متغیرهای برنامه

نام	هدف
i2	اندیس حلقه‌ای که تعداد سکه‌های ۲ ریالی را شمارش می‌کند
i5	اندیس حلقه‌ای که تعداد سکه‌های ۵ ریالی را شمارش می‌کند
i10	اندیس حلقه‌ای که تعداد سکه‌های ۱۰ ریالی را شمارش می‌کند
i20	اندیس حلقه‌ای که تعداد سکه‌های ۲۰ ریالی را شمارش می‌کند
i50	اندیس حلقه‌ای که تعداد سکه‌های ۵۰ ریالی را شمارش می‌کند
sum	مجموع ریالی ترکیبات حاصل از حلقه‌های تکرار
count	تعداد ترکیباتِ درست

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int i2, i5, i10, i20, i50, count = 0;
  unsigned long int sum;
  clrscr();
  for(i2 = 0; i2 <= 50; i2 ++)
    for(i5 = 0; i5 <= 20; i5 ++)
      for(i10 = 0; i10 <= 10; i10 ++)
        for(i20 = 0; i20 <= 5; i20 ++)
          for(i50 = 0; i50 <= 2; i50 ++){
              sum=i2*2 + i5*5 + i10 *10 +i20*20 + i50*50;
              if(sum==100){
                 printf("\n 2Rials=%d, 5Rials=%d, 10Rials=%d", i2, i5, i10);
                 printf(",20Rials=%d, 50Rials=%d", i20, i50);
                 count ++;
              }// end of if
              else
              sum = 0;
          }//end of for
  printf("\n number of correct times:%d", count);
  getch();
  return 0;
}
```

عملگر کاما و حلقه for

عملگر کاما (,)در فصل یک مورد بررسی قرار گرفت. این عملگر به حلقه for قابلیت انعطاف بیشتری می‌بخشد. با استفاده از این عملگر، می‌توان در قسمت مقدار اولیه حلقه و گام حرکت، دو یا چند عبارت را باهم ترکیب کرد. عبارات به ترتیب قرار گرفتن، و از چپ به راست ارزیابی می‌شوند. به عنوان مثال، دستور زیر را در نظر بگیرید:

```
for  (i = 0, m += i; i < 10; i++, m++){
        ...
}
```

در این حلقه تکرار، اولین کاما، دو متغیر i و m را مقدار اولیه می‌دهد و دومین کاما، در هر تکرار، یک واحد به i و یک واحد به m اضافه می‌کند.

مثال ۸-۳

برنامه‌ای که مجموع چند دورهٔ اولیه از سری زیر را محاسبه می‌کند.

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$$

توضیح

این برنامه، ابتدا مجموع یک جمله، سپس مجموع ۲ جمله، بعد مجموع ۳ جمله و ...و در انتها، مجموع ۶ جمله را محاسبه کرده به خروجی می‌برد. به همین دلیل، ثابت NUM برابر با ۶ تعریف شده است. در این برنامه، count تعداد جمله در هر بار، sum مجموع جملات در هر بار، و x مولد مخرج کسر است.

```
#include <stdio.h>
#include <conio.h>
#define NUM 6
int main()
{
  int count;
  float sum, x;
  clrscr();
  for(sum = 0, x = 1.0, count = 1; count <= NUM; count ++, x *= 2)
  {
    sum += 1 / x;
    printf("sum = %7.4f, when count = %d\n", sum, count);
  }
  getch();
  return 0;
}
```

خروجی

```
sum=1.0000 when count=1
sum=1.5000 when count=2
sum=1.7500 when count=3
sum=1.8750 when count=4
sum=1.9375 when count=5
sum=1.9688 when count=6
```

ساختار تکرار while

ساختار تکرار while یکی دیگر از امکاناتی است که برای تکرار اجرای دستورات به کار می‌رود. این ساختار به صورتهای زیر قابل استفاده است:

روش اول:

```
while (شرط)
        دستور ;
```

روش دوم:

```
while (شرط) {
        دستور ۱
        دستور ۲
        .
        .
        .
        دستور n
  }
```

همان‌طور که ملاحظه می‌کنید، وقتی دستورات تکرار شونده، بیش از یکی باشند، باید آنها را در بین { و } قرار داد. پس از اینکه اجرای برنامه به این دستور رسید، شرط حلقه تست می‌شود. اگر این شرط دارای ارزش درستی باشد، دستورات حلقه اجرا می‌شوند وگرنه کنترل برنامه از حلقه تکرار خارج می‌شود. برای اینکه حلقه خاتمه پیدا کند، شرط حلقه باید در داخل حلقهٔ تکرار نقض شود. یعنی باید شرایطی در داخل حلقه فراهم شود تا شرط حلقه ارزش نادرستی پیدا کند و حلقه خاتمه یابد. اگر شرط حلقه همیشه درست باشد (هیچگاه نقض نشود)، حلقه **تکرار بی‌نهایت** ایجاد می‌شود. در ادامه، مثالی را در این مورد مشاهده خواهید کرد.

مثال ۹-۳

برنامه‌ای که جمله‌ای را از ورودی خوانده، تعداد کاراکتر جمله را شمارش می‌کند. انتهای جمله به کلید Enter ختم می‌شود ('\r'). در این برنامه، count تعداد کاراکترهای ورودی است.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int count=0 ;
    clrscr();
    printf("type a statement and ENTER to end:");
    while(getche() !='\r')
       count ++ ;
    printf("\n length of statement is:%d",count) ;
    getch();
    return 0;
}
```

خروجی

```
type a statment and ENTER to end : I learn C language.
length of statment is : 19
```

مثال ۱۰-۳

برنامه‌ای که تعدادی عدد را خوانده مجموع مربعات آنها را محاسبه می‌کند و به همراه تعداد اعداد به خروجی می‌برد.

توضیح

در این برنامه، پس از ورود هر عدد، از کاربر سئوال می‌شود که آیا عدد دیگری دارد یا خیر. اگر پاسخ مثبت بدهد (y را وارد کند)، حلقه تکرار ادامه می‌یابد و عدد بعدی خوانده می‌شود. برای کنترل حلقه تکرار، از متغیر ans استفاده شده است که مقدار اولیه آن 'y' است. شرط حلقه این طور است: تا زمانی که ans برابر با 'y' است حلقه ادامه یابد. چنانچه کاربر در پاسخ به سئوال برنامه، کاراکتری غیر از 'y' را وارد کند، برنامه خاتمه می‌یابد.

متغیرهای برنامه

نام	هدف
x	عدد خوانده شده
sum	مجموع مربعات عدد
n	تعداد اعدادی که خوانده شدند
ans	پاسخ کاربر به سئوال برنامه

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int x, sum = 0, n = 0;
    char ans = 'y';
    clrscr();
    while(ans == 'y'){
       printf("\nEnter a number:");
       scanf("%d",&x);
       sum += x*x;
       n ++;
       printf("\n Do you want to continue?(y/n):");
       ans = getche();
    }//end of while
    printf("\nYou entered %d numbers.",n);
    printf("\nSum of square is:%d",sum);
    getch();
    return 0;
}
```

عملکرد برنامه

```
Enter a number : 5
Do you want to continue ? (y/n) : y
Enter a number : 15
Do you want to continue ? (y/n) : n
You entered 2 numbers.
Sum of square is : 250
```

ساختار تکرار do ... while

ساختار تکرار do... while مانند ساختار تکرار while است؛ با این تفاوت که در ساختار while، شرط حلقه در ابتدای حلقه تست می‌شود، در حالی که در do ... while شرط حلقه در انتهای حلقه تست می‌گردد. بنابراین، دستورات موجود در حلقه do ... while، در هر حال، حداقل یک بار اجرا می‌شوند. این ساختار به صورتهای زیر به کار می‌رود:

روش اول:

```
do
      دستور ;
      while   (شرط) ;
```

روش دوم:

```
do {
        دستور ۱
        دستور ۲
        ...
        دستور n
  } while   (شرط) ;
```

در این ساختار نیز، وقتی تعداد دستورات تکرار شونده بیش از یکی باشد، دستورات در بین } و { قرار می‌گیرند. چنانچه شرط حلقه در داخل حلقه تکرار نقض نشود، این ساختار نیز حلقه تکرار بی‌نهایتی را ایجاد می‌کند.

مثال ۱۱-۳

برنامه‌ای که تعدادی عدد را از ورودی خوانده، وارون آنها را به خروجی می‌برد. وارون عددی مثل x، عددی مثل y است به‌طوری که ارقام عدد xاز راست به چپ مثل ارقام عدد yاز چپ به راست باشد. به‌عنوان مثال ، وارون عددِ ۲۱۵۳، عدد ۳۵۱۲است.

توضیح

در این برنامه، یک حلقه تکرار بی‌نهایت whileایجاد شده است تا تعدادی نامعلوم از اعداد را بخواند و وارون آنها را محاسبه کند. برای ایجاد حلقه تکرار بی‌نهایت، در شرط حلقه، (۱) را قرار دادیم. (۱) مقداری غیر صفر است و در C دارای ارزش درستی است و در طول اجرای برنامه نیز تغییر نمی‌کند. حلقه تکرار دیگر که یک حلقه do...while است وظیفه وارون کردن عدد را به عهده دارد. در این برنامه، متغیر num عددی است که باید وارون شود و digit ارقام آن عدد است. برای جداکردن ارقامِ عدد باقیمانده تقسیم آن عدد را بر ۱۰ پیدا می‌کنیم.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num, digit = 0 ;
    clrscr();
    while(1){
       printf("\nENTER a number:");
       scanf("%d",&num);
       printf("\nInverse=");
       do{
          digit = num % 10;
          printf("%d",digit);
          num /= 10;
       } while(num !=0);
    }//end of while(1)
    getch();
    return 0;
}
```

خروجی

```
Enter a number : 1345
Inverse = 5431
```

از کدام حلقه تکرار استفاده کنیم ؟

معمولاً هنگامی که تعداد دفعات حلقه تکرار مشخص باشد و گام حرکت نیز معین باشد، از حلقه for استفاده

می‌گردد. در موارد دیگر سعی کنید از حلقه‌های while و do ... while استفاده کنید. اگر شرایطی وجود داشت که، چه شرط حلقه درست باشد و چه نادرست، دستورات حلقه حداقل یک بار باید اجرا شوند، از حلقه do...while وگرنه از حلقه while استفاده کنید.

ساختارهای تصمیم

همانطور که دیدید، ساختارهای تکرار، برای تکرار اجرای دستورات مورد استفاده قرار می‌گیرند. اما اگر بخواهیم، تحت شرایطی، تعدادی از دستورات اجرا شوند و یا تعدادی دیگر از دستورات اجرا نشود، باید از ساختارهای تصمیم استفاده کنیم. این ساختارها شرطی را تست کرده در صورت درست بودن شرط، مجموعه‌ای از دستورات را انجام می‌دهند. در زبان C چندین ساختار تصمیم وجود دارد که آنها را در این بخش بررسی می‌کنیم.

ساختار تصمیم if

ساختار if که نام دیگرش، دستور انتقال کنترل شرطی است، شرطی را تست می‌کند و در صورتی که آن شرط دارای ارزش درستی باشد، مجموعه‌ای از دستورات را اجرا می‌کند. این دستور به صورت زیر به کار می‌رود:

روش اول:

```
if      (شرط)
            دستور ;
else
            دستور ;
```

روش دوم:

```
if      (شرط) {
            دستور ۱
            دستور ۲
            ...
            دستور n
}
else {
            دستور n1
            دستور n2
            ...
            دستور n
}
```

در هر یک از روشهای کاربرد، چنانچه شرط مورد بررسی درست باشد، دستور یا دستورات بعد از if وگرنه دستور یا دستورات بعد از else اجرا می‌شوند. اگر بیش از یک دستور بعد از if یا else بیایند، آن دستورات باید در بین { و } قرار گیرند. دستور if می‌تواند فاقد قسمت else باشد. در این صورت، چنانچه شرط مورد بررسی، درست باشد، دستورات بعد از if اجرا می‌شوند وگرنه بدون اجرای این دستورات، کنترل اجرای برنامه از if خارج می‌شود.

مثال ۱۲-۳

برنامه‌ای که با خواندن یک جمله از ورودی، تعداد کاراکترها و کلمات موجود در جمله را شمارش می‌کند. کلمات با فاصله (space) از هم جدا شده‌اند و انتهای جمله به کلید Enter ختم می‌شود. متغیر charcount تعداد کاراکترها و متغیر wordcount تعداد کلمات جمله را شمارش می‌کند و ch کاراکتری است که از ورودی خوانده می‌شود.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int charcount = 0, wordcount = 0;
  char ch;
  clrscr();
  printf("\n Enter a statement(ENTER):");
  while((ch = getche()) != '\r'){
    charcount ++;
    if(ch == ' ')
       wordcount ++;
  }//end of while
  printf("\ncharcount=%d, wordcount=%d", charcount, wordcount+1);
  getch();
  return 0;
}
```

خروجی

Enter a statment (ENTER) : This book is my favourite.
Character count = 26, wordcount = 5

مثال ۱۳-۳

برنامه‌ای که جدول کدهای اسکی را در خروجی چاپ می‌کند. در این برنامه کدهای اسکی از ۴۱ تا ۱۲۰ چاپ می‌شوند.

توضیح

برای این منظور حلقه تکراری از ۴۱ تا ۱۲۰ ایجاد می‌شود و اندیس حلقه تکرار یک بار با فرمت %d و بار دیگر با فرمت %c چاپ می‌گردد. وقتی کاراکتر با فرمت %c چاپ می‌شود، کاراکتر موردنظر و وقتی که با فرمت %d چاپ می‌شود، کد اسکی آن چاپ می‌گردد. برای اینکه هر ۵ کد اسکی بر روی یک سطر چاپ شوند، از دستور if برای این منظور استفاده شد. هر وقت i مضربی از ۵ باشد دستور printf("\n") سطر جدیدی را آماده چاپ می‌کند.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int i;
```

```
    clrscr();
    for(i = 41; i<= 120; i++){
       printf("%4d=%c\t", i, i);
       if(i % 5 == 0)
          printf("\n");
    }//end of for
    getch();
    return 0;
}
```

خروجی

41 =)	42 = *	43 = +	44 = '	45 = -
46 = .	47 = /	48 = 0	49 = 1	50 = 2
51 = 3	52 = 4	53 = 5	54 = 6	55 = 7
56 = 8	57 = 9	58 = :	59 = ;	60 = <
61 = =	62 = >	63 = ?	64 = @	65 = A
66 = B	67 = C	68 = D	69 = E	70 = F
71 = G	72 = H	73 = I	74 = J	75 = K
76 = L	77 = M	78 = N	79 = O	80 = P
81 = Q	82 = R	83 = S	84 = T	85 = U
86 = V	87 = W	88 = X	89 = Y	90 = Z
91 = [	92 = \	93 =]	94 = ^	95 = _
96 = ‘	97 = a	98 = b	99 = c	100 =d
101 = e	102 = f	103 = g	104 = h	105 = i
106 = j	107 = k	108 = l	109 = m	110 = n
111 = o	112 = p	113 = q	114 = r	115 = s
116 = t	117 = u	118 = v	119 = w	120 = x

مثال ۱۴-۳

برنامه‌ای که جمله‌ای را که به کلید Enter ختم می‌شود، خوانده تعداد کل کاراکترها و تعداد ارقام موجود در جمله را شمارش می‌کند. دقت داشته باشید که کد اسکی ارقام از ۴۸ تا ۵۷است. به همین دلیل، اگر کاراکتری بین ۴۸ تا ۵۷(و مساوی این اعداد) باشد، از ارقام ۰ تا ۹است.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int charcnt = 0, digitcnt = 0;
  char ch;
  clrscr();
  printf("\n Enter a statement(ENTER):");
  while((ch = getche()) != '\r'){
    charcnt ++;
```

```
    if(ch >= 48 && ch <=57)
       digitcnt ++;
  }//end of while
  printf("\ncharcount=%d, digitcount=%d", charcnt, digitcnt);
  getch();
  return 0;
}
```

خروجی

```
Enter a statement (ENTER) : I write 21 books.
character = 17, digitcount = 2
```

مثال ۱۵-۳

برنامه‌ای که عدد تولید شده توسط کامپیوتر را حدس می‌زند. در این برنامه، کامپیوتر با استفاده از تابع rand() یک عدد تصادفی تولید می‌کند و کاربر سعی می‌کند آن را حدس بزند. برنامه پس از دریافت عددی از کاربر، آن را با عدد حدس زده شده مقایسه می‌کند، چنانچه این دو عدد با هم برابر باشند، برنامه به اتمام می‌رسد وگرنه پیامی به کاربر ارسال می‌شود و می‌گوید عددی که وارد کرد کوچکتر از عدد موردنظر است یا بزرگتر. سپس از کاربر سئوال می‌شود که آیا به بازی ادامه می‌دهد یا خیر. توجه داشته باشید که تابع rand() در فایل stdlib.h قرار دارد.

متغیرهای برنامه

نام	هدف
magic	عددی که کامپیوتر تولید می‌کند
guess	عددی که کاربر حدس می‌زند
ans	متغیری برای کنترل حلقه تکرار

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
  int magic, guess;
  char ans = 'y';
  magic = rand();
  do {
      clrscr();
      printf("\n Guess the the magic number:");
      scanf("%d", &guess);
      if(guess == magic) {
         printf("\n***** right *****");
         printf("\n %d is the magic number.", magic);
         getch();
         ans = 'n'; //exit from while
      } // end of if
      else {
           printf("\n***** wrong ******");
           if(guess > magic)
              printf("\n your guess is too high.");
           else
```

```
            printf("\n your guess is too low.");
        printf("\n Do you want to continue?(y/n):");
        ans = getche();
    }// end of else
  } while(ans == 'y');
  return 0;
}
```

خروجی

در کامپیوترتان ممکن است عدد دیگری تولید شود و خروجی به این شکل نباشد.

Guess the magic number : 346
******* right *******
346 is the magic number.

ساختار تصمیم else if

اگر بخواهیم از دستور if برای تست شرطهای متعددی استفاده کنیم باید آنها را به طور تودرتو به کار ببریم. کاربرد if به صورت تودرتو، نه تنها موجب طولانی شدن برنامه می‌شود، بلکه از خوانایی برنامه نیز می‌کاهد. ساختار else if می‌تواند به جای ifهای تودرتو به کار گرفته شود و میزان خوانایی برنامه را بالا ببرد. برای روشن شدن موضوع، به مثالهای ۱۶-۳ و ۱۷-۳ که هر دو، یک مسئله را حل می‌کنند (یکی با if و دیگری با else if) توجه کنید.

مثال ۱۶-۳

برنامه‌ای که نمره عددی دانشجویی را خوانده، معادل حرفی آن را در خروجی چاپ می‌کند. در این برنامه grade نمره عددی دانشجو است:

17 <= نمره عددی <= 20 ⇒ 'A'
15 <= نمره عددی <17 ⇒ 'B'
12 <= نمره عددی <15 ⇒ 'C'
نمره عددی <12 ⇒ 'D'

```
#include <stdio.h>
#include <conio.h>
int main()
{
  float grade;
  clrscr();
  while(1){
    printf("\n Enter a grade:");
    scanf("%f",&grade);
    if (grade >= 17 && grade <= 20)
       printf("\n grade=%5.2f, score=%c", grade, 'A');
    else
```

```
    if (grade >= 15 && grade < 17)
      printf("\n grade=%5.2f, score=%c", grade, 'B');
    else
    if (grade >= 12 && grade < 15)
      printf("\n grade=%5.2f, score=%c", grade, 'C');
    else
    if (grade < 12)
      printf("\n grade=%5.2f, score=%c", grade, 'D');
  }//end of while
}
```

خروجی

```
Enter a grade : 12
grade = 12.00 , score = C
Enter a grade : 18
grade = 18.00 , score = A
```

مثال ۱۷-۳

برنامهٔ مثال ۱۶-۳ با استفاده از ساختار else if. این دو برنامه را با هم مقایسه کرده به اهمیت ساختار else if پی ببرید.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  float grade;
  clrscr();
  while(1){
    printf("\n Enter a grade:");
    scanf("%f",&grade);
    if (grade >= 17 && grade <= 20)
      printf("\n grade=%5.2f, score=%c", grade, 'A');
    else if (grade >= 15 && grade < 17)
      printf("\n grade=%5.2f, score=%c", grade, 'B');
    else if (grade >= 12 && grade < 15)
      printf("\n grade=%5.2f, score=%c", grade, 'C');
    else if (grade < 12)
      printf("\n grade=%5.2f, score=%c", grade, 'D');
  }//end of while
}
```

خروجی

```
Enter a grade : 16
grade = 16.00 , score = B
Enter a grade : 14
grade = 14.00 , score = C
```

انتقال کنترل غیر شرطی

دستور if شرطی را بررسی کرده، براساس نتیجه شرط، دستورالعملهایی را انجام می‌دهد. در C دستورالعملهایی وجود دارند که بدون تست شرطی می‌توانند کنترل اجرای برنامه را از نقطه‌ای به نقطه دیگر منتقل کنند. این دستورات را انتقال کنترل **غیر شرطی** گویند.

دستور break

این دستور موجب خروج از حلقه‌های تکرار می‌شود. نحوه کاربرد این دستور به صورت زیر است:

```
break ;
```

اگر چند حلقه تودرتو وجود داشته باشد، این دستور موجب خروج از داخلی‌ترین حلقه تکرار می‌شود. کاربرد دیگر این دستور، خاتمه دادن به ساختار switch است که در ادامه بحث خواهد شد.

مثال ۱۸-۳

برنامه‌ای که تعدادی عدد را از ورودی خوانده تعداد اعداد زوج و فرد را مشخص می‌کند و به خروجی می‌برد. آخرین عدد ورودی، صفر است.

توضیح

در این برنامه، count تعداد اعداد زوج و n تعداد اعداد خوانده شده است. تعداد اعداد فرد برابر با n - count است. یک حلقه تکرار بی‌نهایت ایجاد شده عدد مورد بررسی خوانده می‌شود. چنانچه این عدد صفر باشد، حلقه تکرار با break خاتمه می یابد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int num, count = 0, n = 0;
  clrscr();
  while(1){
     printf("Enter a number:");
     scanf("%d", &num);
     if(num == 0)
        break;
     n ++;
     if(num % 2 == 0)
        count ++;
  }
  printf("\n events=%d, odds=%d", count, n - count);
  getch();
  return 0;
}
```

خروجی

```
Enter a number : 12
Enter a number : 16
Enter a number : 18
Enter a number : 15
Enter a number : 0
events = 3 , odds = 1
```

دستور continue

این دستور در حلقه تکرار موجب انتقال کنترل به ابتدای حلقه می‌شود. پس از انتقال کنترل به ابتدای حلقه، شرط حلقه مورد بررسی قرار می‌گیرد، چنانچه شرط درست باشد، اجرای دستورات حلقه ادامه می‌یابد وگرنه حلقه تکرار خاتمه می‌یابد.

مثال ۱۹-۳

برنامه‌ای که کاراکتری را از ورودی خوانده این کاراکتر و کاراکتر بعد از آن را به خروجی می‌برد. اگر کاراکتر & وارد شود، برنامه خاتمه می‌یابد.

توضیح

برای کنترل حلقه تکرار از متغیر done استفاده شده است. پس از اینکه کاراکتر & وارد شد، done برابر با صفر می‌شود (ارزش نادرستی پیدا می‌کند) و دستور continue کنترل اجرای برنامه را به ابتدای حلقه while می‌برد و شرط تست می‌شود. چون شرط نقص شده است، حلقه تکرار خاتمه می‌یابد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int done = 1;
  char ch;
  clrscr();
  while(done){
    printf("\n Enter a character:");
    ch = getche();
    if(ch == '&'){
       done = 0;
       continue;
    }// end of if
      printf("\n you typed char %c, next char is:%c", ch, ch + 1);
  }// end of while
  getch();
  return 0;
}
```

خروجی

```
Enter a character : r
you typed char r, next char is : s
Enter a character : &
```

دستور goto

این دستور که معمولاً به ندرت در برنامه‌ها استفاده می‌شود، سبب انتقال کنترل از نقطه‌ای به نقطه دیگر از برنامه می‌شود. روش کاربرد این دستور به صورت زیر است:

goto <برچسب>

برچسب دستور، همانند متغیرها نامگذاری می‌شود و به کولن (:) ختم می‌گردد. مثل :L1 و :loop. انتقال کنترل توسط goto فقط در داخل یک تابع امکان‌پذیر است.

مثال ۳-۲۰

برنامه‌ای که با استفاده از دستور goto حلقه تکراری را ایجاد می‌کند.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int x = 1;
    clrscr();
 loop1:
    x ++;
    if(x < 100)
       goto loop1;
    printf("\nThe maximum value of x is:%d", x);
    getch();
    return 0;
}
```

خروجی

```
The maximum value of x is : 100
```

ساختار تصمیم switch

ساختار switch یکی از ساختارهای جالب و مهم در زبان C است. از این ساختار برای تصمیم‌گیریهای چندگانه براساس مقادیر مختلف یک عبارت، استفاده می‌شود. به طور کلی، در تمام تصمیم‌گیریهایی که بیش از سه انتخاب وجود داشته باشد، بهتر است از ساختار switch استفاده شود. به عنوان مثال، فرض کنید، متغیری به نام x دارید که این متغیر مقادیر ۱، ۷، ۹ و ۱۵ را می‌پذیرد و می‌خواهید براساس مقادیر مختلف x، تصمیم‌گیریهای متعددی انجام دهید. اگر x برابر با یک بود، <مجموعه دستورات ۱>، اگر x برابر با ۷ بود، <مجموعه دستورات ۲>، اگر x برابر با ۹ بود، <مجموعه دستورات ۳> و اگر x برابر با ۱۵ بود، <مجموعه دستورات ۴> اجرا شوند و اگر x با هیچکدام از

این مقادیر برابر نبود، <مجموعه دستورات ۵> اجرا شوند. این مفهوم با استفاده از ساختار switch پیاده‌سازی خواهد شد. این ساختار به صورت زیر به کار می‌رود:

```
switch    (عبارت)  {
          case    <مقدار ۱>  :
                  <دستورات ۱>
                  break ;
          case    <مقدار ۲>  :
                  <دستورات ۲>
                  break ;
                    .
                    .
          default:
                  <n دستورات >
}
```

عملکرد switch: ابتدا عبارت موجود در مقابل switch به مقدار صحیح ارزیابی می‌شود و مقدار آن تعیین می‌گردد. اگر این مقدار با <مقدار ۱> برابر بود، <دستورات ۱> اجرا می‌شوند و دستور break که بعد از آنها قرار دارد، کنترل برنامه را از ساختار switch خارج می‌کند. اگر با <مقدار ۱> برابر نبود، با <مقدار ۲> مقایسه می‌شود، اگر با <مقدار ۲> برابر بود، <دستورات ۲> اجرا می‌شوند و دستور break که بعد از آنها قرار دارد، کنترل برنامه را از ساختار switch خارج می‌کند. تا ... که عبارت محاسبه شده با یکی از مقادیر ذکر شده برابر نبود، عمل مقایسه با مقادیر بعدی ادامه می‌یابد. چنانچه مقدار عبارت با هیچکدام از مقادیر <مقدار ۱> ، <مقدار ۲> و ... برابر نبود، دستورات موجود در قسمت default اجرا می‌شوند و کنترل از ساختار switch خارج می‌گردد. در مورد ساختار switch به موارد زیر توجه کنید:

۱. ساختار switch می‌تواند فاقد بخش default باشد، در این صورت، اگر عبارت محاسبه شده، با هیچکدام از مقادیر ذکر شده برابر نباشد، هیچکدام از دستورات داخل switch اجرا نخواهد شد.

۲. مقادیر موجود در caseهای switch نمی‌توانند با هم مساوی باشند. یعنی هیچکدام از مقادیرِ <مقدار ۱> ، <مقدار ۲> و ... نباید مساوی باشند.

۳. اگر ثوابت کاراکتری در ساختار switch مورد مقایسه قرار گیرند، به مقادیر صحیح تبدیل می‌شوند.

۴. اگر در یک case از دستور break استفاده نشود، با مقدار case بعدی or می‌شود. برای اینکه دو یا چند شرط را در ساختار switch با هم or کنید، آنها را بدون دستور break پشت سر هم قرار دهید (به مثال ۲۱-۳ توجه کنید).

۵. یکی از تفاوتهای if و switch در این است که، در ساختار if می‌توان عبارت منطقی یا رابطه‌ای را مورد بررسی قرار داد ولی در ساختار switch فقط "مساوی بودنِ" مقادیر مورد بررسی قرار می‌گیرد.

۶. چند ساختار switch را می‌توان به طور تودرتو مورد استفاده قرار داد. یعنی هر یک از caseها می‌توانند دارای ساختار switch باشند.

مثال ۲۱-۳

برنامه‌ای که یک عملگر و دو عملوند را از ورودی خوانده، عملگر را بر روی عملوند اجرا می‌کند.

توضیح

در این برنامه، می‌خواهیم از عملگرهای '\' یا '/' برای تقسیم دو مقدار استفاده کنیم. لذا باید آنها را در دو case متوالی قرار دهیم، بدون اینکه دستور break در بین آنها وجود داشته باشد. بدین‌ترتیب، دو مقدار را در دو case مختلف با هم or می‌کنیم. پس از دریافت یک عملگر و دو عملوند، محاسبات انجام شده، در خروجی چاپ می‌شوند. اگر عملوند نامشخصی وارد شود، برنامه با صدور پیام، خاتمه می‌یابد.

متغیرهای برنامه

نام	هدف
num1	عملوند اول
num2	عملوند دوم
op	عملگر
flag	متغیر کمکی برای کنترل حلقه تکرار

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
int main()
{
  int num1, num2, flag = 1;
  char op;
  while(flag){
     clrscr();
     printf("Enter num1, num2:");
     scanf("%d%d", &num1, &num2);
     printf("Enter operator:");
     op = getche();
     switch(op){
        case '+' :
            printf("\n sum=%d", num1 + num2);
            break;
        case '-' :
            printf("\n minus=%d", num1 - num2);
            break;
        case '/' :
        case '\\' :
            printf("\n division=%6.2f", (float)num1 / num2);
            break;
        case '*' :
            printf("\n multiply=%d", num1 * num2);
            break;
        default:
            printf("\n operator is illegal.");
            printf("\n press a key to end.");
            flag = 0;
    }//end of switch
    getch();
  }//end of while
  return 0;
}
```

```
Enter num1, num2 : 44  20
Enter operator : \
division = 2.20
Enter num1, num2 : 34  2
Enter  operator : *
multiply = 68
Enter num1, num2 : 36  2
Enter operator : p
operator is illegall.
press a key to end.
```

تمرینات

۱. برنامه‌ای بنویسید که شماره دانشجویی و معدل تعداد n دانشجو را از ورودی خوانده، دانشجویی را که دومین معدل را از نظر بزرگی دارد، پیدا کند و به خروجی ببرد.

۲. برنامه‌ای بنویسید که اعدادی را از ورودی خوانده تشخیص دهد که آیا اعداد موردنظر، کامل هستند یا خیر. عددی کامل است که مجموع مقسوم‌علیه‌های آن (به جز خودش) برابر با آن عدد باشد. پس از بررسی هر عدد، برنامه باید از کاربر سئوال کند که می‌خواهد به کارش ادامه دهد یا خیر.

۳. برنامه‌ای بنویسید که خروجی زیر را در صفحه‌نمایش تولید کند.

```
*
**
***
****
*****
******
```

۴. برنامه‌ای بنویسید که کاراکتری را که نشان دهنده رنگی است، از ورودی خوانده، به شما بگوید که چه رنگی را می‌خواهد انتخاب کند. مثلاً اگر کاربر حرف 'r' را وارد کرد برنامه به او بگوید که دوست دارد رنگ قرمز را انتخاب کند. برای تمام موارد، حروف کوچک و بزرگ کنترل شود. مثل 'r' و 'R' برای رنگ قرمز.

۵. برنامه‌ای بنویسید که تعداد n جمله از سری فیبوناچی را تولید کند.

سری فیبوناچی = ۱ ۱ ۲ ۳ ۵ ۸ ۱۳ ...

۶. برنامه‌ای بنویسید که دو عدد صحیح مثبت را از ورودی خوانده، آنها را به روش تفریق بر هم تقسیم کند.

۷. برنامه‌ای بنویسید که شماره کارمندی و حقوق تعدادی از کارکنان مؤسسه‌ای را دریافت کرده، براساس تعرفه زیر، مالیات حقوق آنها را محاسبه کند و به خروجی ببرد. سپس مشخص کند، بیشترین دریافتی مربوط به کدام کارمند است.

از مالیات معاف	حقوق <400,000
۱۰ درصد نسبت به مازاد	400,001< حقوق <500,000
۱۵ درصد نسبت به مازاد	500,001< حقوق <700,000
۱۷ درصد نسبت به مازاد	حقوق >100,000

۸. خروجی دستورات زیر را بنویسید.

```
x = 5 ;
while (-- x > 0)
      printf("%3d", x) ;
```

۹. دستورات زیر را با for بنویسید.

```
int    i = 1 ;
while  (i <= 10)  {
         if (i < 5 && i != 2)
               printf("%c", 'x') ;
         i ++ ;
}
```

۱۰. برنامه‌ای بنویسید که حاصل عبارت زیر را محاسبه کند (n < 10).

$$1 + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{n!}$$

۱۱. برنامه‌ای بنویسید که عددی را در مبنای a گرفته، آن را به مبنای b ببرد.

۱۲. برنامه‌ای بنویسید که عددی از ۱ تا ۷ را از ورودی خوانده، روزی از هفته را که معادل با آن است در خروجی چاپ کند (با switch).

۱۳. برنامه‌ای بنویسید که سال تولد کاربر و سال فعلی را از ورودی خوانده، مشخص کند که او چند سال، چند ماه، چند روز، چند ساعت، چند دقیقه و چند ثانیه عمر کرده است.

۱۴. برنامه‌ای بنویسید که ضرایب معادله درجه دوم را از ورودی خوانده، معادله را حل کند.

معادله درجه دوم: $ax^2 + bx + c = 0$ $\qquad x = -\frac{b \pm \sqrt{b^2 - 4ac}}{2a}$

۱۵. برنامه‌ای بنویسید که با استفاده از حلقه‌های تودرتو خروجی زیر را تولید کند.

```
$ $ $ $ $ $
$ $ $ $ $ $
$ $ $ $ $ $
$ $ $ $ $ $
```

۱۶. برنامه‌ای بنویسید که دو مقدار اعشاری را از ورودی خوانده، و تفاضلِ حاصلضرب و حاصل تقسیم آنها را محاسبه کرده، به خروجی می‌برد. برنامه وقتی خاتمه می‌یابد که هر دو عدد اعشاری، صفر باشند.

۱۷. برنامه‌ای بنویسید که یک عدد اعشاری مثل 643.21 را خوانده، وارون آن را بیابد. وارون این عدد 12.346 است.

۱۸. برنامه‌ای بنویسید که عدد اعشاری را از ورودی خوانده، هر یک از قسمتهای صحیح و اعشاری آن را به صورت یک عدد صحیح به خروجی ببرد. به عنوان مثال، عدد ۱۳/۴۲ به صورت دو عدد صحیح ۱۳ و ۴۲ به خروجی برود.

تمرین حل شده

برنامه‌ای که اطلاعات تعدادی از دانشجویان را از ورودی خوانده، معدل هر دانشجو را محاسبه می‌کند. اگر معدل دانشجو بیش از ۱۷ باشد، به عنوان دانشجوی ممتاز و اگر کمتر از ۱۲ بود به عنوان دانشجوی مشروط به خروجی می‌رود. تعداد دانشجویان ممتاز و مشروط را نیز شمارش می‌کند (این برنامه بر روی دیسک به نام exam.cpp ذخیره شده است).

توضیح

شیوه محاسبه معدل دانشجویان به این صورت است که حاصلضرب واحد درس در نمره همان درس محاسبه شده، با هم جمع می‌شوند و مجموع واحدها نیز محاسبه می‌گردد. حاصل جمع نمرات بر مجموع واحدها تقسیم می‌شود و معدل محاسبه می‌گردد. به عنوان مثال، اگر دانشجویی سه درس با مشخصات زیر داشته باشد، معدل وی به صورت زیر محاسبه می‌گردد:

واحد	نمره
۲	۱۵
۳	۱۴
۲	۱۶

$$\text{معدل} = \frac{۲\times۱۵+۳\times۱۴+۲\times۱۶}{۲+۳+۲} = ۱۴/۸۵$$

در این برنامه از دو حلقه تکرار استفاده می‌شود. حلقه تکرار اول براساس تعداد دانشجو ساخته می‌شود که در آن، تعداد درس هر دانشجو با شماره دانشجویی خوانده می‌شود. حلقه تکرار دوم برحسب تعداد درس دانشجو ساخته می‌شود تا نمره و واحد هر درس از ورودی خوانده شود و مجموع نمرات و واحدها محاسبه گردد. پس از خروج از حلقه داخلی، با داشتن جمع نمرات و واحدها، معدل محاسبه می‌شود. سپس وضعیت دانشجو تشخیص داده شده (ممتاز یا مشروط) و معدل دانشجو به همراه شماره دانشجویی آن به خروجی می‌رود.

متغیرهای برنامه

نام	هدف
i,j	شمارنده‌های حلقه تکرار
stno	شماره دانشجویی
s1	جمع نمرات هر دانشجو
s2	جمع واحدهای هر دانشجو
ave	معدل هر دانشجو
c1	تعداد دانشجویان ممتاز
c2	تعداد دانشجویان مشروط
n	تعداد دانشجویان
m	تعداد درس هر دانشجو
grade	نمره هر درس
un	واحد هر درس

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int i, j, stno, s2, n, m, un, c1 = 0, c2 = 0;
  float s1, ave, grade;
  clrscr();
  printf("\nEnter number of student:");
  scanf("%d",&n);
  for(i = 0; i < n; i++){
     printf("\n Enter  stno and # of course:");
     scanf("%d%d", &stno, &m);
     s1 = 0;
     s2 = 0;
     for(j = 0; j < m; j++){
        printf("\n Enter grade and unit # %d :", j + 1);
        scanf("%f%d", &grade, &un);
        s1 += grade * un;
```

```
        s2 += un;
    }//end of for j
    ave = s1 / s2 ;
    if(ave >= 17){
       printf("stno=%d, ave=%5.2f, <exelent>", stno, ave);
       c1 ++;
    }
    else if (ave < 12){
       printf("stno=%d, ave=%5.2f, <probation>", stno, ave);
       c2 ++;
    }
    printf("\n press a key to continue...");
    getch();
  }//end of for i
  printf("\n exelent=%d, probation=%d",c1, c2);
  getch();
  return 0;
}
```

Enter number of student : 1
Enter stno and # of course : 125 2
Enter grade and unit # 1 : 17 3
Enter grade and unit # 2 : 18 4
stno = 125 , ave = 17.57 , <exelent>
press a key to continue ...
exelent = 1 , probation = 0

توابع و کلاس‌های حافظه

برنامه‌هایی که تاکنون نوشته شده‌اند، فقط شامل یک تابع اصلی به نام ()main بوده‌اند. در برنامه‌های طولانی و پیچیده که شامل چندین بخش منطقی و مستقل از هم هستند، بهتر است برای هر قسمت منطقی، برنامه جداگانه‌ای نوشته شود. برنامه‌ای که برای هر یک از بخشها نوشته می‌شود، **تابع** نام دارد. در واقع، تابع، برنامه‌ای است که برای حل بخشی از مسئله نوشته می‌شود.

تعدادی از توابع، که در اغلب برنامه‌ها مورد استفاده قرار می‌گیرند و کاربرد زیادی دارند، از قبل نوشته شده، به همراه کامپایلر C ارائه می‌شود. مثل توابع ()sin و ()cos که به ترتیب برای محاسبه سینوس و کسینوس زاویه به کار می‌روند و یا تابع ()clrscr که صفحه‌نمایش را پاک می‌کند. این توابع را توابع **کتابخانه‌ای** گوییم. در این کتاب، توابع کتابخانه‌ای دسته‌بندی شده در فصلهای مناسبی ارائه گردیده‌اند. ولی در سراسر کتاب، هرجا که نیاز به استفاده از آنها باشد، آنها را با توضیح مختصری به کار برده، شرح کامل آن را به فصل مربوط موکول می‌کنیم.

برنامه‌نویس برحسب ضرورت می‌تواند توابعی را بنویسد و در برنامه مورد استفاده قرار دهد. در این فصل، روش نوشتن این‌گونه توابع را مطرح خواهیم کرد. قبل از پرداختن به چگونگی نوشتن توابع، اهمیت آنها را در برنامه‌نویسی ساخت‌یافته مورد بررسی قرار می‌دهیم.

توابع و برنامه‌سازی ساخت‌یافته

با استفاده از توابع می‌توان برنامه‌های **ساخت‌یافته‌ای** نوشت. در این نوع برنامه‌ها، اعمال برنامه، توسط بخشهای مستقلی که تشکیل‌دهنده برنامه‌اند انجام می‌شود. این بخشهای مستقل همان توابع هستند. امتیازات برنامه‌نویسی ساخت‌یافته عبارت‌اند از:

۱. نوشتن برنامه‌های ساخت‌یافته آسان است، زیرا برنامه‌های پیچیده به بخشهای کوچکتری تقسیم می‌شوند و هر بخش توسط تابعی نوشته می‌شود. دستورالعملها و داده‌های موجود در تابع، مستقل از سایر بخشهای برنامه است.

۲. همکاری بین افراد را فراهم می‌کند. به طوری که افراد مختلف می‌توانند بخشهای مختلفی از برنامه را بنویسند.

۳. اشکالزدایی برنامه‌های ساخت‌یافته ساده‌تر است. اگر برنامه اشکالی داشته باشد، بررسی تابعی که این اشکال در آن به وجود آمده است، ساده است.

۴. برنامه‌نویسی ساخت‌یافته موجب صرفه‌جویی در وقت می‌شود. بدین‌ترتیب که، اگر تابعی بنویسید که عملی را در برنامه‌ای انجام دهد، می‌توانید آن تابع را در برنامه دیگری که به این عمل نیاز دارد، به کار برید. حتی اگر، با تغییر اندکی در توابع نوشته شده، بتوانید آنها را در برنامه‌های دیگر به کار ببرید، باز هم مقرون به صرفه است.

برای طراحی برنامه‌های ساخت‌یافته، ابتدا باید اعمالی را که در برنامه انجام می‌شوند، مشخص کنید. سپس بخشهایی از برنامه را که می‌توانند به طور مستقل انجام شوند، به صورت توابع طراحی و پیاده‌سازی کنید. برای این منظور، ساختار تابع، ورودیها و خروجیهای تابع را تعیین نماید. در اغلب برنامه‌های C، تابع ()main بسیار کوچک است و وظایف اصلی برنامه‌ها به عهده توابع است.

نوشتن توابع

برای نوشتن تابع باید اهداف تابع مشخص باشد. تابع چه وظیفه‌ای به عهده دارد، ورودیهای تابع چیست، و خروجیهای تابع کدامند. با دانستن این موارد، نوشتن تابع چندان دشوار نیست.

هر تابع دارای دو جنبه است، جنبه تعریف تابع و جنبه فراخوانی آن. جنبه **تعریف تابع**، مجموعه‌ای از دستورات است که عملکرد تابع را مشخص می‌کند و جنبه **فراخوانی تابع**، دستوری است که تابع را فراخوانی می‌کند. فراخوانی تابع با نام آن انجام می‌شود. نامگذاری برای تابع، از قانون نامگذاری برای متغیرها تبعیت می‌کند. توابع را باید پس از تابع ()main نوشت. ساختار و اجزای توابع C در شکل ۱-۴ آمده‌اند.

<نوع تابع> یکی از انواع موجود در C یا انواع دیگری است که توسط کاربر تعریف می‌شود (تعریف نوع جدید را در فصل ۷ می‌آموزید). اگر تابعی بخواهد مقداری را به تابع فراخوان برگرداند، آن مقدار در نام تابع قرار می‌گیرد. چون هر مقداری دارای نوع است، سپس نام تابع نیز باید دارای نوع باشد. اگر تابع هیچ مقداری را به برنامه فراخوان برنگرداند، نوع آن void منظور خواهد شد. پارامترها اطلاعاتی هستند که هنگام فراخوانی تابع، از برنامه فراخوان به آن ارسال می‌شوند. به عبارت دیگر، پارامترها وسیله‌ای برای تبادل اطلاعات بین تابع فراخوان و فراخوانی شونده هستند. اگر تعداد پارامترها بیش از یکی باشد، باید با کاما از هم جدا شوند. در لیست پارامترها، نوع هر یک از پارامترها نیز مشخص می‌شود. اطلاعاتی که هنگام فراخوانی تابع، در جلوی نام تابع (در داخل پرانتز) ظاهر می‌شوند، **آرگومان** تابع نام دارند. دقت داشته باشید که پارامترها متغیرهایی هستند که هنگام تعریف تابع، در جلوی نام تابع و در داخل پرانتز قرار می‌گیرند. برای به کارگیری تابع در برنامه، باید الگوی آن را در خارج از تابع ()main به کامپایلر اعلان کرد. **الگوی تابع**، مشخص می‌کند که تابع چگونه فراخوانی می‌شود. الگوی تابع نیز به صورت زیر مشخص می‌شود:

<نوع تابع> نام تابع (لیست پارامترها);

```
<نوع تابع>  نام تابع  (لیست پارامترها)
{
        <دستور ۱>
        <دستور ۲>
        .
        .
        دستور n
}
```

شکل ۱-۴ ساختار تابع.

به نمونه‌ای از برنامه با تابعی به نام ()sample که در شکل ۲-۴ آمده است توجه کنید.
در استفاده از توابع در C، موارد زیر را به خاطر داشته باشید:

۱. الگوی تمام توابع را قبل از تابع ()main اعلان کنید (هر چند که می‌توانید در تابع ()main نیز اعلان کنید).
۲. نوع توابع را تعیین نمایید.
۳. برای اجرای توابع، آنها را با نامشان فراخوانی کنید.
۴. الگوی تابع باید همانند عنوان تابع باشد (شکل ۲-۴). عنوان تابع، اولین سطر در تعریف تابع است.
۵. متغیرهای موردنیاز توابع را در داخل توابع تعریف کنید. هیچ تابعی نمی‌تواند از متغیرهای توابع دیگر استفاده کند. مگر اینکه از طریق پارامترها منتقل شوند.
۶. تعریف تابع در داخل تابع دیگر امکان‌پذیر نیست.
۷. هنگام فراخوانی توابع، دقت داشته باشید که تعداد و نوع پارامترها و آرگومان‌ها یکسان باشد.
۸. توابع از نظر تعداد مقادیری که می‌توانند به توابع فراخوان برگردانند به سه دسته تقسیم می‌شوند. ۱. توابعی که هیچ مقداری را برنمی‌گردانند (نوع void)، ۲. توابعی که یک مقدار را برمی‌گردانند، و ۳. توابعی که چندین مقدار را برمی‌گردانند. هر کدام از این توابع در ادامه مورد بحث قرار می‌گیرند.
۹. هنگام اعلان الگوی توابع، نیاز به ذکر اسامی پارامترها نیست. بلکه ذکر نوع آنها کفایت می‌کند. به عنوان مثال، الگوی تابع ()sample را که در شکل ۲-۴ آمده است، می‌توان به صورت زیر نوشت:

```
void    sample (int, int);
```

۱۰. اگر تابعی فاقد آرگومان است، به جای لیست آرگومان‌ها، کلمه void را قرار دهید.

```
#include    <stdio.h>
void    sample (int   x , int   y) ;        الگوی تابع
int     main()
{
        int   a , b ;
        ...                          آرگومان‌های تابع
        sample (a , b) ;                    فراخوانی تابع
        ...
        return  0 ;
}                                    پارامترهای تابع
void    sample (int   x , int   y)          عنوان تابع
{
    printf("\n x = %d, y = %d", x, y) ;     بدنه تابع
    ....
}
```

شکل ۲-۴
شیوه به کارگیری تابع در برنامه.

نکاتی در مورد نوشتن توابع

۱. ابتدا بدون پرداختن به جزئیات پیاده‌سازی توابع، آرگومان‌ها و نتیجه‌ای را که از توابع انتظار دارید، مشخص کرده برنامه اصلی را بنویسید. به عبارت دیگر، در قدم اول لازم نیست به جزئیات پیاده‌سازی تابع بپردازید. پس از نوشتن برنامه اصلی، توابع دیگر را بنویسید.

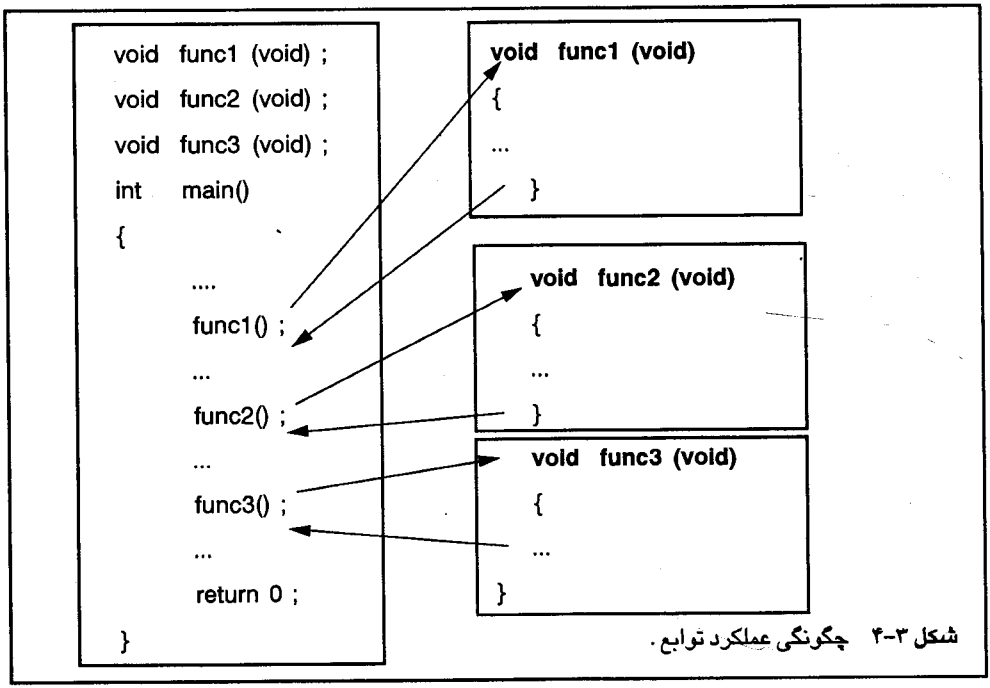
۲. توابع را طوری طراحی و پیاده‌سازی کنید که هر تابع فقط به آنچه که نیاز دارد دسترسی داشته باشد و بقیه قسمتهای برنامه و سایر اطلاعات، توسط توابع غیر مرتبط، قابل دستیابی نباشد. این موضوع را پنهان‌سازی اطلاعات گویند. برای این منظور، هر تابع باید یک نقطه ورود و یک نقطه خروج داشته باشد.

۳. برای ارتباط بین توابع، از آرگومان‌ها و پارامترها استفاده کنید.

تابع چگونه کار می‌کند

وقتی تابعی، توسط تابع دیگری فراخوانی می‌شود، دستورات آن تابع اجرا می‌شوند. پس از اجرای دستورات تابع، کنترل اجرای برنامه به برنامه فراخوان برمی‌گردد. پس از برگشت از تابع فراخوانی شده، اولین دستور بعد از فراخوانی تابع (در تابع فراخوان) اجرا می‌شود.

شکل ۳-۴ سه تابع را نمایش می‌دهد که هر کدام از آنها یک بار فراخوانی شده‌اند. با فراخوانی تابع، دستورات آن تابع اجرا می‌شوند و پس از اجرای تابع، کنترل به برنامه اصلی برمی‌گردد. هر تابع می‌تواند چندین بار فراخوانی شود.



شکل ۳-۴ چگونگی عملکرد توابع.

روشهای ارسال پارامترها به توابع

پارامترها را به دو طریق می‌توان از تابع فراخوان به تابع فراخوانی شونده ارسال کرد. این دو روش عبارتنداز:

۱. روش فراخوانی با مقدار (call by value)
۲. روش فراخوانی با ارجاع (call by reference)

روش فراخوانی، تعداد مقادیری را که توابع فراخوانی شونده می‌توانند برگردانند مشخص می‌کند. در روش فراخوانی با مقدار، دو دسته توابع می‌توانند وجود داشته باشند:

۱. توابعی که هیچ مقداری را برنمی‌گردانند
۲. توابعی که فقط یک مقدار را برمی‌گردانند. اما در روش فراخوانی با ارجاع، توابع می‌توانند چندین مقدار را به تابع فراخوان برگردانند.

در روش فراخوانی با مقدار، هنگام فراخوانی، **مقادیر** آرگومان‌ها در پارامترها کپی می‌شوند و هرگونه تغییری در پارامترها، تأثیری در آرگومان‌ها ندارد. اما در روش فراخوانی باارجاع، آدرس آرگومان‌ها به پارامترها منتقل می‌شود. بنابراین پارامترها باید قابلیت نگهداری آدرس را داشته باشند. نگهداری آدرسها به بحث اشاره‌گرها مربوط می‌شود. به همین دلیل، در این فصل به فراخوانی بامقدار پرداخته و فراخوانی باارجاع توابع را به فصل ۶ موکول می‌کنیم.

توابعی که هیچ مقداری را برنمی‌گردانند

ممکن‌است در برنامه، از توابعی استفاده کنیم که آن توابع، پس از فراخوانی، عملیات موردنظر را انجام دهند و خروجیهای مورد انتظار را تولید و چاپ نمایند و هیچ مقداری را به تابع فراخوان تحویل ندهند. در بسیاری از مسئله‌هایی که باکامپیوتر حل می‌شوند، اینگونه توابع به چشم می‌خورند. در اینجا با ارائه مثالی، نمونه‌ای از این تابع را بررسی می‌کنیم.

مثال ۱-۴

برنامه‌ای که با استفاده از تابعی، زمانی را برحسب ساعت، دقیقه و ثانیه خوانده، زمان را برحسب ثانیه چاپ می‌کند.

توضیح

هدف از این برنامه، آشنایی با توابعی است که هیچ مقداری را برنمی‌گردانند. تابع convert() توسط تابع main() فراخوانی می‌شود و پس از تعریف متغیرهای موردنیاز، آنها را از ورودی خوانده و پس از انجام محاسبات، نتیجه را به خروجی می‌برد. چون این تابع هیچ مقداری را برنمی‌گرداند، نوع آن void منظور شد و چون هیچگونه پارامتری ندارد در الگوی تابع و عنوان آن به جای لیست پارامترها، void قرارگرفته است. متغیر h برای ساعت، m برای دقیقه، s برای ثانیه و sum برای نگهداری کل زمان برحسب ثانیه مورد استفاده قرارگرفته است. در تابع main() سه دستور وجود دارد. دستور اول، تابع convert() را فراخوانی می‌کند. پس از اجرای تابع، کنترل به برنامه اصلی برمی‌گردد و اولین دستور بعد از فراخوانی این تابع، یعنی دستور getch() اجرا می‌شود. این تابع برنامه را متوقف می‌کند تا پس از مشاهده صفحه خروجی، کلیدی را فشار داده، به برنامه برگردید. دستور سوم نیز به تابع main() خاتمه می‌دهد.

```
#include <stdio.h>
#include <conio.h>
void convert(void);                    الگوی تابع
int main()
   {
    convert() ;
    return 0;
   }
//**************************
void convert(void)                     عنوان تابع
{
  int hours, minutes, second;
  long int time ;
  clrscr();
  printf("\n enter time to be convert:hour, minutes, second:") ;
  scanf("%d%d%d", &hours, &minutes, &second) ;
  time=(long int) (60 * hours + minutes) * 60 + second ;
  printf("\n time is :%ld seconds.",time) ;
  getch();
}
```

خروجی

```
enter time to be convert: hour, minutes, second : 12  45  26
time is : 45926 seconds.
```

مثال ۲-۴

برنامه‌ای که سه مقدار صحیح را از ورودی خوانده به تابعی ارسال می‌کند. تابع، بزرگترین مقدار را از بین سه مقدار پیدا کرده، به خروجی می‌برد.

توضیح

در این برنامه، سه مقدار صحیح x، y و m در برنامه اصلی از ورودی خوانده شده، به عنوان آرگومان تابع findmax() به آن ارسال می‌شوند. تابع findmax() از بین این سه مقدار، بزرگترین را پیدا کرده، به خروجی می‌برد. این تابع نیز، هیچ مقداری را به تابع فراخوان برنمی‌گرداند. متغیر maxp در تابع findmax()، بزرگترین عدد از سه عددِ وارد شده است.

```
#include <stdio.h>
#include <conio.h>
void findmax(int, int, int);           الگوی تابع
int main()
{
  int x, y, m;
  clrscr();
  printf("\nEnter three integer numbers:");
  scanf("%d%d%d", &x, &y, &m);
  findmax(x, y, m);
```

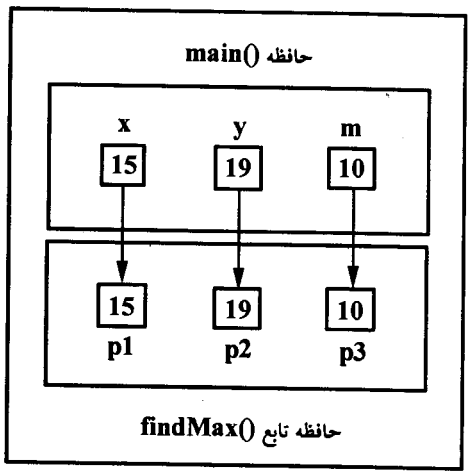
```
    return 0;
}
//**********************
void findmax(int p1, int p2, int p3)
{
  int maxp;
  maxp = (p1 > p2) ? p1 : p2;
  maxp = (p3 > maxp) ? p3 : maxp;
  printf("\maximum=%d", maxp);
  getch();
}
```

عنوان تابع

خروجی

```
Enter three integer numbers : 15  19  10
maximum = 19
```

برای آشنایی با چگونگی ارسال آرگومان‌های x، y و m به پارامترهای p1 و p2 و p3، شکل ۴-۴ را در نظر بگیرید. فرض کنید، مقادیر ۱۵، ۱۹ و ۱۰ به ترتیب در متغیرهای x، y و m قرار داده شده‌اند. پس از فراخوانی، مقدار آرگومان x در پارامتر p1، مقدار آرگومان y در پارامتر p2 و مقدار آرگومان m در پارامتر p3 کپی می‌شوند. از اینجا به بعد مقادیر p1، p2 و p3 در تابع findmax() قابل استفاده‌اند.



شکل ۴-۴ حافظه مربوط به مثال ۲-۴.

مثال ۳-۴

برنامه‌ای که در فراخوانی با مقدار، چگونگی تغییر در پارامترها و عدم تأثیر آنها را در آرگومان‌ها نشان می‌دهد.

توضیح

در این برنامه، دو مقدار x و y از ورودی خوانده شده مقادیر فعلی آنها در برنامه اصلی چاپ می‌شود. سپس این مقادیر، به عنوان آرگومان تابع f1()، به آن ارسال می‌شوند و f1() نیز آنها را چاپ می‌کند. سپس مقادیر پارامترها در تابع f1() تغییر کرده محتویات جدید چاپ می‌شود. پس از برگشت از فراخوانی تابع، محتویات x و y مجدداً چاپ می‌شوند. چاپ این محتویات نشان می‌دهد که تغییراتی که در پارامترها ایجاد شد، تأثیری در آرگومان‌ها ندارد.

```
#include <stdio.h>
#include <conio.h>
void f1(int, int);
int main()
{
  int x, y;
  clrscr();
  printf("\nEnter two integer numbers:");
  scanf("%d%d", &x, &y);
  printf(" You entered :x=%d, y=%d", x, y);
  f1(x, y);
  printf("\n After return from f1 :x=%d, y=%d", x, y);
  getch();
  return 0;
}
 //**********************
void f1(int x, int y)
{
  printf("\n f1 recieves : x=%d, y=%d", x, y);
  x ++;
  y ++;
  printf("\n new values in f1 : x=%d, y=%d", x, y);
}
```

خروجی

```
Enter two integer numbers : 10  15
You entered : x = 10 , y = 15
f1 recieves : x = 10 , y = 15
new values in f1 : x = 11 , y = 16
After return from f1 : x = 10 , y = 15
```

توابعی که یک مقدار را برمی‌گردانند

در بسیاری از مسئله‌هایی که توسط کامپیوتر حل می‌شوند، نیاز به نوشتن توابعی است که یک مقدار را برگردانند. مثل تابع ()sin که سینوس یک زاویه را برمی‌گرداند. اینگونه توابع، کاربردهای فراوانی دارند. برای نوشتن اینگونه توابع، نوع آنها را باید در الگوی تابع و عنوان تابع مشخص کرد. برای برگرداندن مقداری توسط تابع، از دستور return به صورتهای زیر استفاده می‌شود:

```
return     (<عبارت>)
return     <عبارت>
```

تفاوتی بین دو روش کاربرد return وجود ندارد. هم بدون پرانتز و هم با پرانتز قابل استفاده است. مقداری که توسط دستور return برگشت داده می‌شود، در نام تابع قرار می‌گیرد. در برنامه فراخوان، می‌توان نام تابع را به متغیری نسبت داد و از محتویات آن استفاده کرد. به عنوان مثال، اگر ()f1 یک تابع از نوع int و x متغیری از نوع int

باشد، دستور زیر، تابع f1() را فراخوانی کرده مقداری را که توسط دستور return در نام تابع قرار می‌گیرد، در x قرار می‌دهد. در اینجا تابع f1() فاقد آرگومان است:

```
x = f1() ;
```

مثال ۴-۴

برنامه‌ای که کاراکتری را از ورودی خوانده به تابعی ارسال می‌کند. این تابع، کاراکتر را بررسی کرده، چنانچه از حروف کوچک بود، آن را به حروف بزرگ تبدیل می‌کند و در هر حال نتیجه را به تابع فراخوان برمی‌گرداند.

توضیح

چون تابع باید یک مقدار کاراکتری را برگرداند، نوع تابع باید کاراکتری انتخاب شود. در اینجا، برخلاف مثال ۴-۴، تابع را در خود دستور printf() فراخوانی کرده، نتیجه را که در نام تابع قرار می‌گیرد، در خروجی چاپ کردیم. کدهای اسکی حروف کوچک از کدهای اسکی حروف بزرگ، ۳۲ واحد بیشتر است، لذا برای تبدیل حروف کوچک به حروف بزرگ، کافی است، ۳۲ واحد از حروف کوچک کم شود.

```
#include <stdio.h>
#include <conio.h>
char tocapital(char);
int main()
{
   char ch;
   clrscr();
   printf("\n Enter a character:");
   ch = getche();
   printf("\n Result is:%c", tocapital(ch));
   getch();
   return 0;
}
//*********************
char tocapital(char ch)
{
  if (ch >= 'a' && ch <= 'z')
      ch -=32;
  return ch;
}
```

خروجی

```
Enter a character : t
Result is : T
```

مثال ۵-۴

برنامه‌ای که شعاع دایره‌ای را از ورودی خوانده به تابعی ارسال می‌کند و تابع مساحت دایره را محاسبه کرده به برنامه اصلی برمی‌گرداند.

توضیح: برای محاسبه مساحت دایره از رابطه زیر استفاده می‌شود:

$$\text{مساحت دایره} = \pi * r * r$$

r شعاع دایره و π برابر با ۳/۱۴ است. چون مساحت دایره که توسط تابع محاسبه و برگردانده می‌شود از نوع اعشاری است، نوع تابع باید اعشاری باشد. مساحت دایره در تابع area() محاسبه شده در متغیر s قرار می‌گیرد و سپس با دستور return s ; به برنامه اصلی برگردانده می‌شود. این مقدار، در برنامه اصلی در متغیر s قرار می‌گیرد و سپس به خروجی می‌رود.

```
#include <stdio.h>
#include <conio.h>
float area(float);
int main()
{
  float  r, s;
  clrscr();
  printf("\nenter the radius:");
  scanf("%f", &r);
  s = area(r);
  printf("\narea = %5.2f", s);
  getch();
  return 0;
}
 //***********************
float area(float r)
{
  float s;
  s = r * r * 3.14;
  return s;
}
```

خروجی

```
enter the radius : 10
area = 314.00
```

مثال ۶-۴

برنامه‌ای که عددی را از ورودی خوانده، به تابعی تحویل می‌دهد. تابع، تشخیص می‌دهد که عدد موردنظر اول است یا خیر. اگر عدد موردنظر اول باشد مقدار یک (ارزش درستی) وگرنه مقدار صفر (ارزش نادرستی) را برمی‌گرداند. سپس برنامه، برای ادامه کار، از کاربر سئوال می‌کند. اگر پاسخ کاربر مثبت ('y') بود، برنامه عدد بعدی را دریافت می‌نماید.

توضیح

برای تشخیص عدد اول، آن را بر اعداد ۲ تا نصف آن عدد تقسیم می‌کنیم. اگر بر هیچکدام از این اعداد قابل قسمت نبود، اول است. num عدد مورد بررسی است و ans متغیر کمکی برای کنترل حلقه است. در تابع prime() متغیر temp متغیر کمکی برای کنترل حلقه for و متغیر i یک شمارنده است.

```
#include <stdio.h>
#include <conio.h>
int prime(int);
int main()
{
  int num;
  char ans;
  clrscr();
  while(1) {
     printf("\n Enter a number:") ;
     scanf("%d", &num);
     if(prime(num))
        printf("\n Number %d is prime.", num);
     else
        printf("\n Number %d is  not prime.", num);
     printf("\n Do you want to continue?(y/n):");
     ans = getche();
     if(ans != 'y')
         break;
  }// end of while
  getch();
  return 0;
}
//*********************
int prime(int num)
{
  int i, temp = 1;
  for(i = 2; (i <= num / 2)  && temp ; i++)
      if(num % i == 0)
         temp = 0;
  return temp;
}
```

خروجی

```
Enter a number : 97
Number 97 is prime
Do you want to continue ? (y/n) : y
Enter a number : 7865
Number 7865 is not prime.
Do you want to continue? (y/n) : n
```

متغیرهای محلی و عمومی

در برنامه‌هایی که تاکنون نوشته شد، متغیرهای موردنیاز هر تابع، در داخل آن تابع تعریف شده‌اند. متغیرهایی که در داخل تابعی تعریف می‌شوند، **متغیرهای محلی**[1] نامیده می‌شوند و فقط در همان تابع قابل استفاده‌اند. یعنی حوزه (scope) اینگونه متغیرها فقط در همان تابعی است که در آن تعریف می‌شوند. پس، هر متغیری که در تابع main() تعریف شد فقط در همین تابع قابل استفاده است و هر متغیری که در تابعی مثل f1() تعریف شد، فقط در تابع f1() قابل استفاده است. همانطور که قبلاً گفته شد، متغیرهای موجود در یک تابع را می‌توان از طریق آرگومان‌ها به تابع دیگری ارسال کرد و این، تناقضی با مفهوم متغیرهای محلی، که حوزه آنها در یک تابع است، ندارد.

اگر متغیرها در خارج از توابع و در بالای تابع main() تعریف شوند، در تمام توابع موجود در برنامه قابل استفاده‌اند و **متغیرهای عمومی**[2] نام دارند. با توجه به اینکه از طریق پارامترها می‌توان بین توابع ارتباط برقرار کرد، نیازی به استفاده از متغیرهای عمومی نخواهد بود. متغیرهای عمومی، درک و نگهداری برنامه‌ها را مشکل می‌کنند. بنابراین توصیه می‌شود، حتی الامکان از متغیرهای عمومی در برنامه استفاده نگردد، مگر اینکه در برنامه‌ای، اغلب توابع بخواهند از متغیری استفاده کنند. استفاده از ثوابت عمومی به دلیل عدم امکان تغییر آن توسط توابع، اثرات جانبی چندانی نمی‌تواند داشته باشد. لذا کاربرد ثوابت عمومی معمولتر از متغیرهای عمومی است. یکی از تفاوت‌های متغیرهای محلی و عمومی در این است که متغیرهای محلی تا زمانی که مقدار نگرفته‌اند، مقادیر آنها معتبر نیست ولی متغیرهای عمومی دارای مقدار اولیه صفر هستند. کاربرد متغیرهای محلی را در مثال‌های قبلی مشاهده کردید. در اینجا، برای آشنایی با متغیرهای عمومی، مثال‌هایی را در نظر می‌گیریم.

مثال ۷-۴

برنامه‌ای که نحوه استفاده از متغیرهای عمومی را نشان می‌دهد. این برنامه، مقدار ۵عدد صحیح را از ورودی خوانده، مجموع مربعات آنها را محاسبه کرده به خروجی می‌برد.

توضیح

در این برنامه، متغیرهای k، sq و sum به عنوان متغیرهای عمومی تعریف شده‌اند. این متغیرها در کلیه توابع موجود در برنامه قابل استفاده‌اند. تابع input() اعداد را از ورودی می‌خواند و تابع square() را فراخوانی می‌کند تا مربع هر عدد خوانده شده را محاسبه نماید. مربع عدد در متغیر عمومی square() قرار گرفته، با sum جمع می‌شود. sum مجموع مربعات اعداد است.

```
#include <stdio.h>
#include <conio.h>
int k, sq, sum ;
void input(void);
void square(void);
int main()
{
    int j , i = 5 ;
    clrscr();
    printf("\n Enter five number:") ;
    for(j = 0 ;j < i ;j++)
```

1. local variables 2. global variables

```
        input() ;
    printf("\n Sum of square is:%d",sum) ;
    getch();
    return 0;
}
void input(void)
{
   scanf("%d", &k) ;
   square() ;
   sum += sq ;
}
void square(void)
{
  sq = k * k ;
}
```

خروجی

```
Enter five numbers : 4 7 8 4 2
Sum of square is : 149
```

متغیرهای محلی همنام با متغیرهای عمومی

اگر در تابعی، متغیر محلی همنام با متغیر عمومی تعریف شود، در آن تابع، آن متغیر عمومی دیگر قابل استفاده نیست، بلکه از متغیر محلی استفاده می‌شود.

مثال ۸-۴

برنامه‌ای که چگونگی استفاده از متغیرهای محلی و عمومی همنام را نشان می‌دهد.

توضیح

در این برنامه، متغیر x به عنوان متغیر عمومی تعریف شده است. این متغیر در توابع ()main و ()func1 قابل استفاده است. ولی چون در تابع ()func2 متغیری محلی به نام x تعریف شد، متغیر عمومی x در این تابع قابل استفاده نیست (به خروجی برنامه توجه کنید).

```
#include <stdio.h>
#include <conio.h>
int x;                    متغیر عمومی
void func1(void);
void func2(void);
int main()
{
   clrscr();
   x = 100;
   func1();
   func2();
```

```
    printf("\n In main x is:%d", x);
    getch();
    return 0;
}
//**********************
void func1(void)
{
  printf("\n In func1 x is:%d\n", x);
}
//**********************
void func2(void)
{
  int x ;                              xمتغیر محلی است
  printf(" In func2 x changes:");
  for(x = 1; x < 6; x ++)
    printf(" x=%d ", x) ;
}
```

خروجی

```
In func1 x is : 100
In func2 x changes : x = 1  x = 2  x = 3  x = 4  x = 5
In main x is : 100
```

بازگشتی

بازگشتی به مفهومی گفته می‌شود که در آن، تابعی خودش را فراخوانی می‌کند. توابع می‌توانند به طور مستقیم یا غیرمستقیم خودشان را فراخوانی کنند. در روش مستقیم، یکی از دستورات تابع، فراخوانی خودش است. در روش غیرمستقیم، تابعی مثل f1()، تابع f2() را فراخوانی می‌کند و تابع f2() نیز به نوبه خود، تابع f1() را فراخوانی می‌نماید. برای ایجاد بازگشتی، الگوریتمی که توسط تابع پیاده‌سازی می‌شود، باید خصوصیت بازگشتی داشته باشد. طرح کلی الگوریتم‌های بازگشتی به صورت زیر است:

۱. یک یا چند حالت، که در آن، تابع، وظیفه خودش را به صورت بازگشتی انجام می‌دهد. یعنی این حالتها خاصیت بازگشتی دارند.

۲. یک یا چند حالت که در آن، تابع وظیفه خودش را بدون فراخوانی بازگشتی انجام می‌دهد. این حالت را **حالت‌های توقف**[1] گویند.

اغلب، با استفاده از یک دستور if مشخص می‌شود که کدامیک از این حالتها باید انجام شوند. برای اینکه، فراخوانیهای بازگشتی به اتمام برسد، باید حالت توقف اتفاق بیفتد. یعنی، هر فراخوانی تابع، سرانجام باید به حالت توقف ختم شود، در غیر این صورت، فراخوانی تابع خاتمه نمی‌یابد:

---

1. stop cases

```
if (به حالت توقف رسیدی)
        مسئله حالت توقف را حل کن
else
        تابع را بار دیگر فراخوانی کن
```

بازگشتی ابزار قدرتمندی در برنامه‌نویسی است و فهم آن برای برنامه‌نویسان مبتدی قدری دشوار است. به همین دلیل به جنبه‌های تکنیکی مربوط به مسئله بازگشتی نمی‌پردازیم، بلکه جهت آشنایی با مفهوم بازگشتی و آمادگی برای درس ساختمان داده‌ها، چند مثال ساده را بررسی می‌کنیم. ابتدا با ذکر مثالی، حالت بازگشتی و حالت توقف را در بازگشتی تشریح می‌کنیم.

حالت‌های بازگشتی و توقف در محاسبه فاکتوریل

فاکتوریل عدد صحیح و مثبت n به صورت زیر تعریف می‌شود:

$$n! = 1 \qquad \text{اگر } n = 0$$
$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n \qquad \text{اگر } n > 0$$

براساس این تعریف، ۴×۳×۲×۱=!۴ که می‌توان آن را به صورت ۴×!۳=!۴ نوشت. در واقع برای هر n>0 داریم، n!=n*(n-1)!. لذا با علامت‌گذاری ریاضی، این را می‌توان به صورت زیر نوشت:

$$n! = 1 \qquad \text{اگر } n = 0$$
$$n! = n * (n-1)! \qquad \text{اگر } n > 0$$

این تعریف ممکن است عجیب به نظر برسد، زیرا فاکتوریل را برحسب خودش تعریف می‌کند. براساس این تعریف، برای محاسبه فاکتوریل هر عدد، باید فاکتوریل عدد قبلی محاسبه شده باشد. حالت توقف در محاسبه فاکتوریل، حالتی است که n برابر با صفر است. لذا برای محاسبه فاکتوریل عدد ۳ باید فاکتوریل عدد ۲ محاسبه شود. برای محاسبه فاکتوریل عدد ۲ باید فاکتوریل عدد ۱ محاسبه شود. برای محاسبه فاکتوریل عدد یک باید فاکتوریل عدد صفر محاسبه شود و فاکتوریل عدد صفر، یک تعریف شده است. این مکانیزم، مسئله بازگشتی را ایجاب می‌کند. این روند را به صورت زیر خلاصه می‌کنیم:

$$3! = 3 \times 2!$$
$$2! = 2 \times 1!$$
$$1! = 1 \times 0!$$
$$0! = 1$$

اگر تابعی داشته باشیم که عددی مثل n را به عنوان آرگومان بپذیرد و فاکتوریل آن را برگرداند، این تابع باید خودش را آنقدر فراخوانی کند تا به حالت توقف برسد. با رسیدن به حالت توقف، به برنامه فراخوان برمی‌گردد.

مثال ۹-۴

برنامه‌ای که عددی مثل n را از ورودی خوانده، به کمک تابع بازگشتی، فاکتوریل آن را محاسبه می‌کند (به ردیابی تابع محاسبه فاکتوریل که پس از خروجی برنامه آمده است توجه کنید).

```
#include <stdio.h>
#include <conio.h>
unsigned long fact(int) ;
int main()
{
    int m ;
    clrscr();
    printf("\n enter a positive integer5 number:");
    scanf("%d", &m);
    printf("\n number=%d, fact= %ld", m, fact(m));
    getch();
    return 0;
}
 //**********************
unsigned long fact(int x)
{
  if(x != 0)
    return(x * fact(x - 1)) ;
  return 1 ;
}
```

خروجي

```
enter a positive integer number : 5
number = 5 , fact = 120
```

ردیابی تابع محاسبه فاکتوریل: اکنون تابع ()fact را به ازای n=4 دنبال می‌کنیم.

فراخوانی اول: چون 0=!4است، تابع به ازای x=3 فراخوانی می‌شود.

```
unsigned   long   fact(4)
{
        if (4! = 0)
            return (4 * fact(4-1)) ;
        return 1 ;
}
```

فراخوانی دوم: چون 0=!3است، تابع به ازای x=2 فراخوانی می‌شود.

```
unsigned   long   fact(3)
{
        if (3! = 0)
            return (3 * fact(3-1)) ;
        return 1 ;
}
```

فراخوانی سوم : چون 0=!2است، تابع به ازای x=1فراخوانی می‌شود.

```
unsigned   long   fact(2)
{
        if (2! = 0)
            return (2 * fact(2-1)) ;
        return 1 ;
}
```

فراخوانی چهارم : چون 0=!1است، تابع به ازای x=0فراخوانی می‌شود.

```
unsigned   long   fact(1)
{
        if (1! = 0)
            return (1 * fact(1-1)) ;
        return 1 ;
}
```

فراخوانی پنجم : چون 0=0است، دستور return 1 اجرا می‌شود.

```
unsigned   long   fact(0)
{
        if (0! = 0)
            return (1 * fact(0-1)) ;
        return 1 ;
}
```

در فراخوانی پنجم، دستور return 1 ، مقدار یک را به نام تابع در فراخوانی چهارم برمی‌گرداند. این مقدار، در یک ضرب شده ((1-1)fact*1) ، نتیجه آن (۱) به فراخوانی سوم برمی‌گردد. مقدار یک در ۲ ضرب شده، حاصل آن (۲) به فراخوانی دوم برمی‌گردد. این مقدار در ۳ ضرب شده، حاصل آن (۶) به فراخوانی اول برمی‌گردد. این مقدار در ۴ ضرب شده، نتیجه آن (۲۴) به تابع فراخواننده برمی‌گردد. برنامه این مقدار را در خروجی چاپ می‌کند.

مثال ۱۰-۴

برنامه‌ای که دو عدد صحیح مثبت را از ورودی خوانده، حاصلضرب آنها را با استفاده از جمع محاسبه می‌کند.

توضیح

برای محاسبه حاصلضرب دو عدد صحیح و مثبت a و bکافی است، a را bبار با خودش جمع کنید و یا b را aبار با خودش جمع کنید:

a * b = a	**اگر b = 1**
a * b = a * (b -1) + a	**اگر b > 1**

در این تعریف، برای محاسبه ۳×۶ابتدا باید ۲×۶را محاسبه کرد و سپس ۶را به حاصل ۲×۶اضافه نمود. برای محاسبه ۲×۶باید ۱×۶را محاسبه کرد و سپس ۶ را به آن اضافه نمود. اما ۱×۶برابر با ۶است. بنابراین، در این الگوریتم بازگشتی، حالت توقف، حالت b = 1است و حالات دیگر، حالات بازگشتی‌اند. (به ردیابی این تابع بازگشتی که در ادامه آمده است توجه کنید).

```
#include <stdio.h>
#include <conio.h>
int product(int x, int y);
int main()
{
   int x, y;
   clrscr();
   printf("\nEnter two integer numbers:");
   scanf("%d%d", &x, &y);
   printf("\nThier product is :%d", product(x, y));
   getch();
   return 0;
}
 //************************
int product(int x, int y)
{
  if(y == 1)
     return x;
  return(x + product(x, y - 1));
}
```

خروجی

```
Enter two integer numbers : 10  9
Thier product is : 90
```

ردیابی تابع product() به ازای x = 4 , y = 3

(۱)

```
if    (3 == 1))
      return 4 ;
return (4 + product (4, 3 - 1)) ;
```

(۲)

```
if    (2 == 1)
      return 4 ;
return (4 + product (4, 2 - 1)) ;
```

(۳)

```
if    (1 == 1)
      return 4 ;
return (4 + product (4, 1 - 1)) ;
```

در فراخوانی سوم، شرط if (1 == 1) درست است و در نتیجه مقدار ۴ به فراخوانی دوم برمی‌گردد. این مقدار با مقدار ۴ جمع شده (۸) به فراخوانی اول برمی‌گردد. مقدار ۸ با مقدار ۴ جمع شده (۱۲) به تابع اصلی برمی‌گردد. این مقدار در خروجی چاپ می‌شود.

مثال ۱۱-۴

برنامه‌ای که عددی را از ورودی خوانده، هر یک از ارقام آن را در یک سطر از صفحه‌نمایش چاپ می‌کند. تفکیک و چاپ ارقام عدد توسط تابع بازگشتی صورت می‌گیرد.

توضیح

اگر عدد کوچکتر از ۱۰ باشد، آن عدد در خروجی چاپ می‌شود و تابع به برنامه فراخوان برمی‌گردد. چنانچه عدد بزرگتر از ۱۰ باشد، فراخوانی بازگشتی ادامه می‌یابد. به عنوان تمرین، این برنامه را ردیابی کنید.

```
#include <stdio.h>
#include <conio.h>
void write_v(int);
int main()
{
  int x;
  clrscr();
  printf("enter an integer number:");
  scanf("%d",&x);
  write_v(x);
  getch();
  return 0;
}
//*****************
void write_v(int x)
{
  if (x < 10)
     printf("%d\n", x);
  else {
     write_v(x / 10);
     printf("%d\n", x % 10);
  }
}
```

خروجی

```
enter an integer number : 123
1
2
3
```

کلاس‌های حافظه و حوزه متغیرها

کلاس حافظه[1]، ویژگیی از متغیر است که دو چیز را در مورد متغیر مشخص می‌کند:

۱. حوزه متغیر (scope)
۲. طول عمر متغیر (life time)

منظور از حوزه متغیر این‌است‌که، یک متغیر در چه جاهایی از برنامه قابل دستیابی است. به عبارت دیگر، کلاس حافظه متغیری مثل x، مشخص می‌کند که این متغیر در چه جاهایی از برنامه قابل استفاده است و در چه جاهایی نمی‌توان از آن استفاده کرد. منظور از طول عمر متغیر، مدت زمانی است که متغیر در حافظه وجود دارد. به عبارت دیگر، متغیر کی به وجود می‌آید و کی از بین می‌رود.
چهار نوع کلاس حافظه در C قابل استفاده است که عبارت‌انداز:

۱. کلاس حافظه اتوماتیک (automatic)
۲. کلاس حافظه ثبات (register)
۳. کلاس حافظه استاتیک (static)
۴. کلاس حافظه خارجی (extern)

برای تعیین کلاس حافظه برای متغیرها، به صورت زیر عمل می‌شود:

<کلاس حافظه> <نوع متغیر> نام متغیر;

به عنوان مثال، دستورات زیر، کلاس حافظه متغیر x را static و کلاس حافظه y را register تعیین می‌کنند.

```
static  int  x ;
register  char  y ;
```

کلاس حافظه اتوماتیک

متغیرهایی که در داخل تابعی تعریف می‌شوند (متغیرهای محلی)، با فراخوانی تابع ایجاد می‌شوند و با خاتمه اجرای تابع از بین می‌روند. این متغیرها را متغیرهای با کلاس حافظه **اتوماتیک** گویند، زیرا هنگام ورود به تابع به طور اتوماتیک ایجاد می‌شوند و هنگام خروج از تابع به طور اتوماتیک از بین می‌روند. برای تعیین کلاس حافظه اتوماتیک، از کلمه کلیدی auto استفاده می‌شود. به عنوان مثال، دستور زیر، متغیر m را با کلاس حافظه اتوماتیک تعریف می‌کند:

```
auto  float  m ;
```

با توجه به اینکه کلیه متغیرهای محلی دارای کلاس حافظه اتوماتیک هستند، نیازی به استفاده از کلمه auto نیست. در واقع، در این کتاب نیز از آن استفاده نخواهیم کرد. متغیرهای با کلاس حافظه اتوماتیک ویژگیهای زیر را دارند:

۱. فقط در همان تابعی که تعریف می‌شوند قابل استفاده‌اند (حوزه متغیر).
۲. با فراخوانی تابع حافظه به آنها اختصاص می‌یابد (به وجود می‌آیند) و با خاتمه اجرای تابع، از بین می‌روند (طول عمر متغیر).

---

1. storage class

کلاس حافظه ثبات

کلاس حافظه ثبات، به کامپایلر پیشنهاد می‌کند که متغیر اتوماتیک را در ثبات پردازنده (CPU) قرار دهد. بنابراین، حوزه و طول عمر متغیرهای کلاس حافظه ثبات مثل اتوماتیک است. همانطور که می‌دانید، ثباتها حافظه‌هایی در داخل CPU هستند. کامپیوتر برای انجام محاسبات بر روی متغیرها، آنها را از حافظه RAM به ثباتها ارسال می‌کند و پس از انجام محاسبات، به حافظه RAM برمی‌گرداند. اگر کامپایلر بتواند، متغیرهایی را در ثبات نگه دارد، سرعت انجام محاسبات با آن متغیرها افزایش می‌یابد.

اما تعداد ثباتهای پردازنده محدود بوده، پردازنده برای انجام وظایف خود از آنها استفاده می‌کند. در صورتی که ثباتهای خالی وجود داشته باشد، می‌توان از آنها برای ذخیره متغیرها استفاده کرد. به همین دلیل، تعیین کلاس حافظه ثبات، حالت پیشنهاد به کامپایلر دارد. اگر کلاس حافظه متغیری را ثبات تعیین کنید ولی پردازنده نتواند ثبات خالی را در اختیار آن متغیر قرار دهد، کلاس حافظه ثبات از آن متغیر حذف می‌شود. لذا پیشنهاد می‌شود فقط متغیر مهم برنامه که نیاز به انجام سریع محاسبات با آن است با این کلاس حافظه مشخص شود، مثل اندیس حلقه تکرار. برای تعیین کلاس ثبات از کلمه کلیدی register استفاده می‌گردد: به عنوان مثال، دستور زیر، متغیر p را با کلاس‌های حافظه ثبات توصیف می‌کند:

```
register    int    p ;
```

کلاس حافظه ثبات محدودیتهایی دارد که عبارت‌انداز :

۱. فقط برای متغیرهای محلی قابل استفاده است.
۲. انواع کاراکتر، صحیح و اشاره‌گر می‌توانند با کلاس حافظه ثبات تعریف شوند. در مورد سایر انواع باید به مستندات کامپایلری که از آن استفاده می‌کنید، مراجعه نمایید.
۳. چون تعداد ثبات پردازنده محدود است، تعداد اندکی از متغیرها می‌توانند با این کلاس حافظه تعریف شوند.
۴. آدرس متغیرهایی با کلاس حافظه ثبات، معنی ندارد.

کلاس حافظه استاتیک

متغیرهای استاتیک به دو دسته تقسیم می‌شوند: ۱. متغیرهای استاتیک محلی ۲. متغیرهای استاتیک عمومی. متغیرهای استاتیک محلی در داخل تابع و متغیرهای استاتیک عمومی در خارج از تابع تعریف می‌شوند. مقدار اولیه متغیرهای استاتیک محلی و استاتیک عمومی، صفر است.

متغیرهای استاتیک محلی

متغیرهای استاتیک محلی دارای ویژگیهای زیر هستند:

۱. فقط در همان تابعی که تعریف می‌شوند قابل استفاده‌اند (حوزه متغیر).
۲. هنگام فراخوانی تابع ایجاد می‌شوند و هنگام خروج از تابع، آخرین مقدار خودشان را حفظ می‌کنند.
۳. فقط یک بار مقدار اولیه می‌گیرند.

متغیرهای استاتیک محلی، کاربردهای فراوانی می‌توانند داشته باشند. به عنوان مثال، فرض کنید تابعی به نام

output() دارید که وظیفه آن چاپ اطلاعات در صفحه‌نمایش است. این تابع در هر بار فراخوانی، یک سطر از اطلاعات را می‌نویسد. تابع باید شماره سطری را که اطلاعات بعدی در آن نوشته می‌شود نگهداری کند. اگر متغیری که شماره سطر را نگهداری می‌کند، یک متغیر اتوماتیک باشد، با خروج از تابع، مقدار قبلی آن از بین می‌رود. اگر متغیر نگهدارنده شماره سطر، با کلاس حافظه استاتیک تعریف شود، هنگام خروج از تابع آخرین مقدار خود را نگه می‌دارد و دفعه بعد که فراخوانی می‌شود، قابل استفاده است.

مثال ۱۲-۴

برنامه‌ای که با تولید خروجی ساده، تفاوتهای بین متغیرهای اتوماتیک و استاتیک را توصیف می‌کند.

توضیح

در تابع test، متغیر x با کلاس حافظه اتوماتیک (در حالت فرضی)، متغیر y با کلاس حافظه استاتیک و متغیر i در برنامه اصلی با کلاس حافظه ثبات تعریف شده‌اند (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
#include <conio.h>
void test(void);
int main()
{
  register int i;
  clrscr();
  for(i = 0; i < 5; i ++)
      test();
  getch();
  return 0;
}
//*****************
void test(void)
{
  int x = 0; //automatic variable
  static int y = 0;
  printf("\n auto x=%d, static y=%d",x, y);
  x ++;
  y ++;
}
```

خروجی

```
auto  x = 0 , static  y = 0
auto  x = 0 , static  y = 1
auto  x = 0 , static  y = 2
auto  x = 0 , static  y = 3
auto  x = 0 , static  y = 4
```

تحلیل مثال ۱۲-۴

تابع test() به تعداد ۵ بار فراخوانی می‌شود. در فراخوانی اول مقدار اولیه x و yبرابر با صفر می‌شود. همین مقادیر چاپ شده، به هر متغیر یک واحد اضافه می‌گردد. هنگام خروج از تابع، xاز بین می‌رود ولی yدر حافظه وجود دارد و مقدارش یک است. توجه داشته باشید که با اینکه yدر حافظه وجود دارد، توسط هیچ تابع دیگری قابل استفاده نیست. در فراخوانی بعد، x دوباره صفر می‌شود ولی yمقدار اولیه نمی‌گیرد (مقدار آن در فراخوانی قبلی یک بوده است). مقدار xبرابر با صفر و مقدار yبرابر با یک است که در خروجی چاپ می‌شوند. به x و yهر کدام یک واحد اضافه می‌شود. xاز بین می‌رود ولی yمقدار ۲ را نگه می‌دارد. این روند تا پایان حلقه تکرار ادامه می‌یابد.

متغیرهای استاتیک عمومی

متغیرهای استاتیک عمومی، در خارج از توابع تعریف می‌شوند. این متغیرها در توابعی که بعد از آنها تعریف می‌شوند قابل استفاده‌اند. به عنوان مثال، در شکل ۵-۴ متغیرهای x و yقبل از دو تابع f1() و f2() تعریف شده‌اند. لذا این متغیرها در تابع main() قابل دستیابی نیستند ولی توابع f1() و f2() آنها را می‌شناسند.

استفاده از متغیرهای استاتیک عمومی از یک طرف موجب می‌شود تا متغیرها در جایی که به آنها نیاز هست تعریف شوند و از طرف دیگر، فقط توابعی که به آنها نیاز دارند می‌توانند از آنها استفاده کنند.

```
void      f1(void) ;
void      f1(void) ;
#include          <stdio.h>
int   main () {
    ...
    f1() ;
    ...
    f2() ;
    ...
    return 0 ;
}
static  int  x , y ;
void    f1(void)
{
    ...
}
void    f2(void)
{
    ...
}
```

شکل ۵-۴ تعریف متغیرهای استاتیک عمومی.

کلاس حافظه خارجی

متغیرهایی که در خارج از توابع تعریف می‌شوند (متغیرهای عمومی) دارای کلاس حافظه خارجی‌اند. ویژگی‌های این متغیرها عبارتند از:

۱. با شروع اجرای برنامه ایجاد شوند و تا پایان اجرای برنامه حضور دارند (طول عمر)
۲. در سرتاسر برنامه قابل استفاده‌اند (حوزه متغیر)

تفاوت کلاس حافظه خارجی و کلاس حافظه استاتیک عمومی

متغیرهای عمومی که در خارج از تابع ()main تعریف می‌شوند، دارای کلاس حافظه خارجی‌اند. اگر متغیرهایی با کلاس حافظه استاتیک عمومی قبل از تابع ()main تعریف شوند، این متغیرها با متغیرهای عمومی چه تفاوتهایی دارند؟ به عنوان مثال در دستورات زیر، x و y متغیرهای استاتیک عمومی و m و n متغیرهای عمومی‌اند:

```
static int   x , y ;
int m , n ;
int main()
{
   ...
}
```

اگر کل برنامه C تنها در یک فایل وجود داشته باشد، درواقع، هیچ تفاوتی بین این دو تعریف وجود ندارد. اما گاهی ممکن‌است برنامه‌های شما بسیار طولانی باشند و مجبور شوید که بخشهایی از برنامه را در چند فایل قرار دهید (شکل ۶-۴). در این صورت، متغیرهای استاتیک عمومی فقط در یک فایل (در همان فایلی که تعریف می‌شوند) و متغیرهای عمومی در تمام فایل‌ها قابل دستیابی‌اند و برای استفاده از آنها در فایل‌های دیگر، باید آنها را با دستور extern به کامپایلر اعلان کرد. دستور extern به کامپایلر می‌گوید که، برای این متغیرها حافظه جدیدی در نظر نگیرد، بلکه از همان حافظه‌ای که در فایل دیگر به آنها اختصاص داده شد استفاده کند. لذا، در این شکل، متغیرهای x و y فقط در فایل t1.cpp ولی متغیرهای m و n در هر دو فایل t1.cpp و t2.cpp قابل استفاده‌اند. برای چگونگی ایجاد چند فایل برای برنامه و پیونددادن آنها به یکدیگر، به فصل ۱۷ مراجعه شود.

فایل t1.cpp

```
static    int    x , y ;
int       m , n ;
void      f1(void)
int main()
{
...
}
void      f2(void)
{
    m = 5 ;
    n = 10 ;
    x = 6 ;
}
```

فایل t2.cpp

```
extern      int      m , n ;
void        f2(void)
{
m = n / m ;
}
void        f3(void)
{
m = 50 ;
}
```

شکل ۶-۴ کاربرد متغیرهای عمومی و متغیرهای استاتیک عمومی.

کلمه کلیدی extern را برای متغیرهای عمومی در داخل یک فایل نیز می‌توان به کار برد ولی ضرورتی ندارد. اگر این کار انجام شود، فقط جهت اطلاع برنامه‌نویس است و می‌تواند در نگهداری برنامه کمک کند. بدین‌ترتیب، برنامه‌نویس با دیدن این دستور در یک تابع، متوجه می‌شود که این تابع از متغیرهای عمومی استفاده می‌کند(شکل ۷-۴).

```
void      f1(void) ;
int       x , y ;
int  main()
{
    extern      int     x , y ;
    ...
    f1() ;
    return 0 ;
}
void      f1(void)
{
    extern      int     x , y ;
    ...
}
```

شکل ۷-۴

کاربرد extern جهت اعلان متغیرهای عمومی.

نکته‌ای راجع به الگوی تابع

اگر توابعی که در برنامه مورد استفاده قرار می‌گیرند، قبل از تابع ()main تعریف شوند، نیاز به ذکر الگوی آنها نیست.

مثال ۱۳-۴

برنامه‌ای که موجودی اولیه حساب یک مشتری بانک، نرخ بهره بانکی و تعداد ماههایی که موجودی در حساب بوده است را از ورودی می‌خواند، سپس بهره‌ای را که پس از آن مدت به پول مشتری تعلق می‌گیرد محاسبه می‌کند. محاسبه بهره به عهده تابع است. بهره به طور ماهانه محاسبه شده، و برای محاسبه بهره ماه بعدی، بهره ماه قبلی به موجودی قبلی اضافه می‌گردد.

توضیح

چون مقداری که توسط تابع برگردانده می‌شود (میزان بهره)، از نوع float است، تابع باید از نوع float معرفی شود. نرخ بهره را بدون درصد وارد کنید (مثل عدد ۵). برنامه آن را به صورت درصد در نظر می‌گیرد.

الگوریتم برنامه

۱. سود هر ماه را بدین صورت حساب کن

سود ماه = موجودی ماه × نرخ بهره / ۱۰۰

۲. سود ماه گذشته را به موجودی ماه گذشته اضافه کن تا موجودی فعلی به دست آید.

۳. بهره هر ماه را با هم جمع کن

۴. اگر برای همهٔ ماهها حساب نشده، برو به قدم ۱

۵. خروجی را تولید کن

۶. توقف کن

متغیرهای برنامه

تابع	متغیر	هدف
main()	mon	تعداد ماههای پس‌انداز (ورودی)
	balance	موجودی اولیه (ورودی)
	sben	مجموع سود چند ماهه (محاسبه)
	rate	نرخ بهره (ورودی)
calculate()	balance	پارامتر (موجودی اولیه)
	rate	پارامتر (نرخ بهره)
	mon	پارامتر (تعداد ماههای پس انداز)
	sben	متغیر محلی (کل بهره)
	ben	بهره هر ماه (متغیر کمکی)

```
#include <stdio.h>
#include <conio.h>
float calculate(float balance, float rate, int mon)
{
   int i;
   float ben, sben = 0;
   for(i = 0; i < mon; i++){
       ben = balance * rate / 100;
       balance += ben;
       sben +=ben;
   }
   return sben;
}
 //*****************************
int main()
{
   int  mon;
   float balance, rate, sben;
   clrscr();
   printf("\nEnter balance, rate, mon:");
    scanf("%f%f%d",&balance,&rate,&mon);
   sben = calculate(balance, rate, mon);
   printf("\n benefit=%7.1f, balance=%7.1f", sben, balance);
   printf("\n new balance is: %7.1f", balance + sben);
   getch();
   return 0;
}
```

خروجی

```
Enter balance, rate, mon : 5000  10  4
benefit = 2320.5 , balance = 5000.0
new balance is : 7320.5
```

مثال ۱۴-۴

تجزیه و تحلیل مسئله قیمت‌گذاری در فروشگاه

فروشنده فروشگاه عمده فروشی می‌خواهد قیمت خرده‌فروشی کالاهای خود را تعیین کند. او کالاها را به دو دسته تقسیم می‌کند: ۱. کالاهایی که انتظار دارد حداکثر یک هفته در فروشگاه بمانند ۲. کالاهایی که انتظار می‌رود بیش از یک هفته در فروشگاه بمانند. افزایش قیمت کالای دسته اول ۵٪ و افزایش قیمت کالای دسته دوم ۱۰٪ است. پس از تجزیه و تحلیل مسئله، برنامه‌ای برای آن خواهیم نوشت.

ورودی : قیمت عمده‌فروشی و مدت زمانی که انتظار می‌رود کالا به فروش برسد (برحسب روز).

خروجی : قیمت خرده‌فروشی کالا

تحلیل مسئله : این مسئله از سه قسمت تشکیل شده است :

۱. ورود داده‌ها
۲. محاسبه قیمت خرده‌فروشی
۳. چاپ نتایج

سعی می‌کنیم هر یک از سه قسمت مسئله را با یک تابع پیاده‌سازی کنیم. اما اگر بخش ورود داده‌ها را با یک تابع پیاده‌سازی کنیم، تابع باید دو مقدار را از ورودی خوانده، آن دو مقدار را به برنامه اصلی برگرداند. این کار مستلزم ارسال آرگومان‌ها از طریق ارجاع است، ولی این موضوع هنوز مورد بررسی قرار نگرفته است، به همین دلیل، بخش ورود داده‌ها را به عهده برنامه اصلی می‌گذاریم.

تابع ()price قیمت خرده‌فروشی را محاسبه می‌کند. این تابع دو پارامتر دارد که عبارت‌انداز: wholesale (قیمت عمده‌فروشی) و times (مدت زمان انتظار فروش کالا بر حسب روز). تابع ()give_output نتایج برنامه را به خروجی می‌برد. این تابع سه پارامتر دارد که عبارت‌انداز: retail (قیمت خرده‌فروشی)، times (مدت زمان انتظار فروش کالا بر حسب روز) و wholesale (قیمت عمده‌فروشی). الگوهای این دو تابع عبارت‌انداز:

```
float  price (float cost, int times) ;
void   give_output (float wholesale, int times, float retail);
```

طراحی الگوریتم

محاسباتی که در برنامه باید انجام شود، در تابع ()price صورت می‌گیرد. الگوریتم این محاسبه به صورت زیر است:

```
if (times  < = 7)
     return (wholesale + wholesale * 0.05) ;
else
     return (wholesale + wholesale * 0.10) ;
```

برنامه‌نویسی : در این برنامه از سه ثابت به شرح زیر استفاده می‌شود:

```
const    float    LOW     = 0.05
const    float    HIGH    = 0.10
const    int      NUM     = 7
```

افزایش قیمت ٪۵
افزایش قیمت ٪۱۰
مرز بین ٪۵ و ٪۱۰

توجه داشته باشید که برای اینکه مشخص شود ثوابت LOW، HIGH و NUM ثوابت خارجی‌اند، با حروف بزرگ نوشته شده‌اند. برنامه طوری طراحی شد که پس از دریافت قیمت عمده‌فروشی و مدت زمان انتظار برای فروش کالا، قیمت خرده‌فروشی را تعیین کرده، به خروجی می‌برد. سپس از کاربر سئوال می‌کند که آیا به کارش ادامه می‌دهد یا خیر. اگر 'y' وارد شود، برنامه ادامه می‌یابد.

متغیرهای برنامه

نام برنامه	نام متغیر	هدف
عمومی	LOW	افزایش قیمت، اگر کالا کمتر یا مساوی ۷ روز در انبار باشد
	HIGH	افزایش قیمت، اگر کالا بیشتر از ۷ روز در انبار باشد
	NUM	تعداد روزهایی که کالا در انبار می‌ماند
main()	wholesale	قیمت عمده‌فروشی کالا
	retail	قیمت خرده‌فروشی که باید محاسبه شود
	times	تعداد روزهای انبارشدن کالا (ورودی)

زیربرنامه‌های price() و give_output() ، از متغیرهای برنامه اصلی به عنوان پارامتر و ثوابت عمومی استفاده می‌کنند.

زیربرنامه price() قیمت خرده‌فروشی کالا را محاسبه می‌کند و زیربرنامه give_output() قیمت عمده‌فروشی، قیمت خرده‌فروشی و روزهایی راکه کالا انبار می‌شود. به خروجی می‌برد.

```
#include <stdio.h>
#include <conio.h>
const float LOW = 0.05;
const float HIGH = 0.10;
const int NUM = 7;
float price(float, int);
void give_output(float, int, float);
int main()
{
  float wholesale, retail;
  int  times;
  char ans;
  do{
    clrscr();
    printf("\n enter wholesale, self time:");
    scanf("%f%d", &wholesale, &times);
    retail = price(wholesale, times);
    give_output(wholesale, times, retail);
    printf("\n do you want to continue?(y/n):");
    ans = getche();
    if(ans != 'y')
          break;
  } while(1);
  getch();
  return 0;
}
```

```
//*****************************
float price(float wholesale, int times)
{
  if(times <= NUM)
      return(wholesale + wholesale * LOW);
  return(wholesale + HIGH * wholesale);
}
//*****************************
void give_output(float wholesale, int times, float retail)
{
  printf("\n wholesale =%7.1f, times=%3d", wholesale, times);
  printf("\n retail cost = %7.1f", retail);
}
```

خروجی

```
enter wholesale, self time : 4500  5
wholesale = 4500.0 , times = 5
retail cost = 4725.0
do you want to continue ? (y/n) : n
```

تمرینات

۱. منظور از الگوی تابع چیست و در کجا باید تعریف شود؟

۲. روش کار تابع را توضیح دهید (مکانیزم فراخوانی و برگشت).

۳. چه نکات مهمی را هنگام تعریف آرگومان‌ها و پارامترها باید رعایت کرد؟

۴. جنبه‌های مختلف یک تابع را نام ببرید و توصیف کنید.

۵. توابع از نظر تعداد مقادیری که برمی‌گردانند به چند دسته تقسیم می‌شوند؟ تعداد مقادیری که توابع برمی‌گردانند چه ارتباطی با روشهای ارسال آرگومان‌ها دارد؟

۶. چه تفاوتی بین آرگومان و پارامتر وجود دارد؟

۷. برنامه‌نویسی ساخت‌یافته چیست و توابع چه کمکی به این نوع برنامه‌نویسی می‌کنند؟

۸. کلاس‌های حافظه چه امتیازاتی می‌توانند داشته باشند؟ انواع کلاس‌های حافظه را در C تشریح کنید.

۹. منظور از بازگشتی چیست ؟ مثالی غیر از آنچه که در کتاب آمده است ارائه دهید.

۱۰. برنامه‌ای بنویسید که عددی را از ورودی خوانده، به کمک تابع بازگشتی، شمارش معکوس از آن عدد به یک را انجام دهد (شماره‌ها را در خروجی چاپ کند).

۱۱. برنامه‌ای بنویسید که n جمله از سری فیبوناجی را به روش بازگشتی تولید کند.

۱۲. برنامه‌ای بنویسید که فاصله‌ای را برحسب فوت و اینچ دریافت کرده، معادل آن را برحسب متر و سانتیمتر بیان کند. هر فوت ۰/۳۰۴۸ متر، یک متر ۱۰۰ سانتیمتر و هر فوت ۱۲ اینچ است. حداقل از سه تابع استفاده کنید، یکی برای ورودی، یکی برای انجام محاسبات و دیگری برای خروجی.

۱۳. برنامه‌ای بنویسید که ۳ مقدار اعشاری را از ورودی خوانده، به تابعی ارسال کند و تابع میانگین آنها را محاسبه کرده، برگرداند.

۱۴. برنامه‌ای بنویسید که دمای هوا را بر حسب سانتیگراد خوانده، به تابعی تحویل دهد و تابع، آن را به درجه فارنهایت تبدیل کرده، برگرداند.

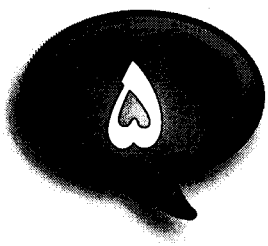
۱۵. برنامه‌ای بنویسید که ضرایب معادله درجه دومی را از ورودی خوانده، آنها را به تابعی ارسال کند. تابع معادله را حل کرده، جوابهای آن را به خروجی ببرد (به برنامه اصلی برنمی‌گرداند).

۱۶. برنامه‌ای بنویسید که دو عدد اعشاری را از ورودی خوانده، به تابعی ارسال کند. تابع، تفاضل، حاصلضرب و حاصل تقسیم آنها را محاسبه کرده، به برنامه ارسال کند.

۱۷. برنامه‌ای بنویسید که کاراکتری را از ورودی خوانده، آن را از ستون i تا ستون j صفحه‌نمایش چاپ کند. برنامه، کاراکتر، و مقدار i و j را به تابعی ارسال می‌کند و تابع عمل چاپ کاراکتر را انجام می‌دهد.

۱۸. تابعی بنویسید که سه آرگومان را بپذیرد: آرگومان اول، یک کاراکتر، و آرگومان دوم مشخص می‌کند که این کاراکتر در هر سطر از صفحه‌نمایش چند بار باید چاپ شود و آرگومان سوم مشخص می‌کند که این کاراکتر در چند سطر باید نوشته شود. برنامه‌ای بنویسید که از این تابع استفاده کند.

۱۹. تابعی بنویسید که دو آرگومان را بپذیرد و آرگومان اول را به توان آرگومان دوم برساند. آرگومان اول یک مقدار double و آرگومان دوم یک مقدار صحیح مثبت یا منفی است. برنامه‌ای بنویسید که از آن استفاده کند.



آرایه‌ها و رشته‌ها

در فصل‌های گذشته، داده‌ها را در متغیرها و ثوابت ذخیره و در موقع لزوم بازیابی کردیم. اما اگر تعداد داده‌ها زیاد باشد، استفاده از متغیرها به روشی که تاکنون دیدید، امکان‌پذیر نیست. به عنوان مثال فرض کنید می‌خواهیم معدل ۱۰ دانشجو را از صفحه‌کلید خوانده، بزرگترین معدل را پیدا کرده، سپس اختلاف هر یک از معدلها را با بزرگترین معدل محاسبه نموده، در خروجی چاپ کنیم. این کار مستلزم این‌است‌که هر یک از ۱۰ معدل دانشجو در حافظه نگهداری شود، زیرا تا همهٔ معدل‌ها خوانده نشده‌اند، بزرگترین معدل پیدا نمی‌شود تا تفاضل آن با سایر معدلها محاسبه شود. برای این کار می‌توانیم ۱۰ متغیر از نوع float را تعریف کنیم که هر متغیر یک معدل را نگهداری می‌کند. متغیر دیگری برای نگهداری بیشترین معدل لازم است. این روش نه تنها مناسب نیست، بلکه در حالتی که مثلاً تعداد معدلها به ۱۰۰ برسد، امکان‌پذیر نیست. برای حل اینگونه مسئله‌ها، از آرایه استفاده می‌کنیم. **آرایه** مجموعه‌ای از عناصر همنوع است. هر آرایه دارای نامی است که مانند متغیرهای معمولی نامگذاری می‌شود. برای دسترسی به عناصر آرایه از متغیری به نام **اندیس** استفاده می‌گردد. به همین دلیل، آرایه را متغیر **اندیس‌دار** نیز می‌گویند.

آرایه‌های یک بعدی

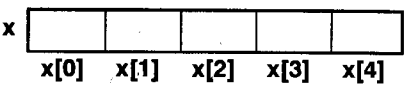
در آرایه یک بعدی که نام دیگر آن لیست است، با یک اندیس می‌توان به عناصر آرایه دست یافت. آرایه‌های یک بعدی در C به صورت زیر تعریف می‌شوند:

[طول آرایه] نام آرایه نوع آرایه

نوع آرایه یکی از انواع قابل قبول در C است. نام آرایه، برای دسترسی به عناصر آرایه مورد استفاده قرار می‌گیرد. طول آرایه با یک عدد صحیح مثبت مشخص می‌گردد. به عنوان مثال، دستور زیر آرایه‌ای به نام x و به طول ۵ از اعداد صحیح تعریف می‌کند.

```
int   x [5] ;
```

اندیس آرایه‌ها در C از صفر شروع می‌شود. به‌این‌ترتیب، عناصر آرایه x به صورت زیر بازیابی می‌شوند (توجه داشته باشید که عناصر آرایه در محلهای متوالی حافظه ذخیره می‌شوند).



میزان حافظه‌ای که به آرایه اختصاص داده می‌شود، به طریق زیر محاسبه می‌گردد (برحسب بایت):

طول آرایه × (طول نوع آرایه) = میزان حافظه آرایه

به عنوان مثال، میزان حافظه آرایه x عبارت است از ۱۰ = ۵ × ۲ (با فرض اینکه طول int برابر با ۲ بایت باشد). با استفاده از اندیس آرایه می‌توان به عناصر آرایه مقدار داد. به عنوان مثال، دستور زیر مقدار ۵ را در دومین عنصر آرایه قرار می‌دهد، که در آن، x نام آرایه، ۱ اندیس و ۵ مقداری است که در x[1] قرار می‌گیرد.

```
x[1] = 5 ;
```

مثال ۱-۵

برنامه‌ای که معدل ۵ دانشجو را از ورودی خوانده در آرایه‌ای قرار می‌دهد و بیشترین معدل و محل وجود آن را پیدا می‌کند و به خروجی می‌برد.

توضیح

حلقه تکرار اول، معدل دانشجویان را خوانده در آرایه قرار می‌دهد. برای پیدا کردن بیشترین معدل، فرض می‌کنیم که اولین عنصر آرایه، بزرگترین معدل است. به همین دلیل قبل از حلقه تکرار دوم، ave[0] در amax و اندیس آن، یعنی ۰ در p قرار می‌گیرد. سپس amax را در یک حلقه تکرار با عناصر آرایه مقایسه کرده، عنصر بزرگتر را در amax و اندیس آن را در p قرار می‌دهیم.

متغیرهای برنامه

نام	هدف
n	ثابتی به طول ۵ برای تعیین طول آرایه
ave	آرایه‌ای به طول n برای نگهداری معدل
amax	بزرگترین معدل
i	شمارنده حلقه تکرار
p	محل وجود بزرگترین عنصر در آرایه

```
#include <stdio.h>
#include <conio.h>
int main()
{
  const int n = 5;
  float ave[n], amax = 0;
  int i, p;
  clrscr();
  for(i = 0; i < n; i++){
     printf(" enter an average:");
     scanf("%f", &ave[i]);
  }
  amax = ave[0];
  p = 0;
  for(i = 1; i < n; i++)
     if(ave[i] > amax) {
        amax = ave[i];
        p = i;
     }
  printf("\n max = %5.2f, position = %d", amax, p + 1);
  getch();
  return 0;
}
```

خروجی

```
enter an average :12
enter an average :15
enter an average :12
enter an average :17
enter an average :16
max = 17.00 , position = 4
```

مثال ۲-۵

برنامه‌ای که تعداد ۱۰ عدد صحیح را از ورودی می‌خواند و ابتدا اعداد منفی و سپس اعداد مثبت را به خروجی می‌برد. تعداد اعداد مثبت و منفی را نیز مشخص می‌کند.

توضیح

با توجه به اینکه، ابتدا تمام اعداد منفی و سپس تمام اعداد مثبت باید به خروجی بروند، این اعداد باید در آرایه ذخیره گردند. پس از ذخیره آنها در آرایه، با یک حلقه تکرار، ابتدا اعداد منفی را به خروجی می‌بریم و سپس با حلقه تکرار دیگر، اعداد مثبت را به خروجی منتقل می‌کنیم.

متغیرهای برنامه

نام	هدف
n	ثابتی به طول ۱۰
arr	آرایه‌ای به طول n
c1	تعداد اعداد منفی
c2	تعداد اعداد مثبت
i	شمارنده حلقه تکرار

```
#include <stdio.h>
#include <conio.h>
int main() {
   const int n = 10;
   int arr[n], i, c1 = 0, c2 = 0 ;
   clrscr();
   printf("\n Enter %d numbers and press ENTER:\n", n);
   for(i = 0; i < n; i++)
     scanf("%d",&arr[i]);
   printf("\ negavives are:");
   for(i = 0 ; i < n ; i++)
      if(arr[i] < 0){
         printf("%d, ",arr[i]);
         c1 ++;
      } //end of if
   printf("\n positives are:");
   for(i = 0 ; i < n ; i++)
      if(arr[i] > 0){
         printf("%d, ",arr[i]);
         c2 ++;
      }//end of if
   printf("\n number of negative = %d", c1);
   printf("\n number of positive = %d", c2);
   getch();
   return 0;
}
```

خروجی

```
Enter 10 numbers and press ENTER :
12 -3 -6 44 -7 11 -7 5 19 9
negatives are : -3, -6, -7, -7,
positives are : 12, 44, 11, 5, 19, 9,
number of negative = 4
number of positive = 6
```

مثال ۳-۵

برنامه‌ای که تعداد ۵عدد را از ورودی خوانده، سپس آنها را به ترتیب معکوس در آرایه دیگری قرار داده، نتیجه را به خروجی می‌برد.

توضیح

چون تمام اعداد باید خوانده شوند تا یکی یکی، از آخرین عدد به اولین عدد به خروجی بروند، باید در آرایه‌ای نگهداری شوند. حلقه تکرار اول، عناصر آرایه x را می‌خواند. حلقه تکرار دوم، که اندیس آن از ۴ تا صفر است، عناصر آرایه x را از آخرین عنصر به اولین عنصر، به آرایه y منتقل می‌کند. به همین دلیل، متغیر j که به‌عنوان اندیس آرایه y است، از صفر شروع می‌شود و هر بار یک واحد به آن اضافه می‌گردد. پس اندیس آرایه x از ۴ به صفر کاهش می‌یابد و اندیس آرایه y از صفر به ۴ افزایش می‌یابد.

```
#include <stdio.h>
#include <conio.h>
int main() {
  int x[5], y[5], i, j;
  clrscr();
  for(i = 0; i < 5 ; i++) {
     printf("Enter number %d : ", i + 1);
     scanf("%d",&x[i]);
  }
  j = 0;
  printf("Number in inverse:\n");
  for(i = 4; i >= 0; i --) {
     y[j] = x[i];
     printf("%3d", y[j]);
     j ++;
  }
  printf("\nPress a key to continue...");
  getch();
  return 0; }
```

خروجی

```
Enter number 1 : 9
Enter number 2 : 11
Enter number 3 : 15
Enter number 4 :14
Enter number 5 :8
Number in inverse:
8  14  15  11 9
press a key to continue...
```

آرایه یک بعدی به عنوان آرگومان تابع

همانطور که می‌توانیم متغیرهای معمولی را به عنوان آرگومان تابع انتخاب کنیم، آرایه‌ها نیز می‌توانند به عنوان آرگومان، به توابع ارسال شوند. برای ارسال آرایه به تابع، باید نام تابع به عنوان آرگومان ذکر شود. اگر آرایه به عنوان آرگومان تابع باشد، پارامتر معادل آن می‌تواند به سه صورت زیر تعریف شود:

۱. آرایه‌ای با طول مشخص

۲. آرایه‌ای با طول نامشخص که در این صورت بهتر است طول آرایه به عنوان آرگومان دیگری منتقل شود.

۳. اشاره‌گر (چون اشاره‌گرها در فصل ۶ بررسی می‌شوند، این روش را در فصل ۶ بحث می‌کنیم).

```
void      func1 (int x[]) ;
void      func2 (int x[] , int len) ;
int       main()
{
   int       x [10] ;
   ...
   func1 (x) ;
   ...
   func2 (x , 10) ;
   return  0 ;
}
void func1 (int x[10])
{
   ...
}
void  func2 (int  x[] , int  len)
{
   ...
}
```

شکل ۱-۵ ارسال آرایه های یک بعدی به توابع.

برای پی بردن به مفاهیم گفته شده، به شکل ۱-۵ مراجعه کنید. در این شکل، آرایه x به عنوان آرگومان دو تابع ()func1 و ()func2 انتخاب شد و پارامترهای معادل آن در هر دو تابع تعریف شده‌اند. در تابع ()func1، پارامتر به صورت آرایه‌ای با طول معین و در تابع ()func2 به صورت آرایه‌ای با طول نامعین تعریف شده است.

مثال ۴-۵

برنامه‌ای که معدل تعداد ۱۰ دانشجو را خوانده، در آرایه‌ای قرار می‌دهد، سپس مشخص می‌کند که چه معدلی بیشتر از همه تکرار شد و دفعات تکرار آن را مشخص می‌کند.

توضیح

برای حل این مسئله، باید تعداد دفعات تکرار هر معدل محاسبه شود و با دفعات تکرار معدل قبلی مقایسه گردد. چنانچه تکرار معدل جدید، بیشتر از تکرار معدل قبلی باشد، تعداد تکرار این عنصر، به عنوان بیشترین تکرار انتخاب شده، این معدل نیز نگهداری می‌شود. عناصر آرایه در برنامه اصلی خوانده می‌شوند و آرایه و طول آن به عنوان پارامتر به تابع ()findt منتقل می‌شوند.

متغیرهای برنامه

متغیر	هدف
k	ثابتی به طول ۱۰ برای طول آرایه
max_count	بیشترین تکرار
max_value	معدلی که بیشترین تکرار را دارد
current_value	معدل مورد بررسی
current_count	تعداد تکرار معدل مورد بررسی
i, j	شمارنده حلقه
arr	آرایه‌ای به طول k

```
#include <stdio.h>
#include <conio.h>
void findt(float arr[], int k);
int main()
{
   const int k = 10;
   float arr[k] ;
   int i;
   clrscr();
   printf("\n enter %d mead and press Enter:\n", k);
   for (i = 0; i < k; i++)
     scanf("%f", &arr[i]) ;
   findt(arr, k);
   return 0;
}
 //*************************
void findt(float arr[], int k)
{
   int max_count = -1 ;
   float max_value;
   int  i, j , current_count;
   float current_value ;
   for (i = 0; i < k; i ++){
       current_value = arr[i] ;
       current_count = 0 ;
       for(j = 0; j < k; j++)
          if (arr[j] == current_value)
              current_count++ ;
       if (current_count > max_count) {
           max_count = current_count ;
           max_value = current_value ;
       }//end of if
   }//end of for (i= 0; i < k ...)
   printf("\n maximum iteration of ave");
   printf(" %5.2f is %d", max_value, max_count);
   getch();
}
```

خروجی

```
enter 10 mead and press Enter :
15.5  14  18  19  14.5  15.5  17  15.5  18  19
maximum iteration of ave 15.50 is 3
```

مرتب سازی آرایه‌ها

اعمال مرتب‌سازی و جستجو، عمومی‌ترین اعمالی هستند که در برنامه‌نویسی انجام می‌شوند. در بسیاری از موارد، مرتب‌سازی از ضروریات برنامه‌نویسی است. به عنوان مثال، فرض کنید، دفترچه تلفنی دارید که حاوی تلفن ۵۰۰ نفر است و می‌خواهید شماره تلفن یک نفر را در دفترچه تلفن پیدا کنید. اگر دفترچه تلفن براساس نام افراد و برحسب حروف الفبا مرتب باشد، به راحتی می‌توانید نام فرد و در نتیجه شماره تلفن وی را بیابید. اما اگر دفترچه تلفن براساس نام افراد مرتب نباشد، پیداکردن شماره تلفن یک نفر از بین ۵۰۰ نفر کار دشواری است. از این موارد، بسیار زیاد است. به همین دلیل، عمل مرتب‌سازی درصد زیادی از کار کامپیوتر را تشکیل می‌دهد. روشهای متعددی برای مرتب‌سازی آرایه‌ها وجود دارد. هدف ما در این فصل آشنایی با مفهوم مرتب‌سازی و بررسی یک الگوریتم ساده است. مرتب‌سازی به طور مفصل در فصل جداگانه‌ای بحث می‌شود.

مرتب‌سازی ممکن است به طور صعودی (از کوچک به بزرگ) و یا به طور نزولی (از بزرگ به کوچک) باشد.

مرتب‌سازی صعودی **x[0] < x[1] < x[2] < ... < x[n]**

مرتب‌سازی نزولی **x[0] > x[1] > x[2] > ... > x[n]**

مرتب‌سازی حبابی

یکی از خصوصیات این روش مرتب‌سازی، سهولت درک برنامه‌نویسی آن است. از طرفی، از بین تمام مرتب‌سازیهای موجود، کارایی این مرتب‌سازی از سایر روشها کمتر است. در این روش، باید چندین مرتبه در طول آرایه حرکت کرد و هر بار، عنصری را با عنصر بعدی مقایسه نمود و در صورتی که عنصر اول از عنصر دوم بزرگتر باشد، جای آنها را عوض کرد (برای مرتب‌سازی صعودی). به عنوان مثال، اعداد زیر را در نظر بگیرید:

4 9 3 1

در **مرحله اول**، مقایسه‌های زیر صورت می‌گیرد:

x[0] با x[1] یعنی (۴ با ۹) جابجایی انجام نمی‌شود.

x[1] با x[2] یعنی (۹ با ۳) جابجایی انجام می‌شود: **4 3 9 1**

x[2] با x[3] یعنی (۹ با ۱) جابجایی انجام می‌شود: **4 3 1 9**

همانطور که می‌بینید، در مرحله اول، بزرگترین عنصر آرایه به انتهای آرایه منتقل شده است. در مرحله بعدی این عنصر در مقایسه شرکت نمی‌کند. لذا طول آرایه یک واحد کمتر می‌شود.

در **مرحله دوم** مقایسه‌های زیر صورت می‌گیرد:

x[0] با x[1] یعنی ۴ با ۳ جابجایی صورت می‌گیرد: **3 4 1 9**

x[1] با x[2] یعنی ۴ با ۱ جابجایی صورت می‌گیرد: **3 1 4 9**

در این مرحله دو مقایسه صورت می‌گیرد و آرایه به صورت زیر درمی‌آید.

3 1 4 9

در **مرحله سوم** مقایسه‌های زیر انجام می‌شود:

x[0] با x[1] یعنی ۳ با ۱ جابجایی صورت می‌گیرد: 1 3 4 9

لیست حاصل، لیست مرتبی است و کار مقایسه عناصر به اتمام می‌رسد.

1 3 4 9

توجه داشته باشید که اگر طول آرایه را در هر مرحله n در نظر بگیریم، تعداد مقایسه‌ها در هر مرحله، n-1 خواهد بود. از طرفی، تعداد مراحلی که مقایسه باید انجام شود، از تعداد عناصر آرایه یک واحد کمتر است. یعنی اگر آرایه ۱۰۰ عنصری داشته باشیم، حداکثر در ۹۹ مرحله این آرایه مرتب می‌شود.

مثال ۵-۵

برنامه‌ای که تعدادی عدد را از ورودی خوانده آنها را به روش حبابی مرتب می‌کند و نتیجه را به خروجی می‌برد.

توضیح

در این برنامه، از سه تابع استفاده شده است. تابع ginput() برای خواندن عناصر آرایه، تابع bubble() برای مرتب‌سازی و تابع göutput() برای نوشتن عناصر آرایه مورد استفاده قرار می‌گیرد.

متغیرهای برنامه

برنامه	متغیر	هدف
main	k	ثابتی به طول ۷ برای تعداد اعداد
	temp	آرایه k عنصری برای نگهداری اعداد
ginput()	temp	آرایه اعداد (پارامتر)
	len	طول آرایه (پارامتر)
	i	شمارنده حلقه
goutput()	i,j	شمارنده حلقه
	temp	آرایه اعداد (پارامتر)
	len	طول آرایه (پارامتر)
	item	متغیر کمکی برای جابجایی عناصر

```
#include <stdio.h>
#include <conio.h>
void ginput(int [], int);
void bubble(int [], int);
void goutput(int [], int);
int main()
{
   const int k=7 ;
   int temp[7];
   clrscr();
   ginput(temp, k);
   bubble (temp, k);
   printf("the sorted data are :\n") ;
   goutput(temp, k);
   getch();
   return 0;
```

```
}
//*************************
void ginput(int temp[], int len)
{
   int i;
   for(i=0; i < len; i++) {
       printf("enter number %d:",i+1);
       scanf("%d",&temp[i]);
   }
}
//**********************
void bubble(int temp[], int len)
{
   int i, j, item;
   for(i = len - 1 ; i > 0; i --)
      for(j = 0; j < i ; j++)
         if(temp[j] > temp[j + 1]) {
            item = temp[j] ;
            temp[j] = temp[j + 1];
            temp[j + 1] = item ;
         }//end of if
}
//*************************
void goutput(int temp[], int len)
{
   int i;
   for(i=0 ; i < len; i++)
     printf(" %3d",temp[i]) ;
}
```

خروجی

```
enter number 1 : 25
enter number 2 : 57
enter number 3 : 48
enter number 4 : 37
enter number 5 : 12
enter number 6 : 92
enter number 7 : 86
the sorted data are:
12  25  37  48  57  86  92
```

جستجو در آرایه

یکی دیگر از اعمالی که در کامپیوتر بسیار زیاد مورد استفاده قرار می‌گیرد، عمل جستجو است. مثل جستجوی نام یک دانشجو در لیست دانشجویان یک دانشگاه و جستجوی نام فردی در دفترچه تلفن. جستجو در آرایه‌های نامرتب به صورت ترتیبی و در آرایه‌های مرتب به صورت دودویی (binary) انجام می‌شود.

جستجوی ترتیبی

در این روش، عنصر مورد جستجو، با هر یک از عناصر آرایه مقایسه می‌شود. چنانچه با هم برابر بودند، جستجو به پایان می‌رسد وگرنه، عمل مقایسه با عنصر بعدی انجام می‌شود. این روند تا یافتن عنصر موردنظر و یا جستجوی کل آرایه ادامه می‌یابد. اگر عنصر موردنظر در آرایه پیدا شود، می‌گوییم جستجو موفق است.

مثال ۶-۵

برنامه‌ای که شماره دانشجویی تعدادی از دانشجویان را از ورودی خوانده در آرایه‌ای قرار می‌دهد. سپس یک شماره دانشجویی را از ورودی می‌خواند و آن را در آرایه جستجو می‌نماید.

توضیح

در این برنامه، از دو تابع استفاده شده است. تابع ginput() برای خواندن عناصر آرایه و تابع lsearch() برای جستجو در آرایه مورد استفاده قرار می‌گیرد. اگر عنصر مورد جستجو در آرایه وجود داشته باشد، اندیس آرایه وگرنه عدد ۱- توسط تابع lsearch() برگردانده می‌شود.

متغیرهای برنامه

برنامه	متغیر	هدف
main	k	ثابتی به طول ۵ برای تعداد دانشجو
	st	آرایه‌ای به طول k
	no	شماره دانشجویی مورد جستجو
ginput()	st	آرایه‌ای از شماره دانشجویان
	len	طول آرایه
	i	شمارنده حلقه
lsearch()	st	آرایه‌ای برای شماره دانشجویان
	len	طول آرایه
	no	عنصر مورد جستجو
	i	شمارنده حلقه

```
#include <stdio.h>
#include <conio.h>
void ginput(int [], int);
int lsearch(int [], int, int);
int main()
{
   const int k = 5 ;
   int st[k], no;
   clrscr();
```

```
    ginput(st, k);
    printf("\nEnter a student # to search:");
    scanf("%d", &no);
    if(lsearch(st, k, no) == -1)
       printf("\n number %d not exist in list ", no);
    else
       printf("\n number %d exist in list.", no);
    getch();
    return 0;
}
//**************************
void ginput(int st[], int len)
{
   int i;
   for(i = 0; i < len; i++) {
     printf("enter student number %d:", i+1);
     scanf("%d",&st[i]);
   }//end of for
}
//**********************
int lsearch(int st[], int len, int no)
{
  int i;
  for(i = 0; i < len; i++)
    if(st[i] == no)
      return i;
  return -1;
}
```

خروجی

```
enter student number 1 : 100
enter student number 2 : 106
enter student number 3 : 108
enter student number 4 : 200
enter student number 5 : 290
Enter a student # to search : 106
number 106 exist in list.
```

جستجوی دودویی

جستجوی دودویی در آرایه مرتب انجام می‌شود. در این روش، عنصر موردنظر با عنصر وسط آرایه مقایسه می‌شود. اگر با این عنصر برابر بود، جستجو خاتمه می‌یابد. اگر عنصر مورد جستجو از عنصر وسط بزرگتر یا کوچکتر بود،

آرایه به دو بخش تقسیم می‌شود: ۱. بخشی که عناصر آن بزرگتر از عنصر مورد جستجو هستند ۲. بخشی که عناصر آن کوچکتر از عنصر مورد جستجو هستند. اگر عنصر مورد جستجو از عنصر وسط بزرگتر بود، جستجو در بخش بالایی آرایه وگرنه جستجو در بخش پایینی آرایه انجام می‌شود. این روند تا یافتن عنصر مورد نظر یا بررسی کل عناصر آرایه ادامه می‌یابد.

مثال ۷-۵

برنامه‌ای که شماره دانشجویانی را از ورودی خوانده، آنها را به‌طور صعودی مرتب می‌کند و با خواندن یک شماره دانشجویی آن را در آرایه جستجو می‌نماید.

توضیح

در این برنامه از سه تابع استفاده شد. تابع ginuput() برای ورود عناصر آرایه، تابع bsearch() برای جستجوی دودویی و تابع bubble() برای مرتب‌سازی آرایه . با متغیرهای توابع ginput() و bubble() در مثال‌های قبلی آشنا شدید.

متغیرهای برنامه

برنامه	متغیر	هدف
main()	k	ثابتی به طول ۵
	st	آرایه‌ای که شماره دانشجویان را نگه می‌دارد
	no	شماره دانشجویی که باید جستجو شود
bsearch()	mid	اندیس عنصر وسط آرایه
	low	اندیس حد پایین آرایه
	high	اندیس حد بالای آرایه
	no	عنصر مورد جستجو
	st	آرایه‌ای از شماره دانشجویی
	len	طول آرایه

```
#include <stdio.h>
#include <conio.h>
void ginput(int [], int);
void bubble(int [], int);
int  bsearch(int [], int, int);
int main()
{
   const int k = 5 ;
   int st[k], no;
   clrscr();
   ginput(st, k);
   printf("\nEnter a student # to search:");
   scanf("%d", &no);
   bubble(st, k);
   if(bsearch(st, k, no) == -1)
      printf("\n number %d not exist in list ", no);
   else
      printf("\n number %d exist in list.", no);
   getch();
```

```
    return 0;
}
 //**************************
void ginput(int st[], int len)
{
   int i;
   for(i = 0; i < len; i++) {
     printf("enter student number %d:", i+1);
     scanf("%d",&st[i]);
   }//end of for
}
 //**********************
int bsearch(int st[], int len, int no)
{
  int mid, low = 0, high = len - 1;
  while(low <= high){
     mid = (low + high) / 2;
     if(no < st[mid])
         high = mid - 1;
     else if(no > st[mid])
         low = mid + 1;
     else return mid;
  }
  return -1;
}
 //**********************
void bubble(int st[], int len)
{
   int i, j, item;
   for(i = len - 1 ; i > 0; i --)
      for(j = 0; j < i ; j++)
         if(st[j] > st[j + 1]) {
            item = st[j] ;
            st[j] = st[j + 1];
            st[j + 1] = item ;
         }//end of if
}
```

خروجی

```
enter student number 1 : 100
enter student number 2 : 180
enter student number 3 : 90
enter student number 4 : 120
enter student number 5 : 80
Enter a student # to search : 180
number 180 exist in list.
```

آرایه‌های چند بعدی

تاکنون مثال‌هایی راجع به آرایه‌های یک بعدی مطرح و مورد بررسی قرار گرفتند. در این بخش شیوه تعریف آرایه‌های بیش از یک بعد را مورد بحث قرار می‌دهیم. حتماً با جدول ضرب اعداد آشنایی دارید. برای پیداکردن عددی در این جدول، باید سطر و ستونی راکه آن عدد در آنجا قرار دارد مشخص کنید. جدول ضرب اعداد، نمونه‌ای از آرایه دو بعدی است. در آرایه‌های دوبعدی، برای دستیابی به عناصر هر آرایه، از دو اندیس استفاده می‌شود. اندیس اول را اندیس سطر و اندیس دوم را اندیس ستون گویند.

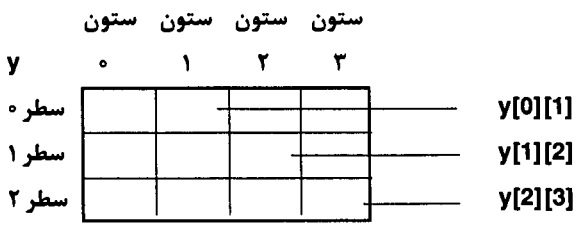
در C، آرایه‌هایی با بیش از دو بُعد (آرایه‌های n بعدی) قابل استفاده‌اند. ولی اغلب برنامه‌نویسان حداکثر از آرایه‌های دوبعدی استفاده می‌نمایند. برای تعریف آرایه دوبعدی در C به صورت زیر عمل می‌شود:

[بُعد ۲] [بُعد ۱] نام آرایه نوع آرایه

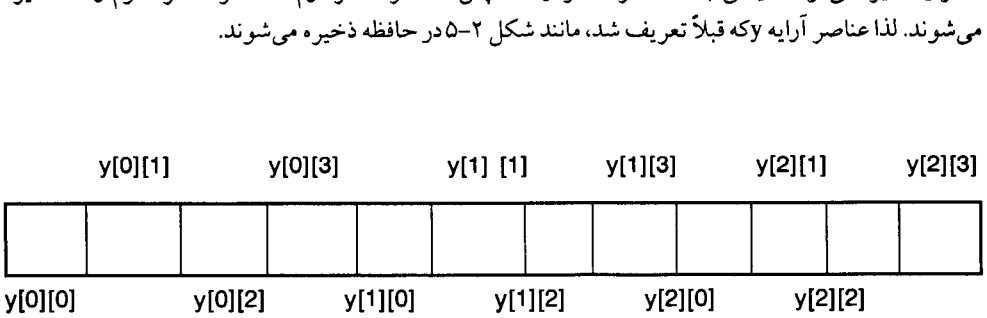
<بُعد ۱> تعداد سطرها و <بُعد ۲> تعداد ستونهای آرایه را مشخص می‌کند. هر کدام از این دو اندیس، از صفر شروع می‌شوند. به عنوان مثال، دستور زیر آرایه دوبعدی ۴×۳را تعریف می‌کند:

```
int     y[3][4] ;
```

این آرایه را می‌توان به صورت زیر نمایش داد:



توجه داشته باشید که آرایه‌ها در حافظه به صورت سطری یا ستونی ذخیره می‌شوند. در زبان C آرایه‌ها به صورت سطری ذخیره می‌گردند. یعنی ابتدا عناصر سطر اول، سپس عناصر سطر دوم، عناصر سطر سوم و ... ذخیره می‌شوند. لذا عناصر آرایه yکه قبلاً تعریف شد، مانند شکل ۲-۵ در حافظه ذخیره می‌شوند.



شکل ۲-۵ نحوه ذخیره آرایه دوبعدی در حافظه.

مثال ۸-۵

برنامه‌ای که جدول ضرب اعداد را تولید کرده در آرایه دوبعدی قرار می‌دهد. سپس عناصر آرایه را طوری به خروجی می‌برد که هر سطر جدول در یک سطر از صفحه‌نمایش چاپ شود.

توضیح

برای تولید جدول ضرب، به دو حلقه نیاز هست. حلقه تکرار بیرونی برای شماره سطرها و حلقه تکرار داخلی برای شماره ستونها مورد استفاده قرار می‌گیرد. از همین اندیسها برای تولید محتویات جدول نیز استفاده شده است.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int table[10][10], i, j ;
    clrscr();
    for(i = 0; i < 10; i++)
       for(j = 0; j < 10; j++)
          table[i][j] = (i + 1)*(j + 1) ;
    for(i = 0; i < 10; i++) {
       for(j = 0; j < 10; j++)
          printf("%4d",table[i][j]) ;
       printf("\n") ;
    }
    getch();
    return 0;
}
```

خروجی

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

آرایه‌های دوبعدی به عنوان آرگومان تابع

برای ارسال آرایه دوبعدی از تابعی به تابع دیگر، باید نام آرایه را به عنوان آرگومان تابع ذکر کرد. برای تعریف پارامتر معادل آن، مانند آرایه‌های یک بعدی عمل می‌شود. با این تفاوت که در حالتی که در آرایه‌های یک بعدی، پارامتر به صورت آرایه بدون طول ذکر می‌شد، در آرایه دوبعدی، طول سطر ذکر نمی‌شود ولی طول ستون حتماً باید ذکر شود (شکل ۳-۵). در این صورت بهتر است، طول سطر، به عنوان آرگومان دیگری به تابع ارسال شود.

مثال ۹-۵

برنامه‌ای که عناصر ماتریس ۲×۳ را از ورودی خوانده، بزرگترین عنصر هر سطر را پیدا می‌کند و به همراه شماره آن سطر به خروجی می‌برد.

توضیح

در این برنامه دو تابع نوشته شده‌اند که عبارت‌انداز: ()minput و ()mcal . در الگوی این توابع، اندیس اول آرایه دوبعدی ذکر نشد ولی اندیس دوم ذکر شده است. تابع ()minput عناصر آرایه را از ورودی می‌خواند و تابع ()mcal بزرگترین عنصر هر سطر را پیدا کرده در خروجی چاپ می‌کند. برای پیدا کردن بزرگترین عنصر هر سطر، اولین عنصر آن سطر به عنوان بزرگترین عنصر انتخاب می‌شود و سپس بقیه عناصر آن سطر یکی یکی با آن مقایسه می‌شوند. هرگاه عنصری در آن سطر، از آن بزرگتر بود، این عنصر به عنوان بزرگترین عنصر آن سطر انتخاب می‌شود. به همین دلیل اندیس حلقه تکرارِ سطر در تابع ()mcall به جای صفر، از یک شروع شده است. در این برنامه متغیر r سطر ماتریس، c ستون ماتریس و mat ماتریس مورد نظر است. متغیرهای i و j شمارنده و rmax بزرگترین عنصر در هر سطر ماتریس است.

```
void      f1 (int x[5][10]);
void      f2 (int x[][10], int);
int       main ()
{
   int       x[5] [10] ;
   ...
   f1 (x) ;
   ....
   f2 (x , 5) ;
   ...
   return 0 ;
}
void      f1 (int x[5] [10])
{
   ...
}
void      f2 (int x[ ] [10] , int row)
{
   ...
}
```

شکل ۳-۵ ارسال آرایه دوبعدی به توابع .

```
#include <stdio.h>
#include <conio.h>
void minput(int [][2], int);
void mcal(int mat[][2], int);
int main()
{
  const int r = 3, c = 2;
  int mat[r][c];
  clrscr();
  minput(mat, r);
  mcal(mat, r);
  getch();
  return 0;
}
//**********************
void minput(int mat[][2], int r)
{
```

```
    int i, j;
    for(i = 0; i < r; i++)
       for(j = 0; j < 2; j++) {
            printf("enter mat[%d][%d]:", i, j);
            scanf("%d", &mat[i][j]);
       }
  }
   //**********************
  void mcal(int mat[][2], int r)
  {
    int i, j, rmax;
    printf(" ROW \t\tMAX");
     printf("\n----------");
    for(i = 0; i < r; i ++) {
       rmax = mat[i][0];
       for(j = 1; j < 2; j++)
           if(mat[i][j] > rmax)
              rmax = mat[i][j];
       printf("\n %d\t\t %d", i+1, rmax);
    }// end of for
  }
```

خروجی

```
enter mat [0] [0] : 3
enter mat [0] [1] : 4
enter mat [1] [0] : 5
enter mat [1] [1] : 6
enter mat [2] [0] : 7
enter mat [2] [1] : 8

   ROW          MAX
----------------------------------
    1           4
    2           6
    3           8
```

مقدار اولیه آرایه‌ها

اگر آرایه‌ای در خارج از تابع ()main تعریف شود (آرایه عمومی)، مقدار اولیه عناصر آن صفر خواهد بود. اگر آرایه در داخل تابع تعریف شود (آرایه محلی)، مقدار اولیه آن تعریف نشده است. هنگام تعریف تابع می‌توان به عناصر آن مقدار اولیه داد. شکل کلی مقدار اولیه دادن به آرایه به صورت زیر است:

```
نوع آرایه   نام آرایه   [بعد ۱]  [بعد ۲]  [...] = {مقادیر} ;
```

به عنوان مثال، دستور زیر، آرایه ۴ عنصری را تعریف کرده، مقادیر ۱، ۳، ۵، ۹را به ترتیب در x[0]، x[1]، x[2]و x[3] قرار می‌دهد.

```
int  x[4] = {1, 3, 5, 9} ;
```

دستورات زیر، آرایه دوبعدی را به دوروش مقدار اولیه می‌دهند:

```
int  y [2] [3] = {1,3,4,7,6,15}
int  m [2] [3] = {{4,6,9) , {7,3,1}};
```

اگر در مقدار اولیه دادن به عناصر آرایه، طول آرایه مشخص نشود، تعداد عناصر آرایه به اندازه تعداد مقادیری است که به آن نسبت داده می‌شود. به عنوان مثال، دستور زیر، آرایه‌ای به طول ۳را تعریف کرده به عناصر آن مقدار می‌دهد.

```
int  p [ ] = {4, 6, 8} ;
```

اگر تعداد مقادیری که به آرایه نسبت داده می‌شود، از طول تعیین شده برای آرایه کمتر باشد، مقادیر بقیه عناصر آرایه صفر می‌شوند. به عنوان مثال، دستور زیر آرایه‌ای صحیح ۱۰ عنصری را تعریف کرده، x[0]را برابر با ۵ و x[1]را برابر با ۱۵ و بقیه مقادیر عناصر آرایه را برابر با صفر قرار می‌دهد:

```
int  x[10] = {5,15} ;
```

در مورد آرایه‌های چندبعدی (دو بعدی و بیشتر)، بهتر است مقادیر هر سطر را در یک جفت آکولاد قرار داد. به نمونه‌ای از مقدار اولیه دادن به آرایه سه بعدی توجه کنید.

```
int m [3] [2] [4] = {{{1,2,3,4} , {5,6,7,8} , } ;
                     {{7,9,3,2} , {4,6,8,3} , } ;
                     {{7,2,6,3} , {0,1,9,4}}} ;
```

مثال ۱۰-۵

برنامه‌ای که دو ماتریس را از ورودی خوانده، حاصلضرب آنها را در ماتریس دیگری قرار می‌دهد.

توضیح

روش ضرب ماتریس‌ها را با توجه به دو ماتریس x و y توضیح می‌دهیم. در ضرب ماتریس‌ها باید تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشند.

$$\underset{x}{\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix}} \times \underset{y}{\begin{bmatrix} 2 & 7 \\ 9 & 6 \end{bmatrix}} = \underset{z}{\begin{bmatrix} 29 & 25 \\ 53 & 58 \end{bmatrix}}$$

$$z[0][0] = x[0][0] \times y[0][0] + x[0][1] \times y[1][0] = ۱\times۲ + ۳\times۹ = ۲۹$$
$$z[0][1] = x[0][0] \times y[0][1] + x[0][1] \times y[1][1] = ۱\times۷ + ۳\times۶ = ۲۵$$
$$z[1][0] = x[1][0] \times y[0][0] + x[1][1] \times y[1][0] = ۴\times۲ + ۵\times۹ = ۵۳$$
$$z[1][1] = x[1][0] \times y[0][1] + x[1][1] \times y[1][1] = ۴\times۷ + ۵\times۶ = ۵۸$$

این برنامه بدون استفاده از توابع نوشته شده است. سه تابع بنویسید که وظایف این برنامه را انجام دهد: تابع minput() برای خواندن ماتریس، تابع mult() برای ضرب ماتریسها و تابع moutput() برای چاپ ماتریسها.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int mat1[2][3], mat2[3][4], mat3[2][4]={0} ;
    int i,j,k,l ;
    clrscr();
    //read mat1
    for(i=0 ; i<2 ; i++)
      for(j=0 ; j<3 ;j++) {
          printf("tenter mat1[%d][%d]: ",i,j);
          scanf("%d",&mat1[i][j]) ;
      }
    //read mat2
    for(i=0 ; i<3 ; i++)
      for(j=0 ; j<4 ;j++) {
          printf("enter mat2[%d][%d]: ",i,j);
          scanf("%d",&mat2[i][j]) ;
      }
    //multiply mat1 by mat2
    for(i=0 ; i<2 ; i++)
       for(j=0 ; j<4 ;j++) {
           mat3[i][j]=0 ;
           for(k=0 ;k<3 ; k++)
           mat3[i][j] = mat3[i][j]+mat1[i][k]*mat2[k][j];
       }
    printf("\n the produc of mat1 & mat2") ;
    printf(" is :\n\n") ;
    for(i=0 ;i<2 ;i++) {
      for(j=0 ; j<4 ;j++)
          printf("%5d", mat3[i][j]) ;
      printf("\n") ;
    }
    getch();
    return 0;
}
```

نکته‌ای راجع به آرایه‌ها

در بعضی از برنامه‌ها ممکن است نیاز به استفاده از آرایه‌ها باشد ولی تعداد عناصری که باید در آرایه قرار گیرد مشخص نباشد. در اینگونه موارد، باید حداکثر تعداد مورد انتظار را در نظر گرفته، آرایه‌ای به همان تعداد از عناصر تعریف نمود. به عنوان مثال، می‌خواهیم برنامه‌ای بنویسیم که از بین حقوق تعدادی از کارکنان، کمترین میزان حقوق را پیدا کند، ولی تعداد دقیق کارکنان را نمی‌دانیم. براساس اطلاعات ناقصی که در اختیار است، تعداد کارکنان ۵۰ نفر اعلام شده است. بنابراین برای تعریف آرایه‌ای جهت نگهداری حقوق کارکنان، یک آرایه ۵۰عنصری تعریف می‌کنیم. روش دیگر انجام این کار، استفاده از تخصیص حافظه پویاست که در فصل ۶ به آن می‌پردازیم.

خروجی

```
enter mat1[0][0]: 1
enter mat1 [0] [1] : 2
enter mat1 [0] [2] : 3
enter mat1 [1] [0] : 5
enter mat1 [1] [1] : 4
enter mat1 [1] [2] : 7
enter mat2 [0] [0] : 8
enter mat2 [0] [1] : 9
enter mat2 [0] [2] : 5
enter mat2 [0] [3] : 8
enter mat2 [1] [0] : 10
enter mat2 [1] [1] : 0
enter mat2 [1] [2] : 12
enter mat2 [1] [3] : 21
enter mat2 [2] [0] : 0
enter mat2 [2] [1] : 5
enter mat2 [2] [2] : 9
enter mat2 [2] [3] : 12
the product of mat1 & mat2 is :
28   24   56   86
80   80   136  208
```

مثال ۱۱-۵

برنامه‌ای که تعدادی مقادیر صحیح را از ورودی خوانده آنها را در آرایه‌ای قرار می‌دهد. سپس توسط تابعی، کوچکترین عنصر آرایه را تعیین می‌کند (حداکثر تعداد اعداد، ۲۰ است). آخرین عدد ورودی، صفر است.

توضیح

چون حداکثر تعداد اعداد ۲۰ است، طول آرایه را ۲۰ در نظر می‌گیریم. در برنامه اصلی، عناصر آرایه را از ورودی می‌خوانیم و آرایه را همراه با تعداد واقعی عناصر آن، به تابعی ارسال می‌کنیم. تابع کوچکترین عنصر و محل وجود آن را پیدا می‌کند و به خروجی می‌برد. تابع findmin() دو پارامتر دارد. پارامتر list مربوط به آرایه و پارامتر size مربوط به تعداد عناصر آرایه است.

```
#include <stdio.h>
#include <conio.h>
int findmin(int [], int);
int main()
{
   int list[20], num, size=0 ;
   clrscr();
   do{
     printf("type list[%d] : ", size) ;
     scanf("%d", &list[size]) ;
   } while(list[size ++] != 0) ;
   size --;
   num = findmin(list, size) ;
```

```
    printf("\n minimum is:%d ",num) ;
    getch();
    return 0;
}
 //***********************
int findmin(int arr[], int size)
{
    int i, min1 ;
    min1 = arr[0] ;
    for(i = 0 ; i < size; i++)
        if(arr[i] < min1)
            min1 = arr[i] ;
    return(min1) ;
}
```

خروجی

```
type list [0] : 10
type list [1] : 5
type list [2] : 18
type list [3] : 0
minimum is : 5
```

رشته‌ها

در زبان C، رشته نوع جدیدی نیست، بلکه به صورت آرایه‌ای از کاراکترها تعریف می‌شود. رشته‌ها برای ذخیره، بازیابی و دستکاری متنها (مثل اسامی افراد)، مورد استفاده قرار می‌گیرند. در C برای تعیین انتهای رشته از کاراکتر خاصی به نام (تهی =NULL) استفاده می‌شود که با '\0' مشخص می‌گردد. بنابراین، آخرین رشته برابر با '\0' می‌باشد. لذا اگر رشته‌ای به نام s را به صورت زیر تعریف کنید، فقط از ۹ کاراکتر می‌توانید استفاده کنید، زیرا کاراکتر آخر '\0' است.

```
char    s[10] ;
```

اگر محتویات این رشته را به طریقی که بعداً گفته می‌شود برابر با "ali" قرار دهید، این رشته به صورت زیر نمایش داده می‌شود:

s[0]	s[1]	s[2]	s[3]	[4]	s[5]	s[6]	s[7]	s[8]	s[9]
a	l	i	\0	?	?	?	?	?	?

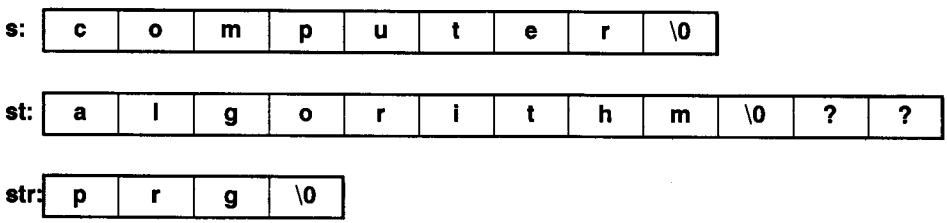
در این نمایش، سه کاراکتر اول برای ذخیره رشته به کار رفت و کاراکتر چهارم برابر '\0' است و بقیه کاراکترها مورد استفاده قرار نگرفته‌اند. به‌این‌ترتیب، دریافتید که طول رشته‌ها را باید یک واحد بیش از آنچه که نیاز دارید تعیین کنید.

مقدار اولیه دادن به رشته‌ها

هنگام تعریف رشته‌ها می‌توان به آنها مقدار اولیه داد. هنگام مقدار اولیه دادن می‌توان طول رشته را مشخص نکرد. در این صورت، اندازه آرایه یک واحد بیش از تعداد کاراکترهایی است که به آن نسبت داده می‌شود. دو روش برای مقدار اولیه دادن به رشته‌ها وجود دارد. **۱.** رشته در داخل کوتیشن قرار گیرد و به متغیر رشته‌ای نسبت داده شود و **۲.** هر یک از کاراکترهای رشته‌ای به عنوان یک عنصر رشته به آرایه نسبت داده شوند. در روش اول، کاراکتر '\0' به طور خودکار در انتهای رشته اضافه می‌شود ولی در روش دوم، '\0' باید توسط برنامه‌نویس در انتهای رشته قرار داده شود. دستورات زیر را در نظر بگیرید:

```
char  s[ ] = "computer"
char  st [12] = "algorithm"
char  str[ ] = {'p' , 'r' , 'g' , '\0'} ;
```

دستور اول، رشتهٔ "computer" را در متغیر رشته‌ای s قرار داده، '\0' را به آن می‌افزاید. **دستور دوم** رشتهٔ "algorithm" را در st قرار می‌دهد و '\0' را به آن اضافه می‌کند. **دستور سوم** سه کاراکتر را به همراه کاراکتر '\0' در str قرار می‌دهد.



ورودی - خروجی رشته‌ها

قبلاً توابع scanf() و printf() را برای ورودی و خروجی داده‌ها دیدید. از همین امکانات می‌توان برای ورودی - خروجی رشته‌ها استفاده کنید. هنگام خواندن رشته توسط scanf()، نام رشته نباید با & همراه باشد. در دستورات زیر، scanf() رشته first را می‌خواند و printf() آن را در صفحه‌نمایش چاپ می‌کند:

```
char first[21];
scanf("%s", first);
printf("\n first is:%s", first);
```

خواندن رشته با تابع gets()

علاوه بر امکانات ذکر شده، با تابع gets() نیز که الگوی آن در فایل stdio.h قرار دارد، می‌توان رشته‌ها را از ورودی خواند. این تابع پس از خواندن رشته، سطر جاری را رد می‌کند. نحوه کاربرد آن به صورت زیر است:

gets (متغیر رشته‌ای) ;

در دستورات زیر، رشته str1 تعریف و سپس از ورودی خوانده شد. پس از واردکردن رشته باید کلید Enter را فشار داد.

```
char  str1[21] ;
gets  (str1) ;
```

تفاوت ()gets و ()scanf در خواندن رشته‌ها

در تابع ()gets فقط کلید Enter انتهای رشته را مشخص می‌کند. لذا رشته می‌تواند حاوی فاصله (space) و یا Tab باشد. در حالی که در تابع ()scanf ، فاصله و Tab نیز به عنوان جداکننده تلقی شده، انتهای رشته را مشخص می‌کند. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
char   s[51] ;
gets   (s) ;
scanf  ("%s" , s) ;
```

اگر در پاسخ به این دستورات، رشتهٔ "computer science" را وارد کنیم، تابع ()gets کل رشته را می‌خواند، در حالی که تابع ()scanf فقط رشتهٔ "computer" را می‌پذیرد. زیرا فضای خالیِ بعد از آن، به عنوان انتهای رشته تلقی می‌شود.

چاپ رشته با تابع ()puts

این تابع که در فایل stdio.h قرار دارد، برای نوشتن رشته‌ها در خروجی مورد استفاده قرار می‌گیرد. این تابع پس از نوشتن رشته، سطر جاری را رد می‌کند. کاربرد آن به صورت زیر است :

```
puts  ("رشته") ;
puts  (متغیر رشته‌ای) ;
```

به عنوان مثال، دستورات زیر را در نظر بگیرید :

```
char  name = "ali" ;
puts  ("your name is :") ;
puts  (name) ;
```

با دستور اول، رشته "ali" در name قرار می‌گیرد. دستور دوم پیامی را در خروجی چاپ می‌کند و دستور سوم متغیر name را در خروجی چاپ می‌کند.

رشته‌ها به عنوان آرگومان تابع

چون رشته‌ها به صورت آرایه‌ای از کاراکترها تعریف می‌شوند، شیوه ارسال رشته‌ها به توابع، همانند آرایه است. یعنی در آرگومان تابع، نام رشته ذکر می‌شود و پارامتر معادل آن می‌تواند آرایه‌ای با طول معین، آرایه‌ای با طول نامعین و یا یک اشاره‌گر باشد.

مثال ۱۲-۵

برنامه‌ای که رشته‌ای را خوانده تمام حروف کوچک آن رشته را به حروف بزرگ تبدیل کرده چاپ می‌کند.

توضیح

در این برنامه، تابعی به نام ()supper نوشته شد که حروف کوچک را به حروف بزرگ تبدیل می‌کند. تفاوت کد اسکی حروف کوچک و بزرگ ۳۲است. یعنی اگر از کد اسکی حروف کوچک ۳۲ واحد کم شود به حروف بزرگ تبدیل می‌شوند. لذا برای تبدیل 'a' به 'A' کافی است از 'a' به اندازه ۳۲ واحد کم کنیم.

```
#include <stdio.h>
#include <conio.h>
void upper(char []);
int main()
{
  char s[21];
  clrscr();
  printf("enter a string:");
  gets(s);
  upper(s);
  puts("result is:");
  puts(s);
  getch();
  return 0;
}
 //************************
void upper(char s[])
{
  int i;
  for(i = 0; s[i]; i++)
      if(s[i] >= 'a' && s[i] <= 'z')
  s[i] -= 32;
}
```

خروجی

```
enter a string : computer science
result is : COMPUTER SCIENCE
```

مثال ۱۳-۵

برنامه‌ای که یک رشته و دو کاراکتر ch1 و ch2 را خوانده، تمام کاراکترهای ch1 را در رشته به ch2 تبدیل می‌کند.

توضیح

در این برنامه، متغیر string برای نگهداری رشته به کار می‌رود، متغیر source_letter کاراکتر موجود و متغیر target_letter کاراکتر جایگزین را مشخص می‌کند. تابع ()replace رشته و دو کاراکتر را از برنامه اصلی گرفته، عمل جایگزینی را انجام می‌دهد.

```
#include <stdio.h>
#include <conio.h>
void replace(char [], char, char);
int main()
{
    char string[50] ;
    char source_letter, target_letter ;
    int i ;
    clrscr();
    printf("\nenter the string: ") ;
    gets(string) ;
    printf("enter source character :") ;
    source_letter = getche() ;
    printf("\nenter target character :") ;
    target_letter = getche() ;
    replace(string, source_letter, target_letter);
    printf("\nthe result string is:");
    puts(string) ;
    getch();
    return 0;
}
//************************
void replace(char string[], char source_letter, char target_letter)
{
    int i;
    if(source_letter != target_letter)
      for(i = 0 ; string[i] ; i++)
          if(string[i] == source_letter)
             string[i] = target_letter;
}
```

خروجی

```
enter the string : in the name of allah
enter source character : a
enter target character : A
The result string is : in the nAme of AllAh
```

مثال ۱۴-۵

برنامه‌ای که دو رشته را از ورودی خوانده محتویات آن دو رشته را با یکدیگر عوض می‌کند.

توضیح

رشته‌های s1 و s2 کاراکتر به کاراکتر به کمک متغیر کمکی temp جابجا می‌شوند. اگر طول رشته‌ها متفاوت باشد، کاراکترهای باقیمانده از رشته طولانی‌تر، به رشته دیگر منتقل می‌شود. سپس انتهای هر رشته برابر با '\0' قرار می‌گیرد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      char s1[81], s2[81], temp;
      int i, j;
      clrscr();
      printf("enter string <s1> : ") ;
      gets(s1) ;
      printf("enter string <s2> : ") ;
      gets(s2) ;
      for(i = 0; s1[i] && s2[i]; i++) {
          temp = s1[i] ;
          s1[i] = s2[i] ;
          s2[i] = temp ;
      } //end of for
      if(s1[i]) {    //s1 has many char          طول s1 بیشتر است
        j = i ;
        while(s1[i])
        s2[i] = s1[i++] ;
        s2[i]='\0' ;
        s1[j]='\0' ;
      }//end of if
      else if (s2[i]) {   //s2 has many char      طول s2 بیشتر است
          j = i ;
          while(s2[i])
          s1[i] = s2[i++] ;
          s2[j]='\0' ;
          s1[i]='\0' ;
      } //end of else if
      printf("new content of s1  is:") ;
      puts(s1) ;
      printf("new content of s2  is:") ;
      puts(s2) ;
      getch();
      return 0;
}
```

خروجی

```
enter string <s1> : computer
enter string <s2> : science
new content of s1 is : science
new content of size : computer
```

مثال ۱۵-۵

برنامه‌ای که رشتهٔ s، کاراکتر ch و عدد n را از ورودی خوانده به تابعی ارسال می‌کند. این تابع، محل nامین وقوع ch را در رشته پیدا می‌کند و برمی‌گرداند. اگر nامین وقوع کاراکتر پیدا نشود، تابع مقدار ۱- را برمی‌گرداند.

توضیح

به عنوان مثال، فرض کنید می‌خواهیم محل دومین وقوع کاراکتر 'o' را در رشته "second content of p" پیدا کنیم. تابع باید مقدار ۹ (با شروع از یک) را برگرداند. مقداری که توسط تابع eventf() برگردانده می‌شود در برنامه اصلی در p قرار می‌گیرد.

```
#include <stdio.h>
#include <conio.h>
int eventf(char s[], char ch, char n);
int main()
{
   char s[21], ch;
   int n, p;
   clrscr();
   printf("enter string:");
   gets(s);
   printf("enter the character:");
   ch = getche();
   printf("\nenter the event:");
   scanf("%d", &n);
   p = eventf(s, ch, n);
   if(p == -1)
      printf("\n %dth event not found:", n);
   else
      printf("\n %dth event is in position:%d", n, p);
   getch();
   return 0;
}
//*****************
int eventf(char s[], char ch, char n)
{
   int i, count = 0;
   for(i = 0; s[i]; i ++){
      if(s[i] == ch)
         count ++;
      if(count == n)
   return(i + 1);
    }
    return -1;
}
```

خروجی

```
enter string : computer science
enter the character : e
enter the event : 2
2th event is in position : 13
```

مثال ۱۶-۵

برنامه‌ای که حروف 'a' تا 'z' را به عنوان مقدار اولیه رشته‌ای تعیین کرده، آن را به طور معکوس در خروجی چاپ می‌کند.

متغیرهای برنامه

متغیر	هدف
t	رشته‌ای که باید معکوس شود
temp	متغیر کمکی برای جابجایی کاراکتر
i	شمارنده حلقه
length	طول رشته

```
#include <stdio.h>
#include <conio.h>
int main()
 {
  char t[]="abcdefghijklmnopqrstuvwxyz";
  char temp ;
  int i, length;
  clrscr();
  for(length = 0 ;t[length]; ++length) ;
    length -- ;
  puts(" the initial text is : ") ;
  puts(t) ;
  for(i = 0; i < length / 2; i++) {
    temp = t[i] ;
    t[i] = t[length - i] ;
    t[length - i] = temp ;
  }
  puts(" the reversed test is : ") ;
  puts(t) ;
  getch();
  return 0;
}
```

خروجی

```
the initial text is :
a b c d e f g h i j k l m n o p q r s t u v w x y z
the reversed text is :
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

مثال ۱۷-۵

برنامه‌ای که یک عبارت محاسباتی را که حاوی عملگرهای یک کاراکتری و عملوندهای یک‌رقمی است از ورودی خوانده، عملگرها را در یک آرایه و عملوندها را در آرایه دیگری قرار داده، در خروجی چاپ می‌کند.

توضیح

وقتی ارقام وارد رشته می‌شوند، به طور کاراکتری ذخیره خواهند شد. بنابراین، ارقام صفر تا ۹ به صورت کاراکترهای '0' تا '9' ذخیره می‌شوند. پس از تشخیص کاراکترهای '0' تا '9' باید آنها را به ارقام صفر تا ۹ تبدیل کرد و در آرایه‌ای قرار داد. برای این منظور کافی است، از این کاراکترها ۴۸ واحد کم شود. به عبارت دیگر برای تبدیل کاراکتر '0' به رقم صفر، از '0' به اندازه ۴۸ واحد کم می‌شود و به همین ترتیب برای تبدیل کاراکتر '8' به رقم 8 نیز باید از '8' به اندازه ۴۸ واحد کم شود. تابع separate() این کار را انجام می‌دهد. expr حاوی عبارت، oper حاوی عملگرها و opnd حاوی عملوندهاست. متغیرهای i، j و k شمارنده‌اند.

```
#include <stdio.h>
#include <conio.h>
void separate(char exp[], char oper[], int opnd[]);
int main() {
   char expr[21], oper[21];
   int opnd[21];
   clrscr();
   printf("enter expression:");
   gets(expr);
   separate(expr, oper, opnd);
   getch();
   return 0;
}
//***************
void separate(char expr[], char oper[], int opnd[])
{
  int i, j = 0 , k = 0;
  for(i = 0; expr[i]; i ++)
     if(expr[i] >= '0' && expr[i] <= '9')
        opnd[j ++] = expr[i] - 48;
     else
        oper[k ++] = expr[i];
  oper[k] = '\0';
  puts("operators are:");
  puts(oper);
  puts("operands are:");
  for(i = 0; i < j; i ++)
    printf("%2d", opnd[i]);
}
```

خروجی

```
enter expression : 6 / 8 - 9 + 4 * 5
operators are :
/ - + *
operands are :
6 8 9 4 5
```

انتساب رشته‌ها (کپی کردن رشته در رشته دیگر)

اگر متغیری به نام x و با مقدار ۵ داشته باشید، دستور ; x = y موجب می‌شود تا هر آنچه در x وجود دارد در y قرار گیرد (کپی شود). اگر رشته‌ای به نام s1 با محتویات "computer" داشته باشیم و بخواهیم آن را در رشته دیگری به نام s2 کپی کنیم، با دستور s2 = s1 امکان‌پذیر نیست. به عبارت دیگر با دستور انتساب نمی‌توان رشته‌ای را در رشتهٔ دیگر کپی کرد. لذا در C دستور زیر نادرست است:

s = "all" ; (نادرست)

برای کپی کردن رشته‌ای در رشته دیگر و یا انتساب رشته‌ای به رشته دیگر، از تابع ()strcpy استفاده می‌شود. این تابع در فایل string.h قرار دارد و به صورت زیر به کار می‌رود:

strcpy (str1 , str2) ;

با اجرای این دستور، آنچه که در str2 است در str1 کپی می‌شود. دستورات زیر را در نظر بگیرید:

```
strcpy (st, "computer") ;
strcpy (s, st) ;
```

دستور اول رشته "computer" را در st قرار می‌دهد و دستور دوم رشته st را در s کپی می‌کند. لذا، رشته s نیز برابر با "computer" خواهد شد.

در کاربرد تابع ()strcpy چنانچه طول str2 بیش از str1 باشد، تا آنجایی که عمل کپی امکان‌پذیر باشد، عمل کپی در آن رشته صورت می‌گیرد و کاراکترهای اضافی بلافاصله پس از str1 قرار می‌گیرند. در نتیجه، اگر متغیرهایی بعد از str1 قرار داشته باشند، محتویات آنها از بین خواهند رفت. به عنوان مثال، دستورات زیر را در نظر بگیرید. با این دستورات سه کاراکتر از رشته موردنظر در s قرار می‌گیرد و بقیه کاراکترها نیز در حافظه بعد از s واقع می‌شوند.

```
char  s[3] ;
strcpy (s, " computer science") ;
```

مثال ۱۸-۵

برنامه‌ای که رشته‌ای به نام s و عددی مثل n را از ورودی خوانده، کاراکتر موجود در محل n را از رشته حذف می‌کند.

توضیح

برای این کار از تابع ()strcpy استفاده شده است. به‌این‌ترتیب که بخشی از رشته با شروع از محل position+1 در محل position رشته کپی شده است.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main() {
   char string[81] ;
   int position ;
   clrscr();
   printf("type a string :") ;
   gets(string) ;
```

```
    printf("enter position for ") ;
    printf("delete character :") ;
    scanf("%d", &position) ;
     strcpy(&string[position],&string[position+1]) ;
    printf("the result string is: ") ;
    puts(string) ;
    getch();
    return 0;
}
```

خروجی

```
type a string : computer science
enter position for delete character : 8
the result string is : computerscience
```

مقایسه رشته‌ها

اگر بخواهید دو متغیر x و y را با هم مقایسه کنید، از دستور if (x == y) استفاده می‌نمایید. اما اگر s1 و s2 دو رشته باشند، برای مقایسه آنها نمی‌توانید از دستور if (s1 == s2) استفاده کنید. برای مقایسه رشته‌ها از تابع strcmp() که در فایل string.h قرار دارد به صورت زیر استفاده می‌شود:

```
strcmp (s1,s2)
```

حاصل کار این تابع یک عدد است که مقدار آن بیانگر وضعیت دو رشته نسبت به هم است. اگر عدد برگردانده شده توسط تابع برابر با صفر باشد، این دو رشته با هم مساویند. اگر این عدد منفی باشد s1 < s2 و اگر این عدد مثبت باشد s1 > s2 است.

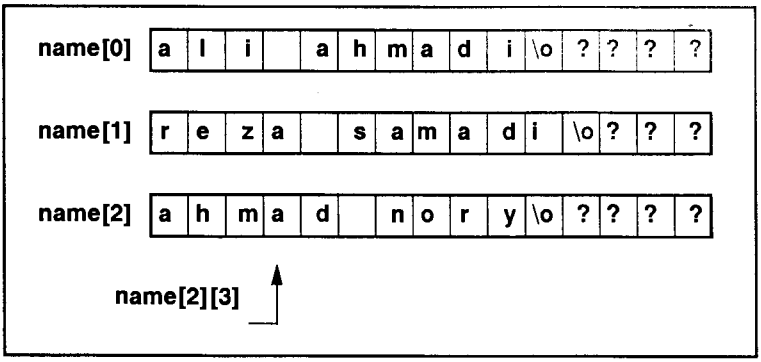
منظور از مقایسه رشته‌ها، مقایسه کاراکتری آنهاست. اولین کاراکتر از رشته اول با اولین کاراکتر از رشته دوم مقایسه می‌شود. اگر مساوی باشند، کاراکترهای بعدی با هم مقایسه می‌شوند. اگر تمام کاراکترهای دو رشته با هم مساوی باشند، آن دو رشته با هم مساویند. با رسیدن به اولین مورد اختلاف، کاراکتری که بزرگتر است، رشتهٔ حاوی آن کاراکتر، بزرگتر خواهد بود. پس از مطالعه آرایه‌ای از رشته‌ها، مثالی را در این خصوص خواهید دید.

الحاق دو رشته

با استفاده از تابع strcat() می‌توان دو رشته را با هم الحاق کرد. الحاق کردن دو رشته به معنی قراردادن یک رشته در انتهای رشته دیگر است. به عنوان مثال، اگر s1 برابر با "computer" و s2 برابر با "science " باشد، الحاق s2 به انتهای s1 موجب می‌شود تا s1 حاوی رشته "computer science" شود. تابع strcat() در فایل string.h قرار دارد و به صورت زیر به کار می‌رود.

```
strcat (s1, s2) ;
```

با این دستور، s2 در انتهای s1 قرار می‌گیرد. چنانچه طول رشته s1 طوری باشد که گنجایش s2 را نداشته باشد، بقیه رشته s2 در ادامه رشته s1 قرار می‌گیرد و در نتیجه چنانچه متغیرهایی بعد از s1 وجود داشته باشند، محتویات آنها از بین می‌رود.



شکل ۶-۵
آرایه‌ای از رشته‌ها.

آرایه‌ای از رشته‌ها

با اطلاعاتی که تاکنون کسب کردید، می‌توانید رشته‌هایی را تعریف کرده آنها را پردازش کنید. اما اگر بخواهید رشته‌ای تعریف کنید که مثلاً نام ۱۰ دانشجو را نگهداری کند، چه کاری انجام می‌دهید؟ در اینگونه موارد، باید آرایه‌ای از رشته‌ها تعریف کنید. به عنوان مثال، اگر بخواهید رشته‌ای را برای نگهداری نام ۳ دانشجو که طول هر نام ۱۵ کاراکتر است تعریف کنید، به صورت زیر عمل کنید:

```
char  name [3] [15] ;
```

همان‌طور که می‌دانید، name یک آرایه دوبعدی است که بُعد اول آن تعداد رشته‌های موردنظر و بُعد دوم آن طول هر رشته است (آرایه‌ای از رشته‌ها یعنی آرایه‌ای از آرایه‌ها که همان آرایه دو بعدی است). اینگونه تعریف کردن، مثل این است که لیستی از نام سه دانشجو را در صفحه کاغذ به طور زیرهم بنویسید. برای ذخیره رشته‌ها یک آرایه دوبعدی تعریف می‌کنیم، ولی برای بازیابی هر یک از رشته‌ها از یک اندیس استفاده می‌کنیم. همانطور که در شکل ۶-۵ می‌بینید، name[0] نام اولین دانشجو، name[1] نام دومین دانشجو و name[2] نام سومین دانشجو است. اگر با دو اندیس به این رشته مراجعه شود، اندیس دوم، شماره کاراکتری از رشته را مشخص می‌کند (مانند name[2][3] در شکل ۶-۵).

مثال ۱۹-۵

برنامه‌ای که اسامی تعدادی از دانشجویان را از ورودی خوانده، در آرایه‌ای قرار می‌دهد و اسامی را به روش حبابی به طور صعودی مرتب می‌کند. سپس نامی را از ورودی خوانده، آن را به روش دودویی در آرایه جستجو می‌کند و پیام مناسبی صادر می‌نماید.

توضیح

در این برنامه از دو تابع استفاده شده است. تابع bubble() اسامی دانشجویان را به روش حبابی به طور صعودی مرتب می‌کند. آرگومان اول آن نام آرایه، و آرگومان دوم آن، تعداد دانشجویان است. تابع bsearch نامی را به روش دودویی در آرایه مرتب شده جستجو می‌کند. اگر نام در آرایه وجود نداشته باشد، مقدار ۱- را برمی‌گرداند. آرگومان اول نام آرایه، آرگومان دوم نام مورد جستجو و آرگومان سوم، تعداد دانشجویان را مشخص می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void bubble(char [][21], int);
int bsearch(char [][21], char [], int);
int main()
{
    const int n = 5;
    int i;
    char name[21], arr [n][21];
    clrscr();
    for(i = 0; i < n; i ++){
       printf("\n enter name %d :", i + 1) ;
       gets(arr[i]);
    }
    bubble(arr, n);
    printf("\n enter one name for search :") ;
    gets(name) ;
    if(bsearch(arr, name, n) == -1)
       printf("\n name <%s> not exist in table.", name) ;
    else
       printf("\n name <%s> exist in table.", name) ;
     getch();
     return 0;
}
//*********************
void bubble(char arr[5][21], int n)
{
  int i, j;
  char temp[21];
  for(i = n - 1; i > 0; i --)
     for(j = 0; j < i; j++)
       if(strcmp(arr[j], arr[j + 1]) > 0){
          strcpy(temp, arr[j]);
          strcpy(arr[j], arr[j + 1]);
          strcpy(arr[j + 1], temp);
       }
}
//*******************
int bsearch(char arr[5][21], char name[21], int n)
{
  int mid, low = 0, high = n - 1;
  while(low <= high){
```

```
        mid = (low + high) / 2;
        if(strcmp(name, arr[mid]) < 0)
          high = mid - 1;
        else if(strcmp(name, arr[mid]) > 0)
        low = mid + 1;
        else return mid;
      }
      return -1;
    }
```

خروجی

```
enter name 1 : mohammad
enter name 2 : ali
enter name 3 : zahra
enter name 4 : ahmad
enter name 5 : sara
enter one name for search : ahmad
name <ahmad> exist in table.
```

تمرینات

۱. برنامه‌ای بنویسید که عدد صحیح n را از ورودی خوانده، تمام اعداد اول قبل از آن عدد را با استفاده تعریف زیر تعیین کرده، به خروجی ببرد:

عددی اول است که بر هیچکدام از اعداد اول قبل از خودش قابل قسمت نباشد.

۲. برنامه‌ای بنویسید که شماره دانشجویی تعدادی از دانشجویان را از ورودی خوانده، در آرایه‌ای قرار دهد و سپس آرایه را به روش انتخابی به طور صعودی مرتب کند. مرتب‌سازی به روش انتخابی به این صورت انجام می‌شود: کوچکترین عنصر آرایه پیدا شده، جای آن با عنصر اول آرایه عوض می‌شود. در مرحله بعد، بقیه عناصر آرایه برای یافتن کوچکترین عنصر آرایه جستجو می‌شود و جای آن با عنصر دوم آرایه عوض می‌شود. این روند تا مرتب‌سازی کامل آرایه ادامه می‌یابد. پس از مرتب‌سازی، نتیجه را در خروجی چاپ کنید. سه تابع به کار ببرید، تابعی برای خواندن عناصر آرایه، تابعی برای مرتب‌سازی و تابعی برای چاپ عناصر آرایه.

۳. برنامه‌ای بنویسید که عناصر دو آرایه ۵ عنصری را که همگی از نوع صحیح هستند از ورودی خوانده، آنها را به روش حبابی مرتب کند و سپس این دو آرایه را به طور مرتب در هم ادغام نماید. تابعی برای اخذ ورودی، تابعی برای مرتب‌سازی، تابعی برای ادغام و تابعی برای چاپ آرایه‌ها بنویسید.

۴. برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده، تمام کاراکترهای تکراری را از رشته حذف کند. رشته در برنامه اصلی خوانده شود و حذف کاراکترهای تکراری در تابع انجام شود. تابع دو آرگومان دارد: رشته و طول رشته.

۵. برنامه‌ای بنویسید که یک عدد ۲۰ رقمی را با عدد ۲۰ رقمی صحیح دیگر جمع کند. برای این منظور، هر یک از دو عدد را به صورت رشته‌ای از ورودی بخوانید. سپس هر کاراکتر را به رقم معادل آن تبدیل کرده، در یک عنصر آرایه قرار دهد و هر عدد را به همین روش در آرایه ذخیره نماید. (هر عدد در یک آرایه). سپس حاصل جمع این دو عدد ۲۰ رقمی را محاسبه کرده، در خروجی چاپ کنید. تابعی برای خواندن رشته‌ها، تابعی برای تبدیل کاراکتر به رقم، تابعی برای انجام عمل جمع و تابعی برای چاپ عدد بنویسید.

۶. برنامه‌ای بنویسید که تعدادی عدد را از ورودی خوانده، آنها را به طور مرتب در آرایه‌ای قرار دهد (دقت داشته باشید که اعداد در موقع قرارگرفتن در آرایه، به طور صعودی مرتب باشند)، سپس آرایه مرتب را به خروجی ببرد.

۷. برنامه‌ای بنویسید که رشته‌ای را که به نقطه ختم می‌شود از ورودی خوانده، کاراکترهای موجود در رشته را به همراه تعداد دفعات تکرار آنها به خروجی ببرد.

۸. برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده، مشخص کند که آیا رشته از هر دو طرف که در نظر گرفته شود یکسان است یا خیر. مثلاً رشته "beeb" چنین خاصیتی دارد.

۹. برنامه‌ای بنویسید که دو رشته s1 و s2 را از ورودی خوانده، رشته s1 را در رشتهٔ s2 جستجو کند. خواندن رشته‌ها توسط تابع اصلی و جستجوی رشته توسط تابعی انجام شود.

۱۰. برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده، تمام کلمات چهارحرفی آن را با کلمه "love" جایگزین کند. مثلاً رشته "I hate you, you doer" باید به رشته "I love you, you love" تبدیل شود.

۱۱. برنامه‌ای بنویسید که با خواندن تعدادی عدد از ورودی، آنها را در آرایه‌ای قرار می‌دهد. سپس کلیه عناصر آرایه را بر عنصر وسط تقسیم کند. اگر عنصر وسط صفر باشد، بر عنصر بعد از عنصر وسط تقسیم کند. چنانچه این عنصر صفر باشد، بر عنصر قبل از عنصر وسط تقسیم کند. چنانچه این عنصر صفر باشد، بر عنصر بعد از عنصر وسط تقسیم کند. اگر این عنصر صفر باشد، برای پیداکردن عنصری غیر صفر و انجام عمل تقسیم، به روند قبلی ادامه دهد. اگر همهٔ عناصر آرایه صفر باشند، پیام مناسبی صادر کند.

۱۲. برنامه‌ای بنویسید که رشته‌ای را خوانده موارد زیر رامشخص کند:

الف)تعداد حروف کوچک　　ب)تعداد حروف بزرگ

ج)تعداد حروف صدا دار　　د)تعداد ارقام موجود در رشته

۱۳. برنامه‌ای بنویسید که رشته‌ای را خوانده مجموع ارقام موجود در رشته را محاسبه کند و چاپ کند.

اشاره‌گرها

آدرس هر متغیر را در حافظه، **اشاره‌گر** گویند. منظور از آدرس حافظهٔ متغیر چیست؟ می‌دانید که بایت، یکی از تقسیمات حافظه است. به عبارت دیگر، حافظه کامپیوتر، مجموعه‌ای از چندین بایت است. هر بایت دارای یک شماره ردیف است. شماره ردیف هر بایت از حافظه را آدرس آن محل از حافظه گویند. ضمناً می‌دانید که متغیرها نامی برای محلهای حافظه‌اند و لذا بایت‌هایی از حافظه را اشغال می‌کنند. آدرس اولین بایتی از حافظه که به متغیر اختصاص می‌یابد، **آدرس** آن متغیر نام دارد.

اشاره‌گرها در C کاربرد فراوانی دارند، به طوری که اغلب قابلیتهای C به نقش اشاره‌گر در این زبان برمی‌گردد. استفاده از اشاره‌گرها در C، قابلیتهای زیر را فراهم می‌کند:

۱. تخصیص حافظه پویا. در این نوع تخصیص حافظه، برنامه در زمان اجرا از سیستم حافظه می‌گیرد و در صورت عدم نیاز، آن حافظه را به سیستم برمی‌گرداند.

۲. موجب بهبود کارایی بسیاری از توابع می‌شود (توابع را با استفاده از آدرس آنها می‌توان فراخوانی کرد).

۳. کار با رشته‌ها و آرایه‌ها را آسان می‌کند.

۴. ارسال آرگومان‌ها از طریق فراخوانی با ارجاع را امکان‌پذیر می‌سازد.

متغیرهای اشاره‌گر

اشاره‌گر می‌تواند در متغیری ذخیره شود. اما، گرچه اشاره‌گر یک آدرس حافظه است و آدرس حافظه نیز یک عدد است، ولی نمی‌توان آن را در متغیرهایی از نوع int، double و یا غیره ذخیره کرد. بلکه متغیری که می‌خواهد اشاره‌گر را ذخیره کند باید از نوع اشاره‌گر باشد. این متغیرها را **متغیرهای اشاره‌گر** گویند. برای تعریف متغیرهای اشاره‌گر در C، به صورت زیر عمل می‌شود:

نوع * متغیر :

به‌این‌ترتیب، برای تعریف متغیر اشاره‌گری که بخواهد آدرس متغیرهایی را نگهداری کند، باید نوع متغیر اشاره‌گر را همنوع با آن متغیرها در نظر گرفت و کنار متغیر اشاره‌گر، علامت * را قرار داد. به عنوان مثال دستور زیر را در نظر بگیرید:

```
int   * p ;
```

این دستور را می‌توان به صورتهای زیر تفسیر کرد:

۱. p متغیر اشاره‌گری از نوع int است.

۲. p آدرس محلهایی از حافظه را که محتویات آنها مقادیری از نوع صحیح‌اند نگهداری می‌کند.

۳. p می‌تواند به محلهایی اشاره کند که محتویات آنها مقادیری از نوع صحیح می‌باشند.

اکنون دستورات زیر را در نظر بگیرید :

```
int     *p1, *p2, v1, v2 ;
double  *f1, f2 ;
char    *ch ;
```

در دستور اول، متغیرهای p1 و p2 متغیرهای اشاره‌گر از نوع صحیح و متغیرهای v1 و v2 متغیرهای معمولی (غیر اشاره‌گر) تعریف می‌شوند. در دستور دوم، f1 متغیر اشاره‌گر و f2 متغیری از نوع double تعریف می‌شوند و در دستور سوم، ch متغیر اشاره‌گری از نوع کاراکتری تعریف می‌شود.

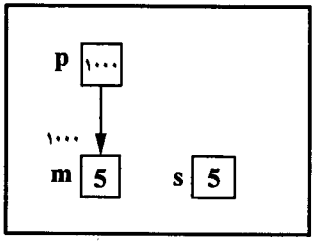
تذکر : در این کتاب، برای سهولت، به جای اینکه بگوییم متغیر اشاره‌گر، به اختصار می‌گوییم اشاره‌گر. یعنی به جای اینکه بگوییم متغیری مثل p، متغیر اشاره‌گر است، می‌گوییم p یک اشاره‌گر است.

عملگرهای اشاره‌گر

دو عملگر & و * در انجام عملیات با اشاره‌گرها مورد استفاده قرار می‌گیرند. هر یک از این دو عملگر، یک عملوند دارند. عملگر & آدرس عملوند خودش را مشخص می‌کند و عملگر * محتویات جایی را مشخص می‌نماید که عملوندش به آن اشاره می‌کند. دستورات زیر را در نظر بگیرید:

```
int   * p, m, s ;
m = 5 ;
p = &m ;
s = * p ;
```

در دستور اول، p یک اشاره‌گر و m و s متغیرهایی از نوع صحیح تعریف می‌شوند. دستور دوم، مقدار ۵ را در متغیر m قرار می‌دهد. دستور سوم، آدرس متغیر m را در p قرار می‌دهد، یعنی p به m اشاره می‌کند. دستور چهارم، محتویات جایی را که آدرس آن‌جا در p است، در s قرار می‌دهد (شکل ۱-۶). در این شکل فرض شد که متغیر m در محل ۱۰۰۰ حافظه قرار دارد. علامت فلش از p به m بیانگر این است که p به m اشاره می‌کند.



شکل ۱-۶ اشاره‌گر و حافظه .

اشاره‌گرها و انواع متغیرها

اکنون که تا حدودی با مفهوم اشاره‌گرها آشنا شدید، دریافتید که اشاره‌گرها در C دارای نوع‌اند. یعنی وقتی اشاره‌گری از نوع int تعریف می‌شود، باید به متغیرهایی از نوع int اشاره نماید و اشاره‌گر از نوع double باید به متغیرهایی از همین نوع اشاره نماید. اگر نوع متغیری که آدرس آن در یک اشاره‌گر قرار می‌گیرد، با نوع اشاره‌گر یکسان نباشد، کامپایلر زبان C خطایی را اعلام نمی‌کند، ولی برنامه‌نویس باید بداند که این امر یقیناً در صحت اجرای برنامه اثر می‌گذارد و به نتایج حاصل از برنامه اطمینانی نیست. برنامه زیر را در نظر بگیرید :

```
#include   <stdio.h>
int   main()
{
      float    x, y ;
      int      *p ;
      x = 5.95 ;
      p = &x ;
      y = *p ;
      printf ("\n y = % f " , y) ;
}
```

در این برنامه، x و y از نوع float و اشاره‌گر p از نوع صحیح تعریف شده‌اند. مقدار x برابر با ۵/۹۵ می‌شود و سپس آدرس آن در p قرار می‌گیرد. انتظار داریم که با دستور y = * p آنچه که در x است، در y قرار گیرد. ولی این کار به درستی انجام نمی‌شود و در نتیجه دستور printf()، چیزی غیر از ۵/۹۵ را چاپ خواهد کرد. این نکته، در C بسیار مهم است و باید آن را به خاطر بسپارید. به نظر شما، در برنامه قبلی، چرا با اینکه با دستور p=&x آدرس x باید در p قرار گیرد، و با دستور y=*p نیز محتویات آن آدرس در y قرار گیرد، این کار به درستی انجام نمی‌شود؟ برای پاسخ به این سئوال، دستورات زیر را در نظر بگیرید:

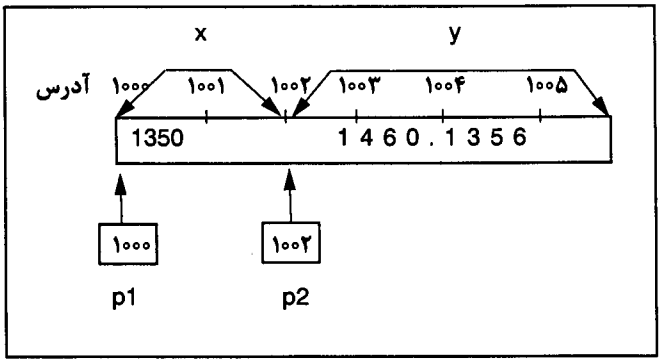
```
int  x, *p1, *p2;
float  y ;
x = 1350 ;
y = 1460.1356
p1 = &x ;
p2 = &y ;
```

با فرض اینکه، x در محل ۱۰۰۰ حافظه قرار دارد، محتویات x و y را به همراه حافظه آنها، می‌توان مانند شکل ۲-۶ در نظر گرفت. در این شکل، بایت‌های ۱۰۰۱ و ۱۰۰۰ در اختیار x و بایت‌های ۱۰۰۲ تا ۱۰۰۵ در اختیار y قرار می‌گیرد. دستور p1=&x موجب می‌شود تا ۱۰۰۰ در p1 و دستور p2=&y موجب می‌شود تا ۱۰۰۲ در p2 قرار گیرد. اکنون اگر دستوری مثل m=*p1 را اجرا کنیم، چون p1 اشاره‌گر صحیح است و متغیر صحیح ۲ بایت (فرض کنید اینطور باشد) را اشغال می‌کند، محتویات دو بایت از حافظه، با شروع از آدرسی که در p1 است (۱۰۰۰) در m قرار می‌گیرد. این مقدار، همان مقدار x است. اما اگر دستوری مثل m=*p2 را اجرا کنیم، چون p2 از نوع صحیح است، محتویات دو بایت از حافظه، با شروع از آدرس ۱۰۰۲ در m قرار می‌گیرد. این مقدار همان مقداری نیست که در y وجود دارد. بنابراین، نتیجه درستی حاصل نخواهد شد.

اعمال روی اشاره‌گرها

اعمالی که بر روی اشاره‌گرها می‌توان انجام داد، به گستردگی اعمالی نیست که بر روی سایر متغیرها انجام می‌شود. اعمالی که بر روی اشاره‌گرها انجام می‌شوند عبارت‌انداز:

۱. عمل انتساب اشاره‌گرها به یکدیگر
۲. اعمال محاسباتی جمع و تفریق
۳. عمل مقایسه رشته‌ها



شکل ۲-۶

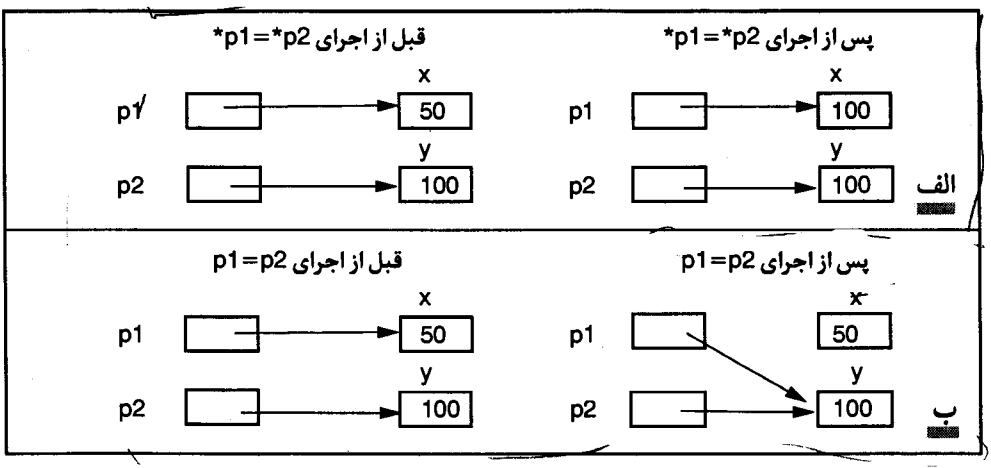
نوع اشاره گرها

انتساب اشاره‌گرها به یکدیگر

می‌دانید که اگر x و y دو متغیر باشند، دستور x=y آنچه را که در y قرار دارد در x قرار می‌دهد. این عمل را انتساب y به x گویند. دو اشاره‌گر را نیز می‌توان به یکدیگر نسبت داد. در این صورت هر دو اشاره‌گر به یک محل از حافظه اشاره خواهند کرد. دستورات زیر را در نظر بگیرید :

```
int   *p1 , *p2 , x , y ;
x = 50 ;
y = 100 ;
p1 = &x ;
p2 = &y ;
* p1 = *p2 ;
```

برای آشنایی با نحوه اجرای این دستورات، شکل ۳-۶الف را ببینید. توجه داشته باشید که منظور از دستور p2*=p1* این است که محتویات جایی که p2 به آن اشاره می‌کند، در جایی که p1 به آن اشاره می‌کند قرار گیرد. اگر در دستورات قبلی، به جای دستور ;p2* = p1* دستور ; p1 = p2 را قرار دهید، شکل ۳-۶ب حاصل می‌شود. معنای دستور ; p1 = p2 این است که p1 به جایی اشاره می‌کند که p2 به آنجا اشاره می‌کند.



شکل ۳-۶ انتساب اشاره گرهابه یکدیگر.

اعمال محاسباتی بر روی اشاره‌گرها

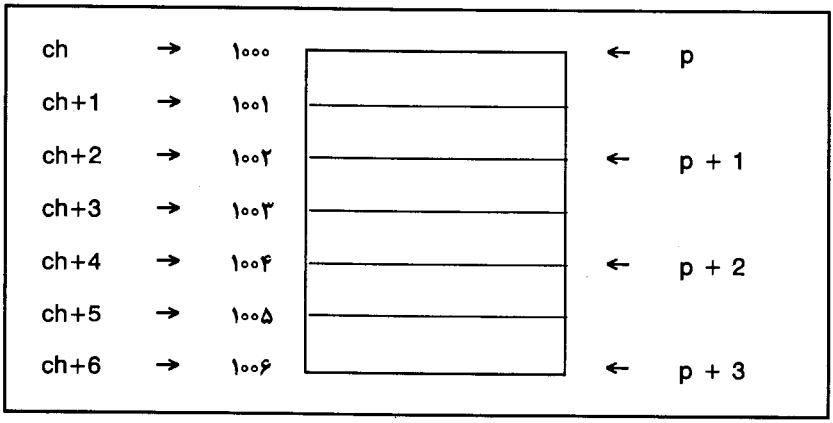
اعمال جمع و تفریق را می‌توان بر روی اشاره‌گرها انجام داد. با افزایش یک واحد به اشاره‌گر، به اندازه طول نوع اشاره‌گر، به آن اضافه می‌شود. به عنوان مثال، اگر p اشاره‌گری از نوع int باشد که به محل ۱۰۰۰ حافظه اشاره می‌نماید، ++p موجب می‌شود تا p به عدد صحیح بعدی اشاره نماید. با فرض اینکه طول نوع صحیح ۲ بایت باشد، آنگاه p به محل ۱۰۰۲ حافظه اشاره خواهد کرد. برای آشنایی با این مفهوم، شکل ۴-۶ را ببینید. در این شکل دو اشاره‌گر ch و p وجود دارند که ch از نوع کاراکتری و p از نوع int است. هر دو اشاره‌گر ابتدا به محل ۱۰۰۰ حافظه اشاره می‌کنند. افزایش هر واحد به p موجب افزایش ۲ بایت به آن و افزایش هر واحد به ch موجب افزایش یک بایت به آن می‌شود.

```
char  *ch;
int   *p;
```

مقایسه اشاره‌گرها

همانطور که می‌دانید، اگر x و y دو متغیر باشند، با عملگرهای رابطه‌ای می‌توان آنها را با هم مقایسه کرد. اگر p1 و p2 دو اشاره‌گر باشند، با استفاده از عملگرهای رابطه‌ای با هم مقایسه می‌شوند. دستورات زیر را در نظر بگیرید:

```
int    x, y, *p1, *p2 ;
p1 = &x ;
p2 = &y ;
if  (p1 == p2)
        ...
else
        ...
```



شکل ۴-۶ اعمال محاسباتی بر روی اشاره گرها.

با این دستورات، دو اشاره‌گر p1 و p2 با هم مقایسه می‌شوند. فرض کنید p1 به محل ۱۰۰۰ حافظه و p2 به محل ۱۲۰۰ حافظه اشاره می‌کند. در این صورت شرط if(p1 == p2) ارزش نادرستی دارد.

متغیرهای پویا

چون اشاره‌گر می‌تواند آدرس محلی از حافظه را نگهداری کند، از طریق آدرس آن محل می‌تواند محتویات آن محل را دستکاری کند. بنابراین لزومی ندارد آدرس محلی که در اشاره‌گر قرار می‌گیرد، دارای نام باشد. برای این منظور باید آدرس محلی از حافظه در اشاره‌گر قرار گیرد. امتیاز این روش این‌است‌که پس از اینکه کار با آن محل حافظه به اتمام رسید، می‌توان آن حافظه را آزاد کرد و به سیستم تحویل داد. این روش تخصیص حافظه را **تخصیص حافظه پویا** گویند. می‌توان اینطور تصور کرد که محلهایی از حافظه، که بدین طریق، آدرس آنها در اشاره‌گر قرار می‌گیرند، **متغیرهای بی‌نام** هستند. این متغیرها را **متغیرهای پویا** نیز می‌گویند، زیرا در زمان اجرای برنامه توسط برنامه‌نویس ایجاد شده و از بین می‌روند.

تخصیص حافظه پویا

برای اخذ حافظه از سیستم، از تابع ()malloc استفاده می‌شود. این تابع، حافظه‌ای را از سیستم گرفته، آدرس آن را در یک اشاره‌گر قرار می‌دهد. تابع ()malloc که در فایل stdlib.h قرار دارد، به صورت زیر به کار می‌رود:

اشاره‌گر = (نوع) malloc (size) ;

در این روش کاربرد، **(نوع)**، نوع اشاره‌گری است که آدرس حافظه باید در آن قرار گیرد و size میزان حافظه به بایت است و مشخص می‌کند که این تابع چند بایت از حافظه را باید از سیستم اخذ کند. اگر تابع ()malloc به هر دلیلی نتواند حافظه‌ای را از سیستم بگیرد و در اختیار برنامه‌نویس قرار دهد، مقدار تهی (NULL) را در اشاره‌گر قرار می‌دهد. اشاره‌گر تهی، اشاره‌گری است که به جایی اشاره نمی‌کند. دستورات زیر را در نظر بگیرید:

```
int  * p ;
p = (int *) malloc (sizeof(int)) ;
```

دستور اول، p را اشاره‌گر صحیح تعریف می‌کند و دستور دوم حافظه‌ای به اندازه sizeof(int) بایت را از سیستم گرفته آدرس آن را در p قرار می‌دهد. عبارت (* int) که قبل از ()malloc آمده است، تبدیل نوع (type casting) است.

برگرداندن حافظه به سیستم

حافظه‌ای که به صورت پویا تخصیص یافت، پس از استفاده باید به سیستم برگردانده شود. حافظه‌ای که به وسیله ()malloc اختصاص یافت با ()free به سیستم برمی‌گردد. تابع ()free در فایل stdlib.h قرار دارد و به صورت زیر به کار می‌رود:

free (اشاره‌گر) ;

به عنوان مثال، حافظه‌ای را که قبلاً با تابع ()malloc تخصیص یافت و آدرسش در p قرار گرفت با دستور زیر به سیستم برمی‌گردد:

```
free (p) ;
```

مثال ۱-۶

برنامه‌ای که از طریق تخصیص حافظه پویا، دو مقدار را از ورودی خوانده مجموع مربعات آنها را محاسبه می‌کند و به خروجی می‌برد.

توضیح

پس از اجرای تابع malloc() برای تخصیص حافظه پویا، باید تست کرد که آیا حافظه اختصاص یافت یا خیر. اگر حافظه اختصاص نیافته باشد، مقدار NULL در اشاره‌گر قرار می‌گیرد. به همین منظور، در این برنامه، چنانچه NULL برگردانده شود، برنامه با تابع exit(1) خاتمه می‌یابد. وظیفه این تابع خاتمه دادن به برنامه است و در فایل stdlib.h قرار دارد.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main()
{
  int *x, *y, s;
  clrscr();
  x = (int *) malloc(sizeof(int));
  if(!x){
     printf("\n allocation failure.");
     exit(1);
  }
  y = (int *) malloc(sizeof(int));
  if(!y){
     printf("\n allocation failure.");
     exit(1);
  }
  printf("\n enter two integers:");
  scanf("%d%d", x, y);
  s = *x * *x + *y * *y;
  printf("\n sum of square is:%d", s);
  free(x);
  free(y);
  getch();
  return 0;
}
```

خروجی

```
enter two integers : 5  10
sum of square is : 125
```

اشاره گرها و توابع

در فصل ۴ دو روش برای ارسال آرگومان‌ها به توابع مطرح شد ۱. از طریق فراخوانی با مقدار ۲. از طریق فراخوانی با ارجاع. روش اول در آن فصل مورد بحث قرار گرفت و در مثال‌هایی به کار برده شد. در ارسال آرگومان‌ها با روش فراخوانی با ارجاع، به جای مقدار آرگومان‌ها، آدرس آنها در پارامتر قرار می‌گیرد. بدین‌ترتیب، در تابعی که فراخوانی می‌شود، به آدرس متغیرهای موجود در برنامه فراخوان دسترسی داریم. از طریق آدرس این متغیرها می‌توانیم محتویات آنها را دستکاری کنیم.

مثال ۲-۶

برنامه‌ای که دو عدد را از ورودی خوانده، با استفاده از تابعی محتویات آنها را با هم عوض می‌کند.

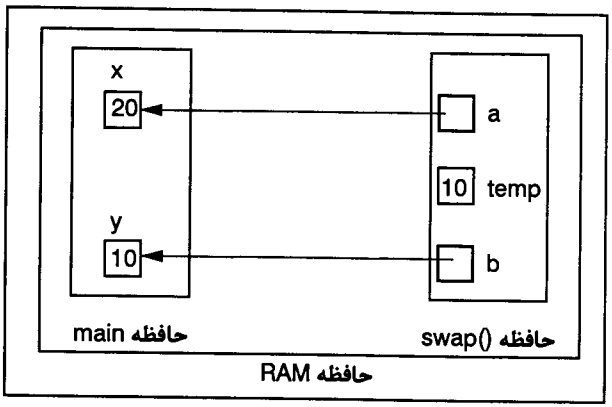
توضیح

چون در این برنامه، تابع باید دو متغیر از تابع اصلی را دستکاری کند، هر دو مقدار از طریق فراخوانی باارجاع ارسال می‌شوند (شکل ۵-۶ پس از اجرای برنامه). تابع ()swap برای جابجایی محتویات دو متغیر از طریق آدرس آنها، از متغیر کمکی temp استفاده می‌کند. ابتدا با دستور temp=*a آنچه که در x است در temp قرار می‌گیرد. سپس با دستور *a=*b آنچه که در y است در x قرار می‌گیرد و در آخر، با دستور *b=temp آنچه که در temp است در y قرار می‌گیرد(زیرا a به x و b به y اشاره می‌کند).

```
#include <stdio.h>
#include <conio.h>
void swap(int *, int *);
int main()
{
    int x = 10, y = 20 ;
    clrscr();
    printf("\n first value of x,y are:%d, %d", x, y) ;
    swap(&x, &y);
    printf("\n final value of x, y are:%d, %d", x, y) ;
    getch();
    return 0;
}
void swap(int *a, int *b)
{
    int temp ;
    temp=*a ;
    *a=*b ;
    *b=temp ;
}
```

خروجی

```
first value of x, y are : 10 , 20
final value of x, y are : 20 , 10
```



شکل ۵-۶
متغیرهای مثال ۲-۶، پس از اجرای برنامه.

مثال ۳-۶

برنامه‌ای که طول و عرض مستطیلی را از ورودی خوانده مساحت و محیط مستطیل را محاسبه می‌کند و به خروجی می‌برد.

توضیح

تابع winput() طول و عرض مستطیلی را از ورودی خوانده، به برنامه ارسال می‌کند. برنامه آنها را به تابع rect() ارسال می‌کند و این تابع مساحت و محیط مستطیل را محاسبه کرده، به برنامه اصلی برمی‌گرداند و برنامه اصلی آنها را به خروجی می‌برد. هدف از این برنامه این است که با ارسال پارامترها از طریق ارجاع و مقدار، آشنا شوید. هر دو پارامتر winput() از طریق ارجاع و دو پارامتر تابع rect() از طریق ارجاع و دو پارامتر دیگر از طریق مقدار ارسال می‌شوند (شکل ۶-۶ را ببینید).

متغیرهای برنامه

نام برنامه	نام متغیر	هدف
main	len	طول مستطیل
	wid	عرض مستطیل
	area	مساحت مستطیل
	per	محیط مستطیل
winput()	p1	پارامتر ارجاع که به طول مستطیل در برنامه اشاره می‌کند
	p2	پارامتر ارجاع که به عرض مستطیل در برنامه اشاره می‌کند
rect()	x	طول مستطیل (پارامتر مقدار)
	y	عرض مستطیل (پارامتر مقدار)
	a	پارامتر ارجاع که به مساحت مستطیل در برنامه اشاره می‌کند
	p	پارامتر ارجاع که به محیط مستطیل در برنامه اشاره می‌کند

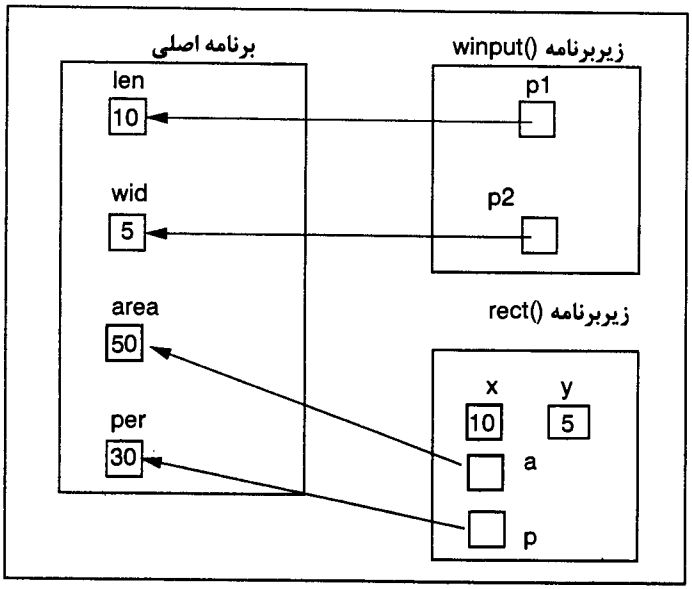
```
#include <stdio.h>
#include <conio.h>
void winput(int *, int *);
void rect(int, int, int *, int *);
int main()
{
  int len, wid, area, per;
  clrscr();
  winput(&len, &wid);
  rect(len, wid, &area, &per);
  printf(" length = %d, width = %d", len, wid);
  printf("\n area = %d, perimer = %d", area, per);
  getch();
  return 0;
}
 //***************************
void winput(int *p1, int *p2)
{
  printf(" enter length, width :");
  scanf("%d%d", p1, p2);
}
 //***************************
void rect(int x, int y, int *a, int *p)
{
  *a = x * y;
  *p = 2 * (x + y);
}
```

خروجی

```
enter length, width : 10  5
length = 10 , width = 5
area = 50 , perimer = 30
```

اجرای تابع با استفاده از آدرس آن

یکی از ویژگیهای مهم اشاره‌گرهای زبان C وجود اشاره‌گرهای تابع است. هر چند که یک تابع، یک متغیر نیست، اما در حافظه، آدرسی دارد که می‌توان آن آدرس را در یک اشاره‌گر قرار داد و از طریق آن اشاره‌گر، آن تابع را اجرا کرد. آدرس تابع را می‌توان با نام تابع (بدون ذکر پارامترهای آن) پیدا کرد. برای تعریف اشاره‌گرهای تابع، باید ضمن تعریف نوع، آن را در داخل پرانتز قرار داد و بعد از آن نیز از پرانتز باز و بسته استفاده کرد (مثال ۴-۶ را ببینید).



شکل ۶-۶
تشریح متغیرهای برنامه
مثال ۶-۳ پس از اجرای برنامه.

مثال ۶-۴

برنامه‌ای که با استفاده از اشاره‌گر تابع، تابعی را فراخوانی کرده، دو رشته را از ورودی خوانده، تشخیص می‌دهد که آیا این دو رشته با هم مساویند یا خیر (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void check(char *a, char *b, int (*cmp)(const char *, const char *));
//int cmp(char *, char *);
int main() {
    char s1[80], s2[80] ;
    int (*p)(const char *, const char *);
    clrscr();
    p = strcmp;
    printf(" enter first string :") ;
    gets(s1) ;
    printf(" enter second string :") ;
    gets(s2) ;
    check(s1, s2, p);
    getch();
    return 0;
}
 //**********************
void check(char *a, char *b, int (*cmp)(const char *, const char *)) {
   if(!(cmp)(a, b))
      printf(" the strings are equal.");
   else
      printf(" strings are not equal.");
}
```

خروجی

```
enter first string : ali
enter second string : ahmad
strings are not equal.
```

تحلیل مثال ۴-۶

آرگومان‌های تابع ()check عبارت‌اند از رشته‌های s1 و s2 و اشاره‌گر p و پارامترهای آن عبارت‌اند از دو اشاره‌گر رشته‌ای و یک اشاره‌گر از نوع تابع. پرانتزهایی که در اطراف cmp* آمده‌اند، ضروری‌اند، زیرا در غیر اینصورت، این اشاره‌گر به عنوان اشاره‌گر تابعی محسوب نخواهد شد. در تابع ()check دستور (a, b)(cmp*) موجب اجرای تابع ()strcmp می‌شود، زیرا آدرس آن تابع در اشاره‌گر cmp قرار دارد. ضمناً این دستور، روش کلی فراخوانی توابع از طریق آدرس آنها را نشان می‌دهد. توجه داشته باشید که تابع ()strcmp، از طریق فراخوانی ()check به صورت زیر نیز اجرا می‌شود:

```
check (s1, s2, strcmp) ;
```

ممکن‌است فراخوانی توابع از طریق آدرس، کمی دشوار به نظر برسد، اما این روش، در نوشتن کامپایلرها و برنامه تجزیه‌کننده کاربرد فراوانی دارد.

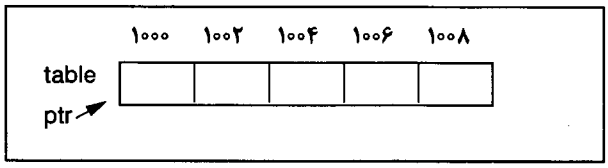
اشاره‌گرها و آرایه‌ها

در زبان C، بین آرایه‌ها و اشاره‌گرها ارتباط نزدیکی وجود دارد. اشاره‌گرها حاوی آدرس‌اند و اسم آرایه نیز یک آدرس است. اسم آرایه اولین عنصر آرایه را مشخص می‌کند؛ به عبارت دیگر، اسم آرایه، آدرس اولین محلی را که عناصر آرایه از آنجا به بعد در حافظه ذخیره می‌شوند، نگهداری می‌کند. پس **اسم آرایه**، یک اشاره‌گر است. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
int  table [5] ;
int  * ptr ;
```

اگر اولین عنصر آرایه در محل ۱۰۰۰ حافظه وجود داشته باشد، table به محل ۱۰۰۰ حافظه اشاره خواهد کرد (شکل ۷-۶). به موارد زیر توجه کنید:

ptr و table به ابتدای آرایه اشاره می‌کنند			**ptr = table ;**
عنصر دوم آرایه	**table [1]**	**معادل است با**	***(ptr + 1)**
عنصر سوم آرایه	***(table + 2)**	**معادل است با**	**ptr [2]**
عنصر اول آرایه	***table**	**معادل است با**	***ptr**



شکل ۷-۶ ارتباط بین آرایه و اشاره‌گر.

مثال ۵-۶

برنامه‌ای که ۵ عدد را از ورودی خوانده، در آرایه‌ای قرار می‌دهد و سپس عناصر آرایه را از آخرین عنصر به اولین عنصر به خروجی می‌برد.

```
#include <stdio.h>
#include <conio.h>
int main() {
    int arr[5] ,i ;
    clrscr();
    printf("\n enter five value :") ;
    for(i=0 ; i<5 ; i++)
      scanf("%d",&arr[i]) ;
    printf("\n the reverse output is :");
    for(i=4 ; i>=0 ; i -- )
        printf("%4d",*(arr+i)) ;
    getch();
    return 0;
}
```

خروجی

```
enter five value : 14  7  8  9  5
the revers output is : 5  9  8  7  14
```

آرایه پویا

در ابتدای این فصل، با مفهوم متغیرهای پویا آشنا شدید. آرایه‌ها نیز می‌توانند پویا باشند. یعنی حافظه موردنیاز آرایه را نیز می‌توان در حین اجرای برنامه از سیستم اخذ کرد و پس از استفاده، به سیستم برگرداند. برای تخصیص حافظه به آرایه نیز از تابع ()malloc استفاده می‌شود. دستورات زیر را در نظر بگیرید:

```
int  *x, n ;
scanf ("%d",&n) ;
x = (int *) malloc (sizeof (int) * n) ;
```

دستور سوم حافظه‌ای را که اندازه آن با sizeof(int) * n مشخص می‌شود اختصاص داده آدرس آن را در x قرار می‌دهد. لذا x یک آرایه صحیح با n عنصر است. اکنون با مثال ساده‌ای، کاربرد آرایه‌های پویا را تشریح می‌کنیم.

مثال ۶-۶

برنامه‌ای که تعداد n عدد را از ورودی خوانده در آرایه‌ای پویا قرار می‌دهد و سپس آرایه را به تابع ارسال می‌کند. تابع، میانه اعداد موجود در آرایه را پیدا کرده، به برنامه برمی‌گرداند.

توضیح

برای محاسبه میانه اعداد، باید آنها را مرتب کرد. اگر تعداد اعداد زوج باشد، میانگین دو عدد وسط برابر با میانه آن اعداد است و اگر تعداد اعداد فرد باشد، میانه عددی است که نیمی از اعداد از آن بزرگتر و نیمی دیگر از آن کوچکتر باشند. در این برنامه از چهار تابع استفاده شده است که شرح وظایف آنها در ادامه آمده است. در این برنامه، برای

ذخیره آرایه از تخصیص حافظه پویا استفاده شد. به‌این‌ترتیب که، تعداد عناصر آرایه (n) از ورودی خوانده شد و سپس با استفاده از تابع ()malloc حافظه به آرایه اختصاص یافت. پس از استفاده از آرایه، حافظه آن به سیستم پس داده شده است. دستور زیر، حافظه‌ای را برای ذخیره n عنصر صحیح ایجاد می‌کند.

```
int   p ;
p = (int *) malloc (n * sizeof (int)) ;
```

شرح وظایف برنامه‌ها

برنامه ()main: تخصیص حافظه به آرایه و فراخوانی زیربرنامه‌ها. در این برنامه، n تعداد عناصر آرایه، i شمارنده حلقه تکرار و mead میانه اعداد است و p به آرایه اشاره می‌کند.

تابع ()pinput: خواندن عناصر آرایه از ورودی، آرگومان اول آن اشاره‌گر به آرایه و آرگومان دوم، طول آرایه است.

تابع ()bubble: آرایه را مرتب می‌کند. آرگومان اول اشاره‌گر به آرایه و آرگومان دوم، طول آرایه است.

تابع ()median: این تابع، اشاره‌گر به آرایه، طول آرایه و متغیر مربوط به میانه عدد را گرفته، میانه را محاسبه می‌کند و در آن متغیر قرار می‌دهد (فراخوانی باارجاع).

تابع ()pout: عناصر آرایه را پس از مرتب شدن به خروجی می‌برد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void pinput(int *, int);
void bubble(int *, int);
void median(int *, int, float *);
void pout(int *, int);
int main() {
  int n, i;
  int  *p;
  float mead;
  clrscr();
  printf(" enter number of items(n):");
  scanf("%d", &n);
  p = (int *)malloc(n * sizeof(int));
  if(!p){
       printf(" aloocation failure!");
       getch();
       exit(1);
  }
  pinput(p, n);
  bubble(p, n);
  printf("\n sorted data are:");
  pout(p, n);
```

```
  median(p, n, &mead);
  printf("\n median = %5.2f", mead);
  getch();
  return 0;
}
 //*****************************
void pinput(int  *p, int n)
{
  int i;
  for(i = 0; i < n; i ++){
     printf("enter number %d:", i + 1);
       scanf("%d", p + i);
  }
}
 //*****************************
void bubble(int  *temp, int len)
{
   int i, j, item;
   for(i = len - 1 ; i > 0; i --)
       for(j = 0; j < i ; j++)
           if(*(temp + j) > *(temp + j + 1)) {
              item = *(temp + j);
              *(temp + j) = *(temp + j + 1);
              *(temp + j + 1) = item ;
           }//end of if
}
 //*************************
void median(int *x, int n, float *mead)
{
  if(n % 2 == 0)
       *mead =(float) (*(x + ((n - 1) / 2)) + *(x + (n / 2))) / 2;
  else
       *mead = *(x + (n - 1) / 2);
}
 //****************
void pout(int *p, int n)
{
  int i;
  for(i = 0; i < n; i ++){
     printf("%3d",*(p+i));
  }
  printf("\n");
}
```

خروجی

```
enter number of items (n) : 6
enter number 1 : 10
enter number 2 : 9
enter number 3 : 8
enter number 4 : 4
enter number 5 : 14
enter number 6 : 19
sorted data are : 4  8  9  10  14  19
median = 9.50
```

اشاره‌گرها و رشته‌ها

در بخش قبلی، ارتباط بین آرایه‌ها و اشاره‌گرها مطرح شد و دیدید که نام آرایه، یک اشاره‌گر است. چون رشته‌ها در C به صورت آرایه‌ای از کاراکترها تعریف می‌شوند، ارتباط بین رشته‌ها و اشاره‌گرها همانند ارتباط بین آرایه‌ها و اشاره‌گرهاست.

مثال ۷-۶

برنامه‌ای که یک رشته و کاراکتری را از ورودی خوانده، تعداد دفعات تکرار کاراکتر را در رشته پیدا می‌کند و در خروجی چاپ می‌نماید.

توضیح

در این برنامه از تابع char_count() برای شمارش تعداد تکرار کاراکتر استفاده شده است. رشته و کاراکتر خوانده شده، به تابع تحویل داده می‌شوند. متغیر string حاوی رشته، ch حاوی کاراکتر و count تعداد تکرار ch است.

```
#include <stdio.h>
#include <conio.h>
int char_count(char *, char);
int main()
{
    char string[40] , ch ;
    int count ;
    clrscr();
    printf(" enter string for search:");
    gets(string) ;
    printf(" enter a character:") ;
    ch = getche() ;
    count = char_count(string, ch) ;
    printf("\n number of occurs of char ") ;
    printf(" <%c> is:%d", ch, count);
    getch();
    return 0;
}
```

```
//*******************
int char_count(char *s, char letter)
{
    int count=0 ;
    while(*s)
      if(*s++ == letter)
          count ++ ;
     return(count) ;
}
```

خروجی

```
enter string for search : a book about computer.
enter a character : a
number of occurs of char <a> is : 2
```

مثال ۸-۶

برنامه‌ای که دو رشته را از ورودی خوانده، سپس آن دو رشته را طوری با هم مقایسه می‌کند که اولین کاراکتر مورد اختلاف دو رشته را می‌یابد.

توضیح

تابع ()main رشته‌های st1 و st2 را از ورودی خوانده تابع ()first_dif را فراخوانی می‌کند. تابع ()first_dif رشته‌ها را کاراکتر به کاراکتر مقایسه می‌کند. اگر دو کاراکتر نامساوی باشند، چک می‌کند که آیا متغیر ignore_case برابر با یک است یا خیر، اگر برابر با یک باشد، کاراکترهای دو رشته را به حروف بزرگ تبدیل می‌کند و سپس آنها را با هم مقایسه می‌کند. اگر پس از تبدیل نیز با هم مساوی نبودند، اولین مورد اختلاف پیدا شده، مقایسه خاتمه می‌یابد. هرگاه دو کاراکترِ مورد مقایسه با هم برابر باشند، کاراکترهای بعدی مقایسه می‌شوند.

متغیرهای برنامه

تابع	متغیر	هدف
main()	st1,st2	رشته‌هایی که با هم مقایسه می‌شوند
	compcase	نوع مقایسه را مشخص می‌کند. اگر برابر با یک باشد، تفاوتی بین حروف کوچک و بزرگ نیست و اگر صفر باشد، بین حروف کوچک و بزرگ تفاوت است.
first_dif()	s1,s2	رشته‌هایی که باید با هم مقایسه شوند (پارامتر تابع)
	ignore_case	مانند compcase در برنامه اصلی عمل می‌کند
	i	شمارنده حلقه
	a,b	کاراکترهایی از رشته‌های s1 و s2 که باید با هم مقایسه شوند.

```
#include <stdio.h>
#include <conio.h>
first_dif(char *, char *, int);
int main()
{
      char st1[40], st2[40] ;
      int compcase = 1 ;
      clrscr();
      printf(" enter first string:") ;
      gets(st1) ;
      printf(" enter second string:");
      gets(st2) ;
      first_dif(st1, st2, compcase) ;
      getch();
      return 0;
}
//*********************
first_dif(char *s1, char *s2, int ignore_case)
    /* if ignore_case is 1 ,ignore case
          of letter */
{
   int i ;
   char a , b ;
   for(i = 0; *s1 && *s2; s1++, s2++, i++)
     if(*s1 != *s2) {
        if(ignore_case) {
          a = (*s1 >= 'a' && *s1 <= 'z') ? *s1 -= 32 : *s1;
          b = (*s2 >= 'a' && *s2 <= 'z') ? *s2 -= 32 : *s2;
          if(a != b)
             break ;
        }//end of if
      else
        break;
      }//end of if
      if(*s1 || *s2) {
        printf("\n strings are not equal,");
        printf(" the first difference occurs in :%d", i + 1);
      }
      else
        printf("\n strings are equal .") ;
}
```

خروجی

```
enter first string : computer
enter second string : command
string are not equal, the first different accurs in : 4
```

مثال ۹-۶

برنامه‌ای که رشتهٔ عددی را از ورودی خوانده، آن را به مقدار عددی تبدیل می‌کند.

توضیح

اگر بخواهیم رشته عددی موجود در برنامه را در محاسبات به کار ببریم، باید آن را به مقدار عددی تبدیل کنیم. به عنوان مثال، اگر رشته "145" موجود باشد، می‌خواهیم آن را به عدد 145 تبدیل کرده، در محاسبات به کار گیریم. برای این منظور تابعی به نام ascii_to_int() نوشته شده است. این تابع رشته عددی و متغیری را به عنوان آرگومان گرفته، رشته را به عدد تبدیل کرده در متغیر عددی قرار می‌دهد. اعمالی که این تابع انجام می‌دهد عبارت‌انداز: حذف فضای خالی ابتدای رشته عددی، تشخیص علامت عدد، تشخیص کاراکترهای نامعتبر، و تبدیل کاراکتر به رقم. برای تبدیل کاراکتر به رقم، باید از کداسکی کاراکتر '0' تا '9' به اندازه ۴۸ واحد کم شود. بنابراین '0'-48 به مقدار عددی صفر تبدیل می‌شود. اگر کاراکتر نامعتبری وجود داشته باشد، تابع ضمن صدور پیام، تا آنجایی که معتبر بوده است را به برنامه برمی‌گرداند. پس از تبدیل کاراکتر '0' تا '9' به ارقام ۰ تا ۹، آنها را از طریق ضرب‌کردن در ۱۰ به هم وصل می‌کنیم. به عنوان مثال، اگر سه رقم ۱، ۵ و ۷ داشته باشیم می‌توانیم به طریق زیر، عدد ۷۵۱ را ایجاد کنیم.

۷ × ۱۰ = ۷۰
۷۰ + ۵ = ۷۵
۷۵ × ۱۰ = ۷۵۰
۷۵۰ + ۱ = ۷۵۱

این عمل در تابع با دستور زیر انجام می‌شود:

```
*value = (*value *10) + (*str ++ -48)
```

پس از تبدیل رشته به عدد، علامت آن که در متغیر sign قرار دارد، در عدد ضرب می‌شود (اگر sign برابر با ۱- باشد یعنی عدد منفی است).

```
#include <conio.h>
#include <stdio.h>
void ascii_to_int(int *, char *);
int main()
{
   int number ;
   char s[10] ;
   clrscr();
   printf("\n enter numeric string:");
   gets(s) ;
    ascii_to_int(&number, s) ;
   printf(" numeric value is:%d", number) ;
   getch();
   return 0;
}
```

```
//**************************
void ascii_to_int(int *value, char *str)
{
  int sign = 1 ;
  *value = 0 ;
  while(*str == ' ') str++ ;
    if(*str== '-' || *str == '+')
      sign = (*str ++ == '-') ? -1 : 1;
    while(*str)
      if((*str >= '0') && (*str <= '9'))
        *value = (*value * 10) + (*str ++ -48) ;
      else {
        printf("Watning :the <%c> is invalid character.\n", *str);
        break ;
      }
    *value *=sign ;
}
```

خروجی

```
enter numeric string : 175
numeric value is : 175
```

مثال ۱۰-۶

برنامه‌ای که مقدار عددی صحیح را از ورودی خوانده، به رشته تبدیل می‌کند.

توضیح

این برنامه برعکس برنامه ۹-۶عمل می‌کند و توضیحات آن برنامه، برای این مثال مفید واقع می‌شود.

متغیرهای برنامه

تابع	متغیر	هدف
main()	number	عددی که باید به رشته تبدیل شود
	s	مقدار رشته‌ای عدد number
int_to_ascii()	value	مقدار عددی که از ورودی خوانده شد
	str	اشاره‌گری که حاوی رشتهٔ عددی است
	saveval	مقدار value را نگهداری می‌کند تا در اثر تغییر value مقدار اولیه آن از بین نرود
	savestr	اشاره‌گر رشته‌ای که به ابتدای رشته اشاره می‌کند تا در اثر تغییر اشاره‌گر str ابتدای رشته مشخص باشد. از این متغیر برای معکوس کردن رشتهٔ عددی حاصل، استفاده می‌شود
	temp	متغیر کمکی برای جابجایی کاراکتر

شرح وظایف برنامه‌ها

تابع main(): عدد صحیح را از ورودی خوانده، به تابع int_to_ascii() ارسال می‌کند تا به رشته تبدیل شود.

تابع int_to_ascii(): این تابع، عدد موردنظر را دریافت می‌کند و رقمهای آن را جدا کرده، پس از تبدیل به کاراکتر، در رشته، str قرار می‌دهد. چون رقمها از سمت راست عدد جدا می‌شوند و از سمت چپ رشته پر می‌شوند، پس از تبدیل تمام ارقام به کاراکتر و تعیین علامت عدد، باید رشته را معکوس کرد. قبل از جداکردن ارقام عدد، آن عدد در saveval نگهداری می‌شود و چنانچه value منفی باشد، در ۱- ضرب می‌شود تا به عدد مثبت تبدیل شود. علتش این است که به راحتی بتوان، ارقام آن را در یک حلقه تکرار و با شرط (value > 0) while جداکرد.

```
#include <stdio.h>
#include <conio.h>
void Int_to_ascii(int , char *);
int main() {
    int number ;
    char s[10] ;
    clrscr();
    printf(" enter a number :");
    scanf("%d", &number) ;
     int_to_ascii(number, s) ;
    printf(" the string value is:%s", s);
    getch();
    return 0;
}
 //**********************************
void int_to_ascii(int value, char *str)
{
    int saveval= value ;
    char temp, *savestr = str ;
    if(value < 0)
       value *= -1 ;
    do{
       *str ++ = (value % 10) + 48 ;
       value = value / 10 ;
    } while(value > 0);
    if (saveval < 0)
       *str++ = '-' ;
    *str -- = '\0' ;
    while(savestr < str) {
      temp = *str ;
      *str -- =* savestr ;
      *savestr ++ = temp ;
    }
}
```

خروجی

```
enter a number : -175
the string value is : -175
```

ارزش دهی اولیه به اشاره گرها

چگونگی ارزش‌دهی اولیه رشته‌ها در فصل ۵، بررسی شد. اکنون می‌آموزیم که چگونه می‌توان رشته‌ها را به عنوان اشاره‌گر مقدار اولیه داد. برای این منظور به صورت زیر عمل می‌شود:

```
char  *نام رشته = "رشته" ;
```

به عنوان مثال، دستور زیر، رشته "computer" را در اشاره‌گر s قرار می‌دهد.

```
char  *s = "computer" ;
```

مثال ۱۱-۶

برنامه‌ای که چگونگی مقدار اولیه دادن به رشته‌ها را به عنوان اشاره‌گرها، تشریح می‌کند.

```
#include <stdio.h>
#include <conio.h>
int main() {
    char *text="your name is:";
    char name[41];
    clrscr();
    printf("\n enter your name :") ;
    gets(name) ;
    printf(" %s" ,text);
    puts(name) ;
    getch();
    return 0;
}
```

خروجی

```
enter your name : ali
your name is : ali
```

اولین دستور برنامه ۱۱-۶ را می‌توان به صورت زیر نوشت:

```
char  text [] = "your name is :" ;
```

این دو دستور از نظر اثری که در حافظه می‌گذارند، تفاوتی با یکدیگر ندارند، ولی چون دستوری که در برنامه آمده است، متغیر text را به صورت اشاره‌گر تعریف کرد، قابلیت انعطاف بیشتری به این متغیر داده است. وقتی رشته‌ها به صورت اشاره‌گر تعریف می‌شوند، بخصوص در مواقعی که طول عناصر مختلف آن متفاوت باشد، موجب صرفه‌جویی در میزان حافظه می‌شود (مثال ۱۲-۶).

مثال ۱۲-۶

برنامه‌ای که برای ذخیره اسامی تعدادی از افراد، از آرایه‌ای از اشاره‌گرها استفاده می‌کند. این برنامه نام ۵ نفر را در آرایه‌ای قرار داده، سپس نامی را از ورودی خوانده، تشخیص می‌دهد که آیا این نام در آرایه وجود دارد یا خیر.

توضیح

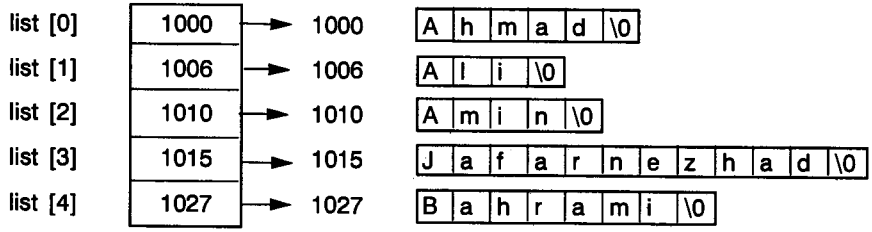
برای ذخیره اسامی افراد، از یک آرایه دو بعدی استفاده شده است. برای دستیابی به هر یک از عناصر آرایه (هر اسم) از یک اندیس استفاده گردیده است.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
    int dex;
    const int k = 5;
    char name[21];
    char *list[k]=
    { "ali",
      "ahmad",
      "alireza",
      "jalal",
      "mohammad"
    } ;
    clrscr();
    printf(" enter a name for search:") ;
    gets(name);
    for(dex = 0; dex <k; dex++)
      if(!strcmp(list[dex], name))
         break ;
    if(dex == k)
      printf(" name <%s> not exist.", name) ;
    else
      printf(" name <%s> exist.", name);
    getch();
    return 0;
}
```

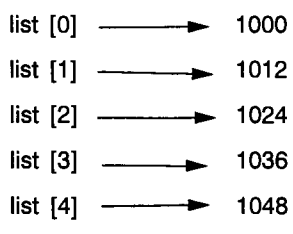
خروجی

enter a name for search : ali
name <ali> exist.

اگر در مثال ۱۲-۶فرض شود که اولین عنصر آرایه list در محل ۱۰۰۰ حافظه قرار داشته باشد، این آرایه به صورتی که در شکل ۸-۶آمده است، در حافظه ذخیره می‌شود. اگر در این مثال، به جای تعریف آرایه‌ای از اشاره‌گرها، آرایه‌ای از رشته‌ها تعریف می‌کردیم، آنگاه می‌بایست آرایه list به صورت آرایه دوبعدی تعریف می‌شد که در آن، تعداد سطرها برابر با تعداد افراد (تعداد عناصر آرایه) و تعداد ستونها برابر با طولِ طولانی‌ترین نامِ موجود بود (شکل ۹-۶).



شکل ۸-۶ نحوه قرارگرفتن آرایه‌ای از اشاره گرهادر حافظه .



A	h	m	a	d	\0						
A	l	i	\0								
A	m	i	n	\0							
J	a	f	a	r	n	e	z	h	a	d	\0
B	a	h	r	a	m	i	\0				

شکل ۹-۶ نحوه قرارگرفتن آرایه‌ای از رشته‌ها در حافظه .

با توجه به ذخیره شدن اطلاعات در اشکال ۸-۶ و ۹-۶، ذخیره کردن رشته‌ها به صورت آرایه‌ای از اشاره‌گرها، در صرفه‌جویی میزان حافظه بسیار مؤثر است ولی در همهٔ موارد اینطور نیست. به عنوان مثال، اگر طول همه رشته‌ها ۱۲ باشد (به اندازه طولِ طولانی‌ترین رشته)، حافظه تخصیص‌یافته در روش اول (آرایه‌ای از اشاره‌گرها)، دارای ۵ اشاره‌گر اضافی خواهد بود که نسبت به روش دوم (آرایه‌ای از رشته‌ها)، حافظه بیشتری را اشغال می‌کند. مزیت عمده بیان رشته‌ها به صورت آرایه‌ای از اشاره‌گرها، سرعت بالاتر و سهولت دسترسی عناصر آرایه است. زیرا نیازی به انجام محاسبات جهت دسترسی به عناصر آرایه نیست.

مثال ۱۳-۶

برنامه‌ای که نام تعدادی از دانشجویان را از ورودی خوانده، در آرایه‌ای قرار می‌دهد. سپس با استفاده از آرایه‌ای از اشاره‌گرها، آرایه را مرتب کرده، در خروجی چاپ می‌کند.

توضیح

حداکثر ۳۰ نام را می‌توانید در آرایه قرار دهید. برای خاتمه ورود اطلاعات، به جای نام دانشجو، فقط کلید enter را فشار دهید. برنامه با دستور if، این وضعیت را تشخیص داده، به دریافت اطلاعات خاتمه می‌دهد. برای درک چگونگی مرتب کردن رشته‌ها از طریق آرایه‌ای از رشته‌ها، به شکلهای ۱۰-۶ و ۱۱-۶ مراجعه شود. این شکلها، نحوه قرارگرفتن عناصر آرایه را در حافظه، قبل و بعد از مرتب شدن، نشان می‌دهند.

متغیرهای برنامه

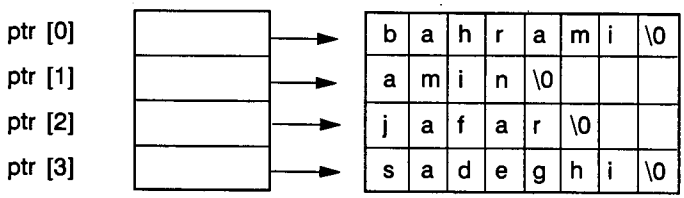
متغیر	هدف
name	آرایه دوبعدی برای ذخیره اسامی دانشجویان
ptr	آرایه‌ای از اشاره‌گرها برای مرتب‌سازی اسامی
in,out	اندیس‌های حلقه‌های تکرار
temp	یک متغیر اشاره‌گر کمکی
count	اندیس حلقه تکرار و شمارنده تعداد اسامی دانشجویان
k	حداکثر تعداد دانشجویان

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
   char name[30][81];
   char *ptr[30], *temp;
   int in, out, count = 0;
   const int k = 30 ;
   clrscr();
   while(count < k) {
      printf(" enter name of number %d:", count + 1);
      gets(name[count]) ;
      if(!name[count][0])
```

```
        if(!name[count][0])
            break ;
        *(ptr + count ++) = name[count] ;
    }
    for(out = 0; out < count - 1; out ++)
      for(in = out + 1; in < count; in++)
          if(strcmp(*(ptr + out), *(ptr + in)) > 0){
            temp = *(ptr + in) ;
            *(ptr + in) = *(ptr + out) ;
            *(ptr + out) = temp ;
      }
      printf("<< the sorted list is:>>\n");
      for(out = 0 ; out < count ; out ++)
          printf("\n name %d is:%s", out + 1, *(ptr + out));
      getch();
      return 0;
}
```

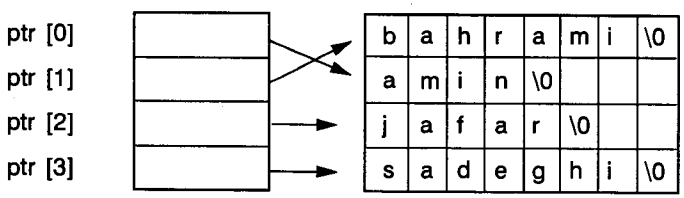
خروجی

```
enter name of number 1 : bahrami
enter name of number 2 : amin
enter name of number 3 : jafar
enter name of number 4 : sadeghi
enter name of number 5 :
<< the sorted list is : >>
name 1 is : amin
name 1 is : bahrami
name 1 is : jafae
name 1 is : sadeghi
```



شکل ۱۰-۶

وضعیت اشاره گرهاقبل از مرتب شدن.



شکل ۱۱-۶

وضعیت اشاره گرهاپس از مرتب شدن.

نکته‌ای راکه باید در مورد مثال ۱۳-۶ توجه داشته باشید این‌است‌که آرایه name به صورت دوبعدی تعریف شد ولی برای دسترسی به عناصر آن، از یک اندیس استفاده شده است. علتش این است که در C می‌توان، قسمتی از آرایه را به عنوان آرایه فرض کرد. یعنی در آرایه دوبعدی، هر سطر را می‌توان یک آرایه یک‌بعدی در نظر گرفت.

اشاره‌گر به اشاره‌گر

اگر متغیری آدرس متغیر دیگر را در خود نگهداری کند، متغیر اول یک اشاره‌گر است؛ اگر متغیر دوم، از نوع اشاره‌گر باشد در این صورت متغیر اول یک اشاره‌گر به اشاره‌گر است (شکل ۱۲-۶). یادآوری می‌شود که آرایه‌ای از اشاره‌گرها، نوعی اشاره‌گر به اشاره‌گر است. برای تعریف متغیرهای اشاره‌گر به اشاره‌گر، از دو علامت * استفاده می‌شود. به عنوان مثال، در دستور زیر، p، اشاره‌گر به اشاره‌گر است:

```
int   **p ;
```

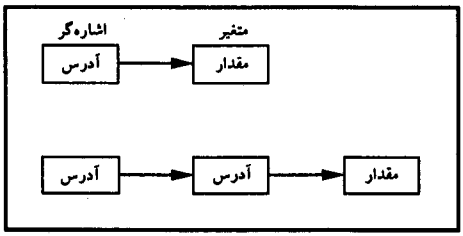
مثال ۱۴-۶

برنامه‌ای که کاربرد اشاره‌گر به اشاره‌گر را نشان می‌دهد.

```
#include <stdio.h>
int main()
{
    int x , *p ,**q ;
    clrscr();
    x = 10 ;
    p = &x ;
    q = &p ;
    printf("\n the q points to value:") ;
    printf("%d",**q) ;
    getch();
    return 0;
}
```

خروجی

```
the q points to value : 10
```



شکل ۱۲-۶ اشاره‌گربه اشاره‌گر.

نکاتی در مورد اشاره‌گرها

اشاره‌گرها، علیرغم امکاناتی که فراهم می‌کنند، اشکالاتی را نیز می‌توانند به‌وجود آورند. برنامه‌نویس برای استفاده بهینه از اشاره‌گرها باید این اشکالات را بشناسد.

اشکال اول : استفاده از اشاره‌گرهایی که قبلاً مقدار نگرفته‌اند.

مثال ۱۵-۶

نمونه‌ای از کاربرد اشاره‌گرهایی که قبلاً مقدار نگرفته‌اند.

```
#include <conio.h>
#include <stdio.h>
main() {
   int x , *p ;
   clrscr();
   x = 10 ;
   *p = x ;
   printf("\n %d", *p) ;
   getch();
   return 0;
}
```

در برنامهٔ مثال ۱۵-۶عدد ۱۰ به متغیر x نسبت داده می‌شود و دستور ;p=x* به ماشین می‌گوید "محتویات متغیر x را در آدرسی که اشاره‌گر p به آن اشاره می‌کند قرار بده". چون اشاره‌گر p قبلاً مقدار نگرفته است ممکن است هر ناحیه‌ای از حافظه، از جمله ناحیه سیستم را تغییر دهد. برای رفع این مشکل کافی است در اشاره‌گر p مقدار معتبری قرار گیرد.

اشکال دوم : عدم استفاده صحیح از اشاره‌گرها .

مثال ۱۶-۶

استفاده نادرست از اشاره‌گرها.

```
#include <stdio.h>
#include <conio.h>
main() {
   int x , *p;
   clrscr();
   x = 10 ;
   p = x ;
   printf("\n %d", *p);
   getch();
   return 0;
}
```

هدف برنامهٔ مثال ۱۶-۶ این بود که مقدار متغیر x را که برابر با ۱۰ است، در خروجی چاپ نماید، ولی به دلیل نادرست بودن دستور ; p = x (با توجه به اشاره‌گر بودن p) نتیجهٔ مطلوب حاصل نخواهد شد. این دستور موجب می‌شود تا عدد ۱۰، نه به عنوان یک مقدار بلکه به عنوان یک آدرس به اشاره‌گر p منتقل گردد. برای رفع این مشکل کافی است این دستور را به صورت ; p = &x نوشت تا آدرس متغیر x به اشاره‌گر p منتقل شود.

اشکال سوم : فرضهایی که برنامه‌نویس در محل قرارگرفتن متغیرها در حافظه دارد.

وقتی متغیرها در حافظه قرار می‌گیرند، جای آنها برای ما مشخص نیست و در هر جایی که فضای کافی وجود داشته باشد این متغیرها در آنجا ذخیره می‌شوند (مثال‌های ۱۷-۶ و ۱۸-۶).

مثال ۱۷-۶

نمونه‌ای از فرضهای غلط که برنامه‌نویس باید از آنها بپرهیزد.

توضیح

از اینکه s قبل از y تعریف شده است، نمی‌توان تصور کرد که s در حافظه قبل از y قرار دارد.

```
#include <stdio.h>
#include <conio.h>
int main() {
    char s[80], y[80] ;
    char *p1 , *p2 ;
    clrscr();
    p1 = s ;
    p2 = y ;
    printf("\n p1 is %p, p2 is %p", p1, p2);
    printf("\n p1 points to lower addree?");
    getch();
    return 0;
}
```

مثال ۱۸-۶

نمونه‌ای از فرضهای نادرستِ برنامه‌نویس راجع به محل قرارگرفتن متغیرها.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int first[10], second[10] ;
    int *p , t ;
    p=first ;
    for(t = 0 ; t < 20; t ++)
      *p++ = t ;
}
```

در مثال ۱۸-۶ سعی شد به عناصر آرایه‌های first و second، اعداد صفر تا ۱۹ نسبت داده شود، گرچه ممکن است این عمل در بعضی از کامپایلرها به درستی انجام گیرد، ولی در حالت کلی این طور نیست، زیرا ممکن است عناصر آرایه‌های first و second در محلهای متوالی حافظه قرار نگیرند.

با اشکالاتی که تاکنون در مورد اشاره‌گرها گفته شد برنامه‌نویس باید در استفاده از اشاره‌گرها دقت کافی به خرج دهد تا با مشکل مواجه نگردد.

آرگومان‌های تابع ()main

اولین تابع یک برنامه که اجرا می‌شود، تابع ()main است. این تابع همانند توابع دیگر می‌تواند دارای آرگومان باشد. تابع ()main دارای دو پارامتر به نامهای argc و argv است. پارامتر argc از نوع صحیح بوده، مشخص کننده تعداد آرگومان‌ها است. چون نام برنامه به عنوان یک آرگومان محسوب می‌شود لذا حداقل مقدار argc برابر با ۱ است. بنابراین اگر برنامه‌ای مانند test دارای دو آرگومان باشد عددی که در پارامتر argc قرار می‌گیرد برابر با ۳ خواهد بود. پارامتر argv به آرایه‌ای رشته‌ای اشاره می‌کند، که عناصر آن، به آرگومان‌های تابع اشاره می‌کنند. به عبارت دیگر، []argv به آرایه‌ای از اشاره‌گرهای کاراکتری اشاره می‌کند. لذا کلیه آرگومان‌های تابع اصلی به صورت رشته‌ای فرض می‌شوند. بنابراین اگر خواسته باشیم از اعدادی که به عنوان آرگومان به تابع اصلی منتقل می‌شوند استفاده کنیم، باید به طریق مقتضی (با استفاده از توابع کتابخانه‌ای و یا توابعی که خودمان می‌نویسیم) آنها را از رشته‌ای، به عددی تبدیل کنیم.

به دو روش می‌توان آرگومان‌ها را به تابع ()main ارسال کرد: ۱. در محیط C و ۲. در محیط سیستم عامل. در روش اول، قبل از اجرای برنامه، گزینه Arguments را از منوی RUN در محیط بورلند C انتخاب کنید و در کادری که ظاهر می‌شود، نامی را تایپ کرده، کلید Enter را فشار دهید. برای استفاده از روش دوم، باید برنامه را به فایل اجرایی تبدیل کنید. وقتی برنامه زبان C توسط کامپایلر زبان ترجمه شد این برنامه در خارج از محیط C و در سطح سیستم عامل قابل اجرا است و در حین اجرای این برنامه اسامی آرگومان‌ها نیز جهت ارسال به تابع اصلی ذکر می‌شوند. به عنوان مثال، فرض کنید برنامه‌ای به نام test.cpp نوشته، توسط کامپایلر C آن را ترجمه کرده و برنامه‌ای به نام test.exe از آن ساخته‌ایم؛ برای اجرای این برنامه کافی است در سطح سیستم عامل به صورت زیر عمل کنیم (با فرض این که این برنامه در درایو جاری وجود دارد):

```
C:\BC> test
```

اگر فرض شود که این برنامه دارای دو پارامتر باشد، برای اجرای آن در سطح سیستم عامل، باید اسامی آرگومان‌ها را با یک فاصله به صورت زیر تایپ کنیم:

```
C:\BC> test par1 par2
```

par1 و par2 اسامی آرگومان‌هایی هستند که به تابع اصلی منتقل می‌شوند. چون در این روش، آرگومان‌ها در خط فرمان به برنامه سیستم‌عامل وارد می‌شوند، آرگومان‌های تابع ()main را **آرگومان‌های خط فرمان** نیز می‌گویند. در مورد ترتیب دسترسی به آرگومان تابع ()main باید دقت داشت که: [0]argv به نام برنامه، [1]argv به اولین آرگومان، [2]argv به دومین آرگومان و [n]argv به nامین آرگومان اشاره می‌کنند.

مثال ۱۹-۶

برنامه‌ای که یک عدد را به عنوان آرگومان پذیرفته، عمل شمارش معکوس، از آن عدد به صفر را انجام می‌دهد. این برنامه می‌تواند آرگومان دوم نیز داشته باشد؛ اگر آرگومان دوم برابر با "display" باشد، نتیجهٔ شمارش معکوس در صفحه‌نمایش چاپ خواهد شد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
int main(int argc, char *argv[]) {
   int disp, count ;
   clrscr();
   if(argc <= 2) {
      printf(" number of parameter is wrong.");
      printf("\n usage: number display");
      getch();
      exit(0);
   }
   if(argc == 3 && !strcmp(argv[2], "display"))
     disp=1 ;
   else
     disp=0 ;
   for(count = atoi(argv[1]); count ; --count)
     if(disp)
   printf("\n %d ", count) ;
   printf("%c", 7);
   getch();
   return 0;
}
```

خروجی

```
C:\BC> 6-26 5 display
5        نام برنامه   آرگومان‌ها
4
3
2
1
```

در مورد مثال ۱۹-۶ باید دو مطلب زیر را به خاطر داشت:

۱. تابع atoi() یکی از توابع کتابخانه‌ای است که رشتهٔ عددی را به مقدار عددی صحیح تبدیل می‌کند. به کار گرفتن این تابع به‌این دلیل بود که عدد وارد شده به عنوان آرگومان تابع، که شمارش معکوس آن باید انجام شود، به صورت رشته‌ای به تابع اصلی منتقل خواهد شد. برای استفاده از آن، باید به صورت عددی تبدیل شود. این تابع در فایل stdlib.h قرار دارد.

۲. آخرین دستور ()printf پس از عمل شمارش معکوس، جهت به صدا درآوردن زنگ سیستم به کار گرفته شده است.

اگر آرایه argv با دو اندیس به کار گرفته شود، موجب دسترسی به هر یک از کاراکترهای آرگومان تابع (به طور جداگانه) می‌گردد (مثال ۲۰-۶).

مثال ۲۰-۶

برنامه‌ای که چگونگی دسترسی به هر یک از کاراکترهای آرگومان تابع اصلی را نشان می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
int main(int argc, char *argv[])
{
  int t , i ;
  clrscr();
  if(argc < 2) {
     printf(" number of parameters is wong.");
     printf("\n usage : PROG arg.");
     exit(0) ;
  }
  for(t = 0; t < argc ; ++t)
    for(i = 0; argv[t][i]; i ++)
       printf("\n %c ",argv[t][i]) ;
   getch();
   return 0;
}
```

در مثال ۲۰-۶اولین اندیس آرایه []argv، یعنی t به آرگومان تابع اصلی (مشخص‌کننده شمارهٔ آرگومان) که یک رشته است اشاره می‌کند، و دومین اندیس این آرایه به کاراکتری از این آرگومان اشاره می‌کند، که شمارهٔ آن کاراکتر با این اندیس مشخص می‌شود. به عنوان مثال، منظور از argv[2][3]، کاراکتر سوم از آرگومان دوم می‌باشد. از نظر تئوری می‌توان حداکثر از ۳۲۷۶۷ عدد آرگومان استفاده نمود که معمولاً در عمل، سیستم عامل کمتر از این تعداد را اجازه می‌دهد.

تمرینات

۱. دستوراتی بنویسید که:

- متغیر fptr را اشاره‌گری از نوع float و متغیر num1 و num2 را از نوع float تعریف کرده، مقدار num1 را 7.3 تعیین کند.
- آدرس num1 را در fptr قرار دهد.
- محتویات محلی را که fptr به آن اشاره می‌کند در num2 قرار دهد.

- مقدار num2 را چاپ کند.
- آدرس num1 را با فرمت %p چاپ کند.
- آدرس موجود در fptr را چاپ کند.

۲. با توجه به دستورات زیر :

```
int  *zptr
int  *aptr = NULL ;
void *sptr = NULL ;
int  number, i ;
int  z [5] = {1, 2, 3, 4, 5}
sptr = z ;
```

اشکالات مربوط به دستورات زیر را مشخص کنید :

```
a) ++zptr ;
b) number = zptr ;
c) number = *zptr [2] ;
d) for (i = 0 ; i <= 5 ; i ++)
       printf ("%d", zptr [i]) ;
```

۳. برنامه زیر چه عملی انجام می‌دهد ؟

```
#include     <stdio.h>
Int      mystery2 (const  char  *) ;
int      main()
{
     char      string [80] ;
     printf("\nEnter a string: ") ;
     scanf("%s",string) ;
     printf("%d\n",mystery2(string)) ;
     return 0 ;
}
 //**************************
Int      mystery2 (const     char    *)
{
      int     x = 0 ;
      for (; *s != '\0', s ++)
              ++ x ;
      return    x ;
}
```

۴. اشکالات هر یک از بخشهای زیر را بیابید :

```
a) int   *num ;
   printf("%d", *num) ;

b) short    *numptr, result ;
   void     *genericptr = numptr ;
   result = *genericptr +7 ;
```

۵. برنامه‌ای بنویسید که نام و شماره تلفن تعدادی از مشتریان مخابرات را از ورودی خوانده، در آرایه‌هایی ذخیره نماید. شماره تلفنها در آرایه عددی و نام مشتریان در آرایه‌ای از اشاره‌گرهای رشته‌ای ذخیره شوند. سپس نامی از ورودی خوانده شده، شماره تلفن وی را مشخص کرده، در خروجی چاپ کند. برنامه باید برای ادامه کار، از کاربر سئوال کند. اگر کاربر جواب منفی داد، برنامه خاتمه پیدا می‌کند. توابعی برای خواندن اطلاعات، جستجو و چاپ نتایج جستجو بنویسید.

۶. برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده، به تابعی ارسال کند و تابع آن را به‌طور معکوس به خروجی ببرد.

۷. برنامه‌ای بنویسید که سه مقدار عددی را به عنوان آرگومان پذیرفته، به تابعی ارسال کند و تابع بزرگترین مقدار آنها را پیدا کند. برنامه باید تعداد آرگومان‌ها را کنترل کند.

۸. برنامه‌ای بنویسید که رشته عددی را که حاوی نقطه اعشار است از ورودی خوانده، آن را به عدد اعشاری تبدیل کند. به عنوان مثال، رشتهٔ "123.42" را به عدد 123.42 تبدیل نماید. تابعی برای خواندن رشته، تابعی برای تبدیل و تابعی برای نوشتن عدد در خروجی بنویسید. پارامترها از طریق فراخوانی باارجاع به توابع ارسال شوند.

۹. برنامه‌ای بنویسید که رشته‌هایی را از ورودی خوانده، فقط آن رشته‌هایی را که با حرف 'b' شروع می‌شوند در خروجی چاپ کند و به جای آخرین رشته، فقط کلید Enter را فشار دهید.

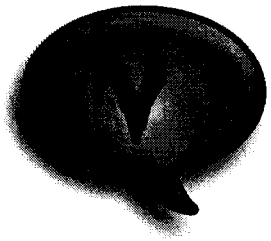
۱۰. برنامه‌ای بنویسید که دو رشته را از ورودی خوانده، یکی را در دیگری کپی کند.

۱۱. تابعی بنویسید که یک رشته و یک مقدار عددی را به عنوان آرگومان پذیرفته، تعدادی از کاراکترهای این رشته را که با این عدد مشخص می‌شود در رشته دیگری قرار داده، برگرداند. سپس برنامه‌ای بنویسید که از آن استفاده کند.

۱۲. خروجی برنامه زیر را تعیین کنید. آرایه x از آدرس ۱۰۲۴ شروع می‌شود.

```
#include <stdio.h>
#define M 5
int main()
{
   int i , j, x[M] ;
   for (i = 0 ; i < M ; i ++)
        x [i] = x + i ;
   for (i = 0 ; i < M ; i ++)
        printf ("\n x[%d] = %d" , i , * (x + i)) ;
   getch () ;
   return 0 ;
}
```

۱۳. تابعی بنویسید که یک اشاره‌گر از نوع صحیح و عدد صحیح n را به عنوان پارامتر پذیرفته، حافظه‌ای برای n عدد صحیح تخصیص دهد و آدرس آن را به برنامه برگرداند.



ساختمان‌ها

تاکنون مشاهده کردیم که چگونه یک متغیر، یک قلم از داده را در خود نگهداری می‌کند و همچنین، دیدیم که چگونه می‌توان چند عنصر همنوع را در یک آرایه ذخیره کرد. گاهی لازم است، چند عنصر غیر همنوع را تحت یک نام در حافظه ذخیره نمود. ولی با مطالبی که تاکنون گفته شد انجام این کار ممکن نیست. به‌عنوان مثال، اگر بخواهیم اطلاعات مربوط به کارکنان مؤسسه‌ای را که شامل نام کارمند (از نوع کاراکتری)، شماره کارمندی (از نوع عددی صحیح)، حقوق (از نوع عددی صحیح)، و غیره است، تحت یک نام ذخیره کنیم، در این صورت متغیر معمولی و آرایه جوابگوی نیاز ما نیستند. برای حل اینگونه مسئله‌ها در زبان C از نوعی داده به نام **ساختمان**[1] استفاده می‌شود. بنابراین می‌توان گفت که ساختمان، در زبان C نامی برای مجموعه‌ای از متغیرها است که این متغیرها می‌توانند همنوع نباشند.

تعریف نوع ساختمان

با توضیحاتی که داده شد، استفاده از ساختمان‌ها ضروری به نظر می‌رسد. نوعی به نام ساختمان در C وجود ندارد، بلکه تمام عناصر ساختمان از انواع موجود ایجاد می‌شوند. بنابراین، قبل از استفاده از ساختمان باید نوع ساختمانی را در برنامه ایجاد و سپس متغیرهایی از آن نوع را تعریف کرد و از آنها استفاده نمود. به عنوان مثال، نوع int در C وجود دارد و برای استفاده از آن نوع، باید متغیرهایی را از نوع int تعریف کرد و از آن متغیرها استفاده نمود. برای تعریف نوع ساختمان، از دستور struct به صورت زیر استفاده می‌شود:

```
struct <ساختمان> {
        عناصر ساختمان
} ;
```

نام ساختمان، از قانون نامگذاری برای متغیرها تبعیت می‌کند. عناصر ساختمان، متغیرهایی هستند که قسمتی از ساختمان می‌باشند و همانند یک متغیر معمولی و یا آرایه، باید اسم و نوع هر کدام مشخص باشد. به یک ساختمان نمونه که برای نگهداری اسم، شماره کارمندی و حقوق کارمندان نوشته شده است توجه نمایید:

```
struct personel  {
        char name [31] ;
        int  persno ;
        int  salary ;
} ;
```

در تعریف نوع ساختمان باید موارد ذیل را در نظر داشت:

۱. کاراکتر } می‌تواند در همان خطی که دستور struct آمده است نباشد.

۲. دستور struct به ; ختم می‌شود.

---

1. structure

تعریف متغیر نوع ساختمان

با مطالبی که تاکنون گفته شد، روش تعریف نوع ساختمان مشخص گردید؛ ولی تعریف ساختمان برای استفاده از آن کافی نیست؛ بلکه پس از تعریف نوع ساختمان باید متغیرهایی از این نوع را تعریف کرده سپس این متغیرها را در برنامه به کار گرفت. برای روشن شدن موضوع، یادآوری می‌کنیم که int و float دو نوع تعریف شده برای کامپیوتر هستند که باید متغیرهایی از این نوع تعریف و سپس از آنها استفاده کرد. تعریف متغیرهایی از نوع ساختمان به دو روش امکان‌پذیر است:

روش اول: پس از تعریف نوع ساختمان، متغیرهایی از نوع ساختمان با دستور struct تعریف می‌شوند. در این روش از شکل کلی زیر استفاده می‌شود:

struct نوع ساختمان <اسامی متغیرهای ساختمان> ;

نوع ساختمان، اسمی است که قبلاً در دستور struct، ساختار آن مشخص شده است. <اسامی متغیرهای ساختمان>، متغیرهایی را مشخص می‌کند که باید از نوع ساختمانی که اسم آن در <نوع ساختمان> مشخص شده است، تعریف شوند. اگر تعداد این متغیرها بیش از یکی باشد با کاما از یکدیگر جدا می‌شوند. به عنوان مثال، دستور زیر دو متغیر ساختمان به اسامی p1 و p2 را تعریف می‌کند که از نوع ساختمان personel هستند.

```
struct  personel    p1, p2 ;
```

روش دوم: در حین تعریف نوع ساختمان (بلافاصله پس از تعریف نوع)، اسامی متغیرهای ساختمان مشخص می‌گردند. در این روش از شکل کلی زیر استفاده می‌شود:

```
struct اسم ساختمان {
        عناصر ساختمان
} اسامی متغیرهای ساختمان ;
```

در این روش، ذکر اسم ساختمان اجباری نیست. دستور زیر دو متغیر ساختمان به اسامی p1 و p2 را تعریف می‌کند که ساختار آنها توسط دستور struct مشخص شده است.

```
struct  {
        char name [31] ;
        int  persno ;
        int  salary ;
} p1, p2 ;
```

دسترسی به عناصر ساختمان

همانطور که مشاهده شد هر ساختمان دارای چند عنصر است که برای دسترسی به هر یک از عناصر آن، از روش کلی زیر استفاده می‌شود:

<اسم متغیر از نوع ساختمان> . <نام عنصر>

به عنوان مثال، با فرض اینکه p1 و p2 دو متغیر از نوع ساختمان و دارای ۳ عنصر به اسامی name، person و salary باشند، دستورات زیر موجب دسترسی به هر یک از عناصر متغیر p1 می‌شوند. در مورد P2 نیز می‌توان به همین طریق عمل کرد:

```
p1.name
p1.person
p1.salary
```

اگر عناصر ساختمان از نوع آرایه باشند، ذکر اندیس آرایه جهت دسترسی به عناصر آن الزامی است. به عنوان مثال، دستور زیر را در نظر می‌گیریم :

```
struct     student {
               char name [31] ;
               int cours [10] ;
               int unit [10] ;
               int grade [10] ;
               int person ;
} st1, st2 ;
```

در این دستور، بعضی از عناصر ساختمان student به صورت آرایه تعریف شده‌اند. دستورات زیر موجب دسترسی به عناصر متغیر ساختمان st1 می‌شوند:

st1.cours [1]	اولین عنصر آرایه cours
st1.cours [i]	امین عنصر آرایه cours
st1.name[i]	امین عنصر آرایه name

همان‌طور که مشاهده شد، اگر عناصر ساختمان رشته‌ای باشند، می‌توان به تک‌تک عناصر آن نیز دسترسی پیدا کرد (مثل st1.name[i]).

برای مقداردادن به عناصر ساختمان می‌توان همانند یک متغیر معمولی عمل کرد؛ لذا با فرض این که st1 متغیر ساختمانی باشد که قبلاً تعریف شده است، دستورات زیر معتبر می‌باشند.

```
st1.persno = 1243 ;                    (مقدار دادن به عنصر ساختمان)
printf ("\n%d", st1.persno) ;          (چاپ کردن عنصر ساختمان)
gets (st1.name) ;                      (خواندن عنصر رشته‌ای ساختمان)
for (i=0 ; i < 10 ; i ++)              (خواندن عنصر آرایه‌ای ساختمان)
    scanf("%d", &st1.unit[i]) ;
```

مثال ۱-۷

برنامه‌ای که دو عدد موهومی را از ورودی خوانده مجموع آنها را محاسبه می‌کند و به خروجی می‌برد.

توضیح

عدد موهومی از دو بخش تشکیل شده است : بخش حقیقی و بخش غیرحقیقی. برای جمع دو عدد موهومی، بخشهای حقیقی با هم و بخشهای غیرحقیقی با هم جمع می‌شوند.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
  struct complex {
      int real ;
      int unreal ;
  } comp1 , comp2 ,result ;
  clrscr();
  printf("enter the real part for number %d:", 1);
  scanf("%d", &comp1.real) ;
  printf("enter the unreal part for number %d:", 1);
  scanf("%d", &comp1.unreal) ;
  printf("enter the real part for number %d:", 2);
  scanf("%d", &comp2.real) ;
  printf("enter the unreal part for number %d:", 2);
  scanf("%d", &comp2.unreal) ;
  result.real = comp1.real + comp2.real;
  result.unreal = comp1.unreal+comp2.unreal;
  printf("result is:real=%d,unreal=%d", result.real, result.unreal);
  getch();
  return 0;
}
```

خروجی

```
enter the real part for number 1 : 12
enter the real part for number 1 : 25
enter the real part for number 2 : 5
enter the real part for number 2 : 19
result is : real = 17 , unreal = 44
```

ارزش دهی اولیه به ساختمان

می‌توان به متغیرهای از نوع ساختمان، همانند آرایه‌ها مقدار اولیه داد. برای این منظور، مقادیر اولیه در بین } و { قرار گرفته، با کاما از یکدیگر جدا می‌شوند. مقادیری که به متغیر ساختمان نسبت داده می‌شوند به ترتیب در عناصر متغیر ساختمان قرار می‌گیرند: اولین مقدار به اولین عنصر، دومین مقدار به دومین عنصر و nامین مقدار به nامین عنصر نسبت داده می‌شود. اگر تعداد مقادیر، از تعداد عناصر ساختمان کمتر باشد، مقدار بقیه عناصر عددی برابر با صفر و مقدار بقیه عناصر رشته‌ای برابر با NULL (تهی) خواهد شد.

نوع ساختمان student را که در زیر آمده است در نظر بگیرید. این دستور علاوه بر تعریف نوع ساختمان، دو متغیر s1 و s2 را نیز از نوع student تعریف می‌کند.

```
struct    student {
          char name [21] ;
          char address [30] ;
          int grade [10] ;
          int stno ;
} s1, s2 ;
```

اکنون مجموعه دستورات زیر را ملاحظه نمایید:

```
struct  student st1 = {"ali" , "mashhad", {0} , 15}          (۱)
struct  student st2 = {0} ;                                  (۲)
struct  student st3 = {"ahmad", '\0' ,{15, 20}, 0};          (۳)
struct  student st4 = {"reza"} ;                             (۴)
```

در دستور ۱، مقدار "ali" در name، "mashhad" در address، صفر در کلیه عناصر آرایه grade و ۱۵ در stno از متغیر st1 قرار می‌گیرد. در دستور ۲، کلیه عناصر عددی متغیر st2، مقدار صفر و عناصر کاراکتری، مقدار تهی را می‌پذیرند. در دستور ۳، مقدار "ahmad" در name، رشته تهی در address، ۱۵ و ۲۰ به ترتیب در عناصر اول و دوم آرایهٔ grade و صفر در stno از متغیر st3 قرار می‌گیرد. در دستور ۴، مقدار "reza" در عنصر name از متغیر st4 قرار گرفته، بقیهٔ عناصر عددی این متغیر مقدار صفر و عناصر غیر عددی مقدار تهی را می‌پذیرند.

انتساب ساختمان‌ها به یکدیگر

در گونه‌های اولیهٔ زبان C، انتساب یک ساختمان به ساختمان دیگر امکان‌پذیر نبود، ولی در گونه‌های جدید C انتساب متغیرهای ساختمان که از یک نوع باشند امکان‌پذیر است. به عنوان مثال، اگر st1 و st2 متغیرهایی از نوع ساختمان student باشند، دستور زیر تمام عناصر st1 را به عناصر متناظرش در st2 نسبت می‌دهد.

```
st2 = st1 ;
```

مثال ۲-۷

برنامه‌ای که مشخصات چند دانشجو را به همراه معدل آنها از ورودی خوانده، سپس دانشجویی را که دومین معدل را از نظر بزرگی دارد پیدا می‌کند. هدف از این برنامه، آشنایی با انتساب ساختمان‌ها به یکدیگر است.

متغیرهای برنامه

متغیر	هدف
n	تعداد دانشجو
student	ساختمان دانشجو
st1, st2, st	متغیرهای ساختمان
y	متغیر کمکی
i	شمارنده حلقه

```
#include <stdio.h>
#include <conio.h>
struct student {
      char name[30] ;
      float grade ;
} ;
int main ()
{
   int i, n ;
   float y ;
   clrscr();
   struct student st1={0} ;
   struct student st, st2={0};
   printf("enter the number of student: ");
   scanf("%d", &n) ;
   for (i = 1; i <= n; i++) {
      printf("enter name of student %d :", i);
      scanf("%s", st.name) ;
      printf("enter grade of student %d :", i);
      scanf("%f", &y) ;
      st.grade = y;
      if (st.grade > st1.grade){
         st2 = st1 ;
         st1 = st ;
      }//enf of if
      else if (st.grade > st2.grade)
         st2=st ;
   }//end of for
   printf("the name of second student is :%s", st2.name) ;
   printf("\nthe grade of second student is:%.2f", st2.grade);
   getch();
   return 0;
}
```

خروجی

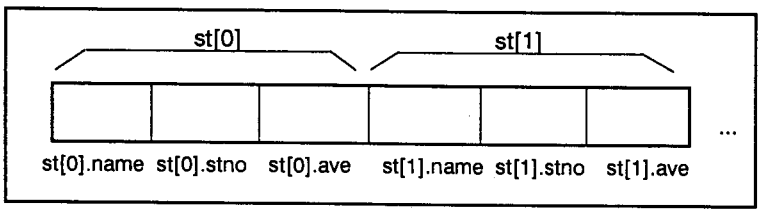
enter the number : 3
enter name of student 1 : ali
enter grade of student 1 : 12.6
enter name of student 2 : ahmad
enter grade of student 2 : 18.2
enter name of student 3 : reza
enter grade of student 3 : 14.5
the name of second student is : reza
the grade of second student is : 14.50

آرایه‌ای از ساختمان‌ها

یکی از بیشترین موارد کاربرد ساختمان‌ها، استفاده از آنها به عنوان عناصری از آرایه است. برای تعریف آرایه‌ای از ساختمان‌ها، ابتدا نوع ساختمان را تعریف کرده سپس همانند متغیرهای معمولی، آرایه‌ای از آن نوع را تعریف می‌کنیم. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
struct   student {
             char name [21] ;
             int stno ;
             float ave ;
} ;
struct   student , st[100] ;
```

در این دستورات، آرایه ۱۰۰ عنصری st طوری تعریف می‌شود که هر یک از عناصر آن، از نوع ساختمان student است. شکل زیر را ببینید:



مثال ۳-۷

برنامه‌ای که مشخصات تعدادی از افراد را که شامل نام و آدرس (شامل : خیابان، شهر و ...) است، از ورودی خوانده در آرایه‌ای از ساختمان‌ها قرار می‌دهد. این برنامه قادر است: فردی را از آرایه حذف کند، فرد جدیدی به آرایه اضافه نماید و محتویات کل آرایه را چاپ کند.

توضیح

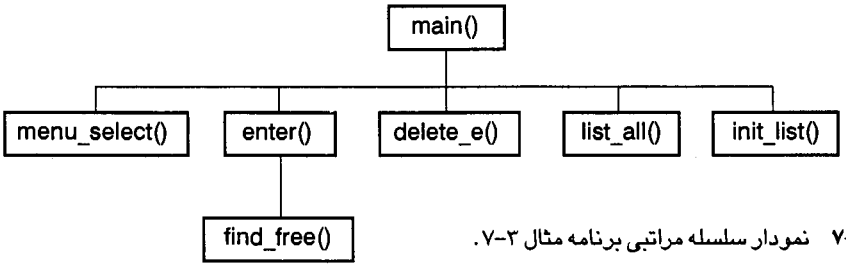
در برنامه‌هایی که چندین عمل انجام می‌شود، بهتر است هر عمل اصلی به‌عنوان گزینه‌های منو باشند تا کاربر با انتخاب آنها بتواند نیازهای خود را برآورده کند. **منو** (menu) ، لیستی از انتخاب‌ها است که در مقابل کاربر قرار می‌گیرد و کاربر می‌تواند برحسب نیاز، یکی از آنها را انتخاب کند تا کار خاصی صورت گیرد. هر یک از انتخابهای منو را **گزینه** می‌نامیم. به عنوان مثال در شکل ۱-۷ منویی با سه گزینه را مشاهده می‌کنید. هر کدام از این گزینه‌ها کار خاصی را انجام می‌دهند. پیامی که در زیر منو آمده است، به کاربر اعلام می‌کند که گزینه‌ای را انتخاب نماید. برای این کار باید یکی از شماره‌های ۱ تا ۳ را وارد کند. بعد از اینکه کاربر گزینه موردنظر را انتخاب کرد، برنامه باید براساس انتخاب کاربر، عملی را انجام دهد. در این برنامه از منو استفاده شده است. قبل از نوشتن این برنامه جهت درک صحیح آن، ساختار سلسله مراتبی (شکل ۲-۷) و وظایف هر یک از توابع را بیان می‌کنیم. در این برنامه از توابع کتابخانه‌ای gotoxy() ، atoi() و strtoul() استفاده شده است.

```
1. Enter date
2. Report the list
3. List
   Enter your delete (1-3) :
```

شکل ۱-۷
نمونه‌ای از منو با سه گزینه .

متغیرهای برنامه

تابع	متغیر	هدف
متغیرهای عمومی	MAX	تعیین کننده طول آرایه که مقدارش ۱۰۰ است
	addr	نوع ساختمان با عناصر: name برای نام شخص، street نام خیابان، city نام شهر، state نام استان و zip کدپستی است
	addr_info	آرایه‌ای از نوع addr
main()	choice	متغیری که شماره گزینهٔ انتخاب شده از منو را نگهداری می‌کند
init_list_all()	t	اندیس حلقه تکرار
menu_select()	s	متغیر کمکی برای دریافت اطلاعات از ورودی
	c	متغیری که شماره گزینه انتخاب شده از منو را نگهداری می‌کند
enter()	slot	اندیسی از آرایه که خالی است و باید عنصری در آن قرار گیرد
	s	متغیر کمکی برای دریافت اطلاعات از ورودی
find_free()	t	اندیس حلقه تکرار
delete_e()	slot	اندیس عنصری از آرایه که باید حذف شود
list_all()	t	اندیس حلقه تکرار
	r	شماره سطر جهت چاپ خروجی



شکل ۲-۷ نمودار سلسله مراتبی برنامه مثال ۳-۷.

شرح وظایف برنامه‌ها

تابع main(): تعریف ساختمان‌ها و فراخوانی توابع مطابق با شکل ۲-۷.

تابع menu_select(): ظاهر نمودن منویی در صفحه‌نمایش جهت انتخاب نوع عمل برنامه.

تابع enter(): فراخوانی تابع find_free() جهت پیداکردن جای خالی در آرایه، خواندن اطلاعات از صفحه کلید و واردکردن آنها در جای خالی در آرایه.

تابع delete_e(): حذف افراد از آرایه (برای این منظور اولین محل از عنصر name برابر با '\0' می‌شود).

تابع list_all(): نمایش محتویات کامل آرایه.

تابع init_list_all(): ارزش‌دهی به عناصر آرایه (اولین محل از عنصر name را برابر با '\0' قرار می‌دهد).

تابع find_free(): پیداکردن جای خالی در آرایه جهت قراردادن اطلاعات جدید در آن محل.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 100
struct  addr{
    char name[10] ;
    char street[15] ;
    char city[10] ;
    char state[3] ;
    unsigned long int zip;
} addr_info[MAX] ;
void init_list_all(void), enter(void) ;
void delete_e(void);
void list_all(void) ;
int menu_select(void), find_free(void);
int main()
{
   char choice , ch ;
   clrscr();
   init_list_all() ;
   for(;;){
       choice = menu_select() ;
       switch(choice) {
          case 1:  enter();  break ;
          case 2:  delete_e(); break ;
          case 3:  list_all();   break ;
          case 4:  exit(0) ;
       } //end of switch
   }//end of for
}
 //*****************************
void init_list_all(void)
{
     register int t ;
     for(t = 0 ; t < MAX; ++t)
         addr_info[t].name[0]='\0';
}
 //*****************************
int  menu_select(void)
{
      char s[10] ;
      int c ;
      clrscr() ;
      gotoxy(29, 6);
      printf("1) << enter a name >>");
      gotoxy(29, 8);
      printf("2) << delete_e a name >>");
```

```
        gotoxy(29, 10);
        printf("3) << list_all the file >>");
        gotoxy(29, 12);
        printf("4) <<       Quit      >>");
        do {
           gotoxy(24, 15) ;
           printf(" Please enter your choice(1-4):");
           gets(s) ;
           c = atoi(s) ;
        } while(c < 0 || c > 4) ;
        return (c);
}
 //*****************************
void enter(void)
{
  int slot ;
  char s[80] ;
  slot = find_free() ;
  if(slot == -1) {
        printf("\n list full");
        return ;
  }
  gotoxy(5, 17) ;
  printf("enter name:") ;
  gets(addr_info[slot].name);
  gotoxy(40, 17) ;
  printf("enter street:") ;
  gets(addr_info[slot].street);
  gotoxy(5, 19) ;
  printf("enter city:") ;
  gets(addr_info[slot].city) ;
  gotoxy(40, 19) ;
  printf("enter state code:") ;
  gets(addr_info[slot].state) ;
  gotoxy(22, 21) ;
  printf("enter zip:") ;
  gets(s) ;
  addr_info[slot].zip = strtoul(s,'\0',10) ;
}
 //*****************************
find_free(void)
{
      register int t ;
      for(t = 0 ; addr_info[t].name[0]
           && t < MAX ; ++t) ;
      if(t == MAX) return -1 ;
           return t ;
```

```
}
 //****************************
void delete_e(void)
{
      int slot ;
      gotoxy(28, 19) ;
      printf("enter record #(0-99):");
      scanf("%d",&slot) ;
      if(slot >= 0 && slot < MAX)
           addr_info[slot].name[0] = '\0';
}
 //****************************
void list_all(void)
{
     register  int t ;
     int r = 0 ;
     char ch ;
     clrscr() ;
     gotoxy(25, 2) ;
     printf(" << all information in list are:>>") ;
     gotoxy(13,3) ;
      printf("**********************");
      printf("********************") ;
      printf("********************") ;
     gotoxy(10, 4);
     printf("     name       street          city       ");
     printf("  state             zip");
     gotoxy(10, 5);
     printf("  ________    ________     _____      ") ;
     printf("  _____          ____");
     for(t = 0; t < MAX ; ++t) {
        if(addr_info[t].name[0]) {
            gotoxy(14, 6+r) ;
            printf("%s ",addr_info[t].name);
            gotoxy(26, 6+r) ;
            printf("%s ",addr_info[t].street);
            gotoxy(40, 6+r) ;
            printf("%s ",addr_info[t].city);
            gotoxy(54, 6+r) ;
            printf("%s ",addr_info[t].state);
            gotoxy(70, 6+r) ;
            printf("%lu ",addr_info[t].zip);
            r++ ;
        }
     }
     gotoxy(13, 6+r) ;
      printf("********************");
```

```
        printf("**********************");
        printf("**********************");
        gotoxy(27, 7+r) ;
        printf("press any key to continue");
        getch() ;
    }
```

عملکرد برنامه : پس از اجرای برنامه، منویی با ۴گزینه ایجاد می‌شود. گزینه اول مشخصات یک نفر را درخواست می‌کند، گزینه دوم فردی را با دریافت شماره رکورد آن حذف می‌کند، گزینه سوم محتویات لیست را نمایش می‌دهد و گزینه چهارم موجب خروج از برنامه می‌شود.

منوی برنامه

```
1. << enter a name  >>
2. << delete a name >>
3. << list the file     >>
4. << quit              >>
please enter your choice (1-4) : 1
enter name : mohammad
enter street : daneshgah
enter city : mashhad
enter state : 23
enter zip : 123
1. << enter a name  >>
2. << delete a name >>
3. << list the file     >>
4. << quit              >>
please enter your choice (1-4) : 1
enter name : naser
enter street : asrar
enter city : mashhad
enter state : 34
enter zip : 432
1. << enter a name  >>
2. << delete a name >>
3. << list the file     >>
4. << quit              >>
please enter your choice (1-4) : 3
*********************************************************************
  name              street            city          state          zip
---------------------------------------------------------------------
mohammad          daneshgah         mashhad       23             123
naser             asrar             mashhad       34             432
*********************************************************************
                    press any key to continue
```

تعریف ساختمان‌ها به صورت لانه‌ای

عناصر یک ساختمان می‌توانند یک متغیر معمولی، آرایه و یا یک ساختمان باشند. دو مورد اول را قبلاً مشاهده کردیم، دراین قسمت به بررسی مورد سوم می‌پردازیم. وقتی که عناصر یک ساختمان از نوع ساختمان باشند برای دسترسی به عناصر آن باید اسامی متغیرهای ساختمان را از بیرونی‌ترین ساختمان به داخلی‌ترین ساختمان ذکر کرد (به مثال ۴-۷ مراجعه شود).

مثال ۴-۷

برنامه‌ای که مشخصات چند کارمند را به همراه تاریخ استخدام آنها از ورودی خوانده، با اخذ تاریخ جاری، سابقه خدمت کارکنان را محاسبه می‌کند و مشخصات کارمندی که بیشترین سابقه خدمت را دارد به همراه متوسط سن کارمندان به خروجی منتقل می‌کند.

متغیرهای برنامه

نام	هدف
i	اندیس حلقه تکرار
n	تعداد کارکنان
sumage	مجموع سن کارکنان
maxold	بیشترین سابقه
employee	ساختمان با عناصر : name نام کارمند، persno شماره کارمندی، workdat تاریخ استخدام، age سن و old سابقه خدمت
emp	متغیری از نوع employee
date	ساختمانی با عناصر : day روز، month ماه و year سال

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 100
int main()
{
    int i, n, sumage = 0, maxold=0 ;
    struct date {
        int day ;
        int month ;
        int year ;
    } today ;
    struct employee {
        char name [30];
        int persno ;
        struct date workdate ;
        int age ;
        int old ;
```

```
    } emp[MAX] ;
    struct employee maxemp = {0} ;
    clrscr();
    printf(" enter the number of employees :") ;
    scanf("%d",&n) ;
    printf(" enter today date(day month year):") ;
    scanf("%d",&today.day) ;
    scanf("%d",&today.month) ;
    scanf("%d",&today.year) ;
    for (i = 0 ; i < n ; i++)  {
       printf(" enter name %d:", i + 1) ;
       scanf("%s", emp[i].name) ;
       printf(" enter personel number %d:", i + 1) ;
       scanf("%d",&emp[i].persno) ;
       printf(" enter work date(day month year) %d:", i + 1) ;
       scanf("%d",&emp[i].workdate.day);
       scanf("%d",&emp[i].workdate.month);
       scanf("%d",&emp[i].workdate.year);
       printf(" enter age for person %d:", i + 1);
       scanf("%d",&emp[i].age) ;
       if (today.month > emp[i].workdate.month ||
          (today.month == emp[i].workdate.month &&
           today.day >= emp[i].workdate.day ))
           emp[i].old = today.year - emp[i].workdate.year ;
       else
           emp[i].old = today.year - emp[i].workdate.year - 1 ;
       if (emp[i].old > maxold) {
           maxold = emp[i].old ;
           maxemp = emp[i] ;
       }//end of if
       sumage += emp[i].age ;
    }//end of for
    printf("\n the average of age is :%.2f", (float)sumage / n) ;
    printf("\n the specification of a older person:") ;
    printf("\n the name is : %s", maxemp.name) ;
    printf("\n the persno is: %d", maxemp.persno) ;
    printf("\n the age  is: %d", maxemp.age) ;
    printf("\n the old is: %d", maxemp.old) ;
    getch();
    return 0;
}
```

خروجی

```
enter the number of employees : 2
enter today date (day month year) : 11  3  78
enter name : ahmad
enter personal number : 1231
enter work date (day month year) : 12  2  65
enter age for person 1 : 30
enter name : mohammad
enter personal number : 4323
enter work date (day month year) : 2  4  62
enter age for person 2 : 35
the average of age is : 32.5
the specification of a older person :
the name is : mohammad
the persno is : 4323
the age is : 35
the old is : 15
```

ساختمان‌ها به عنوان آرگومان تابع

ساختمان‌ها و یا آرایه‌ای از ساختمان که تاکنون در برنامه‌ها به کار گرفته شده‌اند، یا به صورت عمومی تعریف شدند و یا در تابعی که مورد استفاده قرار می‌گرفتند، تعریف شدند. ساختمان‌ها و یا عناصر آنها می‌توانند به عنوان آرگومان، به تابع ارسال شوند.

انتقال عناصر ساختمان به توابع

برای انتقال عناصر ساختمان به تابع، همانند یک متغیر معمولی عمل می‌شود؛ با این تفاوت که نام متغیر ساختمان را باید به همراه عنصر موردنظر، در آرگومان تابع ذکر کرد. باید توجه داشت که پارامتر تابع نیز باید همنوع با آرگومان متناظر با آن تعریف گردد. به عنوان مثال، ساختمان زیر را در نظر بگیرید:

```
struct    student {
          char x ;
          int  stno ;
          char name [31] ;
} st1 ;
```

هر یک از دستورات زیر موجب انتقال مقادیر عناصر متغیر ساختمان st1 به تابعی به نام func() می‌شوند.

```
func (st1.x) ;
func (st1.stno) ;
func (st1.name [5]) ;
```

برای انتقال آدرسهای عناصر متغیر ساختمان به توابع، باید از عملگر & استفاده نمود، به مجموعهٔ دستورات زیر توجه نمایید:

```
func (&st1.x) ;
func (&st1.stno) ;
func (&st1.name [5]) ;
func (st1.name) ;
```

همانطور که در این مجموعه دستورات مشاهده می‌گردد، عملگر & باید قبل از نام متغیر ساختمان ظاهر گردد. ولی در آخرین دستور، چون name یک آرایه است، برای انتقال آن نیازی به عملگر & نیست.

مثال ۵-۷

برنامه‌ای که چگونگی استفاده از عناصر ساختمان‌ها را به عنوان آرگومان تابع نشان می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void test(char *);
int main()
{
    struct ss{
    char name[20] ;
    int x ;
    } s ;
    clrscr();
    printf("enter name and stno:");
    scanf("%s%d", s.name, &s.x) ;
    test(s.name) ;
    getch();
    return 0;
}
 //**********************
void test(char *s)
{
   printf("name is:");
   while(*s){
     printf("%c", *s) ;
     s++ ;
   }
}
```

خروجی

```
enter name and stno : ali  125
name is : ali
```

انتقال ساختمان‌ها به توابع

متغیرهای ساختمان را به دو روش می‌توان به توابع ارسال کرد. ۱. روش فراخوانی با مقدار ۲. روش فراخوانی باارجاع. این روشها در مورد متغیرهای معمولی توضیح داده شده‌اند. فعلاً به بررسی روش اول می‌پردازیم و روش دوم را پس از مطالعه اشاره‌گرهای ساختمان خواهید دید. به طور کلی باید توجه داشته باشید که برای استفاده از متغیرهای ساختمان در توابع، بهتر است نوع ساختمان در خارج از تابع ()main و به صورت عمومی تعریف شود و در داخل توابع، هر جا نیاز به متغیرهای ساختمان بود، از آن نوع برای تعریف آنها استفاده گردد.

مثال ۶-۷

برنامه‌ای که چگونگی انتقال کامل یک ساختمان را به تابع و انتقال اطلاعات از طریق ساختمان به تابع فراخواننده را نشان می‌دهد.

توضیح

چون تابع ()newname باید متغیری از نوع ساختمان personel را برگرداند، از نوع این ساختمان تعریف شده است. پارامتر تابع ()list یک متغیر ساختمان است. به الگوی این توابع و چگونگی تعریف آن توجه کنید.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct personel{
    char name[30] ;
    int agenum ;
};
struct personel newname();
void list(struct personel);
int main ()
{
     struct personel agent1 , agent2 ;
     struct personel newname() ;
     clrscr();
     agent1 = newname() ;
     agent2 = newname() ;
     printf("\n<< agent name   agent number\n") ;
     list(agent1) ;
     list(agent2) ;
     getch();
     return 0;
}
 //**********************
struct personel newname()
{
    char numstr[80] ;
```

```
        struct personel agent ;
        printf(" << New agent >>\n ");
        printf("Enter name:");
        gets(agent.name) ;
        printf("enter agent number(3 digit):");
        gets(numstr) ;
        agent.agenum = atoi(numstr) ;
        return(agent) ;
    }
    //**************************
    void list(struct personel age)
    {
        printf("%15s %10d\n", age.name, age.agenum);
    }
```

خروجی

```
        << new agent >>
enter name : reza
enter agent number (3 digit) 213
        << new agent >>
enter name : mohammad
enter agent number (3 digit) : 123
<< agent name      agent number >>
reza                    213
mohammad                123
```

اشاره‌گرهای ساختمان

در زبان C تعریف اشاره‌گر از نوع ساختمان، همانند تعریف سایر انواع اشاره‌گرها امکان‌پذیر است. اشاره‌گر ساختمان به دو منظور استفاده می‌شود:

۱. امکان فراخوانی به روش ارجاع را در توابعی که دارای آرگومان از نوع ساختمان هستند فراهم می‌کند.

۲. برای ایجاد لیست‌های پیوندی (linked list) و سایر ساختمان داده‌هایی که با تخصیص حافظه پویا سروکار دارند به کار می‌رود (این ساختمان داده‌ها در فصل ۱۰ بررسی می‌شوند).

وقتی که ساختمان‌ها از طریق فراخوانی به روش ارجاع به توابع منتقل می‌شوند، سرعت انجام عملیات بر روی آنها بیشتر می‌گردد؛ لذا در حین فراخوانی توابع، بهتر است به جای ساختمان، آدرس آن را منتقل نمود. عملگر & برای مشخص کردن آدرس ساختمان مورد استفاده قرار می‌گیرد.

تعریف اشاره‌گرهای ساختمان همانند تعریف متغیرهای ساختمان است؛ با این تفاوت که در جلوی اسم متغیر، علامت * قرار می‌گیرد. به عنوان مثال، دستورات زیر را در نظر بگیرید. در این دستورات، person یک متغیر ساختمان و p یک اشاره‌گر ساختمان تعریف شده است.

```
struct   bal   {
         float   balance ;
         char     name [80] ;
} person ;
struct  bal  *p ;
```

اکنون دستور زیر را در نظر بگیرید :

```
p = &person ;
```

با این دستور، آدرس متغیر ساختمان person در اشاره‌گر p قرار می‌گیرد. برای دسترسی به محتویات عناصر ساختمان از طریق اشاره‌گر، باید اشاره‌گر را در داخل پرانتز محصور نمود. به عنوان مثال دستور زیر، موجب دسترسی به عنصر balance از ساختمان person می‌شود. علت قراردادن متغیر اشاره‌گر در پرانتز، این‌است‌که تقدم عملگر نقطه (.) از عملگر * بالاتر است.

(*p).balance

به طور کلی برای دسترسی به عناصر ساختمانی که یک اشاره‌گر به آن اشاره می‌کند به دو روش می‌توان عمل کرد.

۱. ذکر نام اشاره‌گر در داخل پرانتز و سپس نام عنصر موردنظر که با نقطه از هم جدا می‌شوند (مثل دسترسی به عنصر balance از ساختمان person توسط اشاره‌گر p).

۲. استفاده از عملگر <-که روش مناسبتری است. اگر بخواهیم با استفاده از عملگر <- به عنصر balance از ساختمان person دسترسی داشته باشیم باید به طریق زیر عمل کنیم (علامت <- متشکل از علامت منها و علامت بزرگتر است):

p -> balance

مثال ۷-۷

برنامه‌ای که با استفاده از اشاره‌گرهای ساختمان، یک timer را شبیه‌سازی می‌کند.

توضیح

عناصر ساختمان tm عبارتند از: hours برای ساعت، minutes برای دقیقه و second برای ثانیه. تابع ()display زمان را نشان می‌دهد و تابع ()delay یک حلقه تأخیری دارد که تقریباً معادل یک ثانیه است.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct tm {
     int hours ;
     int minutes ;
     int second ;
};
void display(struct tm *);
```

```
void delayt ();
void update (struct tm *);
int main ()
{
     struct tm time ={0} ;
     clrscr();
     for (;;){
          update (&time) ;
          display (&time) ;
     }
}
//*********************
void display(struct tm *t)
{
    gotoxy(70, 2) ;
    printf("\n%2d:", t -> hours) ;
    printf("%2d:", t -> minutes) ;
    printf("%2d", t-> second) ;
}
//************************
void update (struct tm *t)
{
      t-> second ++ ;
      if (t-> second == 60)
      {
         t-> second=0 ;
         t-> minutes ++ ;
      }
      if (t-> minutes == 60)
      {
         t-> minutes=0 ;
         t-> hours ++ ;
      }
      if (t-> hours==24)
         t-> hours=0 ;
      delayt () ;
 }
//****************
void delayt ()
{
    long int t ;
    for (t = 1; t < 20000000 ; ++t) ;
}
```

مثال ۸-۷

برنامه‌ای که مشخصات مربوط به تعدادی از دانشجویان را خوانده در آرایه‌ای از ساختمان‌ها قرار می‌دهد. مشخصات دانشجو عبارتند از: ۱. نام دانشجو ۲. شماره دانشجویی ۳. تعداد درس ترم جاری ۴. نمره هر درس. این برنامه پس از خواندن اطلاعات دانشجو معدل آنها را نیز محاسبه می‌کند و در آرایه قرار می‌دهد. امکان حذف دانشجویی از آرایه، مشاهدهٔ اطلاعات یک یا چند دانشجو، پیداکردن دانشجویان مشروط (معدل کمتر از ۱۲) و دانشجویان ممتاز (معدل بالاتر از ۱۷) از جمله وظایف این برنامه است.

قبل از مشاهده لیست کامل برنامه، نمودار سلسله مراتبی آن را رسم کرده (شکل ۳-۷) و وظایف هر یک از توابع را تشریح می‌کنیم.

شرح وظایف برنامه‌ها

تابع main(): تعریف بعضی از متغیرها و فراخوانی توابع دیگر مطابق نمودار سلسله مراتبی شکل ۳-۷.

تابع init_list: اولین محل عنصر name را برابر با NULL قرار می‌دهد که به معنی خالی بودن آرایه است.

تابع enter(): ورود اطلاعات به آرایه. تابع find_free را فراخوانی می‌کند تا محل خالی در آرایه بیابد.

تابع del_rec(): این تابع برای حذف رکوردهایی از آرایه مورد استفاده قرار می‌گیرد. برای این منظور اولین محل name را برابر با NULL قرار می‌دهد.

تابع list(): انتقال کامل اطلاعات آرایه به خروجی، با استفاده از دو تابع title() و print().

تابع list1(): بررسی آرایه، جهت پیداکردن دانشجویان مشروط و انتقال اطلاعات این دانشجویان به خروجی، توسط دو تابع title() و print().

تابع list2(): بررسی آرایه، جهت پیداکردن دانشجویان ممتاز و انتقال اطلاعات آنها به خروجی، توسط دو تابع title() و print().

تابع menu_select(): ظاهر نمودن منویی در صفحه‌نمایش جهت درخواست انجام کار از برنامه.

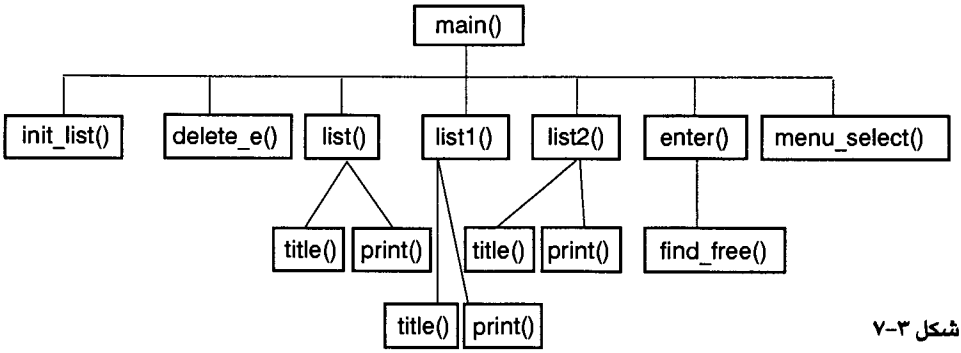
تابع find_free(): پیداکردن اولین محل خالی آرایه، جهت قراردادن اطلاعات جدید در آن.

تابع title(): چاپ عنوان برای خروجی. آرگومان آن، سطر مورد نظر را مشخص می‌کند.

تابع print(): چاپ اطلاعات موجود در آرایه در صفحه‌نمایش. این تابع دو آرگومان دارد، آرگومان r مشخص می‌کند که اطلاعات در چه سطری چاپ شوند و آرگومان t مشخص می‌کند که اطلاعات کدام دانشجو باید چاپ شود.

عناصر ساختمان student

name	نام دانشجو
mead	معدل
unit	واحد
number	تعداد درس
stno	شماره دانشجویی



شکل ۳-۷

نمودار سلسله مراتبی برنامه مثال ۹-۷.

متغیرهای برنامه

برنامه	متغیر	هدف
عمومی	MAX	ماکرویی که تعداد دانشجویان را مشخص می‌کند
	st_info	آرایه‌ای از ساختمان‌ها که مشخصات دانشجو را نگهداری می‌کند
main()	choice	شماره گزینه‌ای از منو که توسط کاربر انتخاب می‌شود
init_list()	t	شمارنده حلقه تکرار برای مقدار اولیه دادن به آرایه
menu_select()	s	رشته‌ای که انتخاب کاربر را نگهداری می‌کند
	c	مقدار عددی رشتهٔ s که به برنامه اصلی برگردانده می‌شود
enter()	grade	نمره درس هر دانشجو
	sumgrade	مجموع نمرات دانشجو
	slot	اندیس آرایه‌ای که خالی است و باید اطلاعات دانشجو در آن ذخیره شود
	j	شمارنده حلقه تکرار
	unit1	واحد هر درس دانشجو
	sumunit	مجموع واحدهای دانشجو
find_free()	t	شمارنده حلقه تکرار
del_rec()	slot	اندیس آرایه که دانشجوی موجود در آن باید حذف شود
list(),list1(),list2()	t	اندیس حلقه تکرار
	r	شماره سطری که اطلاعات باید در آنجا چاپ شوند
title(),print()	r	شماره سطری که اطلاعات باید چاپ شوند

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 100
struct  student {
    char    name[10] ;
    float   mead ;
    int     unit ;
    int     number ;
    int stno;
} st_info[MAX] ;
```

```
void init_list(void), enter(void);
void del_rec(void), list(void);
void list1(), list2();
void print(int *, int), title(int *);
int menu_select(void);
int find_free(void) ;
int main(void)
{
  char choice ;
  init_list() ;
  for(;;){
    choice = menu_select() ;
    switch(choice) {
      case 1: enter(); break;
      case 2: del_rec(); break ;
      case 3: list();  break;
      case 4: list1(); break ;
      case 5: list2();  break ;
      case 6: exit(0) ;
  } // end of switch
  }//end of for
}
//*********************
void init_list(void)
{
   register int t ;
   for(t = 0; t < MAX; ++t)
       st_info[t].name[0] = '\0' ;
}
//*********************
int menu_select(void)
{
  char s[10];
  int c ;
  clrscr() ;
  gotoxy(29, 3) ;
  printf("1) << enter a student    >>");
  gotoxy(29, 5) ;
  printf("2) << delete a student   >>");
  gotoxy(29, 7) ;
  printf("3) << list the array     >>");
  gotoxy(29, 9) ;
  printf("4) << list for probation >>");
```

```
    gotoxy(29, 11) ;
    printf("5) << list for exelent   >>");
    gotoxy(29, 13) ;
    printf("6) <<          Quit       >>");
    do {
       gotoxy(27, 15) ;
       printf(" Please enter your choice(1-6): ") ;
       gets(s);
       c = atoi(s);
       } while(c < 0 || c > 6) ;
    return (c) ;
}
 //*********************
void enter(void)
{
    float grade, sumgrade = 0;
    int slot, j, unit1, sumunit = 0 ;
    slot = find_free() ;
    if(slot == -1){
       printf("\n list full. press a key to continue...");
       getch();
       return ;
    }
    gotoxy(5, 17) ;
    printf("enter name:") ;
    gets(st_info[slot].name) ;
    gotoxy(5, 18) ;
    printf("enter stno:") ;
    scanf("%d", &st_info[slot].stno) ;
    gotoxy(5, 19) ;
    printf("enter number of course:") ;
    scanf("%d", &st_info[slot].number) ;
    for(j = 1; j <= st_info[slot].number; j++) {
        gotoxy(40, 17) ;
        printf("                          ");
        gotoxy(40, 17) ;
        printf("enter grade number %d:",j);
        scanf("%f", &grade) ;
        gotoxy(40, 19) ;
        printf("                          ");
        gotoxy(40,19) ;
        printf("enter unit of grade %d:", j);
        scanf("%d", &unit1) ;
```

```
        sumgrade += grade * unit1 ;
        sumunit += unit1 ;
    } //end of for
    st_info[slot].mead = sumgrade / sumunit ;
    st_info[slot].unit = sumunit;
}
//**********************
int find_free(void)
{
    register int t ;
    for(t = 0; st_info[t].name[0] && t < MAX ; ++t);
    if(t == MAX) return -1 ;
    return t ;
}
//**********************
void del_rec(void)
{
    int slot ;
    gotoxy(28, 19) ;
    printf("enter record #(0 - 99):") ;
    scanf("%d", &slot) ;
    if(slot >= 0 && slot < MAX)
        st_info[slot].name[0]='\0' ;
}
//**********************
void list(void)
{
    int t, r = 0;
    clrscr() ;
    title(&r) ;
    for(t = 0; t < MAX; ++t)
        if(st_info[t].name[0])
    print(&r, t);
    gotoxy(13, r) ;
     printf("*******************************");
     printf("*******************************") ;
    gotoxy(27, r + 1) ;
    printf("press any key to continue ") ;
    getch();
}
//**********************
void list1(void)
{
```

```
    int t, r = 0 ;
    clrscr() ;
    title(&r) ;
    for(t = 0; t < MAX; ++t){
      if(st_info[t].name[0] && st_info[t].mead < 12)
         print(&r, t) ;
    }//end of for
    gotoxy(13, r) ;
     printf("********************************");
     printf("********************************") ;
    gotoxy(27, r + 1) ;
    printf("press any key to continue ") ;
    getch();
  }
 //*********************
void list2(void)
{
    int t, r=0 ;
    clrscr() ;
    title(&r) ;
    for(t = 0; t < MAX; ++t){
      if(st_info[t].name[0] && st_info[t].mead >= 17)
         print(&r, t) ;
    }// end of for
    gotoxy(13, r) ;
     printf("********************************");
     printf("********************************") ;
    gotoxy(27, r + 1) ;
    printf("press any key to continue ") ;
    getch();
  }
 //*********************
void title(int *r)
{
    *r = 2;
    gotoxy(25, *r) ;
    printf(" <<   information in list are:  >>");
    gotoxy(13, *r + 1) ;
     printf("**************************");
      printf("**************************************");
    gotoxy(10, *r + 2);
    printf("     name        mead      ");
    printf("   unit          number              stno       ");
```

```
    gotoxy(10, *r + 3);
    printf("  _______     _______ ");
    printf("  _____       _____       ____   ");
    *r = 6;
}
//**********************
void print(int *r, int t)
{
    gotoxy(14,  *r) ;
    printf("%s ", st_info[t].name) ;
    gotoxy(26,  *r) ;
    printf("%.2f ", st_info[t].mead) ;
    gotoxy(40,  *r) ;
    printf("%d ", st_info[t].unit) ;
    gotoxy(54,  *r) ;
    printf("%d ", st_info[t].number) ;
    gotoxy(70,  *r) ;
    printf("%u ", st_info[t].stno) ;
    (*r) ++ ;
}
```

خروجی

```
1. << enter a student   >>
2. << delete a student  >>
3. << list the array    >>
4. << list for probation >>
5. << list for exelent  >>
6. << quit              >>
please enter your choice (1-6) :
```

عملکرد برنامه

پس از اجرای برنامه، منویی با ۶گزینه ظاهر می‌شود. گزینه اول اطلاعات دانشجو را می‌خواند، گزینه دوم دانشجویی را از آرایه حذف می‌کند، گزینه سوم لیست کاملی از آرایه را نمایش می‌دهد، گزینه چهارم، دانشجویان مشروط و گزینه پنجم دانشجویان باهوش را نمایش می‌دهد و گزینه ششم موجب خروج از برنامه می‌شود. با انتخاب هر کدام از گزینه‌ها می‌توان عمل موردنظر را انجام داد.

ساختمان بیتی

در زبان C، **به هر بیت از یک بایت** حافظه می‌توان دسترسی پیدا کرد. این امر در زبان C به دلایل زیر مفید است:

۱. اگر محدودیتی در میزان حافظه وجود داشته باشد، می‌توان از یک بایت به عنوان چند متغیر منطقی استفاده نمود. برای این منظور می‌توان هر بیت را به عنوان یک متغیر منطقی در نظر گرفت که مقدار صفر به معنی ارزش 'درستی' و مقدار یک به معنی ارزش 'نادرستی' و یا برعکس باشند.

۲. در حین ارتباط کامپیوتر با دستگاههای خارجی، بعضی از رابطها می‌توانند اطلاعات موجود در یک بایت را (که هر بیت آن ممکن است **معنی** خاصی داشته باشد) انتقال دهند.

۳. بسیاری از زیربرنامه‌های سیستم (که معمولاً از دید ما پنهان هستند) نیاز به دسترسی به بیتها دارند.

گرچه همهٔ این اعمال توسط عملگرهای بیتی قابل انجامند، ولی ساختمان‌های بیتی، روش مناسبتر و بهتری برای برآوردن این اهداف هستند. به طور کلی می‌توان گفت که **ساختمان بیتی**، مکانیزم ساختمانی جهت دسترسی به بیتهای یک بایت از حافظه است.

روش کلی تعریف ساختمان بیتی به صورت زیر است:

```
struct    نام ساختمان بیتی  {
            <نوع فیلد ۱>  <نام فیلد ۱>  : <طول فیلد ۱>  ;
            <نوع فیلد ۲>  <نام فیلد ۲>  : <طول فیلد ۲>  ;
                .
                .
                .
            <نوع فیلد n>  <نام فیلد n>  : <طول فیلد n>  ;
}    اسامی متغیرهای بیتی  ;
```

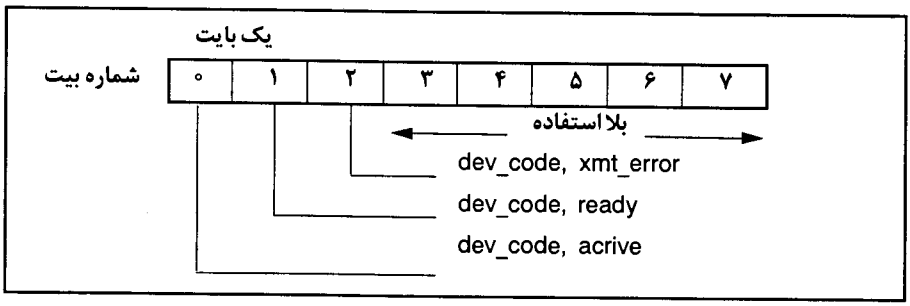
نام ساختمان بیتی، از قانون نامگذاری برای متغیرها تبعیت می‌کند. هر یک از فیلدها در ساختمان بیتی می‌توانند از نوع int، unsigned و یا signed باشند؛ فیلدی که طول آن ۱ باشد باید از نوع unsigned انتخاب گردد؛ زیرا فیلد یک بیتی نمی‌تواند شامل علامت هم باشد. طول فیلدها در ساختمان بیتی به بیت سنجیده می‌شود؛ یعنی طول فیلد مشخص می‌کند که فیلد موردنظر چند بیتی است. به عنوان مثال، ساختمان بیتی زیر را در نظر بگیرید:

```
struct   device {
           unsigned active : 1 ;
           unsigned ready : 1 ;
           unsigned xmt_error : 1 ;
}  dev_code ;
```

ساختمان device، ۳ فیلد را که طول هر کدام یک بیت است تعریف می‌کند و متغیر dev_code از نوع ساختمان بیتی device تعریف شده است. متغیر dev_code مانند شکل ۴-۷ در حافظه قرار می‌گیرد.

همانطور که در شکل ۴-۷ مشاهده می‌گردد، برای دسترسی به عناصر متغیر ساختمان بیتی، از عملگر نقطه (.) استفاده می‌گردد؛ اگر اشاره‌گری به ساختمان بیتی اشاره نماید، همانند ساختمان معمولی می‌توان با استفاده از عملگر <- به عناصر این ساختمان دسترسی پیدا کرد.

یکی از موارد کاربرد ساختمان‌های بیتی، در تجزیه و تحلیل اطلاعات اخذ شده از یک دستگاه سخت‌افزاری است. به عنوان مثال، پورت وضعیت یک **آداپتور ارتباط سری**، یک بایت وضعیت با ساختار جدول ۱-۷ را برمی‌گرداند.



شکل ۴-۷ وضعیت متغیر dev_code .

جدول ۱-۷ وضعیت آداپتور ارتباطی سری.

شماره بیت	مفهوم بیت وقتی که ۱ باشد
۰	change clear_to_send line
۱	change in data_set_ready
۲	trailing edge detected
۳	change in recive line
۴	clear_to_send
۵	data_set_ready
۶	telephone ringing
۷	received signal

بایت وضعیت مربوط به آداپتور ارتباط سری را می‌توان با استفاده از ساختمان بیتی تعریف کرد:

```
struct    status_type  {
          unsigned delta_cts : 1 ;
          unsigned delta_dsr : 1 ;
          unsigned tr_edge : 1 ;
          unsigned delta_rec : 1 ;
          unsigned cts : 1 ;
          unsigned dsr : 1 ;
          unsigned ring : 1 ;
          unsigned rec_line : 1 ;
} status ;
```

برای استفاده از ساختمان بیتی status، کافی است زیربرنامه‌ای نوشته شود تا اطلاعات فرستاده شده را بخواند و سپس آن را تجزیه و تحلیل و بررسی نماید:

```
status = get_port_status() ;
```

در دستور فوق، get_port_status() تابعی است که پورت وضعیت را خوانده در متغیر ساختمان بیتی status قرار می‌دهد.

بعضی از نکاتی که در مورد متغیرهای از نوع ساختمان بیتی باید در نظر داشت عبارتند از :

۱. نمی‌توان به آدرس آنها مراجعه کرد.

۲. نمی‌توانند به صورت آرایه تعریف شوند.

۳. به حد زیادی، وابسته به ماشین هستند، به عنوان مثال ممکن است در یک ماشین، بیتها از چپ به راست قابل دسترسی باشند و در ماشین دیگر از راست به چپ .

۴. ترکیبی از ساختمان بیتی و ساختمان معمولی، ممکن است :

```
struct    emp  {
          struct addr address ;
          float pay ;
          unsigned lay_off : 1 ;
          unsigned hourly : 1 ;
          unsigned shifted : 1 ;
} ;
```

در مورد ساختمان emp توجه داشته باشید که از یک بایت برای نگهداری ۳ قلم اطلاعات (عناصر lay_off ، hourly و shifted) استفاده شده است؛ اگر از ساختمان بیتی استفاده نمی‌شد حداقل به ۳ بایت نیاز بود.

مثال ۹-۷

برنامه‌ای که چگونگی استفاده از ساختمان‌های بیتی را نشان می‌دهد. در این برنامه، تاریخ تولد شما و سال جاری از ورودی خوانده شده، سن شما محاسبه می‌گردد. ۵ بیت برای روز، ۴ بیت برای ماه و ۷ بیت برای سال اختصاص یافت.

```
#include <string.h>
#include <conio.h>
#include <stdio.h>
void enter(char *, struct bdate *);
struct bdate  {
     unsigned int day   :5 ;
     unsigned int month :4 ;
     unsigned int year  :7 ;
};
int main ()
{
   struct bdate birth, today ;
   int age ;
   char p1[40] , p2[40] ;
   clrscr() ;
   strcpy(p1, "enter birth date (day month year):");
   strcpy(p2, "enter today date(day month year) :");
```

```
    enter(p1 , &birth) ;
    enter(p2 , &today) ;
    if (today.month > birth.month ||
       (today.month == birth.month &&
       today.day > birth.day))
       age = today.year - birth.year ;
    else
       age = today.year - birth.year -1;
    printf("\n the age is %d year.", age);
    getch();
    return 0;
}
//************************
void enter(char *prompt, struct bdate *point)
{
    unsigned int d, m, y;
    printf("%s", prompt);
    scanf("%d%d%d", &d, &m, &y);
    point -> day  = d ;
    point -> month = m ;
    point -> year = y ;
}
```

خروجی

```
enter birth date (day  month  year) : 12  11  50
enter today date (day  month  year) : 26  12  78
the age is 27 year.
```

یونیونها

در زبان C ، یونیون (union) محلی از حافظه است که توسط دو یا چند متغیر به طور اشتراکی مورد استفاده قرار می‌گیرد. این متغیرها به طور همزمان نمی‌توانند از این محل استفاده کنند، بلکه هر متغیر می‌تواند در زمانهای متفاوتی این محل را مورد استفاده قرار دهد. عناصر یونیون از یک محل حافظه شروع می‌شوند.

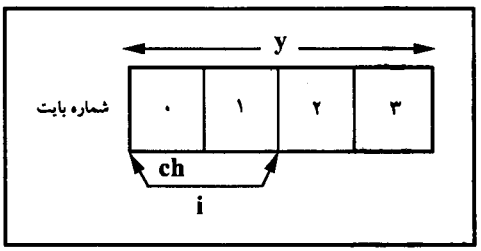
نحوه تعریف یونیون همانند ساختمان است و شکل کلی آن به صورت زیر است :

```
union  <اسم یونیون>  {
       عناصر یونیون
}  اسامی متغیرها  ;
```

اسم یونیون از قانون نامگذاری برای متغیرها تبعیت می‌کند. عناصر یونیون همانند عناصر ساختمان تعریف می‌شوند و چگونگی تعریف متغیرهای یونیون نیز مثل تعریف متغیرهای ساختمان است. طول یک یونیون به اندازهٔ مجموع طول عناصر آن نیست. بلکه به اندازهٔ طول عنصری است که بیشترین طول را دارد، زیرا در واقع یونیون ساختمانی است که آدرس شروع کلیه عناصر آن از یک نقطه است. یونیون زیر را در نظر بگیرید:

```
union   u_type {
          int  i ;
          char ch ;
          float y ;
} ;
```

در تعریف u_type ، سه عنصر به اسامی i ، ch و y تعریف شده‌اند. در این یونیون چون طول y از طول سایر اجزا بیشتر است (طول int برابر با ۲ بایت، طول char برابر با ۱ بایت و طول float برابر با ۴ بایت است)، طول u_type برابر با طول y، یعنی ۴ در نظر گرفته می‌شود. شکل ۵-۷ شیوه استفادهٔ عناصر i ، ch و y را از یک محل حافظه به طول ۴ بایت نشان می‌دهد.



شکل ۵-۷ نحوهٔ استفادهٔ عناصر یونیون از حافظه .

مثال ۱۰-۷

برنامه‌ای که کاربرد یونیون را تشریح می‌کند.

```
#include <conio.h>
#include <stdio.h>
int main ()
{
   union test {
     int i ;
     char ch ;
     float y ;
   } un;
   clrscr();
   printf("\n enter an integer number:");
   scanf("%d", &un.i) ;
   printf(" the value of i is :%d", un.i);
   printf("\n enter a character     :") ;
   un.ch = getche() ;
   printf("\n value of ch is:%c", un.ch);
   printf("\n enter a float number  :");
   scanf("%f", &un.y) ;
   printf(" the value of y is:%.2f", un.y);
   printf("\n the size of union is:%d", sizeof(union test));
   getch();
   return 0;
 }
```

خروجی

```
enter an integer number : 43
the value of I is : 43
enter a character : f
the value of ch is : f
enter a float number : 23.65
the value of y is : 23.65
the size of union is : 4
```

ساختمانی از یونیون

در ابتدای این فصل مشاهده شد که عناصر یک ساختمان می‌توانند، از نوع ساختمان باشند؛ به طور کلی به کارگیری یک ساختمان در ساختمان دیگر، یونیون در یک ساختمان و برعکس، و یونیون در یونیون دیگر امکان‌پذیر است.

مثال ۱۱-۷

برنامه‌ای که استفاده از ساختمان را به عنوان عنصری از یونیون نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
int main ()
{
   struct twoint {
      int num1 ;
      int num2 ;
   };
   union intflo {
     struct twoint  test ;
     float flonum ;
   } unex ;
   clrscr();
   printf(" size of union is : %d", sizeof(union intflo));
   unex.test.num1 = 564 ;
   unex.test.num2 = -231 ;
   printf("\n unex.test.num1=%d", unex.test.num1) ;
   printf("\n unex.test.num2=%d", unex.test.num2) ;
   unex.flonum = 45.76 ;
   printf("\n unex.flonum=%.2f", unex.flonum) ;
   getch();
   return 0;
}
```

خروجی

```
size of union is : 4
unex.test.num1 = 564
unex.test.num2 = -231
unex.flonum = 45.76
```

تغییر نام انواع داده‌ها با typedef

دستور typedef برای تعویض نام نوع موجود استفاده می‌شود؛ یعنی با استفاده از این دستور می‌توان برای انواع ابتدایی مثل int ، char و ... و همچنین انواع تعریف شده در برنامه، اسم دیگری انتخاب نمود. این دستور به دو دلیل اهمیت دارد:

۱. موجب می‌شود تا مسالهٔ قابل حمل بودن زبان C قوت بیشتری پیدا کند. یعنی اگر برنامه‌ای را در یک ماشین نوشته و بخواهیم آن را در ماشین دیگری اجرا کنیم، چنانچه این دو ماشین با یکدیگر مطابقت نداشته باشند، کافی است دستور typedef طوری عوض شود که این مشکل را حل نماید.

۲. موجب می‌شود تا برای انواع دادهٔ طولانی، اسم ساده‌تری انتخاب گردد.

دستور typedef به صورت زیر به کار می‌رود:

typedef <نوع موجود> <اسم جدید>

نوع موجود، یکی از انواع معتبر در زبان C است. اسم جدید، نامی است که نوع موجود در برنامه باید به این نام خوانده شود. دستور زیر را در نظر بگیرید:

```
unsigned int var1, var2 ;
```

این دستور موجب می‌شود تا متغیرهای var1 و var2 از نوع 'صحیح مثبت' انتخاب شوند. با استفاده از دستور typedef ، برای unsigned int به طریق زیر اسم جدیدی انتخاب می‌کنیم :

```
typedef unsigned int subint ;
```

این دستور موجب می‌شود تا بتوان به جای عبارت 'unsigned int' از subint استفاده کرد. دستور زیر را ملاحظه نمایید.

```
subint var1, var2 ;
```

با این دستور، متغیرهای var1 و var2 از نوع subint انتخاب می‌شوند، که همان unsigned int است.
دستور typedef در مورد آرایه‌ها کاربرد جالبی دارد. مجموعه‌دستورات زیر را در نظر بگیرید :

```
typedef char   string [50] ;
string  s1, s2 ;
```

این مجموعه دستورات موجب می‌شوند تا متغیرهای s1 و s2 از نوع کاراکتری و به طول ۵۰ بایت تعریف گردند. به عبارت دیگر، این مجموعه دستورات، با دستور زیر معادل است:

```
char  s1 [50], s2 [50] ;
```

مثال ۱۲-۷

برنامه‌ای که کاربرد دستور typedef را نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
int main ()
{
      const int n = 5;
      typedef unsigned int INT ;
      typedef int ARR[n] ;
      ARR list1, list2 ;
      INT i, sum = 0 ;
      clrscr();
      printf("\n enter %d integers: ", n);
      for (i = 0 ;i < n; i++)
          scanf("%d", &list1[i]) ;
      printf("\n enter %d integers agin: ", n);
      for (i = 0 ;i < n; i++)
          scanf("%d", &list2[i]) ;
      for (i = 0 ;i < n; i++)
          sum += list1[i] + list2[i] ;
      printf("\n sum of two array is:%d", sum) ;
      getch();
      return 0;
}
```

خروجی

```
enter 5 integers : 2  3  4  5  6
enter 5 integers again : 7  6  4  3  2
sum of two array is : 42
```

دستور typedef در مورد ساختمان‌ها نیز به کار رفته و موجب می‌گردد تا تعریف متغیرهای ساختمان و همچنین دسترسی به عناصر آن به سهولت انجام شود (مثال ۱۳-۷).

مثال ۱۳-۷

برنامه‌ای که کاربرد دستور typedef را در مورد ساختمان نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
struct date {
      int day ;
      int month ;
      int year ;
} ;
typedef struct date DATE ;
typedef DATE  *PTRDATE ;
int main() (
```

```
    DATE today ;
    PTRDATE ptr ;
    clrscr();
    ptr = &today ;
    ptr -> day   = 27 ;
    ptr -> month = 9 ;
    ptr -> year  = 1999 ;
    printf("\n date is:%d", ptr->day);
    printf("/%d", ptr -> month) ;
    printf("/%d", ptr -> year) ;
    getch();
    return 0;
}
```

خروجی

date is : 27/9/1999

انواع داده شمارشی

نوع داده شمارشی، این امکان را فراهم می‌کند که بتوان عناصر یک مجموعهٔ متناهی را نامگذاری یا شماره‌گذاری کرده سپس متغیرهایی را تعریف نمود تا مقادیر آن مجموعه را بپذیرند. این مجموعه با شناسه‌هایی (identifiers) معرفی می‌شود که به ثابتهای شمارشی معروفند. انواع شمارشی به صورت زیر تعریف می‌شوند :

```
enum    نام نوع شمارشی  {
                عنصر اول  ,
                عنصر دوم  ,
                عنصر سوم  ,
                .
                .
                .
            عنصر nام
} ;
```

نوع شمارشی همانند یک متغیر معمولی نامگذاری می‌شود؛ **عناصر نوع شمارشی**، شناسه‌هایی هستند که عناصر آن را مشخص کرده با کاما از یکدیگر جدا می‌شوند.

تعریف متغیرهای از نوع شمارشی همانند تعریف متغیرهای نوع ساختمان، به دو روش انجام می‌شود: ۱. هنگام تعریف نوع شمارشی و ۲. پس از تعریف نوع شمارشی. مجموعه دستورات زیر را در نظر بگیرید:

روش اول :

```
enum   color {
       red,
       yellow,
       brown
} color1, color2
```

روش دوم:

```
enum   color {
       red,
       yellow,
       brown
} ;
enum color color1, color2 ;
```

در روش اول ضمن تعریف نوع شمارشی color، دو متغیر color1 و color2 از این نوع تعیین می‌شوند و در روش دوم، پس از تعریف نوع شمارشی color، با استفاده از دستور enum دو متغیر color1 و color2 از این نوع تعیین می‌شوند. متغیرهای color1 و color2 فقط می‌توانند مقادیر red، yellow و brown را بپذیرند؛ مجموعه دستورات زیر در مورد این انواع شمارشی معتبرند:

```
color1 = red ;
color2 = yellow ;
if  (color1 == color2)
{
   ...
}
```

نکته‌ای که در مورد انواع شمارشی باید توجه داشت این است که به هر یک از عناصر نوع شمارشی یک مقدار عددی صحیح نسبت داده می‌شود: به اولین عنصر، مقدار صفر، به دومین عنصر، مقدار یک و به nامین عنصر، مقدار n-1 نسبت داده می‌شود؛ مگر اینکه برنامه‌نویس این مقادیر را تغییر دهد. به عنوان مثال، دستور زیر را در نظر بگیرید:

```
enum  sample {a, b = 5 , c, d} ;
```

در این دستور، مقدار صفر به a، مقدار ۵ به b، مقدار ۶ به c و مقدار ۷ به d نسبت داده می‌شود.

مثال ۱۴-۷

برنامه‌ای که با استفاده از انواع شمارشی، دستمزد هفته‌ای کارمند مؤسسه‌ای را محاسبه می‌کند.

توضیح

در این برنامه ایام هفته به صورت یک نوع شمارشی تعریف شده است. کارمند به ازای هر ساعت با نرخ ثابتی پول دریافت می‌کند. اگر روز پنج‌شنبه کار کند نرخ ثابت در ۱/۵ و اگر روز جمعه نیز کار کند نرخ ثابت در ۲/۰ ضرب می‌شود. در این برنامه، تابع tomorrow() روز بعد از روز جاری را مشخص می‌کند.

متغیرهای برنامه

برنامه	متغیر	هدف
main()	hourse	ساعت کار هر روز هفته
	r1	ضریب کاری پنج شنبه
	r2	ضریب کاری جمعه
	base	نرخ ثابت دستمزد
	rate	متغیر کمکی
	wages	دستمزد هفتگی
	weekday	نوع شمارشی
	day	متغیر نوع weekday
tomorrow()	d	روز فعلی
	nextd	روز بعدی (فردا)

```
#include <conio.h>
#include <stdio.h>
enum weekday { SAT, SUN , MON , TUE , WED , THU , FRI };
typedef enum weekday WEEKDAY;
WEEKDAY tomorrow(WEEKDAY);
int main ()
{
    int hours ;
    const float r1 = 1.2, r2 = 2.0;
    float base, rate, wages = 0.0 ;
    WEEKDAY day ;
    clrscr();
    printf("enter the basic hourly rate :") ;
    scanf("%f", &base) ;
    printf("enter the hours worked ");
    printf("for saturday through friday:\n");
    day = FRI ;
    getch();
    do {
      day = tomorrow(day) ;
      scanf("%d", &hours) ;
      switch(day) {
          case SAT :
          case SUN :
          case MON :
          case TUE :
          case WED :
          rate = base; break ;
          case THU :
          rate = r1 * base; break ;
          case FRI :
          rate = r2 * base; break ;
      }
      wages += rate * hours ;
    } while (day != FRI) ;
    printf("\n total wages for the week is :%.2f", wages) ;
    getch();
    return 0;
}
 //**********************
WEEKDAY tomorrow(WEEKDAY d)
{
    WEEKDAY nextd ;
```

```
        switch (d) {
           case SUN : nextd = MON ; break ;
           case MON : nextd = TUE ; break ;
           case TUE : nextd = WED ; break ;
           case WED : nextd = THU ; break ;
           case THU : nextd = FRI ; break ;
           case FRI : nextd = SAT ; break ;
           case SAT : nextd = SUN ; break ;
        }
        return(nextd) ;
    }
```

خروجی

```
enter the basic hourly rate : 120
enter the hourse worked
for saturday through friday : 7  6  9  12  5  7  5
total wages for the week is : 574.00
```

تمرینات

۱. ساختمانی تعریف کنید که اطلاعات مربوط به یک ماه از سال را نگهداری نماید. شامل نام ماه، تعداد روزهای ماه و شماره ترتیب ماه.

۲. آرایه‌ای از ساختمان‌ها تعریف کنید که براساس آنچه که در تمرین ۱ انجام دادید، اطلاعات کل ماههای سال را نگه دارد.

۳. با استفاده از تعاریفی که در تمرینهای ۲ و ۳ انجام دادید، تابعی بنویسید که شماره ترتیب یک ماه را به عنوان آرگومان پذیرفته، روزهای گذشته از ابتدای سال تا آن ماه را محاسبه کند.

۴. برای مشتریان سازمان تأمین اجتماعی، یک ساختمان به این صورت تعریف کنید: شماره بیمه شده، نام بیمه شده، ساختمان دیگری که شامل تاریخ بیمه و یا عناصر روز، ماه و سال باشد. برنامه‌ای بنویسید که اطلاعات مشتریان این سازمان را خوانده، در آرایه‌ای از ساختمان‌ها قرار دهد. سپس سال جاری را از ورودی خوانده، سابقهٔ بیمه تمام مشتریان را به همراه شماره بیمه، تاریخ بیمه و نام آنها به خروجی ببرد. تابعی برای خواندن اطلاعات و تابعی برای تهیه گزارش بنویسید.

۵. برنامه‌ای بنویسید که اعمال زیر را انجام دهد :

الف) ساختمانی به نام name با دو عنصر نام و نام خانوادگی دانشجو تعریف کنید.

ب) ساختمانی به نام student با این عناصر تعریف کنید: ساختمانی از نوع name ، آرایه‌ای سه عنصری برای نمره سه درس و عنصر بعدی میانگین این سه درس باشد.

ج) آرایه‌ای به نام st از نوع student و چهار عنصر بنویسید و هر نام را مقداردهی کنید.

د) تابعی بنویسید که اسامی موجود در این آرایه را یکی‌یکی در صفحه‌نمایش ظاهر کند و نمرات آنها را خوانده، در آرایه قرار دهد (میانگین نیز محاسبه شود و در محل مربوط قرار گیرد).

ه) تمام عناصر آرایه را به خروجی ببرد.

و) برنامه منوسازی شده باشد.

۶. فرودگاهی دارای ۲۰ باند پرواز است. برنامه‌ای برای تنظیم و رزرو فرود هواپیماها به شرح زیر بنویسید (پروازهای ورودی):

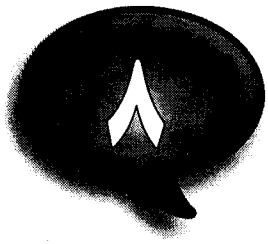
الف) آرایه‌ای ۲۰ عنصری از ساختمان تعریف کنید که عناصر ساختمان به این شرح باشند: شمارهٔ باند فرود، وضعیت باند (۱ = پر یا رزرو شده ، ۰ = خالی)، و نام کسی که باند را کنترل می‌کند.

ب) منویی با گزینه‌های زیر ظاهر کند:

ج) تمام عملیات موجود در منو را انجام دهد.

۱. نمایش تعداد باندهای خالی
۲. نمایش شماره باندهای خالی
۳. نمایش باندهای پر یا رزروشده
۴. تخصیص باند فرودگاه به هواپیما
۵. آزادکردن باند فرودگاه
۶. خروج
انتخاب خود را وارد کنید (۶-۱) :

۷. برنامه تمرین ۶ را طوری کامل کنید که بتواند پروازهای داخلی فرودگاه را نیز کنترل کند. این فرودگاه روزانه ۷ پرواز خروجی دارد (شماره‌ها را خودتان انتخاب کنید). برای این منظور، باید منوی دیگری قبل از منوی تمرین ۶ قرار گیرد که دارای دو گزینه است : ۱. پروازهای خروجی ۲. پروازهای ورودی. پس از انتخاب هر یک از این گزینه‌ها، منوی اصلی برنامه ظاهر شود. بدین‌ترتیب، برنامه باید بتواند اطلاعات پروازهای ورودی و خروجی را ذخیره کند.



فایل‌ها

داده‌های موردنیاز برنامه‌هایی که تاکنون نوشته شده‌اند، در متغیرهای معمولی، آرایه‌ها و ساختمان‌ها ذخیره و مورد پردازش قرار گرفته‌اند. متغیرهای معمولی، آرایه‌ها و ساختمان‌ها، همگی در حافظه RAM قرار دارند؛ لذا پس از قطع جریان برق (و یا خاموش شدن کامپیوتر) و یا خروج از برنامه، داده‌هایی که در آنها ذخیره شده‌اند از بین می‌روند و برای استفاده مجدد از آنها، باید دوباره وارد شوند که قطعاً این کار مقرون به صرفه نیست؛ زیرا نه تنها مستلزم صرف وقت زیادی است، بلکه حوصله انجام کار را نیز از کاربر سلب می‌نماید. برای رفع این مشکل، نوعی ساختمان دادهٔ دیگر به نام **فایل**[1] مورد استفاده قرار می‌گیرد. این نوع ساختمان داده، بر روی حافظه جانبی مثل دیسک، نوار و غیره تشکیل می‌شود. چون اطلاعات موجود در روی حافظه جانبی با قطع جریان برق، قطع اجرای برنامه و یا دلایلی از این قبیل از بین نمی‌روند، به دفعات زیادی مورد استفاده قرار می‌گیرند.

هر فایل شامل مجموعه‌ای از داده‌های مرتبط به هم است؛ مانند داده‌های مربوط به کلیه دانشجویان یک دانشگاه. داده‌های مربوط به هر یک از اجزای فایل، یک **رکورد**[2] نام دارد. به عنوان مثال، در یک دانشگاه داده‌های مربوط به هر دانشجو تشکیل یک رکورد را می‌دهند. لذا می‌توان گفت که هر فایل، مجموعه‌ای از چند رکورد است. اگر باز هم دقیق‌تر به فایل دانشجویان دانشگاه پرداخته شود، مشاهده می‌گردد که هر دانشجو ممکن است چند قلم داده داشته باشد؛ مثل نام دانشجو، تعداد واحدهایی که گذرانده، نمره هر درس و ... . به هر یک از اجزای یک رکورد، **فیلد**[3] گفته می‌شود، لذا می‌توان گفت که هر رکورد مجموعه‌ای از چند فیلد است.

در زبان C فایل داده می‌تواند هر دستگاهی مثل صفحه‌نمایش، صفحه‌کلید، چاپگر، ترمینال، دیسک، نوار و غیره باشد. داده‌ها ممکن‌است به ۴ روش در فایل ذخیره شده سپس بازیابی شوند:

۱. داده‌ها، کاراکتر به کاراکتر در فایل نوشته شده سپس کاراکتر به کاراکتر از فایل خوانده شوند.

۲. داده‌ها به صورت رشته‌ای از کاراکترها در فایل نوشته شده سپس به صورت رشته‌ای از کاراکترها دستیابی شوند.

۳. داده‌ها در حین نوشتن بر روی فایل، با فرمت خاصی نوشته شده سپس با همان فرمت خوانده شوند (عددی، کارکتری، رشته‌ای).

۴. داده‌ها به شکل ساختمان (رکورد) بر روی فایل نوشته شده سپس به صورت ساختمان از فایل خوانده شوند.

برای هر یک از موارد فوق توابع خاصی در زبان C منظور شده‌اند که در این فصل مورد بررسی قرار خواهند گرفت.

انواع فایل از نظر نوع اطلاعات

داده‌ها ممکن‌است در فایل به دو صورت ذخیره شوند: ۱. اسکی یا متن[4] ۲. باینری[5]. این دو روش ذخیره‌شدن داده‌ها، در موارد زیر با یکدیگر تفاوت دارد:

---

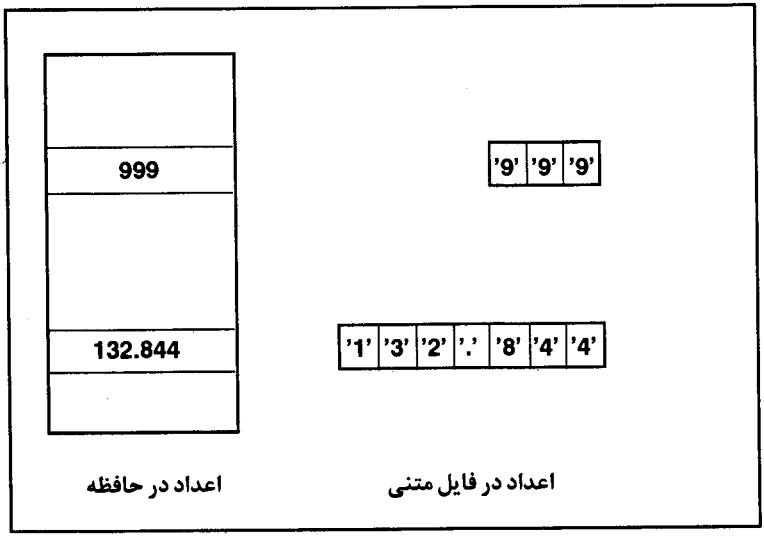
1. file 2. record 3. field 4. text 5. binary

۱. تعیین انتهای خط
۲. تعیین انتهای فایل
۳. نحوه ذخیره‌شدن اعداد بر روی دیسک

در فایل متنی، اعداد به صورت رشته‌ای از کاراکترها ذخیره می‌شوند ولی در فایل باینری اعداد به همان صورتی که در حافظه قرار می‌گیرند بر روی دیسک ذخیره می‌گردند. به عنوان مثال، در فایل متنی، عدد ۲۵۶ سه بایت را اشغال می‌کند. زیرا هر رقم آن، به صورت یک کاراکتر در نظر گرفته می‌شود. ولی در فایل باینری این عدد در ۲ بایت ذخیره می‌گردد (چون عدد ۲۵۶ یک عدد صحیح است و اعداد صحیح در دوبلیت ذخیره می‌شوند) (شکل ۱-۸).

در فایل متنی، کاراکتری که پایان خط را مشخص می‌کند، در حین ذخیره شدن بر روی دیسک، باید به کاراکترهای CR/LF تبدیل شود و در حین خوانده شدن، عکس این عمل باید صورت گیرد: یعنی کاراکترهای CR/LF باید به کاراکتر تعیین کننده پایان خط تبدیل شوند و بدیهی است که این تبدیلات مستلزم صرف وقت است، لذا دسترسی به اطلاعات موجود در فایل‌های متنی کندتر از فایل‌های باینری است.

اختلاف دیگر فایل‌های متنی و باینری در تشخیص انتهای فایل است. در هر دو روش ذخیرهٔ فایل‌ها، طول فایل توسط سیستم نگهداری می‌شود و انتهای فایل با توجه به طول فایل مشخص می‌گردد. در حالت متنی کاراکتر 1A (در مبنای ۱۶) و یا ۲۶ (در مبنای ۱۰) مشخص کنندهٔ انتهای فایل است (این کاراکتر با فشاردادن کلید CTRL به همراه کلید Z تولید می‌شود). در حین خواندن داده‌ها از روی فایل متنی، وقتی کنترل به این کاراکتر رسید بیانگر این است که داده‌های موجود در فایل تمام شده‌اند. در فایل باینری ممکن است عدد 1A (در مبنای ۱۶) و یا ۲۶ (در مبنای ۱۰) جزئی از اطلاعات بوده، بیانگر انتهای فایل نباشد. لذا نحوهٔ تشخیص انتهای فایل در فایل باینری با فایل متنی متفاوت است.



شکل ۱-۸ مقایسهٔ نحوهٔ ذخیره‌شدن اطلاعات در فایل متنی و حافظه.

1. Carriage Return Line Feed

سازمان فایل

منظور از سازمان فایل این است که اطلاعات در فایل چگونه ذخیره می‌شوند و سپس به چه روشهایی مورد بازیابی قرار می‌گیرند. به عبارت دیگر قانون حاکم بر نحوهٔ ذخیره و بازیابی داده‌ها را در فایل، **سازمان فایل** گویند. در این فصل به دو سازمان فایل پرداخته می‌شود:

۱. سازمان فایل ترتیبی (seqenital) ۲. سازمان فایل تصادفی (random)

در سازمان فایل ترتیبی، رکوردها به همان ترتیبی که از ورودی خوانده می‌شوند در فایل قرار می‌گیرند و در هنگام بازیابی، به همان ترتیبی که در فایل ذخیره شده‌اند مورد بررسی قرار می‌گیرند. به عنوان مثال اگر صدمین رکورد فایل بخواهد مورد دسترسی قرار گیرد، باید ۹۹ رکورد قبل از آن، از فایل خوانده شوند. فایل‌های ترتیبی معمولاً دارای یک فیلد کلید هستند(فیلد کلید، فیلدی است که به عنوان شاخص رکورد مورد استفاده قرار می‌گیرد) و براساس آن، مرتب می‌باشند (شکل ۲-۸). به عنوان مثال، در مورد دانشجویان، شماره دانشجویی و در مورد کارمندان شماره کارمندی، فیلد خوبی برای شاخص فرد می‌باشند. در سازمان فایل تصادفی، به هر رکورد یک شماره اختصاص می‌یابد؛ لذا اگر فایل دارای n رکورد باشد. رکوردها از ۱ تا n شماره‌گذاری خواهند شد. وقتی که رکوردی در فایلی با سازمان تصادفی قرار گرفت، محل آن توسط یک الگوریتم پیداکنندهٔ آدرس، که با فیلد کلید ارتباط دارد مشخص می‌شود. در این صورت دو رکورد با فیلد کلید مساوی، نمی‌توانند در فایل تصادفی وجود داشته باشند. در سازمان فایل تصادفی مستقیماً می‌توان به هر رکورد دلخواه دسترسی پیدا کرد (بدون اینکه رکوردهای قبل از آن خوانده شوند). شکل ۳-۸ نمونه‌ای از یک فایل تصادفی را نمایش می‌دهد.

شماره شناسایی	نام	موضوع درس	نمره
12	ALI	PASCAL	20
23	AHMAD	PASCAL	15
34	REZA	PASCAL	18
56	JAFAR	C	20

شکل ۲-۸ نمونه‌ای از یک فایل ترتیبی.

شماره رکورد	شماره شناسایی	نام	موضوع درس	نمره
13	12	ALI	PASCAL	20
20	56	JAFAR	C	20
24	23	AHMAD	PASCAL	15
31	34	REZA	PASCAL	18

شکل ۳-۸ نمونه‌ای از فایل تصادفی.

بازکردن فایل

هر فایل قبل از این که بتواند مورد استفاده قرار گیرد، باید باز شود. مواردی که در حین بازکردن فایل مشخص می‌شود عبارتند از:

۱. نام فایل
۲. نوع فایل از نظر ذخیرهٔ اطلاعات متنی یا باینری
۳. نوع فایل از نظر ورودی - خروجی (آیا فایل فقط به عنوان ورودی است، آیا فقط به عنوان خروجی است یا هم به عنوان ورودی است و هم به عنوان خروجی)

یک فایل ممکن است طوری باز شود که فقط عمل نوشتن اطلاعات بر روی آن مجاز باشد؛ به چنین فایلی، **فایل خروجی** گفته می‌شود. اگر فایل طوری باز گردد که فقط عمل خواندن اطلاعات از آن امکان‌پذیر باشد به چنین فایلی، **فایل ورودی** گفته می‌شود. اگر فایل طوری باز شود که هم عمل نوشتن اطلاعات بر روی آن مجاز باشد و هم عمل خواندن اطلاعات از آن، به چنین فایلی، **فایل ورودی - خروجی** گفته می‌شود. اگر فایلی قبلاً وجود داشته باشد و به عنوان خروجی باز گردد، اطلاعات قبلی آن از بین می‌رود. برای بازکردن فایل از تابع fopen() استفاده می‌گردد. این تابع که در فایل stdio.h قرار دارد به صورت زیر به کار می‌رود:

```
FILE *fopen (char  *filename , *mode)
```

در این الگو، filename به رشته‌ای اشاره می‌کند که حاوی نام فایل و محل تشکیل یا وجود آن است. **نام فایل داده**، از قانون نامگذاری فایل برنامه تبعیت می‌کند و شامل دو قسمتِ نام و پسوند است. بهتر است پسوند فایلِ داده، dat انتخاب گردد. محل تشکیل یا وجود فایل می‌تواند شامل نام درایو و یا هر مسیر موجود روی دیسک باشد. mode مشخص می‌کند که فایل چگونه باید باز شود (ورودی، خروجی و یا ورودی - خروجی). مقادیری که می‌توانند به جای mode در تابع fopen() قرار گیرند، همراه با مفاهیم آنها در جدول ۱-۸ آمده‌اند.

جدول ۱-۸ مقادیر معتبر mode در تابع fopen().

mode	مفهوم
r(rt)	فایلی از نوع text را به عنوان ورودی باز می‌کند
w(wt)	فایلی از نوع text را به عنوان خروجی باز می‌کنند
a(at)	فایل را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه نمود
rb	فایلی از نوع باینری را به عنوان ورودی باز می‌کند
wb	فایلی از نوع باینری را به عنوان خروجی باز می‌کند
ab	فایل موجود از نوع باینری را طوری باز می‌کند که بتوان اطلاعات را به انتهای آن اضافه نمود
r+(r+t)	فایل موجود از نوع text را به عنوان ورودی و خروجی باز می‌کند
w+(w+t)	فایلی از نوع text را به عنوان ورودی و خروجی باز می‌کند
a+(a+t)	فایل موجود از نوع text را به عنوان ورودی و خروجی باز می‌کند
r+b	فایل موجود از نوع باینری را به عنوان ورودی و خروجی باز می‌کند
w+b	فایل از نوع باینری را به عنوان ورودی و خروجی باز می‌کند
a+b	فایل از نوع باینری را به عنوان ورودی و خروجی باز می‌کند (این فایل می‌تواند قبلاً وجود داشته باشد)

برای بازکردن فایل باید یک اشاره‌گر از نوع فایل تعریف گردد تا به فایلی که توسط تابع fopen باز می‌شود اشاره نماید. اگر فایل به دلایلی باز نشود این اشاره‌گر برابر با NULL خواهد بود. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
FILE   *fp ;                          (۱)
fp = fopen ("A:test", "w") ;          (۲)
```

دستور ۱، متغیر fp را از نوع اشاره‌گر فایل تعریف می‌کند و دستور ۲، فایلی به نام test را بر روی درایو A ایجاد می‌نماید (زیرا حالت "w"، فایل را به صورت خروجی باز می‌کند). FILE ماکرویی در فایل stdio.h است.

برای تشخیص اینکه آیا فایل با موفقیت باز شده است یا خیر می‌توان اشاره‌گر فایل را با NULL مقایسه کرد. (NULL ماکرویی است که در فایل stdio.h تعریف شده است و با حروف بزرگ به کار می‌رود)، اگر اشاره‌گر فایل برابر با NULL باشد بدین معنی است که فایل باز نشده است:

```
if  ((fp = fopen ("A:test" , "w")) == NULL  {
    printf ("can not open file \n") ;
    exit (0) ;
}
```

بستن فایل

پس از این که برنامه‌نویس کارش را با فایل تمام کرد، باید آن را ببندد. بستن فایل توسط تابع ()fclose انجام می‌شود که دارای الگوی زیر است:

```
int  fclose (FILE  *fp)
```

در الگوی فوق، fp به فایلی اشاره می‌کند که باید توسط تابع ()fclose بسته شود. به عنوان مثال، دستور fclose(p) فایلی را که p به آن اشاره می‌کند، می‌بندد. اگر چندین فایل به طور همزمان در برنامه باز باشند می‌توان آنها را با تابع ()fcloseall بست. این تابع به صورت زیر به کار می‌رود:

```
fcloseall() ;
```

ورودی - خروجی کاراکترها

برای نوشتن یک کاراکتر در فایل، از توابع ()putc و ()fputc استفاده می‌شود. عملکرد این دو تابع یکسان است. تابع ()putc در گونه‌های جدید C و ()fputc در گونه‌های قدیمی C وجود داشته است. سرعت تابع ()putc از تابع ()fputc بیشتر است؛ لذا در اینجا از تابع ()putc استفاده می‌شود. الگوی این تابع به صورت زیر است:

```
int   putc (int ch, FILE *fp)
```

در الگوی فوق، ch کاراکتری است که باید در فایل نوشته شود و fp اشاره‌گری از نوع فایل است که مشخص می‌کند، کاراکتر موردنظر باید در چه فایلی نوشته شود.

برای خواندن کاراکترها از فایل، می‌توان از دو تابع ()getc و ()fgetc استفاده نمود. نحوه به کارگیری این دو تابع یکسان است. تابع ()fgetc در گونه‌های قدیمی C و ()getc در گونه‌های جدید C وجود دارد. سرعت اجرای

تابع ()getc از تابع ()fgetc بیشتر است؛ لذا در اینجا از تابع ()getc استفاده می‌شود. الگوی این تابع به صورت زیر است:

```
int  getc (FILE  *fp)
```

در الگوی فوق، fp اشاره‌گری است که مشخص می‌کند کاراکتر موردنظر از کدام فایل باید خوانده شود.
در مورد خواندن و نوشتن داده‌ها بر روی فایل باید به نکات زیر توجه داشت:

۱. وقتی کاراکترهایی بر روی فایل نوشته می‌شوند، باید مکان بعدی، که کاراکتر آتی در آنجا قرار می‌گیرد مشخص باشد؛ همچنین وقتی که کاراکترهایی از فایل خوانده می‌شوند باید مشخص باشد که تاکنون تا کجای فایل خوانده شده است و کاراکتر بعدی از کجا باید خوانده شود. برای این منظور، سیستم از یک متغیر به نام 'موقعیت‌سنج‌فایل' (file position) استفاده می‌کند که با هر دستور خواندن و یا نوشتن بر روی فایل، مقدار این متغیر به طور اتوماتیک تغییر می‌کند، تا موقعیت فعلی فایل را مشخص نماید. لذا عمل نوشتن بر روی فایل و عمل خواندن از روی آن، از جایی که این متغیر نشان می‌دهد، شروع می‌شود.

۲. در هنگام خواندن داده‌ها از فایل باید بتوان انتهای فایل را تست نمود؛ یعنی در برنامه باید بتوان این تست را انجام داد که اگر در حین خواندن داده‌ها از فایل، 'موقعیت‌سنج‌فایل' به انتهای فایل رسید دستور خواندن بعدی صادر نگردد؛ چراکه در غیر این صورت، سیستم، پیام خطایی را مبنی بر نبودن اطلاعات در فایل صادر می‌کند.

در حین خواندن داده‌ها از فایل متنی، پس از رسیدن به انتهای فایل، تابع ()getc یا ()fgetc علامت EOF را برمی‌گرداند. لذا در هنگام خواندن داده‌ها از فایل متنی می‌توان آنقدر به عمل خواندن ادامه داد، تا این که کاراکتر خوانده شده برابر با EOF شود. روش دیگر برای تست کردن انتهای فایل، استفاده از تابع ()feof است. الگوی این تابع به صورت زیر است:

```
int  feof (FILE  *fp)
```

در الگوی فوق، fp اشاره‌گری است که مشخص می‌کند این تابع باید بر روی چه فایلی عمل کند. تابع ()feof بر تشخیص انتهای فایل‌های باینری و متنی استفاده می‌شود. اگر اشاره‌گر فایلی که ()feof به آن اشاره می‌کند به انتهای فایل رسیده باشد، این تابع ارزش درستی وگرنه ارزش نادرستی را بر می‌گرداند.

مثال ۱-۸

برنامه‌ای که کاراکترهایی را از ورودی خوانده در یک فایل متنی قرار می‌دهد (آخرین کاراکتر ورودی $ است) و سپس داده‌های موجود در این فایل را خوانده، به فایل دیگری منتقل می‌کند. در این برنامه، ch کاراکتر ورودی، in و out اشاره‌گر فایل هستند.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
   FILE *in, *out ;
   char ch;
   clrscr();
```

```
    in = fopen("test.dat", "w");
    if (!in) {
       printf("cannot open file\n");
       getch();
       exit(1);
    }
    printf("\n enter characters($ for end):");
    do {
       ch = getchar();
       putc(ch, in);
    } while(ch != '$');
    fclose(in);
    out = fopen("output.dat", "w");
    if(!out) {
       printf("cannot open out file");
       getch();
       exit(1);
    }
    in = fopen("test.dat","r");
    if(!in) {
       printf("can not open input file");
       getch();
       exit(1);
    }
    ch = getc(in);
    while(ch != EOF) {
         putc(ch, out);
         ch = getc(in);
    }//end of while
    fclose(in);
    fclose(out);
    printf("\n file test.dat transfered to output.dat.");
    getch();
    return 0;
}
```

خروجی

```
this is a sample file .
this is a text file .
text file is slow .
end of file .
$
```

مثال ۲-۸

برنامه‌ای که برنامهٔ دیگری را به عنوان فایل ورودی می‌پذیرد و تعداد پرانتزهای باز و بسته و همچنین تعداد آکولادهای باز و بسته را شمارش می‌کند. نام فایل ورودی به عنوان آرگومان تابع اصلی، به برنامه وارد می‌شود.

توضیح

متغیر openbraket تعداد آکولاد باز، متغیر closebraket تعداد آکولادهای بسته، متغیر openparant تعداد پرانتزهای باز و متغیر closeparant تعداد پرانتزهای بسته را مشخص می‌کنند.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int  main(int argc, char *argv[])
{
    FILE *in ;
    char ch;
    int openbraket = 0, closebraket = 0;
    int openparant = 0, closparant = 0;
    clrscr();
    if(argc != 2) {
       printf("the number of arguments is incorrect.");
       getch();
       exit(1);
    }
    in = fopen(argv[1],"r");
    if (!in)  {
        printf(" file cannot open!.");
        getch();
        exit(0);
    }
    ch = getchar();
    while (!feof(in)) {
       if (ch == '{')
           openbraket ++ ;
       else if (ch == '}')
           closebraket ++;
       else if (ch == '(')
           openparant ++;
       else if (ch == ')')
           closparant ++;
       ch = getc(in);
    }
    printf(" the number of open braket is:%d", openbraket);
    printf("\n the number of close braket is:%d", closebraket);
    printf("\n the number of open parentheses is:%d", openparant);
    printf("\n the number of close parantheses is:%d", closparant);
    fclose(in);
    getch();
    return 0;
}
```

عملکرد برنامه

قبل از اجرای برنامه، در محیط بورلند C، در منوی RUN گزینه arguments را برگزینید و نام فایلی را که می‌خواهید برنامه بر روی آن عمل کند وارد کنید تا برنامه اجرا شود. نمونه‌ای از خروجی برنامه در زیر مشاهده می‌شود:

```
the number of open braket is : 37
the number of close braket is : 37
the number of open parantheses is : 177
the number of close parantheses is : 177
```

ورودی خروجی رشته‌ها

برای نوشتن رشته‌ها در فایل، از تابع ()fputs و برای خواندن رشته‌ها از فایل، از تابع ()fgets استفاده می‌گردد. الگوهای این دو تابع به صورت زیر می‌باشند:

```
int    fputs (const char   *str, FILE   *fp)
char  *fgets (char  *str , int  length , FILE  *fp)
```

در الگوی فوق، fp اشاره‌گری است که مشخص می‌کند این توابع باید بر روی چه فایل‌هایی عمل کنند. در تابع ()fgets اشاره‌گر str به رشته‌ای اشاره می‌کند که باید در فایل نوشته شود. این اشاره‌گر در تابع ()fputs به رشته‌ای که اطلاعاتِ خوانده شده از فایل در آن قرار می‌گیرند، اشاره می‌کند. length طول رشته‌ای را که باید از فایل خوانده شود مشخص می‌کند. نحوهٔ عمل تابع ()fgets به این صورت است که: از ابتدای فایل شروع به خواندن می‌کند تا به انتهای یک خط برسد و یا رشته‌ای به طول length کاراکتر را از فایل بخواند. برخلاف تابع ()gets، در تابع ()fgets کاراکتری که انتهای خط را مشخص می‌کند جزء رشته‌ای خواهد بود که این تابع از فایل می‌خواند. پس از مطالعه بخش بعدی، مثالی ارائه خواهد شد.

فایل به عنوان وسیله ورودی - خروجی

در مثال‌هایی که تاکنون مطرح شدند، فایل یا فقط به عنوان وسیله ورودی استفاده گردید و یا فقط به عنوان وسیله خروجی. در این قسمت مشاهده خواهد شد که چگونه می‌توان یک فایل را هم به عنوان وسیلهٔ ورودی و هم به عنوان وسیلهٔ خروجی مورد استفاده قرار داد؛ برای این منظور کافی است در تابع ()fopen به جای mode از یکی از عبارات r+ یا r+t (بازکردن فایل متنی موجود به عنوان ورودی و خروجی)، w+ یا w+t (ایجاد یک فایل متنی به عنوان ورودی و خروجی)، a+ یا a+t (ایجاد فایل متنی و یا بازکردن فایل متنی موجود به عنوان ورودی و خروجی)، r+b (بازکردن فایل باینری موجود، به عنوان ورودی و خروجی)، w+b (ایجاد یک فایل باینری به عنوان ورودی و خروجی) و یا a+b (ایجاد و یا بازکردن فایل موجود باینری به عنوان ورودی و خروجی) استفاده نمود. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
fp1 = fopen ("test.dat" , "w+b") ;        (۱)
fp2 = fopen ("sample.dat" , "r+b") ;      (۲)
fp3 = fopen ("test2.dat" , "a+t") ;       (۳)
```

دستور ۱، فایلی به نام test.dat را از نوع باینری و به صورت ورودی و خروجی باز می‌کند که اشاره‌گر fp1 به آن اشاره می‌کند. اگر فایل test.dat قبلاً وجود داشته باشد محتویات قبلی آن از بین خواهند رفت. دستور ۲ فایلی به نام sample.dat را که اکنون بر روی درایو جاری وجود دارد از نوع باینری و به صورت ورودی و خروجی باز می‌کند. اگر فایل sample.dat بر روی درایو جاری وجود نداشته باشد، فایل باز نخواهد شد. دستور ۳، فایلی به نام test2.dat را از نوع متنی و به صورت ورودی و خروجی باز می‌کند. اگر فایل test2.dat قبلاً وجود نداشته باشد، ایجاد خواهد شد و اگر وجود داشته باشد اطلاعات قبلی آن محفوظ بوده، اطلاعات جدید به انتهای آن اضافه خواهد شد.

در حین کار با فایل‌ها (نوشتن اطلاعات بر روی آنها و یا خواندن اطلاعات از آنها) برای برگشت به ابتدای فایل (تغییر 'موقعیت‌سنج‌فایل'، به‌طوری که به ابتدای فایل اشاره کند) باید فایل را بسته و مجدداً آن را باز نمود (البته با توجه به مطالبی که تاکنون در مورد فایل‌ها گفته شد). این مطلب در مثال‌هایی که تاکنون مطرح شد به چشم می‌خورد. اصولاً شاید در فایل‌هایی که فقط به عنوان خروجی و یا فقط به عنوان ورودی باز می‌شوند، نیاز به برگشت به ابتدای فایل (بدون بستن و بازکردن مجدد آن) احساس نشود، ولی این امر در مورد فایل‌های ورودی و خروجی به عنوان یک نیاز مطرح است. برای این منظور، از تابعی به نام ()rewind استفاده می‌گردد. الگوی تابع ()rewind در فایل stdio.h قرار داشته و به صورت زیر است:

```
void  rewind (FILE     *fp)
```

در الگوی فوق، fp به فایلی اشاره می‌کند که 'موقعیت‌سنج' آن باید به ابتدای فایل اشاره نماید.

مثال ۳-۸

برنامه‌ای که رشته‌هایی را از ورودی خوانده، در یک فایل متنی قرار می‌دهد و سپس محتویات این فایل را خوانده به صفحه‌نمایش منتقل می‌کند. از آنجایی که تابع ()gets کاراکتری که پایان خط را مشخص می‌کند به رشته اضافه نمی‌کند، در حین نوشتن بر روی فایل، این کاراکتر به رشتهٔ خوانده شده اضافه می‌شود. برای خاتمه برنامه کافی است به جای رشته، فقط کلید enter وارد شود.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
int main(void)  {
   FILE *fp ;
   char str[80];
   clrscr();
   if(!(fp = fopen("test","w+"))) {
      printf("cannot open file.") ;
      getch();
      exit(1);
   }
   printf("enter strings(Enter to quit):\n");
   while (1)  {
   gets(str);
```

```
        if(!str[0])
            break ;
        strcat(str,"\n");
        fputs(str, fp);
    }
    printf("the content of file is:\n\n");
      rewind(fp);
      fgets(str, 79, fp);
      while (!feof(fp)) {
          printf("%s", str);
          fgets(str, 79, fp) ;
      }
      fclose(fp);
      getch();
      return 0;
}
```

خروجی

(متن وارد شده از طریق صفحه کلید به فایل متنی)

```
enter a string (Enter to quit) :
<< top of file    >>
this is an example for
read/write file .
<< bottom of file >>

the content of file is :
<< top of file    >>
this is an example for
read/write file .
<<bottom of file >>
```

(متن موجود در فایل متنی که به خروجی منتقل شده است)

عیب‌یابی در ورودی - خروجی فایل

در حین انجام کار با فایل‌ها ممکن است خطایی رخ دهد. به عنوان مثال، عدم وجود فضای کافی برای ایجاد فایل، آماده نبودن دستگاهی که فایل باید در آنجا تشکیل گردد و یا مواردی از این قبیل، منجر به بروز خطا می‌شوند. با استفاده از تابع ferror() می‌توان از بروز چنین خطایی مطلع گردید. الگوی تابع ferror() در فایل stdio.h قرار داشته به صورت زیر است:

```
int  ferror (FILE  *fp)
```

در این الگو، fp اشاره‌گری است که مشخص می‌کند تابع باید بر روی چه فایلی عمل کند. این تابع یک تابع منطقی است؛ بدین معنی که اگر خطایی در رابطه با فایل‌ها رخ دهد، ارزش 'درستی' وگرنه ارزش 'نادرستی' را برمی‌گرداند. برای تشخیص خطا در کار با فایل‌ها، بلافاصله پس از هر عملی که روی فایل انجام می‌شود باید از این تابع استفاده نمود.

مثال ۴-۸

برنامه‌ای که کاراکترهای tab را از فایل حذف کرده به جای آن به تعداد کافی فضای خالی قرار می‌دهد. اسامی فایل‌های ورودی و خروجی از طریق آرگومان به برنامه وارد می‌شود.

متغیرهای برنامه

تابع	متغیر	هدف
عمومی	TAB_SIZE	تعیین اندازه حرکت TAB
	IN	کد خطا در فایل ورودی
	OUT	کد خطا در فایل خروجی
main()	in, out	اشاره‌گرهای فایل
	ch	کاراکتری که از فایل خوانده می‌شود
	i	شمارنده حلقه
	tab	متغیر کمکی
err()	IN	کد خطا در فایل ورودی

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#define  TAB_SIZE   8
#define  OUT  1
#define  IN   1
void err(int) ;
int main(int argc , char *argv[])
{
    FILE *in , *out;
    int tab, i ;
    char ch ;
    clrscr();
    if(argc != 3){
      printf("\n incorrect number of parameters") ;
      getch() ;
      exit(1) ;
    }
    in = fopen(argv[1] , "r");
    out = fopen(argv[2] , "wb") ;
    if (!out) {
       printf("\n cannot open output file.") ;
       getch() ;
       exit(1) ;
    }
    tab = 0;
    do {
       ch = getc(in) ;
       if (ferror(in))
```

```
            err(IN) ;
          if(ch == '\t') {
            for (i = tab; i < 8; i++)  {
              putc(' ' , out) ;
              if (ferror(out))
                err(OUT) ;
            } //end of for
            tab = 0;
          } //end of if
          else {
            putc(ch , out) ;
            if (ferror(out))
               err(OUT) ;
            tab ++ ;
            if (tab == TAB_SIZE || ch == '\n' || ch == '\r')
               tab = 0;
          }//end of else
       } while (!feof(in)) ;
       fcloseall() ;
       printf("Successfuly changed!");
       getch();
       return 0;
}
 //************************
void err(int eror)
{
     if (eror == IN)
       printf("\n error on input file. press a key ...");
     getch() ;
     exit(1);
}
```

حذف فایل

برای حذف فایل‌های غیر ضروری می‌توان از تابع ()remove استفاده کرد. الگوی این تابع در فایل stdio.h قرار داشته به صورت زیر است:

int remove(char *filename)

در الگوی فوق، filename به نام فایلی که باید حذف شود، اشاره می‌کند. اگر عمل تابع با موفقیت انجام شود مقدار صفر وگرنه مقداری غیر از صفر برگردانده خواهد شد.

مثال ۵-۸

برنامه‌ای که نام فایلی را به عنوان آرگومان پذیرفته، آن را حذف می‌کند.

توضیح

برنامه هنگام حذف نام فایلی که به عنوان آرگومان وارد شد، پیامی صادر می‌کند که آیا فایل حذف شود یا خیر. اگر 'y' وارد شود، فایل حذف خواهد شد. در این برنامه از تابع کتابخانه‌ای ()toupper برای تبدیل حرف کوچک به بزرگ استفاده شده است. الگوی این تابع در فایل ctype.h قرار دارد. عملکرد تابع ()remove با پیامی مشخص می‌شود (فایل حذف شده است یا خیر).

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
int main(int argc, char *argv[])
{
    char ans;
    clrscr();
    if(argc != 2) {
       printf(" you must type a file name!\n") ;
       getch();
       exit(1) ;
    } //end of if
    printf(" erase %s (y/n):", argv[1]);
    ans = getch();
    if(toupper(ans) == 'Y')
       if(remove(argv[1]))
          printf("\n can not erase file %s.", argv[1]);
       else
          printf("\n file successfully  erased.");
    getch();
    return 0;
}
```

بافر

بافر[1]، حافظه‌ای است که به فایل‌ها اختصاص می‌یابد تا اعمال ورودی - خروجی بر روی آنها با سرعت بیشتری انجام گیرد. در حین خروج از برنامه، خوب است جهت اطمینان بیشتر، به ماشین دستور داد تا کلیهٔ داده‌های موجود در بافرها را به فایل مربوطه منتقل نماید. برای این منظور می‌توان از تابع ()fflush استفاده نمود. اگر کار این تابع با موفقیت انجام شود، مقداری که توسط آن برگردانده می‌شود برابر با صفر وگرنه برابر با EOF خواهد بود (EOF ماکرویی است که در فایل stdio.h تعریف شده است). الگوی تابع ()fflush در فایل stdio.h قرار داشته و به صورت زیر است:

int fflush (FILE *fp)

1. buffer

در الگوی فوق، fp فایلی را مشخص می‌کند که تابع ()fflush باید بر روی آن عمل نماید. اگر fp ذکر نشود تابع ()fflush بر روی کلیه فایل‌هایی که به عنوان خروجی باز شده‌اند عمل می‌کند. به عنوان مثال، دستور fflush(f) کلیه بافرهای مربوط به فایلی با اشاره‌گر f را در این فایل می‌نویسد.

ورودی و خروجی همراه با فرمت

اگر لازم باشد که داده‌ها با فرمت خاصی در فایل نوشته و یا از آن خوانده شوند می‌توان از دو تابع ()fprintf و ()fscanf استفاده نمود. این دو تابع دقیقاً کار توابع ()printf و ()scanf را در ورودی - خروجی معمولی صفحه‌نمایش، انجام می‌دهند. الگوی این توابع در فایل stdio.h قرار دارد و به صورت زیر است:

```
int fprintf(FILE   *fp, "*control_string,..." , char arg, ...)
int fscanf(FILE    *fp,  "control_string,..." , char arg, ...)
```

در الگوهای فوق، fp اشاره‌گری است که مشخص می‌کند اعمال این توابع باید بر روی چه فایلی انجام شود؛ control_string مشخص می‌کند که داده‌ها (arg) باید با چه فرمتی نوشته و یا خوانده شوند.

مثال ۶-۸

برنامه‌ای که نام و شماره دانشجویی را از ورودی خوانده آن را در یک فایل می‌نویسد و سپس اطلاعات این فایل را خوانده در صفحه‌نمایش چاپ می‌کند.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
int main(void)
{
   FILE *fp;
   char name[80];
   int num;
   if((fp = fopen("test", "w")) == NULL) {
      printf("can not open file for write");
      getch();
      exit(1) ;
   }
   clrscr();
   printf(" enter the name :");
   gets(name);
   strcat(name, "\n");
   printf(" enter the number:") ;
   scanf("%d", &num);
   fprintf(fp,"%s%d", name, num) ;
   fclose(fp) ;
```

```
    if((fp = fopen("test", "r")) == NULL) {
        printf("can not open file for read.");
        exit(1);
    }
    fscanf(fp, "%s%d", &name, &num);
    printf("\n name = %s, number = %d", name, num);
    getch();
    return 0;
}
```

خروجی

```
enter the name : ali
enter the number : 1234
name = ali , number = 1234
```

در مورد توابع ()fprintf و ()fscanf باید توجه داشت که علیرغم اینکه ورودی خروجی با این دو تابع آسان است اما اطلاعات به همان صورتی که در صفحه‌نمایش ظاهر می‌شوند در فایل ذخیره می‌گردند. به عنوان مثال، عدد ۲۶۷ که ۳ بایت را در صفحه‌نمایش اشغال می‌کند، اگر توسط تابع ()fprintf بر روی فایل نوشته شود، در فایل نیز ۳ بایت را اشغال خواهد کرد (توجه داریم که عدد ۲۶۷ یک عدد صحیح است و می‌تواند در دو بایت ذخیره شود). یعنی هر رقم به صورت کاراکتر تلقی می‌گردد. اگر این عدد توسط تابع ()fscanf از روی فایل خوانده شود، باید عمل تبدیل کاراکتر به عدد صورت گیرد که مستلزم صرف وقت است. برای جلوگیری از این مشکل پیشنهاد می‌شود که از دو تابع ()fread و ()fwrite که در همین فصل بررسی می‌شوند استفاده گردد.

ورودی - خروجی رکورد

همانطور که تاکنون مشاهده کردید توابع زیادی برای انجام اعمال ورودی - خروجی فایل وجود دارند. دو تابع ()fprintf و ()fscanf برای نوشتن و خواندن انواع مختلفی از داده‌ها و با فرمتهای متفاوت بر روی فایل به کار می‌روند؛ البته این دو تابع از سرعت کمی برخوردارند که توصیه می‌شود از آنها استفاده نگردد. برای ورودی - خروجی رکورد و همچنین سایر ورودی - خروجی‌ها می‌توان از دو تابع ()fread و ()fwrite استفاده نمود که از سرعت بالایی برخوردارند. الگوی این توابع در فایل stdio.h قرار داشته و به صورتهای زیر می‌باشند:

```
int  fread (void  *buffer, int num_byte, int count, FILE  *fp)
int  fwrite(void  *buffer, int num_byte, int count, FILE  *fp)
```

در الگوهای فوق، پارامتر buffer به ساختمان داده یا متغیری اشاره می‌کند که داده‌های خوانده شده از فایل باید در آن قرار گیرند و این پارامتر در تابع ()fwrite به محلی از حافظه اشاره می‌کند که داده‌های موجود در آن محل باید در فایل نوشته شوند. پارامتر num_byte در هر دو تابع طول داده‌ای را مشخص می‌کند که باید خوانده یا نوشته شود؛ پارامتر count تعداد عناصری است که طول آن توسط num_byte مشخص گردید و باید در فایل نوشته و یا از فایل خوانده شوند. اشاره‌گر fp به فایلی اشاره می‌کند که توابع ()fread و ()fwrite باید بر روی آنها عمل کنند. به عنوان مثال مجموعه دستورات زیر را در نظر بگیرید:

```
char table [20] ;                          (۱)
char arr [10] ;                            (۲)
fwrite (table, sizeof (char), 20, fp) ;    (۳)
fread (arr, sizeof (char), 10, fp) ;       (۴)
```

دستورات ۱ و ۲ رشته‌هایی به طولهای ۲۰ و ۱۰ را تعریف می‌کنند. دستور ۳، ۲۰ بایت از اطلاعات موجود در آرایه table را در فایلی که fp به آن اشاره می‌کند می‌نویسد. دستور ۴، تعداد ۱۰ بایت از اطلاعات را از فایلی که fp به آن اشاره می‌کند خوانده در متغیر arr قرار می‌دهد. توابع ()fread و ()fwrite در ورودی - خروجی رکورد مورد استفاده قرار می‌گیرند.

مثال ۷-۸

برنامه‌ای که نام، تعداد ساعت کار و حق الزحمه ساعتی کارمندان مؤسسه‌ای را دریافت کرده در یک فایل قرار می‌دهد و سپس با استفاده از این اطلاعات، حقوق آنها را محاسبه کرده به خروجی منتقل می‌کند.

توضیح

برای محاسبه حقوق کارمند، میزان ساعت کار در حقوق ساعتی ضرب می‌شود. ساختمان em برای نگهداری اطلاعات کارمند تعریف شده است : name نام کارمند، hp دستمزد ساعتی و h میزان ساعت کارکرد است. برای خاتمه ورود داده‌ها، به جای نام دانشجو فقط کلید enter را فشار دهید. در این برنامه از متغیر numstr به عنوان یک متغیر کمکی استفاده شده است. علتش این‌است‌که تابع ()scanf در خواندن چند قلم اطلاعات با مشکل مواجه می‌شود (به پیوست ۱ مراجعه شود).

متغیرهای برنامه

متغیر	هدف
p	متغیر فایل
numstr	متغیر کمکی
i	متغیر کمکی
salary	حقوق کارمند که باید محاسبه شود

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
int main() {
   FILE *p;
   char numstr[10];
   int i, salary ;
   struct  em {
     char name [10] ;
     int hp;
     int h;
   } emp ;
   clrscr() ;
   p = fopen("employ.dat", "wb+") ;
   if (!p) {
        printf("cannot open output file, press a key ...") ;
        getch() ;
        exit(1) ;
   }
```

```
gotoxy(14, 2);
puts("<< data entry >>");
gotoxy(3, 3) ;
printf("     name          hour pay    hour ") ;
gotoxy(3,4) ;
printf("----------        ---------    ------");
i = 5 ;
while (1) {
    gotoxy(3, i) ;
    gets(emp.name);
    if(!emp.name[0])
    break ;
    gotoxy(21, i) ;
    gets(numstr) ;
    emp.hp = atoi(numstr) ;
    gotoxy(34, i);
    gets(numstr) ;
    emp.h = atoi(numstr) ;
    i++ ;
    fwrite(&emp,sizeof(struct em), 1, p);
} //end of while
rewind(p) ;
clrscr();
gotoxy(7, 2) ;
puts(" << OUTPUT >> " ) ;
gotoxy(3,3) ;
puts(" name          salary ");
gotoxy(3,4);
puts("----------      --------");
i = 5 ;
fread(&emp , sizeof(struct em), 1, p);
while(!feof(p)) {
   gotoxy(3,i) ;
   puts(emp.name);
   gotoxy(21,i) ;
   printf("%d", emp.hp*emp.h) ;
   i++ ;
   fread(&emp,sizeof(struct em),1,p);
}
gotoxy(6, ++i) ;
puts("press a key ...");
getch() ;
}
```

خروجی

```
          << data entry >>
name        hour pay          hour
----------  ----------------  ---------------
ali         120               100
reza        500               50
ahmad       200               10


        << outpot >>
name        salary
----------  ----------------
ali         12000
reza        25000
ahmad       2000
press a key ...
```

مثال ۸-۸

برنامه‌ای که یک بانک اطلاعاتی از دانشجویان تشکیل می‌دهد. اطلاعاتی که در بانک قرار می‌گیرند عبارتند از: نام دانشجو، شماره دانشجویی، نمره و جنسیت دانشجو.

توضیح

مشخصات دانشجو با ساختمان student تعریف شد که عناصر آن عبارتند از: name نام دانشجو، stno شماره دانشجویی، sex جنسیت (۱ برای مرد، ۲ برای زن)، و grade نمره دانشجو. تابع tolower() حروف بزرگ را به کوچک تبدیل می‌کند.

شرح وظایف برنامه‌ها

تابع main(): فراخوانی توابع مطابق شکل ۴-۸.

تابع init_list(): مقداردهی اولیه به آرایه (با NULL کردن اولین کاراکتر name).

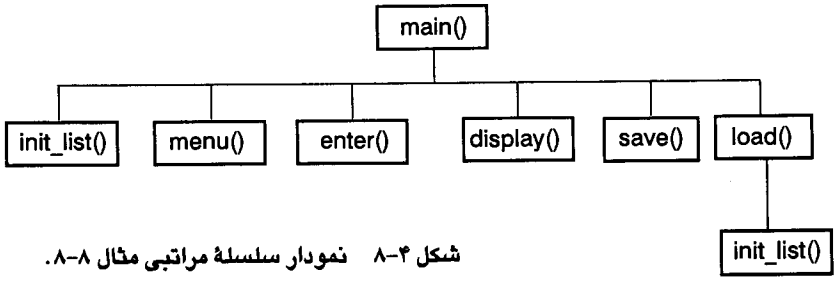
تابع menu(): ظاهر نمودن منویی در صفحه‌نمایش.

تابع enter(): اخذ اطلاعات از صفحه کلید و قراردادن آن در آرایه.

تابع display(): چاپ محتویات لیست در صفحه‌نمایش.

تابع save(): انتقال محتویات لیست به فایل.

تابع load(): انتقال اطلاعات از فایل به لیست.



شکل ۴-۸ نمودار سلسلهٔ مراتبی مثال ۸-۸.

متغیرهای برنامه

برنامه	متغیر	هدف
عمومی	SIZE	طول آرایه‌ای از ساختمانها
	st	آرایه‌ای از ساختمانها برای نگهداری دانشجویان
enter()	i	اندیس آرایه که باید اطلاعات در آن قرار گیرد
	row	سطری از صفحه‌نمایش که اطلاعات از آن دریافت می‌شود
	numstr	متغیر کمکی برای دریافت اطلاعات
display	t	اندیس عنصر آرایه که اطلاعات آن باید چاپ شود
	row	سطری که اطلاعات باید نمایش داده شوند
save()	i	اندیس حلقه‌ای برای دستیابی به عناصر لیست
	fp	متغیر فایل خروجی
load()	fp	متغیر فایل ورودی
	i	اندیس حلقه برای دستیابی به عناصر لیست
menu()	s	انتخاب کاربر

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 100
struct student {
   char name [10] ;
   int stno, sex ;
   float grade ;
} list[SIZE] ;
int menu(void) ;
void init_list(void) , enter(void) ;
void display(void) , save(void) ;
void load(void) ;
int main() {
   init_list();
   for(;;) {
```

```
        switch(menu()) {
            case 'e':  enter(); break;
            case 'd':  display(); break ;
            case 's' : save() ; break;
            case 'l' : load() ; break ;
            case 'q':  exit(0) ;
        }//end of switch
    } //end of for
}
//**********************
void init_list(void)
{
    register int t;
    for (t = 0 ; t < SIZE ; t++)
        *(list[t].name) = '\0' ;
}
//**********************
void enter(void)
{
    register int i ;
    int row ;
    char numstr[10];
    for(i = 0; i < SIZE; i++)
     if (!(*list[i].name))
       break ;
    if(i == SIZE) {
      printf("\n list is full press a key ...") ;
      getch() ;
    }
    clrscr() ;
    gotoxy(10, 2) ;
    puts("<< INPUT DATA >>");
    gotoxy(1, 3) ;
    printf("   name        stno      gread      sex(1, 2)" ) ;
    gotoxy(1,4) ;
    puts (" -------      -----      -------      -------") ;
    row = 5 ;
    for(;;)  {
     gotoxy(1, row);
     gets(list[i].name);
     if(!  (*list[i].name))
        break ;
     gotoxy(16, row);
```

```
        gets(numstr) ;
        list[i].stno = atoi(numstr) ;
        gotoxy(28, row) ;
        gets(numstr) ;
        list[i].grade = atof(numstr) ;
        gotoxy(40, row) ;
        gets(numstr) ;
        list[i].sex = atoi(numstr) ;
        row ++ ;
        i ++ ;
       }
}
 //***********************
void display(void)
  {
    register int t ;
    int row ;
    clrscr() ;
    gotoxy(10, 2) ;
    puts("<< OUTPUT DATA >> ") ;
    gotoxy(1, 3) ;
    printf(" name   stno      grade  sex(1,2)") ;
    gotoxy(1, 4);
    printf("------- ----      -----  --------") ;
    row = 5 ;
    for(t = 0 ; t < SIZE ; t++)
       if(*list[t].name) {
          gotoxy(1, row) ;
          printf("%s", list[t].name);
          gotoxy(12, row) ;
          printf("%d", list[t].stno);
          gotoxy(20, row) ;
          printf("%5.2f", list[t].grade) ;
          gotoxy(30, row) ;
          printf("%d", list[t].sex) ;
          row ++ ;
       } //end of if
    gotoxy(5, row+2);
    printf(" press a key ...");
    getch() ;
}
 //***********************
void save(void)
```

```
{
   FILE *fp ;
   register int i ;
   fp = fopen("st", "wb");
   if(!fp)  {
       printf("\n cannot open file press a key ...");
       getch() ;
       return ;
   }
   for(i = 0 ; i < SIZE ; i++)
     if(*list[i].name)
         fwrite(&list[i], sizeof(struct student), 1, fp) ;
    clrscr() ;
    gotoxy(20, 10) ;
    printf("data saved.press a key.");
    getch() ;
 }
//************************
void load(void)
{
   FILE *fp ;
   register int i ;
   fp = fopen("st", "rb");
   if(!fp) {
      printf("\n cannot open file press a key ...");
      getch() ;
      return ;
   }
   init_list() ;
   for(i = 0 ; i < SIZE ; i++) {
     fread(&list[i], sizeof(struct student), 1, fp);
     if(feof(fp)) {
         clrscr() ;
         gotoxy(20,10) ;
         printf("data loaded.press a key.");
         getch() ;
         return ;
     }//end of if
   }
}
//*****************
int menu(void)
{
```

```
    char s[10] ;
    clrscr();
    do {
       gotoxy(20, 4);
       printf("E) enter data ");
       gotoxy(20, 6);
       printf("D) display on screen");
       gotoxy(20, 8);
       printf("L) load file");
       gotoxy(20, 10);
       printf("S) save in file");
       gotoxy(20, 12);
       printf("Q) quit");
       gotoxy(20, 14);
       printf("enter your select:");
       gets(s) ;
    } while(!strchr("edlsq", tolower(*s)));
    return tolower(*s) ;
}
```

عملکرد برنامه

پس از اجرای برنامه منویی با ۵گزینه ظاهر می‌شود. گزینه اول داده‌ها را از صفحه کلید خوانده در آرایه قرار می‌دهد. گزینه دوم داده‌های موجود در آرایه را در صفحه‌نمایش ظاهر می‌کند، گزینه سوم داده‌های موجود در آرایه را در فایل ذخیره می‌کند، گزینه چهارم داده‌ها را از فایل به آرایه منتقل می‌کند، و گزینه پنجم برنامه را خاتمه می‌دهد:

منوی برنامه

```
E) enter data
D) display on screen
S) save in file
L) load file
Q) quit
enter your select : e
```

حل یک مسأله از طریق فایل‌های ترتیبی

تاکنون با نحوه تشکیل فایل‌های ترتیبی و چگونگی دسترسی به اطلاعات موجود در آن آشنا شده‌ایم. اکنون مساله پیچیده‌تری را در نظر می‌گیریم: یک فایل داده به نام old حاوی اطلاعات مربوط به دانشجویان است که باید بازسازی گردد. منظور از بازسازی فایل، حذف رکورد، تغییر رکورد و یا اضافه‌نمودن رکورد جدیدی به فایل است. فرمت رکوردهای فایل old به صورت زیر می‌باشد:

نمره	درس	نام دانشجو	شماره دانشجویی
۲	۱۰	۲۰	۴

نمونه‌ای از اطلاعات موجود در فایل old به صورت زیر است:

فیلدکلید (شماره دانشجویی)	نام دانشجو	نام درس	نمره
۱۲۳۴	ali	pascal	۱۵
۲۳۴۵	ahmad	pascal	۱۸
۳۴۵۶	reza	C	۱۹
۵۶۷۸	jafar	C	۱۴
۶۷۸۹	jalal	pascal	۱۳
۹۹۹۹	zzzz	zzzz	۹

برای بازسازی فایل old، رکوردهای تراکنش در فایلی به نام trans قرار دارند و با فرمت زیر قرار می‌گیرند:

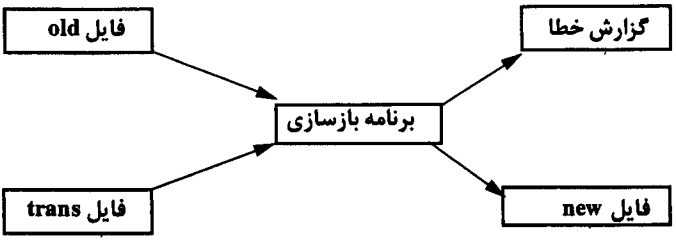
شماره دانشجویی	تراکنش	نام دانشجو	درس	نمره
۲	۱	۲۰	۱۰	۲

در این شکل، **تراکنش** مشخص می‌کند که رکورد باید حذف، اضافه و یا تغییر نماید. اگر تراکنش برابر با I باشد، رکورد باید اضافه گردد، اگر برابر با D باشد، رکورد باید حذف گردد و اگر برابر با C باشد رکورد باید تغییر نماید.
نمونه‌ای از محتویات فایل trans به صورت زیر است:

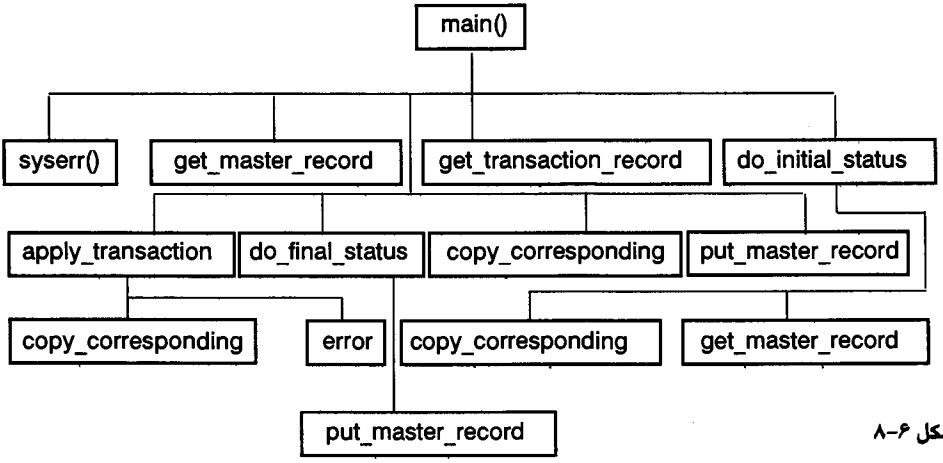
فیلدکلید (شماره دانشجویی)	نوع تغییرات	نام دانشجو	نام درس	نمره
۲۳۴۵	تغییر محتویات رکورد	ahmad	pascal	۲۰
۴۵۶۷	اضافه	rithchie	C	۲۰
۶۷۸۹	حذف	jalal	pascal	۱۳
۹۹۹۹	zzz	zzz	zzz	۹

نتیجهٔ حاصل از ترکیب دو فایل old و trans فایل جدیدی به نام new است که حاوی جدیدترین اطلاعات می‌باشد و محتویات آن به صورت زیر است:

فیلدکلید (شماره دانشجویی)	نام دانشجو	نام درس	نمره
۱۲۳۴	ali	pascal	۱۵
۲۳۴۵	ahmad	pascal	۲۰
۳۴۵۶	reza	C	۱۹
۴۵۶۷	rithchie	C	۲۰
۵۶۷۸	jafar	C	۱۴
۹۹۹۹	zzz	zzz	۱۸



شکل ۵-۸ بازسازی فایل ترتیبی.



شکل ۶-۸
نمودار سلسله مراتبی بازسازی فایل ترتیبی.

چگونگی ترکیب فایل‌های old و trans و تشکیل فایل new در شکل ۵-۸ آمده است. همانطور که در این شکل مشاهده می‌گردد، اگر در حین بازسازی فایل خطایی رخ دهد، مثلاً رکوردی که در فایل old وجود نداشته باشد ولی سعی در حذف آن داشته باشیم، پیام خطای مناسبی در صفحه نمایش چاپ می‌شود. شکل ۶-۸ نمودار سلسله مراتبی برنامه فایل ترتیبی را نشان می‌دهد. این برنامه بر روی دیسک به نام **8-9.cpp** ذخیره شده است.

لیست برنامه بازسازی فایل ترتیبی

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# include <stdlib.h>
# define READONLY      "r"
# define WRITEONLY     "w"
# define FALSE         0
# define TRUE          1
```

```
typedef  int                    boolean;
# define NAMESIZE          21
# define SUBJECTSIZE        11
# define TRAILER           9999
# define INSERT            'I'
# define DELETE            'D'
# define CHANGE             'C'
char      *prog;
FILE      *fopen();
boolean  get_transaction_record(FILE *fp, int *identification,
       char *transaction, char *name, char *subject,int *grade);

boolean  get_master_record(FILE *fp,
int *identification, char *name, char *subject, int *grade);

boolean  put_master_record(FILE *fp,
int identification,char *name,char *subject,int grade);

void   apply_transaction(boolean *allocated,
   int tran_identification, char tran_transaction, char *tran_name,
    char *tran_subject,int tran_grade, int *new_identification, char *new_name,
    char *new_subject, int *new_grade);

void   do_initial_status(int current_key,
  boolean *allocated,FILE *fp, int *old_identification,
  char *old_name, char *old_subject,int *old_grade,
  int *new_identification, char *new_name, char *new_subject,
  int *new_grade);

void   do_final_status(FILE *fp,boolean allocated,
  int identification,char *name, char *subject, int grade);

void   copy_corresponding(int *dest_identification,char *dest_name,
   char *dest_subject, int *dest_grade, int sou_identification,
   char *sou_name, char *sou_subject, int sou_grade);
void   syserr (int errcode, char *message, char *argument);
void   error (char code, char *message, int id);

void main(int argc, char *argv[])
  {
   char b[21], c[11];
   int a, d, current_key;
   FILE     *fpold,*fptrans,*fpnew;
```

```
boolean allocated;
char     tran_transaction;
int      old_identification, tran_identification,  new_identification;
char     old_name[NAMESIZE], tran_name[NAMESIZE],  new_name[NAMESIZE];
char     old_subject[SUBJECTSIZE], tran_subject[SUBJECTSIZE],
         new_subject[SUBJECTSIZE];
int      old_grade, tran_grade, new_grade;
clrscr();
prog = argv[0];
if(argc !=4) syserr(1, "usage:%s file1 file2 file3\n", prog);
 else if((fpold = fopen(argv[1], READONLY)) == NULL)
    syserr(2, "cannot open %s\n",argv[1]);
 else if ((fptrans = fopen (argv[2], READONLY)) == NULL)
  syserr(2, "cannot open %s\n", argv[2]);
 else if (( fpnew = fopen (argv[3], WRITEONLY)) == NULL)
 syserr(2,"cannot open %s\n", argv[3]);
 else
 {
   get_master_record(fpold,  &old_identification,
        old_name, old_subject,&old_grade);
   get_transaction_record(fptrans, &tran_identification,
        &tran_transaction, tran_name,tran_subject,
        &tran_grade) ;
   current_key = tran_identification < old_identification ?
   tran_identification : old_identification;
   while(current_key != TRAILER)
   {
      do_initial_status(current_key, &allocated,
      fpold, &old_identification, old_name,old_subject,
         &old_grade, &new_identification,
         new_name, new_subject, &new_grade);
      while(current_key == tran_identification)
      {
         apply_transaction (&allocated, tran_identification,
          tran_transaction,tran_name, tran_subject,tran_grade,
           &new_identification,new_name, new_subject,&new_grade);
          get_transaction_record(fptrans, &tran_identification,
            &tran_transaction, tran_name, tran_subject, &tran_grade);
      }//end of while
      do_final_status (fpnew, allocated,
         new_identification, new_name, new_subject,new_grade);
       current_key=tran_identification < old_identification ?
        tran_identification : old_identification  ;
```

```
        }//end of while
         copy_corresponding(&new_identification, new_name,
        new_subject, &new_grade, 9999,
          "zzzzzzzzzzzzzzzzzzz", "zzzzzzzzzz", 9);
        put_master_record(fpnew,  new_identification,
              new_name, new_subject, new_grade) ;
        fcloseall();
        printf("\n file new successfully created.\n press a key to continue...");
        getch();
      }
}
 //*************************
boolean  get_transaction_record(FILE *fp, int *identification,
       char *transaction, char *name, char *subject,int *grade)
{
     if (fscanf  (fp,"%4d%c%20s%10s%2d",
     identification, transaction, name, subject, grade) == 5)
          return(TRUE);
     return(FALSE);
}
 //************************
boolean  get_master_record(FILE *fp, int *identification,
      char *name, char *subject, int *grade)
{
       if (fscanf  (fp,"%4d%20s%10s%2d",
            identification,  name,subject,grade)==4)
            return(TRUE);
       return(FALSE);
}
 //************************
boolean  put_master_record(FILE *fp, int identification,
          char *name,char *subject,int grade)
   if(fprintf(fp, "%6d  %-20s  %-10s  %2d\n",
     identification, name, subject, grade) != EOF)
      return(TRUE);
    return(FALSE);
}
 //*****************************

void   apply_transaction(boolean *allocated,
    int tran_identification, char tran_transaction,
    char *tran_name, char *tran_subject,int tran_grade,
    int *new_identification, char *new_name,
```

```
    char *new_subject, int *new_grade)
{
    switch(tran_transaction) {
      case  INSERT:
          if(*allocated == TRUE)
              error(INSERT, "record already exists\n",
                  tran_identification);
           else {
              copy_corresponding( new_identification,
                  new_name, new_subject, new_grade,
              tran_identification, tran_name,tran_subject, tran_grade);
              *allocated = TRUE;
           }//end of else
           break;
      case   DELETE:
           if(*allocated == FALSE)
              error(DELETE, "record dose not exist.", tran_identification);
            else
              *allocated = FALSE;
              break;
      case  CHANGE:
           if(*allocated == FALSE)
           error(CHANGE, "record dose not exist.", tran_identification);
      else
          copy_corresponding( new_identification,
            new_name, new_subject, new_grade,
            tran_identification, tran_name,tran_subject, tran_grade);
       break;
   }//end of switch
}//end of function
 //*******************************
void  do_initial_status(int current_key,
  boolean *allocated,FILE *fp, int *old_identification,
  char *old_name, char *old_subject,int *old_grade,
  int *new_identification, char *new_name,
  char *new_subject, int *new_grade)
{
    if(*old_identification == current_key)  {
      copy_corresponding(new_identification, new_name,
     new_subject, new_grade, *old_identification,
     old_name,old_subject, *old_grade);
     *allocated = TRUE;
     get_master_record(fp, old_identification,
```

```
        old_name, old_subject, old_grade) ;
      } //end of if
      else
      *allocated=FALSE;
}
 //***************************
void   do_final_status(FILE *fp,boolean allocated,
      int identification,char *name, char *subject, int grade)
{
      if(allocated)
      put_master_record(fp, identification, name, subject, grade) ;
}
 //*******************************
void   copy_corresponding(
    int *dest_identification,char *dest_name,
    char *dest_subject, int *dest_grade, int sou_identification,
    char *sou_name, char *sou_subject, int sou_grade)
{
     *dest_identification = sou_identification;
      strcpy(dest_name, sou_name);
      strcpy(dest_subject, sou_subject);
      *dest_grade = sou_grade;
}
 //**********************
void   syserr (int errcode, char *message, char *argument)
{
     fprintf(stderr,"%s[%2d]:", prog, errcode);
     fprintf(stderr, message, argument);
     exit(errcode);
}
 //*****************
void   error (char code, char *message, int id)
{
       printf("%c     %30s    %4d\n", code, message, id);
}
```

اجرای برنامهٔ بازسازی فایل ترتیبی و محتویات فایل‌های داده :

فایل new

2345	ahmad	pascal	20
3456	reza	c	19
4567	richis	c	20
5678	jafar	c	14
9999	zzzzzzzzzzzzzzzzzzz	zzzzzzzzzz	9

فایل old

```
2345ahmad          pascal      20
3456reza           c           19
4567richis         c           20
5678jafar          c           14
9999zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz    9
```

فایل trans

```
2345Cahmad         pascal      20
4567Irichis        c           20
6789Djalal         pascal      13
9999zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz09
```

جدول ۲-۸ مقادیر معتبر پارامتر origin در تابع ()fseek .

مبدأ حرکت	نام ماکرویی که بیانگر این مبدأ است
ابتدای فایل	SEEK_SET
موقعیت فعلی فایل	SEEK_CUR
انتهای فایل	SEEK_END

دسترسی تصادفی به فایل (ورودی - خروجی تصادفی)

در برنامه‌هایی که تاکنون نوشته شده‌اند، از سازمان فایل ترتیبی برای ذخیره و بازیابی اطلاعات استفاده شده است. اکنون چگونگی تشکیل فایل‌های تصادفی مطرح شده، طریقهٔ دسترسی به اطلاعات موجود در آنها بررسی می‌شود. مفهوم دستیابی تصادفی به فایل این است که در هر جایی از فایل بتوانیم بنویسیم و از هر جایی از فایل بتوانیم اطلاعات را بخوانیم. اگر بتوانیم "موقعیت‌سنج فایل" را به هر جایی که خواستیم، منتقل کنیم، به مفهوم تصادفی فایل دست می‌یابیم. برای تغییر مکان موقعیت‌سنج فایل از تابع ()fseek استفاده می‌شود. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر می‌باشد :

```
int  fseek  (FILE   *fp , long num_byte, int origin)
```

اساس کار تابع ()fseek به این صورت است که با شروع از یک محل در فایل، که توسط پارامتر origin مشخص می‌شود، "موقعیت سنج فایل" را به طول num_byte بایت تغییر مکان می‌دهد. یعنی تغییر مکان "موقعیت سنج فایل" به طور نسبی بوده، نسبت به محلی که توسط origin تعیین می‌گردد سنجیده می‌شود. به عنوان مثال، اگر origin محل ۱۰۰ فایل را به عنوان مبدأ حرکت مشخص کند و طول num_byte برابر با ۲۰۰ باشد، "موقعیت سنج فایل" به محل ۳۰۰ منتقل خواهد شد. مقادیری که origin می‌تواند بپذیرد در جدول ۲-۸ آمده‌اند.

جدول ۲-۸ گویای این است که اگر پارامتر origin برابر با SEEK_SET باشد، "موقعیت‌سنج‌فایل" از ابتدای فایل به طول num_byte بایت به طرف انتهای فایل حرکت می‌کند. اگر پارامتر origin برابر با SEEK_CUR باشد، "موقعیت‌سنج‌فایل" نسبت به محل فعلی خودش به اندازهٔ num_byte بایت به طرف انتهای فایل حرکت می‌کند. اگر پارامتر origin برابر با SEEK_END باشد، "موقعیت‌سنج‌فایل" به اندازهٔ num_byte بایت، طول فایل را افزایش می‌دهد و سپس به این محل اشاره می‌نماید. محلی که برای هر رکورد در فایل تصادفی در نظر گرفته می‌شود، "ناحیه

رکورد" نام دارد. چون هر رکورد در ناحیه مربوط به خودش نوشته می‌شود، ممکن‌است چندین ناحیه رکورد بین رکوردهای فایل خالی باقی بماند. به عنوان مثال، فرض می‌کنیم در یک فایل مربوط به کارمندان، طول رکورد ۱۰۰ بوده، هر رکورد در ناحیه‌ای که با شماره کارمندی مشخص می‌گردد قرار می‌گیرد. مجدداً فرض می‌کنیم که تعداد کارمندان ۵ نفر و شماره کارمندی آنها ۱۰۰، ۵۰۰، ۶۰۰، ۸۰۰ و ۹۰۰ باشد. پس از قرارگرفتن مشخصات این کارمندان در فایل، ۹۹ ناحیه رکورد (۱۰۰ × ۹۹ بایت) قبل از رکورد اول و ۳۹۹ ناحیه قبل از رکورد دوم و ... به هدر می‌روند. این موارد را شاید بتوان به عنوان یکی از معایب فایل تصادفی در نظر گرفت. ولی معمولاً مکانیزمهایی برای بهبود این حالت منظور می‌شود که نمونه‌ای از آن، در ادامه بررسی می‌شود.

حل یک مسأله از طریق فایل تصادفی

در قسمت قبلی چگونگی بازسازی یک فایل ترتیبی بررسی شد. در این قسمت مسأله بازسازی فایل تصادفی مطرح می‌گردد. در بازسازی فایل‌های ترتیبی، فایل تراکنش باید مرتب باشد و داده‌های جدید در فایل دیگری قرار می‌گیرند. اما در بازسازی فایل‌های تصادفی، نیازی به این دو مورد نیست. در فایل تصادفی، هر رکورد با یک شماره به نام شماره رکورد نسبی مشخص می‌گردد. به عنوان مثال، اگر فایلی دارای n رکورد باشد، شماره رکورد نسبی از ۱ تا n خواهند بود. فرمت فایل تراکنش در بازسازی فایل تصادفی، همانند فرمت فایل تراکنش در بازسازی فایل ترتیبی است. ولی فرمت فایل دادهٔ اصلی به صورت زیر می‌باشد:

نمره	درس	نام	شماره دانشجویی	وضعیت

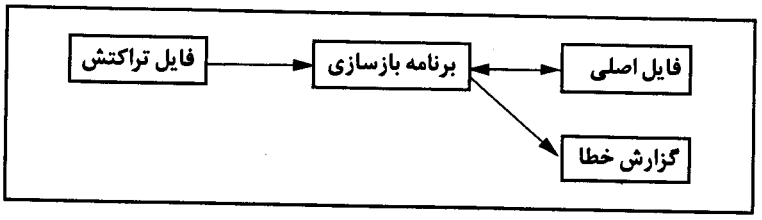
فیلد "وضعیت" مشخص می‌کند که آیا در این ناحیهٔ رکورد، داده‌ای وجود دارد یا خیر. اگر این فیلد برابر با blank باشد یعنی در این ناحیه داده‌ای وجود ندارد، وگرنه (در اینجا اگر برابر با ۱ باشد) به معنی وجود داده در این ناحیه است.

تشکیل و یا سازماندهی فایل تصادفی به روشهای مختلفی انجام می‌شود، یک روش این است که شماره دانشجویی هر دانشجو، شماره رکورد نسبی آن نیز باشد؛ به عنوان مثال، اگر شماره دانشجویی برابر با ۱۲۳۴ باشد، شماره رکورد نسبی آن نیز ۱۲۳۴ باشد. این امر موجب می‌شود تا ناحیه رکورد زیادی، بلااستفاده باشند. یک راه برای رفع این مشکل، استفاده از یک تابع درهم‌سازی است. می‌توان انواع مختلفی از تابع درهم‌سازی را انتخاب نمود. در مثالی که ما در نظر گرفته‌ایم، شماره دانشجویی بر بزرگترین عدد اولِ کوچکتر از تعداد دانشجویان تقسیم شد و یک واحد به باقیمانده تقسیم اضافه شده است (تعداد دانشجویان ۱۰۰ فرض شده و ۹۷ عدد اول مورد نظر می‌باشد) تا شماره رکورد نسبی به دست می‌آید. در این مورد ممکن است رکوردهای زیادی، به یک ناحیه از رکورد مراجعه نمایند؛ به عنوان مثال مقادیر ۱۲۳۴ و ۲۳۹۸ به ناحیهٔ رکورد ۷۱ مراجعه خواهند کرد:

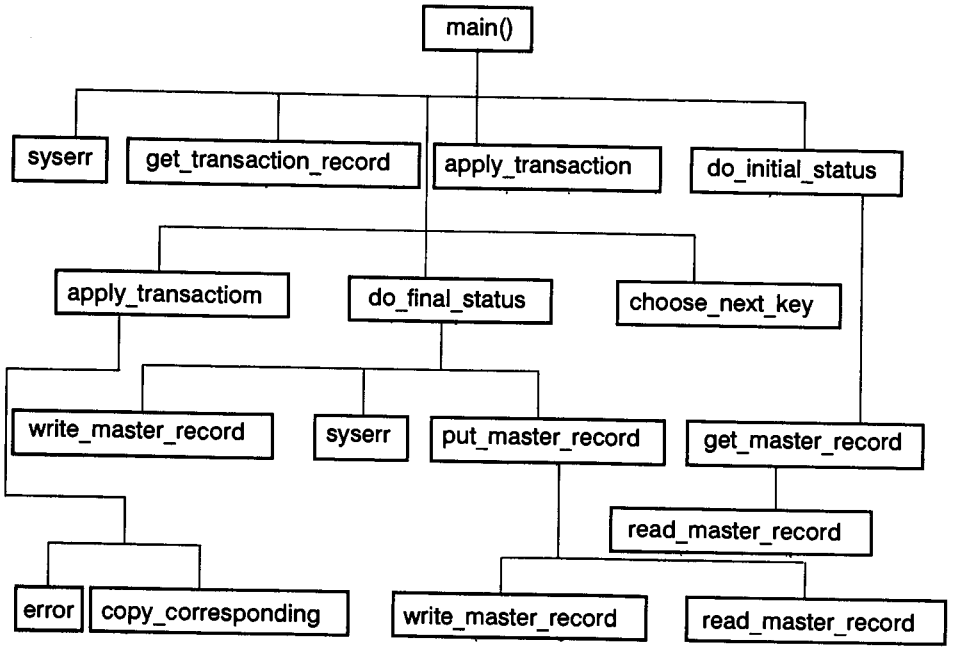
۲۳۹۸ ٪ ۹۷ + ۱ = ۷۱
۱۲۳۴ ٪ ۹۷ + ۱ = ۷۱

اگر این ناحیه رکورد، قبلاً توسط مقدار ۱۲۳۴ اشغال شده باشد، برای قراردادن رکوردی با شماره دانشجویی ۲۳۹۸ در فایل، ناحیه‌های بعد از ۷۱ جستجو می‌شوند تا یک محل خالی پیدا شده، این رکورد در آنجا نوشته شود.

در بازسازی فایل تصادفی، رکوردهای تراکنش از فایل مربوط خوانده شده، تغییرات موردنظر در فایلِ داده اصلی اعمال می‌گردند (بدون این که نیاز به فایل جدیدی باشد) (شکل ۷-۸). شکل ۸-۸ نمودار سلسله مراتبی برنامهٔ بازسازی فایل تصادفی را نشان می‌دهد.



شکل ۷-۸ بازسازی فایل تصادفی.



شکل ۸-۸ نمودار سلسله مراتبی برنامه بازسازی فایل تصادفی.

لیست برنامهٔ بازسازی فایل تصادفی(8-10.cpp بر روی دیسک)

```
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <stdio.h>
#define READONLY        "r"
#define UPDATE          "r+"
#define FALSE           0
#define TRUE            1
typedef  int            boolean;
```

```
# define NAMESIZE          21
# define SUBJECTSIZE       11
# define TRAILER          9999
# define ABSOLUTE          0
# define PRIME            17
# define INSERT           'I'
# define DELETE           'D'
# define CHANGE           'C'
# define ERROR            -1
# define ABSENT            0
# define OCCUPIED          '*'
# define UNOCCUPIED        ' '
typedef  struct {
   int identification ;
   char name[NAMESIZE] ;
   char subject[SUBJECTSIZE] ;
   int  grade ;
} RECORD ;
typedef  struct {
              char type ;
              RECORD student ;
             } TRANSACTION ;
typedef  struct {
               char occupied ;
               RECORD student ;
              } MASTER ;
char *prog ;
boolean  get_transaction_record(FILE *fp, TRANSACTION *transaction);

int read_master_record(FILE *fp, int relative_record_number,
MASTER *master);

Int write_master_record(FILE *fp, Int relative_record_number,
MASTER *master);

int  get_master_record(FILE *fp, int key, MASTER *master);

int  put_master_record(FILE *fp, int key,MASTER *master);

Int choose_next_key(Int identification);

Int address(Int key);
```

```
void apply_transaction(boolean *allocated,TRANSACTION *transaction,
MASTER *master);
int  do_initial_status(int current_key,
     boolean *allocated, FILE *fp, MASTER *master);

void   do_final_status(FILE *fp, int relative_record_number,
       boolean allocated, MASTER *master);

void   copy_corresponding(RECORD *destination, RECORD *source);

void   syserr (int errcode, char *message, char *argument);

void   error (char code, char *message, int id);

void   main(int argc, char *argv[])
 {
   FILE      *fpmas,*fptrans ;
   int       current_key ,   relative_record_number ;
   boolean      allocated;
   TRANSACTION transaction;
   MASTER       master ;
   clrscr();
   prog = argv[0];
   if(argc != 3)
     syserr(1, "usage: %s file1 file2 ", prog);
   else if((fptrans=fopen (argv[1], READONLY)) == NULL)
     syserr(2, "cannot open %s\n", argv[1]);
   else if((fpmas=fopen (argv[2], UPDATE)) == NULL)
    syserr(2, "cannot open %s\n", argv[2]);
   else
   {
    get_transaction_record(fptrans,  &transaction) ;
    current_key =   choose_next_key(transaction.student.identification);
    while(current_key != TRAILER)
    {
     relative_record_number =  do_initial_status(
       current_key,&allocated,  fpmas,&master);
     while(current_key ==   transaction.student.identification)
     {
       apply_transaction (&allocated,  &transaction,&master) ;
       get_transaction_record(fptrans,  &transaction);
     }
     do_final_status (fpmas,  relative_record_number,  allocated,  &master);
```

```
        current_key =  choose_next_key(transaction.student.identification);
      } //end of while
      fcloseall();
    } //end of else
}
 //**************************
boolean get_transaction_record(FILE *fp, TRANSACTION *transaction)
{
    if (fscanf (fp, "%4d%c%20s%10s%2d",
          &transaction -> student.identification,
          &transaction -> type,
          transaction -> student.name,
          transaction -> student.subject,
          &transaction -> student.grade) == 5)
       return(TRUE);
     return(FALSE);
}
 //********************
int read_master_record(FILE *fp,
      int relative_record_number, MASTER *master)
{
   if(fseek(fp, (long) relative_record_number
        *sizeof(MASTER), SEEK_SET) !=0 )
       return(ERROR) ;
   else if(fread((char *)master, sizeof(MASTER), 1, fp) != 1)
       return(ERROR) ;
   else
        return(relative_record_number) ;
}
 //****************************
int write_master_record(FILE *fp,
    int relative_record_number, MASTER *master)
{
    if(fseek(fp, (long) relative_record_number
         *sizeof(MASTER), ABSOLUTE) != 0)
       return(ERROR) ;
    else if(fwrite((char *) master, sizeof(MASTER), 1, fp) != 1)
       return(ERROR);
    else
       return(relative_record_number) ;
 }
 //***********************
int get_master_record(FILE *fp, int key, MASTER *master)
```

```
  {
   int k,  relative_record_number ;
   relative_record_number =  address(key) ;
   for(k = 0 ; k < PRIME ; k++)
   if(read_master_record(fp,
   relative_record_number,master) == ERROR)
     return(ERROR) ;
   else if(master -> occupied == UNOCCUPIED)
      return(ABSENT) ;
   else if(key==  master->student.identification)
      return(relative_record_number) ;
   else
     relative_record_number=  relative_record_number==PRIME ?
         1 :  relative_record_number+1 ;
    return(ABSENT);
  }
//************************
int  put_master_record(FILE *fp, int key,MASTER *master)
    {
     int k ,  relative_record_number ;
     MASTER   temp_master ;
      relative_record_number=address(key);
     for(k=0 ; k<PRIME ; k++)
      if(read_master_record(fp,
       relative_record_number,
       &temp_master)==ERROR)return(ERROR);
      else if
       (temp_master.occupied==UNOCCUPIED)
        return(write_master_record(fp,
       relative_record_number,master));
      else if(key==
       temp_master.student.identification)
          return(write_master_record(fp,
         relative_record_number,master));
       else
        relative_record_number=
        relative_record_number==PRIME ? 1
            : relative_record_number+1 ;
       syserr(20,
       "overflow  in:%s\n","masterfile") ;
       return 0;
    }
//**********************
```

```
int choose_next_key(int identification)
{
      return(identification) ;
}
//****************
int address(int key)
{
     return(key % PRIME + 1);
}
//**********************

void  apply_transaction(boolean *allocated,
     TRANSACTION *transaction, MASTER *master)
{
   switch(transaction -> type)
   {
    case  INSERT:
     if(*allocated==TRUE)
      error(INSERT,  "record already exists\n",
      transaction->  student.identification);
     else
      {
       copy_corresponding( &master -> student,  &transaction -> student);
       *allocated=TRUE;
      }
      break;
    case   DELETE:
      if(*allocated==FALSE)
       error(DELETE, "record dose not exist\n",  transaction->
            student.identification);
       else
        *allocated=FALSE;
       break;
     case  CHANGE:
      if(*allocated==FALSE)
       error(CHANGE,  "record dose not exist\n", transaction->
            student.identification);
      else
       copy_corresponding(&master -> student,  &transaction -> student) ;
      break;
    }
  }
//********************
```

```
int  do_initial_status(int current_key,
    boolean *allocated, FILE *fp, MASTER *master)
    {
     int  relative_record_number ;
     *allocated = FALSE;
     relative_record_number =  get_master_record(fp, current_key, master) ;
     if(relative_record_number != ABSENT)
         *allocated=TRUE;
     return(relative_record_number);
    }
//*********************

void    do_final_status(FILE *fp, int relative_record_number,
         boolean allocated, MASTER *master)
    {
     if(allocated==TRUE)
     {
       if(relative_record_number != ABSENT)
       {
         master -> occupied=OCCUPIED ;
         if(write_master_record(fp,  relative_record_number,  master)==ERROR)
          syserr(21,"error writing %s\n", "master record") ;
       }
       else
       {
         master -> occupied=OCCUPIED ;
         put_master_record(fp,  master -> student.identification,  master) ;
       }
     }
    else
    {
      master -> occupied=UNOCCUPIED ;
      if(write_master_record(fp,  relative_record_number, master)==ERROR)
          syserr(22,  "error writing %s","master record");
     }
    }
//*********************
void  copy_corresponding(RECORD *destination, RECORD *source)
{
    destination ->identification=
    source -> identification;
    strcpy(destination -> name,  source -> name);
    strcpy(destination -> subject,  source -> subject);
```

```
    destination ->grade=source -> grade;
}
 //*********************
void  syserr (int errcode, char *message, char *argument)
   {
    fprintf(stderr,"%s[%2d]:",prog,  errcode);
    fprintf(stderr,message,argument);
    exit(errcode);
   }
//************************
void  error (char code, char *message, int id)
   {
     printf("%c    %30s    %4d\n",code,  message, id);
   }
```

دستگاههای ورودی - خروجی استاندارد

وقتی اجرای یک برنامه به زبان C آغاز می‌شود، ۵ فایل به طور اتوماتیک باز می‌شوند که اشاره‌گرهای آنها در جدول ۳-۸ مشاهده می‌گردند.

جدول ۳-۸ دستگاههای ورودی خروجی استاندارد.

اشاره‌گر دستگاه (فایل)	نام دستگاه (فایل)
stdin	دستگاه ورودی استاندارد (صفحه کلید)
stdout	دستگاه خروجی استاندارد (صفحه‌نمایش)
stderr	دستگاه استاندارد جهت ثبت پیامهای خطا (صفحه‌نمایش)
stdprn	دستگاه استاندارد چاپ (چاپگر موازی)
stdaux	پورت سری (serial port)

به عنوان مثال، مجموعه دستورات زیر را در نظر بگیرید :

```
putc (ch, stout) ;                    (۱)
printf (stout ,"%d, %d" , a, b) ;     (۲)
fscanf (stdin, "%d%d" , &x, &y) ;     (۳)
```

دستور ۱ موجب می‌شود تا کاراکتر ch در صفحه‌نمایش نوشته شود، دستور ۲ موجب می‌شود تا متغیرهای a و b در صفحه‌نمایش نوشته شوند؛ دستور ۳ موجب می‌شود تا متغیرهای x و y از صفحه کلید خوانده شوند.

دستگاههای استاندارد ورودی - خروجی همانطور که به طور اتوماتیک باز می‌شوند، به طور اتوماتیک نیز بسته خواهند شد و نیاز به بستن آنها توسط برنامه‌نویس نیست.

تمرینات

۱. برنامه‌ای بنویسید که کاراکترهایی را از صفحه‌کلید گرفته، در یک فایل باینری قرار دهد و سپس کاراکترهای موجود در این فایل را خوانده به یک فایل باینری دیگر منتقل کند. اسامی فایل‌های ورودی و خروجی به عنوان آرگومان تابع اصلی به برنامه وارد شوند. آخرین کاراکتر ورودی، $ است.

۲. اشکالات دستورات زیر را بیابید:

```
main () {
    int *p ;
    p = fopen ("test.dat") ;
    for (i = 0 ; i < 50 ; i ++)
        fputs (fp, "this is a test.") ;
    fclose (p) ;
}
```

۳. تفاوت فایل‌های باینری و متنی را بیان کنید.

۴. روشهای ذخیره و بازیابی اطلاعات را نام ببرید، در هر روش از چه توابعی استفاده می‌کنید؟

۵. سازمان فایل چیست ؟

۶. سازمان فایل ترتیبی و تصادفی چه تفاوتهایی با هم دارند؟

۷. سازمان فایل تصادفی را چگونه می‌توان ایجاد کرد؟

۸. دستگاههای ورودی - خروجی استاندارد را توصیف کنید.

۹. در حالتهای "w+" ، "r+" و "a+" فایل‌ها به صورتی باز می‌شوند که امکان نوشتن و خواندن اطلاعات از فایل وجود دارد. کدامیک از این روشها برای تغییردادن اطلاعات در این فایل مفیداند؟

۱۰. برنامه‌ای بنویسید که یک کاراکتر و نام فایلی را به عنوان آرگومان پذیرفته، سطرهایی از فایل را که حاوی این کاراکتر است در صفحه‌نمایش ظاهر کند. انتهای هر سطر به '\n' ختم می‌شود و طول هر سطر حداکثر ۲۵۶ کاراکتر است.

۱۱. برنامه‌ای بنویسید که نام چند فایل را به عنوان آرگومان پذیرفته، اطلاعات هر کدام از فایل‌ها را به ترتیب در صفحه‌نمایش ظاهر کند، از argc برای ایجاد حلقه استفاده کنید.

۱۲. برنامه‌ای بنویسید که نام دو فایل را به عنوان آرگومان پذیرفته، محتویات آنها را طوری نمایش دهد که یک سطر از فایل اول، سطر بعدی از فایل دوم و ... باشد.

۱۳. برنامه‌ای بنویسید که نام فایل و یک رشته را به عنوان آرگومان بپذیرد و تمام بخشهایی از فایل را که حاوی این رشته است در خروجی چاپ کند.

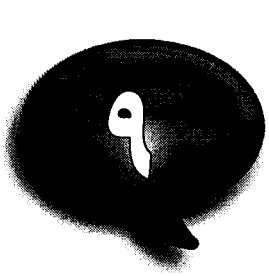
۱۴. برنامه‌ای بنویسید که کتابخانه شخصی شما را به صورت کامپیوتری اداره کند. اطلاعاتی که در فایل قرار می‌گیرند عبارت‌انداز:

۱. نام کتاب ۲. نام نویسنده یا مترجم ۳. تاریخ خرید
۴. قیمت کتاب ۵. موضوع کتاب ۶. شماره یا کد کتاب

گزارشهایی که برنامه باید اخذ نماید عبارت اند از:

۱. مشخصات کلیه کتابها ۲. جستجو براساس موضوع کتاب و چاپ اطلاعات آن
۳. جستجو براساس نام کتاب یا نویسنده و چاپ اطلاعات آن ۴. کتابهایی که در تاریخ خاصی خریداری شده‌اند
۵. جستجو براساس شماره یا کد کتاب و چاپ اطلاعات آن ۶. هزینه‌هایی که در یک دوره برای خرید کتابها شده است

۱۵. برنامه‌ای بنویسید که با استفاده از عملگر sizeof اندازه تمام انواع داده‌ها را در سیستم شما مشخص کند و در فایلی بنویسد. سپس محتویات آن فایل را خوانده، در صفحه‌نمایش ظاهر کند.



توابع کتابخانه‌ای

توابع جهت انجام اعمال ریاضی، کاراکتری، رشته‌ای، مقایسه، تبدیل انواع و غیره مورد استفاده قرار می‌گیرند. در اعمال ریاضی می‌توان تولید اعداد تصادفی، محاسبه قدرمطلق اعداد و لگاریتم را نام برد و در اعمال کاراکتری می‌توان خواندن و نوشتن کاراکترها و در مورد اعمال رشته‌ای می‌توان کپی رشته‌ای در رشتهٔ دیگر، مقایسه رشته‌ها و غیره را نام برد. به طور کلی می‌توان گفت که توابع در زبان C از تنوع بسیار زیادی برخوردارند.

در زبان C هر تابع دارای الگویی است که این الگو نوع تابع و نوع پارامترهای آن را مشخص می‌کند. الگوی هر تابع در یک فایل header قرار دارد. ضمن بررسی توابع، الگوی آنها ذکر شده و فایل header که این الگو در آنجا قرار دارد معرفی می‌گردد. به دلیل اینکه درک مفاهیم بعضی از توابع نیاز به داشتن اطلاعاتی در موضوعات مختلف است، لذا این توابع در ۳ فصل (۹، ۱۵ و ۱۹) مورد بررسی قرار می‌گیرند.

توابع ریاضی

توابع ریاضی در C برای انجام بعضی از اعمال ریاضی از قبیل محاسبه سینوس، کسینوس، تانژانت و کتانژانت، جذرگیری، تعیین لگاریتم اعداد، قدرمطلق و غیره مورد استفاده قرار می‌گیرند که الگوی اکثر آنها در فایل math.h قرار دارد.

تابع ()abs

این تابع برای محاسبهٔ قدر مطلق اعداد صحیح مورد استفاده قرار می‌گیرد. اگر آرگومان این تابع یک عدد منفی باشد نتیجهٔ حاصل از تابع یک عدد مثبت است و اگر آرگومان تابع، مثبت و یا صفر باشد نتیجه، یک عدد مثبت یا صفر خواهد بود. الگوی این تابع که در فایل math.h وجود دارد، به صورت زیر است:

```
Int abs (int num)
```

در این الگو، num عددی است که قدرمطلق آن باید محاسبه گردد.

تابع ()acos

تابع ()acos برای محاسبه آرک کسینوس یک عدد. مورد استفاده قرار می‌گیرد و الگوی آن که در فایل math.h قرار دارد، به صورت زیر است:

```
double acos (double arg)
```

مقادیری را که آرگومان این تابع می‌پذیرد در بازهٔ ۱- تا ۱ است و نتیجهٔ حاصل به صورت رادیان و در بازه صفر و π است.

مثال ۱-۹

برنامه‌ای که آرک کسینوس مقادیر ۱- تا ۱ را محاسبه می‌کند.

```
#include <conio.h>
#include <math.h>
#include <stdio.h>
void main()
{
 double val = -1.0;
 clrscr();
 do {
    printf("arc cosine %.2f is: %.2f\n" ,val, acos(val));
    val += 0.1;
 } while(val <= 1.0);
 getch();
}
```

تابع asin()

تابع asin() برای محاسبه آرک سینوس یک عدد به کار رفته و الگوی آن به صورت زیر است:

double asin (double arg)

آرگومان این تابع در بازه ۱- تا ۱ است و نتیجه حاصل به صورت رادیان و در بازه $-\pi/2$ و $\pi/2$ است.

مثال ۲-۹

برنامه‌ای که آرک سینوس اعداد ۱- تا ۱ را محاسبه می‌کند.

```
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
void  main()
{
    double val = - 1.0;
    clrscr();
    do {
       printf("arc sine of %f is  %f\n",val,asin(val));
       val +=0.1;
    } while(val <= 1.0);
    getch();
}
```

تابع atan()

تابع atan() برای محاسبه آرک تانژانت یک عدد به کار می‌رود و الگوی آن به صورت زیر تعریف شده است:

double atan (double arg)

مقادیری که توسط این تابع محاسبه می‌شوند به صورت رادیان و در بازهٔ $-\pi/2$ و $\pi/2$ است.

مثال ۳-۹

برنامه‌ای که آرک تانژانت مقادیر ۱- تا ۱ را محاسبه می‌کند.

```
#include <conio.h>
#include <math.h>
#include <stdio.h>
void main() {
  double val = -1.0;
  clrscr();
  do {
    printf("arc tangent of %f is %f\n" , val, atan(val));
    val += 0.1;
  } while(val <= 1.0);
  getch();
}
```

تابع atan2()

تابع atan2() دو آرگومان را دریافت کرده، اولین آرگومان را بر دومین آرگومان تقسیم کرده و آرک تانژانت نتیجه حاصل را محاسبه می‌کند. الگوی تابع به صورت زیر تعریف شده است:

double atan2 (double y, double x)

همانطور که مشاهده می‌شود تابع atan2() دو آرگومان، به نامهای y و x دارد که آرک تانژانت y/x را محاسبه می‌کند.

مثال ۴-۹

برنامه‌ای که آرک تانژانت اعداد ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <math.h>
#include <stdio.h>
void main() {
  double y = -1.0;
  clrscr();
  do {
     printf("atan2 of %f is %f\n", y, atan2(y, 1.0));
     y += 0.1;
  } while(y <= 1.0);
  getch();
}
```

تابع ()complex و سایر توابع وابسته به آن

تابع complex بخشهای حقیقی و غیرحقیقی یک عدد موهومی را گرفته، عدد موهومی را تولید می‌کند. ساختمان عدد موهومی، به نام complex در فایل complex.h به صورت زیر تعریف شده است:

```
struct   complex {
            double x, y ;
} ;
```

تابع ()imag بخش غیرحقیقی عدد موهومی و تابع ()real بخش حقیقی عدد موهومی و تابع ()conj کل عدد موهومی را مشخص می‌کند. الگوی این توابع در فایل complex.h قرار دارد و به صورت زیر است:

```
complex complex (double real, double imag) ;
double  real (complex z) ;
double  imag (complex z) ;
double  conj (complex z) ;
```

مثال ۵-۹

برنامه‌ای که کاربرد توابع اعداد موهومی را نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <complex.h>
int main(void)
{
   double x = 3.1, y = 4.2;
   clrscr();
   complex z = complex(x,y);
   cout<< "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary real part = " << imag(z) << "\n";
   cout << "z has complex conjugate = " << conj(z) << "\n";
   getch();
   return 0;
}
```

تابع ()ceil

تابع ()ceil کوچکترین عدد صحیح بزرگتر یا مساوی با یک عدد را که به عنوان آرگومان آن می‌باشد محاسبه می‌کند و الگوی آن به صورت زیر است:

double ceil (double num)

به عنوان مثال اگر 1.02 به عنوان آرگومان تابع باشد نتیجهٔ حاصل برابر با ۲ و اگر آرگومان تابع برابر با 1.02- باشد

نتیجهٔ حاصل برابر با ۱- خواهد بود. خروجی حاصل از اجرای دستور زیر عدد ۱۰ می‌باشد.

```
printf ("%f",ceil(9.9)) ;
```

تابع cos()

تابع cos() برای محاسبه کسینوس یک زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است:

double cos (double arg)

مثال ۶-۹

برنامه‌ای که کسینوس مقادیر ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
int main() {
  double val = -1.0;
  clrscr();
  do {
     printf("cosine of %f is %f\n", val, cos(val));
     val += 0.1;
  } while(val <= 1.0);
  getch();
  return 0;
}
```

تابع cosh()

تابع cosh() برای محاسبهٔ کسینوس هیپربولیک یک عدد به کار می‌رود و الگوی آن به صورت زیر است:

double cosh (double arg)

مثال ۷-۹

برنامه‌ای که کسینوس هیپربولینک مقادیر ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main() {
  double val = -1.0;
  clrscr();
  do {
     printf("hyperbolic cosine of %f is %f\n" , val, cosh(val));
    val += 0.1;
  } while(val <= 1.0);
  getch();
}
```

تابع exp()

تابع exp() برای محاسبهٔ توانی از e (پایه لگاریتم طبیعی) مورد استفاده قرار گرفته و الگوی آن به صورت زیر است :

```
double exp (double arg)
```

به عنوان مثال، دستور زیر مقدار ۲/۷۱۸۲۸۲ را که برابر با e است به عنوان نتیجهٔ عمل برمی‌گرداند:

```
printf ("value of e to the first : %f", exp (1.01) ;
```

تابع fabs()

تابع fabs() برای محاسبهٔ قدر مطلق اعداد اعشاری مورد استفاده قرار می‌گیرد و الگوی آن به صورت زیر است :

```
double fabs (double num)
```

مثال ۸-۹

برنامه‌ای که مقادیر قدر مطلق دو عدد اعشاری را در صفحه‌نمایش چاپ می‌کند :

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
  clrscr();
  printf("%1.1f %1.1f", fabs(1.0), fabs(-1.0));
  getch();
}
```

تابع floor()

تابع floor() بزرگترین مقدار صحیح کوچکتر یا مساوی یک عدد را که به صورت double نمایش داده می‌شود محاسبه می‌کند و الگوی آن به صورت زیر است :

```
double floor (double num)
```

اگر مقدار ۱/۰۲ به عنوان آرگومان تابع باشد نتیجهٔ حاصل برابر با ۱/۰ و اگر عدد ۱/۰۲- به عنوان آرگومان تابع باشد نتیجهٔ حاصل برابر با ۱/۰- خواهد بود. دستور زیر عدد ۱۰ را به عنوان نتیجهٔ عمل چاپ می‌کند:

```
printf ("%f", floor (10.9)) ;
```

تابع fmod()

تابع fmod() دو آرگومان دارد که باقیماندهٔ تقسیم اولین آرگومان را بر آرگومان دوم محاسبه می‌کند و به عنوان نتیجهٔ عمل برمی‌گرداند. الگوی تابع به صورت زیر است :

double fmod (double x , double y)

مثال ۹-۹

برنامه‌ای که مقدار ۱/۰ را به عنوان نتیجهٔ عمل (که باقیمانده ۱۰ بر ۳است) در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
  clrscr();
  printf("%1.1f", fmod(10.0, 3.0));
  getch();
}
```

تابع frexp()

تابع frexp() دارای دو آرگومان است. اولین آرگومان را به دو قسمت تبدیل می‌کند که قسمت اول به صورت کسری و در بازهٔ ۰/۵ تا کمتر از یک و قسمت دوم به صورت توان است. الگوی تابع به صورت زیر است :

double frexp (double num , int *exp)

تابع frexp() عدد num را به صورت $\text{کسر} * 2^{\text{exp}} = \text{num}$ درمی‌آورد که قسمت کسر به عنوان نتیجه عمل توسط تابع برگردانده می‌شود و قسمت توان در متغیر exp قرار می‌گیرد.

مثال ۱۰-۹

برنامه‌ای که مقدار ۰/۶۲۵ را به عنوان قسمت کسری و عدد ۴ را به عنوان توان در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
  int e;
  double f;
  clrscr();
  f = frexp(10.0, &e);
  printf("%.3f %d", f, e);
  getch();
}
```

تابع ldexp()

تابع ldexp() دارای دو آرگومان است و الگوی آن به صورت زیر است:

```
double ldexp (double num , int exp)
```

نتیجهٔ حاصل از تابع فوق عبارت $num * 2^{exp}$ است.

مثال ۱۱-۹

نمونه ای از کاربرد تابع ldexp().

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void  main()
{
    clrscr();
    printf("%.3f", ldexp(1, 2));
    getch();
}
```

تابع hypot()

تابع hypot() با اخذ دو ضلع قائم مثلث قائم‌الزاویه، وتر آن را محاسبه می‌کند و الگوی تابع به صورت زیر است:

```
double hypot (double x , double y)
```

در الگوی فوق، x و y دو ضلع قائم مثلث قائم‌الزاویه می‌باشند. به عنوان مثال، دستور

```
printf ("%f" , hypot (2,1)) ;
```

خروجی ۲/۲۳۶۰۶۸ را تولید می‌کند. این مقدار وتر مثلث قائم‌الزاویه است که دو ضلع قائم آن ۱ و ۲ هستند.

تابع log()

تابع log() لگاریتم طبیعی یک عدد مثبت را محاسبه می‌کند (پایه لگاریتم طبیعی عدد e=۲/۷۱۸۲۰۰۰ است) الگوی این تابع به صورت زیر است:

```
double log (double num)
```

۹-۱۲

برنامه‌ای که لگاریتم طبیعی اعداد ۱ تا ۱۰ را در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void  main()
{
    double val = 1.0;
    clrscr();
    do  {
      printf("%.3f %.3f\n", val, log(val));
      val ++;
    } while (val < 11.0);
    getch();
}
```

تابع log10()

تابع log10() لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می‌کند (برخلاف تابع log() که لگاریتم مبنای e اعداد مثبت را محاسبه می‌نماید). الگوی این تابع به صورت زیر است:

double log10 (double num)

مثال ۱۳-۹

برنامه‌ای که لگاریتم مبنای ۱۰ اعداد از ۱ تا ۱۰ را محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
    double val = 1.0;
    clrscr();
    do  {
      printf("%.3f %.3f\n", val, log10(val));
      val++;
    } while (val < 11.0);
    getch();
}
```

تابع modf()

تابع modf() یک عدد را به دو قسمت صحیح و اعشاری تجزیه می‌کند و الگوی آن به صورت زیر است:

double modf (double num , int *i)

قسمت اعشاری عدد num به عنوان نتیجه عمل تابع برگردانده می‌شود و قسمت صحیح آن در متغیر i قرار می‌گیرد.

مثال ۹-۱۴

برنامه‌ای که مقادیر ۱۰ و ۰/۱۲۳ را در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
int main(void)
{
   double fraction, integer;
   double number = 100000.567;
   clrscr();
   fraction = modf(number, &integer);
   printf("The whole and fractional parts of %lf are %lf and %lf\n",
        number, integer, fraction);
   getch();
   return 0;
}
```

تابع poly()

تابع poly() برای ارزیابی یک چندجمله‌ای به کار می‌رود و دارای الگوی زیر است:

```
double poly (double x , int n , double c [])
```

در الگوی فوق، n تعداد جمله، c آرایه‌ای است که ضرایب چندجمله‌ای را نگهداری می‌کند و x درجه چندجمله‌ای را مشخص می‌نماید. به عنوان مثال، اگر n=3 باشد، چندجمله‌ای که ارزیابی می‌شود به صورت زیر است:

```
c[3]x + c[2]x + c[1]x + c[0]
```

مثال ۹-۱۵

برنامه‌ای که چندجمله‌ای از درجه ۱ را با ۳ جمله محاسبه می‌کند و خروجی آن عدد ۶ است.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main() {
  double c[3];
  clrscr();
  c[0] = 3;
  c[1] = 2;
  c[2] = 1;
  printf("%f", poly(1, 2, c));
  getch();
}
```

تابع pow()

تابع pow() توانهای یک مبنا (base) را محاسبه می‌کند و الگوی آن به صورت زیر می‌باشد:

double pow (double base , double exp)

نتیجهٔ حاصل از تابع، عبارت $base^{exp}$ است. اگر مبنا صفر باشد و یا توان منفی یا صفر باشد این تابع عمل نخواهد کرد. همچنین اگر مبنا منفی باشد و توان (exp) از نوع صحیح نباشد نتیجه‌ای نخواهد داد.

مثال ۹-۱۶

برنامه‌ای که توانهای ۱ تا ۱۰ عدد ۱۰ را محاسبه و در خروجی چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main() {
 double x=10.0, y=0.0;
 clrscr();
 do {
   printf("%.2f", pow(x, y));
   y ++;
 } while(y < 11);
 getch();
}
```

تابع sin()

تابع sin() برای محاسبه سینوس یک زاویه برحسب رادیان به کار می‌رود و دارای آرگومان زیر است:

double sin (double arg)

مثال ۹-۱۷

برنامه‌ای که سینوس مقادیر ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main() {
  double val = -1.0;
  clrscr();
  do {
     printf("sine of %.2f is %.2f\n" , val, sin(val));
     val += 0.1;
  } while(val <= 1.0);
  getch();
}
```

تابع sinh()

تابع sinh() برای محاسبه سینوس هیپربولیک یک زاویه برحسب رادیان به کار رفته، الگوی آن به صورت زیر است :

double sinh (double arg)

مثال ۹-۱۸

برنامه‌ای که سینوس هیپربولیک مقادیر ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
  double val = -1.0;
  clrscr();
  do {
    printf("hyperbolic sine of %f is %f\n" , val, sinh(val));
    val += 0.1;
  } while(val <= 1.0);
  getch();
}
```

تابع sqrt()

تابع sqrt() جذر یک عدد مثبت را محاسبه می‌کند و الگوی آن به صورت زیر است :

double sqrt (double num)

دستور زیر عدد ۴ را به عنوان نتیجهٔ عمل در صفحه‌نمایش چاپ می‌کند :

```
printf ("%f",sqrt(16.0)) ;
```

تابع tan()

تابع tan() برای محاسبه تانژانت یک زاویه که برحسب رادیان می‌باشد به کار می‌رود و الگوی آن به صورت زیر است :

double tan (double arg)

مثال ۹-۱۹

برنامه‌ای که تانژانت مقادیر ۱- تا ۱ را با گام حرکت ۰/۱ محاسبه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
  double val = -1.0;
  clrscr();
  do {
    printf("tangent of %.2f is %.2f\n", val, tan(val));
    val += 0.1;
  } while(val <= 1.0);
  getch();
}
```

تابع tanh()

تابع tanh() برای محاسبه تانژانت هیپربولیک یک زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است :

double tanh (double arg)

مثال ۲۰-۹

برنامه‌ای که تانژانت هیپربولیک مقادیر ۱- تا ۱ را محاسبه و در صفحه‌نمایش چاپ می‌کند:

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main()
{
 double val = -1.0;
 clrscr();
 do {
    printf("HYPerbolic tangent of %f is %f\n" , val, tanh(val));
    val += 0.1;
 } while(val <= 1.0);
 getch();
}
```

توابع کاراکتری

توابع کاراکتری برای ورودی - خروجی کاراکترها و مقایسه آنها با یکدیگر، تبدیل آنها از حروف کوچک به بزرگ و برعکس و غیره مورد استفاده قرار می‌گیرند. الگوی این توابع در فایل ctype.h قرار دارد.

تابع isalnum()

تابع isalnum() کاراکتری را به عنوان آرگومان می‌گیرد؛ اگر این کاراکتر هیچکدام از حروف a تا z (A تا Z) و یا ارقام صفر تا ۹ نباشد صفر را به عنوان نتیجهٔ عمل برمی‌گرداند و در غیر این صورت نتیجهٔ برگردانده شده توسط این تابع، غیر از صفر خواهد بود. الگوی این تابع به صورت زیر است:

```
int isalnum (int ch)
```

مثال ۲۱-۹

برنامه‌ای که کاراکترهای خوانده شده از دستگاه صفحه‌کلید را بررسی می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch ==' ') break;
    if(isalnum(ch))
      printf("%c is alphanumeric\n", ch);
  }
  getch();
}
```

تابع isalpha()

تابع isalpha() کاراکتری را به عنوان آرگومان پذیرفته، تشخیص می‌دهد که این کاراکتر از حروف a تا z (و یا A تا Z) هست یا خیر. اگر این کاراکتر یکی از این حروف نباشند نتیجه حاصل از تابع، عدد صفر و در غیر این صورت صفر نیست. الگوی این تابع به صورت زیر است:

```
int isalpha (int ch)
```

مثال ۲۲-۹

برنامه‌ای که کاراکترهای خوانده شده از ورودی را بررسی کرده، نتیجه را گزارش می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
```

```
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch ==' ') break;
     if(isalpha(ch))
     printf("%c is letter\n", ch);
  }
  getch();
}
```

تابع isascii()

این تابع کاراکتر را به عنوان آرگومان پذیرفته، تشخیص می‌دهد که آیا این کاراکتر در بازهٔ صفر تا ox7f هست یا خیر. اگر چنین نباشد نتیجه حاصل از تابع برابر با صفر وگرنه غیر از صفر خواهد بود. الگوی این تابع به صورت زیر است:

```
int isascii (int ch)
```

مثال ۲۳-۹

برنامه‌ای که کاراکترهای خوانده شده از ورودی را با تابع isascii() بررسی کرده نتیجه را گزارش می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch==' ') break;
     if(isascii(ch))
       printf("%c is ASCII defined\n", ch);
  }
  getch();
}
```

تابع ()iscntrl

تابع ()iscntrl کاراکتری را به عنوان آرگومان پذیرفته و تشخیص می‌دهد که آیا این کاراکتر در بازه صفر تا 1f (در مبنای ۱۶) و صفر تا ۳۱ (در مبنای ۱۰) هست یا خیر. اگر در این بازه نباشد نتیجهٔ حاصل از تابع برابر با صفر و در غیر این صورت، غیر صفر خواهد بود. الگوی تابع به صورت زیر است:

```
int    iscntrl (int ch)
```

مثال ۹-۲۴

برنامه‌ای که کاراکتری را از ورودی گرفته، تشخیص می‌دهد که آیا این کاراکتر در بازه صفر تا 1f است یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch == ' ') break;
    if(iscntrl(ch))
        printf("%c is a control character\n", ch);
  }
  getch();
}
```

تابع ()isdigit

تابع ()isdigit کاراکتری را به عنوان آرگومان پذیرفته، تشخیص می‌دهد که آیا این کاراکتر یکی از ارقام صفر تا ۹ هست یا خیر. اگر این کاراکتر یکی از ارقام صفر تا ۹ نباشد نتیجهٔ حاصل از تابع صفر وگرنه غیر از صفر خواهد بود. الگوی تابع به صورت زیر است:

```
int isdigit (int ch)
```

مثال ۹-۲۵

برنامه‌ای که کاراکتری را از ورودی خوانده، تشخیص می‌دهد که آیا این کاراکتر یکی از ارقام صفر تا ۹ هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
```

```
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch == ' ') break;
    if(isdigit(ch)) printf("%c is a digit\n", ch);
  }
  clrscr();
}
```

تابع isgraph()

تابع isgraph() کاراکتری را به عنوان آرگومان پذیرفته تشخیص می‌دهد که آیا این کاراکتر یکی از کاراکترهای قابل چاپ هست یا خیر. کاراکترهای قابل چاپ در بازهٔ 21 تا 7E در مبنای ۱۶ و (در بازه ۳۳ تا ۱۲۶ در مبنای ۱۰) هستند. اگر این کاراکتر از کاراکترهای قابل چاپ نباشد حاصل کار تابع، عدد صفر وگرنه یک خواهد بود. الگوی تابع به صورت زیر است:

```
int  isgraph (int ch)
```

مثال ۲۶-۹

برنامه‌ای که کاراکترهایی را از ورودی خوانده، تشخیص می‌دهد که آیا این کاراکتر قابل چاپ هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if( ch== ' ') break;
    if(isgraph(ch)) printf("%c is a printing character\n", ch);
  }
}
```

تابع islower()

تابع islower() کاراکتری را به‌عنوان آرگومان گرفته، تشخیص می‌دهد که آیا این کاراکتر یکی از حروف کوچک

انگلیسی (a تا z) هست یا خیر. اگر کاراکتر موردنظر یکی از حروف کوچک انگلیسی نباشد حاصل کار تابع صفر وگرنه غیر از صفر خواهد بود. الگوی این تابع به صورت زیر است:

```
int  islower (int ch)
```

مثال ۹-۲۷

برنامه‌ای که کاراکتری را از ورودی خوانده تشخیص می‌دهد که آیا این کاراکتر از حروف کوچک انگلیسی هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch == ' ') break;
    if(islower(ch)) printf("%c is lowercase\n", ch);
  }
}
```

تابع ()isprint

تابع ()isprint کاراکتری را به عنوان آرگومان پذیرفته تشخیص می‌دهد که آیا این کاراکتر، قابل چاپ هست یا خیر، تفاوت این تابع با تابع ()isprint در این است که تابع isprint فضای خالی (blank) را به عنوان کاراکتر قابل چاپ قبول دارد. اگر کاراکتری که توسط این تابع بررسی می‌شود، قابل چاپ نباشد؛ حاصل کار تابع، صفر وگرنه غیر از صفر خواهد بود. الگوی تابع به صورت زیر است:

```
int isprint (int ch)
```

مثال ۹-۲۸

برنامه‌ای که کاراکتری را از ورودی خوانده تشخیص می‌دهد که آیا این کاراکتر قابل چاپ هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
```

```
  for(;;) {
    ch = getchar();
    if(ch==' ') break;
    if(isprint(ch)) printf("%c is printable\n", ch);
  }
}
```

تابع ispunct()

تابع ispunct() کاراکتری را به عنوان آرگومان پذیرفته تشخیص می‌دهد که آیا این کاراکتر یکی از کاراکترهای ویرایش مثل کاما، نقطه و غیره هست یا خیر. اگر نباشد حاصل کار تابع، عدد صفر وگرنه غیر از صفر خواهد بود. الگوی تابع به صورت زیر است:

int Ispunct (Int ch)

مثال ۲۹-۹

برنامه‌ای که کاراکتری را از ورودی خوانده تشخیص می‌دهد که آیا این کاراکتر از کاراکترهای ویرایشی هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch==' ') break;
    if(ispunct(ch)) printf("%c is punctuation", ch);
  }
}
```

تابع isspace()

تابع isspace() کاراکتری را به عنوان آرگومان پذیرفته مشخص می‌کند که آیا این کاراکتر یکی از کاراکترهای فضای خالی (blank)، horizontal tab، vertical tab، form feed، carriage return یا new line هست یا خیر. اگر نباشد، حاصل کار تابع صفر وگرنه غیر از صفر خواهد بود. الگوی تابع به صورت زیر است:

int Isspace (Int ch)

مثال ۳۰-۹

برنامه‌ای که کاراکتری را از ورودی خوانده سپس تابع ()isspace را بر روی آن عمل می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getch();
    if(ch==';') break;
    if(isspace(ch)) printf("%c is space\n", ch);
  }
}
```

تابع ()isupper

تابع ()isupper کاراکتری را به عنوان آرگومان پذیرفته تشخیص می‌دهد که آیا این کاراکتر از حروف بزرگ انگلیسی (A تا Z) هست یا خیر؛ اگر نباشد حاصل کار تابع عدد صفر وگرنه غیر از صفر خواهد بود. الگوی تابع به صورت زیر است:

int isupper (int ch)

مثال ۳۱-۹

برنامه‌ای که کاراکتری را از ورودی خوانده تشخیص می‌دهد که آیا این کاراکتر از حروف بزرگ انگلیسی هست یا خیر.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main()
{
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch== ' ') break;
    if(isupper(ch)) printf("%c is uppercase\n", ch);
  }
}
```

تابع ()isxdigit

تابع ()isxdigit کاراکتری را به عنوان آرگومان پذیرفته و تشخیص می‌دهد که آیا این کاراکتر یکی از ارقام مبنای ۱۶ است یا خیر؛ ارقام مبنای ۱۶ در بازهٔ صفر تا ۹ یا a تا f و یا A تا F هستند. اگر این کاراکتر، یکی از این ارقام نباشد حاصل کار تابع صفر وگرنه غیرصفر خواهد بود. الگوی این تابع به صورت زیر است:

```
int isxdigit (int ch)
```

مثال ۳۲-۹

برنامه‌ای که کاراکترهایی را از ورودی خوانده و ارقام مبنای ۱۶ را تشخیص می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
void main() {
  char ch;
  clrscr();
  for(;;) {
    ch = getchar();
    if(ch==' ') break;
    if(isxdigit(ch)) printf("%c is hexadecimal\n", ch);
  }
}
```

تابع ()tolower

تابع ()tolower کاراکتری را به عنوان آرگومان پذیرفته و آن را به حروف کوچک انگلیسی تبدیل می‌کند. اگر کاراکتر به صورت حروف کوچک باشد، تغییری در کاراکتر ایجاد نمی‌شود؛ الگوی تابع به صورت زیر است:

```
int tolower (int ch)
```

دستور زیر حرف q را در خروجی چاپ می‌کند:

```
putchar (tolower ('Q')) ;
```

تابع ()toupper

تابع ()toupper کاراکتری را به عنوان آرگومان پذیرفته و آن را به حروف بزرگ انگلیسی تبدیل می‌کند. اگر کاراکتر موردنظر یکی از حروف بزرگ باشد تغییری در آن ایجاد نمی‌شود. الگوی این تابع به صورت زیر است:

```
int toupper (int ch)
```

دستور زیر حرف A را در خروجی چاپ می‌کند:

```
putchar (toupper ('a')) ;
```

توابع رشته‌ای

بعضی از توابع کتابخانه‌ای رشته‌ای، برحسب نیاز در فصل ۵ بررسی شدند. اما در این فصل به بررسی کلیهٔ توابع رشته‌ای پرداخته می‌شود. توابع رشته‌ای در زبان C از تنوع خاصی برخوردارند. این توابع معمولاً برای مقایسه رشته‌ها، پیداکردن کاراکتر و یا رشته‌ای در رشته دیگر و غیره به کار می‌روند. الگوی توابع رشته‌ای در فایل "string.h" قرار دارد.

تابع ()memchr

تابع ()memchr کاراکتری را در یک آرایه مورد جستجو قرار می‌دهد و آدرس اولین وقوع آن را مشخص می‌کند. اگر این کاراکتر پیدا شد آدرس آن محل را در یک اشاره‌گر کاراکتری قرار می‌دهد وگرنه کاراکتر تهی (\0) را در اشاره‌گر منظور می‌کند. الگوی تابع به صورت زیر است:

```
void *memchr (const void  *buffer , int ch , unsigned count)
```

در الگوی فوق buffer به آرایه‌ای اشاره می‌کند که عمل جستجو باید در آن انجام شود و ch کاراکتری را مشخص می‌کند که باید در آرایه مورد جستجو قرار گیرد. count محلی را در آرایه مشخص می‌کند که عمل جستجو باید از ابتدای آرایه تا آن محل انجام شود.

مثال ۹-۳۲

برنامه‌ای که محل اولین وقوع کاراکتر 'a' را پیدا می‌کند و در اشاره‌گر p قرار می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main()
{
  void *p;
  clrscr();
  p = memchr("this is a test", 'a', 14);
  printf((char *) p);
  getch();
}
```

تابع ()memcmp

تابع ()memcmp قسمتی از یک آرایه را با قسمتی از آرایهٔ دیگر مقایسه می‌کند؛ الگوی این تابع به صورت زیر است:

```
int memcmp (const void  *buf1, const void  *buf2, unsigned count)
```

اشاره‌گرهای buf1 و buf2 به ترتیب به اولین و دومین آرایه‌ای که قسمتی از آنها باید مورد مقایسه قرار گیرند اشاره

می‌کنند و count مشخص می‌کند که چه طولی از آرایه‌ها باید با هم مقایسه می‌شوند. مثلاً اگر count برابر با ۵ باشد، یعنی ۵ محل اول دو آرایه باید با هم مقایسه شوند؛ نتیجه حاصل از تابع یک مقدار عددی "صحیح" است که به صورت زیر تفسیر می‌شود.

مقداری که توسط تابع برگردانده می‌شود	مفهوم این مقدار
کوچکتر از صفر	buf1 < buf2
صفر	buf1 = buf2
بزرگتر از صفر	buf1 > buf2

مثال ۳۴-۹

برنامه‌ای که با استفاده از تابع ()memcmp قسمتهایی از دو آرایه را مقایسه می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main(int argc, char *argv[])
 {
   int outcome, len, l1, l2;
   clrscr();
   /* find the length of shortest */
   len = (l1 = strlen(argv[1])) < (l2 = strlen(argv[2])) ? l1:l2;
   outcome = memcmp(argv[1], argv[2], len);
   if(!outcome) printf("equal");
   else if(outcome < 0) printf("first less than second");
   else printf("first greater than second");
   getch();
}
```

تابع ()memcpy

تابع ()memcpy قسمتی از یک آرایه را در آرایه دیگر کپی می‌کند. الگوی این تابع به صورت زیر است:

void *memcpy (void *to, const void *from, unsigned count)

در الگوی فوق، from اشاره‌گری است که به آرایه منبع (آرایه‌ای که کاراکترهای موردنظر در آنجا قرار دارند) اشاره می‌کند و to اشاره‌گری است که به آرایه مقصد (آرایه‌ای که کاراکترها باید به آنجا کپی شوند) اشاره می‌کند. تعداد کاراکترهایی که باید کپی شوند توسط آرگومان count مشخص می‌شود. به عنوان مثال، اگر count برابر با ۱۰ باشد یعنی اولین ۱۰ کاراکتر آرایه from باید به ده محل اول آرایه to کپی شوند. همانطوری که از الگوی این تابع پیداست این تابع یک اشاره‌گر است که پس از مقایسه، به آرایه‌ای اشاره می‌کند که to به آن اشاره می‌کرده است.

مثال ۳۵-۹

برنامه‌ای که با استفاده از تابع ()memcpy قسمتی از آرایه را در آرایه دیگر کپی می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#define SIZE 51
void main() {
   char buf1[SIZE], buf2[SIZE];
   clrscr();
   strcpy(buf1, "when, in the course of ...");
   memcpy(buf2, buf1, SIZE);
   printf(buf2);
   getch();
}
```

تابع ()memmove

تابع ()memmove تعدادی از کاراکترها را از یک آرایه به آرایه دیگر کپی می‌کند. تفاوت آن با تابع ()memcpy در این است که اگر در تابع ()memcpy دو آرایه رویهم قرار گرفته باشند (overlap)، عمل کپی انجام نمی‌شود ولی در تابع ()memmove عمل کپی به درستی انجام می‌شود. الگوی این تابع به صورت زیر است:

void *memmove(void *to, const void *from, unsigned count)

آرایه‌ای که from به آن اشاره می‌کند آرایهٔ منبع و آرایه‌ای که to به آن اشاره می‌کند آرایه مقصد است. count مشخص می‌کند که چه تعداد کاراکتر از آرایه منبع به آرایه مقصد کپی شوند.

مثال ۳۶-۹

برنامه‌ای که محتویات آرایه st1 را به آرایه st2 کپی می‌کند و نتیجه را به خروجی می‌برد.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#define SIZE 50
void main() {
   char str1[SIZE], str2[SIZE];
   clrscr();
   strcpy(str1, "when, in the course of ...");
   memmove(str2, str1, SIZE);
   printf(str2);
   getch();
}
```

تابع memset()

تابع memset() کاراکتری را در چند عنصر آرایه کپی می‌کند. الگوی این تابع به صورت زیر است:

```
void *memset (void  *buf, int ch, unsigned count)
```

در الگوی فوق، اشاره‌گر buf به آرایه‌ای اشاره می‌کند که عمل کپی باید در آن انجام شود و ch به کاراکتری اشاره می‌کند که باید در آرایه کپی شود و count مشخص می‌کند که کاراکتر ch باید در چند عنصر آرایه (با شروع از اولین محل) کپی گردد. این تابع معمولاً برای ارزش‌دهی اولیه آرایه‌ها استفاده می‌شود.

مثال ۹-۳۷

دستورات زیر ۵۰ عنصر از آرایه‌ای را که buf به آن اشاره می‌کند، برابر با کاراکتر تهی (\0) قرار می‌دهد و سپس ۱۰ عنصر اول آرایه را برابر با x قرار داده و چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#define SIZE 50
void main()
{
   char buf[SIZE];
   clrscr();
   memset(buf,'\0', 49);
   memset(buf,'x', 10);
   printf(buf);
   getch();
}
```

تابع strcspn()

تابع strcspn() رشته‌ای را در یک رشته دیگر جستجو می‌کند و اولین محلی از این رشته را که حتی یکی از کاراکترهای رشتهٔ مورد جستجو در آن محل باشد، به عنوان نتیجهٔ عمل برمی‌گرداند. الگوی این تابع به صورت زیر است:

```
int strcspn (const  char *st1, const  char str2)
```

با توجه به الگوی تابع strcspn()، str2 در str1 جستجو می‌شود و محل اولین کاراکتر در رشتهٔ str1 که با هر کدام از کاراکترهای str2 برابر باشد، به عنوان نتیجهٔ عمل تابع برگردانده می‌شود.

مثال ۹-۳۸

نمونه‌ای از کاربرد تابع strcspn().

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main()
{
  char s[80];
  clrscr();
  strcpy(s, "THIS IS A TEST");
  printf("%d", strcspn(s, "IS"));
  getch();
}
```

تابع strerror()

تابع strerror() موجب می‌شود تا بتوان در اثر بروز خطایی در برنامه، پیام خطایی را صادر کرد. نوع پیام خطا در اختیار برنامه‌نویس است. الگوی این تابع به صورت زیر است:

```
char  *strerror (char *str)
```

به عنوان مثال، به تابع زیر توجه نمایید:

```
swap ()
  {
  .
  .
  .
    if (error) printf (strerror ("error in swap")) ;
  .
  .
  .
  }
```

تابع strlwr()

تابع strlwr() رشته‌ای را به عنوان آرگومان پذیرفته و کلیهٔ حروف بزرگ آن را به کوچک تبدیل می‌کند و دارای الگوی زیر است:

```
char  *strlwr (char  *str)
```

در الگوی فوق، str به رشته‌ای اشاره می‌کند که کلیهٔ حروف بزرگ موجود در آن باید به حروف کوچک تبدیل گردند.

مثال ۳۹-۹

برنامه‌ای که چگونگی استفاده از تابع strlwr() را نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main()
{
```

```
    char s[50];
    clrscr();
    strcpy(s, "THIS IS A TEST");
    strlwr(s);
    printf(s);
    getch();
}
```

تابع strncat()

تابع strncat() قسمتی از یک رشته را به انتهای رشتهٔ دیگر الحاق (concate) می‌کند و سپس رشتهٔ نتیجه را به کاراکتر تهی (\0) ختم می‌نماید. الگوی تابع به صورت زیر است:

```
char  *strncat (char  *str1, const char *str2, unsigned int count)
```

در این الگو، count تعداد کاراکترهایی را مشخص می‌کند که از رشتهٔ str2 باید به رشتهٔ str1 الحاق شود.

مثال ۴۰-۹

برنامه‌ای که چگونگی استفاده از تابع strncat() را نشان می‌دهد. اگر در پاسخ به درخواست سیستم، به جای s1 عبارت hello و به جای s2 عبارت there وارد شود؛ نتیجهٔ حاصل از اجرای برنامه، رشتهٔ "there hello" است.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main()
{
  char s1[80], s2[80];
  unsigned int len;
  clrscr();
  printf("enter string1:");
  gets(s1);
  printf("enter string2:");
  gets(s2);
  len = 79 - strlen(s2);
  strncat(s2, s1, len);
  printf(s2);
  getch();
}
```

تابع strncmp()

تابع strncmp() تعداد مشخصی از کاراکترهای دو رشته را با یکدیگر مقایسه می‌کند. (زیررشته‌ای از یک رشته را با

زیررشته‌ای از رشتهٔ دیگر مقایسه می‌نماید). الگوی این تابع به صورت زیر است:

int strncmp (const char *st1, const char *st2, unsigned count)

در این الگو، به تعداد count کاراکتر، از دو رشتهٔ st1 و st2 با شروع از ابتدای رشته‌ها با یکدیگر مقایسه شده، یک عدد "صحیح" به عنوان نتیجهٔ عمل برگردانده می‌شود. مقادیری که توسط این تابع برگردانده می‌شوند عبارتند از:

مقداری که توسط تابع برگردانده می‌شود	مفهوم این مقدار
کوچکتر از صفر	str1 < str2
صفر	str1 = str2
بزرگتر از صفر	str1 > str2

اگر تعداد کاراکترهایی که در دو رشته وجود دارند کمتر از count باشند، عمل مقایسه پس از رسیدن به اولین کاراکتر تهی (\0) خاتمه پیدا می‌کند.

مثال ۹-۴۱

برنامه‌ای که چگونگی استفاده از تابع ()strncmp را نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main(int argc, char *argv[])
{
  clrscr();
  if(!strncmp(argv[1], argv[2], 8))
    printf("the file names are same");
  getch();
}
```

تابع ()strncpy

تابع ()strncpy تعداد مشخصی از کاراکترهای یک رشته را در یک رشته دیگر کپی می‌کند. الگوی این تابع به صورت زیر است:

char *strncpy(char *str1, const char *str2, unsigned count)

در الگوی فوق، به تعداد count کاراکتر از رشتهٔ str2 در رشته str1 با شروع از ابتدای رشته‌ها کپی می‌شوند. اگر تعداد کاراکترهایی که در str2 وجود دارد کمتر از مقدار count باشد، به تعداد لازم کاراکتر تهی (\0) در انتهای str1 کپی می‌شوند.

مثال ۹-۴۲

برنامه‌ای که چگونگی استفاده از تابع ()strncpy را نشان می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
void main()
{
  char str1[50], str2[50];
  clrscr();
  gets(str1);
  strncpy(str2, str1, 7);
  puts(str2);
  getch();
}
```

تابع strnset()

تابع strnset() می‌تواند یک کاراکتر را به تعداد دفعات مشخصی در یک رشته کپی کند. الگوی این تابع به صورت زیر است:

char *strnset (char *str, char ch, signed count)

در الگوی فوق، str به رشته‌ای اشاره می‌کند که این تابع باید در آن رشته عمل نماید؛ ch کاراکتری است که به تعداد count بار باید در رشته موردنظر کپی شود. به عنوان مثال، دستور strnset (str, 'X',10); ، ۱۰ محل اول رشته str را برابر با x قرار می‌دهد.

تابع strpbrk()

تابع strpbrk() تقریباً مشابه تابع strcspn() عمل می‌کند. الگوی تابع به صورت زیر است:

char *strpbrk (const char *str1, const char *str2)

در الگوی فوق، رشته str2 باید در رشتهٔ str1 جستجو شود؛ به این صورت که اگر هر کدام از کاراکترهای موجود در رشته str2 با یکی از کاراکترهای رشته str1 مطابقت داشته باشد (برابر باشد)، آدرس این محل از رشتهٔ str1، در یک اشاره‌گر قرار می‌گیرد.

مثال ۹-۴۳

نمونه‌ای از کاربرد تابع strpbrk().

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main(void)
{
  char *string1 = "abcdefghijklmnopqrstuvwxyz";
  char *string2 = "onm";
```

```
    char *ptr;
    clrscr();
    ptr = strpbrk(string1, string2);
    if (ptr)
       printf("strpbrk found first character: %c\n", *ptr);
    else
       printf("strpbrk didn't find character in set\n");
    getch();
    return 0;
}
```

تابع strrchr()

تابع strrchr() کاراکتری را در یک رشته جستجو می‌کند و محل آخرین وقوع این کاراکتر را در رشته پیدا کرده به یک اشاره‌گر نسبت می‌دهد. اگر این کاراکتر پیدا نشود کاراکتر تهی (\0) در اشاره‌گر قرار خواهد گرفت. الگوی این تابع به صورت زیر است:

```
char *strrchr (const char  *str, int ch)
```

مثال ۴۴-۹

برنامه‌ای که چگونگی استفاده از تابع strrchr() را نشان می‌دهد.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main(void)
{
   char string[15];
   char *ptr, c = 'r';
   clrscr();
   strcpy(string, "This is a string");
   ptr = strrchr(string, c);
   if (ptr)
      printf("The character %c is at position: %d\n", c, ptr-string);
   else
      printf("The character was not found\n");
   getch();
   return 0;
}
```

تابع strspn()

تابع strspn() رشته‌ای رادر یک رشتهٔ دیگر جستجو می‌کند و طول اولین بخشی از رشته str2 راکه رشته str1 در

آن وجود دارد، برمی‌گرداند.

```
unsigned strspn (const char  *str1, const char  *str2)
```

مثال ۹-۴۵

برنامه‌ای که چگونگی استفاده از تابع ()strspn را نشان می‌دهد.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{
  int len;
  clrscr();
  len = strspn("this is a test", "siht");
  printf("%d", len);
  getch();
}
```

تابع ()strrev

تابع ()strrev کاراکترهای یک رشته را معکوس می‌کند. یعنی کاراکتر ابتدا را به انتهای آن منتقل می‌کند و این عمل را برای کلیهٔ کاراکترها انجام می‌دهد.

مثال ۹-۴۶

برنامه‌ای که چگونگی استفاده از تابع ()strrev را نشان می‌دهد.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{
  char s[] = "hello";
  clrscr();
  strrev(s);
  printf(s);
  getch();
}
```

تابع ()strset

تابع ()strset محتویات یک رشته را با یک کاراکتر پر می‌کند. الگوی این تابع به صورت زیر است:

```
char  *strset (char  *str, char ch)
```

در این الگو، str به رشته‌ای اشاره می‌کند که محتویات آن باید با کاراکتر ch پر شود. به عنوان مثال، دستور زیر رشته str را با x پر می‌کند.

```
strset (str,'x') ;
```

تابع strtok()

تابع strtok() نشانه‌های موجود در یک رشته را مشخص می‌کند. الگوی تابع به صورت زیر می‌باشد:

```
char  *strtok (char *str1, const char  *str2)
```

در الگوی فوق، str1 رشته‌ای است که نشانه‌های موجود در آن باید جدا شوند و str2 رشته‌ای است که جداکننده‌ها (delimiters) را مشخص می‌کند؛ یعنی مشخص می‌کند که نشانه‌ها با چه کاراکترهایی از هم جدا شده‌اند. این تابع برای تجزیهٔ رشته‌ها بسیار مفید است.

مثال ۹-۴۷

در برنامهٔ زیر، رشته‌ای توسط str تعریف شده که نشانه‌های موجود در آن با کاما و فضای خالی (blank) از هم جدا شده‌اند (این دو کاراکتر توسط str2 مشخص شده‌اند).

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main()
{
  char *p;
  clrscr();
  p = strtok("the summer soldier, the sunshine patriot"," ");
  printf(p);
  do {
     p = strtok('\0',", ");
     if(p) printf("| %s",p);
  } while(p);
  getch();
}
```

تابع struper()

تابع struper() در یک رشتهٔ کاراکتری، کلیهٔ حروف کوچک را به حروف بزرگ تبدیل می‌کند. الگوی تابع به صورت زیر است:

```
char  *struper (char  *str)
```

در الگوی فوق str به رشته‌ای اشاره می‌کند که حروف کوچک موجود در آن باید به حروف بزرگ تبدیل شوند.

مثال ۹-۴۸

برنامه‌ای که کاربرد تابع ()struper را نشان می‌دهد.

```
#include <string.h>
#include <conio.h>
#include <stlib.h>
void main()
{
  char s[80];
  clrscr();
  strcpy(s,"this is a test");
  strupr(s);
  printf(s);
  getch();
}
```

توابع تخصیص حافظه پویا

قبل از این که به بررسی توابع تخصیص حافظه پویا بپردازیم، بهتر است با مفهوم **تخصیص حافظه پویا** آشنا شویم. گاهی لازم است در حین اجرای برنامه (نه شروع اجرای برنامه) از سیستم حافظه اخذ گردد و در موقع مناسبی این حافظه به سیستم برگردانده شود. به چنین روش اخذ حافظه، "**تخصیص حافظه پویا**" گفته می‌شود. در C توابعی وجود دارند که امکان اخذ و یا آزاد کردن حافظه را فراهم می‌کنند که در این فصل به بررسی آنها می‌پردازیم.

تابع ()calloc

تابع ()calloc برای اخذ حافظه از سیستم در حین اجرای برنامه به کار رفته و دارای الگوی زیر است:

void *calloc (unsigned num, unsigned size)

الگوی این تابع در فایل alloc.h قرار دارد. مقدار حافظه‌ای که تابع ()calloc در اختیار برنامه نویس قرار می‌دهد، برابر با size*num است؛ لذا از این تابع معمولاً جهت اخذ حافظهٔ لازم برای یک آرایه (شامل num عنصر که هر عنصر آن دارای طولی برابر با size باشد) استفاده می‌گردد. آدرس اولین بایت حافظه‌ای که توسط این تابع تخصیص می‌یابد در یک اشاره‌گر قرار داده می‌شود. اگر این اشاره‌گر تهی (NULL) باشد به معنی تخصیص نیافتن این حافظه است (عدم تخصیص حافظه ممکن است به دلیل نبودن حافظهٔ خالی به اندازهٔ موردنیاز باشد). لذا توصیه می‌شود که پس از استفاده از تابع ()calloc حتماً اشاره‌گر تست شود تا از تخصیص حافظه، اطمینان حاصل گردد.

مثال ۹-۴۹

برنامه‌ای که در حین اجرا، یک آرایهٔ ۱۰۰ عنصری از نوع float را ایجاد کرده آدرس آن را در اشاره‌گر p قرار می‌دهد.

```
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
void main() {
  float *p;
  clrscr();
  p = (float *) calloc(100, sizeof(float));
  if(!p) {
    printf("allocation failure - aborting");
    exit(1);
  }
}
```

تابع malloc()

تابع malloc() برای اخذ حافظه از سیستم، در حین اجرای برنامه به کار رفته و دارای الگوی زیر است :

void *malloc (unsigned size)

الگوی این تابع در فایل alloc.h قرار دارد. میزان حافظه‌ای که توسط این تابع از سیستم اخذ می‌شود توسط آرگومان size مشخص می‌گردد. آدرس حافظه در یک اشاره‌گر قرار می‌گیرد. اگر این اشاره‌گر تهی (\0) باشد بدین معنی است که حافظهٔ لازم، تخصیص نیافته است.

مثال ۵۰-۹

برنامه‌ای که ۱۰ بایت از حافظه را تخصیص می‌دهد.

```
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
int main(void) {
   char *str;
   clrscr();
   /* allocate memory for string */
   str = (char *) malloc(sizeof(char) * 10);
   if (!str) {
      printf("Not enough memory to allocate buffer\n");
      getch();
      exit(1);
   }
   strcpy(str, "Hello");
   printf("String is %s\n", str);
   getch();
   return 0;
}
```

تابع realloc()

تابع realloc() برای تغییر میزان حافظهٔ اختصاص یافته توسط توابع تخصیص حافظه، مثل malloc() به کار می‌رود. الگوی این تابع به صورت زیر است:

```
*realloc (void *ptr, unsigned size)
```

الگوی این تابع در فایل "stdio.h" قرار دارد. برای تغییر میزان حافظهٔ تخصیص یافته، باید اشاره‌گر به آن حافظه و اندازهٔ جدید این حافظه به عنوان آرگومان تابع realloc() ذکر شود. میزان جدید حافظه ممکن است کمتر و یا بیشتر از حافظهٔ تخصیص یافته باشد. در صورت وجود اطلاعات در حافظهٔ تخصیص یافتهٔ قبلی، آنها از بین نخواهند رفت.

تابع realloc() پس از اخذ حافظه از سیستم، آدرس آن را در یک اشاره‌گر قرار می‌دهد. اگر این اشاره‌گر تهی (NULL) باشد، بدین معنی است که حافظه، تخصیص نیافته است.

تابع free()

تابع free() موجب می‌شود تا حافظهٔ اخذ شده ازسیستم، به آن برگردانده شود. این تابع دارای الگوی زیر است:

```
void free (void *)
```

برای برگشت حافظه به سیستم کافی است اشاره‌گر آن محل از حافظه، به تابع free() داده شود. الگوی تابع در فایل stdlib.h قرار دارد. توصیه می‌شود جهت سرعت انجام کار توسط سیستم عامل، حافظه‌ای را که مورد استفاده قرار گرفته و دیگر کاری با آن نیست، توسط این تابع به سیستم برگردانده شود.

مثال ۵۱-۹

برنامه‌ای که حافظه‌ای به اندازه ۱۷ کاراکتر را از سیستم اخذ کرده و رشتهٔ "this is 16 chars" را در این محل قرار می‌دهد؛ سپس طول این رشته به ۱۸ کاراکتر افزایش پیدا می‌کند تا کاراکتر نقطه (.) به انتهای آن اضافه شود.

```
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
void main()
{
  char *p;
  clrscr();
  p = (char *) malloc(sizeof(char) * 17);
  if(!p){
       printf("allocation error - aborting");
       getch();
       exit(1);
   }
   strcpy(p, "this is 16 chars");
```

```
    p = (char *) realloc(p, 18);
    if(!p) {
        printf("allocation error - aborting");
        getch();
        exit(1);
    }
    strcat(p, ".");
    printf(p);
    free(p);
    getch();
}
```

توابعی در مورد فایل‌ها و فهرستها

تعدادی توابع در بورلند C وجود دارند که برای ایجاد، حذف و سایر اعمال با فهرستهای روی دیسک به کار می‌روند. کلیه در این قسمت بررسی می‌شوند در فایل dir.h قرار دارند.

تابع mkdir()

تابع mkdir() برای ایجاد یک فهرست به کار می‌رود. الگوی تابع به صورت زیر است:

```
int  mkdir (char  *path)
```

در الگوی فوق، path به رشته‌ای اشاره می‌کند که نام فهرست را مشخص می‌نماید. اگر عمل این تابع با موفقیت انجام شود، مقداری که توسط آن برگردانده می‌شود برابر با صفر و در غیر این صورت برابر با ۱-است.

مثال ۵۲-۹

نتیجهٔ حاصل از اجرای برنامه زیر، ایجاد یک فهرست به نام "DOS" روی درایو جاری است.

```
#include <conio.h>
#include <stdio.h>
#include <dir.h>
void main()
{
   clrscr();
   if(! mkdir("dos"))
       printf("directory created.") ;
   else
       printf("directory  not created.") ;
   getch();
}
```

تابع chdir()

تابع chdir() موجب تغییر مسیر به یک فهرست موجود روی دیسک می‌شود. الگوی تابع به صورت زیر است :

```
int  chdir (char  *path)
```

در الگوی فوق، path رشته‌ای است که نام فهرست موجود روی دیسک را مشخص می‌کند. این رشته ممکن است شامل نام درایو نیز باشند. اگر عمل این تابع با موفقیت انجام شود حاصل کار تابع عدد صفر و در غیر این صورت عدد ۱- خواهد بود.

مثال ۹-۵۳

اجرای برنامه زیر موجب تغییر مسیر به C:\WP\DOS می‌شود. برای معرفی کاراکتر \ به کامپایلر باید از کاراکتر کنترلی \ استفاده شود. به عنوان مثال، \\ موجب معرفی کاراکتر \ به کامپایلر می‌شود.

```
#include <conio.h>
#include <stdio.h>
#include <dir.h>
void main() {
    clrscr();
    if(!chdir("c:\\wp\\dos"))
      printf("done.");
    else
      printf("not done.");
    getch();
}
```

تابع rmdir()

تابع rmdir() برای حذف فهرست موجود روی دیسک به کار می‌رود. الگوی تابع به صورت زیر است :

```
int  rmdir (char *path)
```

در این الگو، path نام فهرستی است که باید حذف شود. این فهرست اولاً باید خالی باشد، ثانیاً نباید فهرست جاری باشد. اگر عمل تابع با موفقیت انجام شود حاصل کار تابع عدد صفر وگرنه ۱- خواهد بود.

مثال ۹-۵۴

برنامه‌ای که فهرست DOS را حذف می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <dir.h>
main() {
  clrscr();
  if(! rmdir("DOS"))
    printf(" 'DOS' directory removed") ;
  else
    printf(" 'DOS' directory not removed") ;
  getch();
}
```

توابع findfirst() و findnext()

تابع findfirst() برای جستجوی یک فایل بر روی دیسک مورد استفاده قرار می‌گیرد. الگوی این تابع به صورت زیر است:

```
int findfirst (char *fname, struct ffblk  *p, int sttrib)
```

الگوی این تابع در فایل dir.h قرار دارد؛ اما این تابع به ماکروهایی نیاز دارد که در فایل dos.h تعریف شده‌اند. لذا در برنامه‌ای که از این تابع استفاده می‌گردد دو فایل dir.h و dos.h باید توسط دستور #include معرفی شوند.

در الگوی تابع findfirst() پارامتر fname به رشته‌ای اشاره می‌کند که حاوی نام فایل است. نام فایل می‌تواند به همراه نام درایو و مسیری باشد که باید در آنجا مورد جستجو قرار گیرد. اگر نام درایو و مسیر ذکر نشوند، این فایل در مسیر جاری مورد جستجو قرار خواهد گرفت. اگر نام فایل به همراه کاراکترهای عمومی (*,?) باشد اولین فایلی که دارای این مشخصه باشد توسط تابع findfirst() پیدا می‌شود. به عنوان مثال، اگر A:\DOS*.COM رشته‌ای باشد که پارامتر fname به آن اشاره می‌کند، تابع findfirst() بر روی درایو A و در مسیر DOS اولین فایلی را که دارای پسوند com است پیدا می‌کند.

اگر فایلِ مورد جستجو پیدا گردد حاصل کار تابع، صفر وگرنه ۱- خواهد بود. پس از پیداکردن فایل موردنظر، مشخصات این فایل در ساختمان ffblk قرار می‌گیرد. این ساختمان در فایل dir.h به صورت زیر تعریف شده است:

```
struct  ffblk {
        char ff-reserved [2] ;          (توسط DOS رزرو شده است)
        char ff-attrib ;                (صفت فایل را مشخص می‌کند)
        int  ff-ftime ;                 (زمانی که فایل تشکیل شده است را مشخص می‌کند)
        int  ff-fdate ;                 (تاریخی راکه فایل تشکیل شده است مشخص می‌کند)
        long ff-fsite ;                 (میزان حافظه مصرفی توسط فایل را مشخص می‌کند)
        char ff-name [13] ;             (نام فایل را مشخص می‌کند)
} ;
```

فایل‌ها ممکن است دارای صفات متفاوتی باشند (مخفی، فقط خواندنی، سیستم و ...) اگر بخواهیم تابع findfirst() فایل‌هایی با صفات خاص را مورد جستجو قرار دهد، این صفت باید توسط پارامتر attrib به آن اعلام گردد. صفات فایل‌ها با چند ماکرو مشخص می‌شود که اسامی این ماکرو در فایل dos.h تعریف شده‌اند. این ماکروها عبارتنداز:

نام ماکرو	فایل‌های موردجستجو	
FA_RDONLY	فایل فقط خواندنی	(Read only)
FA_HIDDEN	فایل مخفی	(Hidden)
FA_SYSTEM	فایل از نوع سیستم	(System)
FA_LABEL	برچسب	(Volume label)
FA_DIREC	زیرفهرست	(Subdirectory)
FA_ARCH	بیت آرشیو فایل، ۱ باشد	(Archive bit set)

به‌عنوان مثال، اگر پارامتر attrib برابر با FA_HIDDEN باشد، تابع ()findfirst برای پیداکردن فایل موردنظر، فقط در بین فایل‌های مخفی به جستجو می‌پردازد. اگر مقدار پارامتر attrib برابر با صفر باشد، بدین معنی است که "مهم نیست فایل مورد جستجو دارای چه صفتی باشد". لذا اولین فایلی که اسم و پسوند آن با فایل مورد جستجو مطابقت داشته باشد، به عنوان نتیجه عمل تابع ()findfirst محسوب می‌شود.

پس از پیدا شدن اولین فایل توسط تابع ()findfirst ، چنانچه خواسته باشیم فایل دیگری با همین مشخصات پیدا شود از تابع ()findnext استفاده می‌گردد. الگوی این تابع به صورت زیر است:

```
int findnext (struct ffblk *ptr) ;
```

در این الگو، پارامتر ffblk همان پارامتری است که در تابع ()findfirst تشریح شد.

مثال ۵۵-۹

برنامه‌ای که نام و میزان حافظه مصرفی کلیه فایل‌های با پسوند CPP را که در فهرست جاری قرار دارند، بر روی صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <dir.h>
void main()
{
  struct  ffblk f ;
  register int done ;
  clrscr();
  done = findfirst("*.cpp", &f, 0) ;
  while(!done){
    printf("\n%s,%ld", f.ff_name, f.ff_fsize) ;
    done = findnext(&f) ;
  }
  getch();
}
```

تابع ()getcurdir

تابع ()getcurdir مسیر جاری درایو مورد نظر را مشخص می‌کند. الگوی این تابع که در فایل **stdio.h** قرار دارد، به صورت زیر است:

```
int getcurdir(int drive, char *directory)
```

در این الگو drive نام درایوی است که مسیر جاری آن باید مشخص شود و directory رشته‌ای است که حاوی مسیر جاری خواهد بود. شماره A برابر با ۱، درایو B برابر با ۲ و ... است. درایو جاری صفر منظور می‌شود.

مثال ۵۶-۹

برنامه‌ای که مسیر جاری درایو جاری را تعیین می‌کند.

```
#include <conio.h>
#include <dir.h>
#include <stdio.h>
void main()
{
   char dir[MAXDIR];
   clrscr();
   getcurdir(0, dir);
   printf("current direxTory is:%s", dir);
   getch();
}
```

تابع getdisk()

تابع getdisk() شمارهٔ درایو جاری را مشخص می‌کند؛ شماره درایو A برابر با صفر، شماره درایو B برابر با ۱ و ... است. تابع دارای الگوی زیر است:

```
int  getdisk (void)
```

مثال ۹-۵۷

برنامه‌ای که نام درایو جاری را در صفحه‌نمایش چاپ می‌کند.

```
#include <stdio.h>
#include <dir.h>
#include <conio.h>
void main ()
{
   clrscr();
   printf("\n current drive is: %c ", getdisk() + 'A') ;
   getch();
}
```

تابع mktemp()

تابع mktemp() برای ساختن یک نام فایل منحصر به فرد (نامی که بر روی مسیر جاری وجود نداشته باشد) به کار می‌رود و دارای الگوی زیر است:

```
char  *mktemp (char  *fname)
```

در این الگو، fname به رشته‌ای اشاره می‌کند که این رشته باید حاوی ۶ عدد x باشد. باید توجه داشت که این تابع، فایلی را ایجاد نمی‌کند بلکه فقط یک نام فایل می‌سازد.

مثال ۹-۵۸

برنامه‌ای که یک نام فایل منحصر به فردی را تولید می‌نماید.

```
#include <stdio.h>
#include <dir.h>
#include <conio.h>
char fname[7]="XXXXXX" ;
void main ()
{
    clrscr();
    printf("%s", mktemp(fname)) ;
    getch();
}
```

تابع searchpath()

تابع searchpath() برای جستجوی یک فایل در مسیرهای تعیین شده توسط دستور path (از دستورهای سیستم عامل DOS) به کار می‌رود و دارای الگوی زیر است:

char *searchpath (char *fname)

در الگوی فوق، fname به رشته‌ای اشاره می‌کند که حاوی نام فایل مورد جستجو است. اگر این فایل در مسیرهای اعلام شده توسط دستور path وجود داشته باشد، تابع searchpath() آن را پیدا کرده و نام این تابع به مسیری که این فایل در آنجا وجود دارد اشاره می‌کند.

مثال ۵۹-۹

برنامه‌ای که محل وجود فایل edit.com را مشخص می‌کند.

```
#include <stdio.h>
#include <dir.h>
#include <conio.h>
void main ()
{
    clrscr();
    printf("\n the path is:%s", searchpath("edit.com")) ;
    getch();
}
```

تابع setdisk()

تابع setdisk() برای تعیین درایو جاری به کار می‌رود و دارای الگوی زیر است:

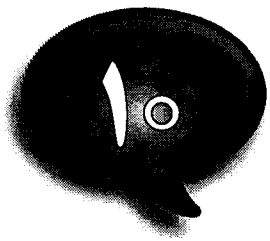
int setdisk (int drive)

برای تعیین درایو جاری کافی است که شمارهٔ درایو در پارامتر drive قرار گیرد. درایو A با شماره صفر، درایو B با شماره ۱ و ... مشخص می‌شود.

مثال ۹-۶۰

برنامه‌ای که درایو A را به عنوان درایو جاری تعیین می‌کند.

```
#include <stdio.h>
#include <dir.h>
#include <conio.h>
void  main()
{
  clrscr();
  printf("\n %d drives", setdisk(0));
  getch();
}
```



صف، پشته، لیست پیوندی و درخت

هر برنامه متشکل از دو جزء است : الگوریتم و ساختمان داده. یک برنامهٔ خوب، برنامه‌ای است که بتواند از یک الگوریتم بهینه و ساختمان دادهٔ مناسبی جهت ذخیره و بازیابی اطلاعات استفاده نماید. ساختمان داده‌هایی که تاکنون مورد بررسی قرار گرفته‌اند عبارتند از: آرایه، ساختمان و فایل. صف، پشته، لیست پیوندی و درخت، ساختمان داده‌های دیگری هستند که در این فصل بررسی می‌شوند.

صف

تقریباً همهٔ مردم با مفهوم صف (queue) آشنایی دارند. شاید هر فردی حداقل یک بار در صف نانوایی برای خرید نان منتظر مانده است. قانونی که بر صف نانوایی (و یا هر صف دیگر) حاکم است این است که: اولین فردی که برای خرید نان مراجعه می‌کند در ابتدای صف منتظر دریافت نان می‌ماند و بقیه افرادی که از راه می‌رسند در پشت سر این شخص قرار خواهند گرفت. لذا می‌توان گفت که هر صف دارای یک انتها است که طول صف از انتهای آن افزایش و از ابتدای آن کاهش می‌یابد. یعنی نانوا از ابتدای صف شروع به دادن نان می‌کند تا به انتهای صف برسد. مفهوم صف در کامپیوتر نیز همینطور است. اولین داده‌ای که وارد صف می‌شود در ابتدای صف و داده‌های بعدی به دنبال آن قرار خواهند گرفت. پس از تشکیل صف، اولین داده، زودتر از همه و آخرین داده آخر از همه قابل دسترسی و استفاده است. به چنین روش دسترسی به داده‌ها، روش First In First Out (FIFO) گفته می‌شود. مفهوم FIFO این است که داده‌ای که زودتر وارد شود زودتر استفاده می‌شود. صف را می‌توان با استفاده از آرایه پیاده‌سازی کرد.

برای تشخیص ابتدا و انتهای صف به دو متغیر نیاز است. یکی از این متغیرها به اولین محل خالی صف (انتهای صف) اشاره می‌کند؛ عنصر بعدی که وارد صف می‌شود در این محل قرار خواهد گرفت. متغیر دیگر به ابتدای صف اشاره می‌کند؛ عنصری که باید از صف خارج شود از این محل انتخاب می‌گردد. در حین بررسی صف، متغیری که انتهای صف را مشخص می‌کند spos و متغیری که ابتدای صف را مشخص می‌کند rpos نامگذاری می‌شود.

برای قراردادن عنصری در انتهای صف، تابعی به نام ()qstore و برای خارج نمودن عنصری از ابتدای صف تابعی به نام ()qretrive را می‌نویسیم. در تابع ()qstore متغیر MAX طول صف را مشخص می‌کند و پارامتر q به عنصری که باید در صف قرار گیرد، اشاره می‌کند و p صف است. صف موقعی پر می‌شود که spos با MAX برابر شود. تابع ()qstore به صورت زیر نوشته می‌شود:

```
void    qstore (char *q)
   {
        if (spos == MAX) {
           printf ("\n list is full.") ;
           getch () ;
           return ;
        }
        p [spos] = q ;
        spos ++ ;
   }
```

تابع ()qretrive می‌تواند به صورت زیر نوشته شود:

```
char *qretrive ()
   {
      if (rpos == spos) {
         printf ("\n the list is empty.") ;
         getch () ;
         return NULL ;
      }
      rpos ++ ;
      return p[rpos-1] ;
}
```

همانطور که در تابع ()qretrive مشاهده می‌شود، وقتی که دو متغیر spos و rpos با یکدیگر مساوی باشند، بدین معنی است که صف خالی است و عنصری قابل برداشت نیست. در این صورت کاراکتر NULL به عنوان نتیجه عمل برگردانده می‌شود. برای آشنایی با نحوه عمل توابع ()qstore و ()qretrive به شکل ۱-۱۰ مراجعه نمایید.

مثال ۱-۱۰

برنامه‌ای که اسامی تعدادی از افراد را از ورودی خوانده در صف قرار می‌دهد. خارج نمودن عناصری از صف و نمایش اسامی موجود در صف، از جمله وظایف این برنامه است.

شرح وظایف توابع

تابع ()main: ارزش دهی اولیه به صف، ظاهر نمودن منویی در صفحه‌نمایش و فراخوانی توابع مطابق با شکل ۲-۱۰.

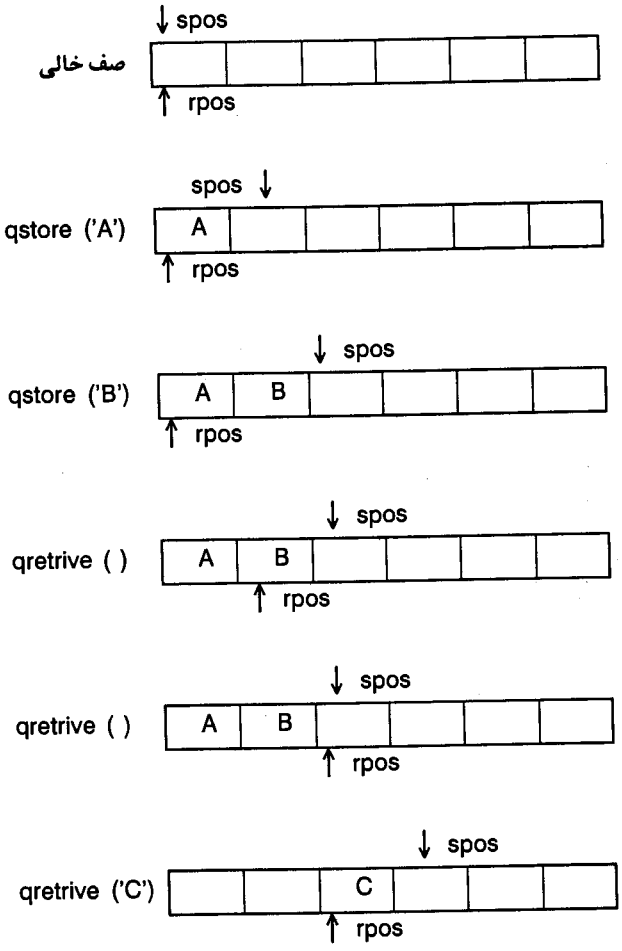
تابع ()enter: خواندن نام از ورودی و اخذ حافظه از سیستم جهت ذخیره کردن نام خوانده شده و فراخوانی تابع ()qstore جهت قراردادن آن در صف.

تابع ()review: ظاهر نمودن محتویات صف در صفحه‌نمایش.

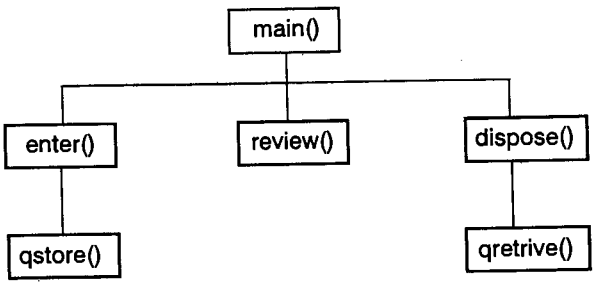
تابع ()dispose: حذف یک نام از ابتدای صف.

تابع ()qstore: قراردادن نام در صف. اگر صف پر باشد پیام مناسبی را صادر می‌کند.

تابع ()qretrive: عنصری از صف را جهت حذف کردن در اختیار تابع ()dispose قرار می‌دهد.



شکل ۱-۱۰ نحوه عمل توابع qstore() و qretrive().



شکل ۲-۱۰ نمودار سلسله مراتبی برنامهٔ مثال ۱-۱۰.

برنامه	متغیر	هدف
عمومی	MAX	ثابتی برای تعیین طول صف
	p	صفی به طول MAX
	spos	ابتدای صف
	rpos	انتهای صف
main()	s	رشته‌ای برای گزینه انتخاب کاربر
	t	اندیس حلقه تکرار
enter()	s	رشته‌ای که نام را نگه می‌دارد
	p	اشاره‌گری که به s اشاره می‌کند
review()	t	اندیس حلقه تکرار
dispose()	p	عنصری که از صف حذف می‌شود

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#define  MAX 100
char *p[MAX] ;
int spos, rpos ;
void enter(void) ;
void qstore(char *) ;
void review(void) ;
void dispose(void) ;
char *qretrive() ;
int main ()
{
    char s[8] ;
    register int t ;
    for (t = 0; t < MAX; ++t )
        p[t] = NULL ;
    spos = 0 ;
    rpos= 0 ;
    while (1) {
       clrscr() ;
       gotoxy(20, 4) ;
       printf(" E)enter name to queue ");
       gotoxy(20, 6) ;
       printf(" R)remove name from queue");
       gotoxy(20, 8) ;
       printf(" L)list the  queue ") ;
       gotoxy(20, 10) ;
       printf(" Q)quit from program ") ;
       gotoxy(20, 12) ;
       printf("enter your select");
       printf("( E R L Q):");
       gets(s) ;
      *s = toupper(*s) ;
       switch (*s) {
           case 'E' :  enter() ; break ;
           case 'L' :  review(); getch() ; break ;
           case 'R' :  dispose() ; break ;
           case 'Q' : exit(0) ;
```

```
        }//end of switch
    }
}
//*****************
void enter ()
{
  char s[256], *p ;
  do {
       printf("\nenter a name");
       printf("(ENTER for none):") ;
       gets(s) ;
       if (*s == 0)
          break;
       p =(char *)  malloc(strlen(s)) ;
       if (!p)  {
         printf("\n out of memory, press a key. ");
           getch();
           return ;
       } //end of if
       strcpy(p, s) ;
       if (*s)
           qstore(p) ;
  } while(*s) ;
}
//*****************
void review ()
{
    register int t ;
    for (t = rpos; t < spos; t++)
        printf("\n\t%d . %s ", t+1, p[t]);
}
//*******************************
void dispose ()
    {
       char *p ;
       if (!(p=qretrive()))
            return ;
       printf("\n %s",p) ;
     }
//*******************************
void qstore (char *q)
    {
       if (spos == MAX)
```

```
        {
          printf("\n list is full.") ;
          getch() ;
          return ;
        }
        p[spos]=q ;
        spos ++ ;
    }
//*******************************
char *qretrive ()
  {
   if (rpos == spos)
   {
     printf("\n the list is empty .");
     ch=getch() ;
     return NULL ;
   }
   rpos ++ ;
   return p[rpos-1] ;
  }
```

عملکرد برنامه

پس از اجرای برنامه، منویی با چهار گزینه ظاهر می‌شود. گزینه اول برای واردکردن نام در صف، گزینه دوم برای حذف نام از صف، گزینه سوم برای نمایش محتویات صف و گزینه چهارم برای خروج از برنامه است:

منوی برنامه

```
E)enter name to quque
R)remove name from queue
L)list the queue
Q)quit from program
enter your select (E R L Q) :
```

صف دایره‌ای

صفی که در قسمت قبلی مورد بررسی قرار گرفت، توسط آرایه پیاده‌سازی شده است. در این مورد، اگر متغیر spos به انتهای آرایه برسد، نمی‌توان عناصر دیگری در صف قرار داد. هر چند که ممکن است تعداد زیادی از عناصر از صف خارج شده باشند و عناصر ابتدای آرایه خالی باشند. اگر این امکان فراهم شود که متغیر spos پس از رسیدن به انتهای صف، بتواند به ابتدای صف برگردد صف جدیدی حاصل می‌شود که به **صف دایره‌ای** معروف است؛ زیرا از حافظه آرایه به جای استفاده خطی، به صورت دایره‌ای استفاده می‌گردد. برنامهٔ مثال ۱-۱۰ را می‌توان با استفاده از صف دایره‌ای نیز نوشت. تابع ()qstore در صف دایره‌ای به صورت زیر نوشته می‌شود. همانطور که در تابع ()qstore می‌بینید، صف دایره‌ای وقتی پر خواهد بود که spos از rpos یک واحد کمتر باشد و یا اینکه spos به انتهای صف رسیده (برابر MAX باشد) و rpos در ابتدای صف قرار داشته باشد.

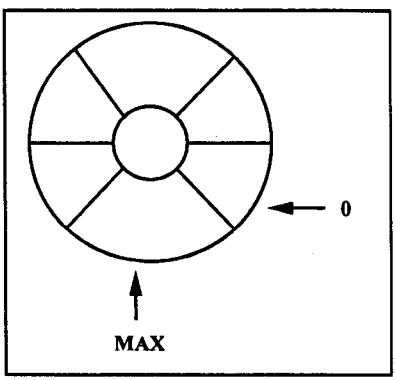
```
/* qstore function for circular queue */
void qstore(char *q)
{
  if(spos + 1 == rpos || (spos + 1 == MAX && !rpos))
  {
     printf("\n queue is full .") ;
     return ;
  }
  p[spos] = q ;
  spos ++ ;
  if(spos == MAX)
    spos = 0;
}
```

تابع ()qretrive در صف دایره‌ای به صورت زیر نوشته می‌شود:

```
char *qretrive()
{
     if(rpos == MAX)
        rpos = 0;
     if(rpos == spos)  {
        printf("\n queue is empty.");
        return NULL ;
     }
     rpos ++ ;
     return p[rpos - 1] ;
}
```

همانطور که در تابع ()qretrive مشاهده می‌گردد، شرط خالی بودن صف این است که rpos و spos با یکدیگر برابر باشند. ضمناً اگر متغیر spos به انتهای آرایه رسید (برابر با MAX شد) تبدیل به صفر می‌گردد تا به ابتدای صف اشاره نماید.

نکته‌ای که در مورد صف دایره‌ای باید توجه داشت این‌است‌که اگر طول صف با MAX مشخص شود، حداکثر تعداد عناصری که می‌توانند در صف قرار گیرند برابر با MAX-1 است، زیرا در غیر این صورت نمی‌توان تشخیص داد که صف پر است یا خالی. برای این که تشکیل صف دایره‌ای با مشکل مواجه نشود باید در شروع کار، مقدار متغیر rpos برابر با MAX باشد (rpos و spos باید با یکدیگر یک واحد فاصله داشته باشند). صف دایره‌ای را به صورت شکل ۳-۱۰ می‌توان در نظر گرفت.



شکل ۳-۱۰ نمونه‌ای از صف دایره‌ای.

بعضی از موارد کاربرد صف‌های دایره‌ای عبارتند از:

۱. یکی از بیشترین کاربرد صف‌های دایره‌ای در سیستم عامل، برای نگهداری اطلاعاتی است که باید بر روی دیسک نوشته یا خوانده شوند.

۲. یکی دیگر از موارد کاربرد صف‌های دایره‌ای در سیستم‌های بلادرنگ (real time) است که باید اطلاعات موجود در بافر را مورد پردازش قرار دهد.

۳. بسیاری از واژه‌پردازها (word processor) برای پاراگراف‌بندی یا تنظیم خطوط، از صف دایره‌ای استفاده می‌کنند.

۴. در بسیاری از برنامه‌ها در حین انجام یک عمل ممکن است کلیدهایی از صفحه‌کلید فشار داده شوند، این کلیدها در صفحه‌نمایش ظاهر نمی‌شوند تا اینکه برنامه و یا اجرای قسمتی از برنامه (یک فرآیند) به پایان برسد. برای این منظور، برنامه‌های کاربردی در حین انجام عمل مربوط، صفحه‌کلید را نیز تست می‌کنند که در صورت فشاردادن کلیدی از صفحه‌کلید، آن را در صف قرار می‌دهند تا عملی که در حال انجام آن هستند به پایان برسد و سپس به نمایش کاراکترهای قرارگرفته در صف می‌پردازند. برای آشنایی با چگونگی این عمل، مسأله‌ای در مثال ۲-۱۰ مطرح و برنامه‌ای جهت حل آن نوشته شده است.

مثال ۲-۱۰

برنامه‌ای که شامل دو فرآیند است: ۱. نمایش اعداد از ۱ تا ۳۲۰۰۰ در صفحه‌نمایش ۲. قراردادن کاراکترهای تایپ شده در یک صف دایره‌ای تا اینکه حرف ;وارد شود. کاراکتر تایپ شده در صفحه ظاهر نمی‌شود، زیرا نمایش اعداد از اولویت بالاتری برخوردار است. وقتی که ;تایپ شد کاراکترهای موجود در صف دایره‌ای از صف خارج شده، در صفحه‌نمایش ظاهر می‌شوند. برای این منظور از دو تابع ()kbhit و ()getch استفاده شده است. تابع ()kbhit فشاردادن یک کلید از صفحه‌کلید را تشخیص می‌دهد (در صورت فشاردادن یک کلید، این تابع عدد صفر را برمی‌گرداند) و تابع ()getch این کاراکتر را خوانده سپس با استفاده از تابع ()qstore در صف قرار می‌گیرد.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#define   MAX 80
char buf[MAX + 1] ;
int spos = 0 ;
int rpos = MAX;
void qstore(char) ;
char qretrive() ;
int  main ()
{
      register char ch ;
      int t ;
      buf[80]=NULL ;
```

```
        for (ch = ' ', t = 0; t < 32000 && ch !=';' ; t++) {
          printf("%d",kbhit()) ;
          if (kbhit())  {
              ch = getch() ;
              qstore(ch) ;
          }//end of if
          printf("%d ",t) ;
        } //end of for
        while ((ch == qretrive()) != NULL)
              putchar(ch);
        getch();
        return 0;
}
//*******************
void qstore(char q)
{
   if ((spos + 1 == rpos) || (spos + 1 == MAX && !rpos)) {
      printf("\n list is full .");
      getch();
      return ;
    }//end of if
    buf[spos] = q ;
    spos ++ ;
    if (spos == MAX)  spos = 0 ;
}
//*******************
char qretrive()
{
   if(rpos == MAX)   rpos = 0 ;
   if(rpos == spos)  return NULL ;
   rpos ++ ;
   return buf[rpos - 1] ;
}
```

پشته

پشته برعکس صف عمل می‌کند. در صف، عناصری که زودتر در لیست قرار می‌گیرند زودتر از لیست خارج می‌شوند و لذا عنصری که آخر از همه وارد لیست شد، آخرین عنصری است که از صف خارج می‌شود. در پشته، عنصری که دیرتر از همه در لیست قرار گرفت زودتر از همه از لیست خارج می‌شود. به این روش دسترسی به اطلاعات، روش Last In First Out (LIFO) گفته می‌شود.

برای پیاده‌سازی پشته می‌توان از آرایه یا تخصیص حافظه پویا استفاده کرد. در هر دو مورد، به متغیری نیاز است تا محل قرارگرفتن عنصر جدید را در پشته مشخص کند. این متغیر را tos در نظر می‌گیریم. همانند صف، در

پشته نیز برای قراردادن یک عنصر در پشته (push) و برداشتن یک عنصر از پشته (pop) به دو تابع نیاز است. این دو تابع را به ترتیب ()push و ()pop نامگذاری کرده، می‌توان آنها را به صورت زیر نوشت (پیاده‌سازی توسط آرایه):

```
#define MAX 100
int stack[MAX] ;
int tos = 0 ;
void push(int i)
{
    if(tos >= MAX) {
        printf("\n stack is full .");
        getch();
        return ;
    }
    stack[tos] = i;
    tos ++;
}
```

در تابع ()push پارامتر i عنصری است که باید در پشته قرار گیرد؛ هرگاه متغیر tos برابر با MAX (طول آرایه) باشد، پشته پر شده و عنصر جدید در آن قرار نمی‌گیرد. همانطور که در تابع ()pop مشاهده می‌شود وقتی که متغیر tos کمتر از صفر باشد، بدین معنی است که پشته خالی است و هیچ عنصری قابل برداشت نیست؛ لذا مقدار HUGE_VAL (ماکرویی در فایل math.h) به تابع فراخواننده برگردانده می‌شود (شکل ۴-۱۰).

```
#include <stdlib.h>
int pop()
{
     tos -- ;
     if(tos < 0)  {
       printf("\n stack is empty.") ;
       return HUGE_VAL;
     }
     return stack[tos];
}
```

محتویات پشته	عمل انجام شده
A	push (A)
BA	push (B)
CBA	push (C)
BA	pop ( )
FBA	push (F)
BA	pop ( )
A	pop ( )
خالی	pop ( )

شکل ۴-۱۰ نحوهٔ عمل پشته.

در روش تخصیص حافظه پویا، توابع ()push و ()pop را می‌توان به صورت زیر نوشت:

```
int *p , *tos , *bos ;
void push(int i)
{
    if(p > bos) {
      printf("\n stack is full.") ;
```

```
        return ;
      }
      *p = i ;
      p ++;
}
 //*************************
int pop()
{
    p -- ;
    if(p < tos) {
       printf("\n stack is empty .");
       return 0 ;
    }
    return *p ;
}
```

قبل از اینکه توابع ()push و ()pop جدید بتواند عمل کند باید حافظه موردنیاز را توسط تابع ()malloc از سیستم اخذ نمود. متغیرهای اشاره‌گر tos و bos به ترتیب به بالا و پایین پشته اشاره می‌کنند.

مثال ۳-۱۰

برنامه‌ای که همانند ماشین حسابی کار می‌کند که فقط چهارعمل اصلی را انجام می‌دهد. برای این منظور ابتدا دو عملوند را گرفته در یک پشته قرار می‌دهد و پس از واردشدن عملگر، دو عملوند قبلی را از پشته آزاد کرده این عملگر را بر روی آنها عمل می‌کند و نتیجه را در پشته قرار می‌دهد. کاراکتر نقطه (.) محتویات پشته را نمایش می‌دهد و با واردکردن حرف q برنامه خاتمه می‌یابد.

متغیرهای برنامه

برنامه	متغیر	هدف
عمومی	p	اشاره‌گری که به حافظه پشته اشاره می‌کند
	tos	اشاره‌گر بالای پشته
	bos	اشاره‌گر پایین پشته
main()	a,b	عملوندهایی که از پشته برداشته می‌شوند
	s	عملگر یا هر کاراکتر ورودی

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define  MAX 100
int *p ;
int *tos ;
int *bos ;
void push(int) ;
int pop ();
int main ()
{
    int a, b ;
    char s[80] ;
    p=(int *) malloc(MAX * sizeof(int));
    if(!p) {
```

```
    printf("\n allocation failure.");
    exit(1) ;
  }
  tos = p ;
  bos = p + MAX - 1 ;
  clrscr() ;
  printf("\n\t << funtion");
  printf(" calculation >>\n\n") ;
  do {
     printf(" :") ;
     gets(s) ;
     switch (*s)  {
        case '+' : a = pop() ;  b = pop() ;
             printf(" sum is : %d\n", a + b);
             push(a + b) ;
             break ;
        case '-' :  a = pop() ; b = pop() ;
             printf(" subtract is:%d\n", b - a);
             push(b - a) ;
             break ;
        case '*' : a = pop() ;  b = pop() ;
             printf(" multiply is:%d\n", a * b);
             push(a*b) ;
             break ;
        case '/' :  a = pop() ;
             b = pop() ;
             if (a == 0) {
               printf(" divided by zero.\n");
               getch();
               exit(1) ;
             }//end of if
             printf(" division is:%d\n", b / a);
             push(b / a) ;
             break ;
        case '.' :
             a = pop() ;
             printf(" current value on top of stack:%d\n", a) ;
             break ;
        default :
            push(atoi(s)) ;
    } // end of swith
} while (*s!='q') ;
return 0;
```

```
}
//**********************
void push ( int i)
{
  if(p > bos)  {
     printf("\n stack is full.") ;
     return ;
   }
   *p=i ;
   p++ ;
}
//*****************
int pop ()
{
    p -- ;
    if (p < tos) {
       printf("\n stack is empty .") ;
       getch();
       return 0 ;
    }
    return *p ;
}
```

عملکرد برنامه

```
: 12
: 30
: +

sum is : 42
: 2
: /

division is : 21
: 3
: *
multiply is : 63
: 40
: -

subtract is : 23
: .

current value on top of stack is : 23
: q
```

لیست پیوندی

ساختمان داده‌های صف و پشته که در این فصل بررسی شدند، دارای دو اشکال زیر هستند:

۱. همانند آرایه به محلهای حافظه نیاز دارند.

۲. عناصر پشته فقط از بالای آن و عناصر صف فقط از ابتدای آن قابل دستیابی‌اند. لذا برای دسترسی به عنصری که در غیر از بالای پشته و ابتدای صف قرار دارند، خارج نمودن عناصری که قبل از آنها واقع‌اند، ضروری است.

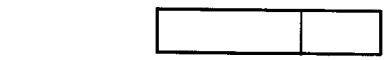
لیست پیوندی، ساختمان داده‌ای پویاست که هیچکدام از اشکالات صف و پشته را ندارد. زیرا هر عنصر آن، از طریق یک اشاره‌گر به عنصر بعدی متصل است و موجب دسترسی سریع به عناصر لیست می‌شود. عناصرِ لیست پیوندی را **گره** (node) می‌گویند.

مشخصات گره‌های لیست

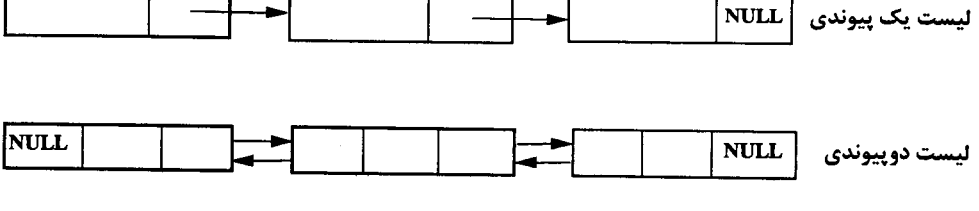
هر گره لیست از دو بخش تشکیل شده است: داده و آدرس. بخش داده می‌تواند از چند قسمت تشکیل شده باشد. نمونه‌ای از یک گره لیست در شکل ۵-۱۰ آمده است. بنابراین، هر گره لیست یک ساختمان است و بخش دادهٔ آن می‌تواند از چندین فیلد تشکیل شده باشد. بخش آدرس هر گره، آدرس عنصر بعدی را در خودش نگه می‌دارد.

لیست‌ها می‌توانند **یک پیوندی** یا **دوپیوندی** باشند، در لیست‌های یک پیوندی یک فیلد آدرس وجود دارد که به عنصر بعدی لیست اشاره می‌کند ولی در لیست دوپیوندی هر عنصر دارای دو اشاره‌گر است. یکی به عنصر بعدی و دیگری به عنصر قبلی اشاره می‌کند. شکل ۶-۱۰ نمونه‌هایی از لیست یک و دوپیوندی را نشان می‌دهد. همانطور که در این شکل می‌بینید، اشاره‌گر آخرین عنصر لیست یک پیوندی برابر با NULL و اشاره‌گر سمت چپ اولین عنصر و اشاره‌گر سمت راست آخرین عنصر لیست دوپیوندی برابر با NULL است.

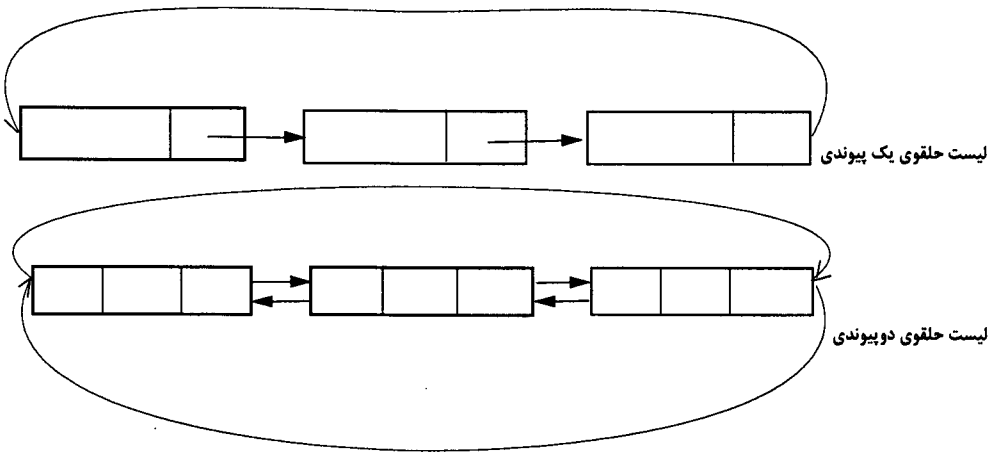
لیست پیوندی می‌تواند **حلقوی** (circular) باشد. اگر در لیست یک پیوندی، اشاره‌گر آخرین گره لیست به اولین گره لیست اشاره کند، لیست حلقوی یک پیوندی به وجود می‌آید. اگر در لیست دوپیوندی، اشاره‌گر سمت راست گره آخر به گره اول و اشاره‌گر سمت چپ گره اول به گره آخر اشاره کند، لیست حلقوی دوپیوندی ایجاد می‌شود (شکل ۷-۱۰).



شکل ۵-۱۰
نمونه‌ای از یک عنصر لیست.



شکل ۶-۱۰ نمونه‌هایی از لیست‌های یک پیوندی و دوپیوندی.



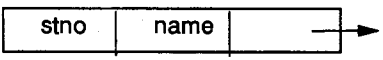
شکل ۷-۱۰ نمونه‌هایی از لیست‌های حلقوی.

تعریف گره لیست پیوندی

برای تعریف گره لیست پیوندی، از دستور struct استفاده می‌شود، زیرا هر گره لیست پیوندی یک ساختمان است. به عنوان مثال برای تعریف گره‌ای از لیست پیوندی که شامل نام و شماره دانشجویی می‌باشد، به صورت زیر عمل می‌شود.

```
struct    node {
          int stno ;
          char name [30] ;
          struct node *link ;
} ;
```

این ساختار به صورت زیر قابل نمایش است:



تعریف اشاره گرهای خارجی

اشاره گرهایی که به عنوان بخشی از ساختمان گره لیست می‌باشند، **اشاره گر داخلی** نام دارند. برای ایجاد لیست پیوندی به اشاره گرهای دیگری نیاز هست که آنها را **اشاره گرهای خارجی** گویند. اشاره گرهای خارجی از نوع گره لیست بوده، به گره‌ها اشاره می‌کنند. دستور زیر، دو اشاره گر start و last را از نوع node تعریف می‌کنند:

```
struct node *start, *last ;
```

ایجاد گره لیست پیوندی

برای ایجاد گره لیست پیوندی، از تابع ()malloc استفاده می‌شود. دستور زیر، گره‌ای را ایجاد کرده، آدرس آن را در

start قرار می‌دهد.

```
start = (struct node *) malloc (sizeof(struct  node)) ;
```

پیونددادن گره‌های لیست پیوندی

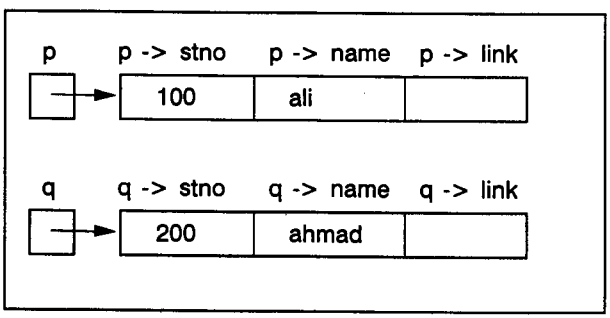
برای ایجاد لیست پیوندی، باید گره‌های لیست را ایجاد و سپس آنها را به هم پیوند داد. دستورات زیر، منجر به شکل ۸-۱۰ می‌شود.

```
struct    node *p, *q ;
p = (struct node *) malloc (sizeof (struct node)) ;
q = (struct node *) malloc (sizeof (struct node)) ;
p -> stno = 100
strcpy (p -> name, "ali") ;
q -> stno = 200 ;
strcpy (q -> name,"ahmad") ;
```

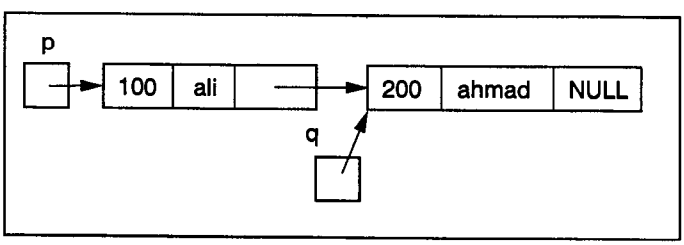
برای پیونددادن دو گره p و q، باید آدرس گره q را در p -> link قرار دهیم. این کار با دستور زیر صورت می‌گیرد:

```
p - > link = q ;
q - > link = NULL ;
```

با این دستور دو گره به هم پیوند خورده، شکل ۹-۱۰ را ایجاد می‌کند.



شکل ۸-۱۰
دو گره از لیست پیوندی، قبل از اتصال.



شکل ۹-۱۰
پیونددادن دو گره در لیست پیوندی.

درج گره‌ای در لیست پیوندی

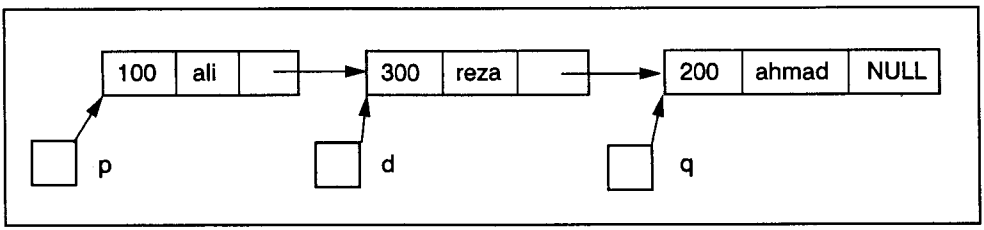
در شکل ۹-۱۰ دو گره وجود دارد که p ابتدای لیست و q انتهای لیست است. گره‌های جدید را می‌توان در ابتدا، انتها یا وسط لیست درج کرد. درج گره در انتهای لیست مثل وصل کردن p و q در شکل ۹-۱۰ است. اکنون روش درج گره در ابتدا و وسط لیست را بررسی می‌کنیم.

درج گره در وسط لیست: فرض کنیم گره‌ای به نام d بخواهد بین گره‌های p و q درج شود. برای این کار، d باید به q اشاره کند و پیوند بین p و q قطع شود و p به d اشاره نماید:

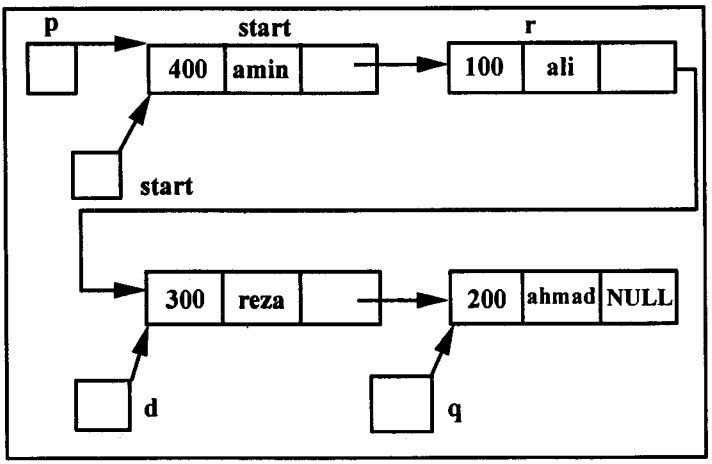
```
d -> link = q ;
p -> link = d ;
```

با اجرای این دستورات، شکل ۱۰-۱۰ به وجود می‌آید.

درج گره در ابتدای لیست پیوندی: برای اینکه گره‌ای مثل start در ابتدای لیست پیوندی درج شود، قسمت آدرس start باید به ابتدای لیست اشاره کند و سپس اشاره‌گری که قبلاً به ابتدای لیست اشاره می‌کرد، باید به گره جدید اشاره نماید. فرض کنید گره start با محتویات زیر باید به ابتدای لیست شکل ۱۰-۱۰ اضافه شود (شکل ۱۱-۱۰).



شکل ۱۰-۱۰ درج گره در وسط لیست پیوندی.



شکل ۱۱-۱۰ درج گره در ابتدای لیست پیوندی.

برای این منظور، باید دستورات زیر را اجرا کرد:

```
start -> link = p ;        اتصال گره start به p
p = start ;                انتقال p به ابتدای لیست
```

حذف گره از لیست پیوندی

حذف گره از لیست پیوندی، از وسط یا ابتدای لیست با هم متفاوت است. در شکل ۱۱-۱۰ برای حذف گره‌ای که d به آن اشاره می‌کند، باید پیوند گره قبل از d را به گره بعد از d (q) وصل کرد و پیوند d را با q قطع نمود. برای این منظور باید اشاره‌گری به گره قبل از d اشاره کند. با فرض اینکه r به گره قبل از d اشاره کند، دستورات زیر گره d را حذف می‌کنند:

```
r -> link = d -> link ;
d -> link = NULL
free (d) ;
```

برای حذف گره از ابتدای لیست پیوندی، باید پس از حذف گره، اشاره‌گر ابتدای لیست، مثل start را به گره بعدی انتقال داد. برای حذف گره ابتدای لیستِ شکل ۱۱-۱۰، به صورت زیر عمل می‌شود:

```
start = start -> link ;
p -> link = NULL ;
free (p) ;
```

پیمایش لیست پیوندی

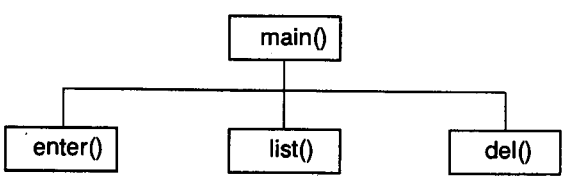
در بسیاری از موارد لازم است عناصر لیست به ترتیب مورد بررسی قرار گیرند. به عنوان مثال، برای جستجوی عنصری در لیست پیوندی، تمام گره‌های لیست باید بررسی شوند. بررسی تمام گره‌های لیست پیوندی را **پیمایش** لیست پیوندی گویند (مثال ۴-۱۰).

مثال ۴-۱۰

برنامه‌ای که یک لیست پیوندی از اطلاعات مربوط به دانشجویان تشکیل می‌دهد. حذف دانشجویی از لیست پیوندی و نمایش کلیه اطلاعات لیست در صفحه‌نمایش، از جمله وظایف این برنامه است.

توضیح

ساختمان student برای نگهداری مشخصات دانشجو تعریف شده است : name نام دانشجو، stno شماره دانشجویی، unit واحدهای گذرانده شده توسط دانشجو، و next اشاره‌گر به گره بعدی لیست. قبل از مشاهده لیست برنامه بهتر است نگاهی به نمودار سلسله مراتبی برنامه و وظایف هر یک از توابع آن داشته باشیم.



شکل ۱۲-۱۰
نمودار سلسله مراتبی برنامهٔ مثال ۴-۱۰.

شرح وظایف توابع

تابع main(): ظاهر نمودن منو و فراخوانی توابع مطابق نمودار سلسله مراتبی شکل ۱۲-۱۰.

تابع enter(): اخذ اطلاعات از ورودی و قراردادن آنها در لیست پیوندی.

تابع list(): نمایش اطلاعات موجود در لیست پیوندی در صفحه‌نمایش.

تابع del(): حذف دانشجویی از لیست پیوندی با استفاده از شماره دانشجویی آن.

متغیرهای برنامه

برنامه	متغیر	هدف
عمومی	TRUE	تعریف مقدار یک برای ارزش درستی
	first	اشاره‌گر به ابتدای لیست
	last	اشاره‌گر به انتهای لیست
	node	گره جدیدی که اخذ می‌شود
	s	انتخاب کاربر از گزینه منو
enter()	numstr	متغیر کمکی برای خواندن اطلاعات
list	i	سطری که اطلاعات باید نوشته شود
del()	numstr	شماره دانشجویی که باید حذف شود

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define  TRUE  1
#define NIL (struct student *)NULL
#define NIL (struct student *)NULL
struct student {
       char name[30] ;
       int stno ;
       int unit ;
       struct student *next ;
} ;
struct student *first , *last , *node ;
void enter ();
void list ();
void del ();
int main () {
    char  s[10] ;
    first = NIL ;
    while (TRUE) {
     clrscr() ;
     gotoxy(20, 4) ;
     printf("E)enter name to list ");
     gotoxy(20, 6) ;
     printf("R)remove name from list");
     gotoxy(20, 8) ;
```

```
      printf("L)print the list on CRT");
      gotoxy(20, 10) ;
      printf("Q)quit from program ") ;
      gotoxy(20, 12) ;
      printf("enter your select");
      printf("( E R L Q):") ;
      gets(s) ;
      *s = toupper(*s) ;
      switch (*s) {
          case 'E' :  enter() ;   break ;
          case 'L' :  list() ;    getch() ;   break ;
          case 'R' :  del() ;     break ;
          case 'Q' :  exit(0) ;
      }//end of switch
    } //end of while
} // end of main
//********************
void enter () {
   char numstr[30] ;
   node = (struct student *)malloc(sizeof(struct student));
   node -> next = NIL ;
   if (first == NIL)
       first = last = node ;
   else {
      last -> next = node ;
      last = node ;
   }
   printf("\n enter name of student:");
   gets(last -> name) ;
   printf("\n enter student number :");
   gets(numstr) ;
   last -> stno = atoi(numstr) ;
   printf("\n enter number of unit :");
   gets(numstr) ;
   last -> unit = atoi(numstr) ;
} /* end of enter */
//********************
void list ( ) {
     int i ;
     if (first == NIL) {
       printf("\n<< the list is empty .>>") ;
       getch() ;
       return ;
     }
     last = first ;
     clrscr() ;
     gotoxy(5,4) ;
     printf("  name              st.number     unit ") ;
```

```
        gotoxy(5, 5);
        printf("---------          ---------     -----") ;
       i = 6 ;
       do {
        gotoxy(5, i);
        printf("%s", last -> name) ;
        gotoxy(25, i);
        printf("%d", last -> stno) ;
        gotoxy(38, i);
        printf("%d", last -> unit) ;
        i ++ ;
        last=last -> next ;
      } while (last != NIL) ;
      gotoxy(5, i++) ;
       printf("**********************") ;
       printf("****************") ;
      gotoxy(10, i++) ;
      printf("press a key to continue.");
      getch() ;
}
//*******************
void del () {
    int stnumber ;
    gotoxy(20, 14) ;
    printf("enter student number for delete:");
    scanf("%d", &stnumber) ;
    last = node = first ;
    while (last != NIL)    {
     if (last -> stno!=stnumber)  {
       node = last ;
       last = last -> next ;
       continue ;
    }
    else {
        if (last == first) {
            first=last -> next ;
            free(last) ;
            free(node) ;
            break ;
        }//end of if
        else  {
           node -> next=last -> next;
           free(last) ;
           break ;
        }//end of else
    }//end of else
   }//end of while
}
```

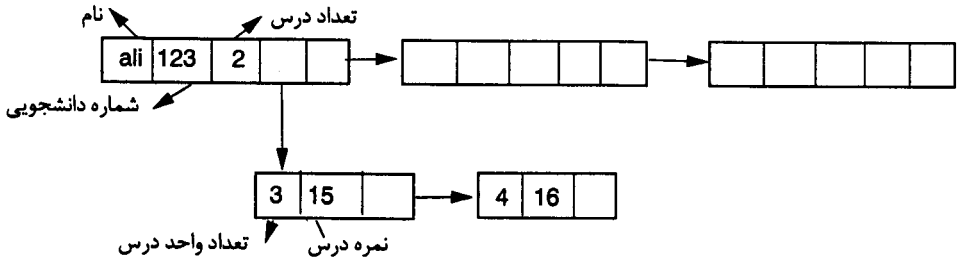
عملکرد برنامه

پس از اجرای برنامه، منویی با ۴گزینه ظاهر می‌شود. گزینه اول دانشجویی را وارد لیست می‌کند، گزینه دوم، دانشجویی را از لیست حذف می‌کند، گزینه سوم محتویات لیست را در صفحه‌نمایش ظاهر می‌کند و گزینه چهارم موجب خروج از برنامه می‌شود. پس از هر عملی که با برنامه انجام می‌دهید، دوباره منو ظاهر می‌شود تا وقتی که q را برای خروج از برنامه تایپ کنید.

```
E)enter name to list
R)remove name from list
L)print the list on CRT
Q)quit from program
enter your select(E R L Q):
```

مثال ۵-۱۰

برنامه‌ای که اطلاعات مربوط به دانشجویان دانشگاهی را از ورودی خوانده و در یک لیست پیوندی که شامل دو قسمت است قرار می‌دهد. در قسمت اول، نام، شماره دانشجویی و تعداد درسی که دانشجو گذرانده است قرار دارد و قسمت دوم شامل نمره هر درس و تعداد واحد آن است. قسمت اول این لیست پیوندی دارای دو اشاره‌گر است که یکی از این اشاره‌گرها به قسمت دوم لیست و دیگری به عنصر بعدی در قسمت اول اشاره می‌کند. نمونه‌ای از این لیست پیوندی در زیر مشاهده می‌شود:



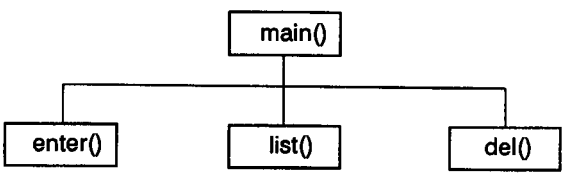
شرح وظایف توابع

تابع main(): ظاهر نمودن منویی در صفحه‌نمایش و فراخوانی سایر توابع مطابق با شکل ۱۳-۱۰.

تابع enter(): اخذ حافظه از سیستم، خواندن اطلاعات از ورودی و قراردادن آنها در لیست پیوندی.

تابع list(): چاپ محتویات لیست پیوندی در صفحه‌نمایش.

تابع del(): اخذ یک شماره دانشجویی از ورودی و حذف این دانشجو از لیست پیوندی.



شکل ۱۳-۱۰
نمودار سلسله مراتبی برنامهٔ مثال ۵-۱۰.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define  TRUE  1
#define NIL (struct student *)NULL
void enter();
void list();
void del();
struct sublist {
      int    unit ;
      float  grade ;
      struct sublist *subnext ;
} ;
struct student {
      char name[30] ;
      int stno ;
      int nocours ;
      struct student *next ;
      struct sublist *link ;
} ;
struct student *first , *last , *node ;
struct sublist *sfirst ,*slast , *snode ;
int main ()
{
   char s[10] ;
   first = NIL ;
   while(TRUE) {
      clrscr() ;
      gotoxy(20, 4) ;
      printf("E)enter name to list ");
      gotoxy(20, 6) ;
      printf("R)remove name from list");
      gotoxy(20, 8) ;
      printf("L)print the list on CRT ");
      gotoxy(20, 10) ;
      printf("Q)quit from program ") ;
      gotoxy(20, 12) ;
      printf("enter your select");
      printf("( E R L Q):") ;
      gets(s) ;
      *s=toupper(*s) ;
      switch (*s) {
```

```
        case 'E' :  enter();  break ;
        case 'L' :  list() ;  getch() ;  break ;
        case 'R' :  del() ;   break ;
        case 'Q' :  exit(0) ;
      }//end of switch
    }//end of while
}
//***************
void enter ()
  {
   register int t ;
   float x ;
   char numstr[30] ;
   node = (struct student *)  malloc(sizeof(struct student));
   node -> next = NIL ;
   if (first == NIL)
       first = last = node ;
   else  {
      last -> next = node ;
      last = node ;
   }//end of else
   printf("\n enter name of student:");
   gets(last -> name) ;
   printf("\n enter student number :") ;
   gets(numstr) ;
   last -> stno = atoi(numstr) ;
   printf("\nenter number of courses:");
   gets(numstr) ;
   last -> nocours = atoi(numstr) ;
   slast = NIL ;
   for (t = 0 ; t < last -> nocours; t++) {
       snode = (struct sublist *)
        malloc(sizeof(struct sublist)) ;
       snode -> subnext = NULL ;
       if (slast == NULL) {
         slast = snode;
         last -> link = slast;
     }// end of if
     else  {
        slast -> subnext = snode ;
        slast = snode ;
     }//end of else
     gotoxy(40,16) ;
```

```
      printf("                              ");
      gotoxy(40,16) ;
      printf("enter grade number %d:", t + 1);
      gets(numstr);
      slast -> grade = atof(numstr);
      gotoxy(40, 18) ;
      printf("                              ") ;
      gotoxy(40, 18) ;
      printf(" enter unit of grade %d:", t + 1);
      gets(numstr);
      slast -> unit = atoi(numstr);
    }//end of for
} /* end of enter */
//**********************
void list ( )
  {
     int i ;
     struct student *help;
     if (first == NIL)  {
       printf("\n<< the list is empty .>>") ;
       getch() ;
       return ;
     }
     help = first;
     clrscr() ;
     gotoxy(5, 4);
     printf("  name              st.numb   grade unit ") ;
     gotoxy(5, 5);
     printf("---------          -------  -----  ---") ;
     i = 6 ;
     do {
       gotoxy(5, i);
       printf("%s", help -> name) ;
       gotoxy(24, i);
       printf("%d", help -> stno) ;
       slast = help -> link ;
       while (slast != NULL) {
          gotoxy(33, i);
          printf("%.2f", slast -> grade);
          gotoxy(39, i);
          printf("%2d", slast -> unit) ;
          slast = slast -> subnext ;
          i++ ;
```

```
        }
        help = help -> next ;
      } while (help != NIL) ;
      gotoxy(5, i++) ;
       printf("************************");
       printf("***************") ;
      gotoxy(10, i++) ;
      printf("press a key to continue.");
      getch() ;
}
//******************
void del ()
{
        struct student *help;
        int stnumber ;
        gotoxy(20,14) ;
        printf("enter student number for delete:") ;
        scanf("%d", &stnumber) ;
        help = node = first ;
        while (help != NIL) {
            if (help -> stno != stnumber)  {
               node = help ;
               help = help -> next ;
               continue ;
            }//end of if
            else if (help == first)  {
               first = help -> next ;
               free(help) ;
               free(node) ;
               break ;
            }//end of else if
            else {
               node -> next = help -> next;
               free(help) ;
               break ;
            }//end of else
        }// end of while
} // end of del()
```

عملکرد برنامه

پس از اجرای برنامه منویی با چهار گزینه ظاهر می‌شود. گزینه اول نامی را خوانده در لیست پیوندی قرار می‌دهد. گزینه دوم نام را از لیست حذف می‌کند، گزینه سوم محتویات لیست را در صفحه نمایش ظاهر می‌کند و گزینه چهارم موجب خروج از برنامه می‌شود.

منوی برنامه

```
E)enter name to list
R)remove name from list
L)print the list on CRT
Q)quit from program
enter your select( E R L Q):
```

لیست حلقوی

اگر اشاره‌گر آخرین گره لیست، به ابتدای لیست پیوندی اشاره کند، لیست حلقوی یک پیوندی ایجاد می‌شود. در لیست حلقوی، یک اشاره‌گر خارجی به ابتدای لیست و یک اشاره‌گر خارجی به انتهای لیست اشاره می‌نماید.

مثال ۶-۱۰

برنامه‌ای که یک لیست پیوندی حلقوی از اطلاعات مربوط به کارکنان تشکیل داده سپس عناصر لیست را به خروجی منتقل می‌کند.

شرح وظایف برنامه‌ها

برنامه main(): دریافت تعداد کارکنان از ورودی، انتقال آن به تابع enter() جهت خواندن اطلاعات کارکنان و ایجاد لیست حلقوی. فراخوانی تابع list() جهت گزارشگیری از لیست.

تابع enter(): دریافت تعداد کارکنان به عنوان آرگومان تابع، خواندن اطلاعات کارکنان و ایجاد لیست حلقوی.

تابع list(): نمایش محتویات لیست پیوندی حلقوی.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define NIL (struct personel *) NULL
struct personel{
        char name[30] ;
        int persno ;
        int salary ;
        struct personel *next ;
} ;
struct personel *first , *last , *node ;
void list() ;
void enter(int) ;
int main ()
{
   int i ;
   clrscr();
```

```
    char numstr[10] ;
    printf("\n enter number of employee:") ;
    gets(numstr) ;
    first = NIL ;
     enter(atoi(numstr));
    list() ;
    return 0;
}
 //****************************
void list ( )
{
     int i ;
     if (first == NIL) {
        printf("\n<< the list is empty .>>") ;
        getch();
        return;
     }
     last = first ;
     clrscr() ;
     gotoxy(5,4) ;
     printf("  name              em.number   salary ") ;
     gotoxy(5,5) ;
     printf("--------           ---------     -----") ;
     i = 6 ;
     do {
       gotoxy(5, i);
       printf("%s", last -> name) ;
       gotoxy(25, i);
       printf("%d", last -> persno) ;
       gotoxy(38, i);
       printf("%d", last -> salary) ;
       i ++ ;
       last=last -> next ;
      } while (last != first) ;
      gotoxy(5, i++) ;
       printf("************************");
       printf("****************") ;
      gotoxy(10,i++) ;
      printf("press a key to continue.");
      getch() ;
   }
 //*********************
void enter(int n)
```

```
{
   int i;
   char numstr[10];
   for (i = 0; i < n; i++) {
      node = (struct personel *) malloc(sizeof(struct personel));
      node -> next = NIL ;
      if (first == NIL)
         first = last = node ;
      else {
         last -> next = node ;
         last = node ;
      }
      printf("\nenter name of employee:");
      gets(last -> name) ;
      printf("\nenter employee number :");
      gets(numstr) ;
      last -> persno=atoi(numstr) ;
      printf("\n enter salary :") ;
      gets(numstr) ;
      last -> salary = atoi(numstr) ;
   } //end of for
   last -> next = first ;
}
```

خروجی

```
enter number of employee:2
enter name of employee:naser
enter employee number:231
enter salary :5432
enter name of employee:ahmad
enter employee number:4325
enter salary :7654

 name        em.number       salary
-------------  ----------------  ----------
 naser          231             5732
 ahmad          4325             7654
 ****************************************
  press a key to continue.
```

لیست‌های دوپیوندی

مفهوم لیست‌های دوپیوندی در ابتدای این فصل بیان گردید و گفته شد که عناصر لیست‌های دوپیوندی دارای دواشاره‌گر هستند که یک اشاره‌گر به عنصر قبلی و اشاره‌گر دیگر به عنصر بعدی اشاره می‌نماید.

تعریف گره لیست دوپیوندی

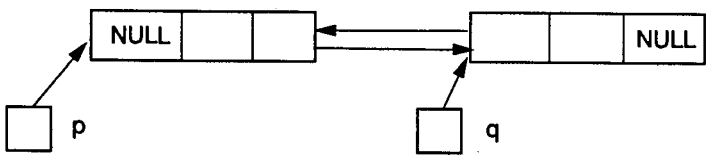
برای تعریف گره لیست دوپیوندی، از دستور struct استفاده می‌شود، زیرا هر گره لیست دوپیوندی یک ساختمان است. دستورات زیر، گره‌ای به نام st را برای لیست دوپیوندی تعریف می‌کند:

```
struct      st{
            struct  st  *link ;
            int stno ;
            struct  st *rlink ;
} ;
```

پیونددادن گره‌های لیست دوپیوندی

اگر p و q دو گره لیست پیوندی باشند، برای اتصال آنها دستورات زیر را اجرا کنید:

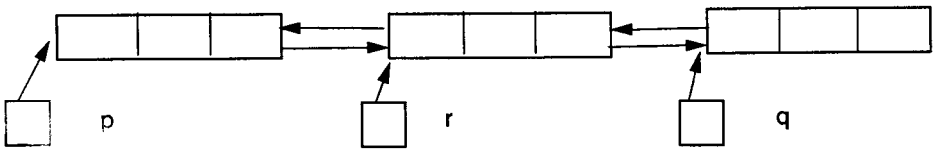
```
p -> llink = NULL ;
p -> rlink = q ;
q -> rlink = NULL ;
q -> llink = p ;
```



درج گره‌ای در لیست دوپیوندی

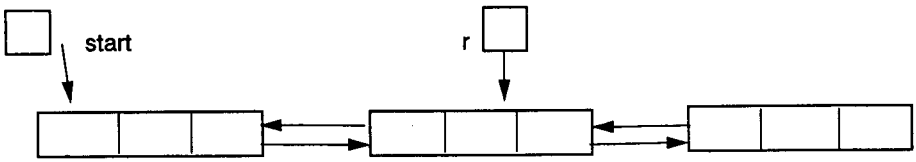
برای درج گره در ابتدای لیست دوپیوندی، باید اشاره‌گر ابتدای لیست را به گره جدید انتقال دهید. برای درج گره r در بین دو گره p و q به صورت زیر عمل می‌شود:

```
r -> rlink = q ;
q -> llink = r ;
p -> rlink = r ;
r -> llink = p ;
```



حذف گره از لیست دوپیوندی

برای حذف از ابتدای لیست، پس از حذف گره باید اشاره‌گر ابتدای لیست را به گره بعدی منتقل کرد. برای حذف گره‌ای که اشاره‌گری مثل r به آن اشاره می‌کند و در وسط لیست قرار دارد، به صورت زیر عمل می‌شود:

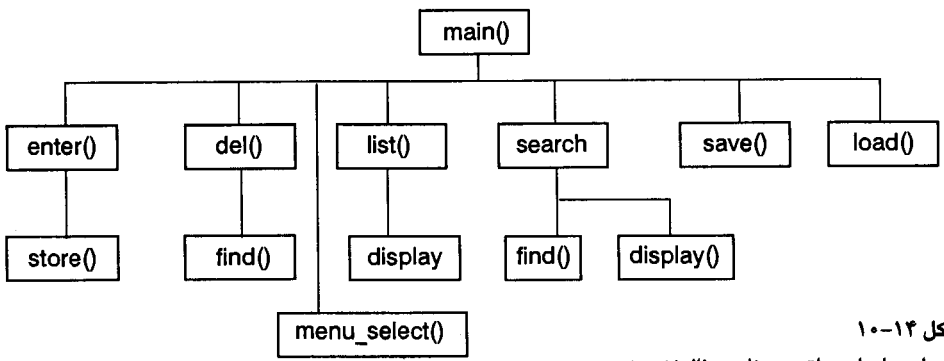



```
r -> llink -> rlink = r -> rlink ;
r -> rlink -> llink = r -> llink ;
r -> llink = NULL ;
r -> rlink = NULL ;
free (r) ;
```

مثال ۷-۱۰

برنامه‌ای که اعمال زیر را انجام می‌دهد:

۱. تشکیل لیست دوپیوندی از مشخصات پستی افراد.
۲. حذف عنصری از لیست دوپیوندی.
۳. مشاهده کلیه عناصر موجود در لیست دوپیوندی.
۴. جستجوی یک اسم در لیست دوپیوندی.
۵. ذخیره کردن لیست دوپیوندی در فایل و انتقال لیست دوپیوندی از فایل به حافظه.



شکل ۱۴-۱۰
نمودار سلسله مراتبی برنامه مثال ۷-۱۰.

شرح وظایف توابع

تابع main(): ارزش‌دهی اولیه اشاره‌گرها و فراخوانی توابع مطابق شکل ۱۴-۱۰.

تابع enter(): خواندن اطلاعات از ورودی و فراخوانی تابع store() جهت واردنمودن این اطلاعات در لیست دوپیوندی.

تابع del(): حذف عنصری از لیست دوپیوندی. برای این منظور نام شخصی را از ورودی خوانده با استفاده از تابع find() این نام را در لیست پیدا می‌کند و آن را حذف می‌نماید و پس از حذف، پیام مناسبی صادر می‌کند.

تابع list(): چاپ عنوان خروجی و فراخوانی تابع display() جهت نمایش اطلاعات در صفحه‌نمایش.

تابع menu_select(): ظاهر نمودن منویی با ۷ گزینه در صفحه‌نمایش و درخواست از کاربر جهت واردنمودن انتخاب وی.

تابع search(): جستجوی یک نام در لیست دوپیوندی به کمک تابع find(). اگر شخص مورد جستجو، در لیست موجود باشد اطلاعات وی با تابع display به خروجی منتقل می‌شود وگرنه پیام مناسبی صادر می‌گردد.

تابع save(): انتقال اطلاعات موجود در لیست دوپیوندی به فایل l.dat.

تابع load(): انتقال اطلاعات موجود در فایل l.dat به لیست دوپیوندی.

تابع store(): تشکیل لیست دوپیوندی.

تابع find(): جستجوی یک نام در لیست دوپیوندی.

تابع display(): چاپ اطلاعات یک عنصر از لیست دوپیوندی در صفحه‌نمایش.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
struct address {
        char name[30] ;
        char street[30] ;
        char city[20] ;
        char state[3] ;
        char zip[10] ;
        struct address *next ;
        struct address *prior ;
} list_entry ;
struct address *start ;
struct address *last ;
```

```
void enter() , display() , search() ;
void save()  , load()    , list() , del();
void display(struct address *info, int *row);
struct address *find(char *);
int menu_select();
struct address *store(struct address *, struct address *);
int main ()
{
      start = last = NULL ;
      for(;;) {
         switch(menu_select()) {
          case 1:  enter();  break ;
          case 2 : del();    break ;
          case 3:  list() ;  break ;
          case 4:  search(); break ;
          case 5:  save();   break ;
          case 6:  load();   break ;
          case 7:  exit(0) ;
         }//end of switch
      }//end of for
}//end of main
//****************
int menu_select()
{
     char s[5];
     clrscr() ;
     gotoxy(25, 4) ;
     printf("1. enter a name ") ;
     gotoxy(25, 6) ;
     printf("2. delete a name ") ;
     gotoxy(25, 8) ;
     printf("3. list all files ") ;
     gotoxy(25, 10) ;
     printf("4. search ") ;
     gotoxy(25, 12) ;
     printf("5. save the file  ") ;
     gotoxy(25, 14) ;
     printf("6. load the file  ") ;
     gotoxy(25, 16) ;
     printf("7. quit ") ;
     do {
       gotoxy(20, 18) ;
       printf("enter your select(1-7):");
```

```
        gets(s);
      } while (atoi(s) < 0 || atoi(s) > 7) ;
      return atoi(s) ;
}
//**********************
void enter ()
{
   struct address *info ;
   int i ;
   char ch ;
   clrscr() ;
   gotoxy(3, 2) ;
   printf("  name        street     city     state    zip");
   gotoxy(3, 3) ;
   printf(" ------------ -------- ");
   printf("--------   -----   ------- ");
   i = 4 ;
   for (;;) {
     info = (struct address *)malloc(sizeof(list_entry)) ;
     if(!info) {
        printf("\n out of memory. press a key ") ;
        getch();
        return ;
     }
     gotoxy(3, i) ;
     gets(info -> name) ;
     if (!info -> name[0]) {
        gotoxy(15, i + 1) ;
        printf("press a key to continue");
        getch() ;
        break ;
     }//end of if
     gotoxy(18, i);
     gets(info -> street) ;
     gotoxy(28, i) ;
     gets(info -> city) ;
     gotoxy(38, i) ;
     gets(info -> state) ;
     gotoxy(45, i) ;
     gets(info -> zip) ;
     i++ ;
     start = store(info, start) ;
     } /* entry loop */
```

```
    }
//**************
struct address *store(struct address *i, struct address *top)
{
   struct address *old, *p ;
   if(last == NULL) {
      i -> next = NULL ;
      i -> prior = NULL ;
      start = i;
      last = i ;
      return i ;
   }
   p = top ;
   old = NULL ;
   while (p != NULL) {
       if(strcmp(p -> name, i -> name) < 0) {
          old = p ;
          p = p -> next ;
       }//end of if
       else {
          if (p -> prior) {
             p -> prior -> next=i ;
             i -> next=p ;
             i -> prior=p -> prior;
             p -> prior=i ;
             return top ;
          }//end of if
          i -> next = p ;
          i -> prior = NULL ;
          p -> prior = i ;
          return i ;
       }//end of else
   } // end of while
   old -> next = i ;
   i -> next = NULL ;
   i -> prior = old ;
   last = i ;
   return start ;
}
//******************
void del()
{
   struct address *info;
```

```
    char name[80];
    gotoxy(20, 20) ;
    printf(" enter name for delete : ") ;
    gets(name) ;
    info = find(name) ;
    if(info == NULL) {
        gotoxy(10, 20) ;
        printf(" name not found! press a key to continue.");
        getch() ;
    }
    if (info)
      if (start == info)   {
          start = info -> next ;
          if(start)
              start -> prior = NULL ;
          else
              last = NULL ;
      } //end of if
      else  {
          info -> prior -> next = info -> next;
              if(info != last)
                  info -> next -> prior = info -> prior;
              else
                  last = info -> prior ;
      } //end of else
      free(info) ;
      gotoxy(10,20) ;
      printf("name deleted, press a key to continue.");
      getch() ;
}
struct address *find(char *name)
{
     struct address *info ;
     info = start ;
     while(info != NULL) {
        if (strcmp(name, info -> name) == 0)
            return info;
        info = info -> next ;
     }
     return NULL ;
}
//*****************
void list ()
```

```
{
    struct address *info ;
    int i ;
    info = start ;
    clrscr() ;
    gotoxy(3, 2) ;
    printf("  name        street     city     state    zip");
    gotoxy(3, 3) ;
    printf(" ------------ -------- -");
    printf("-------  -----  ------- ");
    i = 4 ;
    while(info != NULL) {
        display(info, &i) ;
        info = info -> next ;
    }
    gotoxy(15, i + 2) ;
    printf("press a key to continue.");
    getch() ;
}
//*******************
void display(struct address *info, int *row)
{
  gotoxy(3, *row) ;
  printf("%s", info -> name) ;
  gotoxy(18, *row) ;
  printf("%s", info -> street) ;
  gotoxy(28, *row) ;
  printf("%s", info -> city) ;
  gotoxy(38, *row) ;
  printf(info -> state) ;
  gotoxy(47, *row) ;
  printf(info -> zip) ;
  *row = *row + 1 ;
}
//**************************
void search()
{
    char name[40] ;
    int i ;
    struct address *info;
    gotoxy(20, 20) ;
    printf(" enter name to find : ");
    gets(name) ;
```

```
    info = find(name) ;
    if(info == NULL) {
        gotoxy(10, 20) ;
        printf(" name not found! press a key to continue.");
        getch() ;
    }//end of if
    else {
        clrscr() ;
        gotoxy(3, 2) ;
        printf("  name        street   city     state   zip");
        gotoxy(3, 3) ;
        printf(" ------------ -------");
        printf("-  --------    -----  ------- ") ;
        i = 4 ;
        display(info ,&i) ;
        gotoxy(15, i + 2) ;
        printf("press a key to continue.");
        getch() ;
    }//end of else
}
//**********************
void save()
{
    struct address *info ;
    FILE *fp ;
    if((fp = fopen("I.dat","wb")) == NULL) {
      printf("\n cannot open file. ") ;
      getch();
      exit(1) ;
    }//end of if
    gotoxy(20, 20) ;
    printf("<< saving file >>") ;
    info = start ;
    while(info) {
      fwrite(info, sizeof(struct address), 1, fp);
      info = info -> next ;
    }//end of while
    fclose(fp) ;
    gotoxy(15, 22) ;
    printf("file successfuly saved press a key...") ;
    getch() ;
}
//********************
```

```
void load ()
{
    struct address *info , *temp = NULL;
    FILE *fp ;
    fp = fopen("l.dat","rb") ;
    if(fp == NULL)  {
      printf("\n cannot open file.");
      getch();
     exit(1) ;
    }
    while(start) {
      info = start -> next ;
      free(info) ;
      start = info ;
    }
    gotoxy(20,20) ;
    printf(" << loading file  >> ") ;
    start = NULL ;
    while (!feof(fp)) {
        info = (struct address *)  malloc(sizeof(struct address)) ;
        if(1 != fread(info, sizeof(struct address), 1, fp))
            break ;
        if(start == NULL)  {
           temp = start = info ;
           info -> prior = NULL ;
           info -> next = NULL ;
        }//end of if
        else  {
           info -> next = NULL ;
           temp -> next = info ;
           info -> prior = temp ;
           temp = info;
        } //end of else
    }//end of while
    last = temp ;
    fclose(fp) ;
    gotoxy(15,22) ;
    printf("file successfuly loaded press a key ...") ;
    getch();
}
```

عملکرد برنامه

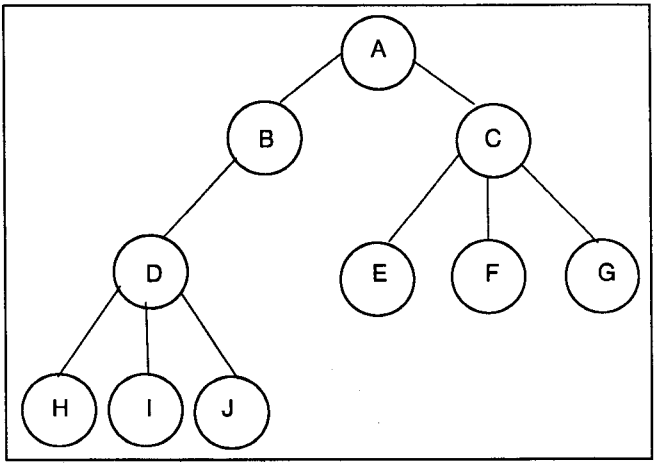
جهت آشنایی با امکانات برنامه، پیشنهاد می‌شود برنامه را دوبار اجرا کنید. در اجرای اول، اطلاعات از طریق صفحه کلید وارد لیست پیوندی شده توسط تابع ()save در فایل l.dat ذخیره شوند. سپس از برنامه خارج شده آن را مجدداً اجرا کنید. در اجرای دوم، با انتخاب گزینه ۶ از منو اطلاعات موجود در فایل l.dat را توسط تابع ()load به لیست پیوندی منتقل کنید. خروجی دو اجرا را با هم مقایسه کنید.

منوی برنامه

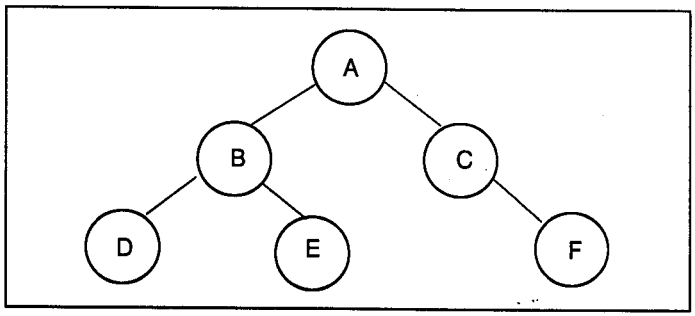
```
1. enter a name
2. delete a name
3. list all files
4. search
5. save th file
6. load the file
7  quit
enter your select(1-7):
```

درختها

درختها نوع دیگری از ساختمان داده هستند که عناصر آن با اشاره‌گرهایی به هم متصل‌اند. هر یک از عناصر درخت را **گره** (node) گویند. نمونه‌ای از درخت را می‌توان مانند شکل ۱۵-۱۰ نمایش داد. هر دایره، یک گره درخت است. این درخت را یک درخت **عمومی** گویند. اولین گره هر درخت **ریشه** نام دارد. بنابراین، ریشه درخت کامپیوتری، برخلاف ریشه درختهای طبیعی، در بالا قرار دارد. لذا می‌توان گفت که هر درخت مجموعه‌ای از یک یا چند گره است که گره اول ریشه نام دارد. گره‌هایی که بلافاصله در زیر گره دیگر قرار دارند، **فرزندان** آن گره‌اند. به عنوان مثال، گره‌های B و C فرزندان گره A و گره‌های H، I و J فرزندان گره D هستند. گره بالاتر را گره **پدر** نیز می‌گویند. مثلاً گره A گره پدر گره‌های B و C است. گره‌های پایانی هر درخت را **برگهای** درخت می‌گویند. برگهای درخت شکل ۱۵-۱۰ عبارت‌انداز: E، F، G، H، I و J.



شکل ۱۵-۱۰
نمونه‌ای از یک درخت.



شکل ۱۶-۱۰
نمونه‌ای از درخت دودویی.

مفهوم دیگری که در درخت مورد استفاده قرار می‌گیرد، **سطح** هر گره است. سطح گره ریشه برابر با یک است و سطح گره‌های بعدی، برابر با سطح گره پدر به اضافه یک است. به عنوان مثال، سطح گره B برابر ۲ و سطح گره F برابر با ۳ است. **ارتفاع** یا عمق درخت، به بیشترین سطح گره‌های آن گفته می‌شود. ارتفاع درخت شکل ۱۵-۱۰ برابر با ۴ است.

درخت دودویی

اگر هر گره حداکثر دو فرزند داشته باشد، درخت حاصل را درخت **دودویی** گویند. لذا درخت دودویی، یا تهی است و یا مجموعه محدودی از گره‌هاست. نمونه‌ای از درخت دودویی در شکل ۱۶-۱۰ آمده است.

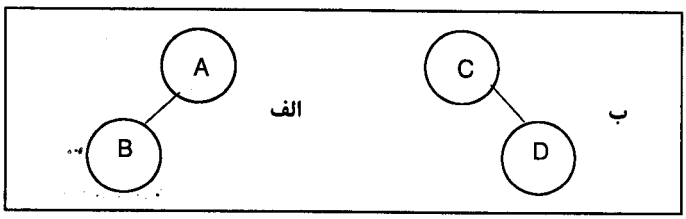
درختهای دودویی در موارد زیر با سایر درختها متفاوت است:

۱. درخت دودویی می‌تواند تهی باشد.

۲. ترتیب فرزندان در درخت دودویی مهم است. لذا دو درخت دودویی شکل ۱۷-۱۰ با هم متفاوت‌اند، زیرا درخت الف دارای فرزند چپ و درخت ب دارای فرزند راست است.

ساختار گره درخت دودویی

هر گره درخت دودویی دارای سه بخش است: دو بخش اشاره‌گر و یک بخش داده. نمونه‌ای از گره درخت دودویی را در شکل ۱۸-۱۰ مشاهده می‌کنید.



شکل ۱۷-۱۰
دو درخت دودویی متفاوت.

اشاره‌گر راست	داده	اشاره‌گر چپ

شکل ۱۸-۱۰
نمونه‌ای از گره درخت دودویی.

اشاره‌گر سمت چپ به فرزند سمت چپ و اشاره‌گر سمت راست به فرزند سمت راست اشاره می‌کند. اگر در شکل ۱۶-۱۰ دقت کنید، هر گره را می‌توان برای گره‌های پایین‌تر از خودش، به عنوان ریشه تلقی کرد. به عبارت دیگر، اگر از ریشه شروع کنیم، این گره دارای دو زیردرخت (subtree) است که ریشه‌های آنها B و C هستند. بنابراین، هر اشاره‌گر سمت چپ یا راست، به یک زیردرخت اشاره می‌کنند.

تعریف گره درخت

برای تعریف گره درخت از دستور struct استفاده می‌شود، زیرا هر گره درخت یک ساختمان است. به عنوان مثال، دستور زیر، گره‌ای به نام tree را تعریف می‌کند:

```
struct      tree {
            struct  tree *left ;
            int stno ;
            struct tree *right ;
} ;
```

با این دستور، ساختار گره درخت طوری تعریف می‌شود که حاوی یک پیوند چپ به نام left و یک پیوند راست به نام right و یک بخش داده به نام stno است.

ایجاد گره درخت

برای ایجاد گره درخت، باید اشاره‌گری از گره درخت تعریف کرد و حافظه‌ای را توسط malloc() اختصاص داد. دستورات زیر را در نظر بگیرید:

```
struct  tree    *t1 ;
t1 = (struct tree *) malloc (sizeof (struct tree)) ;
```

با این دستورات، اشاره‌گر t1 از نوع tree تعریف شده، حافظه‌ای اختصاص می‌یابد و آدرس آن در t1 قرار می‌گیرد.

ایجاد درخت جستجوی دودویی

نوعی از درخت دودویی به نام درخت جستجوی دودویی است. برای ایجاد درخت جستجوی دودویی می‌توان الگوریتم زیر را پیشنهاد کرد:

۱. اولین اطلاعات ورودی در ریشه قرار می‌گیرد.

۲. اطلاعات دومی با ریشه مقایسه شده، چنانچه کوچکتر از ریشه باشد در سمت چپ وگرنه در سمت راست ریشه قرار می‌گیرد.

۳. اطلاعات بعدی، همانند اطلاعات دومی با ریشه مقایسه می‌شود، اگر از ریشه کوچکتر باشد، به سمت چپ، وگرنه به سمت راست می‌رود و با گره بعدی درخت مثل ریشه برخورد می‌کند. این روند ادامه می‌یابد تا جای مناسبی در درخت پیدا کند.

به عنوان مثال، فرض کنید، اعداد ۲۰، ۵، ۲۵، ۱۸، ۲۳، ۱۷، ۲۶ و ۱۹ می‌خواهند در یک درخت جستجوی دودویی قرار گیرند. این اعداد مانند شکل ۱۹-۱۰ در درخت قرار خواهند گرفت. شیوه تشکیل این درخت بدین صورت است: ابتدا عدد ۲۰ وارد می‌شود و در ریشه درخت قرار می‌گیرد. عدد بعدی، ۵است که با ریشه مقایسه می‌شود و در سمت چپ ریشه قرار می‌گیرد. عدد بعدی ۲۵ است که با مقایسه با ریشه، در سمت راست آن قرار می‌گیرد. عدد ۱۸ با ۲۰ مقایسه می‌شود و به سمت چپ می‌رود، با ۵ مقایسه می‌شود و در سمت راست این گره قرار می‌گیرد. عدد ۲۳ با ۲۰ مقایسه می‌شود، به سمت راست می‌رود، با ۲۵ مقایسه می‌شود و در سمت چپ این گره قرار می‌گیرد. عدد ۱۷ با ۲۰ مقایسه می‌شود، به سمت چپ می‌رود، با ۵ مقایسه می‌شود، به سمت راست می‌رود، با ۱۸ مقایسه می‌شود، و در سمت چپ این گره قرار می‌گیرد. عدد ۲۶ با ۲۰ مقایسه می‌شود، به سمت راست می‌رود، با ۲۵ مقایسه می‌شود و در سمت راست این گره قرار می‌گیرد. عدد ۱۹ با ۲۰ مقایسه می‌شود، به سمت چپ می‌رود، با ۵ مقایسه می‌شود. به سمت راست می‌رود، با ۱۸ مقایسه می‌شود و در سمت راست این گره قرار می‌گیرد. بدین ترتیب درخت دودویی شکل ۱۹-۱۰ ایجاد خواهد شد.

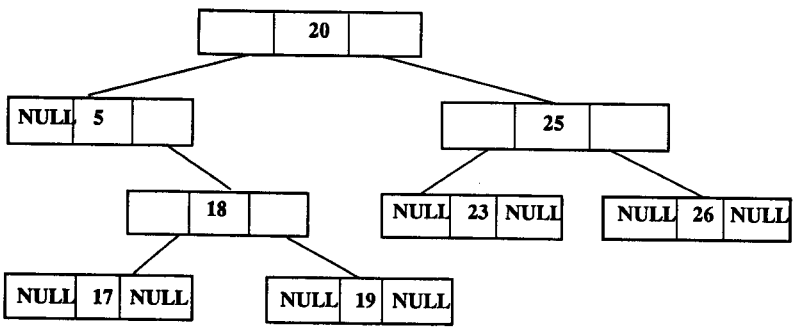
پیمایش درخت

پیمایش درخت، یکی از اعمال متداول در مورد درختهاست. منظور از **پیمایش** درخت، دستیابی به هر یک از گره‌های درخت است، به طوری که هر گره درخت فقط یک بار ملاقات شود. سه روش برای پیمایش درخت وجود دارد که عبارت‌انداز: روش inorder ، روش preorder و روش postorder .

پیمایش inorder درخت دودویی

پیمایش inorder درخت دودویی به این صورت انجام می‌شود: حرکت به طرف پایین و به سمت چپ انجام می‌شود و این عمل تا آخرین گره صورت می‌گیرد. این گره را بازیابی می‌کنیم و به سمت راست حرکت می‌کنیم. اگر حرکت به سمت راست امکان‌پذیر نباشد، یک یا چند گره به طرف بالا حرکت می‌کنیم. در گره جدید، مثل ریشه عمل می‌کنیم. لذا الگوریتم پیمایش inorder را می‌توان به صورت زیر نوشت :

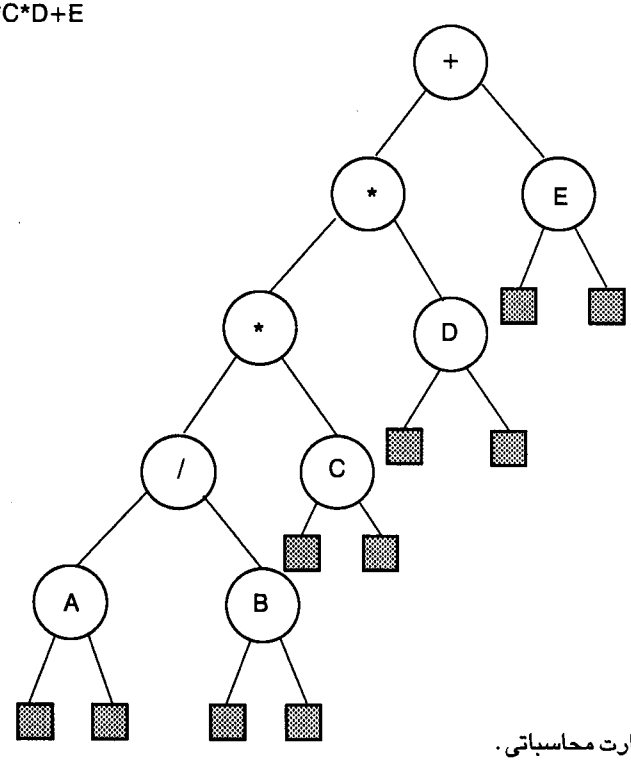
۱. پیمایش زیردرخت چپ ۲. ملاقات ریشه ۳. پیمایش زیردرخت راست



شکل ۱۹-۱۰ درخت دودویی .

اکنون درخت شکل ۲۰-۱۰ را در نظر بگیرید. پیمایش inorder این درخت به صورت زیر خواهد بود:

A/B*C*D+E



شکل ۲۰-۱۰

نمونه‌ای از درخت دودویی برای عبارت محاسباتی.

مثال ۸-۱۰

تابع ()inorder، درخت را به طور بازگشتی به صورت inorder پیمایش می‌کند. در این تابع، آرگومان s ابتدا به ریشه درخت اشاره می‌کند.

```
inorder (struct tree *s) {
        if (s == NULL)
            return ;
        inorder(s -> left) ;
        printf("\n %d", s -> id) ;
        inorder(s -> right) ;
}
```

پیمایش preorder

در این روش، از ریشه شروع کرده، آن را ملاقات می‌کنیم، سپس به طرف چپ حرکت کرده، تمام گره‌ها را ملاقات می‌کنیم و این روند را تا رسیدن به گره‌ای که اشاره‌گر چپ آن تهی باشد ادامه می‌دهیم. در این گره، به سمت راست حرکت می‌کنیم و با گره جدید مثل ریشه برخورد می‌کنیم. اگر حرکت به سمت راست امکان‌پذیر نباشد، به طرف بالا

می‌رویم. پیمایش preorder درخت شکل ۲۰-۱۰ به صورت زیر است:

+/ABCDE**

بنابراین الگوریتم روش پیمایش preorder را می‌توان به صورت زیر نوشت:

۱. ملاقات ریشه
۲. پیمایش زیردرخت سمت چپ
۳. پیمایش زیردرخت سمت راست

مثال ۹-۱۰

تابع ()preorder درخت را به طور بازگشتی و به صورت preorder پیمایش می‌کند. در این تابع، اشاره‌گر s ابتدا به ریشه درخت اشاره می‌کند.

```
preorder (struct tree *s) {
    if (s == NULL)
        return ;
    printf("\n %d", s -> id);
    preorder(s -> left);
    preorder(s -> right);
}
```

پیمایش postorder

در پیمایش postorder ابتدا زیردرخت چپ، سپس زیردرخت راست و بعد، ریشه ملاقات می‌شود. آنقدر به سمت چپ حرکت می‌کنیم تا به گرهی برسیم که امکان حرکت به چپ وجود نداشته باشد. آن را ملاقات می‌کنیم، سپس به سمت راست حرکت می‌کنیم، اگر حرکت به راست ممکن نباشد، به طرف بالا حرکت می‌کنیم، در گره جدید، مثل ریشه عمل می‌کنیم. الگوریتم این روش را می‌توان به صورت زیر نوشت.

۱. پیمایش زیردرخت چپ
۲. پیمایش زیردرخت راست
۳. ملاقات ریشه

پیمایش postorder درخت شکل ۲۰-۱۰ به صورت زیر است:

AB/C*D*E+

مثال ۱۰-۱۰

تابع ()postorder درخت را به طور بازگشتی و به صورت postorder پیمایش می‌کند. در این تابع، اشاره‌گر s ابتدا به ریشه درخت اشاره می‌نماید.

```
postorder (struct tree *s) {
      if (s == NULL)
         return ;
      postorder(s -> left) ;
      postorder(s -> right) ;
      printf("\n %d", s -> id) ;
}
```

مثال ۱۱-۱۰

برنامه‌ای که نام و شماره تلفن تعدادی از مشتریان شرکت مخابرات را از ورودی خوانده، در یک درخت دودویی قرار می‌دهد. جستجوی فرد خاصی در درخت، شماره تلفن آن و نمایش کلیه اطلاعات موجود در درخت، از جمله وظایف این برنامه است.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#define  TRUE  1
void input ();
int menu_select();
void maketree(struct tree *);
void find (struct tree *);
void inorder (struct tree *s);
struct tree {
      struct tree *left ;
      char   name[20] ;
      int    telephone ;
      struct tree *right ;
} ;
struct tree *start , *node ;
char pname[20] ;
int main ()
{
   start = NULL ;
   while (TRUE)  {
      switch(menu_select())  {
         case 1: input() ; break ;
         case 2:
            gotoxy(20,14) ;
            printf("enter name for search:");
            gets(pname) ;
            find(start);
            getch() ;
            break ;
         case 3:
            clrscr() ;
            printf("\n<<travers tree in inorder>>\n");
            gotoxy(6,4) ;
            printf("    name          telephone");
            gotoxy(6, 5) ;
            printf("-------------      -------");
```

```
                inorder(start) ;
                printf("\n press a key  to continue ..") ;
                getch() ;
                break ;
            case 4:
                exit(0) ;
        } //end of switch
    } //end of while
}//end of main
//**********************
void input ()
{
    int i ;
    char numstr[10] ;
    clrscr() ;
    gotoxy(4, 3) ;
    printf("    name          telephone     \n");
    gotoxy(4,4) ;
    printf("------------      -------     ");
    i = 5 ;
    while (TRUE)  {
      node = (struct tree *) malloc(sizeof(struct tree)) ;
      node -> left = NULL ;
      node -> right = NULL ;
      gotoxy(4, i) ;
      gets(node -> name) ;
      if(!node -> name[0])  {
          gotoxy(5,i + 2) ;
          printf("press a key to continue");
          getch() ;
          break ;
      }
      gotoxy(23,i) ;
      gets(numstr) ;
      node -> telephone = atoi(numstr) ;
      if (start == NULL)
          start = node ;
      else
          maketree(start) ;
      i++ ;
    }//end of while
}
//**********************
```

```
int menu_select()
{
   int choice ;
   char s[5] ;
   clrscr() ;
   gotoxy(20, 4) ;
   printf("1- enter customer in tree.");
   gotoxy(20, 6) ;
   printf("2- search for customer in tree.");
   gotoxy(20,8) ;
   printf("3- print information on screen. ") ;
   gotoxy(20, 10) ;
   printf("4- quit from program. ") ;
   do {
      gotoxy(20, 12) ;
      printf("Enter your select(1-4):");
      gets(s) ;
      choice = atoi(s) ;
   } while (choice < 0 || choice > 4);
   return choice ;
}
//*************************
void maketree(struct tree *top)
  {
     struct tree *help ;
     help = top ;
     while (help != NULL) {
        if(node ->telephone>help ->telephone) {
          if (help -> right != NULL)
             help = help -> right ;
          else {
             help -> right=node ;
             break ;
          }//end of else
        } //end of if
        else {
           if (help -> left != NULL )
              help = help -> left ;
           else {
              help -> left=node ;
              break ;
           } //end of else
        } //end of else
```

```
    }//end of while
}
//**********************
void find (struct tree *s)
{
   if (s == NULL)
      return ;
   find(s -> left) ;
   if (strcmp(s -> name, pname) == 0) {
         printf("\n <<%s>>has telphone:", pname);
         printf(" %d",s -> telephone) ;
         return ;
   }
   find(s -> right) ;
}
//*************************
void inorder (struct tree *s)
{
      if (s == NULL)
          return ;
      inorder(s -> left) ;
      printf("\n %s", s -> name) ;
      printf("\t\t\t %d", s -> telephone);
      inorder(s -> right) ;
}
```

عملکرد برنامه

پس از اجرای برنامه، منویی با ۴گزینه ظاهر می‌شود. گزینه اول مشخصات مشتریان را در درخت دودویی قرار می‌دهد. گزینه دوم با دریافت نام مشتری از ورودی، آن را در درخت جستجو می‌کند. گزینه سوم محتویات کل درخت را چاپ می‌نماید و گزینه چهارم، از برنامه خارج می‌شود.

منوی برنامه

1. enter customer is tree.
2. search for customer in tree.
3. print information on screen.
4. quit from program.
Enter your select (1-4) :

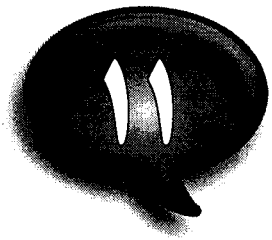
تمرینات

۱. چه تفاوتهای عمده‌ای بین آرایه و لیست پیوندی وجود دارد؟

۲. تفاوتهای صف و پشته را نام ببرید.

۳. لیست پیوندی حلقوی و صف حلقوی را تشریح کنید.

۴. امتیازات لیست دوپیوندی نسبت به لیست یک پیوندی چیست ؟
۵. مفهوم پیمایش لیست پیوندی و درخت را بیان کنید.
۶. روشهای پیمایش درخت را با هم مقایسه کنید.
۷. روشهای پیمایش postorder و preorder درخت را به طور غیربازگشتی بنویسید.
۸. یک لیست پیوندی از کارکنان دانشگاه تشکیل دهید. اطلاعاتی که باید ذخیره شوند عبارت‌انداز: نام کارمند، شماره کارمندی، حقوق پایه، وضعیت تأهل. لیست پیوندی براساس شماره کارمندی مرتب باشد. برنامه با استفاده از منویی، کلیه اعمال زیر را انجام دهد :
   ۱. ورود اطلاعات کارکنان
   ۲. حذف کارمندی از لیست پیوندی
   ۳. گزارش کامل از لیست پیوندی
   ۴. ذخیره لیست پیوندی در فایل
   ۵. بازیابی لیست پیوندی از فایل
۹. تابعی بنویسید که آدرس شروع دو لیست پیوندی از کاراکترها را دریافت کرده، آنها را با هم الحاق کند و اشاره‌گر ابتدای لیست جدید را برگرداند. سپس برنامه‌ای بنویسید که از آن استفاده کند.
۱۰. تابعی بنویسید که آدرس شروع دو لیست پیوندی مرتب از اعداد صحیح را دریافت کرده، آن دو را طوری با هم ادغام کند که لیست نتیجه نیز مرتب باشد (لیست‌ها به طور صعودی مرتب‌اند).
۱۱. برنامه‌ای بنویسید که ۱۰۰۰ عدد تصادفی صحیح را ایجاد کند و به طور صعودی مرتب، در یک لیست پیوندی قرار دهد. برنامه باید میانگین این اعداد را محاسبه نماید.
۱۲. برنامه‌ای بنویسید که اسامی دانشجویان کلاسی را در یک درخت دودویی قرار دهد و سپس نامی را از ورودی خوانده، آن را جستجو نماید. برنامه باید قادر باشد درخت را بر روی دیسک ذخیره و بازیابی کند.



روشهای مرتب‌سازی و جستجو

مرتب‌سازی و جستجو متداولترین اعمال در کامپیوتر هستند. الگوریتم‌های مرتب‌سازی و یا جستجو در اکثر برنامه‌های بانک‌های اطلاعاتی، کامپایلرها، مفسرها و سیستم‌های عامل مورد استفاده قرار می‌گیرند. مرتب‌سازی و جستجو به روش‌های مختلفی انجام می‌شوند که در این فصل چند روش مورد بررسی قرار می‌گیرند.

روشهای مرتب‌سازی

منظور از مرتب‌سازی یک لیست و یا یک فایل، قراردادن عناصر آنها به ترتیب از کوچک به بزرگ (مرتب‌سازی صعودی) و یا از بزرگ به کوچک (مرتب‌سازی نزولی) است. اگر یک لیست مرتب به نام L و به طول n داشته باشیم، در این صورت خواهیم داشت:

صعودی: `L(0) <= L(1) <= L(2) <= ... <= L(n-1)`

نزولی: `L(0) >= L(1) >= L(2) >= ... >= L(n-1)`

گرچه در توربو C و بورلند C تابعی به نام ()qsort وجود دارد که عمل مرتب‌سازی یک لیست را انجام می‌دهد ولی هر برنامه‌نویس به دلایل زیر باید با الگوریتم‌های مرتب‌سازی آشنا باشد.

۱. تابع ()qsort ممکن است در بسیاری از موارد قابل استفاده نباشد.
۲. ممکن است تابع ()qsort برای مرتب‌سازی داده‌های موردنظر برنامه‌نویس، مفید نباشد.

به طور کلی دو دسته از الگوریتم‌های مرتب‌سازی وجود دارند:

۱. الگوریتم‌هایی که آرایه‌ها را مرتب می‌کنند (چه در حافظه و چه در فایل تصادفی موجود روی دیسک).
۲. الگوریتم‌هایی که فایلهای ترتیبی را که بر روی دیسک یا نوار مغناطیسی قرار دارند، مرتب می‌کنند.

در این فصل فقط به بررسی الگوریتم‌های دسته اول پرداخته می‌شود. زیرا این الگوریتم‌ها بیشتر از دسته دوم مورد استفاده قرار می‌گیرند. سه روش مرتب‌سازی که در این فصل مورد بررسی قرار می‌گیرند عبارتنداز:

۱. روش مرتب‌سازی تعویضی
۲. روش مرتب‌سازی انتخابی
۳. روش مرتب‌سازی درج

برای پی بردن به مفاهیم هر یک از روشهای مرتب‌سازی، فرض کنید که مشخصات مربوط به دانشجویان شرکت‌کننده در یک کلاس که تعداد آنها ۱۰۰ نفر است بر روی برگه‌های کاغذ نوشته شده، به هر یک از آنها شماره‌ای اختصاص داده شده، ترتیب آنها نیز از بین رفته است. برای مرتب‌سازی این ۱۰۰ برگ کاغذ به هر یک از سه روش مرتب‌سازی باید به طریق زیر عمل شود:

۱. **روش تعویضی** : در این روش، ۱۰۰ برگ کاغذ را روی یک میز قرار داده از اولین برگه تا آخرین برگه را بررسی کرده دو برگه متوالی را با یکدیگر مقایسه می‌کنیم. اگر شماره اولین برگه از شماره دومین برگه بزرگتر باشد، جای این دو را عوض می‌کنیم و سپس برگه دوم و سوم را مقایسه کرده و مثل حالت قبلی عمل می‌کنیم و این روند را تا آخرین برگه ادامه می‌دهیم و سپس از ابتدای لیست شروع کرده و عمل قبلی را تا مرتب‌شدن کامل برگه‌ها ادامه می‌دهیم.

۲. **روش انتخابی** : ۱۰۰ برگ کاغذ را بر روی میز قرار داده و سپس برگه‌ای که کوچکترین شماره را دارد انتخاب کرده در دست خود نگهداری می‌کنیم و سپس از میان برگه‌های باقیمانده، کوچکترین شماره را انتخاب و بر روی برگهٔ قبلی موجود در دست خود قرار می‌دهیم و این روند را تا تمام شدن کلیه برگه‌های موجود روی میز ادامه می‌دهیم. در خاتمه، برگه‌هایی که در دست ما است، مرتب می‌باشند.

۳. **روش درجی** :کلیه برگه‌های کاغذ را در دست خود می‌گیریم (برگه‌های نامرتب) و سپس برگه‌ها را یکی‌یکی بر روی میز طوری قرار می‌دهیم که کوچکترین شماره در سمت چپ و بزرگترین شماره در سمت راست قرار گیرد. به عنوان مثال، اگر اولین برگه‌ای که روی میز گذاشته شده است شماره ۹۰ باشد و شماره برگه دوم برابر با ۷۰ باشد، برگه با شماره ۷۰ در سمت چپ برگهٔ شماره ۹۰ قرار می‌گیرد. این روند را تا تمام شدن برگه‌های موجود در دست خود ادامه می‌دهیم. در خاتمه، برگه‌های موجود روی میز مرتب می‌باشند.

مقایسه الگوریتم‌های مرتب‌سازی

برای پیاده‌سازی هر یک از روشهای مرتب‌سازی، ممکن‌است الگوریتم‌های مختلفی وجود داشته باشند. این الگوریتم‌ها را می‌توان براساس فاکتورهای زیر با یکدیگر مقایسه نمود.

۱. سرعت متوسط مرتب‌سازی اطلاعات
۲. سرعت الگوریتم مرتب‌سازی در بهترین و بدترین حالت
۳. رفتار الگوریتم (طبیعی یا غیرطبیعی)
۴. شرکت عناصر مساوی در مرتب‌سازی

سرعت الگوریتم مرتب‌سازی یکی از خصیصه‌های مهم الگوریتم است که به تعداد مقایسه‌ها و همچنین تعداد تعویضهایی که باید انجام گیرد بستگی دارد. سرعت اجرای بعضی از الگوریتم‌ها با تعداد عناصر لیست نسبت توانی دارد و بعضی دیگر نسبت لگاریتمی.

سرعت اجرای الگوریتم در بدترین و بهترین حالت نیز مهم است. زیرا ممکن است یکی از این دو حالت برای عمل مرتب‌سازی انتخاب گردد. ممکن است سرعت متوسط اجرای الگوریتم خوب باشد ولی در بدترین حالت، سرعت اجرای آن خیلی کم باشد.

نحوه عمل الگوریتم مرتب‌سازی باید حالت طبیعی داشته باشد. مثلاً اگر الگوریتم بر روی یک لیست مرتب شده عمل کند باید نسبت به حالتی که بر روی لیست نامرتب عمل می‌کند متفاوت و بهتر باشد. این مطلب نیز به تعداد مقایسه‌ها و تعویضهایی که باید انجام گیرد بستگی دارد.

برای فهم آخرین فاکتور مقایسه الگوریتم‌ها (شرکت عناصر مساوی در مقایسه) تصور کنید که یک بانک اطلاعاتی از آدرس افراد موجود است. این بانک اطلاعاتی براساس کدپستی و در داخل کدپستی براساس نام افراد مرتب است. اگر آدرس جدیدی به بانک اطلاعاتی اضافه گردد و این بانک اطلاعاتی بر اساس کدپستی مرتب گردید، نباید مجدداً براساس نام مرتب شود. برای تضمین این مطلب، الگوریتم مرتب‌سازی نباید جای دو کلید مساوی را تعویض نماید.

نمونه‌ای از یک روش مرتب‌سازی تعویضی

یکی از روشهای مرتب‌سازی که ذکر گردید، روش تعویضی بود. یکی از الگوریتم‌هایی که این روش را پیاده‌سازی می‌کند الگوریتم مرتب‌سازی حبابی است که یکی از بدترین الگوریتم‌ها می‌باشد.

در روش مرتب‌سازی حبابی، عناصر متوالی یک لیست از ابتدای لیست با یکدیگر مقایسه شده و چنانچه مرتب باشند، جای آنها عوض می‌شود و به این ترتیب در اولین مرحله، بزرگترین عنصر لیست به انتهای لیست منتقل می‌گردد. مراحل بعدی نیز مثل مرحلهٔ اول انجام می‌شوند تا این که لیست به طور کامل مرتب می‌گردد (در هر مرحله، بزرگترین عنصر، به انتهای لیست) منتقل می‌گردد. گاهی ممکن است مقایسه از انتهای لیست به طرف ابتدای آن انجام گیرد.

به عنوان مثال، فرض کنید اعداد زیر باید به روش حبابی مرتب شوند. در مرحله اول، ۹ با ۷ مقایسه می‌شود و جابجا می‌شوند. سپس ۹ با ۱۰ مقایسه می‌شود و جابجایی انجام نمی‌شود. در آخر ۴ با ۱۰ مقایسه می‌شود و جابجایی صورت می‌گیرد. این روند را ادامه دهید تا نتایج زیر حاصل شود:

شروع	۴	۱۰	۷	۹
مرحله اول	۱۰	۴	۹	۷
مرحله دوم	۱۰	۹	۴	۷
مرحله سوم	۱۰	۹	۷	۴

مثال ۱-۱۱

برنامه‌ای که رشته‌ای را از ورودی خوانده و عناصر آن را به طور صعودی مرتب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void bubble(char *, int);
int main()
{
    char s[80] ;
    clrscr();
    printf(" enter a string :");
```

```
    gets(s) ;
    bubble(s, strlen(s));
    printf("\n the sorted string is: %s", s);
    getch();
    return 0;
}
//**********************
void bubble(char *item, int count)
{
    register int a,b ;
    register char t ;
    for(a= 1 ; a < count; a++)
      for(b = count - 1; b >= a; --b)
        if(item[b-1] > item[b]) {
          t = item[b-1] ;
          item[b-1]=item[b] ;
          item[b]=t ;
        } //end of if
}
```

با فرض این که s حاوی رشتهٔ dcab باشد مراحل مرتب‌سازی به صورت زیر خواهد بود:

d	**c**	**a**	**b**	**شروع**
c	**a**	**b**	**d**	**مرحله اول**
a	**b**	**c**	**d**	**مرحله دوم**
a	**b**	**c**	**d**	**مرحله سوم**

برای تجزیه و تحلیل الگوریتم مرتب‌سازی باید تعداد مقایسه‌ها و تعویضها را در بدترین، متوسط و بهترین حالت تعیین نمود. در الگوریتم مرتب‌سازی حبابی، تعداد مقایسه‌ها در هر حال یکسان است. اگر طول لیست، n فرض شود، همانطور که در الگوریتم حبابی مشاهده می‌گردد، دو حلقه تکرار تودرتو وجود دارد که اولین حلقه n-1 بار و دومین حلقه، n/2 بار تکرار می‌شود لذا:

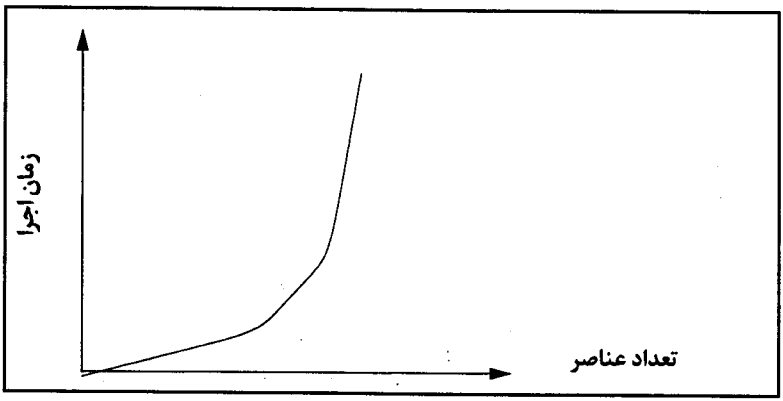
تعداد مقایسه‌ها = n / 2 (n-1) = 1 / 2 (n^2-n)

تعداد تعویضها در مرتب‌سازی حبابی: در بهترین حالت (وقتی که لیست مرتب باشد) صفر است و در بدترین حالت (وقتی که لیست به طور معکوس مرتب باشد) برابر با 3 / 2 (n^2-n) و تعداد متوسط تعویضها برابر با 3/4(n^2-n) است. رابطهٔ بدترین حالت به صورت زیر به دست آمده است:

تعداد تعویضها * تعداد مقایسه‌ها

1 / 2 (n^2-n) * 3 = 3 / 2 (n^2-n)

با توجه به روابط فوق، می‌توان گفت که الگوریتم مرتب‌سازی حبابی، یک الگوریتم توانی است. زیرا زمان اجرای آن با مربع تعداد عناصر آن متناسب است (شکل ۱-۱۱).



شکل ۱-۱۱
ارتباط بین تعداد عناصر و زمان اجرا در مرتب‌سازی حبابی.

روش مرتب‌سازی انتخابی

در روش مرتب‌سازی انتخابی به این صورت عمل می‌گردد که در بین عناصر لیست، کوچکترین عنصر لیست پیدا شده، جای آن با عنصر اول لیست عوض می‌شود. دفعه بعد، از محل دوم لیست به بعد کوچکترین عنصر لیست پیدا شده، جای آن با عنصر دوم عوض می‌شود و این روند تا مرتب‌شدن کامل لیست ادامه می‌یابد. اگر رشتهٔ bdca در روش انتخابی مرتب گردد، مراحل عمل به صورت زیر خواهد بود:

b	d	c	a	شروع
a	d	c	b	مرحله اول
a	b	c	d	مرحله دوم
a	b	c	d	مرحله سوم

مثال ۲-۱۱

برنامه‌ای که رشته‌ای را از ورودی خوانده، به روش انتخابی آن را مرتب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void select(char *, int);
int main()
{
    char s[80] ;
    clrscr();
    printf(" enter a string :");
    gets(s) ;
    select(s, strlen(s));
    printf("\n the sorted string is: %s", s);
    getch();
    return 0;
```

```
}
//*******************
void select(char *item, int count)
  {
    register int a, b, c ;
    int exchange;
    char t ;
    for(a = 0 ; a < count - 1; a++) {
        exchange = 0;
        c = a ;
        t = item[a] ;
        for(b = a + 1; b < count; ++b)
            if(item[b]<t)  {
                c = b ;
                t = item[b] ;
                exchange = 1;
            } //end of if
        if(exchange)  {
            item[c] = item[a] ;
            item[a] = t;
        }
    }//end of for
}
```

در مثال ۲-۱۱ دو حلقه تکرار وجود دارد که حلقه خارجی n-1 بار (n تعداد عناصر لیست است) و حلقه داخلی n / 2 بار اجرا می‌شود، لذا:

تعداد مقایسه = $1 / 2 (n^2-n)$

تعداد تعویضها در بدترین حالت = $n^2 / 4 + 3 (n-1)$

تعداد تعویضها در بهترین حالت = $3 (n-1)$

در بهترین حالت فقط n-1 عنصر باید تغییر مکان یابند و در هر تغییر مکان نیز سه مقایسه باید انجام گیرد. لذا تعداد تعویضها در بهترین حالت برابر با 3(n-1) است. تعداد تعویضها در حالت متوسط از فرمول پیچیده‌ای محاسبه می‌شود که نتیجه آن عبارت است از:

تعداد تعویضها در حالت متوسط = $x(\log n + y)$

که در فرمول فوق، yمقدار ثابتی برابر با ۰/۵۷۷۲۱۶ است. گرچه تعداد مقایسه‌ها در الگوریتم حبابی و انتخابی با هم برابر ولی الگوریتم انتخابی روش بهتری است.

مرتب‌سازی به روش درجی

در مرتب‌سازی به روش درج، ابتدا دو عنصر اول لیست مرتب می‌شوند و سپس عنصر سوم نسبت به عنصر دوم و

اول در جای مناسبی قرار می‌گیرد (به طور مرتب) و سپس عنصر چهارم نسبت به عناصر اول، دوم و سوم در جای مناسبی قرار می‌گیرد و این روند تا مرتب شدن کامل لیست ادامه پیدا می‌کند. اگر رشتهٔ dcab در این روش مرتب شود، مراحل عمل به صورت زیر خواهد بود.

d	**c**	**a**	**b**	**شروع**
c	**d**	**a**	**b**	**مرحله اول**
a	**c**	**d**	**b**	**مرحله دوم**
a	**b**	**c**	**d**	**مرحله سوم**

مثال ۳-۱۱

برنامه‌ای که رشته‌ای را از ورودی خوانده، به روش درج آن را مرتب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void insert(char *, int);
int main()
{
    char s[80] ;
    clrscr();
    printf(" enter a string :");
    gets(s) ;
    insert(s, strlen(s));
    printf("\n the sorted string is: %s", s);
    getch();
    return 0;
}
//*********************
void insert(char *item, int count)
{
     register int a, b;
     char t;
     for( a = 1; a< count; a++)  {
       t = item[a];
       for(b = a-1; b >=0 && t < item[b]; b --)
           item[b+1] = item[b];
           item[b+1] = t ;
     }//end of for
}
```

برخلاف الگوریتم‌های مرتب‌سازی حبابی و انتخابی، در الگوریتم درج تعداد مقایسه‌ها به وضعیت اولیه لیست بستگی دارد. اگر لیست مرتب باشد، تعداد مقایسه‌ها برابر با n-1 و اگر لیست مرتب نباشد، تعداد مقایسه‌ها برابر با $1/2(n^2+n)-1$ و در حالت متوسط برابر با $1/4(n^2+n-2)$ است و تعداد تعویضها در حالات مختلف عبارتند از:

در بهترین حالت :	2(n-1)
در حالت متوسط :	1 / 4 (n^2 - 9n - 10)
در بدترین حالت :	1 / 2 (n^2 + 3n - 4)

بدترین حالت روش مرتب‌سازی درج، از بدترین حالت مرتب‌سازی حبابی بدتر است ولی حالت متوسط آن کمی بهتر می‌باشد. به هر حال الگوریتم مرتب‌سازی درج دارای دو مزیت است: اولاً یک الگوریتم طبیعی است. یعنی اگر لیست مرتب باشد ساده‌تر عمل می‌کند و اگر لیست نامرتب باشد (مثلاً به طور معکوس مرتب باشد) تعداد مقایسه‌ها و تعویضها بیشتر خواهد بود. این خاصیت باعث می‌شود که الگوریتم درج در مواقعی که لیست نسبتاً مرتب باشد، مفید واقع شود. ثانیاً در مورد مقادیر مساوی، عمل تعویض را انجام نمی‌دهد. یعنی اگر لیستی براساس دو مقدار مرتب باشد، پس از انجام عمل مرتب‌سازی، این ترتیب در هر دو مقدار باقی می‌ماند.

چند روش مرتب‌سازی خوب

در الگوریتم‌هایی که تاکنون مورد بررسی قرار گرفته‌اند، زمان اجرا با مربع تعداد عناصر لیست ارتباط دارد. لذا در لیستهای طولانی، عمل مرتب‌سازی به کندی انجام می‌گیرد. یک راه رفع این مشکل این است که این الگوریتم‌ها به زبان اسمبلی نوشته شوند که در این صورت منحنی رسم شده در شکل ۱-۱۱کمی به سمت راست منحرف خواهد شد. راه دوم رفع مشکل کندی الگوریتم‌های مرتب‌سازی این است که از الگوریتم‌های بهتری استفاده گردد. در این قسمت به بررسی دو روش مرتب‌سازی به نامهای shellsort و quicksort می‌پردازیم.

الگوریتم مرتب‌سازی shellsort

نام الگوریتم مرتب‌سازی shell از نام مخترع آن، SHELL گرفته شده است. در این الگوریتم از روش درج استفاده می‌گردد. اگر رشتهٔ fdacbe تحت این الگوریتم مرتب گردد، مراحل عمل به صورت زیر خواهد بود:

f	d	a	c	b	e	شروع :
c	b	a	f	d	e	مرحله اول :
a	b	c	d	e	f	مرحله دوم :
a	b	c	d	e	f	نتیجه :

در مرحله اول، داده‌های با فاصله ۳ از یکدیگر، مقایسه و مرتب شده، در مرحله دوم داده‌های با فاصله ۲ از یکدیگر، مقایسه و مرتب می‌شوند و در مرحله دوم داده‌ها با فاصله ۱ از یکدیگر مرتب شده‌اند. منظور از فاصله ۳این است که عنصر اول با عنصر چهارم (۳+۱)، عنصر دوم با عنصر پنجم (۵=۳+۲) و عنصر سوم با عنصر ششم (۶=۳+۳) مقایسه شده، در جای مناسب خود قرار می‌گیرند.

برای انتخاب فاصله در اولین مرحله، تعداد عناصر لیست بر ۲ تقسیم می‌شود (n/2) و فاصله‌های بعدی نیز با تقسیم فاصله فعلی بر ۲ حاصل می‌گردند و الگوریتم تا زمانی ادامه پیدا می‌کند که این فاصله به صفر برسد. به عنوان مثال، اگر تعداد عناصر برابر با ۱۰ باشد، فاصله در مرحله اول برابر با ۵، در مرحله دوم برابر با ۲ و در مرحله سوم برابر با ۱ و در مرحله بعدی برابر با صفر خواهد بود.

مثال ۴-۱۱

تابعی که الگوریتم مرتب‌سازی shell را پیاده‌سازی می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void shell(int *, char *, int);
int main()
{
    char s[80];
    int gap[80];
    clrscr();
    printf(" enter a string :");
    gets(s) ;
    shell(gap, s, strlen(s));
    printf("\n the sorted string is: %s", s);
    getch();
    return 0;
}
//********************
void shell(int gap [], char *item , int count)
  {
   register int i, j, step, k, p;
   char x ;
  k = 0;
  gap[0] = count / 2;
  while(gap[k] > 1) {
     k ++;
     gap[k] = gap[k - 1] / 2;
  }//end of while
  for(i = 0; i <= k; i++)  {
    step = gap[i] ;
    for(j = step;  j < count; j++) {
      x = item[j];
      p = j - step;
      while(p >= 0 && x < item[p]) {
        item[p + step] = item[p];
        p = p - step;
      }
      item[p + step] = x;
    }
  }
}
```

همانطور که در مثال ۴-۱۱ مشاهده می‌گردد، حلقه تکرار داخلی، ۲ شرط را بررسی می‌کند که شرط اول x<item[j] است که برای عمل مرتب‌سازی مفید است و شرط دوم 0<jاست که حدود آرایه را تست می‌کند. انجام شرط دوم، از کارآیی الگوریتم shell تا حدودی می‌کاهد.

زمان اجرای روش مرتب‌سازی shell از رابطهٔ $n^{1.2}$ پیروی می‌کند که نسبت به n^2 بهبود خوبی پیدا کرده است. لذا سرعت عمل روش مرتب‌سازی shell از سایر روشهایی که تاکنون مطالعه گردیدند بیشتر است.

الگوریتم مرتب‌سازی quicksort

الگوریتم مرتب‌سازی quicksort از روش تعویضی استفاده می‌کند و به این صورت عمل می‌نماید: ابتدا عنصری (مقداری) را به عنوان ملاک مقایسه انتخاب کرده، لیست را با توجه به این مقدار به دو قسمت تقسیم می‌کند. یک قسمت شامل عناصری است که کوچکتر از مقدار انتخابی می‌باشند و قسمت دیگر شامل عناصری بزرگتر از مقدار انتخابی هستند. در هر یک از این دو قسمت حاصل، همانند قسمت اول عمل می‌گردد و این روند تا مرتب‌شدن کامل لیست ادامه پیدا می‌کند. لذا می‌توان الگوریتم مرتب‌سازی quicksort را به صورت بازگشتی نوشت. انتخاب مقداری جهت ملاک مقایسه به دو طریق ممکن است:

۱. انتخاب به طور تصادفی

۲. مقدار متوسط تعدادی از عناصر موجود در لیست. در یک روش مرتب‌سازی خوب، این مقدار باید طوری انتخاب گردد که میانگین مقادیر موجود در لیست باشد (در بسیاری از لیستها این عمل ممکن نیست).

مثال ۵-۱۱

تابعی که الگوریتم مرتب‌سازی quicksort را پیاده‌سازی می‌کند (در این الگوریتم عنصر وسط لیست به عنوان عنصر ملاک مقایسه انتخاب گردیده است).

```
void quick(char *item,int left,int right) {
    register int i,j ;
    char x,y ;
    i = left ;
    j = right;
    x = item[(left+right)/2];
    do {
       while(item[i] < x && i < right) i++ ;
       while(x < item[j] && j > left)  j -- ;
       if(i < =j) {
          y = item[i] ;
          item[i] = item[j];
          item[j] = y;
          i++ ;
          j -- ;
       }
    } while(i<=j);
    if(left < j) quick(item, left, j) ;
    if(i < right) quick(item, i, right);
}
```

تابعی که در مثال ۵-۱۱ آمده است برای مرتب‌سازی آرایه‌ای از اعداد مورد استفاده قرار می‌گیرد. چنانچه خواسته باشیم آرایه‌ای از رشته‌ها، ساختمانها و یا فایلهای تصادفی را به روش quicksort مرتب کنیم کافی است تغییرات جزئی در این تابع اعمال گردد که در این فصل مثالهایی مطرح خواهد شد. تابع ()quicksort دارای ۳ پارامتر است. پارامتر اول نام آرایه و یا فایلی است که باید مرتب گردد، پارامتر دوم و سوم طول آرایه یا فایل را مشخص می‌کنند. به عنوان مثال، در مورد مرتب‌کردن آرایه، دومین پارامتر برابر با صفر و سومین پارامتر برابر با n-1 (n طول آرایه است) می‌باشد.

در روش مرتب‌سازی quicksort تعداد متوسط مقایسه‌ها برابر با n log n و تعداد تعویضها برابر با n/6)log) است. این مقادیر در مقایسه با سایر الگوریتم‌هایی که تاکنون بررسی شده‌اند بسیار کمتر است. لذا الگوریتم quicksort از کلیه الگوریتم‌های مرتب‌سازی که تاکنون بررسی شده‌اند، بهتر است. برای مقایسه این الگوریتم با الگوریتم مرتب‌سازی حبابی، اگر طول آرایه برابر با ۱۰۰ فرض شود، در روش مرتب‌سازی quicksort به ۲۰۰ مقایسه و در روش مرتب‌سازی حبابی به ۹۹۰ مقایسه نیاز است.

مرتب‌سازی رشته‌ها

ساده‌ترین راه برای مرتب‌سازی رشته‌ها این است که آرایه‌ای از اشاره‌گرها به رشته‌ها داشته باشیم. این عمل موجب سرعت دسترسی به رشته‌ها می‌گردد.

مثال ۶-۱۱

تابعی که آرایه‌ای رشته‌ای را با استفاده از الگوریتم quicksort مرتب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void quick(char *, int, int);
int main()
{
    char s[80];
    clrscr();
    printf(" enter a string :");
    gets(s) ;
    quick(s, 0, strlen(s) - 1);
    printf("\n the sorted string is: %s", s);
    getch();
    return 0;
}
//**************
void quick(char item[], int left, int right)
{
   register int i, j;
   char x, y ;
```

```
    i = left;
    j = right;
    x = item[(left + right) / 2];
    do {
       while(item[i] < x && i < right) i++ ;
       while(x < item[j] && j > left)  j -- ;
       if(i <= j) {
          y = item[i] ;
          item[i] = item[j];
          item[j] = y;
          i++ ;
          j -- ;
       }//end of if
    } while(i <= j);
    if(left < j)  quick(item, left, j) ;
    if(i < right) quick(item, i, right);
}
```

مرتب‌سازی ساختمانها

در بسیاری از برنامه‌های کاربردی لازم است که به جای مرتب‌سازی لیستی که عناصر آن از انواع ساده (مثل int و float) هستند، خواسته باشیم لیستهایی را که هر یک از عناصر آن از نوع ساختمان هستند مرتب کنیم. به عنوان مثال، مرتب‌کردن آرایه‌ای که هر عنصر آن شامل کلیه مشخصات یک دانشجو است نمونه‌ای از مرتب‌سازی ساختمانها است.

مثال ۷-۱۱

برنامه‌ای که مشخصات تعداد n نفر از دانشجویان کلاسی را از ورودی خوانده در آرایه‌ای از ساختمانها قرار می‌دهد و سپس این آرایه را براساس نام دانشجویان مرتب می‌کند. اطلاعاتی که از ورودی خوانده می‌شوند عبارتنداز: نام دانشجو، شماره دانشجویی، معدل دانشجو و تعداد واحدی که دانشجو گذرانده است. در این برنامه با استفاده از امکان تخصیص حافظه پویا، حافظه لازم برای ذخیره کردن n دانشجو از سیستم اخذ می‌گردد.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
void quick_struct(struct student [], int, int);
struct student {
      char name[10] ;
      int stno ;
      int unit ;
      float ave ;
```

```
} ;
void main()
{
    int n, i;
    float x ;
    struct student *p ;
    clrscr() ;
    printf("\n");
    printf("enter number of student:");
    scanf("%d",&n);
    p = (struct student *) malloc(n * sizeof(struct student));
    if(!p)  {
      printf("\n allocation failure.");
      exit(1);
    }
    for(i = 0 ; i < n ; i++)   {
       clrscr() ;
       printf("\n enter name of student");
       printf(" %d:", i+1);
       scanf("%s", p[i].name);
       printf("\n enter student number:");
       scanf("%d", &p[i].stno);
       printf("\n enter number of unit:");
       scanf("%d", &p[i].unit) ;
       printf("\n enter average :");
       scanf("%f", &x);
       p[i].ave = x ;
    }
    quick_struct(p, 0, n-1) ;
    clrscr() ;
    gotoxy(15, 2) ;
    printf( "<<SORTED DATA DEPEND ON NAME>>");
    gotoxy(10, 3) ;
     printf("**************************");
     printf("**************************");
    gotoxy(10, 4);
    printf(" NAME        STUDENT#       UNIT      AVERAGE");
    gotoxy(10,5);
     printf("**************************");
     printf("**************************");
    for(i = 0 ; i < n; i++)  {
        gotoxy(10, i + 6) ;
        printf("%s", p[i].name) ;
```

```
            gotoxy(23, i + 6);
            printf("%d", p[i].stno);
            gotoxy(39, i + 6) ;
            printf("%d", p[i].unit);
            gotoxy(50, i + 6) ;
            printf("%.2f", p[i].ave);
        }
        gotoxy(10, i + 6) ;
         printf("**************************");
         printf("**************************");
        gotoxy(23, i + 7);
        printf("press a key to continue.");
        getch();
}
 //*****************************
void quick_struct(struct student item[], int left,int right)
{
    register int i,j ;
    char *x;
    struct student temp ;
    i= left ;
    j= right;
    x = item[(left + right) / 2].name;
    do {
        while(strcmp(item[i].name, x) < 0 && i < right) i++ ;
        while(strcmp(item[j].name, x) > 0 && j > left)  j -- ;
       if(i <= j) {
            temp = item[i] ;
            item[i] = item[j];
            item[j] = temp;
            i++ ;
            j -- ;
       } //end of if
    } while( i <= j);
    if(left < j)   quick_struct(item, left, j) ;
    if(i<right)    quick_struct(item, i, right);
}
```

عملکرد برنامه

```
enter number of student :2
enter name of student 1:ali
enter student number:12345
enter number of unit:134
enter average:15.5
enter name of student 2:abbas
enter student number:23124
enter number of unit:120
enter average:17.5

    <<SORTED DATA DEPEND ON NAME>>
 ******************************************************
NAME     STUDENT#      UNIT     AVERAGE
 ******************************************************
abbas      23124         120      17.00
ali        12345         134      15.50
 ******************************************************
    press a key to continue.
```

مرتب سازی فایلهای تصادفی

همانطور که قبلاً گفته شد دو نوع سازمان فایل عبارتنداز: فایلهای ترتیبی و تصادفی. اگر تعداد رکوردهای فایل کم باشد، برای مرتب‌کردن اطلاعات موجود در فایل، آنها را از روی دیسک خوانده، به آرایه منتقل می‌کنیم و با استفاده از الگوریتم‌های مرتب‌سازی که برای مرتب‌کردن آرایه‌ها به کار می‌روند، آنها را مرتب می‌کنیم. اما در بسیاری از موارد، بخصوص در کاربردهای بانکهای اطلاعاتی، از فایلهای تصادفی برای ذخیره‌کردن اطلاعات استفاده می‌شود که نسبت به فایلهای ترتیبی دارای مزایای زیر است:

۱. انجام تغییرات در فایلهای تصادفی ساده‌تر از فایلهای ترتیبی است؛ زیرا برای انجام تغییرات در فایل ترتیبی، باید فایل مجدداً بازنویسی گردد اما در مورد فایلهای تصادفی این طور نیست.

۲. فایلهای تصادفی را می‌توان همانند آرایه‌ای بزرگ بر روی دیسک در نظر گرفت و به آسانی آنها را مرتب نمود. لذا می‌توان با کمی تغییر در الگوریتم quicksort آن را برای مرتب‌سازی فایلهای تصادفی به کار برد. به این صورت که در الگوریتم quicksort به جای استفاده از اندیس آرایه، باید از تابع ()fseek برای انتخاب رکورد موردنظر استفاده نمود.

مثال ۸-۱۱

برنامه‌ای که با استفاده از الگوریتم quicksort یک فایل تصادفی را مرتب می‌کند. این برنامه یک فایل تصادفی از اطلاعات دانشجویان تشکیل می‌دهد، این فایل را مرتب کرده در صفحه‌نمایش چاپ می‌کند.

شرح وظایف برنامه‌ها

main(): باز کردن فایل، تعیین طول فایل به بایت و فراخوانی توابع. در این برنامه از تابع filelength() برای تعیین طول فایل و از تابع fileno() برای تعین شماره (handle) فایل استفاده شده است.

quick_file(): فایل تصادفی را به روش quicksort مرتب می‌کند.

sawp_all_fields(): رکوردهایی راکه باید جابجا شوند، پیدا کرده، آنها را جابجا می‌کند.

get_name(): نام دانشجو را در فایل پیدا کرده، برمی‌گرداند.

menu_select(): منویی با چهار گزینه ظاهر می‌کند و انتخاب کاربر را به برنامه برمی‌گرداند.

enter(): رکوردهای دانشجویان را از ورودی خوانده، در فایل تصادفی قرار می‌دهد.

report(): از فایل تصادفی گزارش صفحه‌به‌صفحه تهیه می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
struct student {
        char name[30] ;
        int stno ;
        int unit ;
        float ave ;
} ainfo ;
int menu_select(void);
void enter();
void report();
void quick_file(FILE *pf,int left, int right);
void swap_all_fields(FILE *fp, long i , long j);
char *get_name(FILE *fp, long rec) ;
void main()
{
   FILE *fp ;
   long int x;
   int handle, y;
   if((fp = fopen("mlist", "a+b")) == NULL) {
       printf("\cannot open file .");
       getch();
       exit(1) ;
   }
   for(;;) {
```

```
        clrscr();
        switch(menu_select()) {
          case 1: enter(); break;
          case 2:
            if((fp = fopen("mlist", "r+b")) == NULL) {
                printf("\cannot open file .");
                getch();
               exit(1) ;
            }
            handle = fileno(fp);
            x = filelength(handle);
            y = x / (sizeof(struct student));
            quick_file(fp, 0, y - 1);
            fclose(fp);
            break;
          case 3: report(); break;
          case 4: exit(1);
      } //end of switch
     } //end of for
}
//****************
void quick_file(FILE *fp, int left, int right)
{
    long int i, j;
    char x[10];
    i = left;
    j = right;
    strcpy(x, get_name(fp, (long)(i + j) / 2));
    do {
        while(strcmp(get_name(fp, i), x) < 0 && i< right) i ++ ;
        while(strcmp(get_name(fp, j), x) > 0 && i < right) j -- ;
        if(i <= j) {
            swap_all_fields(fp, i, j) ;
            i ++ ;
            j -- ;
        }
    } while(i <= j) ;
    if(left < j) quick_file(fp, left, (int) j) ;
    if(i < right)  quick_file(fp,(int) i,right);
}
//*********************
void swap_all_fields(FILE *fp, long i, long j)
{
```

```
    char a[sizeof(ainfo)], b[sizeof(ainfo)];
    fseek(fp, sizeof(ainfo) * i, 0) ;
    fread(a,  sizeof(ainfo),  1, fp) ;
    fseek(fp, sizeof(ainfo) * j, 0) ;
    fread(b,  sizeof(ainfo),  1, fp) ;
    fseek(fp, sizeof(ainfo) * j, 0) ;
    fwrite(a, sizeof(ainfo),  1, fp) ;
    fseek(fp, sizeof(ainfo) * i, 0) ;
    fwrite(b, sizeof(ainfo),  1, fp) ;
}
 //*****************
char *get_name(FILE *fp, long rec)
{
    struct student *p ;
    p = &ainfo ;
    fseek(fp, rec * sizeof(ainfo), 0);
    fread(p, sizeof(ainfo), 1, fp) ;
    return(ainfo.name) ;
}
 //**********************
int menu_select(void)
{
     char s[10] ;
     int c ;
     clrscr() ;
     gotoxy(29, 6);
     printf("1) << enter a recorde >>");
     gotoxy(29, 8);
     printf("2) << sort the file >>");
     gotoxy(29, 10);
     printf("3) << list  the file >>");
     gotoxy(29, 12);
     printf("4) <<       Quit     >>");
     do {
        gotoxy(24, 15) ;
        printf(" Please enter your choice(1-4):");
        gets(s) ;
        c = atoi(s) ;
     } while(c < 0 || c > 4) ;
     return (c);
}
 //****************************
void enter()
```

```
{
  int slot ;
  char numstr[10] ;
    FILE *fp;
  if((fp = fopen("mlist", "a+b")) == NULL) {
      printf("\cannot open file .");
      getch();
      exit(1) ;
  }
  gotoxy(5, 17) ;
  printf("enter name:") ;
  gets(ainfo.name);
  gotoxy(40, 17) ;
  printf("enter stno:") ;
  gets(numstr);
  ainfo.stno = atoi(numstr);
  gotoxy(5, 19) ;
  printf("enter unit:") ;
  gets(numstr);
  ainfo.unit = atoi(numstr);
  gotoxy(40, 19) ;
  printf("enter average:") ;
  gets(numstr);
  ainfo.ave = atof(numstr);
  fwrite(&ainfo, sizeof(struct student), 1, fp);
  fclose(fp);
}
//***********
void report()
{
    register  int t ;
    int r = 0 ;
    FILE *fp;
    if((fp = fopen("mlist", "r+b")) == NULL) {
      printf("\cannot open file .");
      getch();
      exit(1) ;
    }
    clrscr() ;
    gotoxy(20, 2) ;
    printf(" << all information in list are:>>") ;
    gotoxy(13, 3) ;
     printf("**********************");
```

```
        printf("********************") ;
        gotoxy(10, 4);
        printf("  name      stno      unit      average");
        gotoxy(10, 5);
        printf("  _______   _______   _____     _____");
        fread(&ainfo, sizeof(struct student), 1, fp);
        while(!feof(fp)) {
              gotoxy(14, 6 + r) ;
              printf("%s ", ainfo.name);
              gotoxy(26, 6 + r) ;
              printf("%d ", ainfo.stno);
              gotoxy(40, 6 + r) ;
              printf("%d ", ainfo.unit);
              gotoxy(52, 6 + r) ;
              printf("%.2f ", ainfo.ave);
              r++ ;
              fread(&ainfo, sizeof(struct student), 1, fp);
        }
        gotoxy(13, 6 + r) ;
        printf("********************");
        printf("**********************");
        gotoxy(27, 7 + r) ;
        printf("press any key to continue");
        fclose(fp);
        getch() ;
}
```

روشهای جستجو

در بسیاری از موارد ممکن است خواسته باشیم در یک لیست یا فایل، اطلاعات و یا رکوردهای خاصی را جستجو کنیم. همانند عمل مرتب‌سازی، عمل جستجو نیز ممکن است به روشهای گوناگونی انجام شود. اگر در یک لیست نامرتب بخواهیم اطلاعاتی را پیدا کنیم، تنها راه حل این است که عناصر لیست را از اولین عنصر به آخرین عنصر (و یا برعکس) با اطلاعات مورد جستجو مقایسه کنیم و هرگاه عنصری از لیست با عنصر موردنظر ما برابر باشد و یا به انتهای لیست رسیده باشیم، عمل جستجو پایان می‌یابد.

مثال ۹-۱۱

تابعی که الگوریتم جستجوی ترتیبی را برای یک آرایه رشته‌ای پیاده‌سازی می‌کند.

```
int seq_search(char *item, int count, int key)
{
    register int i ;
    for(i = 0; i < count; i++)
      if(key == item[i])
        return i ;
    return -1 ;
}
```

تابع seq_search() یک آرایه، تعداد عناصر آن و اطلاعات مورد جستجو (key) را به عنوان پارامتر پذیرفته، چنانچه اطلاعات موردنظر در آرایه پیدا شود محل وجود آن در آرایه، وگرنه مقدار ۱-را به برنامه فراخواننده برمی‌گرداند.

الگوریتم جستجوی ترتیبی در بدترین حالت (موقعی که اطلاعات موردجستجو در انتهای لیست باشد) n مقایسه (n تعداد عناصر لیست است) و در بهترین حالت (موقعی که اطلاعات مورد جستجو اولین عنصر لیست باشد) یک مقایسه و در حالت متوسط (موقعی که اطلاعات مورد جستجو در وسط لیست باشد) n/2 مقایسه انجام می‌دهد.

در لیستهای مرتب، علاوه بر روش جستجوی ترتیبی می‌توان از روش جستجوی باینری استفاده نمود. در این روش، ابتدا عنصر وسط لیست پیدا می‌شود و لیست به دو نیمه تقسیم می‌گردد (نیمه‌ای که کمتر از عنصر وسط هستند و نیمه‌ای که بزرگتر از عنصر وسط می‌باشند). اگر اطلاعات مورد جستجو برابر با عنصر وسط باشد عمل جستجو خاتمه می‌یابد؛ اگر کوچکتر از عنصر وسط باشد، نیمه پایینی وگرنه نیمه بالایی لیست همانند لیست کامل مورد جستجو قرار خواهد گرفت. در یکی از این دو نیمه (برحسب مورد) عنصر وسط به دست می‌آید و با این عنصر مانند عنصر وسط قبلی رفتار خواهد شد. این روند تا پیداکردن اطلاعات مورد نظر و یا بررسی همهٔ عناصر لیست ادامه پیدا می‌کند.

مثال ۱۰-۱۱

تابعی که الگوریتم جستجوی باینری را برای یک لیست رشته‌ای پیاده‌سازی می‌کند.

```
int binary_search(char *item ,int count, char key) {
   int low, high, mid ;
   low = 0 ;
   high = count - 1 ;
   while(low <= count)  {
      mid = (low + high)/2 ;
      if(key < item[mid])
          high = mid-1 ;
      else if(key > item[mid])
          low = mid + 1 ;
      else
          return mid ;
   }
   return -1 ;
}
```

تابع binary_search() یک آرایه، طول آرایه و اطلاعاتی را که باید مورد جستجو قرار گیرد به عنوان پارامتر می‌پذیرد. چنانچه اطلاعات مورد جستجو در لیست وجود داشته باشد محل وجود آن در لیست، وگرنه عدد ۱- به تابع فراخواننده برمی‌گردد.

تمرینات

۱. روشهای مرتب‌سازی را نام برده، مختصراً توضیح دهید.

۲. مرتب‌سازی حبابی را با مرتب‌سازیهای درجی و انتخابی مقایسه کنید.

۳. برنامه‌ای بنویسید که یک فایل از دانشجویان کلاس تشکیل دهد و سپس این فایل را مرتب کند. مرتب‌سازی باید برحسب شماره دانشجویی صورت گیرد. سپس شماره دانشجویی را از ورودی خوانده، از طریق جستجوی دودویی، در فایل جستجو کند.

۴. برنامه‌ای بنویسید که معدل تعدادی از دانشجویان را از ورودی خوانده، در آرایه‌ای قرار دهد و سپس به هر یک از روشهای مرتب‌سازی که در این فصل بیان شد، مرتب کرده، نتیجه را به خروجی ببرد.

۱۲

ساختمان کامپیوتر و وقفه‌ها

بحث عمدهٔ این فصل، به وقفه‌ها مربوط می‌شود. چون درک صحیح وقفه‌ها و کاربرد آنها مستلزم آشنایی با ساختمان کامپیوتر است، ابتدا بحث مختصری در این خصوص مطرح می‌شود.

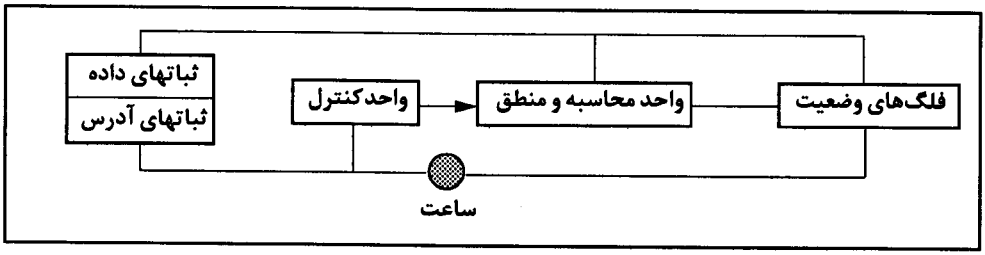
ساختمان کامپیوتر

از آنجایی که اغلب کامپیوترهای مورد استفاده، ریزکامپیوترهای همساز با آی.بی.اِم (IBM) هستند، ساختمان آنها را برای رسیدن به اهداف خود، بررسی می‌کنیم. پردازنده‌های (CPU) موجود در این نوع کامپیوترها، از شرکت اینتل (intel) است. شرکت اینتل دو دسته از ریزپردازنده‌ها را ارائه کرده است: ۱۶ بیتی و ۳۲ بیتی. پردازنده‌های ۸۰۸۶، ۸۰۸۸، ۸۰۲۸۶ ساختار ۱۶ بیتی و پردازنده‌های ۸۰۳۸۶، ۸۰۴۸۶ و پنتیوم ساختار ۳۲ بیتی دارند.

وظیفه پردازنده را می‌توان در سه عمل خلاصه کرد:

- مَکِش دستور بعدی (قراردادن آن در صف، بازسازی شمارنده برنامه).
- رمزگشایی دستور (ترجمهٔ آدرس، مَکِش عملوندها از حافظه).
- اجرای دستور (انجام محاسبات موردنیاز، ذخیره در حافظه و ثبات، تغییر وضعیت فلگ‌ها (flags).

شکل ۱-۱۲ نمونه ساده‌ای از یک پردازنده را نشان می‌دهد. پردازنده به دو بخش تقسیم شده است: **واحد محاسبه و منطق** (ALU) و واحد کنترل (CU). واحد ALU عملیات محاسباتی، منطقی و شیفت را انجام می‌دهد و واحد CU دستورات و داده‌ها را دریافت کرده، آدرس را برای ALU رمزگشایی می‌کند. ساعت درونی پردازنده، هر یک از عملیات پردازنده و حافظه را همزمان می‌کند.



شکل ۱-۱۲ طراحی ساده‌ای از پردازنده.

پردازنده، از طریق پینهایی (pins) متصل به سوکت پردازنده در مادر بورد (motherboard) ، به بخشهای دیگر کامپیوتر اتصال پیدا می‌کند. این پینها، گذرگاه داده‌ای (data bus) ، گذرگاه آدرس، و پینهای ولتاژ و کنترلی گوناگونی را برای پردازنده ایجاد می‌کنند.

گذرگاههای داده و آدرس. گذرگاه داده‌ای داخلی، مجموعه‌ای از سیمهای موازی است که داده‌ها را بین بخشهای مختلف پردازنده انتقال می‌دهند. وقتی داده‌ها از حافظه خارجی خوانده می‌شوند، واحد کنترل آدرس آن را محاسبه کرده، آن آدرس را در گذرگاه آدرس قرار می‌دهد. واحد حافظه (پردازنده دیگری در مادر بورد)، گذرگاه آدرس را می‌خواند، داده‌های درخواستی را در گذرگاه داده قرار می‌دهد و سیگنالی به پردازنده مرکزی می‌فرستد و اعلام می‌دارد که داده‌ها آماده است. پردازنده مرکزی، داده‌ها را از گذرگاه دادهٔ خارجی به ناحیه‌های حافظه داخلی خود منتقل می‌کند.

ثباتها. در داخل پردازنده مرکزی، حافظه‌های سریعی به نام ثباتها وجود دارند که مستقیماً به واحد کنترل و واحد محاسبه و منطق متصل هستند. چون دستیابی به ثباتها سریعتر از دستیابی به حافظه است، دستوراتی که فقط از ثباتها استفاده می‌کنند، بسیار سریعتر از دستوراتی که عملوندهای آنها در حافظه‌اند اجرا می‌شوند.

ساعت (clock) . هر عملی که در پردازنده مرکزی اتفاق می‌افتد، باید توسط یک ساعت داخلی همزمان شود. اساسی‌ترین واحد ساعتِ مربوط به دستورات ماشین، **سیکل ماشین** یا **سیکل ساعت** نام دارد و برحسب میلیون سیکل در ثانیه (MHZ) سنجیده می‌شود. ساعت موجود در اغلب پردازنده‌های پنتیوم، در ۲۰۰ تا ۴۰۰MHZ اجرا می‌شود (میلیونها بار در ثانیه).

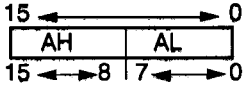
ثباتهای پردازنده‌های ۱۶ بیتی

ثباتها حافظه‌های ۸، ۱۶ یا ۳۲ بیتی در داخل پردازنده مرکزی هستند. پردازنده مرکزی دارای گذرگاه دادهٔ داخلی است که پهنای آن دو برابر گذرگاه دادهٔ خارجی آن است. به عنوان مثال، برای افزایش سرعت حلقه‌های تکرار، محاسبات و تصمیم‌گیری در داخل حلقه را در ثباتها انجام می‌دهیم. شکل ۲-۱۲ ثباتهای ۱۶ بیتی را نشان می‌دهد.

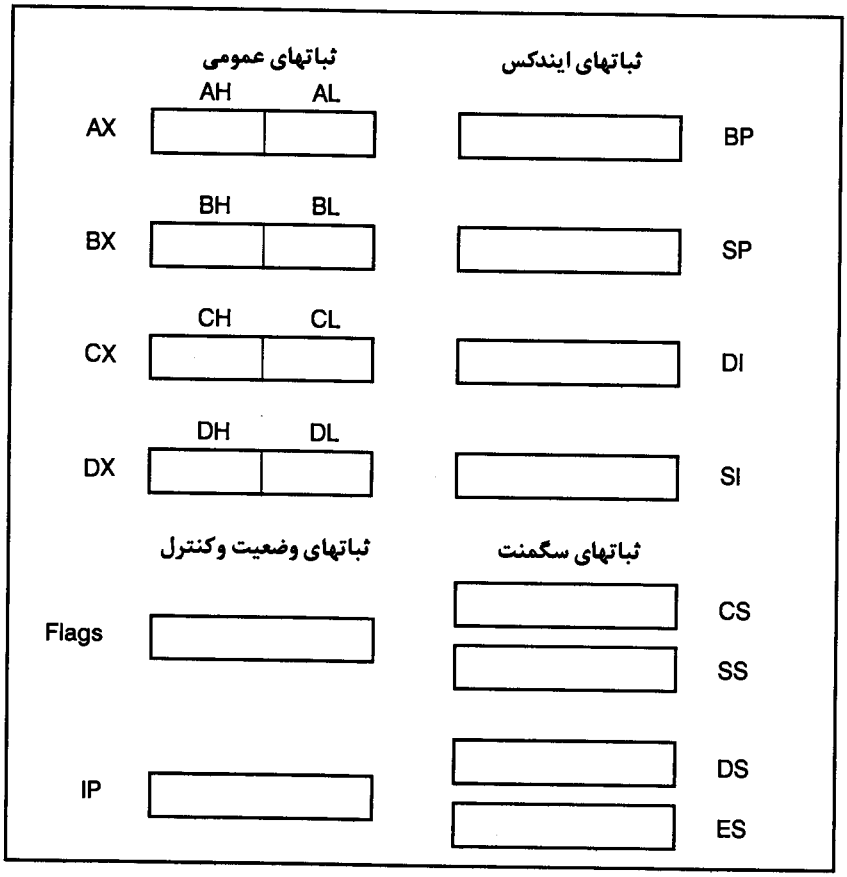
همانطور که در شکل ۲-۱۲ مشاهده می‌شود، ثباتها به چهار دسته تقسیم می‌شوند: ثباتهای عمومی، ثباتهای سگمنت، ثبات ایندکس و ثباتهای وضعیت و کنترل. ممکن است در کتابهای مختلف اسامی دیگری برای اینها انتخاب شود.

ثباتهای عمومی

ثباتهای عمومی را ثباتهای داده نیز می‌گویند و عبارت‌انداز: AX ، BX ، CX و DX . هر ثبات عمومی به صورت یک ثبات ۱۶ بیتی و یا دو ثبات ۸ بیتی قابل دستیابی است (شکل ۲-۱۲). به عنوان مثال، ثبات AX به صورت AL و AH نیز قابل دستیابی است. یعنی AL و AH هر کدام یک ثبات ۸ بیتی‌اند:



ثبات AX. این ثبات توسط پردازنده مرکزی برای اعمال محاسباتی مورد استفاده قرار می‌گیرد. سایر اعمال نیز ممکن است با این ثبات انجام شوند. این ثبات را انباشتگر (accumulator) نیز می‌گویند.



شکل ۲-۱۲ ثباتهای ۱۶ بیتی.

ثبات BX. این ثبات می‌تواند آدرس زیربرنامه یا متغیر را نگهداری کند. سه ثبات دیگر که این ویژگی را دارند عبارت‌انداز: SI، DI و BP. این ثبات برای اعمال محاسباتی و انتقال داده‌ها نیز به کار می‌رود. ثبات BX به ثبات **پایه** نیز معروف است.

ثبات CX. این ثبات به عنوان شمارنده‌ای برای حلقه‌های تکرار مورد استفاده قرار می‌گیرد. این دستورات، به طور خودکار اجرا شده، از CX یک واحد کم می‌کنند. ثبات CX به ثبات **شمارنده** نیز معروف است.

ثبات DX. این ثبات در اعمال ضرب و تقسیم، نقش خاصی را بازی می‌کند. نام دیگر این ثبات **ثبات داده** است.

سگمنت‌ها

سگمنت (segment)، بخشی از حافظه است که اندازه آن 64K است. در هر برنامه حداکثر چهار سگمنت قابل استفاده است که عبارت‌انداز: سگمنت کُد (code segment)، سگمنت داده (data segment)، سگمنت پشته (stack segment) و سگمنت اضافی (extra segment). سگمنت کُد برای ذخیره دستورالعملهای برنامه، سگمنت داده برای ذخیره داده‌های برنامه، سگمنت پشته برای ایجاد پشته و سگمنت اضافی برای بعضی از اعمال رشته‌ای مورد استفاده قرار می‌گیرد.

ثباتهای سگمنت

تعداد چهار ثبات سگمنت وجود دارد که هر کدام از آنها، آدرس سگمنتی را نگهداری می‌کند.
ثبات سگمنت کد (CS) : آدرس سگمنت کد (دستورالعملهای اجرایی برنامه) را نگهداری می‌کند.
ثبات سگمنت داده (DS) : این ثبات، ثبات پایه فرضی متغیرهاست. پردازنده با استفاده از آدرس موجود در DS (آدرس سگمنت داده)، آدرس متغیرها را پیدا می‌کند.
ثبات سگمنت پشته (SS): آدرس سگمنت پشته در این ثبات قرار دارد.
ثبات سگمنت اضافی (ES) : این ثبات، ثبات پایه دیگری برای متغیرهای حافظه است (آدرس سگمنت اضافی را نگهداری می‌کند).

ثباتهای ایندکس

ثباتهای ایندکس حاوی آدرس آفست (offset) داده‌ها و دستورات هستند. منظور از آفست، فاصله متغیر، برچسب یا دستور از ابتدای سگمنت است. ثباتهای ایندکس، پردازش رشته‌ها، آرایه‌ها و سایر ساختمان داده‌هایی را که حاوی چندین عناصر هستند، تسریع می‌کند. ثباتهای ایندکس عبارتنداز: BP، SP، SI و DI.
ثبات BP (Base Pointer) : این ثبات حاوی یک آفست فرضی از ثبات SS است. ثبات BP با استفاده از زیربرنامه‌ای، متغیرهایی را که در اثر فراخوانی برنامه در پشته قرار گرفته‌اند، پیدا می‌کند.
ثبات SP (Stack Pointer) : این ثبات حاوی آفست بالای پشته است. ترکیب ثبات SP و SS موجب دستیابی به بالای پشته می‌شود.
ثبات SI (Source Index) : این ثبات برای دستورالعملهای رشته‌ای مورد استفاده قرار می‌گیرد. آدرس رشته‌های منبع (در دستورات رشته‌ای)، در این ثبات قرار می‌گیرد.
ثبات DI (Destination Index) : این ثبات در دستورات رشته‌ای، آدرس رشته مقصد را نگهداری می‌کند.

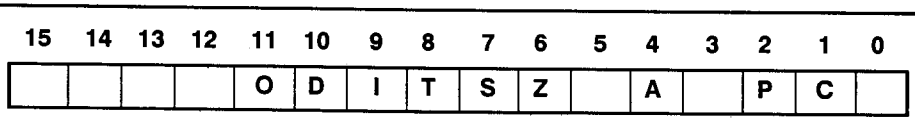
ثباتهای وضعیت وکنترلی

ثبات IP(Instruction Pointer) : این ثبات همواره حاوی آفست دستور اجرایی بعدی در سگمنت کد است. ثباتهای IP و CS برای تعیین آدرس دستور بعدی به کار می‌روند.
ثبات فِلگ (Flag) : ثبات مخصوصی است که بیتهای آن، وضعیت پردازنده مرکزی یا نتیجه عملیات محاسباتی را نشان می‌دهند. هر بیت دارای نامی است؛ تعدادی از بیتها نیز بلااستفاده‌اند. شکل ۳-۱۲ ثبات فِلگ ۸۰۸۶/۸۰۸۸ را نشان می‌دهد.

فلگ‌های کنترلی

برنامه‌نویس می‌تواند بعضی از بیتهای ثبات فلگ را مقداردهی کند تا عمل پردازنده را تحت کنترل درآورد. این بیتها عبارت‌انداز: DF، IF و TF.

- **فلگ DF** : این فلگ دستورات انتقال داده‌ها مثل mov ، cmps (دستورات اسمبلی) را تحت تأثیر قرار می‌دهد. اگر این بیت برابر با یک باشد، عمل مقایسه یا انتقال از راست به چپ و اگر صفر باشد، از چپ به راست انجام می‌شود.



O = Overflow (سرریز)
D = Direction (جهت)
I = Interrupt (وقفه)
T = Trap (ردیابی)
S = Sign (علامت)
Z = Zero (صفر)
A = Auxiliary carry (نقل کمکی)
P = Parity (توازن)
C = Carry (رقم نقلی)

توجه: برای مراجعه به هر یک از این فلگ‌ها، حرف F به معنای فلگ، در آخر آنها قرار می‌گیرد: OF، DF IF، TF، SF، ZF، AF، PF و CF.

شکل ۳-۱۲ بیتهای ثبات فلگ.

- **فلگ IF**: مشخص می‌کند که وقفه سیستم اتفاق بیفتد یا خیر (با وقفه‌ها در ادامه آشنا می‌شوید). برنامه‌نویس می‌تواند وقفه‌ها را غیرفعال کند تا چنانچه کارهای مهمی در حال انجام است، ادامه یابد. اگر برابر با یک باشد، سیستم وقفه فعال است وگرنه غیرفعال خواهد بود.
- **فلگ TF**: مشخص می‌کند که آیا پردازنده پس از اجرای هر دستور متوقف شود یا خیر. اگر این فلگ برابر با یک باشد، برنامه اشکالزدای (debug) می‌تواند برنامه را دستور به دستور اجرا کند.

فلگ‌های وضعیت

فلگ‌های وضعیت، نتایج عملیات منطقی و محاسباتی را که توسط پردازنده انجام شده‌اند نشان می‌دهند. این فلگ‌ها عبارتند از: OF، SF، ZF، AF، PF و CF.

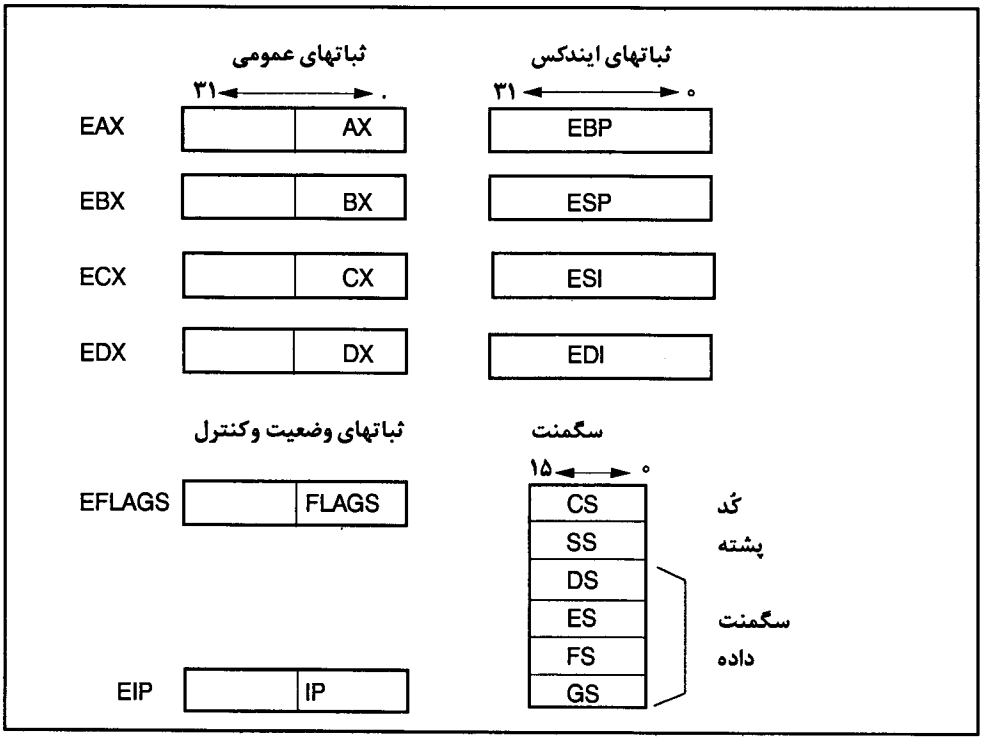
فلگ CF: وقتی نتیجه یک عمل محاسباتی بدون علامت آنقدر بزرگ باشد که در مقصد جا نشود، این فلگ برابر با یک می‌شود. به عنوان مثال، اگر مجموع دو عدد ۲۰۰ و ۸۵ در یک ثبات ۸بیتی مثل AL قرار گیرد، سرریز اتفاق افتاده، فلگ CF برابر با یک می‌شود. اگر این فلگ یک باشد، رقم نقلی وجود دارد و اگر صفر باشد، رقم نقلی وجود ندارد.

فلگ OF: وقتی نتیجهٔ یک عمل محاسباتیِ علامت‌دار طوری باشد که در مقصد جا نشود، این بیت برابر با یک می‌شود. به عنوان مثال، اگر مجموع دو مقدار ۱۲۸- و ۲- در یک ثبات ۸بیتی مثل BL قرار گیرد، این بیت برابر با یک می‌شود.

فلگ SF: اگر نتیجهٔ یک عبارت محاسباتی یا منطقی، یک مقدار منفی باشد، این بیت برابر با یک می‌شود. چون بالاترین بیت عدد منفی برابر با یک است، فلگ علامت، همواره برابر با بیت علامتِ مقدار مقصد است.

فلگ ZF: اگر نتیجه عمل محاسباتی یا منطقی برابر با صفر باشد، این فلگ برابر با یک قرار می‌گیرد. این فلگ معمولاً برای دستورات حلقه به کار می‌رود. به طوری که براساس مقایسه دو مقدار، می‌توان از نقطه‌ای به نقطه دیگر پرش (jump) کرد.

فلگ AF: اگر انجام عملی موجب شود تا یک رقم نقلی از بیت ۳ به بیت ۴ بوجود آید (یا یک رقم قرضی از بیت ۴ به بیت ۳ داده شود)، این فلگ برابر با یک می‌شود. با وجود فلگ توازن این فلگ به‌ندرت مورد استفاده قرار می‌گیرد.



شکل ۴-۱۲

فلگ PF: این فلگ تعداد بیتهایی را که در یک عمل برابر با یک هستند مشخص می‌کند. اگر تعداد بیتهای یک، زوج باشد، توازن زوج است و اگر تعداد بیتهای یک، فرد باشد، توازن فرد است. سیستم عامل با استفاده از این فلگ، صحت داده‌ها را در انتقال بین دستگاههای مختلف بررسی می‌کند.

ثباتهای ۳۲ بیتی

تمام پردازنده‌های اینتل از ۸۰۳۸۶ به بعد، ثباتهای ۳۲ بیتی دارند (شکل ۴-۱۲). با استفاده از ثباتهای ۳۲ بیتی، به فضای آدرس زیادی می‌توان دست یافت. در این پردازنده‌ها، ثباتهای سگمنت، ۱۶ بیتی‌اند، اما توجه داشته باشید که دو ثبات جدید به نام FS و GS به این پردازنده‌ها اضافه شدند. همانطور که در شکل ۴-۱۲ می‌بینید، نیمهٔ بالایی ثباتهای عمومی ۳۲ بیتی، نامگذاری نشده‌اند. برای نوشتن مقدار ۱۶ بیتی در نیمه بالایی این ثباتها، ابتدا باید آن را در نیمه پایینی قرار دهید، سپس آن را به سمت چپ شیفت دهید.

مفهوم آدرس‌دهی

هر بایت از حافظه اصلی کامپیوتر دارای یک آدرس است؛ اولین بایت دارای آدرس صفر، دومین بایت دارای آدرس ۱ و nامین بایت دارای آدرس n-1 است. حداکثر فضایی از حافظه که با استفاده از یک ثبات ۱۶ بیتی قابل آدرس‌دهی است برابر ۶۵۵۳۶=2^{16} بایت است که از صفر تا ۶۵۵۳۵ شماره‌گذاری می‌شود (شماره هر بایت آدرس آن بایت را

مشخص می‌کند). این مقدار فضای حافظه، بسیار ناچیز بوده قابل توجه نیست، در ریزپردازنده‌های خانواده ۸۰۸۶ تکنیکی به نام "آدرس سگمنت و تفاوت مکان" به کار گرفته می‌شود تا به جای خط آدرس ۱۶ بیتی از خط آدرس ۲۰ بیتی استفاده گردد (بعضی از ریزپردازنده‌های این خانواده مثل ۸۰۴۸۶ دارای خط آدرس ۳۲ بیتی هستند، زیرا دارای ثبات ۳۲ بیتی می‌باشند)، که با استفاده از ۲۰ خط آدرس، می‌توان یک مگابایت از حافظه را آدرس‌دهی نمود. برای ایجاد ۲۰ خط آدرس از ترکیب آدرس سگمنت (۱۶ بیتی) و آدرس تفاوت مکان (۱۶ بیتی) استفاده می‌شود؛ همانطور که گفته شد دستورات برنامه پس از تبدیل به دستورات ماشین در ناحیه کد قرار می‌گیرند. هر دستور در این ناحیه نسبت به ابتدای ناحیه دارای یک آدرس است (آدرس تفاوت مکان) که این آدرس ۱۶ بیتی است (لذا طول ناحیه نمی‌تواند بیش از ۶۴کیلوبایت باشد). هر سگمنت در حافظه دارای آدرس است که این آدرس نیز ۱۶ بیتی است (آدرس سگمنت). برای به دست آوردن واقعی یک دستور از برنامه، محتویات ثبات سگمنت کد، چهار محل به سمت چپ شیفت داده شده با آدرس تفاوت مکان جمع می‌شود. نتیجهٔ حاصل، آدرس واقعی دستور می‌باشد. به عنوان مثال، اگر محتویات ثبات سگمنت برابر با 20H و تفاوت مکان برابر با 100H باشد، برای تولید آدرس واقعی دستور، به صورت زیر عمل می‌شود:

محتویات ثبات سگمنت :	**0000000000100000**
محتویات ثبات سگمنت پس از شیفت :	**0000000000010000**
تفاوت مکان :	**0000001000000000**
تفاوت مکان + ثبات سگمنت :	**0000000001100000000**

در مثال فوق، محتویات ثبات سگمنت 0020 است که پس از شیفت به چپ، 00200 حاصل شده و با تفاوت مکان 100H جمع می‌شود. نتیجهٔ حاصل 300H است. برای روشن‌تر شدن موضوع به مثالی دیگر توجه نمایید.

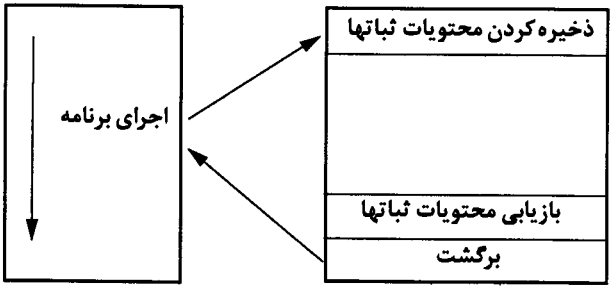
محتویات ثبات سگمنت :	**1111000000000000**
محتویات ثبات سگمنت پس از شیفت :	**1111000000000000**
تفاوت مکان :	**0000000000000001**
تفاوت مکان + ثبات سگمنت :	**11110000000000000001**

در مثال فوق، محتویات ثبات سگمنت F000H است که پس از شیفت به چپ F0000H حاصل شده، با آدرس تفاوت مکان که برابر با یک است جمع می‌شود. نتیجه حاصل F0001H است.

مقدمه‌ای بر وقفه‌ها

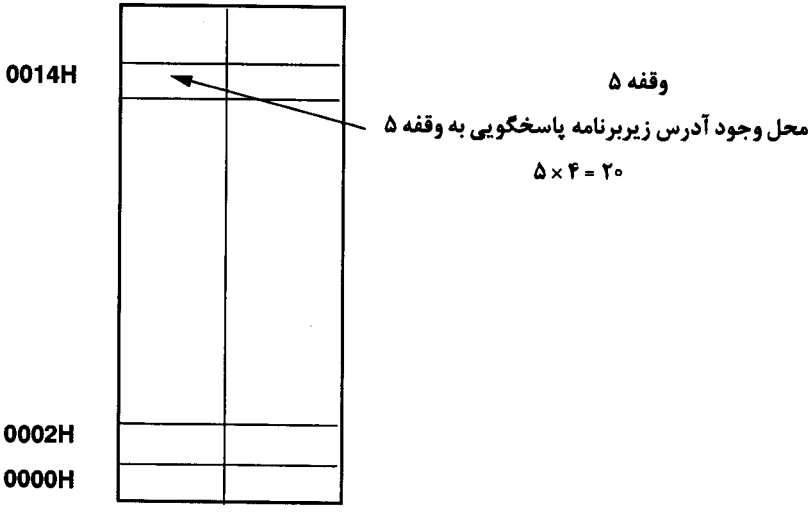
وقفه، سیگنالی از دستگاههای جانبی و یا برنامهٔ در حال اجرا است که عمل خاصی را درخواست می‌کند. وقتی وقفه‌ای اتفاق می‌افتد، اجرای برنامه به تعویق افتاده محتویات ثباتهای CS و IP در پشته نگهداری شده، کنترل به روال پاسخگویی به این وقفه منتقل می‌شود. پس از اجرای این روال، محتویات CS و IP از پشته بازیابی می‌شوند و اجرای برنامه‌ای که به تعویق افتاده بود، ادامه می‌یابد (شکل ۵-۱۲).

همانطوری که در شکل ۵-۱۲ مشاهده می‌شود، در شروع اجرای زیربرنامهٔ وقفه، محتویات ثباتها در پشته ذخیره شده در خاتمه اجرای آن، مجدداً بازیابی می‌شوند تا مشکلی در اجرای برنامه ایجاد نشود.



شکل ۵-۱۲
نحوهٔ اجرای زیربرنامه وقفه.

تعداد وقفه‌هایی که در سیستم مورد استفاده قرار می‌گیرند ۲۵۶ عدد است که از صفر تا ۲۵۵ شماره‌گذاری می‌شوند. هر وقفه دارای زیربرنامه‌ای مخصوص به خود است که می‌تواند به آن وقفه پاسخ دهد. آدرس زیربرنامه‌های پاسخگویی به وقفه‌ها در جدولی به نام جدول بردار وقفه قرار دارد. وقتی که وقفه‌ای اتفاق می‌افتد آدرس زیربرنامه‌ای که باید به این وقفه پاسخ دهد، از جدول بردار وقفه پیدا شده اجرای آن آغاز می‌گردد. آدرس شروع هر زیربرنامه پاسخگویی به وقفه، براساس آدرس ناحیه و تفاوت مکان در آن ناحیه محاسبه می‌شود (یعنی آدرس شروع زیربرنامه پاسخگویی وقفه، ۳۲ بیتی است). چون ۲۵۶ نوع وقفه ممکن است وجود داشته باشد، لذا میزان حافظه‌ای که برای جدول بردار وقفه در نظر گرفته می‌شود برابر با ۱۰۲۴=۴×۲۵۶ بایت است که فضای آدرس 0H تا 3FFH را اشغال می‌کند. برای پیداکردن آدرس زیربرنامه مربوط به یک وقفه، شماره آن وقفه در ۴ ضرب می‌شود، نتیجهٔ حاصل، محلی از جدول بردار وقفه است که آدرس زیربرنامه پاسخگویی به آن وقفه را مشخص می‌کند (شکل ۶-۱۲).



شکل ۶-۱۲
جدول بردار وقفه.

بعضی از وقفه‌ها دارای یک یا چند تابع هستند که با استفاده از آنها می‌توان اعمال متفاوتی را انجام داد. هر تابع همانند وقفه دارای یک شماره است و شماره‌گذاری هر تابع در یک وقفه مختص همان وقفه است. به عنوان مثال وقفهٔ شماره 10H دارای تابعی با شماره ۱ است و وقفه شماره ۱۳ نیز دارای تابعی با همین شماره می‌باشد. در حین فراخوانی وقفه‌ها، شماره تابع در ثبات AH قرار می‌گیرد و اگر اطلاعات دیگری جهت اجرای وقفه نیاز باشد در ثباتهای AL، BX، CX و DX قرار خواهند گرفت.

انواع وقفه‌ها

وقفه‌ها به طور کلی به دو دسته تقسیم می‌شوند :

۱. وقفه‌های سخت‌افزاری.
۲. وقفه‌های نرم‌افزاری.

بحث در مورد وقفه‌های سخت‌افزاری از این مقوله خارج است ولی آشنایی با وقفه‌های نرم‌افزاری توشهٔ این فصل می‌باشد. وقفه‌های نرم‌افزاری به دو دسته تقسیم می‌شوند:

۱. وقفه‌هایی که توسط برنامه‌نویس ایجاد می‌شوند.
۲. وقفه‌های سیستم.

هر برنامه‌نویس می‌تواند علاوه بر وقفه‌هایی که در سیستم وجود دارد (از شماره صفر تا شماره ۲۵۵) وقفه‌های دیگری را پیاده‌سازی نماید. مقدمه‌ای که در این فصل در مورد وقفه‌ها گفته شد به وقفه‌های سیستم برمی‌گردد. وقفه‌های سیستم به دو دسته تقسیم می‌شوند:

۱. وقفه‌های بایوس (BIOS)
۲. وقفه‌های DOS

DOS و BIOS دو قسمت مختلف از سیستم عامل ریزکامپیوتر هستند. BIOS زیربرنامه‌های سطح پایینی را جهت اعمال ورودی - خروجی تدارک می‌بیند ولی DOS اعمال سطح بالاتری را انجام می‌دهد. به طور کلی، قسمتی از سیستم عامل که توسط بارکننده خودکار به حافظه منتقل شده و اجرا می‌گردد، DOS نام دارد.

وقفه‌های بایوس

تعداد ۱۲ وقفه BIOS در سیستم عامل مورد استفاده قرار می‌گیرد (جدول ۱-۱۲).

معرفی توابع چند وقفه

برای بررسی اعمال توابع مربوط به وقفه‌ها به دو نکته توجه داشته باشید :

۱. کلیه اعداد در مبنای ۱۶ بیان می‌شوند.
۲. منظور از "ورودی" اطلاعاتی است که در حین فراخوانی وقفه باید به سیستم داده شوند و منظور از "خروجی" اطلاعاتی است که سیستم در اختیار قرار می‌دهد.

جدول ۱-۱۲ وقفه‌های BIOS.

شماره وقفه	عمل وقفه
5H	فراهم نمودن امکانات چاپ محتویات صفحه‌نمایش
10H	ورودی - خروجی صفحه نمایش
11H	تشخیص دستگاههای جانبی
12H	تعیین میزان حافظهٔ اصلی کامپیوتر
13H	ورودی - خروجی دیسک
14H	ورودی - خروجی پورت سری
15H	ورودی - خروجی کاست
16H	ورودی - خروجی صفحه کلید
17H	ورودی - خروجی چاپگر
18H	اجرای ROM BASIC
19H	اجرای بارکنندهٔ خودکار
1AH	زمان و تاریخ

جدول ۲-۱۲ حالات مختلف صفحهٔ نمایش.

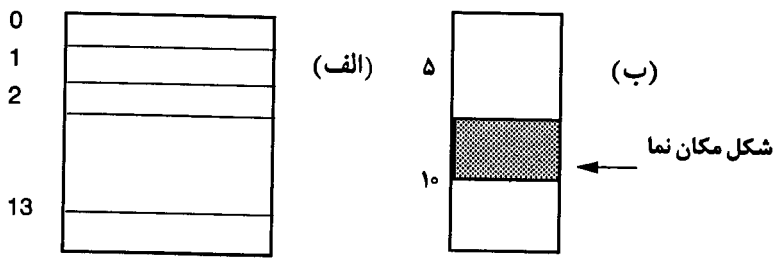
	حالت صفحه‌نمایش	ابعاد آرایه گرافیکی	ابعاد آرایه متن
۰	متن، سیاه و سفید	-	40x25
۱	متن و ۱۶ رنگ	-	40x25
۲	متن، سیاه و سفید	-	80x25
۳	متن و ۱۶ رنگ	-	80x25
۴	گرافیک و ۴ رنگ	320x200	40x25
۵	گرافیک و سیاه و سفید	320x200	40x25
۶	گرافیک و دو رنگ	640x200	80x25
۷	متن، سیاه و سفید	-	80x25
۸	گرافیک و ۱۶ رنگ در PCjr	160x200	20x25
۹	گرافیک و ۱۶ رنگ در PCjr	310x200	40x25
۱۰	گرافیک و ۴ رنگ	640x200	80x25
۱۱	رزرو شده		
۱۲	رزرو شده		
۱۳	گرافیک و ۱۶ رنگ	320x200	40x25
۱۴	گرافیک و ۱۶ رنگ	640x200	80x25
۱۵	گرافیک و ۲ رنگ	640x350	80x25
۱۶	گرافیک و ۱۶ رنگ	640x350	80x25
۱۷	گرافیک و ۲ رنگ	640x480	80x30
۱۸	گرافیک و ۱۶ رنگ	640x480	80x30
۱۹	گرافیک و ۲۵۶ رنگ	640x200	40x25

شماره وقفه : ۱۰	شماره تابع : ۰
عمل تابع : تعیین حالت صفحه‌نمایش(جدول ۲-۱۲)	
ورودی	خروجی
AH : ۰ AL : شماره تعیین کننده حالت صفحه‌نمایش	محتویات ثبات AX از بین می‌رود

شکل مکان‌نما در حالت عادی به صورت خط زیر (underline) است. در صفحه‌نمایش تک رنگ و بورد EGA ماتریسی که شکل مکان‌نما را تعیین می‌کند دارای ۱۴ سطر و در صفحه‌نمایش رنگی و بورد CGA دارای ۸ سطر است که وضعیت آن در صفحه‌نمایش تک‌رنگ، به صورت شکل ۷-۱۲(الف) خواهد بود. اگر محتویات ثبات CH برابر با ۵ و محتویات ثبات CL برابر با ۹ باشد و سپس تابع ۱ از وقفه ۱۰ فراخوانی شود شکل مکان‌نما به صورت شکل ۷-۱۲(ب) خواهد بود.

شماره وقفه : ۱۰	شماره تابع : ۱
عمل تابع : تعیین اندازه مکان‌نما	
ورودی	خروجی
AH : ۱ CH : سطر شروع CL : سطر پایان	محتویات ثبات AX از بین می‌رود

شماره وقفه : ۱۰	شماره تابع : ۲
عمل تابع : تعیین محل مکان‌نما	
ورودی	خروجی
AH : ۲ DH : شماره سطری که مکان‌نما باید منتقل شود DL : شماره ستونی که مکان‌نما باید منتقل شود BH : شماره صفحه‌ای از حافظه نمایش	



شکل ۷-۱۲ وضعیت مکان‌نما.

شماره وقفه : ۱۰	شماره تابع : ۳
عمل تابع : خواندن موقعیت فعلی مکان‌نما	
ورودی	خروجی
AH : ۳ BH : شماره صفحهٔ حافظه نمایش	DH : سطری که مکان‌نما قرار دارد DL : ستونی که مکان‌نما قرار دارد CX : حالت صفحه‌نمایش

شماره وقفه : ۱۰	شماره تابع : ۴
عمل تابع : خواندن موقعیت قلم نوری	
ورودی	خروجی
AH : ۴	AH : ۰ : قلم نوری وجود ندارد ۱ : قلم نوری وجود دارد DH : شماره سطری که قلم نوری وجود دارد DL : شماره ستونی که قلم نوری وجود دارد CH : سطر شروع از ماتریس مکان‌نما BX : ستون شروع از ماتریس مکان‌نما

شماره وقفه : ۱۰	شماره تابع : ۵
عمل تابع : تعیین صفحه حافظه نمایش فعلی	
ورودی	خروجی
AH : ۵ AL : شماره صفحه (۰ تا ۷)	

شماره وقفه : ۱۰	شماره تابع : ۶
عمل تابع : چرخش صفحه‌نمایش یا قسمتی از صفحه به طرف بالا (page up)	
ورودی	خروجی
AH : ۶ AL : تعداد خطوطی که باید پاک شوند CH : شماره سطر گوشه بالای سمت چپ CL : شماره ستون گوشه بالای سمت چپ DH : شماره سطر گوشه پایین سمت راست DL : شماره ستون گوشه پایین سمت راست BH : صفت محدوده پاک شده	

شماره وقفه : ۱۰	شماره تابع : ۷
عمل تابع : چرخش صفحه‌نمایش یا قسمتی از صفحه به طرف پایین (page down)	
ورودی	خروجی
همانند تابع ۶	

شماره وقفه : ۱۰	شماره تابع : ۸
عمل تابع : خواندن کاراکتری که در موقعیت مکان‌نما قرار دارد	
ورودی	خروجی
AH : ۸	AL : کاراکتر خوانده شده
BH : شماره صفحه‌ای از حافظه نمایش	AH : صفت کاراکتر

شماره وقفه : ۱۰	شماره تابع : ۹
عمل تابع : نوشتن کاراکتری با صفت مشخص در موقعیت مکان‌نما	
ورودی	خروجی
AH : ۹	
BH : شماره صفحه حافظه نمایش	
BL : صفت کاراکتر	
CX : تعداد کاراکترهایی که باید نوشته شوند	
AL : کاراکتر موردنظر	

شماره وقفه : ۱۰	شماره تابع : A
عمل تابع : نوشتن کاراکتری در موقعیت مکان‌نما	
ورودی	خروجی
A : AH	
BH : شماره صفحه حافظه نمایش	
CX : تعداد کاراکترهایی که باید نوشته شوند	
AL : کاراکتر موردنظر	

شماره وقفه : ۱۰	شماره تابع : B
عمل تابع : تعیین رنگی از جعبهٔ رنگ	
ورودی	خروجی
B : AH	
BH : شماره جعبه رنگ	
BL : شماره رنگ	

شماره وقفه : ۱۰	شماره تابع : C
عمل تابع : نوشتن یک پیکسل(pixel)	
ورودی	خروجی
C : AH	
DX : شماره سطری که پیکسل باید نوشته شود	
CX : شماره ستونی که پیکسل باید نوشته شود	
AL : رنگ	

شماره وقفه : ۱۰	شماره تابع : D
عمل تابع : خواندن یک pixel	
ورودی	خروجی
D : AH DX : شماره سطر CX : شماره ستون	AL : نقطهٔ خوانده شده

شماره وقفه : ۱۰	شماره تابع : E
عمل تابع : نوشتن کاراکتری در صفحه‌نمایش و تغییر موقعیت مکان‌نما	
ورودی	خروجی
E : AH AL : کاراکتر موردنظر BL : رنگ زمینه BH : صفحه حافظه نمایش	

شماره وقفه : ۱۰	شماره تابع : F
عمل تابع : تعیین حالت صفحه‌نمایش	
ورودی	خروجی
F : AH	AL : حالت فعلی صفحه‌نمایش AH : تعداد ستونهای صفحه نمایش BH : شماره صفحهٔ حافظه نمایش

شماره وقفه : ۱۱	شماره تابع : ندارد
عمل وقفه : تشخیص تجهیزات متصل به کامپیوتر	
ورودی	خروجی
ندارد	AH : حاوی تجهیزات سخت‌افزاری کامپیوتر است اگر بیت شماره صفر برابر با یک باشد به معنی وجود یک فلاپی است. بیت شمارهٔ یک مورد استفاده قرار نگرفته است. اگر بیتهای ۲ و ۳ برابر با ۱۱ باشند به معنی وجود ۶۴کیلو بایت حافظه RAM روی بورد سیستم است. بیتهای ۴ و ۵ وضعیت صفحه‌نمایش را مشخص می‌کند: 01 40x25 و متن 10 60x25 و متن رنگی 11 80x25 ، متن و تک رنگ بیتهای ۶ و ۷ تعداد درایوها را مشخص می‌کنند بیت ۸ وجود آی‌سی DMA را تشخیص می‌دهد (اگر صفر باشد به معنی وجود این آی‌سی است) بیتهای ۹ و ۱۰ و ۱۱ تعداد پورتهای RS-232 را مشخص می‌کند. اگر بیت ۱۲ برابر با یک باشد به معنی وجود تطبیق دهنده بازی است اگر بیت ۱۳ برابر با یک باشد به معنی وجود چاپگر سری است (در کامپیوتر PCjr) بیتهای ۱۴ و ۱۵ تعداد چاپگرها را مشخص می‌کند

شماره وقفه : ۱۲	شماره تابع : ندارد
عمل وقفه : تعیین میزان حافظه اصلی	
ورودی	خروجی
ندارد	AX : میزان حافظه اصلی کامپیوتر (RAM) به کیلوبایت

شماره وقفه : ۱۳	شماره تابع : ۰
عمل وقفه : RESET نمودن سیستم دیسک	
ورودی	خروجی
AH : ۰	

شماره وقفه : ۱۳	شماره تابع : ۱
عمل وقفه : خواندن وضعیت دیسک	
ورودی	خروجی
AH : ۱	AL : وضعیت دیسک

شماره وقفه : ۱۳	شماره تابع : ۲
عمل وقفه : خواندن سکتورهایی از دیسک و انتقال آن به حافظه	
ورودی	خروجی
AH : ۲	AL : تعداد سکتورهایی که خوانده شدند
DL : شماره درایو (۰ برای A و ...)	AH :
DH : شماره هد خواندن و نوشتن	صفر : عمل تابع با موفقیت انجام شد
CH : شماره تراک	غیر از صفر: تابع با موفقیت عمل نکرده است
CL : شماره سکتور	
AL : تعداد سکتورهایی که باید خوانده شود	
BX : ES : آدرس بافر	

شماره وقفه : ۱۳	شماره تابع : ۳
عمل وقفه : نوشتن محتویات حافظه بر روی سکتورهایی از دیسک	
ورودی	خروجی
AH : ۳	AL : شماره درایو
AL : تعداد سکتورهایی که باید نوشته شود	
BX : ES : آدرس محلی از حافظه که اطلاعات باید از آنجا برداشته شوند	AH
CL : شماره سکتوری که نوشتن اطلاعات باید از آنجا شروع شود	صفر : عمل تابع با موفقیت انجام شد
CH : شماره تراک	غیر از صفر: با خطا مواجه شده است
DL : شماره درایو	
DH : شماره هد خواندن و نوشتن	

شماره وقفه : ۱۳	شماره تابع : ۴
عمل وقفه : بازبینی سکتورهای دیسک	
ورودی	خروجی
AH : ۴ همانند تابع ۳	همانند تابع ۳

شماره وقفه : ۱۳	شماره تابع : ۵
عمل وقفه : فرمت کردن تراکهایی از دیسک	
ورودی	خروجی
AH : ۵ DL : شماره درایو DH : شماره هد خواندن و نوشتن CH : شماره تراک BX : ES : مشخصات تراک (شماره هد خواندن و نوشتن، ظرفیت)	

شماره وقفه : ۱۳	شماره تابع : ۸
عمل وقفه : خواندن پارامترهای یک درایو	
ورودی	خروجی
AH : ۸ DL : شماره درایو	DL : تعداد دیسکهای سخت کامپیوتر DH : حداکثر تعداد هد خواندن و نوشتن CH : حداکثر تعداد سیلندرها CL : حداکثر تعداد سکتورها

شماره وقفه : ۱۳	شماره تابع : A
عمل تابع : خواندن تعدادی سکتور از روی دیسک و انتقال آن به حافظه	
ورودی	خروجی
A : AH DL : شماره درایو DH : شماره هد خواندن و نوشتن CH : شماره سیلندر CL : شماره سکتور AL : تعداد سکتورها BX : ES : محل نوشتن سکتورهای خوانده شده	AH : وضعیت عمل صفر : با موفقیت انجام شد غیر از صفر : با موفقیت انجام نشد

شماره وقفه : ۱۳	شماره تابع : B
عمل تابع : نوشتن اطلاعات با حجم زیاد از حافظه بر روی دیسک	
ورودی	خروجی
B : AH همانند تابع A	همانند تابع A

شماره وقفه : ۱۳	شماره تابع : ۱۵
عمل تابع : تشخیص نوع دیسک	
ورودی	خروجی
AH : ۱۵	AH : ۰ : دیسکی وجود ندارد ۱ : دیسک ۳۶۰کیلوبایتی دو طرفه ۲ : دیسک ۱/۲ مگابایتی ۳ : دیسک سخت
DL : شماره درایو	

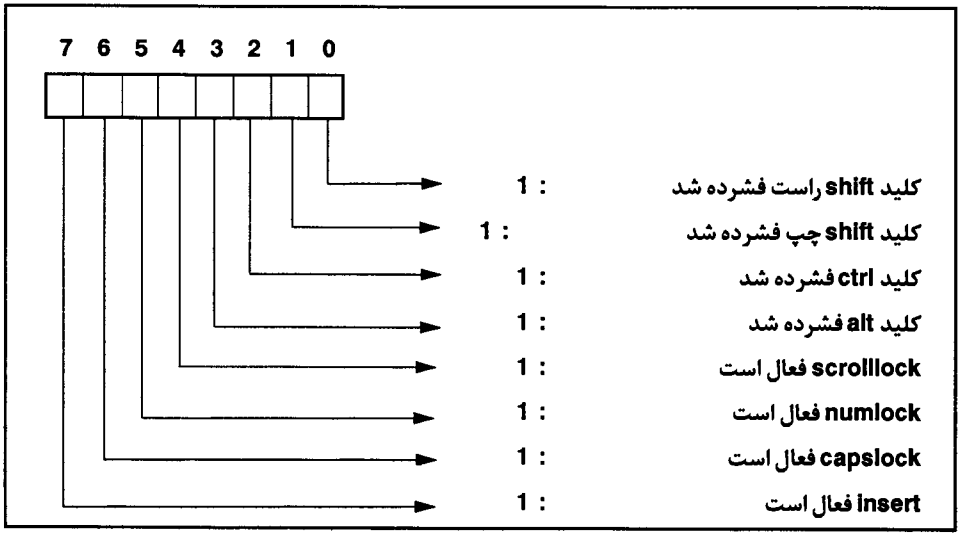
شماره وقفه : ۱۵	شماره تابع : ۸۰
عمل تابع : تشخیص میزان حافظه توسعه یافته	
ورودی	خروجی
AH : ۸۰	AH : میزان حافظه توسعه یافته

شماره وقفه : ۱۶	شماره تابع : ۰
عمل تابع : خواندن کد پیمایش	
ورودی	خروجی
AH : ۰	AH : کد توسعه یافته AL : کد کاراکتر

شماره وقفه : ۱۶	شماره تابع : ۱
عمل تابع : تشخیص وضعیت بافر صفحه کلید	
ورودی	خروجی
AH : ۱	ZF : 1 : بافر خالی است 0 : کاراکتری در بافر وجود دارد AL : کد کاراکتر AH : کد توسعه یافته

شماره وقفه : ۱۶	شماره تابع : ۲
عمل تابع : اخذ وضعیت کلیدهای SHIFT ، CTRL ، ALT و...	
ورودی	خروجی
AH : ۲	AL : وضعیت صفحه کلید(شکل۸-۱۲)

شماره وقفه : ۱۷	شماره تابع : ۰
عمل تابع : چاپ یک کاراکتر در چاپگر	
ورودی	خروجی
AH : ۰	AH : نتیجه عمل : صفر : کاراکتر چاپ شد غیر صفر : با اشکال مواجه شده است
AL : کاراکتر موردنظر	
DX : شماره چاپگر (پورتی که چاپگر به آن متصل است)	



شکل ۸-۱۲ بایت وضعیت صفحه کلید.

شماره وقفه : ۱۷	شماره تابع : ۱
عمل تابع : ارزش دهی اولیه به چاپگر	
ورودی	خروجی
AH : ۱ DX : شماره چاپگر	AHوضعیت چاپگر

شماره وقفه : ۱۷	شماره تابع : ۲
عمل تابع : تشخیص وضعیت چاپگر*	
ورودی	خروجی
AH : ۲ DX : شماره چاپگر	AH : وضعیت چاپگر به شرح زیر:

شماره بیت	وضعیت چاپگر
۰	پایان مهلت زمانی
۳	خطای ورودی - خروجی
۴	چاپگر انتخاب شده است
۵	کاغذ تمام شده است
۶	ارسال پیام از چاپگر
۷	چاپگر آزاد است

شماره وقفه : 1A	شماره تابع : ۰
عمل تابع : خواندن ساعت جاری سیستم	
ورودی	خروجی
AH : ۰	CX : ۱۶ بیت باارزش ساعت جاری DX : ۱۶ بیت کم‌ارزش ساعت جاری AL : سرریز (براساس ۲۴ ساعت)

شماره وقفه : 1A	شماره تابع : ۱
عمل تابع : مقدار دادن به ساعت سیستم	
ورودی	خروجی
AH : ۱ CX : ۱۶ بیت باارزش DH : ۱۶ بیت کم ارزش	

شماره وقفه : 1A	شماره تابع : ۲
عمل تابع : خواندن زمان سیستم (ساعت - دقیقه - ثانیه)	
ورودی	خروجی
AH : ۲ CL : دقیقه DH : ثانیه	CX : ساعت

شماره وقفه : 1A	شماره تابع : ۳
عمل تابع : اعلام ساعت به سیستم	
ورودی	خروجی
AH : ۳ CX : ساعت CL : دقیقه DH : ثانیه	

شماره وقفه : 1A	شماره تابع : ۴
عمل تابع : اخذ تاریخ جاری سیستم	
ورودی	خروجی
AH : ۴	CH : قرن (۱۹ یا ۲۰) (BCD) CL : سال (۱۹ یا ۲۰) (BCD) DH : ماه (۱۹ یا ۲۰) (BCD) DL : روز (۱۹ یا ۲۰) (BCD) AL : قرن (۱۹ یا ۲۰)

شماره وقفه : 1A	شماره تابع : ۵
عمل تابع : اعلام تاریخ به سیستم	
ورودی	خروجی
AH : ۵ CH : قرن (BCD) CL : سال (BCD) DH : ماه (BCD) DL : روز (BCD)	

اجرای وقفه‌ها در C

تاکنون عمل هر یک از وقفه‌های بایوس(BIOS) و بعضی از توابع آنها مورد بررسی قرار گرفته‌اند. اکنون باید مشخص گردد که در زبان C چگونه می‌توان آنها را مورد استفاده قرار داد. برای فراخوانی وقفه‌های بایوس در C از تابع int86() استفاده می‌شود. الگوی تابع int86() در فایل dos.h قرار دارد و به صورت زیر است:

```
int86 (int intnum, union REGS  *in, union REGS  *out)
```

در الگوی فوق، intnum شماره وقفه، in (پارامتر ورودی) و out (پارامتر خروجی) و از نوع یونیون می‌باشند. REGS ساختمانی از نوع یونیون است که در فایل dos.h تعریف شده است. علاوه بر این ساختمان، ساختمانهای دیگری نیز در این فایل تعریف شده‌اند که عبارتنداز:

```
struct WORDREGS {
    unsigned int ax, bx, cx, dx, si, di, cflag, flags ;
} ;

struct BYTEREGS {
      unsigned char al, ah, bl, bh, cl, ch, dl, dh ;
} ;

union REGS {
     struct WORDREGS x ;
     struct BYTEREGS h ;
} ;
```

```
struct SREGS {
      unsigned int es ;
      unsigned int cs ;
      unsigned int ss ;
      unsigned int ds ;
} ;

struct REGPACK {
      unsigned r_ax, r_bx, r_cx, r_dx ;
      unsigned r_bp, r_si, r_dl, r_ds, r_es, r_flags ;
} ;
```

مثال ۱-۱۲

برنامه‌ای که با استفاده از وقفه 10H و تابع ۶، صفحه‌نمایش را پاک می‌کند و با استفاده از تابع ۲، مکان‌نما را به موقعیت مناسبی منتقل کرده، متنی را در آنجا می‌نویسد.

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
void goto_xy(int x , int y);
void main() {
   void cls() ;
   register int x, y ;
   cls() ;
   goto_xy(10, 10) ;
   printf("this is a test.");
   getch();
}
//*******************
void goto_xy(int x , int y) {
     union REGS r ;
     r.h.ah = 2 ;
     r.h.dl = y ;
     r.h.dh = x ;
     r.h.bh = 0 ;
     int86(0x10, &r, &r) ;
}
//*******************
void cls(void) {
    union  REGS r ;
    r.h.ah = 6 ;
    r.h.al = 0 ;
    r.h.ch = 0 ;
    r.h.cl = 0 ;
    r.h.dh = 23 ;
    r.h.dl = 79 ;
    r.h.bh = 7 ;
    int86(0x10, &r, &r) ;
}
```

مثال ۲-۱۲

برنامه‌ای که با استفاده از وقفه ۵، محتویات صفحه‌نمایش را به چاپگر منتقل می‌کند.

```
#include <dos.h>
void print_screen() ;
void main()
{
     print_screen() ;
}
//**************
void print_screen()
{
     union REGS intreg, outreg ;
     int86(5, &intreg, &outreg) ;
}
```

مثال ۳-۱۲

برنامه‌ای که اندازه مکان‌نما را تعیین می‌کند (وقفه 10H و تابع ۱).

```
#include <dos.h>
#include <conio.h>
void set_cur_size(int, int) ;
void main()
{
    int start, stop ;
    clrscr();
    start = 0 ;
    stop = 7 ;
     set_cur_size(start, stop) ;
    getch();
}
//**********************
void set_cur_size(int start, int stop)
{
    union REGS in,out ;
    in.h.ah = 1 ;
    in.h.ch = start ;
    in.h.cl = stop ;
    int86(0x10, &in, &out) ;
}
```

مثال ۴-۱۲

برنامه‌ای که با استفاده از وقفه 10H و تابع ۳، موقعیت مکان‌نما را می‌خواند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void get_cur_pos(int, int *, int *, int *, int *);
void main() {
  int page, row, column, start, stop;
  clrscr();
  get_cur_pos(0, &row, &column, &start, &stop);
  printf("\n row=%d, col=%d, start=%d, stop=%d", row, column, start, stop);
  getch();
}
//**********************
void get_cur_pos(int page, int *row, int *column, int *start, int *stop)
{
      union REGS in, out ;
      in.h.ah = 3 ;
      in.h.bh = page;
      int86(0x10, &in, &out);
      *row = out.h.dh ;
      *column = out.h.dl ;
      *start = out.h.ch ;
      *stop = out.h.cl ;
}
```

مثال ۵-۱۲

برنامه‌ای که جعبه رنگ و رنگی از این جعبه را انتخاب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void set_palette(int palette, int color);
void main() {
   int p, color;
   clrscr();
   set_palette(p, color);
   getch();
}
//********************
void set_palette(int palette, int color)
{
    union REGS in, out;
    in.h.ah = 0x0B;
    in.h.bh = palette;
    in.h.bl = color;
    int86(0x10, &in, &out);
}
```

مثال ۶-۱۲

برنامه‌ای که موجب تغییر حالت صفحه‌نمایش می‌گردد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void set_mode(int mode);
void main() {
   int mode = 3;
   set_mode(mode);
   getch();
}
//************
void set_mode(int mode)
{
    union REGS r ;
    r.h.al=mode ;
    r.h.ah=0 ;
    int86(0x10,&r,&r) ;
}
```

مثال ۷-۱۲

برنامه‌ای که کاراکترهایی با صفت خاص را در یک صفحه از حافظه نمایش به تعداد دفعات دلخواهی می‌نویسد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void write_char(int page, int ch, int attr, int count);
void main() {
 char ch;
 clrscr();
 printf("\enter char to write:");
 ch = getch();
 write_char(0, ch, 7, 5);
 getch();
}
//*****************
void write_char(int page, int ch, int attr, int count)
{
      union REGS in,out ;
      in.h.ah = 9 ;
      in.h.al = ch ;
      in.h.bh = page ;
      in.h.bl = attr ;
      in.h.ch = 0 ;
      in.h.cl = count ;
      int86(0x10, &in, &out) ;
}
```

مثال ۸-۱۲

برنامه‌ای که میزان حافظه اصلی کامپیوتر را تشخیص می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int memory_size();
void main() {
   clrscr();
   printf("base memory size = %dk", memory_size()) ;
   getch();
}
//****************
int memory_size()
{
    union REGS in, out;
    int86(0x12, &in, &out);
    return(out.x.ax) ;
}
```

مثال ۹-۱۲

برنامه‌ای که هر کلید فشار داده شده از صفحه کلید را تشخیص می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int get_key(void);
void main() {
     union scan {
       int c ;
       char ch[2] ;
     } sc ;
     clrscr();
     do {
        sc.c = get_key() ;
        if(sc.ch[0] == 0)
           printf("\n key is special.");
        else
          printf("\n %c", sc.ch[0]) ;
     } while (sc.ch[0] != 'q') ;
}
//******************
int get_key(void)
{
     union REGS in, out;
     in.h.ah = 0 ;
     int86(0x16, &in, &out) ;
     return out.x.ax;
}
```

مثال ۱۰-۱۲

برنامه‌ای که رنگ حاشیه صفحه‌نمایش را تعیین می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void set_border(int) ;
void main() {
   clrscr();
   set_border(10);
   getch();
}
//********************
void set_border(int color) {
     union REGS in, out ;
     in.h.ah = 0x0b ;
     in.h.bh = 0 ;
     in.h.bl = color ;
     int86(0x10, &in, &out) ;
}
```

مثال ۱۱-۱۲

تابعی که وضعیت صفحه کلید را تست می‌کند و تشخیص می‌دهد که آیا کلید Numlock فعال است یا خیر. اگر بیت ششم از بایت وضعیت صفحه کلید برابر با یک باشد، یعنی Numlock فشار داده شده است. اگر این بیت یک باشد، یعنی مقدار بایت وضعیت برابر با ۳۲ است. به همین دلیل، مقدار بایت وضعیت با ۳۲ که نمایش بیتی آن 00100000 است، AND می‌شود. اگر حاصل کار ۳۲ باشد، یعنی Numlock، فعال است.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int get_state();
void main() {
 unsigned char x;
 clrscr();
 x = (unsigned char) get_state();
 x = x & 32 ;
 if (x == 32)
     printf("NUM LOCK key is on");
 else
     printf("NUM LOCK key is not  on");
 getch();
}
//******************
int get_state() {
     union REGS in, out;
     in.h.ah = 2 ;
     int86(0x16, &in, &out);
     return(out.h.al);
}
```

توابع DOS

DOS قسمتی از سیستم عامل است که بعضی از اعمال سطح بالاتری را نسبت به بایوس انجام می‌دهد. وقفه 21H برای فراخوانی توابع DOS مورد استفاده قرار می‌گیرد. این وقفه در حدود ۱۰۰ تابع دارد که بعضی از آنها در جدول ۳-۱۲ آمده‌اند. بعضی از اعمالی که این توابع انجام می‌دهند عبارتنداز:

۱. اعمال روی فایل (بازکردن، خواندن، نوشتن و بستن).
۲. خواندن از صفحه کلید.
۳. اجرا و پایان برنامه.
۴. ایجاد، حذف و تغییر نام فایل.
۵. مدیریت حافظه (اخذ حافظه، آزادسازی و تغییر).
۶. دستکاری کنترل‌کننده‌های دیسک.
۷. دستکاری فهرستها (ایجاد، حذف، تغییر مسیر).

جدول ۳-۱۲ بعضی از توابع وقفه 21H.

تابع	عمل
1	خواندن کاراکتری از صفحه کلید. AL حاوی این کاراکتر خواهد بود
2	نوشتن کاراکتری در صفحه‌نمایش. DL حاوی کاراکتری است که باید نوشته شود
3	خواندن کاراکتری از پورت غیرهمزمان. AL حاوی این کاراکتر خواهد بود
4	نوشتن کاراکتری در پورت غیرهمزمان. DL حاوی کاراکتری است که باید نوشته شود
5	نوشتن کاراکتری بر روی چاپگر. DL حاوی کاراکتری است که باید چاپ شود
7	خواندن کاراکتری از صفحه کلید، بدون نمایش آن در صفحه‌نمایش
B	وضعیت صفحه کلید را تست می‌کند
D	reset کردن دیسک
E	تعیین درایو فعال. DL حاوی شماره درایوی است که باید به عنوان درایو فعال انتخاب گردد (A=0، B=1و...)
1A	تعیین آدرس ناحیه انتقال اطلاعات (DTA) DX حاوی آدرس است
2A	اخذ تاریخ از سیستم: CX شامل سال، DH شامل ماه و DL شامل روز خواهد بود
2B	اعلام تاریخ به سیستم، CX حاوی سال، DH حاوی ماه و DL حاوی روز است
2C	اخذ زمان از سیستم: CH شامل ساعت، CL شامل دقیقه، DH شامل ثانیه و DL شامل صدم ثانیه خواهد بود
2D	اعلام زمان به سیستم: CH حاوی ساعت، CL حاوی دقیقه، DH حاوی ثانیه و DL حاوی صدم ثانیه است
4E	جستجو برای نام فایل، DX حاوی آدرس FCB و CX حاوی صفت فایل است. اگر فایل پیدا نشود مقدار ۲ در AL قرار می‌گیرد و اگر هیچ فایلی پیدا نشود مقدار ۱۸ در AL قرار خواهد گرفت.
4F	ادامه جستجو برای پیداکردن نام فایل پس از پیداشدن اولین وقوع آن توسط تابع 4E. اگر فایلی وجود نداشته باشد، مقدار ۱۸ در AL قرار می‌گیرد.

برای فراخوانی توابع DOS در زبان C از توابع intdos()، bdos() و bdosptr() استفاده می‌گردد. الگوی این توابع عبارتند از:

```
int intdos (union REGS *in, union REGS *out)
int bdos (int fnum, unsigned dx, unsigned al)
int bdosptr (int fnum, void *dsdx, unsigned al)
```

در الگوهای فوق، in و out ساختمانهای ورودی و خروجی هستند و fnum شماره تابع وقفه می‌باشد. مقادیر dx و al به‌ترتیب در ثباتهای DX و AL قرار می‌گیرند. نتیجهٔ عمل در هر ۳ تابع در ثبات AX قرار خواهد گرفت. تابع bdosptr() موقعی مورد استفاده می‌گیرد که یک اشاره‌گر به عنوان آرگومان انتخاب می‌شود.

مثال ۱۲-۱۲

برنامه‌ای که لیست فایلهای موجود در درایو جاری را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void dir_list() ;
void main()
{
    clrscr();
    dir_list();
    getch();
}
//********************
void dir_list()
{
    union REGS in, out ;
    char pp[44] ;
    bdosptr(0x1a, pp, 0) ;
    bdosptr(0x4e, "*.*", 0) ;
    printf("\n %s \n", &pp[30]) ;
    for(;;) {
      if(bdos(0x4f, 0, 0) == 18)
        break ;
      printf("\n %s \n", &pp[30]) ;
    }//end of for
}
```

مثال ۱۳-۱۲

برنامه‌ای که کاراکتری را از بافر صفحه‌کلید می‌خواند، این برنامه، منتظر فشاردادن کلیدی از صفحه‌کلید نمی‌ماند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int stdin_char();
void main()
{
  int ch;
  clrscr();
  ch = stdin_char();
  printf("%d", ch);
  getch();
}
//*****************
int stdin_char()
{
     union REGS in, out ;
     in.h.ah = 0x10 ;
     intdos(&in, &out) ;
     return(out.h.al) ;
}
```

مثال ۱۴-۱۲

تابعی که مشخصات دیسک مربوط به درایو جاری (تعداد cluster، اندازه هر سکتور و تعداد سکتورهای موجود در یک cluster) را مشخص می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void disk_info(int *spc, int *sector_size, int *num_cluster);
void  main() {
    clrscr();
    int spc, sector_size, num_cluster;
    disk_info( &spc, &sector_size, &num_cluster);
    printf("spc=%d, siz=%d, num=%d", spc, sector_size, num_cluster);
    getch();
}
//******************
void disk_info(int *spc, int *sector_size, int *num_cluster)
{
     union REGS in, out;
     in.h.ah = 0x1b;
     intdos(&in, &out);
     *spc = out.h.al;
     *sector_size = out.x.cx;
     *num_cluster = out.x.dx;
}
```

مثال ۱۵-۱۲

برنامه‌ای که موجب نوشتن کاراکتر a در صفحه‌نمایش می‌شود.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void stdout_char(char) ;
void main()
{
   clrscr();
   stdout_char('a');
   getch();
}
//*********************
void stdout_char(char character)
{
    union REGS in , out ;
    in.h.ah = 0x02 ;
    in.h.dl = character ;
    intdos(&in, &out) ;
}
```

مثال ۱۶-۱۲

برنامه‌ای که وضعیت صفحه‌کلید را تست می‌کند. وضعیت صفحه‌کلید در ثبات AL قرار می‌گیرد و توسط تابع check() برگردانده می‌شود. با تست بیتهای آن می‌توان تشخیص داد که کدام کلید فشار داده شده است.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int check();
void main() {
  unsigned char  x;
  clrscr();
  x = (unsigned char) check();
  x = x & 32;
  if (x == 32) printf("nnnnnn");
  getch();
}
//*****************
int check()
{
   union REGS in, out;
   in.h.ah = 0xb;
   intdos(&in, &out);
   return(out.h.al);
}
```

مثال ۱۲-۱۷

برنامه‌ای که درایو A را به عنوان درایو جاری انتخاب می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void set_drive(int drive);
void main() {
    set_drive(0) ;
}
//****************
void set_drive(int drive)
{
    union REGS in, out ;
    in.h.ah = 0x0e ;
    in.h.dl = drive ;
    intdos(&in, &out) ;
}
```

مثال ۱۲-۱۸

برنامه‌ای که تاریخ سیستم را چاپ می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void get_date(int *day, int *month, int *year, int *day_of_week);
void main()
{
  int day, month, year, day_of_week;
  get_date(&day, &month, &year, &day_of_week);
  printf("date is:day %d, %d/%d/%d",day_of_week, day, month, year);
  getch();
}
//*************************
void get_date(int *day, int *month, int *year, int *day_of_week)
{
      union REGS in, out ;
      in.h.ah = 0x2a ;
      intdos(&in, &out) ;
      *day = out.h.dl ;
      *day_of_week = out.h.al ;
      *month = out.h.dh ;
      *year = out.x.cx ;
}
```

مثال ۱۲-۱۹

برنامه‌ای که میزان فضای آزاد موجود در درایو جاری را تشخیص می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
long free_disk(int drive);
void main() {
  int drive = 0; // current drive
  long int spc;
  clrscr();
  spc = free_disk(drive);
  printf("\n free space is : %ld", spc);
  getch();
}
//*****************
long free_disk(int drive)
   {
     union REGS in , out ;
     in.h.ah = 0x36 ;
     in.h.dl = drive ;
     intdos(&in, &out) ;
     return((long) out.x.ax * (long) out.x.bx * (long) out.x.cx);
}
```

مثال ۱۲-۲۰

برنامه‌ای که گونه سیستم عامل را تشخیص می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void dos_ver(int *major, int *minor);
void main() {
     int major, minor;
     clrscr();
     dos_ver(&major, &minor);
     printf("version is:%d.%d", major, minor);
     getch();
}
//****************
void dos_ver(int *major, int *minor)
{
     union REGS in , out ;
     in.h.ah=0x30 ;
     intdos(&in,&out) ;
     *major=out.h.al ;
     *minor=out.h.ah ;
}
```

مثال ۱۲-۲۱

برنامه‌ای که قادر است فهرستی (directory) ایجاد نماید. اگر عمل تابع make_dir() با موفقیت انجام نشود عدد ۱- را به تابع اصلی برمی‌گرداند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int make_dir(char sname[]);
void main()
{
    int st ;
    clrscr();
    st = make_dir("\\basic");
    if(st == -1)
       printf("\n directory not created.");
    else
       printf("\n directory created.");
    getch();
}
//***************
int make_dir(char sname[])
{
    return(bdosptr(0x39, sname, 0));
}
```

مثال ۱۲-۲۲

برنامه‌ای که قادر است فهرستی را حذف نماید. اگر عمل تابع remove_dir() با موفقیت انجام نشود مقدار ۱- را به تابع اصلی برمی‌گرداند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int remove_dir(char sname[]);
void main()
{
    int st ;
    clrscr();
    st = remove_dir("\\basic");
    if(st == -1)
       printf("\n directory not removed.");
    else
       printf("\n directory remove.");
    getch();
}
//***************
int remove_dir(char sname[])
{
    return(bdosptr(0x3a, sname, 0));
}
```

مثال ۱۲-۲۳

برنامه‌ای که فهرست جاری را عوض می‌کند. اگر عمل تابع change_dir() با موفقیت انجام نشود مقدار ۱-را به تابع اصلی برمی‌گرداند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int change_dir(char sname[]);
void main() {
     int st ;
     clrscr();
     st = change_dir("\\basic");
     if(st == -1)
        printf("\n directory not changed.");
     else
        printf("\n directory changed.");
     getch();
}
//***************
int change_dir(char sname[]) {
     return(bdosptr(0x3b, sname, 0));
}
```

مثال ۱۲-۲۴

برنامه‌ای که فایلی را حذف می‌کند. اگر عمل تابع remove_file() با موفقیت انجام نشود، مقدار ۱-را برمی‌گرداند.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
int remove_file(char fname[]);
void main(int argc, char *argv[]) {
     int st ;
     clrscr();
     if(argc < 2) {
        printf(" usage : prog arg");
        getch();
        exit(0);
     }
     st = remove_file(argv[1]);
     if(st == -1)
        printf("\n file  not removed.");
     else
        printf("\n file removed.");
     getch();
}
//***************
int remove_file(char fname[]) {
     return(bdosptr(0x41, fname, 0));
}
```

مدلهای حافظه و مدیریت صفحه کلید

کامپایلر زبان C، در ریزپردازنده‌های خانواده ۸۰۸۶، هر برنامه را می‌تواند به شش صورت ترجمه نماید. هر روش، حافظه کامپیوتر را به شکل مختلفی سازماندهی می‌کند سازماندهی مختلف حافظه، به مدلهای حافظه معروف است. مدلهای مختلف حافظه عبارتنداز: tiny، small، medium، compact، large و huge که در ادامه به بررسی هر کدام از آنها می‌پردازیم.

مدل حافظهٔ tiny

در مدل حافظه tiny، برنامه به زبان C طوری ترجمه می‌شود که کلیه ثباتهای سگمنت حاوی یک مقدار بوده و همهٔ آدرس‌دهی‌ها با ۱۶ بیت انجام شوند. این بدین معنی است که کد برنامه، داده‌های برنامه و پشته در یک سگمنت ۶۴ کیلوبایتی قرار دارند. برنامه‌ای که به این روش ترجمه می‌شود نه تنها حجم کمی را اشغال می‌کند بلکه از سرعت بالایی نیز برخوردار است و توسط دستور EXE2BIN (از دستورات سیستم عامل DOS) قابل تبدیل به فایل COM می‌باشد. مدل حافظه tiny برای برنامه‌هایی که دارای کد و دادهٔ کمی باشند مفید است.

مدل حافظهٔ small

توربو C و بورلند C در حالت معمولی از این مدل حافظه برای ترجمه برنامه‌ها استفاده می‌کنند که برای بسیاری از برنامه‌ها مفید است. گرچه در این مدل حافظه کلیه آدرس‌دهی‌ها با ۱۶ بیت انجام می‌شوند ولی سگمنت کد برنامه از سایر سگمنت‌ها جدا است. برنامه‌ها در این مدل حافظه می‌توانند ۱۲۸ کیلوبایت را اشغال کنند که بین کد برنامه و داده‌های آن مشترک است. زمان آدرس‌دهی در مدلهای حافظه tiny و small یکسان است ولی برنامه‌ای که در مدل حافظه small می‌تواند ترجمه شود دو برابر برنامه‌ای است که در مدل حافظه tiny ترجمه می‌گردد.

مدل حافظهٔ medium

مدل حافظه medium برای برنامه‌های طولانی مورد استفاده قرار می‌گیرد و کد برنامه می‌تواند در چند سگمنت باشد. کد برنامه نیاز به ۲۰ بیت آدرس دارد ولی سایر سگمنت‌های برنامه (داده، پشته و سگمنت اضافی) از ۱۶ بیت آدرس استفاده می‌کنند. این مدل حافظه برای برنامه‌هایی مفید است که کد برنامه طولانی دارند ولی داده‌های برنامه کم است، سرعت اجرای برنامه در این مدل حافظه از مدلهای حافظه tiny و small کمتر است ولی سرعت دسترسی به داده‌های برنامه تقریباً مشابه مدلهای حافظه tiny و small است.

مدل حافظهٔ compact

مدل حافظه compact برعکس مدل حافظه medium است. بدین صورت که در مدل حافظه compact کد برنامه در یک سگمنت محدود می‌شود ولی داده‌های برنامه‌ها می‌توانند در چند سگمنت گسترش یابند؛ لذا برای دسترسی به داده‌های برنامه به ۲۰ بیت آدرس نیاز است ولی کد برنامه به ۱۶ بیت آدرس نیاز دارد. این مدل حافظه برای برنامه‌هایی مفید است که دارای کد کم و حجم داده‌های زیادی هستند. سرعت اجرای برنامه در این مدل حافظه مشابه مدل حافظه small است ولی دسترسی به داده‌ها کمی طولانی‌تر می‌باشد.

مدل حافظهٔ large

در مدل حافظه large کد برنامه و داده‌های آن می‌توانند از چند سگمنت استفاده کنند؛ اما طول دادهٔ استاتیک، مثل یک آرایه محدود به ۶۴کیلوبایت است. این مدل حافظه برای برنامه‌های طولانی با حجم داده‌های زیاد مفید است. سرعت اجرای برنامه‌ها در این مدل حافظه بسیار کندتر از سایر مدلهای حافظه است.

مدل حافظهٔ huge

مدل حافظه huge مشابه مدل حافظه large است با این تفاوت که طول دادهٔ استاتیک مثل آرایه می‌تواند بیش از ۶۴ کیلوبایت باشد. اجرای برنامه‌ها در این مدل حافظه از سایر مدلهای حافظه کندتر است.

انتخاب مدل حافظهٔ مناسب

در حالت معمولی برای ترجمه برنامه‌ها از مدل حافظه small استفاده می‌شود؛ مگر این که به دلایلی مجبور به استفاده از مدلهای دیگر حافظه باشیم. اگر برنامه نسبتاً طولانی و داده‌های برنامه کم باشد بهتر است از مدل حافظه medium استفاده گردد. اگر داده‌های برنامه زیاد بوده ولی حجم برنامه کم باشد بهتر است از مدل حافظه compact استفاده گردد. اگر داده‌ها و کد برنامه هر دو طولانی باشند بهتر است از مدل حافظه large استفاده گردد. مگر این که به داده‌های استاتیک بیش از ۶۴کیلو بایت نیاز باشد که در این مورد بهتر است از مدل حافظه huge استفاده شود.

معرفی یک مدل حافظه به کامپایلر

بهترین روش برای تعیین مدل حافظه در بورلند C این‌است‌که در منوی Options گزینه Compiler را انتخاب کنید و در منویی که ظاهر می‌شود گزینه Code generation را برگزینید. سپس در قسمت Memory model مدل مورد نظر را انتخاب کنید.

آدرس‌دهی به خارج از یک سگمنت حافظه

همانطور که می‌دانید هر ۶۴کیلوبایت از حافظه تشکیل یک سگمنت را می‌دهند. فضای حافظه یک سگمنت توسط یک ثبات ۱۶ بیتی قابل آدرس‌دهی است. اگر در برنامه‌ای نیاز به دسترسی به آدرسهایی خارج از یک سگمنت حافظه باشد اشاره‌گرهای معمولی که دو بایت از آدرس را در خود نگهداری می‌کنند جوابگوی نیاز ما نیستند. یک راه حل این است که از مدل حافظه large به جای مدل حافظه small استفاده گردد. در این مدل حافظه کلیه اشاره‌گرها ۴ بایتی

(۳۲ بیتی) هستند که مشکل آدرس‌دهی را حل می‌کند. اما در این مدل حافظه سرعت اجرای برنامه پایین است. راه حل دیگر این است که ضمن استفاده از مدل حافظه smallاز اشاره‌گرهای دیگری به نام اشاره‌گر far استفاده شود. در کنار اشاره‌گرهای far اشاره‌گرهای دیگری به نامهای near و huge نیز قابل استفاده هستند که در این قسمت مورد بحث قرار خواهند گرفت. لازم به ذکر است که کلمات کلیدی far، near و huge در مورد توابع نیز مورد استفاده قرار می‌گیرند که در این صورت در نحوه فراخوانی و برگشت از توابع تأثیر خواهند گذاشت.

کلمهٔ کلیدی far

کلمه کلیدی far برای تعریف اشاره‌گرهایی که نیاز به ذخیره آدرسهایی بیش از ۲ بایت دارند به کار می‌رود و قبل از نام متغیر اشاره‌گر و بعد از تعیین نوع اصلی ذکر می‌شود. به عنوان مثال، دستور char far *f_pointer; اشاره‌گر f_pointer را از نوع char و به صورت far تعریف می‌کند. اشاره‌گرهای از نوع far آدرسهای سگمنت (۲ بایت) و تفاوت مکان (۲ بایت) را در خود نگهداری می‌کنند.

دو نکته مهم در مورد اشاره‌گرهای far وجود دارند که باید آنها را به خاطر داشت:

۱. انجام محاسبات بر روی اشاره‌گرها فقط قسمت تفاوت مکان را تحت تأثیر قرار می‌دهد. به عنوان مثال، اگر به اشاره‌گری با مقدار 0000:FFFF یک واحد اضافه گردد نتیجهٔ حاصل برابر با 0000:0000 خواهد بود، نه برابر با 0001:FFF0؛ یعنی آدرس سگمنت عوض نخواهد شد.

۲. دو اشاره‌گر از نوع far نباید با یک عملگر رابطه‌ای با یکدیگر مقایسه شوند؛ زیرا فقط قسمتهای تفاوت مکان در مقایسه شرکت می‌کنند و یقیناً نتیجه درستی حاصل نخواهد شد.

برای رفع این دو مشکل، به جای اشاره‌گرهای far می‌توان از اشاره‌گرهای huge استفاده نمود.

کلمه کلیدی huge

کلمه کلیدی huge برای تعریف اشاره‌گرهایی که آدرسهایی بیش از ۱۶ بیت را در خود ذخیره می‌کنند به کار می‌رود. اشاره‌گرهای huge همانند اشاره‌گرهای far هستند، با دو تفاوت زیر:

۱. مقایسه دو اشاره‌گر huge مشکلی را تولید نمی‌کند.

۲. وقتی که عمل محاسباتی بر روی اشاره‌گرهای huge انجام می‌شود آدرس سگمنت نیز همانند آدرس تفاوت مکان تغییر می‌کند.

کلمه کلیدی near

کلمه کلیدی near برای تعریف اشاره‌گرهای معمولی (اشاره‌گرهایی که ۱۶ بیت آدرس را پشتیبانی می‌کنند) به کار می‌رود. اگر خواسته باشیم در مدلهای حافظه medium، large و یا huge از حالت فرضی اشاره‌گرها (آدرس ۳۲ بیتی) صرفنظر شود و از اشاره‌گرهای ۱۶ بیتی استفاده گردد. باید با استفاده از کلمه کلیدی near اشاره‌گرهای موردنظر را ۱۶ بیتی تعریف نمود.

مثال ۱-۱۳

برنامه‌ای که صفحه‌نمایش را با یک کاراکتر پر می‌کند. برای خاتمه برنامه، حرف q را وارد کنید.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int far *p;
  int i;
  char ch;
  clrscr();
  p = (int far *) 0xb8000000;
  while((ch = getch()) != 'q')
      for(i = 0; i < 2000; i++)
          *(p + i) = ch | 0x0700;
  return 0;
}
```

در مثال ۱-۱۳ به نکات زیر توجه داشته باشید:

۱. عدد 0xB8000000 آدرس شروع حافظه صفحه‌نمایش رنگی است. این آدرس در صفحه‌نمایش تک رنگ برابر با 0xB0000000 است).

۲. عدد ۲۰۰۰ مشخص کننده میزان حافظه صفحه‌نمایش است (۲۰۰۰ = ۸۰ × ۲۵).

۳. (farptr+addr) محلی از صفحه‌نمایش را مشخص می‌کند که کاراکتر موردنظر باید در آنجا نوشته شود.

۴. همان‌طور که می‌دانید هر کاراکتر برای ظاهر شدن در صفحه‌نمایش دو بایت از حافظه نمایش را اشغال می‌کند که یک بایت برای خود کاراکتر و بایت دیگر، صفت کاراکتر را مشخص می‌کند (صفات کاراکترها در ادامه بحث می‌شود). عدد 0x07 صفت نرمال را مشخص می‌کند که اگر یک بایت به سمت چپ شیفت داده شود عدد 0x0700 حاصل می‌شود و سپس با کاراکتر موردنظر، بیت به بیت OR می‌شود تا ضمن نوشتن کاراکتر موردنظر در صفحه‌نمایش، بایت صفت آن نیز پر شود.

۵. دستور دیگری که باید توضیح داده شود، دستور farptr = (int far *) oxB8000000; است.

عدد 0xB8000000 یک عدد صحیح بزرگ (long int) است ولی متغیر farptr یک اشاره‌گر far است، جمله داخل پرانتز به کامپایلر می‌گوید که عدد 0xB8000000 را به صورت یک اشاره‌گر far از نوع int در نظر بگیرد. در غیر این صورت از طرف کامپایلر پیام خطایی صادر خواهد شد.

برنامهٔ مثال ۱-۱۳ را با استفاده از سطر و ستونهای صفحه‌نمایش نیز می‌توان نوشت که در این صورت برنامهٔ مثال ۲-۱۳ حاصل می‌شود.

مثال ۲-۱۳

برنامه‌ای که کاراکتری را از ورودی خوانده صفحه‌نمایش را با آن کاراکتر پر می‌کند. برای خروج از برنامه حرف q را وارد کنید.

```
#include <conio.h>
#define ROWMAX  25
#define COLMAX  80
void main()
{
   int far *ptr ;
   int col, row ;
   char ch;
   clrscr();
   ptr = (int far *) 0xb8000000;
   while((ch = getch()) != 'q')
    for(row = 0; row < ROWMAX; row ++)
       for(col = 0; col< COLMAX; col ++)
          *(ptr + row * COLMAX + col) = ch | 0x0700;
}
```

در بورلند C علاوه بر کلمات کلیدی for، near و hung از کلمات کلیدی _cs، _ds، _ss و _es نیز برای آدرس‌دهی استفاده می‌شود. این کلمات کلیدی مشخص می‌کنند که اشاره‌گر موردنظر می‌تواند در انواع سگمنت حضور داشته باشد، به‌عنوان مثال، دستور زیر موجب می‌شود تا اشاره‌گر ptr یک اشاره‌گر ۱۶ بیتی تعریف شود و در سگمنت اضافی قابل استفاده باشد:

```
int _es *ptr ;
```

مثال ۳-۱۳

برنامه‌ای که محتویات ۲۵۶ بایت از حافظه را با شروع از آدرسی که بنا به درخواست برنامه مشخص می‌شود، در صفحه‌نمایش ظاهر می‌کند. برنامه می‌تواند محتویات همین محدوده را تغییر دهد.

```
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#define ROWMAX  25
#define COLMAX  80
void display_mem() ;
void change_mem() ;
void main() {
    char ch ;
    for(;;)  {
       clrscr();
       printf("\n enter a  character(d,c,q)|>") ;
       ch = getch() ;
       printf("\n") ;
```

```
        switch(tolower(ch))  {
            case 'd' :
            display_mem() ;
            break ;
            case 'c' :
            change_mem() ;
            break ;
            case 'q' :
            exit(0) ;
        }//end of switch
        printf("\n press a key to continue...");
        getch();
    }//end of for
}
//*********************
void display_mem() {
        register int i ;
        unsigned char ch ;
        unsigned char far *p ;
        char s[80] ;
        /* get bit 20 address */
        printf("begining address in hex:");
        scanf("%p %*c", &p) ;
        printf("%p:", p) ;
        for(i = 1 ; i < 256 ; i++)  {
            ch = *p ;
            /* display in hex */
            printf("%o2x",ch) ;
            p ++ ;
                if(!(i % 16)) {
                    printf("\n") ;
                        if(i != 256)
                            printf(" %p: ", p) ;
                }//end of if
        }//end of for
}
//*****************
void change_mem()
{
    unsigned char far *p ;
    char s[80], value ;
    printf("\n enter address to change");
    printf("(in hex):") ;
    scanf("%p %*c", &p) ;
    printf("\n enter new value(in hex):");
    scanf("%x", &value) ;
    *p = (unsigned char) value ;
}
```

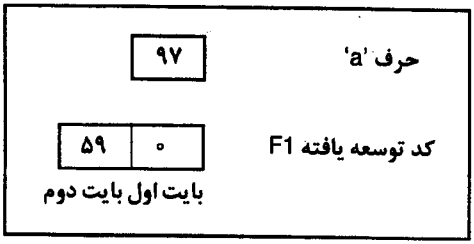
تشخیص کلیدهای صفحه کلید

با فشردن هر یک از کلیدهای صفحه کلید، کدی تولید می‌شود که مشخص می‌کند چه کلیدی از صفحه کلید فشار داده شده است. بعضی از این کدها یک بایتی و بعضی دیگر دوبایتی‌اند. کلیدهایی مثل A، B، X، *، (، $ کد یک بایتی و کلیدهایی مثل F1، F2، HOME کدهای دوبایتی تولید می‌کنند. کلیدهایی که کد دوبایتی تولید می‌کنند بایت اول آن‌ها صفر و بایت دوم بیان‌کننده کد آن کلید است. کد دوبایتی ممکن است در اثر فشردن یک کلید یا ترکیبی از کلیدها ایجاد شود. کلیدهایی که کد دوبایتی تولید می‌کنند در جدول ۱-۱۳ آمده‌اند.

جدول ۱-۱۳ کدهای صفحه کلیدتوسعه یافته.

کلیدها	کد کلیدها (دهدهی)
SHIFT + TAB	۱۵
ALT + Q (W, E, R, T, Y, V, I, O, P)	۱۶ تا ۲۵
ALT + A (S, D, F, G, H, J, K, L)	۳۰ تا ۳۸
ALT + Z (X, C, V, B, N, M)	۴۴ تا ۵۰
F1 - F10	۵۹ تا ۶۸
HOME	۷۱
UP ARROW	۷۲
PGUP	۷۳
LEFT ARROW	۷۵
RIGHT ARROW	۷۷
END	۷۹
DOWN ARROW	۸۰
PGDN	۸۱
INSERT	۸۲
DELETE	۸۳
SHIFT + F1 (F2, ... , F10)	۸۴ تا ۹۳
CTRL + F1 (F2, ... , F10)	۹۴ تا ۱۰۳
ALT + F1 (F2, ... , F10)	۱۰۴ تا ۱۱۳
CTRL + PRTSCR	۱۱۴
CTRL + LEFT ARROW	۱۱۵
CTRL + RIGHT ARROW	۱۱۶
CTRL + END	۱۱۷
CTRL + PGDN	۱۱۸
CTRL + HOME	۱۱۹
ALT + 1 (2, ... , 9, 0, , ')	۱۲۰ تا ۱۳۱
CTRL + PGUP	۱۳۲

به عنوان مثال، شکل ۱-۱۳ وضعیتی را نشان می‌دهد که در آن، کد کلید معمولی (حرف 'a') و کد توسعه یافته (کلید F1) مشخص شده است. کد کلید 'a' برابر با ۹۷ و بایت اول کد کلید F1 برابر با صفر و بایت دوم برابر با ۵۹ است.



شکل ۱-۱۳ کدهای توسعه‌یافته و معمولی.

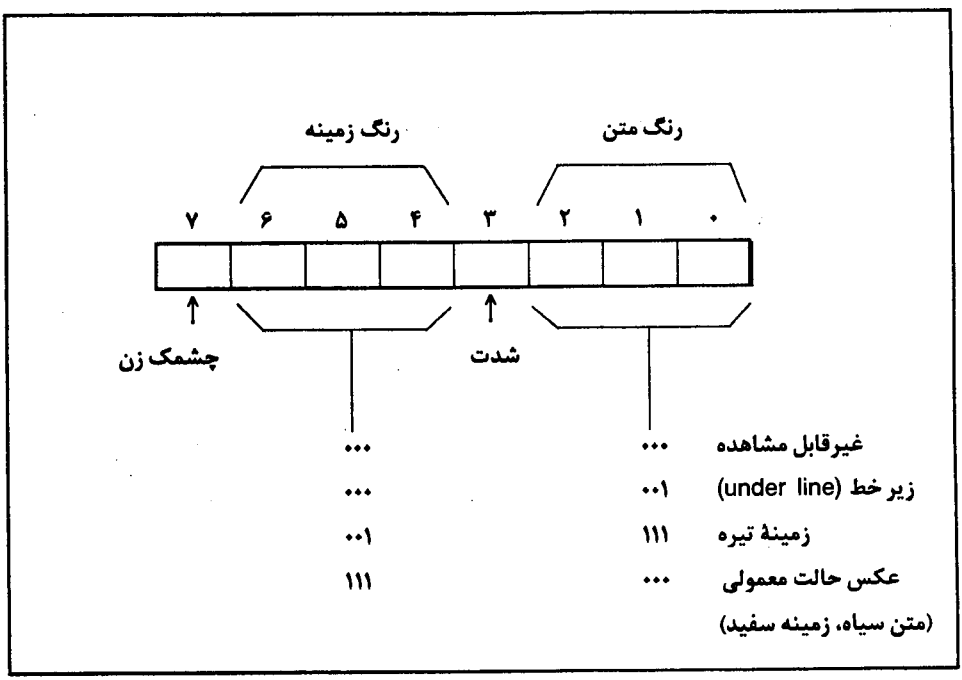
مثال ۴-۱۳

برنامه‌ای که کلیدی را از صفحه کلید خوانده تشخیص می‌دهد که آیا کد توسعه یافته تولید می‌کند یا خیر.

توضیح

کلید توسط تابع ()getch خوانده می‌شود. چنانچه کد کلید خوانده شده برابر با صفر باشد، به معنای این است که این کلید، کد توسعه یافته تولید می‌کند و دستور getch بعدی، آن کد را می‌خواند. برای خاتمه اجرای برنامه، حرف e را فشار دهید.

```
#include <conio.h>
#include <stdio.h>
void main()
{
    char key1, key2 ;
    clrscr();
    while((key1 = getch()) != 'e')
    if(key1 == 0) {
      key2 = getch() ;
      printf("%3d %3d \n", key1, key2);
    }
    else
     printf("%3d\n", key1) ;
}
```



شکل ۲-۱۳ وضعیت بایت صفت.

صفات کاراکتر و تغییر آنها

هر کاراکتری که در صفحه‌نمایش کامپیوتر ظاهر می‌شود، دو بایت از حافظه نمایش را اشغال می‌کند. یک بایت برای ذخیره کد کاراکتر و بایت دیگر برای ذخیره صفت (attribute) کاراکتر است. صفت کاراکتر، نحوه ظاهر شدن کاراکتر را بر روی صفحه‌نمایش مشخص می‌کند: آیا کاراکتر پررنگ نوشته شود یا کم‌رنگ، چشمک‌زن باشد یا خیر و مواردی از این قبیل، شکل ۲-۱۳ وضعیت بایت صفت را نشان می‌دهد.

همان‌طور که در شکل ۲-۱۳ مشاهده می‌شود، بیت شماره ۳ میزان شفافیت کاراکتر را کنترل می‌کند و بیت شماره ۷ مشخص می‌کند که آیا کاراکتر چشمک‌زن باشد یا خیر. دو قسمت دیگر از بایت صفت، ۳ بیتی هستند که بیت‌های صفر تا ۳ رنگ متن و بیت‌های ۴ تا ۶ رنگ زمینه را مشخص می‌کند. در صفحه‌نمایش تک‌رنگ، یکی از رنگ‌های سیاه و سفید یا سبز می‌تواند به عنوان رنگ زمینه انتخاب گردد و رنگ‌های سیاه و سفید می‌توانند به عنوان رنگ متن انتخاب شود.

مثال ۵-۱۳

برنامه‌ای که با استفاده از وقفه‌ها، صفات کاراکترها را تغییر می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
void write_char(int page, int ch, unsigned char attr, int count);
void main() {
  char ch;
  unsigned char attrib = 7, intense = 0, blink = 0;
  while((ch = getch()) != 'e')   {
      clrscr();
      printf("\n type 'n' for normal.");
      printf("\n type 'u' for underline.");
      printf("\n type 'i' for intensified.");
      printf("\n type 'b' for blinking.");
      printf("\n type 'r' for reverse .");
      printf("\n type 'e' for end .");
      printf("\n\t");
      switch(tolower(ch)) {
         case 'n' :  attrib = 7 ;  break ;
         case 'u' :  attrib = 1 ;  break ;
         case 'i' :
               intense = (intense == 1) ? 0 : 1;
               attrib |= (intense << 3);
               break ;
         case 'b' :
               blink  = (blink == 1) ? 0 : 1 ;
               attrib |= (blink << 7);
```

```
                break ;
            case 'r' :
                attrib = 112 ;
                attrib |= (intense << 3);
                attrib |= (blink << 7);
                break ;
        }
        write_char(0, ch, attrib, 5);
    }
}
//******************
void write_char(int page, int ch, unsigned char attr, int count)
{
        union REGS in,out ;
        in.h.ah = 9 ;
        in.h.al = ch ;
        in.h.bh = page ;
        in.h.bl = attr ;
        in.h.ch = 0 ;
        in.h.cl = count ;
        int86(0x10, &in, &out) ;
}
```

عملکرد برنامه

با اجرای برنامه منویی با ۶گزینه ظاهر می‌شود. گزینه اول صفت نرمال، گزینه دوم صفت زیرخط، گزینه سوم شدت، گزینه چهارم صفت چشمک‌زن و گزینه پنجم عکس حالت معمولی را تعیین می‌کند. کاراکتر e برنامه را خاتمه می‌دهد.

type 'n' for normal.
type 'u' for underline.
type 'i' for intensified.
type 'b' for blinking.
type 'r' for reverse.
type 'e' for end.

مثال ۶-۱۳

برنامه‌ای که واژه‌پرداز ساده‌ای را پیاده‌سازی می‌کند. در این برنامه توابعی برای اخذ رشتهٔ فارسی از صفحه‌کلید طراحی شده‌اند. دانشجو می‌تواند با استفاده از این برنامه، توابعی فارسی برای دریافت اطلاعات رشته‌ای در C بنویسد.

```
#include  <dos.h>
#include  <stdio.h>
#include  <conio.h>
#include  <stdlib.h>
#include  <string.h>
#define   COMAX 80
#define   R_ARRO  77
#define   L_ARRO  75
#define   U_ARRO  72
#define   F10     68
#define   D_ARRO  80
#define   BK_SPC  8
#define   FALSE   0
#define   TRUE    1
#define   VIDEO   0x10
#define   DEL     83
#define   HOME    71
#define   END     79
#define   INS     82
void getstr(int x,int y,char str[], int length,int color);
void cursor(int x,int y);
void printstr(int x,int y,char str[], int length,int color);
void backspace(char str[],int length,int j);
void del(char str[],int length,int j);
void insert(char str[],int length , int j);
char *s ;
char str[20] ;
int main()
  {
    int x=12,y=12 ;
    clrscr() ;
    getstr(x,y,str,20,32) ;
    return 0;
  }
//***************
void getstr(int x, int y, char *str, int length, int color)
   {
     char str1[200], str2[200];
     char *s2=":\'\"hHfFQqWwEeRr";
     char *s3="TtYyUuIiOoPp{[]}|";
     char *s4="\\ZzXxCcVvbBnNmM,.";
     char *s5="?    1234567890^&-";
     char *f2="قق ثث صص ضض ب با ا گ گ ک ک";
     char *f3="چچ جج حح خخ هه ع ع غ غ ف ف";
     char *f4="وو هز لا و ذ ر ع ژ غ ژ ی ط ظ پ پ";
     char *f5=".؟آ°۰۱۲۳۴۵۶۷۸۹+ـعـغـ-";
     char ch ;
     int col, col1, insrt = 0, j = 0 , i , linput = 0;
```

```
int latin = FALSE, llatin = 0, k = 0;
strcpy(str1, "AaSsdDgGJjkKlL;");
strcat(str1, s2) ;
strcat(str1, s3);
strcat(str1, s4);
strcat(str1, s5);
strcpy(str2, "کم من نتت ل لی یسس شش");
strcat(str2, f2) ;
strcat(str2, f3);
strcat(str2, f4);
strcat(str2, f5);
for(i = 0 ; i < length ; i++)
    str[i] = '\0' ;
col = y ;
y = 80 - y ;
col1 = y;
cursor(x, y) ;
printstr(x, y, str, length, 31) ;
while( (ch = getch()) !='\r' && linput < length)
{
   if(ch==0)  {
      ch = getch();
      switch(ch)   {
         case  R_ARRO: if(y -j < col1)  --j ;
               if(latin == TRUE) k++ ; break;
         case  L_ARRO: if(j < linput) ++j ;
               if(latin == TRUE) k-- ; break;
         case  DEL :
               if(j <= linput)  {
                  del(str, length, j) ;
                  printstr(x, y, str, length, 31);
                  -- linput;
   }
   break ;
case  HOME  :j = 0; break ;
case  END   :j = linput ;break ;
case  INS   :insrt = (insrt == 1) ? 0 : 1 ;
   break;
case  F10   :
    if(latin == TRUE) {
      j = llatin + j + k;
      insrt = 0;
      llatin = 0 ;
      latin = FALSE ;
      k = 0;
    }
    else  {
        latin = TRUE ;
```

```
            insrt = 1;
            llatin=0;
          }
          break ;
      }//end of switch
    } //end of if
    else
      switch(ch) {
         case BK_SPC:
             if(j > 0)  {
                backspace(str, length, j - 1);
                linput-- ;
                j --;
                printstr(x,y,str,length,31);
             }
             break;
         default:
         if (latin == FALSE) {
            i = 0;
            s = str1;
            while(*s != ch && i < 84) {
                 s++;
                 i++;
         }
         if(i < 84) {
           ch = str2[i] ;
              if(insrt == 1)
                 insert(str, length, j);
                 str[j] = ch ;
                 j++ ;
                 linput++ ;
                 if(j < linput && insrt == 0)
                        linput -- ;
                 printstr(x, y, str, length, 31);
              }//end of if
         }//end of if
         else  {
              insert(str, length, j);
              str[j] = ch ;
              linput++ ;
              printstr(x, y, str, length, 31);
              llatin++ ;
         } //end of else
      }//end of switch
      cursor(x, y - j) ;
    }//end of while
    str = strrev(str) ;
}
```

```
//***************************
void cursor(int x, int y)
{
    union  REGS  regs;
    regs.h.ah = 2;
    regs.h.dl = y;
    regs.h.dh = x;
    regs.h.bh = 0;
    int86(VIDEO, &regs, &regs);
}
//*****************
void printstr(int x,int y,char str[], int length,int color)
   {
     char far *crt ;
     int j;
     crt = (char far *)0xb8000000l;
     for(j = 0 ;j < length ;j++)  {
         *(crt + y * 2 + x * 160) = str[j] ;
         *(crt + y * 2 + x * 160 + 1) = color;
         y -- ;
     }
   }
//*********************
void backspace(char str[],int length,int j)
  {
    int i ;
    for(i=j;i<length-1;i++)
        str[i]=str[i+1];
  }
//******************
void del(char str[],int length,int j)
  {
    int i ;
    for(i=j ;i<length-1;i++)
        str[i]=str[i+1] ;
    str[i+1]=' ';
   }
//*************************
void insert(char str[],int length , int j)
 {
     int i ;
     for(i=length ; i>=j ; i--)
         str[i]=str[i-1];
 }
```

رمزگذاری و فشرده‌سازی متن‌ها

در این فصل دو موضوع مهم و جالب دیگر را که معمولاً در کامپیوتر مورد استفاده قرار می‌گیرند، مورد بررسی قرار می‌دهیم. این دو موضوع عبارت‌انداز: **رمزگذاری** و **فشرده‌سازی**.

رمزگذاری به دو دلیل مهم است: دلیل اول اینکه موجب امنیت و سرّی شدن اطلاعات می‌شود و دلیل دوم اینکه در انتقال داده‌ها از نقطه‌ای به نقطهٔ دیگر، موجب می‌شود تا اطلاعات در ایستگاههای مختلف در آمان باشند.

فشرده‌سازی داده‌ها نیز در کاهش حجم اطلاعات مفید است و موجب صرفه‌جویی در حافظه و هزینه‌ها می‌شود. به عنوان مثال، اگر بخواهیم حجم زیادی از اطلاعات را از طریق خط تلفن از نقطه‌ای به نقطهٔ دیگر انتقال دهیم، فشرده‌سازی موجب کاهش حجم داده‌ها و در نتیجه کاهش هزینه انتقال می‌شود.

انواع رمزگذاری

روشهای رمزگذاری متعددی وجود دارد که دو نوع آن عبارت‌انداز **رمزگذاری جانشینی** (substitution) و **رمزگذاری جابجایی** (transposition). در رمزگذاری جانشینی، یک کاراکتر به جای کاراکتر دیگر قرار می‌گیرد ولی ترتیب کاراکترها حفظ می‌گردد. در رمزگذاری جابجایی، کاراکترها با قواعد خاصی جابجا می‌شوند. هر کدام از این روشها می‌توانند به طور ساده یا پیچیده انجام شوند. ترکیبی از دو روش نیز امکان‌پذیر است، در کامپیوترهای دیجیتال، نوع رمزگذاری به نام **دستکاری بیت‌ها** مورد استفاده قرار می‌گیرد که در آن، نمایش کامپیوتری داده‌ها با الگوریتم خاصی تغییر می‌کند.

هر سه روش رمزگذاری ممکن‌است از کلیدی استفاده کنند. **کلید**، رشته‌ای از کاراکترهاست که برای رمزگشایی پیام لازم است. کلید باید مشخص و واضح باشد و الگوریتم رمزگذاری نیز باید معین باشد. متن بدون رمز را **متن ساده** و متنی که به رمز تبدیل شده است را **متن رمزی** گویند.

رمزگذاری جانشینی

ساده‌ترین روش رمزگذاری جانشینی این است که تمام کاراکترهای متن، از کاراکتر خاصی تفریق شود. به عنوان مثال، متن ساده زیر را در نظر بگیرید:

متن ساده = **abcdefghijklmnopqrstuvwxyz**

اگر کد کاراکتر c را از هر یک از کاراکترهای این متن کم کنیم، متن زیر حاصل می‌شود:

متن رمزی = defghijklmnopqrstuvwxyzabc

در روش دیگر می‌توان به جای کاراکترهای متن، کاراکترهای دیگری را قرار داد، به عنوان مثال، متن ساده زیر را در نظر بگیرید:

متن ساده = meet me at sunset

این متن را می‌توان به صورت زیر به رمز تبدیل کرد (کاراکترهای جایگزین شده را حدس بزنید) :

متن رمزی = phhw ph dw vxqvhw

مثال ۱-۱۴

برنامه‌ای که متن ساده‌ای را به متن رمزی تبدیل می‌کند. در این برنامه، باید نام فایلِ حاوی متن ساده، فایلی که متن رمزی باید در آنجا قرار گیرد، عملی که باید انجام شود (رمزگذاری یا رمزگشایی)، و کاراکتری که رمزگذاری بر اساس آن انجام می‌شود، به عنوان آرگومان به برنامه داده شوند. به عملکرد برنامه توجه کنید. زیربرنامه code متن را رمزی و زیربرنامه decode متن را رمزگشایی می‌کند.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output, char start);
void decode(char *input, char *output, char start);
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 5) {
      printf("usage:input output encode/decode offset\n");
      getch();
      exit(0);
  }
  if(!isalpha(*argv[4])) {
     printf("start leter must be alphabetical character\n");
     exit(0);
   }
   if (toupper(*argv[3]) == 'E')
       code(argv[1], argv[2], *argv[4]);
   else
       decode(argv[1], argv[2], *argv[4]);
   return 0;
}
 //**********************
void code(char *input, char *output, char start)
{
   int ch;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
```

```
        exit(0);
    }
    if((fp2 = fopen(output, "wt")) == 0) {
        printf("cannot open output file.\n");
        exit(0);
    }
    start = tolower(start);
    start = start - 'a';
    ch = getc(fp1);
    while(!feof(fp1)) {
        ch = tolower(ch);
        if(isalpha(ch)) {
            ch += start;
            if(ch > 'z')
                ch -= 26;
        }
        putc(ch, fp2);
        ch = getc(fp1);
    }
    fcloseall();
}//end of code
//**********************
void decode(char *input, char *output, char start)
{
    int ch;
    FILE *fp1, *fp2;
    if((fp1 = fopen(input, "r")) == 0) {
        printf("cannot open input file.\n");
        exit(0);
    }
    if((fp2 = fopen(output, "w")) == 0) {
        printf("cannot open output file.\n");
        exit(0);
    }
    start = tolower(start);
    start = start - 'a';
    ch = tolower(getc(fp1));
    while(!feof(fp1)) {
        if(isalpha(ch)) {
            ch -= start;
            if( ch < 'a')
                ch += 26;
        }
        putc(ch, fp2);
        ch = tolower(getc(fp1));
    }
    fcloseall();
}
```

عملکرد برنامه

فایلی به نام meet با محتویات زیر تشکیل دادم :

```
meet me at sunset.
you can see me in garden.
```

سپس آرگومانهای برنامه را به صورت زیر انتخاب کردم:

```
meet outp  encode c
```

پس از اجرای برنامه محتویات فایل outp به صورت زیر است :

```
oggv og cv uwpugv.
aqw ecp ugg og kp ictfgp.
```

برای رمزگشایی فایل outp برنامه را با آرگومانهای زیر اجراکنید:

```
outp out dencode c
```

این بار محتویات فایل out مانند فایل meet خواهد بود.

اگر بتوان کاراکتری را که به عنوان کلید رمز به کار می‌رود، به طور تصادفی انتخاب کرد، متن رمزی پیچیده بوده، رمزگشایی آن دشوارتر است. در این خصوص، به تعداد !۲۶ می‌توان ترتیب کاراکترها را تعیین کرد. اگر فضای خالی (space) نیز حساب شود، این حالت به !۲۷ تبدیل می‌شود. متن ساده زیر را در نظر بگیرید:

متن ساده = meet me at sunset

اگر این متن را با کلیدی که به طور تصادفی انتخاب می‌شود رمزگذاری کنید، متن زیر حاصل می‌شود که رمزگشایی این متن، دشوارتر است.

متن رمزی = tssjptspqjpumgusj

مثال ۲-۱۴

برنامه‌ای که متن ساده‌ای را به طریق رمزگذاری جانشینی، به متن رمزی تبدیل می‌کند. در این روش، کلید رمز به طور تصادفی انتخاب می‌شود. برای اجرای برنامه، فایل ورودی، فایل خروجی و نوع عمل (رمزگذاری = code و رمزگشایی = decode) باید به عنوان آرگومان وارد شوند.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output);
void decode(char *input, char *output);
int find(char *s, char ch);
char sub[28] = "qazwsxedcrfvtgbyhnujm ikolp";
char alphabet[28] = "abcdefghijklmnopqrstuvwxyz ";
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 4) {
      printf("usage:input output encode/decode\n");
```

```
        getch();
        exit(0);
    }
    if (toupper(*argv[3]) == 'E')
        code(argv[1], argv[2]);
    else
        decode(argv[1], argv[2]);
    return 0;
}
//**********************
void code(char *input, char *output)
{
    int ch;
    FILE *fp1, *fp2;
    if((fp1 = fopen(input, "r")) == 0) {
      printf("cannot open input file.\n");
      exit(0);
    }
    if((fp2 = fopen(output, "wt")) == 0) {
      printf("cannot open output file.\n");
      exit(0);
    }
    ch = getc(fp1);
    while(!feof(fp1)) {
      ch = tolower(ch);
      if(isalpha(ch) || ch == ' ')
      ch = sub[find(alphabet, ch)];
      putc(ch, fp2);
      ch = getc(fp1);
    }
    fcloseall();
}//end of code
//**********************
void decode(char *input, char *output)
{
    int ch;
    FILE *fp1, *fp2;
    if((fp1 = fopen(input, "r")) == 0) {
      printf("cannot open input file.\n");
      exit(0);
    }
    if((fp2 = fopen(output, "w")) == 0) {
      printf("cannot open output file.\n");
```

```
        exit(0);
    }
    ch = tolower(getc(fp1));
    while(!feof(fp1)) {
        if(isalpha(ch) || ch == ' ')
            ch = alphabet[find(sub, ch)];
        putc(ch, fp2);
        ch = tolower(getc(fp1));
    }
    fcloseall();
}
//*******************
int find(char *s, char ch)
{
    register int t;
    for(t = 0; t < 28; t++)
        if(ch == s[t])
    return t;
}
```

عملکرد برنامه

فایلی به نام meet با محتویات زیر تشکیل دادم :

```
meet me at sunset.
you can see me in garden.
```

سپس آرگومانهای برنامه را به صورت زیر انتخاب کردم:

```
meet  outp   encode
```

پس از اجرای برنامه محتویات فایل outp به صورت زیر است :

```
tssjptspqjpumgusj.
obmpzqgpussptspcgpeqnwsg.
```

برای رمزگشایی فایل outp برنامه را با آرگومانهای زیر اجراکنید:

```
outp   out   dencode
```

این بار محتویات فایل out مانند فایل meet خواهد بود.

متنهایی که به وسیله مثالهای ۱-۱۴ و ۲-۱۴ رمزگذاری شدند، به راحتی قابل کشف هستند (شیوه کشف رمز در ادامه مورد بررسی قرار می‌گیرد). یک روش برای پیچیده‌تر کردن رمزگذاری جانشینی این است که از **جانشینی چندگانه** استفاده شود. در این روش، حروف متن ساده لازم نیست دقیقاً با حروف متن رمزی یکسان باشند. این کار را می‌توان با واردکردن یک کاراکتر (فیلد) تصادفی دیگر، و ترکیب آن با فیلد اولیه، انجام داد. به عنوان مثال، متن ساده زیر را در نظر بگیرید:

متن ساده = meet me at sunset

این متن با روش رمزگذاری جانشینی چندگانه به صورت زیر رمزگذاری می‌شود:

متن رمزی = polvytrewqasdfghjklmnbvcxz

مثال ۳-۱۴

برنامه‌ای که رمزگذاری جانشینی چندگانه را پیاده‌سازی می‌کند. این برنامه فقط با کاراکترهای (a تا z) یا (A تا Z) کار می‌کند. برای اجرای برنامه باید نام دو فایل را با encode برای رمزگذاری و decode برای رمزگشایی وارد کنید (عملکرد مثال ۲-۱۴ را ببینید).

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output);
void decode(char *input, char *output);
int find(char *s, char ch);
int index(char ch);
char sub[28] = "qazwsxedcrfvtgbyhnujm ikolp";
char sub2[28] = "poi uytrewqasdfghjklmnbvcxz";
char alphabet[28] = "abcdefghijklmnopqrstuvwxyz ";
char count[27];
int main(int argc, char *argv[])
{
  register int t;
  for (t = 0; t < 27; t ++)
      count[t] = 0;
  clrscr();
  if (argc != 4) {
      printf("usage:input output encode/decode\n");
      getch();
      exit(0);
  }
  if (toupper(*argv[3]) == 'E')
       code(argv[1], argv[2]);
   else
       decode(argv[1], argv[2]);
   return 0;
}
//**********************
void code(char *input, char *output)
{
   int ch, change, t;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "wt")) == 0) {
```

```
    printf("cannot open output file.\n");
    exit(0);
  }
  change = 1;
  ch = getc(fp1);
  while(!feof(fp1)) {
    ch = tolower(ch);
    t = index(ch);
    count[t] ++ ;
   if(isalpha(ch) || ch == ' ')
       if(change)
          ch = sub[find(alphabet, ch)];
       else
          ch = sub2[find(alphabet, ch)];
       putc(ch, fp2);
       if(count[t] == 2) {
           change = !change;
       count[t] = 0;
   }
   ch = getc(fp1);
  }//end of while
  fcloseall();
}//end of code
//**********************
void decode(char *input, char *output)
{
   int ch, change;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "w")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   change = 1;
   ch = tolower(getc(fp1));
   while(!feof(fp1)) {
       if(isalpha(ch) || ch == ' ')
           if(change) {
               ch = alphabet[find(sub, ch)];
               count[index(ch)]++;
```

```
            }
            else {
                ch = alphabet[find(sub2, ch)];
                count[index(ch)]++;
            }//end of else
            putc(ch, fp2);
            if(count[index(ch)] == 2) {
                change = !change;
                count[index(ch)] = 0;
            }
            ch = tolower(getc(fp1));
    }
    fcloseall();
}
//******************
int find(char *s, char ch)
{
    register int t;
    for(t = 0; t < 28; t++)
        if(ch == s[t])
            return t;
}
//******************
int index(char ch)
{
    ch = tolower(ch);
    if(isalpha(ch))
        return ch - 'a';
    else
        return 26;
}
```

رمزگذاری جابجایی

در رمزگذاری جابجایی، می‌توان پیام را به روش خاصی در آرایه‌ای قرار داد و سپس به روش دیگری آن را نوشت. به عنوان مثال، یونیون (union) زیر را در نظر بگیرید:

```
union          message {
               char s[100] ;
               char s2[10][5] ;
} skytale ;
```

m	e	e	t	
m	e		a	t
	s	0	n	s
e	t	0	0	0
0	0	0	0	0

شکل ۱-۱۴
نمونه‌ای از رمزگذاری جابجایی.

اگر این یونیون ابتدا برابر با تهی (NULL) شود و سپس متن سادهٔ 'meet me at sunset' در skytale قرار گیرد، می‌توان آن را از نظر متغیر s2[20][5] مانند شکل ۱-۱۴ تصور کرد.

اگر متن موجود در جدول را به طور ستونی به خروجی ببریم، متن رمزی زیر حاصل می‌شود (نقطه‌ها نشان دهنده کاراکتر تهی هستند:

mm e...eest...e u...tan... ts...

برای رمزگشایی پیام، ستونها به آرایه skytale.s2 منتقل می‌شوند و سپس آرایه skytale.s به صورت معمولی به خروجی می‌رود.

مثال ۴-۱۴

برنامه‌ای که رمزگذاری جابجایی را پیاده‌سازی می‌کند. برای اجرای برنامه باید نام دو فایل را با encode برای رمزگذاری و decode برای رمزگشایی وارد کنید (عملکرد مثال ۲-۱۴ را ببینید).

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output);
void decode(char *input, char *output);
union message {
      char s[100];
      char s2[20][25];
} skytale;
int main(int argc, char *argv[])
{
  register int t;
  for (t = 0; t < 100; t ++)
      skytale.s[t] = 0;
  clrscr();
  if (argc != 4) {
      printf("usage:input output encode/decode\n");
      getch();
      exit(0);
```

```
  }
  if (toupper(*argv[3]) == 'E')
      code(argv[1], argv[2]);
  else
      decode(argv[1], argv[2]);
  return 0;
}
//*********************
void code(char *input, char *output)
{
  int  t, t2;
  FILE *fp1, *fp2;
  if((fp1 = fopen(input, "r")) == 0) {
    printf("cannot open input file.\n");
    exit(0);
  }
  if((fp2 = fopen(output, "wt")) == 0) {
    printf("cannot open output file.\n");
    exit(0);
  }
  for(t = 0; t < 100; t ++) {
      skytale.s[t] = getc(fp1);
      if(feof(fp1)) {
          skytale.s[t] = 0;
          break ;
      }
  }
  for(t = 0; t < 5; t++)
      for(t2 = 0; t2 < 20; t2++)
          putc(skytale.s2[t2][t], fp2);
  fcloseall();
}//end of code
//**********************
void decode(char *input, char *output)
{
  int t, t2;
  FILE *fp1, *fp2;
  if((fp1 = fopen(input, "r")) == 0) {
    printf("cannot open input file.\n");
    exit(0);
  }
  if((fp2 = fopen(output, "w")) == 0) {
    printf("cannot open output file.\n");
```

```
        exit(0);
    }
    for(t = 0; t < 5; t++)
        for(t2 = 0; t2 < 20; t2++) {
            skytale.s2[t2][t] = getc(fp1);
            if(feof(fp1))
                break;
        }
    for(t = 0; t < 100; t++)
        putc(skytale.s[t], fp2);
    fcloseall();
}
```

رمزگذاری به روش دستکاری بیت‌ها

در این روش، بیت‌های تشکیل دهنده کاراکترها در متن ساده، دستکاری می‌شوند. گرچه این روش، شکل دیگری از رمزگذاری جانشینی است، مفاهیم، متدها و گزینه‌ها در این روش، متفاوت است. در روش دستکاری بیت‌ها، الگوی بیت‌های متن ساده با عملگرهای AND، OR، NOT، XOR و متمم یک، تغییر می‌کنند. زبان C بهترین وسیله برای رمزگذاری به روش دستکاری بیت‌ها است. زیرا عملگرهای بیتی | (OR)، & (AND)، ^ (XOR)، ~(متمم یک) در این زبان قابل استفاده‌اند.

ساده‌ترین رمزگذاری دستکاری بیت‌ها، از عملگر ~ استفاده می‌کند. این عملگر موجب می‌شود تا بیت‌های یک به صفر و بیت‌های صفر به یک تبدیل شوند. لذا اگر این عملگر دوبار بر روی بیتی عمل کند، آن بیت به حالت اول برمی‌گردد.

مثال ۵-۱۴

برنامه‌ای که متنی را به روش دستکاری بیت‌ها به رمز تبدیل کرده، آن را از حالت رمز نیز خارج می‌کند. برای اجرای برنامه باید نام دو فایل را با encode برای رمزگذاری و decode برای رمزگشایی وارد کنید. (عملکرد مثال ۲-۱۴ را ببینید).

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output);
void decode(char *input, char *output);
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 4) {
      printf("usage:input output encode/decode\n");
      getch();
      exit(0);
```

```
    }
    if (toupper(*argv[3]) == 'E')
        code(argv[1], argv[2]);
    else
        decode(argv[1], argv[2]);
    return 0;
}
//**********************
void code(char *input, char *output)
{
    int  ch;
    FILE *fp1, *fp2;
    if((fp1 = fopen(input, "r")) == 0) {
      printf("cannot open input file.\n");
      exit(0);
    }
    if((fp2 = fopen(output, "wt")) == 0) {
      printf("cannot open output file.\n");
      exit(0);
    }
    ch = getc(fp1);
    while(!feof(fp1)) {
        ch = ~ch;
        if (ch == EOF)
            ch ++;
        putc(ch, fp2);
        ch = getc(fp1);
    }
    fcloseall();
}//end of code
//**********************
void decode(char *input, char *output)
{
    int ch;
    FILE *fp1, *fp2;
    if((fp1 = fopen(input, "r")) == 0) {
      printf("cannot open input file.\n");
      exit(0);
    }
    if((fp2 = fopen(output, "w")) == 0) {
      printf("cannot open output file.\n");
      exit(0);
    }
```

```
    ch = getc(fp1);
    while(!feof(fp1)) {
        ch = ~ch;
        if (ch == EOF)
            ch --;
        putc(ch, fp2);
        ch = getc(fp1);
    }
    fcloseall();
}
```

رمزگذاری بیتی با استفاده از متمم یک، دو اشکالِ عمده دارد: **۱.** برای رمزگذاری از کلیدی استفاده نمی‌شود، لذا هرکس به برنامه دسترسی داشته باشد می‌تواند آن را رمزگذاری یا رمزگشایی کند. **۲.** این روش، توسط برنامه‌نویسان مجرب به زودی قابل کشف است.

روش دیگر رمزگذاری بیتی، استفاده از عملگر XOR است. همانطور که می‌دانید، نتیجه این عملگر وقتی یک است که یکی از بیت‌ها یک و دیگری صفر باشد. بدین‌ترتیب، XOR دارای این ویژگی است: اگر بایتی را با بایت دیگری به نام **کلید**، XOR کنید و سپس نتیجه کار را دوباره با آن کلید XOR نمایید، بایت اولیه حاصل می‌شود. استفاده از XOR، دو مشکل مربوط به متمم یک را از بین می‌برد.

مثال ۶-۱۴

برنامه‌ای که متنی را به کمک عملگر XOR (^ در C)، به رمز تبدیل می‌کند و سپس آن را رمزگشایی می‌نماید. برای اجرای برنامه باید نام دو فایل را با encode برای رمزگذاری و decode برای رمزگشایی وارد کنید. (عملکرد مثال ۲-۱۴ را ببینید).

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void code(char *input, char *output, char key);
void decode(char *input, char *output, char key);
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 5) {
      printf("usage:input output encode/decode key\n");
      getch();
      exit(0);
  }
  if (toupper(*argv[3]) == 'E')
       code(argv[1], argv[2], *argv[4]);
   else
       decode(argv[1], argv[2], *argv[4]);
   return 0;
}
//**********************
```

```
void code(char *input, char *output, char key)
{
   int  ch;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "wt")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   ch = getc(fp1);
   while(!feof(fp1)) {
       ch = ch ^ key;
       if (ch == EOF)
          ch ++;
       putc(ch, fp2);
       ch = getc(fp1);
   }
   fcloseall();
}//end of code
 //**********************
void decode(char *input, char *output, char key)
{
   int ch;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "w")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   ch = getc(fp1);
   while(!feof(fp1)) {
       ch = ch ^ key;
       if (ch == EOF + 1)
           ch --;
       putc(ch, fp2);
       ch = getc(fp1);
   }
   fcloseall();
}
```

فشرده‌سازی داده‌ها

تکنیکهای فشرده‌سازی داده‌ها، اطلاعات را در فضای کمتری از حافظه ذخیره می‌کنند. فشرده‌سازی داده‌ها برای کاهش حجم داده‌ها، و کاهش هزینه انتقال (بخصوص از طریق خط تلفن) و برقراری امنیت مورد استفاده قرار می‌گیرد. گرچه روشهای زیادی برای فشرده‌سازی داده‌ها وجود دارد، در این فصل دو روش مورد بررسی قرار می‌گیرد. روش اول **فشرده‌سازی بیتی** است که در آن، در یک بایت، بیش از یک کاراکتر ذخیره می‌شود. روش دوم، **حذف کاراکتر** است که در آن، کاراکترهایی از داده‌ها، از فایل حذف می‌شوند.

ذخیره ۸کاراکتر در ۷ بایت (فشرده‌سازی بیتی)

در کامپیوترها، طول بایتها توان زوجی از ۲ است که داده‌ها را ذخیره می‌کند. حروف بزرگ و کوچک و حروف نشانه گذاریها فقط به ۶۳کد مختلف نیاز دارند. برای ذخیره این کاراکترها، ۶ بیت کافی است. اما، اغلب کامپیوترها از ۸بیت استفاده می‌کنند، لذا در فایلهای متنی ساده، ۲۵٪ از فضای فایل به هدر می‌رود. بنابراین، اگر دو بیتِ اضافی هر بایت، برای فشرده‌سازی به کار رود، هر سه کاراکتر در ۲ بایت ذخیره می‌شود. تنها مشکل این کار، تشخیص کدهای اسکی است، زیرا بیش از ۶۳کد کاراکتر اسکی وجود دارد، و حروف کوچک و بزرگ در بین این کاراکترها هستند. یعنی، بعضی از کاراکترها، به ۷بیت نیاز دارند. می‌توان از نمایش غیراسکی نیز استفاده کرد، اما این کار، منطقی نیست. لذا می‌توان کاراکترها را در ۷ بیت ذخیره کرد. در نتیجه یک بیت از ۸بیت هر بایت، اضافی خواهد بود و بیت هشتم هر بایت برابر با صفر است. بیت هشتم هر ۷بایت از داده‌ها را می‌توان برای ذخیره کاراکتر هشتم به کار برد. اما باید به این نکته توجه داشته باشید که بسیاری از کامپیوترها، از جمله کامپیوترهای آی.بی.ام ، برای نمایش کاراکترهای ویژه، یا کاراکترهای گرافیکی از ۸ بیت استفاده می‌کنند. بعضی از واژه‌پردازها نیز برای دستورالعملهای پردازش متن، از بیت هشتم استفاده می‌کنند. لذا، استفاده از این نوع فشرده‌سازی، فقط در فایلهای دقیقاً اسکی، که از بیت هشتم برای کار دیگری استفاده نمی‌کنند مفید است. برای پی بردن به این تکنیک، ۸کاراکتر زیر را در نظر بگیرید:

بایت ۱	0	1	1	1	0	1	0	1
بایت ۲	0	1	1	1	1	1	0	1
بایت ۳	0	0	1	0	0	0	1	1
بایت ۴	0	1	0	1	0	1	1	0
بایت ۵	0	0	0	1	0	0	0	0
بایت ۶	0	1	1	0	1	1	0	1
بایت ۷	0	0	1	0	1	0	1	0
بایت ۸	0	1	1	1	1	0	0	1

همانطور که می‌بینید، بیت هشتم همیشه صفر است. در غیر از مواردی که بیت هشتم برای کنترل توازن به کار می‌رود، این امر صادق است. ساده‌ترین روش فشرده‌سازی ۸کاراکتر در ۷بایت، ذخیره ۷بیت باارزش بایت ۱ در ۷بیتِ بلااستفاده از بایتهای ۲ تا ۸است. در نتیجه، ۷بایت باقیمانده به صورت زیر درمی‌آید :

بایت ۱، از بالا به پایین

بایت ۲	1	1	1	1	1	1	0	1
بایت ۳	1	0	1	0	0	0	1	1
بایت ۴	1	1	0	1	0	1	1	0
بایت ۵	0	0	0	1	0	0	0	0
بایت ۶	1	1	1	0	1	1	0	1
بایت ۷	0	0	1	0	1	0	1	0
بایت ۸	1	1	1	1	1	0	0	1

برای بازیابی بایت ۱، باید این بیت‌ها را از بیت هشتم بایتهای ۲ تا ۸ بازیابی کنید و بیت هشتمِ همهٔ بایتها را برابر با صفر قرار دهید.

این روش فشرده‌سازی، اندازه هر فایل متنی را به $\frac{۱}{۸}$ یا ۱۲/۵ درصد کاهش می‌دهد. در نتیجه، اگر یک فایل بزرگ را از طریق خط تلفن به جای دوردستی انتقال می‌دهید، ۱۲/۵ درصد در هزینه ارسال صرفه‌جویی می‌شود.

مثال ۷-۱۴

برنامه‌ای که فایل متنی را به روشِ ذخیره ۸کاراکتر در ۷ بایت، فشرده‌سازی می‌کند. برای اجرای برنامه نام دو فایل و compress را برای فشرده‌سازی، و decompress برای بازکردن متن فشرده وارد کنید.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void compress(char *input, char *output);
void decompress(char *input, char *output);
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 4) {
      printf("usage:input output  compress/decompress\n");
      getch();
      exit(0);
  }
  if (toupper(*argv[3]) == 'C')
       compress(argv[1], argv[2]);
   else
       decompress(argv[1], argv[2]);
   return 0;
}
//*********************
void compress(char *input, char *output)
```

```
{
   char  ch, ch2, t;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "wt")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   do {
      ch = getc(fp1);
      if(feof(fp1)) break;
      ch = ch << 1;
      for(t = 0; t < 7; ++t) {
         ch2 = getc(fp1);
         if (feof(fp1))
            ch2 = 0;
         ch2 = ch2 & 127;    // turn off top bit
         ch2 = ch2 | ((ch << t) & 128);
         putc(ch2, fp2);
      }
   } while(!feof(fp1));
   fcloseall();
}//end of code
 //**********************
void decompress(char *input, char *output)
{
   unsigned char ch, ch2, t, done;
   char s[7], temp;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "w")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   do {
      ch = 0;
      for(t = 0; t < 7; t++) {
```

```
            temp = getc(fp1);
            if(temp == EOF) break;
               ch2 = temp;
            s[t] = ch2 & 127 ; // turn off top bit
            ch2 = ch2 & 128 ; // turn off all but top bit
            ch2 = ch2 >> t + 1;
            ch = ch | ch2;
         }
         putc(ch, fp2);
         for(t = 0; t < 7 && s[t]; t ++)
         putc(s[t], fp2);
         for(t = 0; t < 7; t ++)  //nul the s sting for next use
               s[t] = 0;
      } while (!feof(fp1));
      fcloseall();
   }
```

همان‌طور که مشاهده می‌کنید، این برنامه کمی پیچیده است، زیرا بیت‌های مختلفی باید شیفت داده شوند. توجه داشته باشید که در C، کاراکترهای علامت‌دار با کاراکترهای بدون علامت، متفاوت هستند. به همین دلیل، تبدیل انواع نیز در برنامه فوق انجام شده است.

فشرده‌سازی از طریق حذف کاراکترها

یکی از روشهای جالب در فشرده‌سازی داده‌ها، کاراکترهای اضافی را از کلمات حذف می‌کند. در این روش، کلمات به طور اختصاری ذخیره می‌شوند. استفاده از کلمات اختصار برای ذخیره‌سازی داده‌ها، مرسوم است. مثلاً "Mr" به جای "Mister" ذخیره می‌گردد. به جای استفاده از اختصار واقعی، روشی که در اینجا استفاده می‌شود، بعضی از حروف را از متن حذف می‌کند. برای این کار، نیاز به الفبای کمینه (minimal alphabet) است. در الفبای کمینه، کاراکترهایی که به ندرت مورد استفاده قرار می‌گیرند حذف می‌شوند. به طوری که موجب ابهام می‌شوند. لذا، هر کاراکتر که در الفبای کمینه وجود ندارد، از هر کلمه‌ای که در آن ظاهر می‌شود، استخراج می‌گردد. اینکه دقیقاً چند کاراکتر در الفبای کمینه وجود داشته باشد، به انتخاب برنامه‌نویس بستگی دارد. به هر حال، در این بخش، از ۱۴ کاراکترِ متداول، از جمله، فضای خالی و کاراکتر خط جدید، استفاده می‌شود.

خودکارسازی فرآیند اختصار، مستلزم این‌است‌که بدانید چه حروفی در الفبا، اغلب مورد استفاده قرار می‌گیرند، به طوری که، بتوانید الفبای کمینه را ایجاد کنید. از نظر تئوری، می‌توانید حروف هر کلمه دیکشنری را بشمارید؛ اما نویسندگان مختلف از ترکیب‌های مختلفی استفاده می‌کنند، لذا نمودار تکراری که براساس تعداد کلمات زبان انگلیسی ایجاد می‌شود، تعداد واقعی تکرار حروف را نشان نمی‌دهد (شمارش کاراکترها نیز زمان زیادی می‌برد). به عنوان یک روش دیگر، می‌توانید تعداد تکرار حروف موجود در یک فایل را بشمارید و آن را به عنوان مبنایی برای الفبای کمینه استفاده کنید. برای این کار می‌توانید از برنامه مثال ۸-۱۴ استفاده کنید.

مثال ۸-۱۴

برنامه‌ای که می‌تواند تعداد تکرار حروف A تا Z، فضای خالی، نقطه و کاما را شمارش کند. برای اجرای برنامه نام فایل متنی را به عنوان آرگومان وارد کنید.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
   FILE *fp1;
   int alphabet[26], t;
   int space = 0, period = 0, comma = 0;
   char ch;
  clrscr();
  if (argc != 2) {
      printf("usage:textfile\n");
      getch();
      exit(0);
  }
   if((fp1 = fopen(argv[1], "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   for(t = 0; t < 26; ++t)
      alphabet[t] = 0;
   ch = getc(fp1);
   while(!feof(fp1)) {
      if(isalpha(ch))
          alphabet[toupper(ch) - 'A'] ++;
      else
        switch(ch) {
            case ' ' :  space ++;  break;
            case '.' :  period ++ ; break;
            case '/' :  comma ++;   break;
        } //end of switch
      ch = getc(fp1);
   }//end of while
   for(t = 0; t < 26; ++t)
      printf("%c: %d \n", 'A' + t, alphabet[t]);
   printf("\n period = %d", period);
   printf("\n space = %d", space);
   printf("\n comma = %d", comma);
   fclose(fp1);
   return 0; }
```

برای فشرده‌سازی داده‌ها، باید حروفی را که کمترین تکرار را داشته‌اند حذف کنید. گرچه نظرات زیادی در خصوص الفبای کمینهٔ کارآمد وجود دارد، ولی حروف متداول، ۸۵٪کاراکترها را تشکیل می‌دهند. ۱۴ حرف متداول و کاراکترها فضای خالی و خط جدید، عبارت‌انداز:

A C D E H I L M N O R S T U <space> <newline>

مثال ۹-۱۴

برنامه‌ای که فایل متنی را خوانده، تمام کاراکترها، به جز ۱۶کاراکتر انتخابی را از فایل حذف می‌کند و نتیجه را در فایل دیگری قرار می‌دهد. برای اجرای برنامه نام فایلهای ورودی و خروجی را به عنوان آرگومان وارد کنید.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void comp2(char *input, char *output);
int is_in(char ch, char *s);
int main(int argc, char *argv[])
{
  clrscr();
  if (argc != 3) {
      printf("usage:input output \n");
      getch();
      exit(0);
  }
  comp2(argv[1], argv[2]);
  return 0;
}
//**********************
void comp2(char *input, char *output)
{
   char ch;
   FILE *fp1, *fp2;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   if((fp2 = fopen(output, "wt")) == 0) {
     printf("cannot open output file.\n");
     exit(0);
   }
   ch = getc(fp1);
   while(!feof(fp1)) {
      if(is_in(toupper(ch), "ACDEJILMNORSTU '\n'")) {
```

```
            if(ch == '\n') putc('\r', fp2);
                putc(ch, fp2);
        }
        ch = getc(fp1);
    } // end of while
    fcloseall();
}
//*****************
int is_in(char ch, char *s)
{
  while(*s) {
      if(ch == *s) return 1;
        s ++ ;
  }
  return 0;
}
```

متنی که بدین ترتیب رمزگذاری می‌شود تقریباً قابل خواندن است، ولی ابهاماتی نیز دارد. اگر با دایره لغات نویسنده آشنا باشید، به راحتی می‌توانید متن فشرده شده را بخوانید و در تبدیل متن ساده به متن رمزی نیز از الفبای کمینه مناسبی استفاده کنید. هر یک از دو روش فشرده‌سازی بیتی و حذف کاراکتر، در رمزگذاری داده‌ها نیز مورد استفاده قرار می‌گیرند. فشرده‌سازی بیتی، رمزگذاری خوبی را ایجاد می‌کند که کشف آن مشکل است.

کشف رمز متنهای رمزی

هر کشف رمز، یک عمل آزمایش و خطاست. با استفاده از آزمایش و خطا، رمزهای بسیاری از متنها را می‌توان پیدا کرد. اما رمزگذاریهای پیچیده را یا نمی‌توان کشف کرد و یا نیازمند تکنیکهای مناسبی هستند که موجود نیست. برای سهولت، رمزگشایی کدهای ساده را بررسی می‌کنیم.

اگر متنی با استفاده از روش جانشینی رمزگذاری شده باشد، می‌توانید تمام ۲۶ کاراکتر را مورد بررسی قرار دهید تا به رمز آن پی ببرید.

مثال ۱۰-۱۴

برنامه‌ای که رمز متنی را که به روش جانشینی رمزگذاری شده است کشف می‌کند.

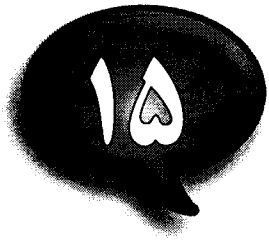
```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void codebreak(char *input);
int main(int argc, char *argv[])
{
  clrscr();
```

```
  if (argc != 2) {
      printf("usage:input \n");
      getch();
      exit(0);
  }
  codebreak(argv[1]);
  return 0;
}
//**********************
void codebreak(char *input)
{
  char ch, s[1000], q[10];
  register int t, t2;
  FILE *fp1;
  if((fp1 = fopen(input, "r")) == 0) {
    printf("cannot open input file.\n");
    exit(0);
  }
  for(t = 0; t < 1000; ++t) {
      s[t] = getc(fp1);
      if (feof(fp1)) break;
  }
  s[t] = '\0';
  fclose(fp1);
  for(t = 0; t< 26; ++t) {
      for(t2 = 0; s[t2]; t2 ++) {
          ch = s[t2];
          if(isalpha(ch)) {
              ch = tolower(ch) + t;
              if(ch > 'z') ch -= 26;
          }
          printf("%c", ch);
      }
      printf("\n");
      printf("decoded ?(y/n):");
      gets(q);
      if(*q == 'n') break;
  }
  printf("\n offset is : %d", t);
}
```

مثال ۱۱-۱۴

برنامه‌ای که رمز متنی را که با کاراکتر تصادفی رمزگذاری شده است کشف می‌کند.

```
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void codebreak(char *input);
int find(char *s, char ch);
char sub[28];
char alphabet[28] = "abcdefghijklmnopqrstuvwxyz ";
int main(int argc, char *argv[]) {
  char s[80];
  clrscr();
  if (argc != 2) {
      printf("usage:input \n");
      getch();
      exit(0);
  }
  do {
      printf("enter test alphabet:");
      gets(sub);
      codebreak(argv[1]);
      printf("\n Is this right? (y/n):");
      gets(s);
  } while (tolower(*s) != 'y');
  return 0;
}
//**********************
void codebreak(char *input)
{
   char ch;
   FILE *fp1;
   if((fp1 = fopen(input, "r")) == 0) {
     printf("cannot open input file.\n");
     exit(0);
   }
   ch = getc(fp1);
   while(!feof(fp1)) {
     if(isalpha(ch) || ch == ' ')
         putch(alphabet[find(sub, ch)]);
          ch = getc(fp1);
   }
   fcloseall();
}
//************************
int find(char *s, char ch) {
  register int t;
  for(t = 0; t < 28; t ++)
      if(ch == s[t])  return t;
}
```



توابع کتابخانه‌ای

در C استاندارد هیچ گونه تابعی در مورد گرافیک و یا سایر توابع در مورد صفحه‌نمایش، پیش‌بینی نشده است. ولی بورلند C دارای چندین تابع کتابخانه‌ای در این زمینه است که در این فصل مورد بحث قرار می‌گیرند. الگوی توابع گرافیکی در فایل graphics.h و الگوی سایر توابع در مورد صفحه‌نمایش که در این قسمت مورد بحث قرار می‌گیرند در فایل conio.h قرار دارند. برای درک توابع گرافیکی و دیگر توابع صفحه‌نمایش، باید مفهوم پنجره مشخص گردد، زیرا بسیاری از توابع با آن سروکار دارند. **پنجره** قسمتی از صفحه‌نمایش است که خروجی برنامه در آنجا قرار می‌گیرد. پنجره می‌تواند به اندازهٔ صفحه‌نمایش (در حالت عادی) و یا کوچکتر از آن باشد. به پنجرهٔ در حالت گرافیکی "محدوده گرافیک" گفته می‌شود.

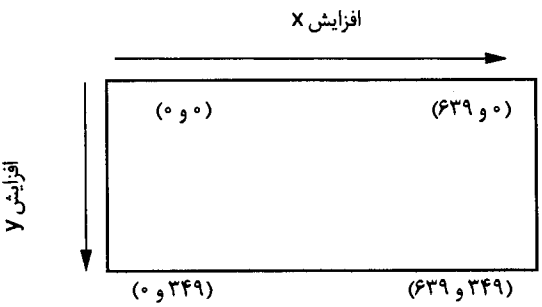
صفحه‌نمایش کامپیوتر به دو صورت می‌تواند مورد استفاده قرار گیرد:

۱. حالت متن (text)

۲. حالت گرافیکی (graphic)

هر یک از حالتهای متن و گرافیک می‌توانند وضعیتهای متفاوتی داشته باشند. همانطور که می‌دانید در ریزکامپیوترها بوردهای گرافیکی مختلفی مورد استفاده قرار می‌گیرند که بعضی از آنها عبارتنداز: monochorome ، CGA ، EGA ، VGA و VGA . هر کدام از بوردهای گرافیکی، وجوه (حالات) مختلفی را برای صفحه‌نمایش فراهم می‌کنند. بعضی از وجوه صفحه‌نمایش مربوط به متن و بعضی دیگر مربوط به گرافیک است. در وجوه گرافیکی امکان ظاهر شدن متن در صفحه‌نمایش وجود دارد ولی در وجوه متن، انجام امور گرافیکی ممکن نیست.

کوچکترین واحد قابل دسترس (آدرس‌دهی) در وجه گرافیکی، **پیکسل** نام دارد. مختصات گوشهٔ بالای سمت چپ صفحه‌نمایش در وجه گرافیکی، (۰ و ۰) است. مختصات هر نقطه از صفحه‌نمایش با دو مقدار x و y مشخص می‌شود که x محور افقی و y محور عمودی است (x , y) (شکل ۱-۱۵).



شکل ۱-۱۵
وضعیت صفحه‌نمایش در یکی از وجوه گرافیکی (۶۴۰×۳۵۰).

جدول ۱-۱۵ رنگهای موجود در جعبه رنگ بورد CGA.

شماره رنگ / جعبه رنگ	۱	۲	۳
۰	سبز	قرمز	زرد
۱	کبود	بنفش	سفید
۲	سبز روشن	قرمز روشن	زرد
۳	کبود روشن	بنفش روشن	سفید

جدول ۲-۱۵ مبدل‌های گرافیک

نام ماکرو	شماره	نام ماکرو	شماره
DETECT	۰	EGAMONO	۵
CGA	۱	IBM8514	۶
MCGA	۲	HERCMONO	۷
EGA	۳	ATT400	۸
EGA64	۴	VGA	۹
		PC3270	۱۰

در وجوه گرافیکی، همانند وجوه متن می‌توان از امکان رنگ‌آمیزی استفاده نمود، که چگونگی این عمل در ادامه این فصل مورد بررسی قرار می‌گیرد. در حالت متن، رنگهای زمینه و متن و در حالت گرافیکی رنگهای زمینه و شکل گرافیکی قابل تغییر می‌باشند. انتخاب رنگ شکل گرافیکی با استفاده از جعبه رنگ انجام می‌شود. به عنوان مثال، در بورد گرافیکی CGA تعداد ۴ جعبه رنگ وجود دارد که از ۰ تا ۳ شماره‌گذاری می‌شوند و شماره رنگ ۰، همان رنگ زمینه می‌باشد (جدول ۱-۱۵). در بوردهای گرافیکی EGA، VGA و SVGA هر جعبه رنگ دارای ۱۶ رنگ است که رنگهای آنها قابل تغییر است. چون ۴ جعبه رنگ وجود دارد، ۶۴ رنگ قابل استفاده است.

توابع گرافیکی

تابع initgraph()

تابع initgraph برای انتقال یک مبدل گرافیک مناسب به حافظه مورد استفاده قرار می‌گیرد. قبل از انجام هر کار گرافیکی، باید یک مبدل گرافیک مناسب به حافظه منتقل شود، زیرا در غیر این صورت هیچیک از توابع گرافیکی عمل نخواهند کرد. تابع initgraph() دارای الگوی زیر است:

```
void far initgraph (int far *driver, int far *mode, char far *path)
```

در این الگو، driver به مبدل گرافیک اشاره می‌کند. فایلهای مبدل گرافیک دارای پسوند BGI می‌باشند. برای معرفی این فایلها لزومی به حفظ کردن اسامی آن نیست بلکه می‌توان از شماره‌ها و ماکروهایی که در فایل graphics.h تعریف شده‌اند استفاده نمود (جدول ۲-۱۵).

ماکروی DETECT موجب می‌شود تا تابع initgraph به طور اتوماتیک، بورد گرافیکی را تشخیص داده آن را در وجه گرافیکی با دقت بالا قرار دهد. مثلاً در بورد گرافیکی CGA، ۶۴۰×۲۰۰ دقت بالا و ۳۲۰×۲۰۰ دقت پایین می‌باشد. mode وجه گرافیکی موردنظر را تعیین می‌کند. مقادیر و ماکروهای معتبر برای mode در جدول ۳-۱۵ آمده‌اند. به عنوان مثال، مجموعه دستورات زیر را در نظر بگیرید:

```
int driver, mode ;
driver = DETECT ;
mode = 0 ;
initgraph(&driver,&mode, " ") ;
```

این مجموعه دستورات موجب می‌شوند تا بورد گرافیکی توسط سیستم تشخیص داده شده و در وجه گرافیکی مناسبی قرار گیرد. چون به جای path رشته تهی قرار گرفته است، برای پیدا کردن مبدل موردنیاز، در مسیر جاری جستجو می‌شود. به عنوان مثال دیگر، مجموعه دستورات زیر را در نظر بگیرید:

```
int driver, mode ;
driver = EGA ;
mode = EGAHI ;
initgraph(&driver,&mode, " ") ;
```

این دستورات، مبدل گرافیک بورد EGA را به حافظه منتقل کرده وجه گرافیکی را EGAHI تعیین می‌کند.

تابع setgraphmode()

تابع setgraphmode() برای تعیین وجه گرافیکی مورد استفاده قرار می‌گیرد. این تابع دارای الگوی زیر است:

void far setgraphmode (int mode)

در الگوی فوق، mode یکی از وجوه گرافیکی معتبر در جدول ۳-۱۵ است. به عنوان مثال، دستور setgraphmode(CGAHI) بورد گرافیکی CGA را در وضعیت دقت بالا قرار می‌دهد.

تابع restorecrtmode()

همانطور که قبلاً مشاهده شد، تابع initgraph() موجب تغییر حالت صفحه‌نمایش می‌شود. تابع restorecrtmode() موجب می‌گردد تا حالت صفحه‌نمایش به حالت قبل از عمل تابع initgraph() برگردد. تابع restorecrtmode() دارای الگوی زیر است:

void far restorecrtmode( )

تابع moveto()

تابع moveto() موجب انتقال موقعیت جاری (current position) به یک نقطهٔ دلخواه می‌شود و دارای الگوی زیر است:

void far moveto (int x, int y)

در الگوی فوق، x و y موقعیت جاری را تعیین می‌کنند. به عنوان مثال، دستور ; moveto (100,100) موجب انتقال موقعیت جاری به محل (100,100) می‌شود.

جدول ۳-۱۵ وجوه گرافیکی معتبر برای تابع initgraph() .

مبدل	وجه گرافیکی (ماکرو)	معادل عددی وجه گرافیکی	دقت
CGA	CGAC0	0	320*200
	CGAC1	1	320*200
	CGA2	2	320*200
	CGAC3	3	320*200
	CGAHI	4	640*200
MCGA	MCGAC0	0	320*200
	MCGAC1	1	320*200
	MCGAC2	2	320*200
	MCGAC3	3	320*200
	MCGAMED	4	640*200
	MCGAHI	5	640*480
EGA	EGALO	1	640*200
	EGAHI	1	640*350
EGA64	EGA64LO	0	640*200
	EGA64HI	1	640*350
EGAMONO	EGAMONOHI	3	640*350
HERC	HERCMONOHI	0	720*384
ATT400	ATT400C0	0	320*200
	ATT400C1	1	320*200
	ATT400C2	2	320*200
	ATT400C3	3	320*200
	ATT400CMED	4	640*200
	ATT400CHI	5	640*400
VGA	VGALO	0	640*200
	VGAMED	1	640*350
	VGAHI	2	640*480
PC3270	PC3270HI	0	720*350
IBM8514	IBM8514LO	0	640*480
	IBM8514HI	1	1024*760

تابع moverel()

تابع moverel() موجب انتقال موقعیت جاری به یک نقطهٔ دلخواه نسبت به موقعیت فعلی می‌شود و دارای الگوی زیر است:

void far moverel (int deltax, int deltay)

در الگوی فوق، deltax و deltay مقدار انتقال موقعیت جاری را از موقعیت فعلی نشان می‌دهند. به عنوان مثال، اگر موقعیت فعلی نقطهٔ (10,10) باشد، پس از اجرای دستور ; moverel (10,20) موقعیت جاری به نقطهٔ (۳۰ و ۲۰) منتقل خواهد شد.

توابع line() ، lineto() و linerel()

تابع line() برای رسم یک خط مورد استفاده قرار می‌گیرد و دارای الگوی زیر است:

void far line (int startx, int starty, int endx, int endy)

تابع line() از نقطهٔ (startx , starty) تا نقطهٔ (endx , endy) خطی را رسم می‌کند و موقعیت جاری را تغییر نمی‌دهد. تابع lineto() خطی را از موقعیت جاری تا یک نقطهٔ دلخواه رسم می‌کند و دارای الگوی زیر است:

void far lineto (int x, int y)

در الگوی فوق x و y انتهای خط را مشخص می‌کنند. تابع lineto() پس از رسم خط، موقعیت جاری را به نقطهٔ (x,y) منتقل می‌کند. تابع linerel() از موقعیت جاری تا نقطه‌ای که مختصات آن نسبت به موقعیت جاری سنجیده می‌شود، خطی را رسم می‌کند و دارای الگوی زیر است:

void far linerel (int deltax, int deltay)

اگر موقعیت جاری، نقطه (x,y) باشد، تابع linerel() از نقطهٔ (x,y) تا نقطهٔ (x+deltax,y+deltay) خطی را رسم می‌کند و موقعیت جاری را نیز به همین نقطه منتقل می‌کند.

مثال ۱-۱۵

برنامه‌ای که چگونگی استفاده از توابع line را نشان می‌دهد:

```
#include <graphics.h>
#include <conio.h>
int main()
{
   int driver, mode ;
   struct palettetype p;
   driver=DETECT;
   mode=0;
    initgraph(&driver,&mode,"");
    line(100,100,200,200);
    lineto(100,50);
```

```
    linerel(30,40);
    getch();
    restorecrtmode();
    return 0;
}
```

تابع getmoderange()

تابع getmoderange() پایین‌ترین و بالاترین وجه گرافیکی یک مبدل گرافیک را مشخص می‌کند. این تابع دارای الگوی زیر است:

void far getgraphmode (int driver, int far *lowermode, int far *himode)

در الگوی فوق، driver مبدلی را مشخص می‌کند که پایین‌ترین و بالاترین وجه گرافیکی آن باید مشخص گردد. این پارامتر می‌تواند یکی از مقادیر (یا ماکروی) جدول ۴-۱۵ را بپذیرد.

جدول ۴-۱۵ مقادیر معتبر پارامتر driver در تابع getgraphrange() .

ماکرو	معادل عددی
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
RESERVED	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

مثال ۲-۱۵

برنامه‌ای که بورد گرافیکی سیستم را تشخیص داده محدوده وجه گرافیکی آن را مشخص می‌کند.

```
#include "graphics.h"
#include "stdio.h"
int main(void) {
   int driver,mode;
   int far *high,*low ;
   driver=DETECT;
   initgraph(&driver,&mode,"") ;
   getmoderange(driver,low,high);
   printf("\nmode range: %d-%d",low,high);
   return 0;
}
```

تابع getgraphmode()

تابع getgraphmode() وجه گرافیکی فعلی صفحه‌نمایش را تشخیص می‌دهد و دارای الگوی زیر است :

int far getgraphmode (void)

این تابع با توجه به بورد گرافیکی، مقادیری را که بیانگر وجه گرافیکی در آن بورد گرافیکی است برمی‌گرداند. به عنوان مثال، اگر بورد گرافیکی CGA باشد و این تابع عدد ۴ را برگرداند، بدین معنی است که وجه گرافیکی فعلی در دقت بالا (۲۰۰×۶۴۰) می‌باشد. برای اطلاعات بیشتر در مورد نتایج این تابع می‌توانید به جدول ۳-۱۵ مراجعه نمایید. به عنوان مثال، دستور printf("\n graphics mode is %d",getgraphmode()); وجه گرافیکی فعلی صفحه‌نمایش را مشخص می‌کند.

تابع setpalette()

تابع setpalette() برای تغییر رنگ سیستم نمایش رنگ در وجه گرافیکی فعلی به کار می‌رود و دارای الگوی زیر است:

void far setpalette (int index, int color)

بورلند C برای استفاده از رنگ از جدولی استفاده می‌کند که رنگهای وجوه گرافیکی در آنجا قرار دارند. مراجعه به آن جدول توسط یک index انجام می‌شود. با استفاده از این index می‌توان رنگهای جدول را تغییر داد. در الگوی فوق، index به عنصری از جدول رنگ اشاره می‌کند که باید با رنگی که توسط color مشخص می‌شود جایگزین گردد. رنگهای معتبری که پارامتر color می‌تواند بپذیرد در جدولهای ۵-۱۵ و ۶-۱۵ آمده‌اند.

جدول ۵-۱۵ رنگهای معتبر در بورد CGA (۱۶ رنگ اول).

نام ماکرو	شماره معدل	رنگ
BLACK	0	سیاه
BLUE	1	آبی
GREEN	2	سبز
CYAN	3	کبود
RED	4	قرمز
MAGENTA	5	بنفش
BROWN	6	قهوه‌ای
LIGHTGRAY	7	خاکستری روشن
DARKGRAY	8	خاکستری تیره
LIGHTBLUE	9	آبی روشن
LIGHTGREEN	10	سبز روشن
LIGHTCYAN	11	کبود روشن
LIGHTRED	12	قرمز روشن
LIGHTMAGENTA	13	بنفش روشن
YELLOW	14	زرد
WHITE	15	سفید

جدول ۶-۱۵ رنگهای معتبر در بوردهای EGA و VGA.

نام ماکرو	شماره معادل	رنگ
EGA_BLACK	1	سیاه
EGA_BLUE	2	آبی
EGA_CYAN	3	کبود
EGA_RED	4	قرمز
EGA_MAGENTA	5	بنفش
EGA_BROWN	6	قهوه‌ای
EGA_LIGHTCRAY	7	خاکستری روشن
EGA_DARKGRAY	56	خاکستری تیره
EGA_LIGHTBLUE	57	آبی روشن
EGA_LIGHTGREEN	58	سبز روشن
LIGHTCYAN	59	کبود روشن
EGA_LIGHTRED	60	قرمز روشن
EGA_LIGHTMAGENTA	61	بنفش روشن
EGA_YELLOW	62	زرد
EGA_WHITE	63	سفید

در بوردهای گرافیکی CGA فقط می‌توان رنگ زمینه را تغییر داد و index رنگ زمینه برابر با صفر است. به عنوان مثال، در بورد CGA دستور ; (setpalette(0,GREEN موجب می‌شود تا رنگ زمینه به سبز تغییر کند.
در بوردهای EGA و VGA می‌توان به هر کدام از ۱۶ index مختلف رنگ خاصی را نسبت داد. به عنوان مثال، دستور setpalette(5,EGA_CYAN) موجب می‌شود تا رنگ کبود، به شماره رنگ ۵ نسبت داده شود.

تابع setallpalette()

تابع setallpalette() کلیه رنگهای جعبه رنگ در بوردهای گرافیکی EGA و VGA را تغییر می‌دهد و دارای الگوی زیر است:

void far setallpalette (struct palettetype far *pal)

در الگوی فوق، palettetype ساختمانی است که به صورت زیر تعریف شده است:

```
struct palettetype {
        unsigned char size ;
        signed char colors [16] ;
} ;
```

برای اجرای تابع setallpalette() باید size را برابر با تعداد رنگهای موجود در جعبه رنگ قرار داد و سپس رنگ مناسبی را برای هر عنصر آرایه رنگها انتخاب نمود. رنگهای معتبر در بوردهای گرافیکی مختلف را می‌توان از جدولهای ۵-۱۵ و ۶-۱۵ پیدا کرد. به عنوان مثال، مجموعه دستورات زیر، جعبه رنگ بوردهای EGA یا VGA را با اولین ۱۶ رنگ پر می‌کند.

```
struct palettetype p ;
int i ;
for (i = 0 ; i < 16 ; i ++)
            p.colors [i] = i ;
p.size = 16 ;
setallpalette (p) ;
```

تابع getpalette()

تابع getpalette() برای تعیین رنگهای موجود در جعبه رنگ فعلی به کار می‌رود و دارای الگوی زیر است:

void far getpalette (struct palettetype far *pal)

ساختمان palettetype به صورت زیر تعریف شده است :

```
#define MAXCOLORS 15
struct palettetype {
      unsigned char size ;
      signed char colors [MAXCOLORS+1] ;
} ;
```

در ساختار فوق، size تعداد رنگهای موجود در جعبه رنگ و آرایه color شماره رنگهای موجود در جعبه رنگ را مشخص می‌کند. شماره رنگها و اسامی ماکروهای مربوط، همانند جدولهای ۵-۱۵ و ۶-۱۵ می‌باشند.

مثال ۳-۱۵

برنامه‌ای که تعداد رنگهای قابل استفاده در وجه گرافیکی تعیین شده توسط initgraph() را مشخص می‌کند.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int main(void)
{
   int driver,mode;
   struct palettetype p;
   driver=DETECT;
   mode=0;
   initgraph(&driver,&mode," ");
   getpalette(&p);
   printf("\n number of color in ") ;
   printf(" palette:%d",p.size);
   getch();
   restorecrtmode();
return 0;
}
```

تابع setbackcolor()

تابع setbackcolor() برای تغییر رنگ زمینه به کار می‌رود و دارای الگوی زیر است:

void far setbackcolor (int color)

در الگوی فوق، color یکی از مقادیر (ماکروهای) معتبر در جدول ۵-۱۵ است. به عنوان مثال، دستور setbackcolor(LIGHTBLUE) ; موجب می‌شود تا رنگ زمینه به صورت آبی روشن منظور شود.

تابع setcolor()

تابع setcolor() برای تعیین رنگ گرافیک مورد استفاده قرار گرفته، دارای الگوی زیر است:

void far setcolor (int color)

در الگوی فوق، color یکی از رنگهای معتبر از جدولهای ۵-۱۵ و ۶-۱۵ می‌باشد.

مثال ۴-۱۵

برنامه‌ای که موجب رسم ۱۶ خط با رنگهای مختلف در صفحه‌نمایش می‌شود.

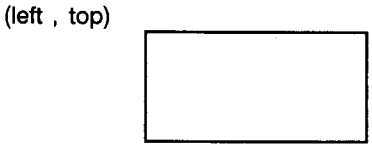
```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int main(void)
 {
   int driver,mode,i;
   driver=DETECT;
   mode=0;
   initgraph(&driver,&mode,"");
   moveto(0,200);
   for(i=0;i<16;i++)
   {
     setcolor(i);
     linerel(20,0);
   }
   getch();
   restorecrtmode();
 return 0;
 }
```

تابع rectangle()

تابع rectangle() برای رسم مستطیل به کار می‌رود و دارای الگوی زیر است:

void far rectangle (int left, int top, int right, int bottom)

در الگوی فوق، دو نقطهٔ (left , top) و (right , bottom) دو سر قطر اصلی مستطیل را مشخص می‌کنند:



مثال ۵-۱۵

برنامه‌ای که مستطیل‌هایی به ابعاد مختلف را در صفحه‌نمایش رسم می‌کند.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int main(void)
{
   int driver,mode,i;
   driver=DETECT;
   mode=0;
   initgraph(&driver,&mode,"");
   setbkcolor(LIGHTGRAY);
   rectangle(100,100,300,300);
   rectangle(150,90,34,300);
   rectangle(0,0,2,2);
   getch();
   restorecrtmode();
return 0;
}
```

تابع setfillpattern()

تابع setfillpattern() الگویی را برای پرکردن اشکال گرافیکی انتخاب می‌کند که توسط بعضی از توابع مثل floodfill() (این تابع در ادامه همین فصل بررسی می‌شود) مورد استفاده قرار می‌گیرد. تابع setfillpattern() دارای الگوی زیر است:

void far setfillpattern (char far *pattern, int color)

در الگوی فوق، pattern آرایه‌ای حداقل ۸بایتی است که الگوی موردنظر را مشخص می‌کند. الگوی ۸بایتی را می‌توان با یک بایت ۸بیتی نقش کرد. اگر بیتی برابر با یک (۱) باشد، رنگِ مشخص شده توسط پارامتر color منظور خواهد شد وگرنه همان رنگ زمینه اعمال می‌شود.

مثال ۶-۱۵

برنامه‌ای که الگویی را ایجاد کرده در پرکردن مستطیل مورد استفاده قرار می‌دهد.

```
#include <graphics.h>
#include <conio.h>
int main(void)
{
  int driver,mode;
  char p[8]={10,20,30,40,50,60,70,80};
  driver=DETECT;
  mode=0;
  initgraph(&driver,&mode,"");
  setcolor(GREEN);
  rectangle(100,200,200,300);
  setfillpattern(p,RED);
  floodfill(150,250,GREEN);
  getch();
  restorecrtmode();
  return 0;
}
```

تابع floodfill()

تابع floodfill() داخل یک شکل گرافیکی بسته را با الگو و رنگ تعیین شده توسط تابع setfillpalette()، پر می‌کند و دارای الگوی زیر است:

void far floodfill (int x, int y, int border)

در الگوی فوق، نقطهٔ (x, y) نقطه‌ای در داخل شکل گرافیکی بسته می‌باشد و border رنگ محیط شکل را مشخص می‌کند.

تابع setfillstyle()

تابع setfillstyle() برای تعیین رنگ و سبک پرشدن شکلهای گرافیکی که توسط اکثر توابع گرافیکی مورد استفاده قرار می‌گیرد، به کار رفته دارای الگوی زیر است:

void far setfillstyle (int pattern, int color)

در الگوی فوق، color یکی از رنگهای معتبر در جدولهای ۵-۱۵ و ۶-۱۵ است و pattern یکی از مقادیر یا ماکروهای جدول ۷-۱۵ می‌باشد.

جدول ۷-۱۵ الگوهای قابل تعریف در تابع setfillstyle() .

ماکرو	معادل عددی	مفهوم
EMPTY_FILL	0	پرکردن شکل با رنگ زمینه
SOLID_FILL	1	پرکردن شکل با رنگی پررنگ
LINE_FILL	2	پرکردن شکل توسط خطوط
LTSLASH_FILL	3	پرکردن شکل توسط / روشن
SLASH_FILL	4	پرکردن شکل توسط /
BKSLASH_FILL	5	پرکردن شکل توسط \
LTBKSLASH_FILL	6	پرکردن شکل توسط \ روشن
XHATCH_FILL	8	پرکردن شکل توسط هاشورزنی
INTERLEAVE_FILL	9	پرکردن شکل از کاراکترهای گوناگون
WIDE_DOT_FILL	10	پرکردن شکل توسط نقاط توخالی
CLOSE_DOT_FILL	11	پرکردن شکل با نقاط پر
USER_FILL	12	الگوی انتخابی توسط برنامه‌نویس با استفاده از تابع setfillpattern()

مثال ۷-۱۵

برنامه‌ای که مستطیلی را با خطوط پر می‌کند.

```
#include "graphics.h"
#include "conio.h"
int main(void)
{
  int driver,mode;
  char p[8]={10,20,30,40,50,60,70,80};
  driver=DETECT;
  mode=0;
  initgraph(&driver,&mode,"");
  setcolor(GREEN);
  rectangle(100,200,200,300);
  setfillpattern(p,RED);
  setfillstyle(LINE_FILL,RED);
  floodfill(150,250,GREEN);
  getch();
  restorecrtmode();
return 0;
}
```

توابع bar() و bar3d()

تابع bar() برای رسم هیستوگرام به کار می‌رود و دارای الگوی زیر است :

void far bar (int left, int top, int right, int bottom)

در الگوی فوق، نقطهٔ (left , top) نقطهٔ بالای سمت چپ و نقطهٔ (right , bottom) نقطهٔ پایین سمت راست را مشخص می‌کنند. هیستوگرام رسم شده، با الگوی تعیین شده توسط تابع setfillpattern() پر می‌شود.
تابع bar3d() همانند تابع bar() عمل می‌کند؛ با این تفاوت که هیستوگرام رسم شده، ۳ بعدی خواهد بود. تابع bar3d() دارای الگوی زیر است :

void far bar3d (int left, int top, int right, int bottom, int depth, int topflag)

در الگوی فوق، depth میزان بعد سوم را مشخص می‌کند که اگر برابر با صفر باشد، هیستوگرام دوبعدی رسم خواهد شد. اگر topflag صفر باشد، هیستوگرام دارای قله خواهد بود.

مثال ۸-۱۵

برنامه‌ای که هیستوگرام ۲ بعدی و ۳ بعدی رسم می‌کند.

```
#include "graphics.h"
#include "conio.h"
int main(void) {
  int driver,mode;
  driver=DETECT;
  mode=0;
   initgraph(&driver,&mode,"");
   setfillstyle(SOLID_FILL,GREEN);
   bar(100,100,120,200);
   setfillstyle(SOLID_FILL,RED);
   bar3d(200,100,220,200,10,1);
   getch();
   restorecrtmode();
return 0;
}
```

تابع arc()

تابع arc() برای رسم کمان مورد استفاده قرار می‌گیرد و دارای الگوی زیر است :

void far arc (int x, int y, int start, int end, int radius)

در الگوی فوق، (x , y) مرکز کمان، start نقطهٔ شروع، end نقطهٔ پایان و radius شعاع کمان می‌باشد (start و end برحسب درجه می‌باشند).

مثال ۹-۱۵

برنامه‌ای که کمانی از زاویه ۰ تا ۹۰ درجه و با مرکز (۱۰۰ و ۱۰۰) رسم می‌کند.

```
#include "graphics.h"
#include "conio.h"
int main(void)
{
  int driver,mode;
  driver=DETECT;
  mode=0;
  initgraph(&driver,&mode,"");
  setcolor(WHITE);
  arc(100,100,0,90,20);
  getch();
  restorecrtmode();
return 0;
}
```

تابع ()circle

تابع ()circle برای رسم دایره مورد استفاده قرار گرفته دارای الگوی زیر است:

```
void far  circle(int x, int y, int radius)
```

در الگوی فوق، نقطهٔ (x , y) مرکز دایره و radious شعاع آن را مشخص می‌کند.

مثال ۱۰-۱۵

برنامه‌ای که ۵ دایره به مرکز (۲۰۰ و ۲۰۰) رسم می‌کند.

```
#include "graphics.h"
#include "conio.h"
int main(void) {
  int driver,mode;
  driver=DETECT;
  mode=0;
  initgraph(&driver,&mode,"");
  circle(200,200,20);
  circle(200,200,30);
  circle(200,200,40);
  circle(200,200,50);
  circle(200,200,60);
  getch();
  restorecrtmode();
return 0;
}
```

تابع ellipse()

تابع ellipse() برای رسم بیضی و یا قسمتی از بیضی مورد استفاده قرار گرفته دارای الگوی زیر است :

void far ellipse (int x, int y, int start, int end, int xradiuse, int yradius)

در الگوی فوق، نقطهٔ (x , y) مرکز بیضی، xradious شعاع افقی و yradious شعاع عمودی را مشخص می‌کند. برای رسم قسمتی از بیضی باید نقطهٔ شروع و پایان را به ترتیب با start و end تعیین نمود. (start و end برحسب درجه بیان می‌شوند، اگر start برابر با ۰ و end برابر با ۳۶۰ باشد، بیضی رسم شده، کامل خواهد بود).

مثال ۱۱-۱۵

برنامه‌ای که موجب رسم بیضی می‌شود.

```
#include "graphics.h"
#include "conio.h"
int main(void)
{
  int driver,mode;
  driver=DETECT;
  initgraph(&driver,&mode,"");
  ellipse(100,100,0,360,80,40);
  getch();
  return 0;
}
```

تابع setviewport()

تابع setviewport() برای ایجاد یک محدودهٔ گرافیک مورد استفاده قرار گرفته و دارای الگوی زیر است:

void far setviewport (int left, int top, int right, int bottom, int clip) ;

در الگوی فوق، نقطهٔ (left, top) گوشه بالای سمت چپ و نقطهٔ (right, bottom) گوشه پایین سمت راست محدوده گرافیک را تعیین می‌کند.اگر clip برابر با ۱ باشد،شکل گرافیکی یا متن نمی‌تواند از محدوده گرافیکی خارج گردد.

توابع setactivepage() و setvisualpage()

اطلاعاتی که در حافظه نمایش نوشته می‌شوند در صفحه‌نمایش ظاهر می‌گردند. بعضی از بوردهای گرافیکی مثل EGA و VGA دارای حافظه زیادی هستند، به طوری که همهٔ اطلاعات موجود در حافظه نمایش همزمان قابل ظاهرشدن در صفحه‌نمایش نیستند. لذا این حافظه به قسمتهایی به نام **صفحه** تقسیم‌بندی می‌شود که زبان C در حالت فرضی از صفحه شماره صفر استفاده می‌کند. انتقال از صفحه‌ای به صفحهٔ دیگر و نمایش اطلاعات آنها در صفحه نمایش امکان‌پذیر می‌باشد. یعنی گرچه در آن واحد فقط اطلاعات یک صفحه در صفحه‌نمایش ظاهر می‌شود ولی می‌توان اطلاعاتی را بر روی صفحات دیگر قرار داد و با انتقال به آن صفحات، اطلاعات را در صفحه‌نمایش مشاهده نمود. این امر بخصوص در مواقعی که ساختن تصویرهای گرافیکی مستلزم وقت زیادی است مفید واقع

می‌شود. تابع ()setactivepage صفحه خروجی برنامه‌های گرافیکی را مشخص می‌کند و دارای الگوی زیر است :

void far setactivepage (int page)

در این الگو، page شماره صفحه‌ای است که خروجی‌های برنامه باید به آنجا منتقل شوند (صفحه فعال). به عنوان مثال، دستور setactivepage (1) موجب می‌شود تا صفحه شماره ۱ به عنوان صفحه فعال انتخاب گردد.

تابع ()setvisualpage موجب می‌شود تا اطلاعات موجود در یک صفحه دلخواه بر روی صفحه‌نمایش ظاهر شوند. این تابع دارای الگوی زیر است:

void far setvisualpage (int page)

در این الگو، page شماره صفحه‌ای است که اطلاعات موجود در آن باید در صفحه‌نمایش ظاهر شوند. به‌عنوان مثال، دستور setvisualpage(1); موجب می‌شود تا اطلاعات موجود در صفحه شماره ۱ بر روی صفحه نمایش ظاهر شوند.

توابع outtext() و outtextxy()

تابع ()outtext در وجوه گرافیکی برای نمایش متن در موقعیت جاری صفحه‌نمایش به کار رفته و دارای الگوی زیر است :

void far outtext (char far *str)

در الگوی فوق، str رشته‌ای است که باید در صفحه‌نمایش ظاهر شود.

تابع ()outtextxy در وجوه گرافیکی برای نمایش متن در سطر و ستون خاصی مورد استفاده قرار گرفته و دارای الگوی زیر است :

void far outtextxy (int x, int y, char str)

در الگوی فوق، نقطهٔ (x , y) محلی از صفحه‌نمایش را مشخص می‌کند که رشتهٔ str باید در آنجا نوشته شود.

مثال ۱۲-۱۵

برنامه‌ای که چگونگی استفاده از ()outtext و ()outtextxy را نشان می‌دهد.

```
#include "graphics.h"
#include "conio.h"
void main(void)
{
  int driver=DETECT,mode=0,i;
  struct palettetype p;;
   initgraph(&driver,&mode,"");
  outtext("this is an example.");
  outtext("another line");
  for(i=100;i<200;i+=11)
    outtextxy(200,i,"c is good");
  getch();
  restorecrtmode();
}
```

تابع settextstyle()

تابع settextstyle() در وجوه گرافیکی برای تغییر اندازه‌های کاراکترهایی که توسط توابع نمایش متن (مثل توابع outtext() و outtextxy()) بر روی صفحه نمایش ظاهر می‌شوند به کار می‌رود و دارای الگوی زیر است:

void far settextstyle (int font, int direction, int size)

در الگوی فوق، font اندازه کاراکتر را مشخص می‌کند و یکی از مقادیر یا ماکروهای زیر را می‌پذیرد:

نام ماکرو	معادل عددی	مفهوم
DEFAULT_FONT	0	8×8bit_mapped
TRIPLEX_FONT	1	stroked triplex
SMALL_FONT	2	small stroked font
SCANS_SERIF	3	stroked scan serif
GOTHIC_FONT	4	stroked gothic

پارامتر direction نحوه نمایش متن را در صفحه نمایش مشخص می‌کند - از چپ به راست یا از بالا به پایین - که مقادیر یا ماکروهای معتبر آن عبارتنداز:

نام ماکرو	معادل عددی	مفهوم
HORIZ_DIR	0	از چپ به راست
VERT_DIR	1	از بالا به پایین

size در font ضرب می‌شود و اندازه کاراکتر را افزایش می‌دهد. مقادیر معتبر برای پارامتر size از ۰ تا ۱۰ می‌باشند.

مثال ۱۳-۱۵

برنامه‌ای که چگونگی استفاده از تابع settextstyle() را نشان می‌دهد.

```
#include "graphics.h"
#include "conio.h"
void main(void) {
  int driver=DETECT,mode=0,i;
   initgraph(&driver,&mode,"");
   outtext("NORMAL");
    settextstyle(GOTHIC_FONT,HORIZ_DIR,10);
   outtext("GOTHIC");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
   outtext("TRIPLEXE");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,7);
   getch();
   restorecrtmode();
}
```

تابع gettextsettings()

تابع gettextsettings() الگوی متنی را که اکنون توسط توابع نمایش متن در وجوه گرافیکی، مورد استفاده قرار می‌گیرد مشخص می‌کند و دارای الگوی زیر است:

void far gettextsettings (struct textsettingstype far *info)

در الگوی فوق، info ساختمانی از نوع textsettingstype است که الگوی متن را نگهداری می‌کند و دارای ساختار زیر است:

```
struct textsettingstype {
        int font ;
        int direction ;
        int charsize ;
        int horiz ;
        int vert ;
} ;
```

در ساختار فوق، font یکی از مقادیر زیر را می‌پذیرد:

نام ماکرو	معادل عددی	مفهوم
defalt_font	0	8*8 bit_mapped
TRIPLEX_FONT	1	stroked triplex
SMALL_FONT	2	stroked small
SCANS_SERIF	3	stroked scans
GOTHIC_FONT	4	stroked gothic

مقادیر معتبر direction یکی از ماکروها یا مقادیر زیر می‌باشد:

نام ماکرو	معادل عددی	مفهوم
HORIZ_DIR	0	از چپ به راست
VERT_DIR	1	از بالا به پایین

charsize ضریبی است که در اندازه کاراکتر تأثیر می‌گذارد و مقادیر horiz و vert مشخص می‌کنند که متن موردنظر نسبت به موقعیت جاری مکان‌نما چگونه باشد، که مقادیر یا ماکروهای معتبر آنها عبارتنداز:

ماکرو	مقدار عددی	مفهوم
LEFT_TEXT	0	موقعیت جاری در سمت چپ متن باشد
CENTER_TEXT	1	موقعیت جاری در وسط متن باشد
RIGHT_TEXT	2	موقعیت جاری در سمت راست متن باشد
BOTTOM_TEXT	3	موقعیت جاری در پایین متن باشد
TOP_TEXT	4	موقعیت جاری در بالای متن باشد

توابع getmaxx() و getmaxy()

توابع getmaxx() و getmaxy() به ترتیب، بزرگترین مقدار x و y را در وجه گرافیکی جاری مشخص می‌کنند و دارای الگوی زیر هستند:

```
int far getmaxx (void)
int far getmaxy (void)
```

به عنوان مثال، دستور زیر بزرگترین مقدار x و y در وجه گرافیکی جاری را در صفحه نمایش چاپ می‌کند:

```
printf ("\n maxx = %d, maxy = %d", getmaxx( ), getmaxy( )) ;
```

تابع getpixel()

تابع getpixel() رنگ پیکسل موجود در نقطهٔ دلخواهی از صفحه‌نمایش را مشخص می‌کند و دارای الگوی زیر است:

```
int far getpixel (int x, int y)
```

در الگوی فوق، نقطهٔ (x, y) محلی از صفحه‌نمایش را مشخص می‌کند که رنگ پیکسل موجود در آن محل باید تعیین گردد.

تابع settextjustify()

تابع settextjustify() چگونگی نمایش متن را در وجوه گرافیکی نسبت به موقعیت جاری تعیین می‌کند و دارای الگوی زیر است:

```
void far settextjustify (int horiz, int vert)
```

در الگوی فوق، پارامترهای horiz و vert چگونگی نمایش متن را مشخص می‌کنند و یکی از مقادیر یا ماکروهای زیر را می‌پذیرند:

نام ماکرو	معادل عددی	مفهوم
LEFT_TEXT	0	موقعیت جاری در سمت چپ متن باشد
CENTER_TEXT	1	موقعیت جاری در وسط متن باشد
RIGHT_TEXT	2	موقعیت جاری در سمت راست متن باشد
BOTTOM_TEXT	3	موقعیت جاری در پایین متن باشد
TOP_TEXT	4	موقعیت جاری در بالای متن باشد

حالت فرضی نمایش متن به صورت (LEFT_TEXT , TOP_TEXT) می‌باشد. به عنوان مثال دستور settextjustify (RIGHT_TEXT , TOP_TEXT) ; موجب می‌شود تا موقعیت جاری در سمت راست متن قرار گیرد.

توابع cleardevice() و clearviewport()

تابع cleardevice() در **وجوه گرافیکی** موجب پاک کردن صفحه‌نمایش می‌شود و موقعیت جاری را به (۰ و ۰) منتقل می‌کند و دارای الگوی زیر است:

```
void far cleardevice (void)
```

تابع clearviewport() برای پاک کردن یک محدودهٔ گرافیکی مورد استفاده قرار گرفته و دارای الگوی زیر است:

```
void far clearviewport (void)
```

مثال ۱۴-۱۵

برنامه‌ای که یک محدوده گرافیکی را ایجاد کرده، متنهایی را در آن نوشته و سپس آن را پاک می‌کند.

```
#include <graphics.h>
#include <conio.h>
void box(int ,int ,int ,int );
void main(void)
{
  void box(int,int,int,int,int);
  int driver=DETECT,mode=0,i;
  initgraph(&driver,&mode,"");
  box(0,0,639,349,WHITE);
  setviewport(20,20,200,200,1);
  box(0,0,179,179,RED);
  outtext("this is a test of view port ");
  outtextxy(20,10,"press a key");
  getch();
  clearviewport();
  getch();
  restorecrtmode();
}
//**********************
void box(int startx,int starty,int endx, int endy,int color)
{
  setcolor(color);
  line(startx,starty,startx,endy);
  line(startx,starty,endx,starty);
  line(endx,starty,endx,endy);
  line(endx,endy,startx,endy);
}
```

تابع getviewsettings()

تابع getviewsettings() مشخصات محدوده گرافیکی موجود در صفحه‌نمایش را در اختیار کاربر قرار می‌دهد و دارای الگوی زیر است:

```
void far getviewsettings (struct viewporttype far *info)
```

در الگوی فوق، viewporttype ساختمانی است که مشخصات محدوده گرافیکی در آن قرار می‌گیرند و دارای ساختار زیر است:

```
struct viewporttype {
        int left, top, right, bottom ;
        int clipflag ;
} ;
```

در ساختمان viewporttype اجزای left و top گوشه بالای سمت چپ و اجزای right و bottom گوشهٔ پایین سمت راست را مشخص می‌کنند. به عنوان مثال، دستورات زیر، مشخصات صفحه‌نمایش را در اختیار قرار می‌دهند:

```
struct viewporttype info;
getviewsettings (&info) ;
printf ("\n view port is : %dx%d" ,
      info.left, info.right, info.top, info.bottom) ;
```

توابع getx() و gety()

توابع getx() و gety() در وجوه گرافیکی، موقعیت جاری مکان‌نما را پیدا می‌کنند و دارای الگوهای زیر هستند:

```
int far getx (void)
int far gety (void)
```

به عنوان مثال، دستور زیر موقعیت جاری مکان‌نما را در صفحه‌نمایش چاپ می‌کند.

```
printf ("current position : (%d,%d)",getx(),gety()) ;
```

تابع closegraph()

تابع closegraph() سیستم را از حالت گرافیکی خارج می‌کند و کلیه حافظه‌هایی را که در اختیار مبدلهای گرافیک و font قرار گرفته است به سیستم عامل برمی‌گرداند. این تابع دارای الگوی زیر است:

```
void far closegraph (void)
```

تابع closegraph() موجب می‌شود تا حالت صفحه‌نمایش به حالت قبل از عمل تابع initgraph() برگردد. اگر کار برنامه به اتمام رسیده باشد به جای تابع closegraph() می‌توان از تابع restorecrtmode() استفاده نمود که در این صورت کلیه حافظه‌های تخصیص یافته برای انجام اعمال گرافیکی به سیستم برمی‌گردند.

تابع detectgraph()

تابع detectgraph() نوع بورد گرافیکی را تشخیص می‌دهد و دارای الگوی زیر است :

void far detectgraph (int far *driver, int far *mode)

در الگوی فوق، پارامتر driver شمارهٔ مبدل گرافیک معادل بورد گرافیکی می‌باشد (جدول ۲-۱۵). اگر کامپیوتر دارای بورد گرافیکی نباشد مقدار ۲- در این پارامتر قرار می‌گیرد. بالاترین دقتی که بورد گرافیکی موجود در کامپیوتر می‌تواند آن را پشتیبانی کند در پارامتر mode قرار خواهد گرفت. به عنوان مثال، مجموعهٔ دستورات زیر، بورد گرافیکی جاری را مشخص می‌کنند :

```
int driver, mode ;
detectgraph (&driver, &mode) ;
if (driver == -2)
{
   printf ("no graphics adapter in system") ;
   exit (1) ;
}
```

تابع drawpoly()

تابع drawpoly() برای رسم چند ضلعی مورد استفاده قرار می‌گیرد و دارای الگوی زیر است :

void far drawpoly (int numpoints, int far *points)

در الگوی فوق، numpoints تعداد اضلاع (تعداد گوشه‌های) چند ضلعی را مشخص می‌کند و points به آرایه‌ای اشاره می‌کند که مختصات گوشه‌های چندضلعی در آنجا قرار دارند. طول این آرایه حداقل باید دوبرابر تعداد گوشه‌های چندضلعی منظور شود. در این آرایه، هر گوشه با xو yمخصوص به خود تعریف می‌گردد که ابتدا مقدار x و سپس مقدار yدر آرایه منظور می‌شود.

مثال ۱۵-۱۵

برنامه‌ای که موجب رسم یک ۵ضلعی در صفحه‌نمایش می‌گردد.

```
#include "graphics.h"
#include "conio.h"
int main(void) {
  int driver=0,mode=DETECT;
  int shape[]= {  10,10, 100, 80, 200, 200, 350, 90, 0, 0} ;
   initgraph(&driver,&mode,"");
   drawpoly(5,shape);
   getch();
   restorecrtmode();
  return 0;
}
```

تابع fillpoly()

تابع fillpoly() پس از رسم چندضلعی (همانند drawpoly())، داخل آن را با یک الگو و رنگ جاری پر می‌کند. برای انتخاب الگو می‌توان از تابع setfillpattern() استفاده نمود. الگوی تابع fillpoly() به صورت زیر است:

```
void far fillpoly (int numpoint, int far *point)
```

پارامترهای تابع fillpoly() همانند تابع drawpoly() می‌باشد.

مثال ۱۶-۱۵

برنامه‌ای که پس از رسم چهارضلعی، داخل آن را با الگوی خاصی پر می‌کند.

```
#include "conio.h"
#include "graphics.h"
void  main(void)
{
  int  driver=0,mode=DETECT;
  int shape[]= {
     10,10,
     100,80,
     200,200,
     350,90
   } ;
   initgraph(&driver,&mode,"");
   setfillstyle(INTERLEAVE_FILL,MAGENTA);
   fillpoly(4,shape);
   getch();
   restorecrtmode();
}
```

تابع getarccoords()

تابع getarccoords() مختصات آخرین کمانی که توسط تابع arc() رسم شده است را تشخیص داده در اختیار قرار می‌دهد و دارای الگوی زیر است:

```
void far getarccoords (struct arccoordstype far *coords)
```

در الگوی فوق ساختمان arccoordstype مختصات پیدا شده توسط تابع را نگهداری می‌کند و دارای ساختار زیر است:

```
struct arccoordstype {
    int x , y ;
    int xstart, ystart, xend, yend ;
} ;
```

در ساختار فوق، نقطه (x , y) مرکز کمان، نقطهٔ (xstart , ystart) نقطهٔ شروع و نقطهٔ (xend , yend) نقطهٔ پایان کمان را مشخص می‌کنند.

مثال ۱۷-۱۵

برنامه‌ای که یک ربع از دایره را به مرکز (۱۰۰ و ۱۰۰) رسم می‌کند و سپس ابتدا و انتهای آن را توسط خطی به یکدیگر وصل می‌کند.

```
#include "conio.h"
#include "graphics.h"
void main(void) {
  int driver=DETECT,mode=0 ;
  struct arccoordstype ac ;
   initgraph(&driver,&mode,"") ;
   arc(100,100,0,90,100) ;
   getarccoords(&ac) ;
   line(ac.xstart,ac.ystart,ac.xend,ac.yend);
  getch() ;
   restorecrtmode() ;
}
```

تابع getbkcolor()

تابع getbkcolor() برای تشخیص رنگ زمینه مورد استفاده قرار می‌گیرد و دارای الگوی زیر است:

int far getbkcolor (void)

مقادیر یا ماکرویی که توسط این تابع برگردانده می‌شوند در جدول ۵-۱۵ آمده‌اند. به عنوان مثال، دستور printf ("\n background color is : %d" , getbkcolor ()) ; رنگ زمینه را تشخیص داده در صفحه‌نمایش چاپ می‌کند.

تابع getcolor()

تابع getcolor() رنگی را که اکنون در رسم اشکال گرافیکی مورد استفاده قرار می‌گیرد تشخیص می‌دهد. مقادیر و یا ماکروهایی که توسط این تابع برگردانده می‌شوند در جدولهای ۵-۱۵ و ۶-۱۵ آمده‌اند. به عنوان مثال، دستور printf ("drawing color is : %d", getcolor ()) ; رنگی را که اکنون در رسم اشکال گرافیکی مورد استفاده قرار می‌گیرد تشخیص می‌دهد.

تابع getfillpattern()

تابع getfillpattern() الگویی را که اکنون در رسم اشکال گرافیکی مورد استفاده قرار می‌گیرد مشخص می‌کند و دارای الگوی زیر است:

void far getfillpattern (char far *pattern)

در الگوی فوق، pattern آرایه‌ای است که الگوی موجود را در خود نگهداری می‌کند. برای اطلاعات بیشتر در این مورد می‌توانید به تابع setallfillpattern() و setfillpattern() مراجعه نمایید.

مثال ۱۸-۱۵

برنامه‌ای که الگوی فعلی اشکال گرافیکی را مشخص می‌کند.

```
#include "conio.h"
#include "stdio.h"
#include "graphics.h"
void main(void)
{
  int driver=DETECT,mode=0 ;
  struct arccoordstype ac;
  char f[8];
  int i;
   initgraph(&driver,&mode,"");
   getfillpattern(f) ;
  for (i=0;i<8;i++)
      printf("%d",f[i]);
  getch();
   restorecrtmode();
}
```

تابع getfillsettings()

تابع getfillsettings() شماره و یا ماکروی تعیین کننده الگو و رنگ را مشخص می‌کند و دارای الگوی زیر است:

void far getfillsettings (struct fillsettingstype far *info)

در الگوی فوق، fillsettingstype ساختمانی است که شماره یا ماکروی الگو و رنگ را نگهداری می‌کند و دارای ساختار زیر است:

```
struct fillsettingstype ;
      int pattern ;
      int color ;
} ;
```

مقادیر و یا ماکروهای معتبر برای الگو در جدول ۷-۱۵ و رنگ در جدولهای ۵-۱۵ و ۶-۱۵ آمده‌اند. به عنوان مثال، مجموعه دستورات زیر شماره الگو و رنگ جاری را مشخص می‌کنند:

```
struct fillsettingstype p;
getfillsetting (&p) ;
printf ("pattern is : , color is :",  p.pattern, p.color) ;
```

توابع getimage() و imagesize() و putimage()

تابع getimag() برای ذخیره کردن قسمتی از صفحه‌نمایش (حاوی گرافیک) در حافظه مورد استفاده قرار گرفته و دارای الگوی زیر است:

void far getimage (int left, int top, int right, int bottom, void far *buf)

در الگوی فوق، نقاط (left, top) و (right, bottom) مختصات دو سر قطر مستطیلی (محدوده‌ای) از صفحه‌نمایش را مشخص می‌کند که گرافیک موجود در آن محدوده باید در حافظه‌ای که پارامتر buf به آن اشاره می‌کند ذخیره گردد. برای تعیین میزان حافظه لازم جهت ذخیره کردن شکل (حافظه‌ای که buf به آن اشاره خواهد کرد) از تابع imagesize() استفاده می‌شود. تابع imagesize() دارای الگوی زیر است:

unsigned far imagesize (int left, int top, int right, int bottom)

در الگوی فوق، نقاط (left, top) و (right, bottom) قسمتی از صفحه‌نمایش را مشخص می‌کنند که میزان حافظه لازم برای ذخیره کردن این قسمت از صفحه‌نمایش باید توسط تابع imagesize() تعیین گردد. به عنوان مثال، دستورات:

```
unsigned size ;
size = imagesize (10,10,100,100) ;
```

میزان حافظه لازم را برای ذخیره کردن قسمتی از صفحه‌نمایش که در محدوده تعیین شده توسط نقاط (۱۰ و ۱۰) و (۱۰۰ و ۱۰۰) قرار دارد، تعیین می‌کنند و در متغیر size قرار می‌دهند.

تابع putimage()، شکلی را که قبلاً توسط تابع getimage() در قسمتی از حافظه ذخیره شده است در نقطهٔ دیگری از حافظه کپی می‌کند و دارای الگوی زیر است:

void far putimage (int x, int y, void far *buf, int op)

در الگوی فوق، buf به شکلی اشاره می‌کند که قبلاً توسط تابع getimage() ذخیره شده است و نقطهٔ (x,y) محلی از صفحه‌نمایش را مشخص می‌کند که شکل موجود در حافظه buf با شروع از این نقطه، در صفحه‌نمایش ذخیره می‌شود. پارامتر op چگونگی ظاهر شدن شکل را در صفحه‌نمایش مشخص می‌نماید که مقادیر و یا ماکروهای معتبر آن در زیر آمده‌اند:

نام ماکرو	مقدار معادل	مفهوم
COPY_PUT	0	شکل به همان صورتی که هست کپی شود
XOR_PUT	1	XOR
OR_PUT	2	OR
AND_PUT	3	AND
NOT_PUT	4	NOT

مثال ۱۹-۱۵

برنامه‌ای که شکلی را در صفحه نمایش رسم کرده، آن را در حافظه‌ای ذخیره می‌کند و در نقطهٔ دیگری از صفحه‌نمایش رسم می‌نماید.

```
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
void main(void)
 {
   void box(int, int, int, int, int) ;
   int driver = DETECT, mode = 0 ;
   unsigned  size ;
   void *buf ;
   initgraph(&driver, &mode,"");
   box(20, 20, 200, 200, 15) ;
   setcolor(RED) ;
   line(20, 20, 200, 200) ;
   setcolor(GREEN) ;
   line(20, 200, 200, 20) ;
   getch();
   size = imagesize(20, 20, 200, 200) ;
   if(size != 1)  {
      buf = malloc(size) ;
      if(buf){
         getimage(20, 20, 200, 200, buf) ;
         putimage(100, 100, buf, COPY_PUT) ;
         putimage(300, 50, buf, COPY_PUT) ;
      }
   }
   outtext("press a key.") ;
   getch() ;
   restorecrtmode();
 }
//**********************
void box(int startx,int starty,int endx, int endy,int color)
  {
    setcolor(color) ;
    rectangle(startx,starty,endx,endy) ;
  }
```

جدول ۸-۱۵ ماکروهای‌معتبر پارامتر style.

نام ماکرو	مفهوم
SOLID_LINE	خط (-)
DOTTED_LINE	خط نقطه‌چین (...)
CENTERED_LINE	ترکیبی از خط و نقطه (-.-.-)
DASHED_LINE	خط‌چین
USERBIT_LINE	خطی که توسط برنامه‌نویس تعریف می‌شود

تابع setlinestyle()

تابع setlinestyle() نوع خطوطی راکه توسط توابع گرافیکی (توابعی که خط رسم می‌کنند) باید رسم شوند مشخص می‌کند و دارای الگوی زیر است:

void far setlinestyle (int style, unsigned pattern, int width)

در الگوی فوق، style نوع خط را مشخص می‌کند که مقادیر معتبر آن، ماکروهای جدول ۸-۱۵ می‌باشند. اگر style برابر با USERBIT_LINE باشد، پارامتر ۱۶ بیتی pattern چگونگی نمایش خط را مشخص می‌کند. هر بیت در این پارامتر متناظر با یک پیکسل است. اگر بیتی برابر با ۱ باشد پیکسل متناظر با آن روشن می‌شود وگرنه خاموش خواهد بود. پارامتر width پهنای خط را مشخص می‌کند که ماکروهای معتبر آن در جدول ۹-۱۵ آمده‌اند.

مثال ۲۰-۱۵

برنامه‌ای که نوع خطوط تعریف شده در محیط C را نمایش می‌دهد.

```
#include "graphics.h"
#include "conio.h"
void main(void) {
  int i, driver = 0, mode = DETECT ;
  initgraph(&driver, &mode, "");
  for (i = 0; i < 4; i++)  {
     setlinestyle(i, 0, 1);
     line(i * 50, 100, i * 50 + 50, 100);
  }
  getch();
  restorecrtmode();
}
```

جدول ۹-۱۵ ماکروهای‌معتبر پارامتر width.

نام ماکرو	پهنای خط
NORM_WIDTH	1 pixel
THIK_WIDTH	3 pixel

تابع getlinesettings()

تابع getlinesettings() نوع خطی را که توسط توابع گرافیکی رسم می‌شوند مشخص می‌کند و دارای الگوی زیر است:

void far getlinesettings (struct linesettingstype far *info)

در این الگوی، info ساختمانی از نوع linesettingstype است که نوع خطوط را نگهداری می‌کند. این ساختمان به صورت زیر تعریف می‌شود:

```
struct linesettingstype {
        int linestyle ;
        unsigned upattern ;
        int thickness ;
} ;
```

در این ساختار، linestyle() نوع خط را مشخص می‌کند که مقادیر معتبر آن در جدول ۸-۱۵ آمده‌اند. upattern همانند پارامتر pattern در تابع setlinestyle() است و thickness پهنای خط را مشخص می‌کند که مقادیر معتبر آن در جدول ۹-۱۵ آمده‌اند. به عنوان مثال، دستورات زیر، نوع خط جاری را مشخص می‌کنند:

```
struct linesettingstype info ;
getlinesettings (&info) ;
```

تابع graphdefaults()

تابع graphdefaults() موجب reset شدن سیستم گرافیک می‌شود. با اجرای این تابع، محدوده گرافیکی تعریف شده از بین می‌رود و نقطه (۰ و ۰) به عنوان نقطهٔ جاری منظور می‌گردد. رنگهای زمینه، رنگ رسم شکل گرافیکی، الگوها و ... به حالت فرضی اولیه برمی‌گردند. این تابع دارای الگوی زیر است:

void far graphdefaults (void)

توابع graphresult() و grapherrormsg()

تابع graphresult() نتیجهٔ حاصل از آخرین عمل گرافیکی را مشخص می‌کند. برای این منظور کدی را که بیانگر این نتیجه است برمی‌گرداند. این تابع دارای الگوی زیر است:

int far graphresult (void)

مقادیری که توسط تابع graphresult() برگردانده می‌شود در جدول ۱۰-۱۵ مشاهده می‌گردد.

تابع grapherrormsg() با اخذ کد خطای برگردانده شده توسط تابع graphresult() ، پیام خطای متناسب با آن را مشخص می‌کند و دارای الگوی زیر است:

char far *grapherrormsg (int error)

در الگوی فوق، error کدی است که توسط تابع graphresult() برگردانده می‌شود. به عنوان مثال، دستور

printf(%s",grapherrormsg(graphresult()); پیام خطای حاصل از آخرین عمل گرافیکی را در صفحه نمایش چاپ می‌کند.

تابع ()getmaxcolor

تابع ()getmaxcolor حداکثر تعداد رنگهای قابل استفاده در وجه گرافیکی جاری را مشخص می‌کند و دارای الگوی زیر است:

int getmaxcolor (void)

مثال ۱۵-۲۱

برنامه‌ای که حداکثر تعداد رنگهای قابل استفاده در وجه گرافیکی جاری را مشخص می‌نماید.

```
#include "stdio.h"
#include "conio.h"
#include "graphics.h"
void main(void) {
  int driver = 0, mode = DETECT ;
  initgraph(&driver, &mode, "");
  printf("Largest color is:%d", getmaxcolor());
  getch();
  restorecrtmode();
}
```

جدول ۱۰-۱۵ نتیجه حاصل از عمل گرافیکی.

مقداری که تابع برمی‌گرداند	علت	عمل
0	موفقیت	grok
-1	عدم وجود مبدل گرافیک	grnoinitgraph
-2	عدم وجود بورد گرافیکی	grnotdetected
-3	پیدانشدن فایل مبدل گرافیک	grfilenotfound
-4	معتبر نبودن فایل مبدل گرافیک	grinvaliddriver
-5	عدم وجود حافظه کافی	grnoloadmem
-6	عدم وجود حافظه کافی جهت الگوی پرکردن اشکال به‌طور سطحی	grnoscanmem
-7	عدم وجود حافظه کافی جهت الگوی پرکردن اشکال به‌طور کامل و فشرده	grnofloodmem
-8	پیدانشدن فایل font	grfontnotfound
-9	عدم وجود حافظه کافی برای font	grnofontmem
-10	معتبر نبودن وجه گرافیکی	grinvalidmode
-11	خطای کلی در گرافیک	grerror
-12	خطای ورودی - خروجی	grioerror
-13	معتبر نبودن فایل font	grinvalidfont
-14	معتبر نبودن شماره انتخاب font	grinvalidfontnum
-15	معتبر نبودن شماره دستگاه	grinvaliddevicenum

توابع ()graphgetmem_ و ()graphfreemem_

تابع ()graphgetmem_ حافظه لازم جهت انجام اعمال گرافیکی را از سیستم اخذ می‌کند و تابع ()graphfreemem_ پس از انجام اعمال گرافیکی، این حافظه را به سیستم برمی‌گرداند. این دو تابع توسط بورلند C اجرا می‌گردند و نیازی به اجرای آنها توسط برنامه‌نویس نیست. الگوی این توابع به صورت زیر است:

```
void far _graphgetmem (unsigned size)
void far _graphfreemem (void far *ptr, unsigned size)
```

تابع ()pieslice

تابع ()pieslice برای رسم قسمتی از دایره مورد استفاده قرار گرفته، دارای الگوی زیر است:

```
void far pieslice (int x, int y, int start, int end, int radius)
```

در الگوی فوق، نقطهٔ (x,y) مرکز دایره، radius شعاع دایره، start زاویه شروع (درجه) و end زاویه پایان قسمتی از دایره را مشخص می‌کند.

مثال ۲۲-۱۵

برنامه‌ای که با کنار هم قراردادن قسمتهایی از دایره (با رنگهای متفاوت)، دایره کاملی را رسم می‌نماید.

```
#include "conio.h"
#include "graphics.h"
void main(void)
{
  int driver = 0, mode = DETECT ;
  int i, start = 0, end = 45 ;
  struct palettetype  p ;
  initgraph(&driver,&mode,"") ;
  for (i = 0; i < 8; i++);
  {
     setfillstyle(SOLID_FILL,i);
     pieslice(300, 200, start, end, 100);
     start += 45;
     end += 45;
  }
  getch();
  restorecrtmode();
}
```

تابع putpixel()

تابع putpixel() یک پیکسل با رنگ دلخواهی را در صفحه‌نمایش می‌نویسد و دارای الگوی زیر است:

```
void far putpixel (int x, int y, int color)
```

در الگوی فوق، نقطهٔ (x,y) محل پیکسل و color رنگ آن را مشخص می‌کند. به عنوان مثال، دستور putpixel(10 ,20, GREEN); یک پیکسل به رنگ سبز را در نقطهٔ (۲۰ و ۱۰) می‌نویسد.

تابع setusercharsize()

تابع setusercharsize() برای تغییر اندازهٔ متنهایی که در وجوه گرافیکی نوشته می‌شوند به کار رفته و دارای الگوی زیر است:

```
setusercharsize (int mulx, int divx, int muly, int divy)
```

با اجرای تابع setusercharsize() طول هر کاراکتری که در صفحه‌نمایش ظاهر می‌شود در mulx/divx و پهنای آن در multy/divy ضرب می‌گردد.

مثال ۲۳-۱۵

برنامه‌ای که متنهایی را در وجوه گرافیکی با اندازه‌های استاندارد و بزرگ می‌نویسد.

```
#include "conio.h"
#include "graphics.h"
void main(void)
{
  int driver = 0, mode = DETECT ;
  initgraph(&driver, &mode," ");
  outtext("normal text  ");
  setlinestyle(TRIPLEX_FONT, HORIZ_DIR, USER_CHAR_SIZE);
  setusercharsize(5, 1, 5, 1);
  outtext("BIG TEXT");
  getch();
  restorecrtmode();
}
```

توابع texthight() و textwidth()

توابع texthight() و textwidth() به ترتیب برای تعیین طول و عرض یک رشته به پیکسل به کار می‌روند و دارای الگوی زیر هستند:

```
int far texthight (char far *str)
int far textwidth (char far *str)
```

مثال ۲۴-۱۵

برنامه‌ای که طول و عرض رشته‌هایی را پیدا می‌کند.

```
#include "conio.h"
#include "stdio.h"
#include "graphics.h"
void main(void)
{
  int driver = 0, mode = DETECT;
  initgraph(&driver, &mode,"");
  printf("width:%d  ", textwidth("hello"));
  printf("hieght:%d", textheight("hello"));
  getch();
  restorecrtmode();
}
```

توابع غیرگرافیکی صفحه‌نمایش

بعضی از توابع کتابخانه‌ای در مورد صفحه‌نمایش وجود دارند که در وجوه غیرگرافیکی صفحه‌نمایش عمل می‌کنند و در این قسمت مورد بررسی قرار می‌گیرند.

تابع ()window

تابع ()window برای رسم پنجره در صفحه‌نمایش مورد استفاده قرار می‌گیرد و دارای الگوی زیر است:

void window (int left, int top, int right, int bottom)

در الگوی فوق، نقطهٔ (left, top) گوشهٔ بالای سمت چپ پنجره و نقطهٔ (right, bottom) گوشهٔ پایین سمت راست پنجره را مشخص می‌کند. به عنوان مثال، دستور window (10, 10, 10, 10) پنجره‌ای را ایجاد می‌کند که نقاط (۱۰و ۱۰) و (۱۵و ۶۰) مختصات آن را مشخص می‌کنند.

تابع ()delline

تابع ()delline خطی را که مکان‌نما در آنجا قرار دارد حذف می‌کند و سپس بقیه خطوط موجود زیر این خط را به طرف بالا منتقل می‌کند. الگوی این تابع به صورت زیر است:

void delline(void)

مثال ۲۵-۱۵

برنامه‌ای که ۲۴ خط را در صفحه‌نمایش چاپ و سپس خط سوم را حذف می‌نماید.

```
#include "stdio.h"
#include "conio.h"
#include "graphics.h"
void main(void)
```

```
{
  register int i;
  clrscr();
  for (i = 0; i < 24; i++)
   printf("this is line%d\n", i);
  getch();
  gotoxy(1,3);
  delline();
  getch();
}
```

توابع gettext() و puttext()

تابع gettext() متن موجود در محدوده‌ای از صفحه‌نمایش را در حافظه‌ای ذخیره می‌کند و دارای الگوی زیر است:

int gettext (int left, int top, int right, int bottom, void *buf)

در الگوی فوق، نقطهٔ (left,top) گوشهٔ بالای سمت چپ محدوده و نقطهٔ (right,bottom) گوشهٔ پایین سمت راست محدوده و buf به حافظه‌ای اشاره می‌کند که متن موجود در محدوده باید در آن قرار گیرد (مختصات محدوده نسبت به پنجره نمی‌باشند). میزان حافظه لازم جهت ذخیره کردن محدوده از رابطه زیر حساب می‌شود:

میزان حافظه = (right - left) * (top - bottom) * 2

کاربرد ضریب ۲ در رابطه بالا به این دلیل است که هر کاراکتر برای ظاهرشدن در صفحه‌نمایش به ۲ بایت حافظه نیاز دارد: یک بایت برای خود کاراکتر و بایت دیگر برای صفت آن. اگر عمل تابع با موفقیت انجام شود، مقدار ۱ وگرنه مقدار ۰ را برمی‌گرداند. تابع puttext() برای نوشتن متنی بر روی صفحه‌نمایش که توسط gettext() در حافظه ذخیره شده است به کار می‌رود و دارای الگوی زیر است:

int puttext (int left, int top, int right, int bottom, void *buf)

در الگوی فوق، نقاط (left,top) و (right,bottom) محدوده‌ای از صفحه‌نمایش را که متن باید در آنجا نوشته شود مشخص می‌کند و buf به حافظه‌ای اشاره می‌کند که متن موردنظر در آنجا قرار دارد. مجموعه دستورات زیر، عمل توابع puttext() و gettext() را نشان می‌دهند.

```
buf = (char *) malloc (10* 10* 2) ;
gettext (10, 10, 20, 20, buf)
puttext (0, 0, 30, 30, buf)
```

تابع movetext()

تابع movetext() متن موجود در محدوده‌ای از صفحه‌نمایش را در محدودهٔ دیگری از صفحه‌نمایش کپی می‌کند و دارای الگوی زیر است:

int movetext (int left, int top, int right, int bottom, int newleft, int newtop)

در الگوی فوق، نقاط (left,top) و (right,bottom) محدوده‌ای از صفحه‌نمایش را مشخص می‌کنند که متن موجود در آن باید در محدوده دیگری که گوشه بالای سمت چپ آن با نقطهٔ (newleft,newtop) مشخص می‌شود کپی گردد. به عنوان مثال، دستور movetext (1, 1, 8, 8, 10, 10); متن موجود در محدوده‌ای که توسط نقاط (۱و۱) و (۸و۸) مشخص می‌شود را به محدودهٔ دیگری که گوشه بالای سمت چپ آن نقطهٔ (۱۰و۱۰) است کپی می‌کند.

توابع highvideo()، lowvideo() و normvideo()

کاراکترهایی که پس از اجرای تابع highvideo() بر روی صفحه‌نمایش ظاهر می‌شوند، پررنگ و کاراکترهایی که پس از اجرای تابع lowvideo() بر روی صفحه‌نمایش ظاهر می‌شوند، کم‌رنگ و تابع normvideo() باعث برگشت به حالت معمولی می‌شود. الگوی این توابع به صورت زیر است:

```
void highvideo (void)
void lowvideo (void)
void normvideo (void)
```

تابع insline()

تابع insline() موجب درج یک خط در جایی که مکان‌نما در آنجا قرار دارد می‌شود. کلیه خطوطی که زیر این خط قرار دارند را یک سطر به طرف پایین منتقل می‌کند. این تابع دارای الگوی زیر است:

```
void insline (void)
```

مثال ۲۶-۱۵

برنامه‌ای که چگونگی استفاده از تابع insline() را نشان می‌دهد.

```
#include "stdio.h"
#include "conio.h"
#include "graphics.h"
void main(void)
{
  register int i;
  clrscr();
  for (i = 0; i < 24; i++)
  {
   gotoxy(1, i);
   printf("this is line%d\n",i);
  }
  getch();
  gotoxy(1,10);
  insline();
  getch();
}
```

توابع wherex() و wherey()

توابع wherex() و wherey() موقعیت فعلی مکان‌نما را در پنجره یا صفحه‌نمایش مشخص می‌کنند و دارای الگوی زیر هستند:

```
int wherex (void)
int wherey (void)
```

تابع textmode()

این تابع حالت صفحه‌نمایش را در حالت متن تغییر می‌دهد و دارای الگوی زیر است:

```
void textmode (int mode)
```

در این الگو، mode حالت صفحه‌نمایش را تعیین می‌کند و مقادیر معتبر آن در زیر آمده‌اند:

نام ماکرو	مقدرعددی ماکرو	حالت صفحه‌نمایش
BW40	0	۴۰ ستون و سیاه و سفید
C40	1	۴۰ ستون و رنگی
BW80	2	۸۰ ستون و سیاه و سفید
C80	3	۸۰ ستون و رنگی
MONO	7	۸۰ ستون و تک رنگ
LASTMODE	-1	حالت قبلی

به عنوان مثال، دستور ; (C80) textmode، صفحه‌نمایش را در حالت رنگی و ۸۰ستونی قرار می‌دهد.

تابع textbkground()

تابع textbkground() رنگ زمینه را تغییر می‌دهد و دارای الگوی زیر است:

```
void textbkground (int color)
```

در الگوی فوق، color تعیین کننده رنگ زمینه است و مقادیر معتبر آن در زیر آمده‌اند:

نام ماکرو	معادل عددی ماکرو	رنگ
BLACK	0	سیاه
BLUE	1	آبی
GREEN	2	سبز
CYAN	3	کبود
RED	4	قرمز
MAGENTA	5	بنفش
BROWN	6	قهوه‌ای

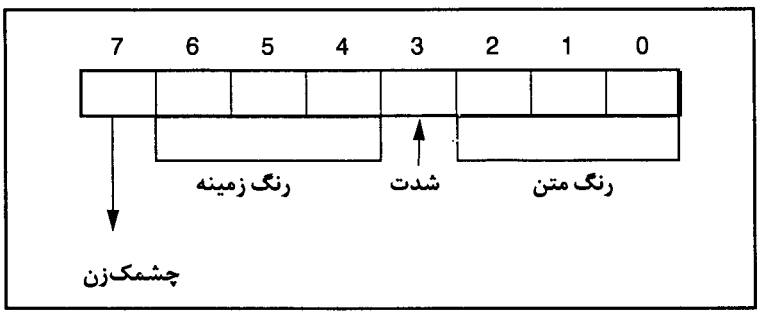
به عنوان مثال، دستور ; textbkground (CYAN) رنگ کبود را به عنوان رنگ زمینه انتخاب می‌کند.

تابع textattr()

تابع textattr() در حالت متن، رنگ زمینه و رنگ متن را تغییر می‌دهد و دارای الگوی زیر است:

```
void textattr (int attr)
```

پارامتر attr صفت کاراکترها را مشخص می‌کند(رنگ نیز جزئی از صفت کاراکتر می‌باشد) و به صورت زیر است:



اگر بیت ۷ برابر با ۱ باشد، کاراکترها چشمک‌زن خواهند بود. بیتهای ۴ و ۵ و ۶ رنگ زمینه و بیتهای ۰ و ۱ و ۲ رنگ متن را مشخص می‌کنند. ساده‌ترین راه برای تعیین رنگ زمینه این است که شماره رنگ را در ۱۶ ضرب کرد و با رنگ متن OR نمود. به عنوان مثال، برای ایجاد رنگ سبزِ زمینه و رنگ متن آبی، باید به صورت زیر عمل کرد:

```
attr = GREEN * 16 | BLUE
```

برای ایجاد وضعیت چشمک‌زن باید رنگ متن، زمینه و BLINK (۱۲۸) را با هم OR نمود. به عنوان مثال، دستور زیر، موجب می‌شود تا رنگ زمینه، آبی و رنگ متن، قرمزِ چشمک‌زن باشد.

```
textattr (RED | BLINK * 16) ;
```

تابع textcolor()

تابع textcolor() رنگ متن را مشخص کرده و دارای الگوی زیر است:

```
void textcolor (int color)
```

در الگوی فوق، color تعیین کننده رنگ متن بوده و مقادیر معتبر آن عبارتند از:

نام ماکرو	معادل عددی ماکرو	رنگ
BLACK	0	سیاه
BLUE	1	آبی
GREEN	2	سبز
CYAN	3	کبود
RED	4	قرمز
MAGENTA	5	بنفش
BROWN	6	قهوه‌ای
LIGHTGRAY	7	خاکستری روشن
DARKGRAY	8	خاکستری تیره
LIGHTBLUE	9	آبی روشن
LIGHTGREEN	10	سبز روشن
LIGHTCYAN	11	کبود روشن
LIGHTRED	12	قرمز روشن
LIGHTMAGENTA	13	بنفش روشن
YELLOW	14	زرد
WHITE	15	سفید
BLINK	16	چشمک‌زن

به عنوان مثال، دستور زیر موجب می‌شود تا متن به صورت چشمک زن در صفحه‌نمایش ظاهر گردد.

```
textcolor (BLINK) ;
```

گرافیک

در بسیاری از زبانهای برنامه‌سازی، از جمله زبان C، انجام اعمال گرافیکی امکان‌پذیر می‌باشد. در فصل شانزدهم توابع متعددی در مورد گرافیک مطرح شده‌اند و مشاهده شد که اعمالی از قبیل رسم خط، دایره، بیضی، کمان و ... توسط توابع کتابخانه‌ای انجام‌پذیر است؛ اما فراگیری روش رسم این اشکال، یا به عبارت دیگر آشنایی با عملکرد توابع کتابخانه‌ای گرافیکی، برای برنامه‌نویسان ماهر ضروری به نظر می‌رسد. لذا در این فصل سعی می‌شود توابعی جهت انجام اعمال زیر نوشته شوند:

۱. رسم خط، مستطیل، دایره و بیضی.
۲. ذخیره کردن گرافیک در فایل.
۳. بازیابی گرافیک از فایل.
۴. چرخش شکل، کپی کردن و انتقال اشکال در یک محل جدید.
۵. انجام عمل طراحی توسط کاربر.

همانطور که در فصل ۱۶ ملاحظه کردید، اولین قدم در انجام اعمال گرافیکی، تعیین یک وجه گرافیکی مناسب برای صفحه‌نمایش است (جدول ۱-۱۶).

در صفحه‌نمایش رنگی می‌توان از امکانات رنگ‌آمیزی استفاده نمود. رنگ زمینه در وجوه گرافیکی را می‌توان از رنگهای جدول ۲-۱۶ انتخاب کرد.

رنگ اشکال گرافیکی به کمک جعبهٔ رنگ و یا یکی از رنگهای جدول ۲-۱۶ مشخص می‌گردد. به عنوان مثال، در بورد گرافیکی CGA تعداد ۴ جعبه رنگ وجود دارد (۰ تا ۳)، که هر جعبه رنگ شامل ۴ رنگ (۰ تا ۳) است. رنگ شمارهٔ صفر، همان رنگ زمینه می‌باشد (جدول ۳-۱۶).

تولید رنگ

شاید این سئوال مطرح شود که رنگهای موجود در جدول ۲-۱۶ چگونه تولید می‌گردند؟ در بوردهای گرافیکی CGA و بعضی از وجوه گرافیکی همساز با آن در بوردهای گرافیکی دیگر، مثل EGA برای هر پیکسل چهار سیگنال تولید شده، به صفحه‌نمایش فرستاده می‌شوند. این سیگنالها عبارتنداز: آبی، سبز، قرمز و شدت. این چهار سیگنال می‌توانند به طریقهٔ خاصی با یکدیگر ترکیب شده، تولید ۱۶ رنگ نمایند (جدول ۲-۱۶). هر ترکیبی از سیگنالهای آبی، سبز و روشن، تولید ۸ رنگ (رنگهای ۰ تا ۷) می‌نماید که اگر سیگنال شدت نیز به این سیگنالها اضافه شود، ۸ رنگ روشن بعدی (۸ تا ۱۵) تولید می‌شوند (جدول ۴-۱۶).

جدول ۱-۱۶ وجوه مختلف صفحه‌نمایش.

شماره حالت	حالت صفحه‌نمایش	ابعاد ماتریس در گرافیک	ابعادماتریس درمتن
۰	متن، سیاه و سفید	-	40 × 25
۱	متن، ۱۶ رنگ	-	40 × 25
۲	متن، سیاه و سفید	-	80 × 25
۳	متن، ۱۶ رنگ	-	80 × 25
۴	گرافیک، ۴ رنگ	320 × 200	40 × 25
۵	گرافیک، تک‌رنگ	320 × 200	40 × 25
۶	گرافیک، ۲ رنگ	640 × 200	80 × 25
۷	متن، سیاه و سفید	-	80×25
۸	PCjr ،گرافیک، ۱۶ رنگ	160 × 200	20 × 25
۹	PCjr ،گرافیک، ۱۶ رنگ	320 × 200	40 × 25
۱۰	گرافیک، ۴ رنگ	640 × 200	80 × 25
۱۱	رزرو شده		
۱۲	رزرو شده		
۱۳	گرافیک، ۱۶ رنگ	320 × 200	40 × 25
۱۴	گرافیک، ۱۶ رنگ	640 × 200	80 × 25
۱۵	گرافیک، ۲ رنگ	640 × 350	80 × 25
۱۶	گرافیک، ۱۶ رنگ	640 × 350	80 × 25
۱۷	گرافیک، ۲ رنگ	640 × 480	80 × 30
۱۸	گرافیک، ۱۶ رنگ	640 × 480	80 × 30
۱۹	گرافیک، ۲۵۶ رنگ	320×200	40×25

جدول ۲-۱۶ رنگهای زمینه در وجوه گرافیکی.

کد رنگ	رنگ	کد رنگ	رنگ
0	سیاه	8	خاکستری تیره
1	آبی	9	آبی روشن
2	سبز	10	سبز روشن
3	کبود	11	کبود روشن
4	قرمز	12	قرمز روشن
5	بنفش	13	بنفش روشن
6	قهوه‌ای	14	زرد
7	خاکستری روشن	15	سفید

جدول ۳-۱۶ رنگهای جعبه رنگ در بورد CGA.

رنگ / جعبه رنگ	۱	۲	۳
۰	سبز	قرمز	زرد
۱	کبود	بنفش	سفید
۲	سبز روشن	قرمز روشن	زرد
۳	کبود روشن	بنفش روشن	سفید

جدول ۴-۱۶ چگونگی تولید رنگ.

شماره رنگ	رنگ	شدت	قرمز	سبز	آبی
۰	سیاه	0	0	0	0
۱	آبی	0	0	0	1
۲	سبز	0	0	1	0
۳	کبود	0	1	0	0
۴	قرمز	0	1	1	0
۵	بنفش	0	1	0	1
۶	قهوه‌ای	0	1	1	1
۷	خاکستری روشن	1	0	0	0
۸	خاکستری تیره	1	0	0	0
۹	آبی روشن	1	0	0	1
۱۰	سبز روشن	1	0	1	0
۱۱	کبود روشن	1	0	1	1
۱۲	قرمز روشن	1	1	0	0
۱۳	بنفش روشن	1	1	0	1
۱۴	زرد	1	1	1	0
۱۵	سفید	1	1	1	1

از کجا شروع کنیم ؟

قبل از انجام هر گونه کار گرافیکی، باید صفحه‌نمایش در یک وجه گرافیکی مناسب قرار گیرد، و سپس رنگهایی جهت رسم اشکال گرافیکی انتخاب شود.

مثال ۱-۱۶

تابعی که با استفاده از وقفه 10H وجه گرافیکی را تعیین می‌کند.

```
#include <dos.h>
void mode(int mode_code)
{
    union REGS r ;
    r.h.al = mode_code ;
    r.h.ah = 0 ;
    int86(0x10, &r, &r) ;
}
```

مثال ۲-۱۶

تابعی که جعبه رنگی را انتخاب می‌کند.

```
#include <dos.h>
void palette(int pnum)
 {
    union REGS r ;
    r.h.bh = 1 ; /* code for mode 4 */
    r.h.bl = pnum ;
    r.h.ah = 0xb ;
    int86(0x10, &r, &r) ;
 }
```

نوشتن پیکسل‌ها

اشکال گرافیکی با در کنار هم نوشتن پیکسل‌ها نتیجه می‌شوند؛ لذا چگونگی نوشتن پیکسل در صفحه‌نمایش از اهمیت ویژه‌ای برخوردار است. برای نوشتن هر پیکسل در صفحه نمایش به دو طریق می‌توان عمل کرد:

۱. استفاده از روال‌های بایوس
۲. نوشتن پیکسل به طور مستقیم در حافظه نمایش.

روش اول. استفاده از وقفه‌های بایوس جهت نوشتن پیکسل از سرعت پایینی برخوردار است. در این کتاب از این روش استفاده نمی‌شود ولی جهت آشنایی با چگونگی عملکرد این روش، مثالی را در نظر می‌گیریم.

مثال ۳-۱۶

تابعی که با استفاده از وقفه‌های بایوس، پیکسلهایی را در صفحه‌نمایش رسم می‌کند.

```
#include "dos.h"
void write_point(int x, int y, int color)
 {
    union REGS r ;
    if(x < 0 || x > 199 || y < 0 || y > 319)
       return ;
```

```
    r.h.ah = 12 ;
    r.h.al = color ;
    r.x.dx = x ; /* row */
    r.x.cx = y ; /* column */
    int86(0x10, &r, &r) ;
}
```

روش دوم. سرعت نوشتن مستقیم پیکسل در حافظه نمایش، تقریباً ۱۵ برابر سرعت نوشتن آنها از طریق وقفه‌های بایوس است. لذا در این فصل برای نوشتن پیکسل از روش دوم استفاده می‌گردد. آدرس حافظه نمایش در بورد گرافیکی CGA و بعضی از وجوه گرافیکی همساز با آن در سایر بوردهای گرافیکی، از B8000000H شروع می‌شود. پیکسلهایی که در سطرهای زوج نوشته می‌شوند در آدرس B8000000H و پیکسلهایی که در سطرهای فرد نوشته می‌شوند در آدرس B8002000H (با اختلاف 2000H) قرار می‌گیرند.

مثال ۴-۱۶

تابعی که پیکسلهایی را به‌طور مستقیم در حافظه می‌نویسد(به توضیحی که بعد از تابع آمده است توجه نمایید).

```
void mempoint(int x , int y ,int color) {
    union mask {
      char c[2] ;
      int i ;
    }  bit_mask ;
    int i , index , bitpos ;
    unsigned char t ;
    char xor ;
    char far *ptr = (char far *) 0xB8000000 ;
    bit_mask.i=0xFF3F ; /* 11111111 00111111 */
    if(x < 0 || x > 199 || y < 0 || y > 319)
        return ;
    xor = color & 128 ;
    color = color & 127 ;
    bitpos = y % 4 ;
    color << = 2 * (3 - bitpos) ;
    bit_mask.i >>= 2 * bitpos ;
    index = x * 40 + (y / 4) ;
    if(x % 2)
        index += 8152 ;
        if(!xor) {
         t =* (ptr + index) & bit_mask.c[0] ;
         *(ptr + index) = t | color ;
    }
    else {
       t =* (ptr + index) | (char) 0 ;
       *(ptr + index) = t ^ color ;
     }
}
```

توضیحی در مورد تابع ()mempoint(مثال ۴-۱۶)

۱. پیداکردن بایتی از حافظه نمایش که پیکسل موردنظر باید در آنجا نوشته شود. تابع ()mempoint در وجه ۴ نوشته شده است. در این وجه برای نوشتن هر پیکسل از دو بیت استفاده می‌شود؛ لذا هر پیکسل می‌تواند به چهار ($۴=۲^۲$) رنگ نوشته شود و هر بایت می‌تواند شامل چهار پیکسل باشد. آدرس بایتی از حافظهٔ نمایش که پیکسل باید در آنجا نوشته شود از مجموعهٔ دستورات زیر حاصل می‌گردد:

```
index = x * 40 + y / 4
if   (x %2)
   index + = 8152
```

در دستور اول، x در ۴۰ (تعداد ستونها) ضرب شده، y بر ۴ (تعداد پیکسلها در بایت) تقسیم شده و حاصل آنها با یکدیگر جمع گردیده است. اگر سطری که پیکسل باید در آنجا نوشته شود، فرد باشد باید مقدار ثابتی به آدرس به دست آمده در دستور اول، اضافه گردد. این مقدار ثابت، فاصلهٔ بین B8002000H و B8000000H یعنی 2000H و یا ۸۱۹۲ است که از این مقدار، باید یک سطر (۴۰) کسر گردد؛ زیرا به عنوان مثال، اگر شماره سطری که پیکسل باید در آنجا نوشته شود، برابر با ۱ باشد، این سطر در ناحیهٔ دوم حافظه نمایش (جایی که پیکسل با شمارهٔ سطر فرد باید در آنجا نوشته شوند)، با صفر مشخص می‌گردد.

۲. تعیین محل پیکسل در بایتی که آدرس آن بایت در متغیر index قرار دارد و نوشتن آن در حافظه نمایش. پیکسل‌ها در یک بایت، از سمت چپ به راست قرار می‌گیرند. به عنوان مثال، پیکسل شمارهٔ صفر در بیتهای ۶ و ۷ و پیکسل شمارهٔ ۳ در بیتهای ۰ و ۱ قرار می‌گیرند. چون در هر بایت، ۴ عدد پیکسل قرار می‌گیرد، وقتی یکی از آنها تغییر می‌کند، مقادیر بقیه پیکسل‌ها باید حفظ گردد (تغییر نکند). بهترین راه برای حفظ این بیتها، انتخاب یک نقاب(mask) است، به طوری که همه بیتهای آن به جز بیتهایی که باید تغییر کند برابر با ۱ باشد. این بیتها با بیتهای موجود در حافظهٔ نمایش، AND می‌شود و سپس با اطلاعات جدید OR می‌گردد. این وضعیت با حالتی که در آن مقادیر قبلی با مقدار صفر OR می‌شود و سپس با مقدار جدید XOR گردد، متفاوت است. زیرا در این حالت کافی است که مقادیر قبلی با مقدار صفر OR شود و سپس با مقدار جدید OR شود. یادآوری می‌شود که عمل XOR موجب می‌شود تا نقطهٔ موردنظر همواره روشن باشد. برای پیداکردن موقعیت پیکسل در بایت، کافی است که باقیماندهٔ y بر ۴ محاسبه گردد (دستور bitpos = y % 4).

رسم خط

گرچه رسم خطوط عمودی و افقی ساده می‌باشد ولی رسم خطوط مورب از ظرافت خاصی برخوردار است. به عنوان مثال، رسم خط از نقطهٔ (۰ و ۰) تا (۱۲۰ و ۸۰) یک خط مورب است که پیداکردن نقاط بین این دو نقطه چندان آسان نیست. یک راه حل برای رسم خط، استفاده از رابطهٔ بین تغییرات X و Y است. برای روشن‌شدن مطلب فرض کنید که خطی از نقطهٔ (۰ و ۰) تا (۱۰ و ۵) باید رسم گردد. تغییرات X برابر با ۵ و تغییرات Y برابر با ۱۰ می‌باشد که نسبت آنها برابر با $\frac{۱}{۲}$ است. این نسبت، تغییرات X و Y را در حین رسم خط مشخص می‌کند. در این مورد خاص، تغییرات X به اندازهٔ نصف تغییرات Y است. این روش رسم خط، از سرعت کمی برخوردار است؛ مگر این که از پیش‌پردازنده استفاده گردد. معمولاً برنامه‌نویسان مبتدی از این روش استفاده می‌کنند.

یکی از بهترین روشها جهت رسم خط، استفاده از الگوریتم برِسنهام (Bresenham) است. گرچه این الگوریتم با نسبت فواصل X و Y سروکار دارد؛ ولی هیچگونه عمل تقسیم و یا ضرب ممیز شناور انجام نمی‌گیرد و به جای آن، نسبت بین تغییرات X و Y به طور غیرمستقیم از اعمال جمع و تفریق ایجاد می‌گردد. مسأله اصلی در الگوریتم برِسنهام نگهداری میزان خطای بین محل واقعی نقاط و محلی که در حافظه نمایش قرار می‌گیرند می‌باشد. این خطا بستگی به محدودیتهای سخت‌افزار دارد و بدین دلیل است که صفحه‌نمایش دارای دقت کافی جهت قرارگرفتن نقطه در محل واقعی خود نیست، ولی در بهترین محلِ ممکن قرار خواهد گرفت. در تابعی که برای رسم خط نوشته شده است دو متغیر به نامهای xerr و yerr وجود دارد که در هر بار تکرار تا رسم کامل خط، مقدار تغییرات X و Y به ترتیب به آنها افزوده می‌شود. وقتی که مقدار خطا به حد معینی رسید، مجدداً مقدار اولیه جدیدی به آن نسبت داده می‌شود.

مثال ۵-۱۶

تابعی که با استفاده از الگوریتم برسنهام خطی را در صفحه‌نمایش رسم می‌کند.

```
void line(int startx, int starty, int endx, int endy, intcolor)
{
      register int t , distance ;
      int xerr = 0 , yerr = 0, deltax, deltay ;
      int incx , incy ;
      deltax = endx - startx ;
      deltay = endy - starty ;
      if(deltax > 0)
         incx = 1 ;
       else if(deltax == 0)
         incx = 0 ;
      else
         incx =- 1 ;
      if(deltay > 0)
         incy = 1 ;
      else if(deltay == 0)
         incy = 0 ;
      else
         incy =- 1 ;
      deltax = abs(deltax) ;
      deltay = abs(deltay) ;
      if(deltax > deltay)
            distance = deltax ;
      else
            distance = deltay ;
      for(t = 0 ; t < distance + 1 ; t++)  {
            mempoint(startx , starty , color) ;
            xerr += deltax ;
            yerr += deltay ;
```

```
            if(xerr > distance) {
                xerr -= distance ;
                startx += incx ;
            }
            if(yerr > distance) {
                yerr -= distance ;
                starty += incy ;
            }
        } // end of for
}
```

رسم مستطیل و پرکردن آن

رسم مستطیل، با استفاده از تابع ()line که برای رسم خطوط مورد استفاده قرار می‌گیرد، بسیار ساده است.

مثال ۶-۱۶

تابعی که مستطیلی را رسم می‌کند.

```
void box(int startx, int starty, int endx, int endy, int color)
{
      line(startx , starty , endx , starty , color) ;
      line(startx , starty , startx , endy , color) ;
      line(startx , endy , endx , endy , color) ;
      line(endx , starty , endx , endy , color) ;
}
```

مثال ۷-۱۶

تابعی که برای پرکردن داخل یک مستطیل مورد استفاده قرار می‌گیرد. برای پرکردن یک مستطیل، کافی است کلیهٔ پیکسلهای داخل آن، با رنگ خاصی نوشته شود.

```
fill_box(int startx, int starty, int endx, int endy, int color)
{
      register int i , begin , end ;
      begin=startx < endx ? startx : endx ;
      end=startx > endx ? startx : endx ;
      for(i=begin ; i < end ; i++)
         line(i , starty , i , endy , color) ;
}
```

رسم دایره و بیضی و پرکردن آنها

ساده‌ترین و سریعترین راه برای رسم دایره و بیضی، استفاده از الگوریتم برِسنهام است. در رسم دایره و بیضی مشابه

الگوریتم رسم خطوط، نیازی به هیچ گونه اعمال ضرب و تقسیم ممیز شناور نیست؛ به جز در معادله نسبت که تعیین کنندهٔ دایره یا بیضی است. برای رسم دایره، مقداری به مختصات X و Y، براساس مقدار خطای بین آنها افزوده می‌شود که این مقدار با delta مشخص شده است. برای رسم دایره یا بیضی دو تابع به نامهای ()circle و ()plot_circle نوشته شده است. البته نوشتن الگوریتم رسم دایره و بیضی به صورت یک تابع ممکن است، ولی از خوانایی برنامه کاسته می‌شود. اگر متغیر asp_ratio برابر با ۱ باشد، دایره و اگر کوچکتر از ۱ باشد بیضی افقی وگرنه بیضی عمودی رسم خواهد شد.

مثال ۸-۱۶

توابعی که دایره یا بیضی را رسم می‌کنند.

```
double asp_ratio ;
circle(int x_center, int y_center, int radius, int color)
 {
    register int x , y , delta ;
    asp_ratio = 2.0 ;
    y = radius ;
    delta = 3 - 2 * radius ;
    for(x = 0 ; x < y ; ) {
        plot_circle(x,y,x_center,y_center,color) ;
       if(delta < 0)
            delta+=4*x+6 ;
       else {
            delta+=4*(x-y)+10 ;
            y -- ;
       } //end of else
       x++ ;
    } //end of for
    x = y ;
    if(y)
          plot_circle(x,y,x_center,y_center,color) ;
 } //end of circle
 //**************************
plot_circle(int x, int y, int x_center, int y_center, int color)
 {
      int startx , endx , endy , x1 , starty , y1 ;
      starty = y * asp_ratio ;
      endy = (y + 1) * asp_ratio ;
      startx = x * asp_ratio ;
      endx = (x + 1) * asp_ratio ;
      for(x1 = startx ; x1 < endx ; ++x1)  {
           mempoint(x1 + x_center , y + y_center , color) ;
           mempoint(x1 + x_center , y_center - y , color) ;
```

```
            mempoint(x_center - x1 , y_center - y , color) ;
            mempoint(x_center - x1 , y + y_center , color) ;
        }
        for(y1 = starty ; y1 < endy ; ++y1) {
          mempoint(y1 + x_center , x + y_center , color) ;
          mempoint(y1 + x_center , y_center - x , color) ;
          mempoint(x_center - y1 , y_center - x , color) ;
           mempoint(x_center - y1 , x + y_center , color) ;
        }
    }//end of plot_circle
```

مثال ۹-۱۶

تابعی که داخل بیضی را پر می‌کند. همانند مستطیل، می‌توان داخل دایره یا بیضی را نیز پر نمود. برای این منظور می‌توان دایره یا بیضی‌هایی با شعاعهای مختلف در داخل دایره و یا بیضی اصلی رسم نمود.

```
fill_circle(int x , int y , int r , int c)
{
    while(r) {
        circle(x, y, r, c) ;
        r -- ;
    }
}
```

مثال ۱۰-۱۶

برنامه‌ای که خط، دایره و مستطیلی را رسم کرده، داخل آنها را نیز پر می‌کند. در این برنامه، از توابعی که در مثالهای قبلی آمده است، استفاده گردید.

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
void mode(int), palette(int);
void box(int startx, int starty, int endx, int endy, int color);
void line(int startx, int starty, int endx, int endy, int color);
void fill_box(int startx, int starty, int endx, int endy, int color);
void circle(int x_center, int y_center, int radius, int color);
void plot_circle(int x, int y, int x_center, int y_center, int color);
void fill_circle(int x , int y , int r , int c);
void mempoint(int x , int y ,int color);
double asp_ratio ;
int main()
{
    mode(4) ;
```

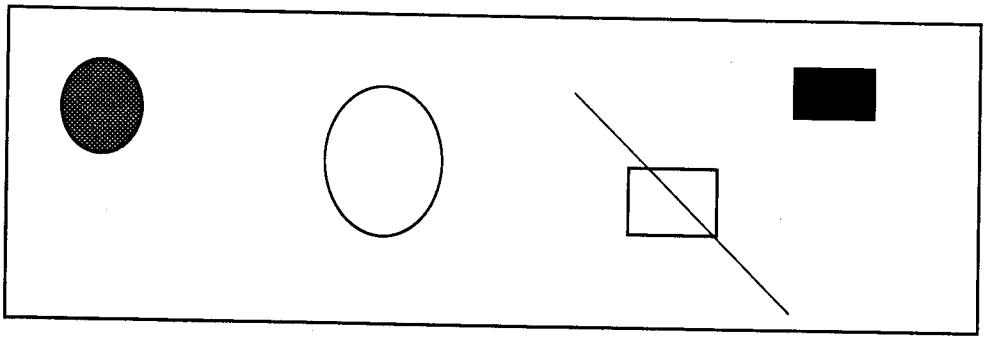
```
    palette(0) ;
    line(0, 0, 100, 100, 1) ;
    box(50, 50, 80, 90, 2) ;
    fill_box(100, 0, 120, 40, 3) ;
    circle(100, 160, 30, 2);
    fill_circle(150, 250, 20, 1) ;
    getch() ;
    mode(2) ;
    return 0;
}
//************************
void palette(int pnum)
 {
     union REGS r ;
     r.h.bh = 1 ;
     r.h.bl = pnum ;
     r.h.ah = 0xB ;
     int86(0x10, &r, &r) ;
  }
//******************
void mode(int mode_code)
  {
     union REGS r ;
     r.h.al = mode_code ;
     r.h.ah = 0 ;
     int86(0x10, &r, &r) ;
  }
//**************
void box(int startx, int starty, int endx, int endy, int color)
  {
     line(startx , starty , endx , starty , color) ;
     line(startx , starty , startx , endy , color) ;
     line(startx , endy , endx , endy , color) ;
     line(endx , starty , endx , endy , color) ;
  }
//********************
void line(int startx, int starty, int endx, int endy, int color)
   {
     register int t , distance ;
     int xerr = 0 , yerr = 0 , deltax , deltay ;
     int incx , incy ;
     deltax = endx - startx ;
     deltay = endy - starty ;
```

```
        if(deltax > 0)
          incx = 1 ;
        else if(deltax == 0)
          incx = 0 ;
        else
           incx =- 1 ;
        if(deltay > 0)
           incy = 1 ;
        else if(deltay == 0)
           incy = 0 ;
        else
           incy =- 1 ;
        deltax = abs(deltax) ;
        deltay = abs(deltay) ;
        if(deltax > deltay)
          distance = deltax ;
        else
          distance = deltay ;
        for(t = 0 ; t < distance + 1 ; t++)  {
          mempoint(startx , starty , color) ;
          xerr += deltax ;
          yerr += deltay ;
          if(xerr > distance) {
             xerr -= distance ;
             startx+=incx ;
          }
          if(yerr > distance) {
            yerr -= distance ;
            starty+=incy ;
          }
        }
  }
  //******************
void fill_box(int startx, int starty, int endx, int endy, int color)
   {
      register int i , begin , end ;
      begin = startx < endx ? startx : endx ;
      end = startx > endx ? startx : endx ;
      for(i = begin ; i < end ; i++)
        line(i , starty , i , endy , color) ;
   }
 //************************
void circle(int x_center, int y_center, int radius, int color)
```

```
{
    register int x , y , delta ;
    asp_ratio = 1.0 ;
    y = radius ;
    delta = 3 - 2 * radius ;
    for(x = 0 ; x < y ; ) {
       plot_circle(x, y, x_center, y_center, color) ;
       if(delta < 0)
          delta += 4 * x + 6 ;
       else {
           delta+=4*(x-y)+10 ;
           y -- ;
       }//end of else
       x++ ;
    } //end of for
    x = y ;
    if ( y )
    plot_circle(x, y, x_center, y_center, color) ;
}
//*********************
void plot_circle(int x, int y, int x_center, int y_center, int color)
  {
      int startx , endx , endy , x1 , starty , y1 ;
      starty = y * asp_ratio ;
      endy = (y + 1) * asp_ratio ;
      startx = x * asp_ratio ;
      endx = (x + 1) * asp_ratio ;
      for(x1 = startx ; x1 < endx ; ++x1) {
          mempoint(x1 + x_center , y + y_center , color) ;
          mempoint(x1 + x_center , y_center - y , color) ;
          mempoint(x_center - x1 , y_center - y , color) ;
          mempoint(x_center - x1 ,  y + y_center , color) ;
      }
     for(y1=starty ; y1 < endy ; ++y1) {
          mempoint(y1 + x_center , x + y_center , color) ;
          mempoint(y1 + x_center , y_center - x , color) ;
          mempoint(x_center - y1 , y_center - x , color) ;
          mempoint(x_center - y1 , x + y_center , color) ;
    }
}
//*********************
void fill_circle(int x , int y , int r , int c)
 {
```

```
    while(r) {
      circle(x, y, r, c) ;
      r -- ;
    }
  }
 //************************
void mempoint(int x , int y ,int color)
  {
    union mask {
      char c[2] ;
      int i ;
    } bit_mask ;
    int i , index , bitpos ;
    unsigned char t ;
    char xor ;
    char far *ptr = (char far *) 0xB8000000 ;
    bit_mask.i = 0xFF3F ; /* 11111111 00111111 */
    if(x < 0 || x > 199 || y < 0 || y > 319)
        return ;
    xor = color & 128 ;
    color = color & 127 ;
    bitpos = y % 4 ;
    color <<= 2 * (3 - bitpos) ;
    bit_mask.i >>= 2 * bitpos ;
    index = x * 40 + (y / 4) ;
    if(x % 2)
        index += 8152 ;
    if(!xor) {
        t =* (ptr + index) & bit_mask.c[0] ;
       *(ptr + index) = t | color ;
     }
    else {
      t =* (ptr + index) | (char) 0 ;
      *(ptr + index) = t ^ color ;
    }
 }
```

خروجی



ذخیره و بازیابی گرافیک

ذخیره کردن گرافیک در فایل و بازیابی آن، از مسایل مهمی است که در اکثر امور گرافیکی ضروری می‌باشد. با در نظر گرفتن این نکته که "آنچه که در صفحه‌نمایش ظاهر می‌شود در حافظه نمایش قرار دارد"، انتقال محتویات حافظه RAM به دیسک و برعکس، به سادگی امکان‌پذیر است.

مثال ۱۱-۱۶

توابعی برای ذخیره و بازیابی گرافیک از فایل . بزرگترین مسأله در ذخیره و بازیابی گرافیک این‌است‌که به کاربر اجازه داده شود تا نام فایلی را وارد کند؛ زیرا این کار باعث نابودی گرافیکِ موجود در صفحه‌نمایش می‌شود. برای رفع این مشکل، توابع save_pic() و load_pic() ، قبل از اخذ نام فایل، ۱۴ سطر بالای صفحه‌نمایش را در آرایه‌ای ذخیره می‌کند و سپس محتویات آن را پاک می‌کنند. پس از اخذ نام فایل، محتویات آرایه را به این ۱۴ سطر منتقل می‌کنند. این توابع با استفاده از تابع goto_xy() مکان‌نما را در صفحه‌نمایش حرکت می‌دهند. لیست این تابع را در مثال ۱۴-۱۶ ببینید.

```
void save_pic()
 {
     char fname[80] ;
     FILE *fp ;
     register int i , j ;
     char far *ptr = (char far *)0xB8000000 ;
     char far *temp ;
     unsigned char buf[14][80] ;
     temp = ptr ;
     for(i = 0 ; i < 14 ; i++)
       for(j = 0 ; j < 80 ; j += 2) {
          buf[i][j] =* temp ;
          buf[i][j+1] =* (temp + 8152) ;
          *temp = 0 ;
          *(temp + 8152) = 0 ;
```

```
            temp ++ ;
          }
        goto_xy(0, 0) ;
        printf("enter file nmae for put : ") ;
        gets(fname) ;
        fp = fopen(fname, "wb") ;
        if(fp == NULL) {
            printf("cannot open file.press a key ...") ;
            getch();
            return ;
        }
        temp = ptr ;
        /* restore the top of current screen */
        for(i = 0 ; i < 14 ; i++)
            for(j = 0 ; j < 80 ; j += 2) {
              *temp = buf[i][j] ;
              *(temp + 8152) = buf[i][j + 1] ;
              temp ++ ;
            }
        for(i=0 ; i < 8152 ; i++)  {
           putc(*ptr, fp) ; /* even byte */
           putc(*(ptr + 8152), fp) ;
           ptr++ ;
        }
        fclose(fp) ;
}
//*********************************
void load_pic()
{
        char fname[80] ;
        FILE *fp ;
        register int i , j ;
        char far *ptr = (char far *) 0xB8000000 ;
        char far *temp ;
        unsigned char buf[14][80] ;
        temp = ptr ;
        for(i = 0 ; i < 14 ; i++)
          for(j = 0 ; j < 80 ; j += 2) {
              buf[i][j]=*temp ;
              buf[i][j+1]=*(temp+8152) ;
              *temp=0 ;
              *(temp+8152)=0 ;
              temp ++ ;
```

```
        }
      goto_xy(0,0) ;
      printf("enter file nmae for get : ") ;
      gets(fname) ;
      fp = fopen(fname, "rb") ;
      if(fp == NULL) {
         printf("cannot open file .press a key ...") ;
         temp = ptr ;
         /* restore the top of current screen */
         for(i = 0 ; i < 14 ; i++)
           for(j = 0 ; j < 80 ; j += 2) {
             *temp=buf[i][j] ;
             *(temp + 8152) = buf[i][j + 1] ;
             temp ++ ;
           }
         return ;
      }//end of if
      /* load image from file */
      for(i = 0 ; i < 8152 ; i++)  {
         *ptr = getc(fp) ; /* even byte */
         *(ptr + 8152) = getc(fp) ; /* odd byte */
         ptr ++ ;
      }
      fclose(fp) ;
}
//*********************
```

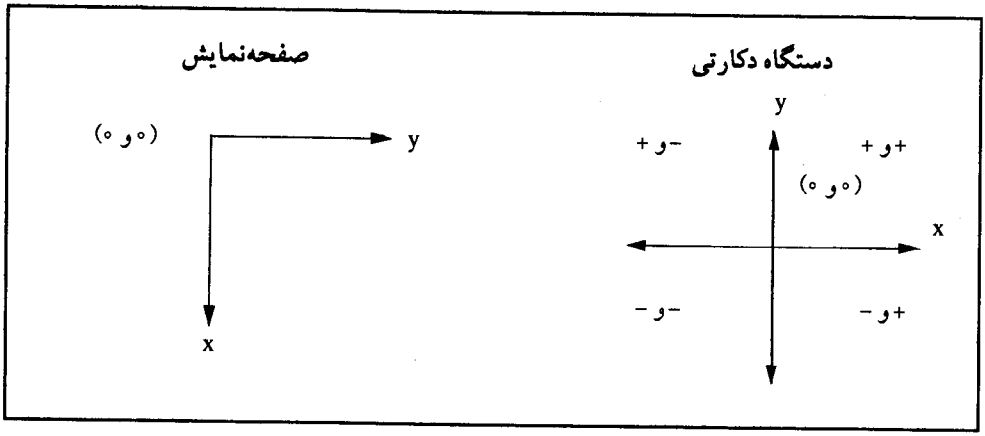
کپی و انتقال گرافیک از نقطه‌ای به نقطهٔ دیگر

گاهی لازم است که گرافیک موجود در قسمتی از صفحه‌نمایش را در قسمت دیگری از آن صفحه کپی نمود. برای این منظور کافی است محتویات قسمتی از حافظه که شکل گرافیکی در آنجا قرار دارد، خوانده شده در قسمت دیگری نوشته شود. برای انتقال گرافیک از قسمتی از صفحه‌نمایش به قسمت دیگر، کافی است محتویات قسمتی از حافظه که گرافیک در آنجا قرار دارد خوانده شده در قسمت دیگری نوشته شود و سپس محتویات قسمت قبلی پاک گردد.

مثال ۱۲-۱۶

توابعی که گرافیک را از ناحیه‌ای به ناحیه دیگر از صفحه‌نمایش کپی و انتقال می‌دهند. تابع ()copy گرافیک را کپی می‌کند، تابع ()read_point نقاط گرافیکی را می‌خواند و تابع ()move عمل انتقال را انجام می‌دهد.

```
void copy(int startx, int starty, int endx, int endy, int x, int y)
 {
    int i , j ;
    unsigned char c ;
    for(; startx <= endx ; startx ++ , x++)
      for(i = starty, j = y ; i < endy ; i++ , j++) {
         c = read_point(startx , i) ;
         mempoint(x, j, c) ;
      }
 }
 //*************************************
unsigned char read_point(int x , int y)
 {
    union mask  {
      char c[2] ;
      int i ;
    } bit_mask ;
    int i , index , bitpos ;
    unsigned char t ;
    char xor ;
    char far *ptr = (char far *) 0xB8000000 ;
    bit_mask.i = 0xFF3F ; /* 11111111 00111111 */
    if(x < 0 || x > 199 || y < 0 || y > 319)
       return ;
    bitpos = y % 4 ;
    bit_mask.i <<= 2 * (3 - bitpos) ;
    bit_mask.i >>= 2 * bitpos ;
    index = x * 40 + (y >> 2) ;
    if(x % 2)
       index += 8152 ;
    t = *(ptr + index) & bit_mask.c[0] ;
    t >>= 2 *( 3 - bitpos) ;
    return t ;
 }
 //*************************
void move(int startx, int starty , int endx, int endy, int x, int y)
 {
    int i , j ;
    unsigned char c ;
    for(; startx <= endx ; startx ++ , x++)
      for(i = starty, j = y ; i < endy ; i++ , j++) {
         c = read_point(startx , i) ;
         read_point(startx, i, 0) ;
         mempoint(x, j, c) ;
      }
 }
```



شکل ۱-۱۶ دستگاه دکارتی و صفحهٔ نمایش.

چرخش اشکال گرافیکی

چرخش اشکال گرافیکی در دستگاه دکارتی (دوبعدی) چندان مشکل نیست. یادآوری می‌شود که چرخش یک نقطه حول نقطه‌ای دیگر به اندازهٔ زاویه theta، طبق روابط زیر انجام می‌شود:

```
newx = oldx * cos (theta) - oldy * sin (theta)
newy = oldx * sin (theta) - oldy * cos (theta)
```

تنها مشکل در استفاده از این فرمولها این‌است‌که صفحه‌نمایش همانند یک دستگاه دکارتی نیست (شکل ۱-۱۶). برای حل این مشکل کافی است مبدأ جدیدی انتخاب کرده، مقادیر x و y را نسبت به آن مبدأ تنظیم نمود. هر یک از نقاط صفحه‌نمایش، می‌توانند به عنوان مبدأ انتخاب گردند ولی نقطه‌ای مناسب است که در مرکز شکل و یا نزدیک به مرکز شکل وجود داشته باشد.

مثال ۱۳-۱۶

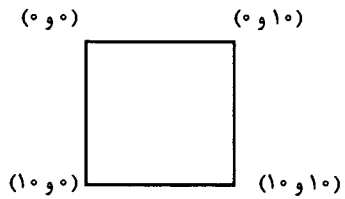
تابعی که یک نقطه را حول یک مبدأ می‌چرخاند.

```
void rotate_point(double theta, double *x, double *y,
     int x_org, int y_org) {
          double tx, ty ;
          tx = *x - x_org ;
          ty = *y - y_org ;
          *x = tx * cos(theta) - ty * sin(theta) ;
          *y = tx * sin(theta) - ty * cos(theta) ;
          *x += x_org ;
          *y += y_org ;
}
```

۳	۲	۱	۰	اندیس دوم ←
ey1	ex1	sy1	sx1	۰ اندیس اول
ey2	ex2	sy2	sx2	۱ ↓
ey3	ex3	sy3	sx3	۲
				.
				.
				.
eyn	exn	syn	sxn	n

شکل ۲-۱۶ ساختمان آرایه جهت تعریف شکل.

گرچه ()rotate_point موجب چرخش یک نقطه می‌شود، اما برای چرخش کامل یک شکل حول یک مبدأ، تابع دیگری نیز لازم است نوشته شود. برای این منظور هر شکل به صورت مجموعه‌ای از خطوط راست در نظر گرفته می‌شود. هر یک از این خطوط در یک آرایه دوبعدی ممیز شناور ذخیره می‌گردند. هر سطر از این آرایه، مختصات ابتدا و انتهای یک خط را نگهداری می‌کند. این بدین معنی است که اولین بعد آرایه، برابر با تعداد خطوط شکل و دومین بعد آرایه، برابر با ۴ خواهد بود. به عنوان مثال، آرایه ;[4] [10] double object می‌تواند شکلی با حداکثر ۱۰ خط را تعریف نماید.برای تعریف یک شکل باید مختصات ابتدا و انتهای هر خط را در یک آرایه قرار داد. به عنوان مثال، اگر شکل موردنظر در مستطیل زیر وجود داشته باشد، شکل ۲-۱۶ سازمان یک آرایه را جهت تعریف شکل نشان می‌دهد.



مستطیل حاوی شکل

مجموعه دستورات زیر، آرایه‌ای را جهت تعریف شکل (تعریف مستطیل) ایجاد می‌کند:

```
object[0][0]=0 ; object[0][1]=0 ;
object[0][0]=0 ; object[0][3]=10 ;
object[1][0]=0 ; object[1][1]=0 ;
object[1][0]=10 ; object[1][3]=10 ;
object[2][0]=10 ; object[2][1]=10 ;
object[2][0]=10 ; object[2][3]=0 ;
object[3][0]=10 ; object[3][1]=0 ;
object[3][0]=0 ; object[3][3]=0 ;
```

مثال ۱۴-۱۶

تابعی که موجب چرخش یک شکل می‌شود. برای چرخش در جهت عقربه‌های ساعت باید در پاسخ به درخواست تابع، حرف R و برای چرخش در جهت خلاف عقربه‌های ساعت، باید حرف L وارد شود.

```
void rotate_object(double ob[][4], double theta, int x, int y, int sides)
 {
   register int i , j ;
   double tempx , tempy ;
   char ch ;
   for(;;) {
     ch = getch() ;
     switch(tolower(ch)) {
       case 'l' :
          theta=theta<0 ? -theta : theta;
          break ;
       case 'r' :
          theta=theta>0 ? -theta : theta;
          break ;
       default : return ;
     } //end of switch
     for(j = 0 ; j < sides ; j++) {
        line((int)ob[j][0], (int)ob[j][1], (int)ob[j][2], (int)ob[j][3], 0);
        rotate_point(theta, &ob[j][0], &ob[j][1], x, y) ;
        rotate_point(theta, &ob[j][2], &ob[j][3],x,y) ;
        line((int) ob[j][0],(int)ob[j][1], (int) ob[j][2],(int) ob[j][3],2);
     } //end of for j =
 } //end of for(;;)
}
```

تابع rotate_object() ، شکل موردنظر را حول مبدأ (x, y) و به اندازهٔ زاویه theta می‌چرخاند. زاویه theta باید برحسب رادیان باشد. کوچکترین مقدار theta برابر با ۰/۰۱ است. برای انجام عمل چرخش، شکل از محل قبلی پاک شده در محل جدید کپی می‌شود. تعداد خطوط شکل باید در پارامتر sides قرار گیرد.

مثال ۱۵-۱۶

تابعی که شکل تعریف شده در یک آرایه را، در صفحه‌نمایش ظاهر می‌کند.

```
void display_object(double ob[][4], int sides)  {
   register int i ;
   for(i = 0 ; i < sides ; i++)
    line((int) ob[i][0], (int)ob[i][1],
       (int) ob[i][2], (int)ob[i][3], 2);
 }
```

مثال ۱۶-۱۶

برنامه‌ای که شکلهایی را رسم کرده، می‌چرخاند. پس از ظاهر شدن شکل، اگر کلید r وارد شود، شکل به سمت راست و اگر کلید l وارد شود، شکل به سمت چپ چرخانده می‌شود. با ورود هر کلید دیگری، برنامه خاتمه می‌یابد. در این برنامه از توابعی که شرح آنها گذشت استفاده می‌شود.

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <math.h>
void mode(int mode);
void palette(int pnum);
void line(int startx, int starty, int endx, int endy, int color);
void mempoint(int x , int y ,int color);
void rotate_point(double theta, double *x, double *y, int x_org, int y_org);
void rotate_object(double ob[][4], double theta, int x, int y, int sides);
void display_object(double ob[][4], int sides);
double house[][4]= {
   120, 120, 120, 200 , /* house */
   120, 200, 80,  200 ,
   80,  120, 80,  200 ,
   80,  120, 120, 120 ,
   60,  160, 80,  120 ,  /* roof */
   60,  160, 80,  200 ,
   120, 155, 100, 155 , /* door */
   100, 155, 100, 165 ,
   100, 165, 120, 165 ,
   90,  130, 100, 130 , /* window */
   90,  130, 90,  140 ,
   100, 130, 100, 140 ,
   90,  140, 100, 140 ,
   90,  180, 100, 180 ,
   90,  180, 90,  190 ,
   100, 180, 100, 190 ,
   90,  190, 100, 190
 } ;
int main()
{
    union k {
        char c[2] ;
        int i ;
    } key ;
    mode(4) ;
    palette(0) ;
    /* drow a box around the house */
    line(30, 70, 30, 260, 2) ;
    line(160, 70, 160, 260, 2) ;
```

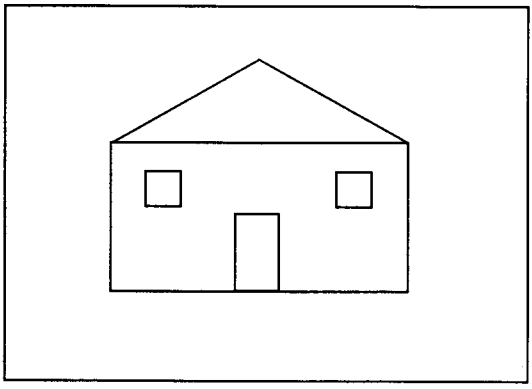
```
    line(30, 70, 160, 70, 2) ;
    line(30, 260, 160, 260, 2) ;
    display_object(house, 17) ;
    rotate_object(house, 0.25, 90, 160, 17);
    mode(3) ;
    return 0;
  }
 //**********************
void mode(int mode_code)
    {
      union REGS r ;
      r.h.al=mode_code ;
      r.h.ah=0 ;
      int86(0x10,&r,&r) ;
    }
 //**********************
void palette(int pnum)
  {
      union REGS r ;
      r.h.bh = 1 ;
      r.h.bl = pnum ;
      r.h.ah = 0xB ;
      int86(0x10, &r, &r) ;
  }
 //**********************
void line(int startx, int starty, int endx, int endy, int color)
    {
      register int t, distance ;
      int xerr = 0, yerr = 0, deltax, deltay ;
      int incx , incy ;
      deltax = endx - startx ;
      deltay = endy - starty ;
      if(deltax > 0)
         incx = 1 ;
      else if(deltax == 0)
         incx = 0 ;
      else
         incx =- 1 ;
      if(deltay > 0)
         incy = 1 ;
      else if(deltay == 0)
         incy = 0 ;
      else
         incy =- 1 ;
      deltax = abs(deltax) ;
      deltay = abs(deltay) ;
      if(deltax > deltay)
         distance = deltax ;
```

```
        else
           distance = deltay ;
        for(t = 0 ; t < distance + 1 ; t++)  {
            mempoint(startx , starty , color) ;
            xerr += deltax ;
            yerr += deltay ;
            if(xerr > distance) {
              xerr -= distance ;
              startx += incx ;
            }
            if(yerr > distance) {
               yerr -= distance ;
               starty += incy ;
            }
         }//end of for
}
//*********************
void mempoint(int x , int y ,int color)
{
     union mask {
        char c[2] ;
        int i ;
     } bit_mask ;
     int i, index, bitpos ;
     unsigned char t ;
     char xor ;
     char far *ptr = (char far *) 0xB8000000 ;
     bit_mask.i = 0xFF3F ; /* 11111111 00111111 */
     if(x < 0 || x > 199 || y < 0 || y > 319)
        return ;
     xor = color & 128 ;
     color = color & 127 ;
     bitpos = y % 4 ;
     color <<= 2 * (3 - bitpos) ;
     bit_mask.i >>= 2 * bitpos ;
     index = x * 40 + (y / 4) ;
     if(x % 2)   index += 8152 ;
     if(!xor) {
          t = *(ptr + index) & bit_mask.c[0] ;
          *(ptr + index) = t | color ;
     }
     else  {
        t= *(ptr + index) | (char) 0 ;
        *(ptr + index) = t ^ color ;
     }
}
//************************
void rotate_point(double theta, double *x, double *y, int x_org, int y_org)
```

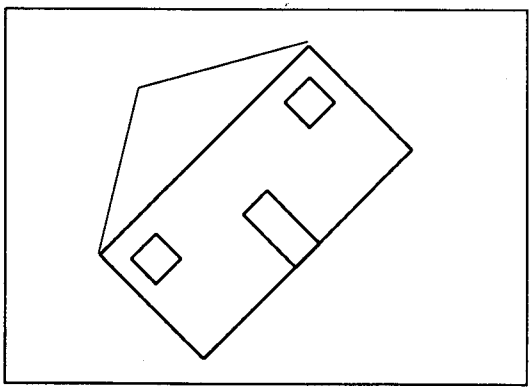
```
{
    double tx, ty ;
    tx = *x - x_org ;
    ty = *y - y_org ;
    /* rotate */
    *x = tx * cos(theta) - ty * sin(theta) ;
    *y = tx * sin(theta) - ty * cos(theta) ;
    *x += x_org ;
    *y += y_org ;
}
//************************
void rotate_object(double ob[][4], double theta, int x, int y, int sides)
 {
    register int i , j ;
    double tempx , tempy ;
    char ch ;
    puts("r = right, l = left, otherkey = exit");
    for(;;)  {
        ch = getch() ;
        switch(tolower(ch))  {
          case 'l' :
            theta = theta < 0 ? - theta : theta ;
            break ;
          case 'r' :
            theta = theta > 0 ? - theta : theta ;
            break ;
          default : return ;
        }
        for(j = 0 ; j < sides ; j++) {
          /* erase old line */
          line((int) ob[j][0] , (int)ob[j][1] ,
            (int) ob[j][2] , (int)ob[j][3] , 0) ;
          rotate_point(theta , &ob[j][0], &ob[j][1], x, y) ;
          rotate_point(theta , &ob[j][2], &ob[j][3], x, y) ;
          line((int) ob[j][0] ,(int) ob[j][1] ,
            (int) ob[j][2] , (int) ob[j][3], 2) ;
        } //end of for j =
    }//end of for (;;)
}
//****************************
void display_object(double ob[][4], int sides)
{
    register int i ;
    for(i = 0 ; i < sides ; i++)
        line((int) ob[i][0] , (int)ob[i][1] ,
          (int) ob[i][2] , (int)ob[i][3] , 2) ;
}
```

خروجی مثال ۱۶-۱۶

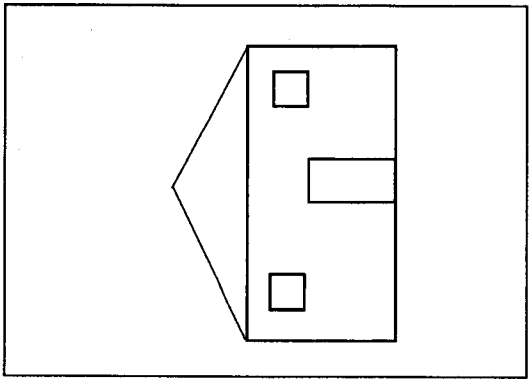
a



b



c



مثال ۱۶-۱۷

در این مثال نمونه‌ای از یک طراحی ساده معرفی گردید تا شروعی برای طراحی CAD باشد. با ترکیب توابعی که تاکنون بررسی شده‌اند، می‌توان برنامه‌ای نوشت که کلیهٔ اعمال زیر را انجام دهد:

۱. رسم خطوط.
۲. رسم مستطیل.
۳. پرکردن مستطیل.
۴. رسم دایره (بیضی).
۵. پرکردن دایره (بیضی).
۶. انتخاب رنگ و جعبه رنگ.
۷. خاموش و روشن کردن مکان‌نما.
۸. تعیین پارامتر سرعت حرکت اشکال.
۹. ذخیره کردن گرافیک در فایل.
۱۰. بازیابی گرافیک از فایل.
۱۱. چرخش شکل در حول هر نقطهٔ دلخواه.
۱۲. کپی و انتقال شکل.

توابعی که در این برنامه وجود دارند، تقریباً همان توابعی هستند که تاکنون در این فصل بررسی شده‌اند، بنابراین نیازی به تشریح وظایف آنها نیست.

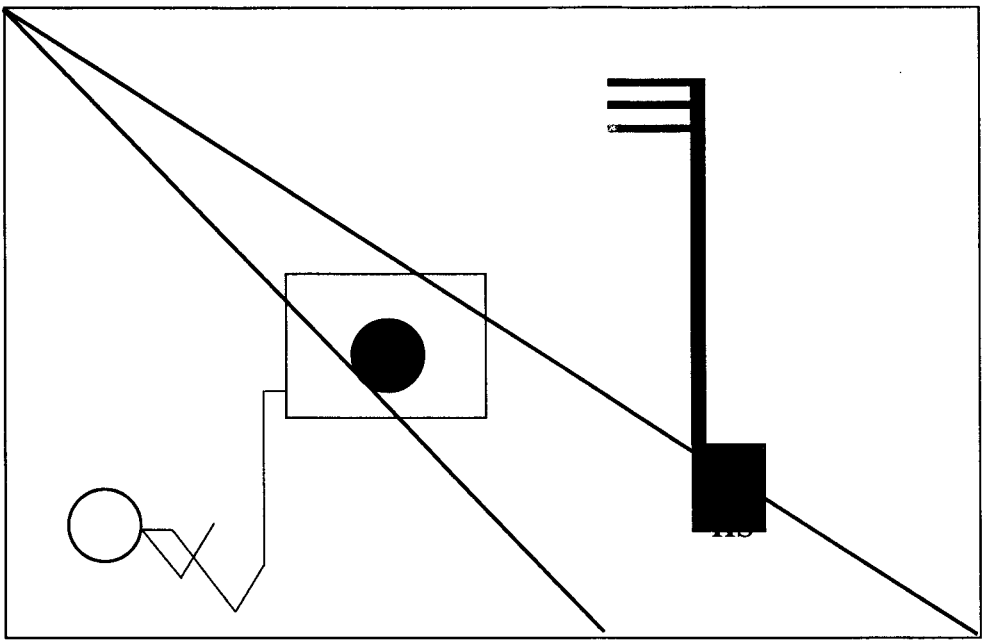
برنامه به این صورت عمل می‌کند: صفحه‌نمایش در حالت گرافیکی و در وجه ۴ قرار می‌گیرد. جعبه رنگ شماره صفر انتخاب می‌شود و مکان‌نما در گوشه بالای سمت چپ قرار می‌گیرد. با کلیدهای (→ ↑ ← ↓) می‌توان مکان‌نما را حرکت داد. با هر بار فشار دادن این کلیدها، مکان‌نما به اندازه یک پیکسل حرکت می‌کند. اگر کلید F2 را فشار دهید، از آن پس، با فشردن این کلیدها، مکان‌نما به اندازه ۵ پیکسل حرکت می‌کند. اگر کلید F1 فشار داده شود، اثر کلید F2 خنثی می‌شود. با وارد کردن اعداد ۰ تا ۳ می‌توان رنگ گرافیک را تغییر داد. با فشردن کلید O مکان‌نما ظاهر یا ناپدید می‌شود. کلیدهای Home، Pgup، Pgdn و End مکان‌نما را با زاویه ۴۵ درجه حرکت می‌دهند.

برنامه، دستوراتی را برای رسم کادرها، پرکردن داخل کادرها، رسم خطوط، رسم دایره‌ها، پرکردن دایره‌ها، کپی یا انتقال تصاویر، ذخیره و بازیابی محتویات صفحه‌نمایش، و تعریف و چرخش تصاویر دارد.

برای رسم خطوط، کادرها، دایره‌ها باید دو مختصات را تعریف کنید. برای رسم کادرها و پرکردن آنها، دوگوشه مقابل (دو سطر قطر) را مشخص کنید. برای رسم خطوط، نقاط ابتدا و انتهای خط را مشخص کنید. برای رسم دایره و پرکردن داخل آن، مرکز و نقطه‌ای از دایره را انتخاب کنید. برای انتخاب نقطه، مکان‌نما را به آن محل منتقل کرده، کلید enter را فشار دهید. به‌عنوان مثال، برای تعیین نقاط انتهایی یک خط، مکان‌نما را به نقطه‌ای که خط باید شروع شود منتقل کنید و کلید enter را فشار دهید. سپس مکان‌نما را به نقطه انتهای خط منتقل کرده، کلید enter را فشار دهید. با فشردن کلید enter متغیرهای startx، starty، endx و endy مقدار می‌گیرند و به‌عنوان آرگومانها به توابع مناسبی منتقل می‌شوند. پس از تعیین نقاط، با وارد کردن حرف B کادر رسم می‌شود. با وارد کردن F کادر توپر رسم می‌شود، با وارد کردن L خط رسم می‌شود، با وارد کردن C دایره رسم می‌شود.

برای انتقال یا کپی ناحیه‌ای از صفحه، باید گوشه بالا سمت چپ و گوشه پایین سمت راست را با کلید enter انتخاب کنید. سپس مکان‌نما را به جایی که عمل انتقال یا کپی باید انجام شود منتقل کرده، با M عمل انتقال و با X عمل کپی را انجام دهید.

برای چرخش تصویر، باید آن را با حرف D انتخاب کنید. سپس با استفاده از کلید entert هر قطعه خط را در شیء انتخاب کنید. یعنی، شکل را ردیابی کنید و در ابتدا و انتها هر خط تصویر، کلید enter را فشار دهید. پس از تعریف شیء، کلید F1 را فشار دهید تا فرایند تعریف شکل خاتمه یابد. فرایند تعریف تصویر توسط تابع define_objectl انجام می‌شود. پس از تعریف شکل، با وارد کردن کلید A و با استفاده از کلید L و R می‌توان شکل را حول مبدائی که با موقعیت مکان‌نما مشخص می‌شود، چرخاند. برای خاتمهٔ عمل چرخش، کلیدهایی غیراز L، A را وارد کنید. با وارد کردن حرف Q برنامه خاتمه می‌یابد. نمونه‌ای از عملکرد برنامه در زیر مشاهده می‌شود:



لیست برنامهٔ مثال ۱۷-۱۶ :

```
#include <dos.h>
#include <ctype.h>
#include <math.h>
#include <bios.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define NUM_SIDES 20
void mode(int mode);
void palette(int pnum);
void line(int startx, int starty, int endx, int endy, int color);
void mempoint(int x , int y ,int color);
void rotate_point(double theta, double *x, double *y, int x_org, int y_org);
void rotate_object(double ob[][4], double theta, int x, int y, int sides);
void display_object(double ob[][4], int sides);
void box(int startx, int starty, int endx, int endy, int color);
void fill_box(int startx, int starty, int endx, int endy, int color);
void circle(int x_center, int y_center, int radius, int color);
void plot_circle(int x, int y, int x_center, int y_center, int color);
void fill_circle(int x , int y , int r , int c);
void save_pic();
void load_pic();
void copy(int startx, int starty, int endx, int endy, int x, int y);
```

```
void move(int startx, int starty, int endx, int endy, int x, int y);
void xhairs(int x, int y, int c);
unsigned char read_point(int x, int y);
void goto_xy(int x, int y);
int define_object(double ob[][4], int x, int y);
double object [NUM_SIDES][4] ;
double asp_ratio ;
int main ()
 {
     union k {
       char c[2] ;
       int i ;
     } key ;
     int x = 10 , y = 10, cc = 2, i ;
     int on_flag = 1, pal_num = 1 ;
     int startx = 0, starty = 0, endx = 0, endy = 0 ;
     int first_point = 1 , inc = 1 , sides = 0 ;
     mode(4) ;
     palette(0) ;
     xhairs(x, y, 1) ;
     do {
       key.i = bioskey(0) ;
       if(!key.c[0])
          switch(key.c[1])  {
          case 75 :
             if(on_flag)
                 line(x, y, x, y - inc, cc) ;
             y -= inc ;
             break ;
          case 77 :
             if(on_flag)
                 line(x, y, x, y + inc, cc) ;
             y += inc ;
             break ;
          case 72 :
             if(on_flag)
                line(x, y, x - inc, y, cc) ;
             x -= inc ;
             break ;
          case 80 :
             if(on_flag)
                line(x, y,x + inc, y, cc) ;
             x += inc ;
             break ;
          case 71 :
```

```
            if(on_flag)
              line(x, y, x - inc, y - inc, cc) ;
            x -= inc ;
            y -= inc ;
            break ;
          case 73 :
            if(on_flag)
              line(x, y, x - inc, y + inc, cc) ;
            x -= inc ;
            y += inc ;
            break ;
          case 79 :
            if(on_flag)
              line(x, y, x + inc, y - inc, cc) ;
            x += inc ;
            y -= inc ;
            break ;
          case 81 :
            if(on_flag)
              line(x, y, x + inc, y + inc, cc) ;
            x += inc ;
            y += inc ;
            break ;
          case 59 :
            inc = 1 ;
            break ;
          case 60 :
            inc = 5 ;
            break ;
        }// end of switch
        else  switch(tolower(key.c[0]))
        {
          case 'o' : on_flag = !on_flag ;
            break ;
          case '1' : cc = 1 ;
            break ;
          case '2' : cc = 2 ;
            break ;
          case '3' : cc = 3 ;
            break ;
          case '0' : cc = 0 ;
            break ;
          case 'b' : box(startx, starty, endx, endy, cc) ;
            break ;
          case 'f' : fill_box(startx, starty, endx, endy, cc) ;
```

```
                break ;
            case 'l' :line(startx, starty, endx, endy, cc) ;
                break ;
            case 'c' : circle(startx, starty, endy - starty, cc) ;
                break ;
            case 's' : save_pic() ;
                break ;
            case 'r' : load_pic() ;
                break ;
            case 'm' : move(startx, starty, endx, endy, x, y) ;
                break ;
            case 'x' : copy(startx, starty, endx, endy, x, y) ;
                break ;
            case 'd' : sides = define_object(object, x, y) ;
                break ;
            case 'a' :rotate_object(object, 0.05, x, y, sides) ;
                break ;
            case '\r' :
                if(first_point) {
                    startx = x ;
                   starty = y ;
                } //end of if
                else {
                    endx = x ;
                    endy = y ;
                }//end of else
                first_point = !first_point ;
                break ;
            case 'p': pal_num = pal_num == 1 ? 2 : 1 ;
                palette(pal_num) ;
        } //end of switch
    } while(key.c[0] != 'q') ;
    getch() ;
    mode(2) ;
    return 0;
} //end of main
//**********************
void mode(int mode_code)
  {
     union REGS r ;
     r.h.al = mode_code ;
     r.h.ah = 0 ;
     int86(0x10, &r, &r) ;
  }
//*************************
```

```
void palette(int pnum)
  {
      union REGS r ;
      r.h.bh = 1 ;
      r.h.bl = pnum ;
      r.h.ah = 0xB ;
      int86(0x10, &r, &r) ;
  }
//*********************
void mempoint(int x , int y , int color)
 {
     union mask {
         char c[2] ;
         int i ;
     } bit_mask ;
     int i , index , bitpos ;
     unsigned char t ;
     char xor ;
     char far *ptr = (char far *) 0xB8000000 ;
     bit_mask.i = 0xFF3F ; /* 11111111 00111111 */
     if(x < 0 || x > 199 || y < 0 || y > 319)
          return ;
     xor = color & 128 ;
     color = color & 127 ;
     bitpos = y % 4 ;
     color <<= 2 * (3 - bitpos) ;
     bit_mask.i>>= 2 * bitpos ;
     index = x * 40 +( y / 4) ;
     if(x % 2)
          index += 8152 ;
     if(!xor) {
        t = *(ptr + index) & bit_mask.c[0] ;
        *(ptr + index) = t | color ;
      }
     else  {
         t = *(ptr + index) | (char) 0 ;
         *(ptr + index) = t ^ color ;
     }
 }
//*********************
void line(int startx, int starty, int endx, int endy, int color)
 {
      register int t , distance ;
      int xerr = 0 , yerr = 0 , deltax , deltay ;
      int incx , incy ;
```

```
        deltax = endx - startx ;
        deltay = endy - starty ;
        if(deltax > 0)
          incx = 1 ;
        else if(deltax == 0)
          incx = 0 ;
        else
          incx =- 1 ;
        if(deltay > 0)
          incy = 1 ;
        else if(deltay == 0)
          incy = 0 ;
        else
          incy =- 1 ;
        deltax = abs(deltax) ;
        deltay=abs(deltay) ;
        if(deltax > deltay)
             distance = deltax ;
        else
          distance = deltay ;
        for(t = 0 ; t < distance + 1 ; t++) {
             mempoint(startx , starty , color) ;
            xerr += deltax ;
            yerr += deltay ;
            if(xerr > distance) {
                xerr-=distance ;
              startx+=incx ;
            }
            if(yerr > distance) {
              yerr -= distance ;
              starty += incy ;
            }
        }
    }
    //*****************************
void box(int startx, int starty, int endx, int endy, int color)
    {
        line(startx , starty , endx , starty , color) ;
        line(startx , starty , startx , endy , color) ;
        line(startx , endy , endx , endy , color) ;
        line(endx , starty , endx , endy , color) ;
    }
    //*****************************
void fill_box(int startx, int starty, int endx, int endy, int color)
    {
```

```
    register int i, begin, end ;
    begin = startx < endx ? startx : endx ;
    end = startx > endx ? startx : endx ;
    for(i = begin ; i < end ; i++)
        line(i , starty , i , endy , color) ;
  }
//*******************
void circle(int x_center, int y_center, int radius, int color)
 {
   register int x , y , delta ;
   asp_ratio = 2.0 ;
   y = radius ;
   delta = 3- 2 * radius ;
   for(x = 0 ; x < y ; ) {
       plot_circle(x, y, x_center, y_center, color) ;
       if(delta < 0)
          delta += 4 * x + 6 ;
       else {
          delta += 4 * (x - y) + 10 ;
          y -- ;
       }
       x++ ;
   }//end of for
   x = y ;
   if(y)
      plot_circle(x, y, x_center, y_center, color) ;
 }
 //************************
void plot_circle(int x, int y, int x_center, int y_center, int color)
 {
     int startx , endx , endy , x1 , starty , y1 ;
     starty =y * asp_ratio ;
     endy = (y + 1) * asp_ratio ;
     startx = x * asp_ratio ;
     endx = (x + 1) * asp_ratio ;
     for(x1 = startx ; x1 < endx ; ++x1) {
        mempoint(x1 + x_center, y + y_center, color) ;
        mempoint(x1 + x_center, y_center - y, color) ;
        mempoint(x_center - x1, y_center - y, color) ;
        mempoint(x_center - x1 , y + y_center, color) ;
     }
     for(y1 = starty; y1 < endy ; ++y1) {
         mempoint(y1 + x_center, x+y_center, color) ;
         mempoint(y1 + x_center, y_center - x, color) ;
         mempoint(x_center - y1, y_center - x, color) ;
```

```
            mempoint(x_center - y1, x+y_center, color) ;
        }
    }
    //*****************************
void fill_circle(int x , int y , int r , int c)
    {
        while(r) {
            circle(x,y,r,c) ;
            r -- ;
        }
    }
    //*************************
void save_pic()
    {
        char fname[80] ;
        FILE *fp ;
        register int i , j ;
        char far *ptr =( char far *)0xB8000000 ;
        char far *temp ;
        unsigned char buf[14][80] ;
        temp = ptr ;
        for(i = 0 ; i < 14 ; i++)
            for(j = 0 ; j < 80 ; j += 2) {
                buf[i][j]=*temp ;
                buf[i][j+1]=*(temp+8152) ;
                *temp = 0 ;
                *(temp + 8152) = 0 ;
                temp ++ ;
            }
        gotoxy(0,0) ;
        printf("enter file nmae for put : ") ;
        gets(fname) ;
        fp = fopen(fname, "wb") ;
        if(fp == NULL) {
            printf("cannot open file .press a key ...") ;
            getch();
            return ;
        }
        temp = ptr ;
        /* restore the top of current screen */
        for(i = 0 ; i < 14 ; i++)
            for(j = 0 ; j < 80 ; j += 2) {
                *temp = buf[i][j] ;
                *(temp + 8152) = buf[i][j + 1] ;
                temp ++ ;
```

```
    }
    for(i=0 ; i < 8152 ; i++) {
        putc(*ptr, fp) ; /* even byte */
        putc(*(ptr + 8152), fp) ;
        ptr++ ;
    }
    fclose(fp) ;
 }
//**********************
void load_plc()
{
    char fname[80] ;
    FILE *fp ;
    register int i , j ;
    char far *ptr = (char far *) 0xB8000000 ;
    char far *temp ;
    unsigned char buf[14][80] ;
    temp = ptr ;
    for(i = 0 ; i < 14 ; i++)
       for(j = 0 ; j < 80 ; j += 2) {
          buf[i][j] =* temp ;
          buf[i][j+1] =* (temp + 8152) ;
          *temp = 0 ;
          *(temp + 8152) = 0 ;
          temp ++ ;
       }
    gotoxy(0, 0) ;
    printf("enter file nmae for get : ") ;
    gets(fname) ;
    fp = fopen(fname,"rb") ;
    if(fp == NULL) {
        printf("cannot open file .press a key ...") ;
        temp = ptr ;
        /* restore the top of current screen */
        for(i=0 ; i < 14 ; i++)
           for(j = 0 ; j < 80 ; j += 2) {
               *temp = buf[i][j] ;
               *(temp + 8152) = buf[i][j + 1] ;
               temp ++ ;
           }
        return ;
    }//end of if
    /* load image from file */
    for(i =0  ; i < 8152 ; i++)  {
       *ptr = getc(fp) ; /* even byte */
```

```
            *(ptr + 8152) = getc(fp) ; /* odd byte */
            ptr ++ ;
        }
        fclose(fp) ;
    }
    //************************
    void copy(int startx, int starty, int endx, int endy, int x, int y)
    {
        int i , j ;
        unsigned char c ;
        for(; startx <= endx ; startx ++ , x++)
            for(i = starty, j = y ; i < endy ; i++ , j++) {
                c = read_point(startx, i) ;
                mempoint(x, j, c) ;
            }
    }
    //*********************
    void move(int startx, int starty, int endx, int endy, int x, int y)
    {
        int i , j ;
        unsigned char c ;
        for(; startx <= endx ; startx ++ , x++)
            for(i = starty, j = y ; i < endy ; i++ , j++) {
              c = read_point(startx , i) ;
              //   read_point(startx, i, 0) ;
              mempoint(x, j, c) ;
            }
    }
    //************************
    void xhairs(int x, int y, int c)
    {
        line(x-3, y, x + 3, y, c) ;
        line(x, y + 3, x, y - 3, c) ;
    }
    //*************************
    unsigned char read_point(int x, int y)
    {
        union mask {
          char c[2] ;
          int i ;
        } bit_mask ;
        int i, index, bitpos ;
        unsigned char t ;
        char xor ;
        char far *ptr = (char far *) 0xB8000000 ;
```

```
    bit_mask.i = 0xFF3F ; /* 11111111 00111111 */
    if(x < 0 || x > 199 || y < 0 || y > 319)
       return -1;
    bitpos = y % 4 ;
    bit_mask.i <<= 2 * (3 - bitpos) ;
    bit_mask.i >>= 2 * bitpos ;
    index = x * 40 +( y >> 2) ;
    if(x % 2)
          index += 8152 ;
     t = *(ptr + index) & bit_mask.c[0] ;
     t >>= 2.* (3 - bitpos) ;
     return t ;
}
//***************************
void goto_xy(int x, int y)
 {
     union REGS r ;
     r.h.ah = 2 ;
     r.h.dl = y ;
     r.h.dh = x ;
     r.h.bh = 0 ;
     int86(0x10 , &r , &r) ;
 }
//***************************
void rotate_point(double theta, double *x, double *y, int x_org, int y_org)
 {
     double tx, ty ;
     tx =* x - x_org ;
     ty =* y - y_org ;
     *x = tx * cos(theta) - ty * sin(theta) ;
     *y = tx * sin(theta) - ty * cos(theta) ;
     *x += x_org ;
     *y += y_org ;
 }
//************************
void rotate_object(double ob[][4], double theta, int x, int y, int sides)
{
     register int i , j ;
     double tempx , tempy ;
     char ch ;
     for(;;) {
        ch = getch() ;
        switch(ch) {
          case 'l' :
             theta = theta < 0 ? -theta : theta ;
```

```
                break ;
            case 'r' :
                theta = theta > 0 ? -theta : theta ;
                break ;
            default : return ;
          }//end of switch
          for(j = 0 ; j < sides ; j++)  {
                line((int) ob[j][0] , (int)ob[j][1] ,
                (int) ob[j][2] , (int)ob[j][3] , 0) ;
                rotate_point(theta , &ob[j][2], &ob[j][3], x, y) ;
                line((int) ob[j][0] , (int) ob[j][1] ,
                (int) ob[j][2] , (int) ob[j][3] , 2) ;
          }//end of for j =
        }//end of for (;;)
 }
 //*******************************
void display_object(double ob[][4], int sides)
 {
      register int i ;
      for(i=0 ; i < sides ; i++)
         line((int) ob[i][0] , (int)ob[i][1] ,
          (int) ob[i][2] , (int)ob[i][3] , 2) ;
  }
 //*************************
int define_object(double ob[][4], int x, int y)
 {
      union k {
          char c[2] ;
          int i ;
      } key ;
      register int i , j ;
      char far *ptr = (char far *) 0xB8000000 ;
      char far *temp ;
      unsigned char buf[14][80] ;
      int sides = 0 ;
      temp = ptr ;
      for(i = 0 ; i < 14 ; i++)
        for(j = 0 ; j < 80 ; j += 2) {
            buf[i][j] = *temp ;
            buf[i][j+1] = *(temp + 8152) ;
            *temp=0 ;
            *(temp+8152)=0 ;
            temp ++ ;
        }//end of for j =
      i = 0 ;
```

```
    do {
        goto_xy(0, 0) ;
        printf("define sides %d", sides + 1) ;
        if(i == 0)
          printf("enter first endpoint") ;
        else
          printf("enter second endpoints") ;
        key.i = bioskey(0) ;
       if(key.c[0] == 13) {
          ob[sides][i++] = (double) x ;
          ob[sides][i++] = (double) y ;
          if(i == 4) {
              i = 0 ;
              sides ++ ;
          } //end of if
        }//end of if
        if(!key.c[0])
          switch(key.c[1])  {
              case 75 : y -= 1 ; break ;
              case 77 : y += 1 ; break ;
              case 72 : x =- 1 ; break ;
              case 80 : x += 1 ; break ;
              case 71 : x -= 1 ; y -= 1 ; break ;
              case 73 : x -= 1 ; y += 1 ; break ;
              case 79 : x += 1 ; y -= 1 ; break ;
              case 81 : x += 1 ; y += 1 ; break ;
          }//end of switch
    } while(key.c[1] != 59) ;
    temp = ptr ;
    for(i = 0 ; i < 14 ; i++)
      for(j = 0 ; j < 80 ; j += 2) {
          *temp=buf[i][j] ;
          *(temp + 8152) = buf[i][j + 1] ;
          temp ++ ;
      }
    return sides ;
}
//*****************************
```

مهندسی نرم‌افزار به کمک C

در اوایل ۱۹۷۰ بین مهندسین طراح و سازنده کامپیوتر و کسانی که طراح نرم‌افزار بودند تفاوتی نبود؛ هر کس که کامپیوتر را فهمیده بود، به عنوان یک مهندس کامپیوتر سخت‌افزار و نرم‌افزار شناخته می‌شد. در طی سالهای ۱۹۷۰ دو گروه از مهندسین کامپیوتر مطرح شده‌اند که عبارتنداز "مهندس سخت‌افزار" و "مهندس نرم‌افزار" و اکنون نیز در دانشگاههای سراسر دنیا این دو علم کامپیوتر، به عنوان دو رشته تدریس می‌شود. اکنون هنر و علم نرم‌افزار وسعت زیادی پیدا کرده است. ایجاد یک برنامهٔ کامپیوتری بزرگ همانند ساختن یک ساختمان است، نکات ظریف و مهمی در طراحی یک برنامه وجود دارد که با روشهای مهندسی مناسب قابل حل می‌باشند. در این فصل به بررسی بعضی از این روشها پرداخته می‌شود. بعضی از روشهای طراحی نرم‌افزار عبارتنداز: **۱.** طراحی بالا به پایین **۲.** طراحی پایین به بالا **۳.** روش موردی.

بدون شک تنها چیزی که موجب سهولت در ایجاد یک برنامه می‌شود، انتخاب یک روش مناسب جهت طراحی آن است، در روش **طراحی بالا به پایین** ابتدا به روال‌های سطح بالا و سپس به روال‌های سطح پایین پرداخته می‌شود. در روش **طراحی پایین به بالا** برعکس روش طراحی بالا به پایین، ابتدا به روال‌های سطح پایین و سپس به روال‌های سطح بالا پرداخته می‌شود. روش **موردی**، یک روش از پیش تعیین شده‌ای نیست، بلکه برحسب مورد، روش خاصی برای طراحی اتخاذ می‌شود.

در زبانهای ساخت‌یافته‌ای مثل C، طراحی برنامه به روش بالا به پایین انجام می‌شود. روش بالا به پایین موجب طراحی برنامه‌هایی می‌شود که از خوانایی بالایی برخوردار بوده، نگهداری آنها ساده است. روش طراحی بالا به پایین همچنین موجب دسته‌بندی ساختمانها و اعمال اصلی برنامه، قبل از تبدیل آنها به اعمال سطح پایین می‌شود. این امر موجب جلوگیری از اتلاف وقت می‌گردد.

طراحی برنامه

روش طراحی بالا به پایین از یک سری تعاریف کلی شروع شده به سمت موارد جزئی پیش می‌رود. در واقع یک روش مناسب جهت طراحی برنامه، تعیین خواسته‌های برنامه در سطوح مختلف است. به عنوان مثال، اگر خواسته باشیم برنامه‌ای برای تشکیل بانک اطلاعاتی، از دانشجویان دانشکده‌ای نوشته شود، باید مشخص گردد که این برنامه چه اعمالی را باید انجام دهد (رئوس کارهایی که باید انجام شود کدام است) و سپس این اعمال در یک لیست جمع‌آوری شده به طوری که هر یک از اعمال موجود در لیست فقط یک واحد کاری (function unit) را تشکیل دهد و هر واحد کاری باید فقط یک عمل را انجام دهد. به عنوان مثال، لیست تهیه شده ممکن است شامل واحدهای

کاری زیر باشد:

۱. وارد نمودن دانشجوی جدید در لیست.
۲. حذف دانشجو از لیست.
۳. چاپ محتویات لیست.
۴. جستجوی یک دانشجوی خاص.
۵. ذخیره کردن لیست بر روی دیسک.
۶. انتقال لیست از فایل به حافظه.
۷. خروج از برنامه.

این واحدهای کاری، اساس کار برنامه را تشکیل می‌دهند.

پس از مشخص شدن واحدهای کاری برنامه، می‌توان به جزئیات هر کدام از آنها پرداخت. به عنوان مثال، ساختار کلی برنامه را می‌توان به صورت زیر در نظر گرفت:

```
main_loop {
    do {
        display menu
        get user selection
        process the selection
    } while selection does not equal quit
}
```

استفاده از علایم الگوریتمی مطابق آنچه که در main_loop دیده می‌شود، کمکی است در دسته‌بندی ساختار کلی برنامه، حتی قبل از ورود به کامپیوتر.

برای هر یک از واحدهای کاری نیز می‌توان مانند main_loop عمل کرد و ساختار کلی آنها را مشخص نمود. به عنوان مثال، واحد کاری ذخیره کردن لیست بر روی دیسک را می‌توان به صورت زیر نوشت:

```
save_to_disk()
{
    open disk file
    while data left to write  {
        write data to disk
    }
    close disk file
}
```

در تابع save_to_disk() نیز چند واحد کاری مشاهده می‌شوند که هر کدام از آنها نیز باید تعریف گردند. اگر با تعریف این واحدهای کاری، واحدهای کاری دیگری ایجاد شوند، آنها نیز باید تعریف شوند تا این که واحد کاری دیگری وجود نداشته باشد. به عنوان مثال، واحد کاریِ "بازکردن فایل" را می‌توان به صورت زیر نوشت:

```
file *fp ;
if ((fp ==fopen ("list","r+")) ==NULL) {
    printf("\n can not open list file.");
    exit(1) ;
}
```

همانطور که در واحد کاری "بازکردن فایل" مشاهده می‌شود، واحد کاری دیگری نیز در آن وجود ندارد و لذا کار این واحد کاری به اتمام رسیده است.

لازم به ذکر است که در تعریف واحدهای کاری، اشاره‌ای به متغیرها و یا ساختمان داده‌ها نمی‌شود؛ زیرا هدف از تعریف واحدهای کاری این است که مشخص شود برنامه چه اعمالی را باید انجام دهد، نه این که این اعمال چگونه باید انجام شوند. تعریف واحدهای کاری، در انتخاب ساختمان داده‌ها و انجام اعمالی که قبل از کدشدن واحدهای کاری باید صورت گیرد، مفید است.

انتخاب یک ساختمان داده

پس از طرح کلی برنامه باید در مورد ساختمان داده‌های موردنیاز برنامه تصمیم‌گیری نمود. انتخاب ساختمان داده مناسب و پیاده‌سازی آن، یکی از مسایل مهم برنامه‌نویسی است؛ زیرا در طراحی محدودیتهای برنامه مؤثر است.

با استفاده از روش طراحی بالا به پایین، مشخص می‌شود که لیست تهیه شده جهت تعیین واحدهای کاری، شامل مجموعه‌ای از اطلاعات از قبیل نام دانشجو، شماره دانشجویی و ... است. برای نگهداری این اطلاعات می‌توان از ساختمان استفاده کرد. زیرا ساختمان، تنها ساختمان داده‌ای است که موجب می‌شود تا بتوان انواع مختلفی از اطلاعات را تحت یک نام ذخیره کرد. پس از انتخاب ساختمانها برای نگهداری اطلاعات دانشجویان، طریقهٔ ذخیره و بازیابی این ساختمانها نیز باید مشخص گردد. برای این منظور از یک آرایه با طول ثابت می‌توان استفاده کرد. اما آرایه با طول ثابت، دارای دو اشکال عمده است:

۱. طول آرایه در برنامه با یک مقدار فرضی ثابت مشخص می‌شود که در صورت کم و زیادشدنِ تعداد دانشجویان، باید در برنامه دستکاری شود.

۲. چنانچه تعداد دانشجویان بیشتر گردد، ممکن است حافظه لازم جهت ذخیره کردن اطلاعات وجود نداشته باشد و اگر از میزان حافظه کامپیوتر به دلیل خراب شدن کارت حافظه، کاسته شود، کل آرایه قابل انتقال به حافظه نیست.

بنابراین بهتر است حافظه لازم جهت ذخیره کردن اطلاعات دانشجویان به طور پویا از سیستم اخذ گردد. دراین صورت لیست مربوط می‌تواند هر اندازه بزرگ و یا کوچک باشد (در صورت وجود حافظه کافی).

گرچه استفاده از تخصیص حافظه پویا به جای آرایه‌ای با طول ثابت در نظر گرفته شده است، ولی شکل واقعی ذخیره و بازیابی داده‌ها هنوز مشخص نیست. راه‌حلهای متفاوتی برای این مساله وجود دارد: استفاده از لیستهای پیوندی ساده، لیستهای دوپیوندی، درخت دودویی و یا حتی روشهای درهم‌سازی (hashing). هر کدام از این روشها دارای مزایا و معایب خاص خودشان هستند. در عمل جستجو، اگر از درخت دودویی، به علت سرعت زیاد آن، استفاده شود، ساختمان زیر را می‌توان برای نگهداری اطلاعات در نظر گرفت:

```
struct student {
                char name [30] ;
                char addr [30] ;
                long int stno ;
                char cours [30] ;
                int unit ;
                struct student *left ;
                struct student *right ;
 } ;
```

اگر روش طراحی بالا به پایین جهت طراحی این برنامه دنبال شود، برنامه حاصل، نه تنها از خوانایی بالایی برخوردار خواهد بود، بلکه زمان کمتری را جهت نگهداری می‌طلبد.

پنهان سازی اطلاعات و کد

اشکالزدایی برنامه، به‌خصوص در برنامه‌های طولانی، از جمله مواردی است که در حین طراحی برنامه باید مورد توجه قرار گیرد. یکی از تکنیک‌هایی که می‌تواند در رفع اشکالات برنامه مفید واقع شود، پنهان‌سازی اطلاعات و کد است. منظور از پنهان‌سازی اطلاعات و کد، این است که هر تابع فقط به آنچه که نیاز دارد دسترسی داشته باشد و بقیه قسمتهای برنامه (سایر توابع) و اطلاعات، توسط این تابع قابل دسترسی نباشد. جهت حصول پنهان‌سازی اطلاعات و کد، باید قواعد زیر به کار گرفته شود.

۱. **هر واحد کاری فقط باید یک نقطهٔ ورود و یک نقطهٔ خروج داشته باشد.**

این بدین معنی است که گرچه هر واحد کاری ممکن‌است شامل چندین تابع باشد، ولی ارتباط بقیه قسمتهای برنامه با این واحد کاری از طریق یکی از این توابع انجام می‌شود. برای درک این مطلب، برنامهٔ تشکیل بانک اطلاعاتی دانشجویان را مجدداً در نظر می‌گیریم. در این برنامه چندین واحد کاری وجود دارد. کلیه توابع موردنیاز هر واحد کاری می‌توانند در یک فایل جداگانه قرار گرفته سپس به طور مجزا ترجمه شوند (روش انجام این کار در ادامه همین فصل بیان خواهد شد). اگر این عمل به درستی انجام گیرد تنها نقطهٔ ورود به یک واحد کاری و یا خروج از آن، فقط از طریق یکی از این توابع (تابعی که در بالاترین سطح قرار دارد) انجام خواهد شد. این توابع در برنامه تشکیل بانک‌اطلاعاتی از دانشجویان، فقط توسط تابع ()main فراخوانی می‌شوند (شکل ۱-۱۷).

۲. **از آرگومانها برای تبادل اطلاعات بین توابع استفاده گردد و سعی شود که از متغیرهای عمومی استفاده نشود.**

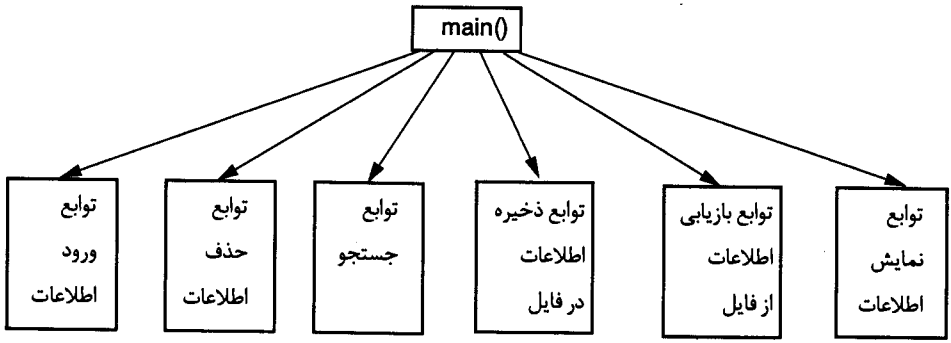
گرچه این عمل ممکن است از کارآیی برنامه بکاهد، ولی تنها راه جلوگیری از اثرات جانبی (side effect)، توسط توابع این است که تبادل اطلاعات بین توابع فقط از طریق آرگومانها صورت پذیرد.

۳. **اگر چند تابع مرتبط به هم، به متغیرهای عمومی نیاز داشته باشند، این توابع و متغیرهای عمومی موردنیاز آنها، در یک فایل جداگانه قرار داده شود و متغیرهای عمومی به صورت static تعریف گردند.**

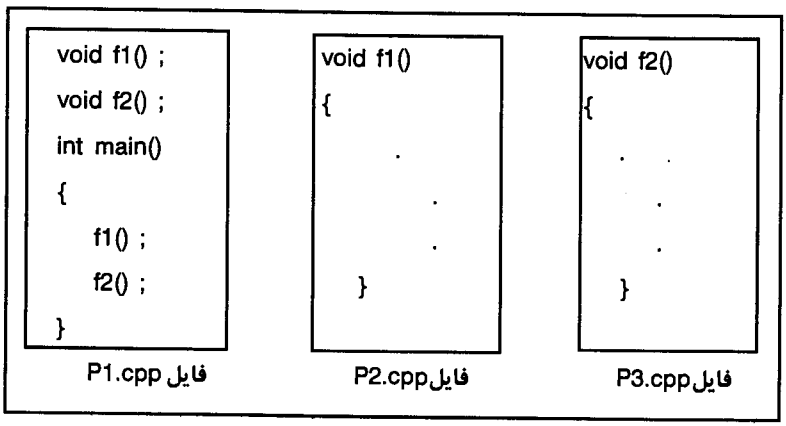
با رعایت ۳ مورد فوق تکنیک پنهان‌سازی اطلاعات و کد حاصل می‌گردد و اگر نتواند کارآیی برنامه را بهبود بخشد، می‌تواند در بازبینی برنامه جهت بهینه‌سازی آن مفید واقع شود.

برنامه‌های متشکل از چند فایل

برنامه‌هایی که تاکنون نوشته شدند، در یک فایل قرار داشته‌اند. اگر حجم برنامه زیاد باشد، می‌توان یک برنامه را در چند فایل قرار داد و آنها را تحت عنوان یک **پروژه** (project) اجرا کرد. برنامه‌ای که از چند فایل تشکیل شده باشد، فقط باید یک تابع ()main داشته باشد. در شکل ۲-۱۷ نمونه‌ای از برنامه که از ۳ فایل تشکیل شده است، مشاهده می‌شود.



شکل ۱-۱۷

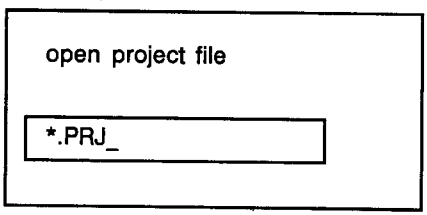


شکل ۲-۱۷ نمونه‌ای از برنامه که از ۳ فایل تشکیل شده است.

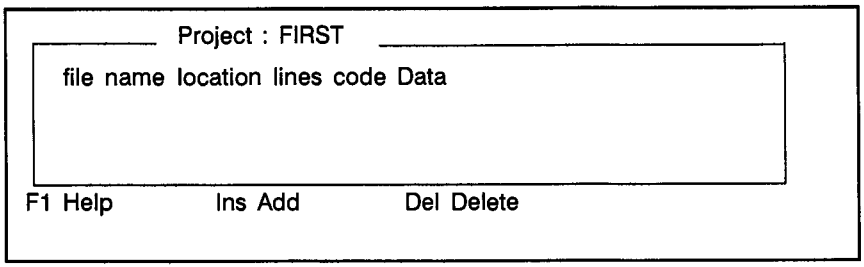
در زبان C برای اجرای این سه فایل، باید از آنها یک فایل پروژه ایجاد کرد. برای این منظور، مراحل زیر را انجام دهید:

۱. فایلهای p1.cpp ، P2.cpp و P3.cpp را ایجاد کنید.

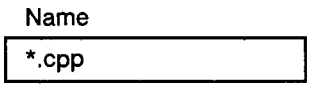
۲. در منوی project بورلند C++ (یا توربو C)، گزینه open project را انتخاب کنید. اکنون نام فایل پروژه از شما درخواست می‌شود:



۳. نام فایل پروژه را first.prj انتخاب کرده، کلید Enter را فشار دهید (می‌توانید دکمه OK را کلیک کنید). اکنون پنجره زیر ظاهر می‌شود:



۴. برای افزودن فایلها به پروژه، دکمه Insert را از صفحه کلید وارد کنید یا دکمه Add را کلیک کنید. در کادری که ظاهر می‌شود، نام اولین فایل را وارد کنید (P1.cpp) :



۵. مرحله چهار را برای هر فایلی که باید به پروژه اضافه شود، تکرار کنید.

۶. برای اجرای فایل پروژه، گزینه RUN را از منوی RUN یا گزینه make را از منوی Compile انتخاب کنید. منوی make فایلهای پروژه را اجرا نمی‌کند، بلکه فقط آن فایلهایی را که نیاز به ترجمه داشته باشند، ترجمه می‌کند. برای این منظور، زمان و تاریخ تشکیل فایل را با زمان و تاریخ فایل obj مقایسه می‌کند. اگر فایل اصلی جدیدتر از فایل obj آن باشد، فایل را ترجمه می‌کند وگرنه از ترجمه قبلی (فایل obj موجود) استفاده می‌کند. این موضوع در مورد فایل EXE نیز صادق است. یعنی، اگر فایل EXE جدیدتر از obj باشد، فایل منبع ترجمه نمی‌گردد، وگرنه پس از ترجمه، عمل لینک (link) انجام می‌شود. ولی در منوی RUN، تمام فایلهای پروژه، پس از ترجمه، اجرا می‌شوند.

مثال ۱-۱۷

برنامه‌ای که از دو فایل مجزا تشکیل شده است و با تشکیل فایل پروژه، اجرا شده، خروجی مناسبی تولید می‌کند (برنامه‌های test1.cpp و test2.cpp بر روی دیسک ذخیره شده‌اند).

فایل test1.cpp

```
#include <conio.h>
#include <stdio.h>
void count();
int main() {
   clrscr();
   printf("\n this is file 1.");
   count();
   return 0;
}
```

فایل test2.cpp

```
//**********************
#include <stdio.h>
void count()
{
  int i, j;
  printf("\n now in file 2.\n");
  for(i = 1; i <= 5; i++) {
     for(j = 1; j <= 5; j++)
        printf("%3d", i);
     printf("\n");
  }
}
```

فایل پروژه

```
test.prj
test1.cpp
test2.cpp
```

خروجی

```
this is file 1.
now in file 2.

  1   1   1   1   1
  2   2   2   2   2
  3   3   3   3   3
  4   4   4   4   4
  5   5   5   5   5
```

ایجاد کتابخانه

در بورلند C و توربو C با استفاده از برنامه TLIB می‌توان کتابخانه‌ای از توابع تشکیل داد و سپس از این توابع در برنامه‌های دیگر استفاده نمود. کتابخانه، مجموعه‌ای از توابع هست که به صورت OBJ. می‌باشند. گرچه یک کتابخانه به برنامه‌ای حاوی چند فایل مجزا شباهت دارد ولی از جهات مختلفی با یکدیگر متفاوتند؛ وقتی که فایلهای مختلف یک پروژه ترجمه و سپس به یکدیگر اتصال داده می‌شوند، کلیه توابع موجود در فایلها، در فایل اجرایی کپی می‌شوند. گرچه یک فایل کتابخانه با فایل دیگری اتصال داده می‌شود اما فایل اجرایی، فقط شامل توابعی خواهد بود که توسط برنامه قابل استفاده است. به عنوان مثال کتابخانه بورلند C حاوی صدها تابع می‌باشد، اما برنامهٔ اجرایی که توسط بورلند C ساخته می‌شود، فقط شامل توابعی خواهد بود که این برنامه به آن نیاز دارد.

ایجاد کتابخانه و قراردادن توابع در آن، در پروژه‌های بزرگی که شامل چندین تابع باشند مفید است.

با استفاده از TLIB می‌توان ۳ عمل در مورد کتابخانه‌ها انجام داد:

۱. اضافه نمودن ماژول‌هایی به کتابخانه
۲. حذف ماژول‌ها از کتابخانه
۳. استخراج ماژول‌ها از کتابخانه

روش کلی استفاده از TLIB به صورت زیر است:

TLIB <نام کتابخانه> [OP][نام ماژول][OP][نام ماژول] ... TLIB +first

در این شکل کلی، OP یکی از عملگرهای معتبر زیر است که عمل TLIB را مشخص می‌کند:

عملگر	عملی که توسط TLIB انجام می‌شود
+	افزودن ماژول‌هایی به کتابخانه موردنظر
-	حذف ماژول‌هایی از کتابخانه موردنظر
*	استخراج ماژول‌هایی از کتابخانه موردنظر
+- یا -+	جایگزین کردن یک ماژول با محتویات جدید (حذف و اضافه)
-* یا *-	استخراج یک ماژول از کتابخانه و حذف آن (استخراج و حذف)

نام کتابخانه از قانون نامگذاری برای فایلها تبعیت می‌کند و پسوند آن LIB. فرض می‌شود. **نام ماژول‌ها**، اسامی فایلهایی با پسوند OBJ. هستند که حاوی توابعی جهت قرارگرفتن در کتابخانه می‌باشند. گرچه در حین استفاده از کتابخانه، یک تابع می‌تواند از ماژول مربوطه جدا شود و در برنامه اجرایی قرار گیرد (link)، ولی استخراج یک تابع از کتابخانه ممکن نیست بلکه فقط ماژول‌ها می‌توانند توسط TLIB از کتابخانه استخراج گردند.

برای ایجاد کتابخانه کافی است نام کتابخانه موردنظر را در TLIB ذکر کرد و ماژولی را به آن اضافه نمود. به عنوان مثال، دستور زیر موجب ایجاد کتابخانه‌ای به نام NEWLIB و اضافه‌نمودن ماژول first به آن می‌شود:

```
TLIB NEWLIB +first
```

همانطور که گفته شد، ماژول first باید ترجمه شده و پسوند آن OBJ. باشد.

مثال ۲-۱۷

فایلی به نام first.cpp ایجاد کرده و توابع زیر را در آن تایپ کنید:

```
#include <stdio.h>
void func1()
{
  printf("\n in function 1.");
}
//***************
void func2()
{
  printf("\n in function 2.");
}
//*************
```

```
void func3()
{
  printf("\n in function 3.");
}
void func4()
{
  printf("\n in function 4.");
}
```

فایل first.cpp را ترجمه کنید تا فایل first.obj به وجود آید (با گزینه Compile از منوی Compile).
دستور زیر را در خط فرمان صادر کنید:

```
TLIB NEWLIB +first
```

این دستور، کتابخانه‌ای به نام newlib.lib را ایجاد می‌کند و ماژول first را که شامل ۴ تابع است به آن اضافه می‌نماید.
برنامهٔ زیر را جهت استفاده از توابع ()func1 ، ()func2 ، ()func3 و ()func4 در فایل test.cpp تایپ کنید:

```
void func1();
void func2();
void func3();
void func4();
int main()
{
  func1();
  func2();
  func3();
  func4();
  return 0;
}
```

یک فایل پروژه تحت نام project ایجاد کرده فایلهای زیر را به آن اضافه کنید و سپس آن را اجرا کنید.

test.cpp
newlib.lib

با اجرای فایل پروژهٔ project خروجی زیر حاصل می‌شود:

```
in function 1.
in function 2.
in function 3.
in function 4.
```

مشاهده محتویات فایل کتابخانه

برای مشاهده محتویات فایل کتابخانه (اسامی ماژول‌ها و توابع) باید برنامه TLIB را به صورت زیر به کار برد:

TLIB <نام کتابخانه> , <نام فایل خروجی>

نام کتابخانه، یکی از فایلهای کتابخانه موجود است که محتویات آن باید مشاهده گردند. **نام فایل خروجی**، فایلی است که توسط TLIB ایجاد شده، اسامی ماژول‌ها و توابع موجود در فایل کتابخانه در آن قرار می‌گیرند. به عنوان مثال، دستور زیر را در نظر بگیرید:

```
TLIB NEWLIB.LIB START
```

با اجرای این دستور فایلی به نام START.LST ایجاد می‌شود که محتویات آن را با استفاده از دستور type (یکی از دستورات داخلی سیستم عامل) می‌توان مشاهده کرد:

```
FIRST  size = 120
func1()  func2()
func3()  func4()
```

برنامه GREP

در بورلند C و توربو C گونه ۱/۵ و به بالا، برنامه‌ای به نام GREP وجود دارد که تقریباً مشابه دستور FIND در سیستم عامل DOS عمل می‌کند. این برنامه رشته‌ای را در یک یا چند فایل مورد جستجو قرار می‌دهد و به صورت زیر به کار می‌رود:

GREP <رشتهٔ مورد جستجو> <اسامی فایلها>

رشتهٔ مورد جستجو، در فایلهایی که اسامی آنها مشخص می‌شود، جستجو می‌گردد و فایلی که این رشته در آن قرار دارد به همراه شماره خط حاوی این رشته، مشخص می‌گردد. به‌عنوان مثال، دستور زیر را در نظر بگیرید:

```
GREP print F1.CPP  F2.CPP F3.CPP
```

این دستور، رشتهٔ printf را در فایلهای f1.cpp، f2.cpp و f3.cpp جستجو می‌کند.

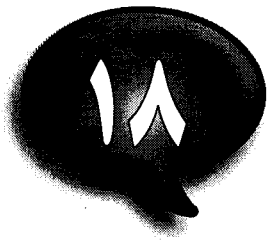
نکات زیر را در مورد برنامه GREP در نظر داشته باشید:

۱. رشتهٔ مورد جستجو در علامت نقل قول (") قرار نمی‌گیرد.
۲. استفاده از علایم * و ? در اسامی فایلها امکان‌پذیر است:

```
GREP scanf("%d")  *.cpp
```

این دستور رشتهٔ scanf("%d") را در کلیه فایلهای با پسوند CPP جستجو می‌کند.

۳. اسامی فایلهایی که رشتهٔ موردنظر باید در آنها جستجو شود، با فاصله (blank) از یکدیگر جدا می‌شوند.



طراحی مفسر زبان‌های برنامه‌سازی

برنامه‌ای که در یک زبان برنامه‌سازی مثل C نوشته شده است، قبل از اجرا باید ترجمه گردد. **مفسر** نوعی مترجم برای زبانهای برنامه‌سازی است که برنامه‌ای به یک زبان برنامه‌سازی را به عنوان ورودی پذیرفته آن را تجزیه و تحلیل نحوی کرده و سپس آن را اجرا می‌کند. مفسر زبانها به دلایل زیر مهم‌اند :

۱. یک وسیله محاوره‌ای (interactive) خوبی برای کارکردن با برنامه است، مثل مفسر زبان بیسیک؛ بسیاری از برنامه‌نویسان مبتدی کارکردن با مفسر را بر کامپایلر ترجیح می‌دهند.

۲. می‌توانند یک وسیله باهوش در رفع اشکالات برنامه باشند.

۳. بسیاری از برنامه‌های مدیریت بانکهای اطلاعاتی، تقاضای زبانهای بانکهای اطلاعاتی را تفسیر می‌کنند.

در این فصل یک مفسر برای زبان برنامه‌سازی بیسیک نوشته می‌شود (برای قسمتی از امکانات بیسیک). دلیل انتخاب زبان برنامه‌سازی بیسیک، سادگی آن نسبت به زبانهای ساخت‌یافته‌ای مثل C است. توابعی که در این فصل نوشته می‌شوند نقطهٔ شروع خوبی برای نوشتن مفسرها می‌باشند و با اعمال تغییراتی در آنها می‌توان مفسری برای زبانهای برنامه‌سازی ساخت‌یافته نوشت.

تجزیه عبارات (expression parsing) ، یکی از قسمتهای مهم یک مفسر می‌باشد که گاهی قلب مفسر نامیده می‌شود. تجزیه کردن عبارات، مکانیزمی است که عباراتی مثل 23/(X-10) را به شکلی درمی‌آورد که کامپیوتر بتواند آنها را ارزیابی نماید. قبل از ادامه بحث در مورد مفسرها بهتر است مفهوم عبارت مشخص گردد.

عبارات

گرچه عبارات می‌توانند ترکیبی از همه نوع اطلاعات باشند ولی در این فصل جهت سهولت در کار، فقط عبارات عددی در نظر گرفته می‌شوند. عبارات عددی می‌توانند ترکیبی از عناصر زیر باشند:

۱. اعداد
۲. عملگرها (+ - / * ^ % = ( ) > < ; ,)
۳. پرانتزها
۴. متغیرها

عملگر ^ برای انجام عمل توان و عملگر = برای عمل انتساب و تساوی به کار می‌رود. عناصر فوق می‌توانند به روشهای جبری با یکدیگر ترکیب گردند و عبارات گوناگونی را به وجود آورند:

```
7-8
(100-5) * 14/16
a + b - c
10 ^ 5
a = 7 - b
```

گرچه علایم =، < ، > ،کاما و ;از عملگرها محسوب می‌شوند، ولی با توجه به رفتار بیسیک نسبت به آنها، در تجزیه‌کننده‌ای که در این فصل نوشته می‌شود منظور نشده بلکه در توابعی که برای اجرای دستوراتی مثل IF ، PRINT و ... نوشته می‌شوند پردازش خواهند شد. تقدم عملگرهای بیسیک را می‌توان به صورت زیر در نظر گرفت:

```
( )          بالاترین تقدم
^
* / %
+ -
=            پایین‌ترین تقدم
```

در تقدم فوق، عملگرهایی که دارای تقدم یکسانی هستند (مثل –و +) تقدم مکانی آنها منظور خواهد شد (از چپ به راست). یعنی هر عملگری که زودتر ظاهر شود، زودتر عمل می‌کند. در مفسری که خواهیم نوشت، فرض می‌شود که نام هر متغیر فقط یک کاراکتر است (از حروف A تا Z) و ضمناً تفاوتی بین حروف کوچک و بزرگ منظور نخواهد شد. کلیه متغیرهای عددی به صورت صحیح در نظر گرفته می‌شوند؛ هر چند که نوشتن توابعی جهت ارزیابی سایر انواع اعداد، چندان مشکل نیست. متغیرهای رشته‌ای قابل استفاده نیستند ولی ثوابت رشته‌ای می‌توانند در برنامه به کار گرفته شوند؛ انتهای برنامه نیز با کاراکتر NULL مشخص می‌شود.

نشانه‌ها (tokens)

قبل از طراحی یک تجزیه‌کننده برای تجزیه عبارات، باید راههای تجزیه یک عبارت مشخص گردد. به عنوان مثال، عبارت زیر ترکیبی از نشانه‌های A، *، B، -،)، W، +، 10 و (است.

```
A * B - (W + 10)
```

تابعی که برای تجزیه عبارات به نشانه‌های مختلف نوشته می‌شود، باید چهار عمل زیر را انجام دهد:

۱. صرفنظر از کاراکترهای blank و tab.
۲. استخراج هر نشانه.
۳. در صورت لزوم، تبدیل نشانه به یک فرمت داخلی.
۴. تعیین نوع نشانه.

هر نشانه دارای دو فرمت است: فرمت خارجی و فرمت داخلی. **فرمت خارجی** همان شکلی است که در برنامه

مورد استفاده قرار می‌گیرد. به عنوان مثال، PRINT فرمت خارجی دستور PRINT در بیسیک است. گرچه می‌توان مفسر را طوری طراحی کرد که از فرمت خارجی استفاده نماید ولی به علت کندی کار، به جای آن از فرمت داخلی استفاده می‌گردد.

در **فرمت داخلی**، هر یک از کلمات کلیدی با یک عدد مشخص می‌شوند. به عنوان مثال، دستور PRINT با عدد ۱، دستور INPUT با عدد ۲ و ... مشخص می‌گردند. علت سرعت مفسر به هنگام استفاده از فرمت داخلی، سروکار داشتن آن با اعداد صحیح به جای رشته‌ها است. مفسری که برای بیسیک نوشته می‌شود از ۶ نوع دادهٔ مختلف استفاده می‌کند: DELIMITER برای عملگرها و پرانتزها؛ VARLABLE برای متغیرها؛ NUMBER برای اعداد؛ COMMAND برای فرمانهای بیسیک؛ STRING به‌عنوان یک متغیر کمکی تا تکمیل نشانه؛ و QUOTE برای رشته‌هایی که در نقل‌قول (") قرار می‌گیرند.

مثال ۱-۱۸

تابع ()get_token که نشانه‌های موجود در یک عبارت را جدا می‌کند. این تابع از چند تابع دیگر نیز استفاده می‌کند که شرح وظایف آنها در اینجا آمده است. لیست این توابع در مثال ۱۵-۱۸ آمده است.

شرح وظایف توابع

تابع ()get_token: از توابعی به نامهای ()iswhite ، ()serror ، ()isdelim و look_up استفاده می‌کند که لیست آنها در مثال ۱۵-۱۸ مشاهده می‌گردد ولی توضیح مختصری در مورد عملکرد این توابع به درک تابع ()get_token کمک می‌کند. چهار متغیر مهم در تابع ()get_token مورد استفاده قرار می‌گیرند که عبارتند از : prog ، tok ، token و token_type . متغیر prog به کاراکتر بعدی که باید از فایل خوانده شود اشاره می‌کند. متغیر tok حاوی فرمت داخلی نشانه، متغیر token حاوی نشانه و متغیر token_type حاوی نوع نشانه می‌باشد. به عنوان مثال، نتیجه عمل تابع ()get_token بر روی عبارت PRINT A + 100 - (B * C / 2) به صورت جدول ۱-۱۸ است.

تابع ()iswhite: کاراکتری را به عنوان پارامتر می‌پذیرد، اگر این پارامتر برابر با یکی از کاراکترهای blank یا tab باشد، عدد ۱ وگرنه عدد صفر را به تابع ()get_token برمی‌گرداند.

تابع ()serror: برای صدور پیام خطا مورد استفاده قرار می‌گیرد.

تابع ()isdelim: کاراکتری را به عنوان پارامتر پذیرفته، چنانچه این کاراکتر یک جداکننده (DELIMITER) باشد عدد ۱، وگرنه عدد ۰ را به تابع ()get_token برمی‌گرداند.

تابع ()look_up: در جدولی که متغیرها در آنجا قرار می‌گیرند، متغیری را جستجو می‌کند. اگر این متغیر در جدول پیدا شود، فرمت داخلی آن، وگرنه عدد صفر را به تابع ()get_token برمی‌گرداند.

چگونگی تولید عبارات

جدول ۱-۱۸

Token	Token type
PRINT	COMMAND
A	VARLABEL
+	DELIMITER
100	NUMBER
-	DELIMITER
(	DELIMITER
B	VARLABEL
*	DELIMITER
C	VARLABEL
)	DELIMITER
/	DELIMITER
2	NUMBER
null	DELIMITER

برای تجزیه و ارزیابی عبارات راههای گوناگونی وجود دارد. برای استفاده از تجزیه کننده بازگشتی، باید عبارات را به صورت یک ساختمان دادهٔ بازگشتی در نظر گرفت. یعنی عبارات توسط خودشان تعریف می‌گردند. با فرض اینکه عبارات می‌توانند فقط با ترکیبی از +، -، *، /و پرانتز ساخته شوند قواعد زیر را می‌توان برای تعریف آنها در نظر گرفت:

Expression => Term[+Term] [-Term]
Term => Factor[* Factor] [/ Factor]
Factor => Var, Num or (Expression)

در رابطهٔ فوق، < = به معنی "تولید می‌شود" و [] به معنی "اختیاری بودن" است. روابط بالا را **قواعد تولید** نیز می‌گویند. لذا از قاعدهٔ Term = > Factor [*Factor] [/Factor] می‌توان نتیجه گرفت که Term، با ضرب یک factor در factor دیگر و یا از تقسیم factor بر factor دیگر حاصل می‌شود. به عنوان مثال، عبارت 10+5*B متشکل از دو Term به نامهای 10 و 5*B و سه factor به نامهای 10، 5 و B است که یکی از آنها متغیر و دوتای دیگر عدد هستند. عبارت (7-C)*14 متشکل از دو Term به نامهای 14 و (7-C) است که یکی از آنها عددی و دیگری یک عبارت پرانتزی است. عبارت پرانتزی شامل یک متغیر و یک عدد است.

قواعد تولید عبارات را می‌توان به صورت توابع بازگشتی نوشت که مجموع آنها تشکیل یک تجزیه کنندهٔ بازگشتی می‌دهند. برای روشن شدن مطلب، به چگونگی تجزیه عبارت زیر توجه کنید:

9/3 - (100 + 56)

۱. Term اول را استخراج نمایید: (9/3).
۲. هر factor را از این Term جدا کرده عمل تقسیم را انجام دهید (۳ حاصل می‌شود).
۳. با استفاده از خاصیت بازگشتی، Term دوم را جدا کنید (56+100).
۴. هر factor را از این term جدا کرده عمل جمع را انجام دهید (۱۵۶).
۵. پس از فراخوانی بازگشتی عدد ۱۵۶ را از ۳ کم کنید. عدد ۱۵۳- به عنوان نتیجه عبارت موردنظر حاصل می‌گردد.

مثال ۲-۱۸

اکنون شرح وظایف و نمودار سلسله مراتبی توابعی را که یک تجزیه کننده بازگشتی را پیاده‌سازی می‌کنند مطالعه می‌کنیم. لیست کامل این توابع در مثال ۱۵-۱۸ که مفسر را پیاده‌سازی می‌کند آمده است. برای پی بردن به چگونگی عملکرد توابع تجزیه کننده بازگشتی، نمودار سلسله مراتبی (شکل ۱-۱۸) و شرح وظایف هر یک از توابع را مطالعه کنید.

شرح وظایف توابع

تابع get_exp(): تشخیص وجود یا عدم وجود عبارت و فراخوانی سایر توابع مطابق شکل ۱-۱۸.

تابع get_token(): جداکردن نشانه‌ها از عبارت (به مثال ۱-۱۸ مراجعه شود).

تابع serror(): صدور پیام خطای مناسب.

تابع level2(): جمع و یا تفریق دو factor.

تابع level3(): ضرب و یا تقسیم دو factor.

تابع level4(): انجام عمل توان برای اعداد صحیح.

تابع level5(): تشخیص عملگرهای + و - یکانی و اثر آنها بر عبارت.

تابع level6(): پردازش عبارات شامل پرانتز.

تابع primitive(): پیداکردن مقادیر اعداد و یا متغیرها.

تابع arith(): انجام اعمال محاسباتی.

تابع unary(): تغییر علامت از + به - و برعکس.

تابع find_var(): تعیین مقدار یک متغیر (از یک جدول تعیین می‌کند).

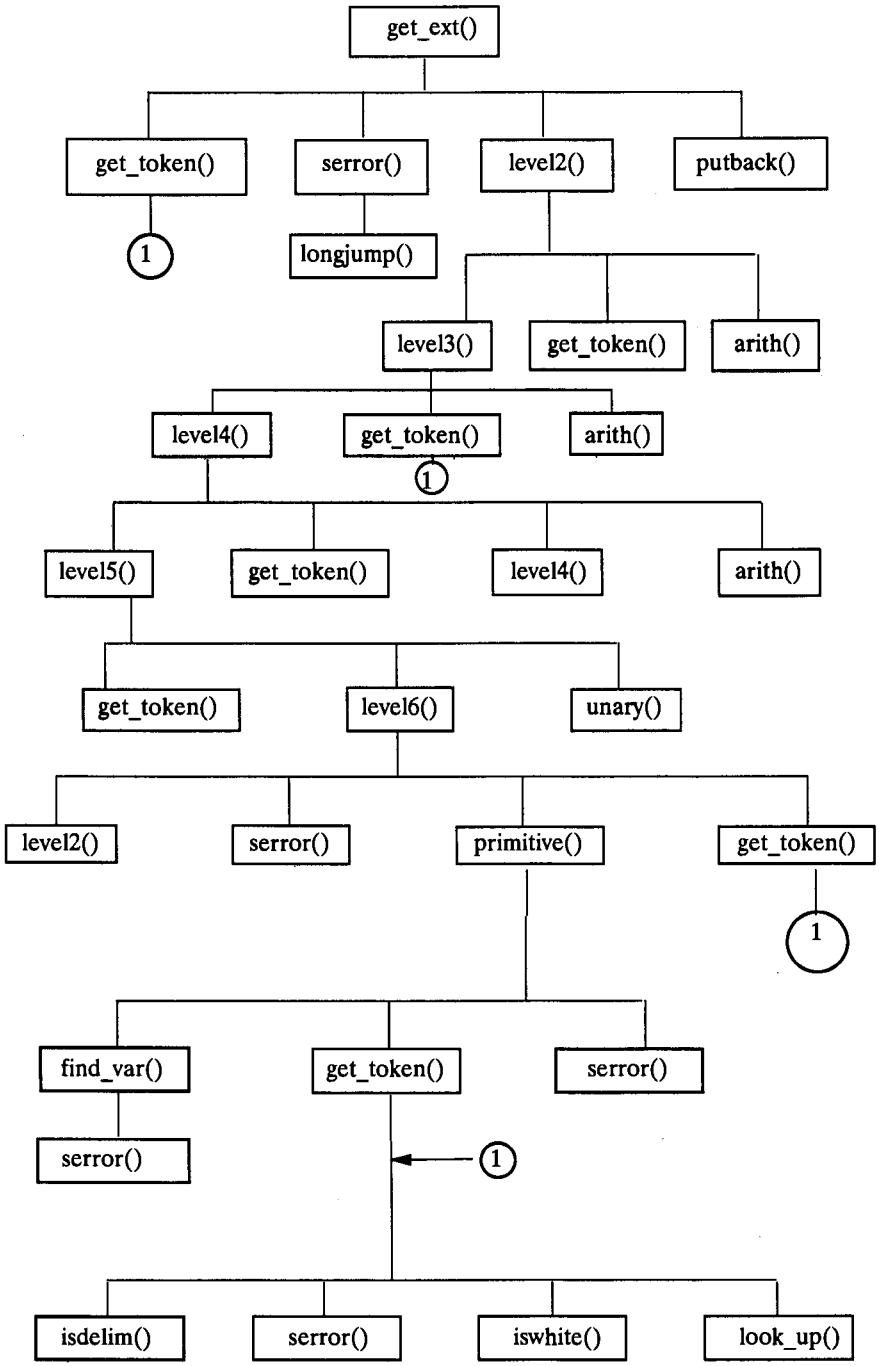
تابع iswhite(): تشخیص کاراکتر blank و tab.

تابع putback(): برگرداندن آخرین نشانهٔ خوانده شده به عبارت ورودی. بعضی از توابع موجود در مفسر برای تعیین عمل بعدی، نیاز به پیش‌بینی یک نشانه دارند. در بسیاری از موارد در صورت عدم نیاز روالها به نشانه، باید به رشته ورودی برگردانده شوند.

تابع look_up(): تعیین فرمت داخلی نشانه‌ها یا جستجو در جدول مربوط.

تابع isdelim(): تشخیص جداکننده.

تجزیه‌کننده برای پیداکردن متغیرها و مقادیر آنها، از جدولی به نام variable استفاده می‌کند که مقادیر اولیهٔ همهٔ متغیرها برابر با صفر است. چون فقط متغیرهای یک حرفی در این تجزیه‌کننده بررسی می‌شوند، برای پیداکردن یک متغیر از جدول variable، کافی است کد اسکی کاراکتر A از کد اسکی نام متغیر کم گردد. محتویات جدولِ variable و لیست تابع find_var() را در مثال ۱۵-۱۸ ببینید.



شکل ۱-۱۸ نمودار سلسله مراتبی توابع تجزیه کنندهٔ عبارات.

مفسر زبان بیسیک

پس از آشنایی با مفهوم عبارات و چگونگی تجزیه آنها، پرداختن به چگونگی نوشتن یک مفسر برای زبان بیسیک، زمان مناسبی است. همانطور که می‌دانید در زبان بیسیک در حدود ۱۵۹ کلمهٔ کلیدی و چندین تابع کتابخانه‌ای وجود دارد که نوشتن مفسر برای آن خارج از توان این کتاب است ولی برای سهولت در کار، کلمات کلیدی زیر در این مفسر در نظر گرفته خواهند شد:

PRINT INPUT IF THEN FOR NEXT TO GOTO GOSUB RETURN END

هر یک از این کلمات کلیدی و EOL (تعیین کنندهٔ انتهای خط) و FINISHED (تعیین کنندهٔ انتهای فایل) در زیر مشاهده می‌گردند:

```
#define PRINT      1
#define INPUT      2
#define IF         3
#define THEN       4
#define FOR        5
#define NEXT       6
#define TO         7
#define GOTO       8
#define EOL        9
#define FINISHED   10
#define GOSUB      11
#define RETURN     12
#define END        13
```

چون برای تبدیل فرمت خارجی نشانه‌ها به فرمت داخلی، نیاز به فرمت خارجی نشانه نیز هست، لذا فرمت داخلی و خارجی در یک جدول به نام table به صورت زیر ذخیره می‌شوند:

```
struct commands {
    char commands[20] ;
    char tok ;
} table[] = {
    "print", PRINT, "input", INPUT, "if",IF,
    "then", THEN, "goto", GOTO, "for", FOR ,
    "next", NEXT , "to", TO , "gosub", GOSUB ,
    "return", RETURN , "end", END , "", END
  } ;
```

مثال ۳-۱۸

تابع look_up() برای پیداکردن فرمت داخلی یک نشانه مورد استفاده قرار می‌گیرد. اگر نشانه موردنظر یکی از نشانه‌های معتبر نباشد، صفر را به عنوان نتیجه عمل برمی‌گرداند. لیست این تابع در مثال ۱۵-۱۸ آمده است.

مثال ۴-۱۸

پس از نوشتن برنامهٔ بیسیک در یک ویراستار، باید آن را به حافظه منتقل کرد و سپس اجرا نمود. برای انتقال برنامهٔ بیسیک به حافظه، از تابع load_program() استفاده می‌شود. لیست این تابع در مثال ۱۵-۱۸ آمده است.

مثال ۵-۱۸

هر مفسر تقریباً به این صورت عمل می‌کند :نشانه بعدی را از عبارت خوانده سپس تابع مناسبی را جهت پردازش آن نشانه فراخوانی می‌کند. یعنی دارای یک حلقه تکرار اصلی است. حلقه تکرار اصلی برای مفسر بیسیک به صورت زیر می‌باشد.

```
do {
  token_type = get_token() ;
  if(token_type == VARIABLE) {
      putback() ;
      assignment() ;
  }
  else
      switch(tok) {
         case PRINT :  print() ;  break ;
         case GOTO  :  exec_goto() ; break ;
         case IF    :  exec_if() ;  break ;
         case FOR   :  exec_for() ; break ;
         case NEXT  :  next() ;   break ;
         case INPUT :  input() ;  break ;
         case GOSUB :  gosub() ;  break ;
         case RETURN:  greturn() ; break ;
         case END   :  exit(0) ;
       }
} while (tok != FINISHED) ;
```

نحوهٔ عمل حلقه تکرار اصلی به این صورت است: اولین نشانه هر خط از برنامه خوانده می‌شود. با فرض اینکه هیچ‌گونه خطای نحوی (syntax error) وجود نداشته باشد، این نشانه یا یک متغیر است که باید مقداری به آن نسبت داده شود (استفاده از LET مجاز نیست) و یا یک دستور است که باید تابع مناسبی جهت اجرای آن دستور فراخوانی گردد.

برای استفاده صحیح از مفسر باید نحوه کاربرد هر یک از دستورات موجود در آرایه table (که در صفحات قبلی آمده است) مشخص شده، توابعی که برای اجرای هر کدام از آنها لازمند نوشته شوند.

دستور انتساب

در بیسیک برای انتساب یک مقدار به یک متغیر از رابطه زیر استفاده می‌شود:

عبارت = نام متغیر

مثال ۶-۱۸

تابعی که احکام انتساب را اجرا می‌کند. تابع ()assignment نشانه‌ای را خوانده (این نشانه متغیری است که باید مقداری به آن نسبت داده شود) و سپس علامت = را می‌خواند. اگر نشانهٔ اولی که خوانده شد یک متغیر نباشد و یا نشانهٔ دوم علامت = نباشد، پیام خطای مناسبی صادر می‌شود. پس از خواندن علامت =، تابع ()get_exp برای انتساب یک مقدار به متغیر فراخوانی می‌گردد. لیست این تابع در مثال ۱۵-۱۸ آمده است.

دستور PRINT

در بیسیک استاندارد، دستور PRINT دارای شکل پیچیده‌ای است؛ از جمله : وجود USING در این دستور، انعطاف‌پذیری آن را افزایش داده است. در این مفسر شکل ساده‌ای از این فرمان در نظر گرفته می‌شود:

```
PRINT <arg>
```

arg شامل متغیرها یا رشته‌هایی است که با کاما یا ; از یکدیگر جدا می‌شوند.

مثال ۷-۱۸

تابعی به نام ()print نوشته شده است که اجرای دستور PRINT را در بیسیک را به عهده دارد. لیست این تابع در مثال ۱۵-۱۸ آمده است.

دستور INPUT

برای خواندن اطلاعات از ورودی، از دستور INPUT استفاده می‌گردد و به دو صورت به کار گرفته می‌شود:

```
INPUT  <var>
INPUT  "prompt", <var>
```

در هر دو روش، VAR اسم متغیری است که باید از ورودی خوانده شود؛ prompt پیامی است که در حین اخذ مقدار متغیر از ورودی، بر روی صفحه‌نمایش ظاهر می‌گردد.

مثال ۸-۱۸

تابعی به نام ()input نوشته سده است که دستور INPUT را در بیسیک اجرا می‌کند. لیست این تابع در مثال ۱۵-۱۸ آمده است.

دستور GOTO

برای انتقال کنترل از نقطه‌ای به نقطهٔ دیگرِ برنامه، از دستور GOTO به صورت زیر استفاده می‌شود:

```
GOTO   <برچسب>
```

برچسب، شماره خطی است که انتقال کنترل باید به آنجا منتقل گردد. مشکلی که در اجرای دستور GOTO وجود دارد این است که انتقال از ابتدا به انتهای برنامه و یا برعکس، باید امکان‌پذیر باشد. برای این منظور قبل از اجرای برنامه، باید برچسب‌های موجود در برنامه شناسایی شده اسامی آنها به همراه اشاره‌گری که به محل وجود آنها در

برنامه اشاره می‌کند، در جدولی نگهداری گردد. سپس برای اجرای دستور GOTO کافی است آدرس خطی از برنامه که باید اجرا گردد، از جدول مربوط پیدا شود. جدولی که برچسب‌ها و آدرس آنها را نگهداری می‌کند به صورت زیر تعریف شده است:

```
struct label  {
              char name [LAB_LEN] ; /* name of label */
              char *P ; /* point to place of label in source file */
              } ;
struct label label_table [NUM_LAB] ;
```

مثال ۹-۱۸

تابع scan_label() که برچسبها و آدرس آنها را در جدولِ label قرار می‌دهد. این تابع با استفاده از تابع label_init() جدول برچسبها را مقدار اولیه می‌دهد. تابع get_token() را برای اخذ نشانه بعدی، و تابع find_eol() را برای پیدا کردن انتهای سطر فراخوانی می‌کند. اگر دو برچسب مساوی در برنامه وجود داشته باشد و یا جدول مربوط به برچسبها پر نشده باشد، پیام خطای مناسبی صادر می‌گردد. لیست این تابع در مثال ۱۵-۱۸ آمده است.

مثال ۱۰-۱۸

تابع exec_goto() برای اجرای دستور GOTO. تابع find_label() برچسبی را در جدول برچسبها جستجو می‌کند. در صورت پیداکردن، اشاره‌گر مربوط به آن را در متغیر prog قرار می‌دهد وگرنه NULL را برمی‌گرداند و پیام مناسبی صادر می‌کند. لیست این توابع در مثال ۱۵-۱۸ آمده است.

دستور IF

دستور if برای انتقال کنترل شرطی استفاده می‌شود و شکلی از آن که در مفسر به کار می‌رود به صورت زیر می‌باشد:

IF <عبارت> <عملگر> <عبارت> THEN <دستور>

عملگرهایی که در دستور if بررسی می‌شوند عبارتنداز: =، > و <.

مثال ۱۱-۱۸

تابع exec_if() که دستور IF را اجرا می‌کند. لیست این تابع در مثال ۱۵-۱۸ آمده است. نحوهٔ عمل تابع exec_if() به صورت زیر می‌باشد:

۱. عبارت سمت چپ ارزیابی می‌شود.
۲. عملگر خوانده می‌شود.
۳. عبارت سمت راست ارزیابی می‌گردد.
۴. شرط ذکر شده تست می‌گردد.
۵. اگر شرط ذکرشده، درست باشد دستور بعد از THEN اجرا می‌گردد وگرنه تابع find_eol() برای پیداکردن خط بعدی فراخوانی می‌شود.

دستور FOR

برای تکرار اجرای مجموعه‌ای از دستورات، از FOR استفاده می‌گردد:

FOR شمارنده = مقدار اولیه TO مقدار نهایی

:

NEXT

همانطور که مشاهده می‌گردد، استفاده از STEP مجاز نمی‌باشد. چند دستور for ممکن است به صورت لانه‌ای مورد استفاده قرار گیرند که نگهداری اطلاعاتی در مورد هر دستور for از نکات مهمی است که نیاز به درنظر گرفتن پشته دارد: در ابتدای حلقه تکرار، اطلاعاتی از قبیل وضعیت شمارنده (مقدار اولیه و مقدار نهایی) و آدرس شروع حلقه تکرار در پشته نگهداری می‌شود. پس از مشاهدهٔ NEXT، این مقادیر از پشته خارج شده، شمارنده بازسازی می‌گردد و با مقدار نهایی مقایسه می‌شود، در صورتی که مقدار فعلی شمارنده از مقدار نهایی بیشتر گردد، حلقهٔ تکرار خاتمه می‌یابد و کنترل به دستور بعد از NEXT برمی‌گردد، وگرنه مقادیر جدید در پشته قرار خواهند گرفت و اجرای دستورات از ابتدای حلقه تکرار آغاز می‌گردد. این روند اجرای دستورات for، به صورت لانه‌ای نیز امکان‌پذیر است.

مثال ۱۲-۱۸

پشته موردنیاز برای اجرای دستور for و توابع ()fpush و ()fpop برای دستکاری این پشته. متغیر FOR_NEST تعداد دستورات for را که می‌توانند در داخل یکدیگر اجرا شوند مشخص می‌کند.

```
struct for_stack {
   int var ;
   int target ;
   char * loc ;
} fstack[FOR_NEST] ;
int ftos;
void fpush(struct for_stack i)
 {
      if(ftos > FOR_NEST)
     serror(10) ;
      fstack[ftos] = i ;
      ftos++ ;
 }
//**********************
struct for_stack fpop()
 {
      ftos -- ;
      if(ftos < 0) serror(11);
       return(fstack[ftos]);
 }
```

مثال ۱۳-۱۸

تابع exec_for() برای اجرای دستور FOR. همانطور که در لیست این تابع مشاهده می‌شود(در مثال ۱۵-۱۸)، می‌توان با دستور GOTO از حلقهٔ تکرار FOR خارج شد. ولی برای جلوگیری از اشتباهات احتمالی بهتر است از آن پرهیز شود.

دستور GOSUB

برای اجرای زیربرنامه‌ها از دستور GOSUB استفاده می‌گردد و شکل کلی آن به صورت زیر می‌باشد:

```
GOSUB      <line>
:
:
RETURN
```

برای اجرای دستور GOSUB ، همانند دستور FOR به پشته نیاز است. این پشته به صورت زیر تعریف شده است. متغیر SUB_NEST تعداد زیربرنامه‌های متداخل (لانه‌ای) را مشخص می‌کند :

```
char  *gstack [SUB_NEST] ;
int   gtos ;
```

مثال ۱۴-۱۸

تابع gosub() برای اجرای دستور GOSUB .لیست این تابع در مثال ۱۵-۱۸ آمده است.

برنامهٔ کامل مفسر بیسیک

تاکنون با مفاهیم اصلی و توابع مهم تهیه مفسر بیسیک آشنایی پیدا کردید. اکنون لیست کامل برنامه مفسر بیسیک را به همراه نمونه‌ای از اجرای آن، در مثال ۱۵-۱۸ می‌آوریم.

مثال ۱۵-۱۸

برنامه کامل مفسر زبان بیسیک.

```
#include <stdlib.h>
#include <conio.h>
#include <setjmp.h>
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#define NUM_LAB 100
#define LAB_LEN 10
#define FOR_NEST 25
#define SUB_NEST 25
```

```
#define PROG_SIZE 10000
#define DELIMITER 1
#define VARIABLE  2
#define NUMBER    3
#define COMMAND   4
#define STRING    5
#define QUOTE     6
#define PRINT     1
#define INPUT     2
#define IF        3
#define THEN      4
#define FOR       5
#define NEXT      6
#define TO        7
#define GOTO      8
#define EOL       9
#define FINISHED  10
#define GOSUB     11
#define RETURN    12
#define END       13
int load_program(char *p ,char  *fname);
char get_token();
void assignment();
int get_next_label(char *s);
int iswhite(char c);
int isdelim(char c);
char look_up(char *s);
int find_var(char *s);
char *gpop();
char *find_label(char *s);
struct for_stack fpop() ;
void print() , scan_labels() , find_eol(),exec_goto() ;
void exec_if() , exec_for(), next() ,fpush(struct for_stack i) ,input() ;
void gosub() , greturn() , gpush(char *s) , label_init() ;
void putback() , serror(int) , get_exp(int *) , primitive(int *result) ;
void level2(int *), level3(int *), level4(int *), level5(int *), level6(int *);
void arith(char o , int *r , int *h) , unary(char o , int *r) ;
char *prog ;
jmp_buf e_buf ;
int variable[26] = {0} ;
struct commands {
   char commands[20] ;
   char tok ;
```

```
} table[] =
  {
    "print", PRINT, "input", INPUT, "if",IF,
    "then", THEN, "goto", GOTO, "for", FOR ,
    "next", NEXT , "to", TO , "gosub", GOSUB ,
    "return", RETURN , "end", END , "", END
  } ;
char token[80], token_type, tok ;
struct label {
       char name[LAB_LEN] ;
       char *p ;
    } ;
struct label label_table[NUM_LAB] ;

struct for_stack {
     int var ;
     int target ;
     char * loc ;
 } fstack[FOR_NEST] ;
char *gstack[SUB_NEST] ;
int ftos, gtos ;

int main(int argc,  char *argv[])
 {
      char in[80] ;
      int answer ;
      char *p_buf , *t ;
      clrscr();
      if(argc != 2) {
          printf("\n usage :  run < filename > ") ;
          getch() ;
          exit(1) ;
      }
      p_buf = (char *) malloc(PROG_SIZE) ;
      if(!p_buf)  {
        printf("\n allocation failure ") ;
        getch() ;
        exit(1) ;
      }
       if(!load_program(p_buf, argv[1]))
       {
          printf("\n file not loaded ") ;
          getch() ;
```

```
            exit(1) ;
        }
        if(setjmp(e_buf))
            exit(1) ;
        prog = p_buf ;
        scan_labels() ;
        ftos = 0 ;
        gtos = 0 ;
        do {
            token_type = get_token() ;
            if(token_type == VARIABLE) {
                putback() ;
                assignment() ;
            }
            else
            switch(tok) {
                case PRINT :   print() ;  break ;
                case GOTO  :   exec_goto() ; break ;
                case IF    :   exec_if() ;  break ;
                case FOR   :   exec_for() ; break ;
                case NEXT  :   next() ;   break ;
                case INPUT :   input() ;  break ;
                case GOSUB :   gosub() ;  break ;
                case RETURN:   greturn() ; break ;
                case END   :   exit(0) ;
            }//end of switch
        } while (tok != FINISHED) ;
        free(p_buf) ;
        return 0;
    }
//************
int load_program(char *p , char  *fname)
{
        FILE *fp ;
        int i = 0 ;
        fp = fopen(fname, "rb") ;
        if(!fp)
            return 0 ;
        i = 0 ;
        do {
            *p = getc(fp) ;
            p++ ; i++ ;
        } while(!feof(fp) && i < PROG_SIZE) ;
```

```
        *(p - 2) = '\0' ;
        fclose(fp) ;
        return 1 ;
 }
 //****************
void assignment()
 {
      int var , value ;
      get_token() ;
      if(!isalpha(*token))  {
          serror(4) ;
          return ;
      }
      var = toupper(*token) - 'A';
      get_token() ;
      if(*token != '=') {
          serror(3) ;
          return ;
      }
      get_exp(&value) ;
      variable[var]=value ;
 }
 //****************
void print()
     {
       int answer , len=0 , spaces ;
       char last_delim ;
       do {
            get_token() ;
            if(tok == EOL || tok == FINISHED)
                 break ;
            if(token_type == QUOTE)  {
               printf(token) ;
               len += strlen(token) ;
               get_token() ;
            }
            else {
                putback() ;
                get_exp(&answer) ;
                get_token() ;
                len += printf("%d",answer) ;
            }
            last_delim=*token ;
```

```
            if(*token == ';') {
                spaces = 8 - (len % 8) ;
                len += spaces ;
                while(spaces) {
                    printf(" ") ;
                  spaces -- ;
                }
            }//end ofif
            else if(*token == ',') ;//do nothing
            else if(tok != EOL && tok != FINISHED)
                  serror(0) ;
        } while(*token == ';' || *token == ',') ;
        if(tok == EOL || tok == FINISHED) {
              if(last_delim != ';' && last_delim !=',')
                  printf("\n") ;
        }
        else serror(0) ;
    }
    //************************
    void scan_labels()
    {
        int addr ;
        char *temp ;
        label_init() ;
        temp = prog ;
        get_token() ;
        if(token_type == NUMBER) {
            strcpy(label_table[0].name, token) ;
            label_table[0].p = prog ;
        }
        find_eol() ;
        do {
            get_token() ;
            if(token_type == NUMBER) {
                  addr = get_next_label(token) ;
                  if(addr == -1 || addr == -2) {
                      (addr == -1) ? serror(5) : serror(6) ;
                  } //end of if
                  strcpy(label_table[addr].name , token) ;
                  label_table[addr].p = prog ;
            }//end of if
            if(tok !=EOL)
            find_eol() ;
```

```
    } while(tok != FINISHED) ;
    prog = temp ;
}
//******************
void find_eol()
{
   while(*prog != '\n' && *prog != '\0')
      ++prog ;
    if(*prog)
      prog ++ ;
}
//******************
int get_next_label(char *s)
{
    register int t ;
    for(t = 0 ; t < NUM_LAB ; t++) {
      if(label_table[t].name[0] == 0)
          return t ;
      if(!strcmp(label_table[t].name,s))
        return -2 ;
    } //end of for
    return -1 ;
}
//**********************
char *find_label(char *s)
{
    register int t ;
    for(t = 0 ; t < NUM_LAB ; t++)
       if(!strcmp(label_table[t].name, s))
         return label_table[t].p ;
    return '\0' ;
}
void exec_goto()
{
    char *loc ;
    get_token() ;
    loc = find_label(token) ;
    if(loc == '\0')
        serror(7) ;
    else prog = loc ;
}
//*******************
void label_init()
```

```
{
    register int t ;
    for(t = 0 ; t < NUM_LAB ; t++)
        label_table[t].name[0] = '\0' ;
}
//*********************
void exec_if()
 {
    int x , y , cond ;
    char op ;
    get_exp(&x) ;
    get_token() ;
    if(!strchr("=<>", *token))  {
        serror(0) ;
        return ;
    }
    op = *token ;
    get_exp(&y) ;
    cond = 0 ;
    switch(op) {
        case '<' :
            if(x < y) cond=1 ;
            break ;
        case '>' :
            if(x > y) cond=1 ;
            break ;
        case '=' :
            if(x == y) cond=1 ;
            break ;
    }//end of switch
    if(cond) {
        get_token() ;
        if(tok != THEN)  {
            serror(8) ;
            return ;
        }
    }//end of if
    else find_eol() ;
}
//*********************
void exec_for()
{
    struct for_stack i ;
```

```
        int value ;
        get_token() ;
        if(!isalpha(*token)) {
              serror(4) ;
              return ;
        }
        i.var = toupper(*token) - 'A' ;
        get_token() ;
        if(*token != '=') {
            serror(3) ;
            return ;
        }
        get_exp(&value) ;
        variable[i.var] = value ;
        get_token() ;
        if(tok != TO) serror(9) ;
        get_exp(&i.target) ;
        if(value >= variable[i.var]) {
              i.loc = prog ;
              fpush(i) ;
        }
        else
           while(tok!=NEXT) get_token() ;
 }
 //**************************
void next()
 {
        struct for_stack i ;
        i = fpop() ;
        variable[i.var]++ ;
        if(variable[i.var] > i.target) return ;
        fpush(i);
        prog = i.loc ;
 }
 //************************
void fpush(struct for_stack i)
 {
        if(ftos > FOR_NEST)
            serror(10) ;
        fstack[ftos] = i ;
        ftos++ ;
 }
 //*********************
```

```
struct for_stack fpop()
 {
      ftos -- ;
      if(ftos < 0) serror(11);
      return(fstack[ftos]);
 }
 //***********************
void input()
  {
      char str[80], var ;
      int i , ch , digit = 0 ;
      get_token() ;
      if(token_type == QUOTE) {
          printf(token) ;
          get_token() ;
          if(*token != ',') {
              serror(1) ;
           }
           get_token() ;
      }
      else printf("?") ;
      var = toupper(*token) - 'A' ;
      while(1) {
          ch = getche() ;
          if(ch <= 57 && ch >= 48)  {
              ch = ch - 48 ;
              digit = (digit * 10) + ch ;
          }
          else if(ch == 13)
             break ;
          else {
              printf("\Redo from start ?\n") ;
              continue ;
          }
      }//end of while
      variable[var]=digit ;
 }
//**************
void gosub()
 {
      char *loc ;
      get_token() ;
      loc = find_label(token) ;
```

```
        if(loc == '\0')
           serror(7) ;
        else {
             gpush(prog) ;
             prog = loc ;
        }//end of else
   }
//*********************
void greturn()
    {
      prog = gpop() ;
    }
//**************
void gpush(char *s)
  {
       gtos++ ;
       if(gtos == SUB_NEST) {
            serror(12) ;
            return ;
       }
       gstack[gtos] = s ;
  }
//*******************
char *gpop()
  {
       if(gtos == 0) {
          serror(13) ;
          return 0 ;
       } //end of return
        return(gstack[gtos --]) ;
  }
//*****************
void putback()
  {
       char *t ;
       t = token ;
       for(; *t ; t++)
           prog -- ;
  }
//*****************
void serror(int error)
    {
      static char *e[] = {
```

```
            "syntax error ",
            "un balanced parantheses",
            "no expressoin present" ,
            "equals sign expexted" ,
            "not a variable" ,
            "label table full" ,
            "dulicate label" ,
            "undifined label" ,
            "THEN expected" ,
            "TO expected" ,
            "too many nested FOR loops" ,
            "NEXT without FOR" ,
            "too many nested GOSUB",
           "RETURN without GOSUB"
        } ;
        printf("\n %s \n", e[error]) ;
        longjmp(e_buf , 1) ;
    }
//*****************
char get_token()
{
     register char *temp ;
     token_type = 0 ;
     tok = 0 ;
     temp = token ;
     if(*prog == '\0')  {
         *token = 0 ;
         tok = FINISHED ;
         return(token_type = DELIMITER) ;
     }//end of if
     while(iswhite(*prog))
          ++ prog ;
          if(*prog == '\r')  {
              ++ prog ;
              ++ prog ;
              tok = EOL ;
              *token = '\r' ;
              token[1] = '\n' ;
              token[2] = 0 ;
              return(token_type = DELIMITER) ;
          }
          if(strchr("+-* ^ /%=;(),><" , *prog))  {
              *temp=*prog ;
```

```
            prog ++ ;
            temp ++ ;
           *temp = 0 ;
            return(token_type = DELIMITER) ;
        }
        if(*prog == '"') {
              prog ++ ;
              while(*prog != '"' && *prog != '\r')
                  *temp ++ = *prog ++ ;
              if(*prog == '\r')
                    serror(1) ;
               prog ++ ;
               *temp = 0 ;
               return(token_type = QUOTE) ;
        } //end of if
        if(isdigit(*prog)) {
              while(!isdelim(*prog))
                    *temp ++= *prog ++ ;
              *temp='\0' ;
              return(token_type=NUMBER) ;
        }
        if(isalpha(*prog)) {
             while(!isdelim(*prog))
                *temp ++= *prog ++ ;
             token_type = STRING ;
        }
        *temp='\0' ;
        if(token_type == STRING) {
            tok = look_up(token) ;
            if(!tok)
                token_type = VARIABLE ;
            else
                token_type = COMMAND ;
        }
        return token_type ;
}
char look_up(char *s)
{
     register int i , j ;
     char *p ;
     p = s ;
     while(*p) {
          *p = tolower(*p) ;
```

```
            p ++ ;
        }
        for( i= 0 ; *table[i].commands ; i++)
                if(!strcmp(table[i].commands , s))
                        return table[i].tok ;
        return 0 ;
 }
 //************
int isdelim(char c)
  {
    if(strchr(" ;,+-<>/*%^=()",c) || c == 9 ||  c=='\r' || c==0)
            return 1 ;
     return 0 ;
  }
 //**************
int iswhite(char c)
  {
       if(c == ' ' || c == '\t')
             return 1 ;
        else
             return 0 ;
  }
 //***********************
void get_exp(int *result)
 {
       get_token() ;
       if(!*token)  {
           serror(2) ;
           return ;
       }
       level2(result) ;
       putback() ;
 }
 //**********************
void level2(int *result)
  {
      register char op ;
      int hold ;
      level3(result) ;
      while((op = *token) == '+' || op == '-')  {
          get_token() ;
          level3(&hold) ;
          arith(op , result , &hold) ;
```

```
    }//end of while
 }
//****************************
void level3(int * result)
{
      register char op ;
      int hold ;
      level4(result) ;
      while((op = *token) == '*' || op == '/' || op == '%')
      {
           get_token() ;
           level4(&hold) ;
           arith(op , result , &hold) ;
      } //end of while
}
//******************
void level4(int * result)
{
     int hold ;
     level5(result) ;
     if(*token == '^') {
           get_token() ;
           level4(&hold) ;
           arith('^' , result , &hold) ;
     }
}
//*********************
void level5(int *result)
 {
      register char op ;
      op = 0 ;
      if((token_type == DELIMITER) && *token == '+' || *token == '-')
      {
        op = *token ;
        get_token() ;
      }
      level6(result) ;
      if(op)
           unary(op , result) ;
}
//*******************************
void level6(int *result)
{
```

```
    if((*token == '(') &&(token_type == DELIMITER))
    {
        get_token() ;
        level2(result) ;
        if(*token != ')')
            serror(1) ;
        get_token() ;
    }
    else
        primitive(result) ;
}
//***********************
void primitive(int *result)
{
    switch(token_type) {
        case VARIABLE :
            *result = find_var(token) ;
            get_token() ; return ;
        case NUMBER :
            *result = atoi(token) ;
            get_token() ; return ;
        default :
            serror(0) ;
    }//end of switch
}
//*********************
void arith(char o , int *r , int *h)
{
    register int t , ex ;
    switch(o) {
      case '-' : *r = *r - *h ; break ;
      case '+' : *r=*r + *h ; break ;
      case '*' : *r=*r * *h ; break ;
      case '/' : *r=(*r) / (*h) ; break ;
      case '%' : t = (*r) / (*h) ;  *r = *r - (t * (*h)) ; break ;
      case '^' : ex = *r ;
                 if(*h == 0) {
                   *r = 1 ;
                   break ;
                 }
                 for(t = *h - 1 ; t > 0 ; -- t)
                       *r=(*r) * ex ;
                 break ;
```

```
    }//end of switch
}
//*************************
void unary(char o , int *r)
{
     if(o == '-')
         *r = -(*r) ;
}
//***************
int find_var(char *s)
 {
     if(!isalpha(*s))   {
         serror(4) ;
         return 0 ;
     }
     return  variable[toupper(*token)-'A'] ;
}
//**********************
```

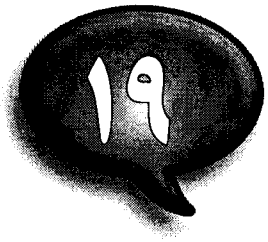
برای اطمینان از صحت برنامهٔ مثال ۱۵-۱۸، یک برنامهٔ بیسیک به نام BASIC، برای جدول ضرب نوشته شده است که لیست این برنامه و نتیجهٔ اجرای آن توسط مفسر، در زیر مشاهده می‌گردد:

```
print "          << product table >> "
for i=1 to  5
   for k=1 to 5
     s=i*k
     print i,"*",k,"=",s ;
   next
   print ""
next
print  "----------------------";
print  "----------"
```

خروجی

```
C:\BC> run calc2

                              << product table >>
     1*1=1        1*2=2        1*3=3        1*4=4        1*5=5
     2*1=2        2*2=4        2*3=6        2*4=8        2*5=10
     3*1=3        3*2=6        3*3=9        3*4=12       3*5=15
     4*1=4        4*2=8        4*3=12       4*4=16       4*5=20
     5*1=5        5*2=10       5*3=15       5*4=20       5*5=25
   ------------------------------------------------------  ---------------------------------------
```



توابع کتابخانه‌ای

توابعی که در این فصل مورد بررسی قرار می‌گیرند عبارتنداز: توابعی در مورد تاریخ، زمان و دیگر توابع سیستم، تعدادی از توابع تخصیص یافته حافظه پویا، توابع کنترلی، توابع ورودی - خروجی و توابع متفرقه.

توابعی در مورد تاریخ، زمان و دیگر توابع سیستم

الگوی توابعی که در رابطه با زمان هستند، در فایل time.h قرار دارد. در این فایل دو نوع ساختمان تعریف شده است که عبارت‌اند از: date و tm. نوع ساختمان دیگری به‌نام time در فایل dos.h تعریف شده است که در توابع تاریخ به‌کار می‌روند. نوع time_t نیز در فایل time.h تعریف شد که قادر است زمان و تاریخ سیستم را به صورت یک عدد صحیح بزرگ نمایش دهد. انواع date، tm و time به صورت زیر تعریف شده‌اند:

```
struct tm {
        int tm_sec ;
        int tm_min ;
        int tm_hour ;
        int tm_mday ;
        int tm_mon ;
        int tm_year ;
        int tm_wday ;
        int tm_yday ;
        int tm_isdst ; /* daylight savings
                          time indicator */
} ;

 //**************************
struct date {
        int   da_year ;
        char  da_day ;
        char  da_mon ;
} ;

struct time {
        unsigned char ti_min ;
        unsigned char ti_hour ;
        unsigned char ti_hund ;
        unsigned char ti_sec ;
} ;
```

توابع absread() و abswrite()

تابع absread() برای خواندن اطلاعات از روی دیسک و انتقال آن به بافر و تابع abswrite() برای نوشتن اطلاعات موجود در یک بافر بر روی دیسک به کار می‌رود. این توابع، با سکتورها کار می‌کنند و الگوی آنها در فایل dos.h قرار دارد و به صورت زیر است :

```
int absread (int drive, int numsects, int sectnum, void  *buf)
int abswrite (int drive, int numsects, int sectnum, void  *buf)
```

در الگوی فوق، drive مشخص کنندهٔ درایوی است که اطلاعات موجود در آن باید خوانده شود (0 برای درایو A، 1 برای درایو B و ...). numsects ، تعداد سکتورهایی که باید خوانده یا نوشته شوند و sectnum، شماره سکتوری را که خواندن و یا نوشتن باید از آنجا آغاز شود، مشخص می‌کند. buf بافری است که اطلاعات خوانده شده توسط تابع absread() باید در آنجا قرار گیرد و یا حاوی اطلاعاتی است که در تابع abswrite() باید بر روی دیسک نوشته شود. اگر اعمال توابع فوق با موفقیت انجام شود عدد صفر وگرنه مقداری غیر از صفر را برمی‌گرداند.

مثال ۱-۱۹

برنامه‌ای که چگونگی عملکرد تابع absread() را نشان می‌دهد.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
int main()
 {
    char buf[512] ;
    int sector, i, j ;
    for(;;) {
      clrscr();
      printf("enter sector(-1 to end):") ;
      scanf("%d",&sector) ;
      if(sector == -1) exit(0) ;
      absread(3, 1, sector, buf) ;
      for(i = 0, j = 0; i < 512; i++)   {
         printf("%x ", buf[i]) ;
         if(!(i % 16))  {
             for( ; j < i ; j++)
                printf("%c", buf[j]) ;
             printf("\n") ;
         } //end of if
      } //end of for i = 0
      printf("\n press a key to continue..");
      getch();
    } //end of for ;;
 }
```

توابع time()، asctime() و localtime()

تابع time()، زمان جاری سیستم را مشخص می‌کند، تابع localtime() به یک ساختمان از نوع tm که حاوی اجزای زمان و تاریخ است اشاره می‌کند و تابع asctime() ساختمانی از نوع tm را به رشته‌ای با فرمت زیر تبدیل می‌کند:

```
day month date  hours:minutes:second year\n\0
Wed Jun 19 12:05:34 1999
```

الگوی توابع فوق به صورت زیر است:

```
time_t time(time_t  *time)
struct tm  *localtime(time_t  *time)
char *asctime(struct tm *ptr)
```

مثال ۲-۱۹

برنامه‌ای که چگونگی عملکرد توابع time()، asctime() و localtime() را نشان می‌دهد.

```
#include <stdio.h>
#include <time.h>
#include <conio.h>
#include <stddef.h>
int main()
  {
    struct tm *ptr ;
    time_t  lt ;
    clrscr();
    lt = time(NULL) ;
    ptr = localtime(&lt) ;
    printf(asctime(ptr)) ;
    getch();
    return 0;
  }
```

تابع bioscom()

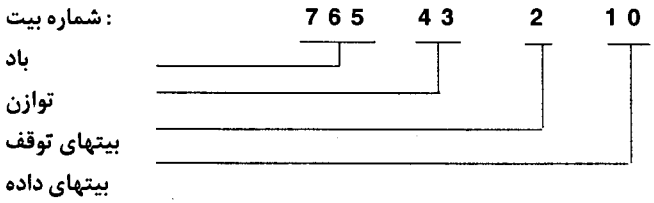
تابع bioscom() برای دستکاری پورت ارتباطی غیرهمزمان RS232 به کار می‌رود. الگوی این تابع در فایل bios.h قرار دارد و به صورت زیر است:

```
int bioscom (int cmd, char byte, int port)
```

در الگوی فوق، پارامتر cmd نوع عمل تابع را مشخص می‌کند که مقادیر معتبر آن به صورت زیر می‌باشد:

cmd	مفهوم
۰	ارزش دهی به پورت
۱	فرستادن کاراکتر به پورت
۲	دریافت کاراکتر از پورت
۳	تشخیص وضعیت پورت

port برای تعیین پورت مورد استفاده قرار می‌گیرد. قبل از کارکردن با پورت باید آن را ارزش‌دهی نمود که برای این منظور از پارامتر byte استفاده می‌شود:



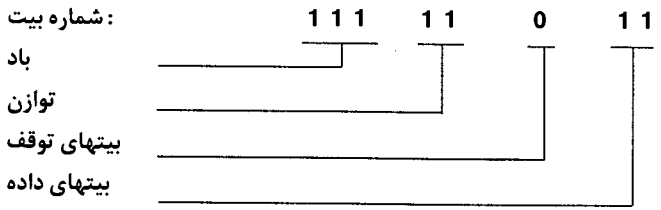
برای تعیین میزان باد(baud) باید بیتهای مربوط را به یکی از روشهای زیر، مقداردهی کرد:

بیت	باد
111	9600
110	4800
101	2400
100	1200
011	600
010	300
001	150
000	110

برای تعیین توازن باید به صورت زیر عمل شود:

بیت	توازن
1 0 یا 0 0	عدم توازن
0 1	فرد
1 1	زوج

تعداد بیتهای توقف، توسط بیت شماره ۲ مشخص شده است. اگر این بیت برابر با ۱ باشد ۲ بیت توقف وگرنه ۱ بیت توقف مورد استفاده قرار خواهد گرفت. تعداد بیتهای قابل انتقال نیز توسط بیتهای ۰ و ۱ مشخص می‌شود که از ۴ حالت ممکن، فقط ۲ حالت قابل قبول می‌باشد. در حالت ۱۰ تعداد بیتها برابر با ۷ و در حالت ۱۱ تعداد بیتها برابر با ۸ منظور خواهد شد. به عنوان مثال، برای ارزش‌دهی پورت جهت: سرعت انتقال ۹۶۰۰، اولویت زوج، یک بیت توقف و دادهٔ ۸ بیتی، باید از الگوی زیر استفاده کرد:



مقداری که تابع bioscom() بر می‌گرداند، عددی ۱۶ بیتی است که ۸ بیت باارزش، وضعیت پورت را مشخص می‌کند:

بیت	مفهوم آن، وقتی یک باشد
0	Data ready
1	Overnun error
2	Parity error
3	Framing error
4	Break-detect error
5	Transfer holding register empty
6	Transfer shift register empty
7	Time-out error

اگر پارامتر cmd برابر با ۲ باشد، ۸ بیت کم‌ارزش حاوی مقداری است که از پورت خوانده می‌شود. به عنوان مثال، دستور ; (0, 251, 0) bioscom پورت شماره صفر را با سرعت انتقال ۹۶۰۰، اولویت زوج، بیت توقف ۱ و دادهٔ ۸ بیتی ارزش‌دهی می‌کند. اگر پارامتر cmd یکی از مقادیر ۰، ۱ یا ۳ باشد، ۸ بیت کم‌ارزش دارای مفهوم زیر خواهند بود:

بیت	مفهوم آن، وقتی یک باشد
0	Change in clear-to-send
1	Change in data-set-ready
2	Trailing-edge ring detector
3	Change in line signal
4	Clear-to-send
5	Data-set-ready
6	Ring indicator
7	Line signal detected

تابع ()biosdisk

تابع ()biosdisk برای انجام اعمالی بر روی دیسک با استفاده از وقفهٔ 0x13 به کار می‌رود و دارای الگوی زیر است (الگوی این تابع در فایل bios.h است):

int biosdisk (int cmd, int drive, int head, int track, int sector, int nsects, void *buf)

در الگوی فوق، drive درایوی را مشخص می‌کند که عمل تابع باید بر روی آن انجام شود (۰ برای A، ۱ برای B، 0x80 برای اولین دیسک سخت، 0x81 برای دومین دیسک و ...). پارامترهای head ، track ، sector و nsect به ترتیب شمارهٔ هِد خواندن و نوشتن، شماره تراک، شماره سکتور و تعداد سکتورها را مشخص می‌کنند. پارامتر buf محلی برای ذخیره کردن اطلاعات جهت خواندن و یا نوشتن بر روی دیسک است.

تابع ()biosequip

تابع ()biosequip اطلاعاتی را در مورد کامپیوتر در اختیار قرار می‌دهد. الگوی این تابع در فایل bios.h قرار دارد و به صورت زیر است :

int biosequip (void)

مقداری که توسط این تابع برگردانده می‌شود یک عدد ۱۶ بیتی است که مفهوم بیتهای آن در جدول ۱-۱۹ آمده‌است.

مثال ۳-۱۹

برنامه‌ای که تعداد درایوهای فلاپی را که بر روی کامپیوتر نصب شده است مشخص می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <bios.h>
int main()
 {
    unsigned equip ;
    clrscr();
    equip = biosequip() ;
    equip >>= 6 ; /* shift bit 6 , 7
      into lowest position */
    printf("\n number of disk driver: %d", equip & 3 ) ;
    getch();
    return 0;
 }
```

جدول ۱-۱۹ مقادیری که تابع ()biosequip برمی‌گرداند.

شماره بیت	تجهیزات
۰	از روی A باید راه‌اندازی شود
۱	پیش پردازنده ۸۰۲۸۷ نصب شده است
۲ و ۳	میزان حافظه RAM بورد اصلی : ۰۰:۱۶ کیلوبایت ۰۱:۳۲ کیلوبایت ۱۰:۴۸ کیلوبایت ۱۱:۶۴ کیلوبایت
۴ و ۵	حالت صفحه‌نمایش ۰۰: استفاده نشده است ۰۱:۲۵ × ۴۰ ، سیاه و سفید با تطبیق دهنده رنگی ۱۰:۲۵ × ۸۰ ، سیاه و سفید با تطبیق دهنده رنگی ۱۱:۲۵ × ۸۰ سیاه و سفید
۶ و ۷	تعداد درایوهای فلاپی ۰۰:یک عدد ۰۱:دو عدد ۱۰:سه عدد ۱۱:چهار عدد
۸	تراشه DMA نصب شده است
۹ و ۱۰ و ۱۱	تعداد پورتهای سری ۰۰۰:ندارد ۰۰۱:یک عدد ۰۱۰:دو عدد ۰۱۱:سه عدد ۱۰۰:چهار عدد ۱۰۱:پنج عدد ۱۱۰:شش عدد ۱۱۱:هفت عدد
۱۲	آداپتور بازی نصب شده است
۱۳	چاپگر سری نصب شده است (فقط در PCjr)
۱۴ و ۱۵	تعداد چاپگرها: ۰۰:ندارد ۰۱:یک عدد ۱۰:دو عدد ۱۱:سه عدد

تابع bioskey()

تابع bioskey() بر روی صفحه کلید عمل می‌کند و الگوی آن در تابع bios.h قرار دارد و به صورت زیر است:

```
int bioskey (in cmd)
```

در الگوی فوق، cmd نوع عمل تابع را مشخص می‌کند. اگر برابر با صفر باشد، کد کلیدبعدی را برمی‌گرداند (منتظر می‌ماند تا کلیدی فشار داده شود). نتیجه عمل تابع یک مقدار ۱۶ بیتی است؛ اگر یک کلید معمولی فشار داده شود ۸ بیت باارزش آن حاوی کداسکی وگرنه حاوی صفر خواهد بود. اگر کلید وارد شده یکی از کلیدهای باکد توسعه‌یافته باشد، ۸بیت کم‌ارزش برابر با صفر و ۸بیت باارزش حاوی کد کلیدهای باکدتوسعه‌یافته می‌باشد.

اگر cmd برابر با ۱ باشد، فشار دادن کلید را تست می‌کند. اگر کلیدی فشار داده شد، مقداری غیر از صفر وگرنه مقدار صفر توسط این تابع برگردانده می‌شود. اگر cmd برابر با ۲ باشد، وضعیت صفحه‌کلید در ۸بیت کم‌ارزشِ مقدار برگردانده شده توسط این تابع، قرار می‌گیرد:

شماره بیت	مفهوم
۰	کلید shift راست فشار داده شده است
۱	کلید shift چپ فشار داده شده است
۳	کلید ctrl فشار داده شده است
۴	کلید alt فشار داده شده است
۵	scroll lock روشن است
۶	numlock روشن است
۷	capslock روشن است
۸	حالت درج (Insert on)

تابع biosmemory()

تابع biosmemory() مقدار حافظه سیستم را مشخص می‌کند (کیلوبایت). الگوی این تابع در فایل bios.h قرار داشته و به صورت زیر است:

```
int  biosmemory (void);
```

مثال ۴-۱۹

برنامه‌ای که میزان حافظه سیستم را در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <bios.h>
int main()  {
    clrscr();
    printf("\n %d kilo byte of RAM.", biosmemory()) ;
    getch();
    return 0;
  }
```

تابع biosprint()

تابع biosprint() برای کنترل چاپگر مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل bios.h قرار داشته و به صورت زیر است:

int biosprint (int cmd, int byte, int port)

در الگوی فوق، port شمارهٔ پورتی است که چاپگر به آن متصل است (۰ برای LPT1 و ۱ برای LPT2). cmd نوع عمل تابع را مانند جدول مقابل مشخص می‌کند:

مقدار cmd	مفهوم
۰	چاپ کاراکتر موجود در پارامتر byte
۱	ارزش دهی به پورت چاپگر
۲	گزارش وضعیت پورت

وضعیت پورت در ۸ بیت کم‌ارزش مقدار برگردانده شده توسط تابع قرار می‌گیرد و به صورت جدول مقابل می‌باشد:

مقدار cmd	مفهوم
۰	خطای اتمام وقت
۴	خطای ورودی - خروجی
۵	چاپگر در حال کار است
۶	کاغذ چاپگر تمام شده است
۷	چاپگر آماده است
۸	چاپگر مشغول کار نیست

مثال ۵-۱۹

برنامه‌ای که وضعیت چاپگر را تست می‌کند و گزارش مناسبی ارائه می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>
int main(void)
{
   #define STATUS  2    /* printer status command */
   #define PORTNUM 0    /* port number for LPT1 */
   int status, abyte=0;
   clrscr();
   printf("Please turn off your printer. Press any key to continue\n");
   getch();
   status = biosprint(STATUS, abyte, PORTNUM);
   if (status & 0x01)  printf("Device time out.\n");
   if (status & 0x08)  printf("I/O error.\n");
   if (status & 0x10)  printf("Selected.\n");
   if (status & 0x20)  printf("Out of paper.\n");
   if (status & 0x40)  printf("Acknowledge.\n");
   if (status & 0x80)  printf("Not busy.\n");
   getch();
   return 0;
}
```

تابع disable()

تابع disable() از فراخوانی وقفه‌ها (به جز وقفه‌هایی که جلوگیری از آنها غیرممکن باشد) جلوگیری می‌کند. الگوی این تابع در فایل dos.h قرار دارد و به صورت زیر است:

```
void disable (void)
```

تابع enable()

تابع enable() عکس عمل تابع disable() را انجام می‌دهد؛ یعنی موجب می‌شود تا بتوان وقفه‌ها را فراخوانی نمود. الگوی این تابع در فایل dos.h قرار دارد و به صورت زیر است:

```
void enable (void)
```

توابع FP_OFF()، FP_SEG() و MK_FP()

تابع FP_OFF() قسمت تفاوت مکان از یک اشاره‌گر far و تابع FP_SEG() قسمت آدرس سگمنت از یک اشاره‌گر far را مشخص می‌کنند. تابع MK_PF با توجه به آدرس سگمنت و تفاوت مکان، یک اشاره‌گر far مناسب با آنها مشخص می‌کند. الگوی این توابع در فایل dos.h قرار داشته و به صورت زیر است:

```
unsigned  FP_OFF (void far  *ptr)
unsigned  FP_SEG (void far  *ptr)
unsigned  MK_FP  (unsigned seg, unsigned ofs)
```

در الگوهای فوق، ptr اشاره‌گری از نوع far است که این توابع بر روی آن عمل می‌کنند (این توابع به‌صورت ماکرو پیاده‌سازی شده‌اند).

مثال ۶-۱۹

برنامه‌ای که آدرس سگمنت و تفاوت مکان یک اشاره‌گر را مشخص می‌کند.

```
#include <dos.h>
#include <stdio.h>
#include <graphics.h>
int fp_off(void)
{
    char *str = "fpoff.c";
    printf("The offset of this file name in memory\
      is: %Fp\n", FP_OFF(str));
    return 0;
}
/* FP_SEG */
int fp_seg(void)
{
    char *filename = "fpseg.c";
```

```
    printf("The segment of this file in memory is: %Fp\n", FP_SEG(filename));
    return(0);
}
/* MK_FP */
int main(void)
{
      int gd, gm, i;
      unsigned int far *screen;
      detectgraph(&gd, &gm);
      if (gd == HERCMONO)
        screen = (unsigned int *) MK_FP(0xB000, 0);
      else  screen = (unsigned int *) MK_FP(0xB800, 0);
      for (i = 0; i < 26; i++)
        screen[i] = 0x0700 + ('a' + i);
      return 0;
}
```

توابع ()getdate و ()gettime

توابع ()getdate و ()gettime برای اخذ تاریخ و زمان سیستم به صورتی که در DOS نمایش داده می‌شود به کار می‌روند. الگوی این توابع در فایل dos.h قرار دارد و به صورت زیر است:

```
void  getdate  (struct date  *d)
void  gettime  (struct time  *t)
```

در الگوی فوق، t و d ساختمانهایی هستند که زمان و تاریخ جاری در آنها قرار خواهند گرفت.

مثال ۷-۱۹

برنامه‌ای که زمان و تاریخ جاری سیستم را نمایش می‌دهد.

```
#include <time.h>
#include <stddef.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
int main()
 {
   time_t t ;
   struct time dos_time ;
   struct date dos_date ;
   struct tm *local ;
   clrscr();
   getdate(&dos_date) ;
   gettime(&dos_time) ;
```

```
    t = dostounix(&dos_date, &dos_time) ;
    local = localtime(&t) ;
    printf("time & date:%s", asctime(local)) ;
    getch();
    return 0;
}
```

تابع getdfree()

تابع getdfree() میزان فضای آزاد دیسک را مشخص می‌کند. الگوی این تابع در فایل dos.h قرار دارد و به صورت زیر است:

void getdfree (int drive, struct dfree *dfptr)

در الگوی فوق، drive مشخص‌کنندهٔ درایوی است که این تابع باید بر روی آن عمل کند (۱ برای A، ۲ برای B و ...)، اگر برابر با صفر باشد، درایو جاری منظور خواهد شد. اگر در حین عمل تابع خطایی رخ دهد، مقدار عنصر df_sclus از ساختمان dfree برابر با ۱- خواهد بود. ساختمان dfree به صورت زیر تعریف شده است:

```
struct dfree {
        unsigned df_avail ;
        unsigned df_total ;
        unsigned df_bsec ;
        unsigned df_sclus ;
} ;
```

مثال ۸-۱۹

برنامه‌ای که فضای خالی دیسک C را مشخص می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
int main()
{
   struct dfree  p ;
   clrscr();
   getdfree(3, &p) ; /* drive c */
   printf("number of free cluster is: %d", p.df_avail) ;
   getch();
   return 0;
}
```

تابع getdta()

تابع getdta() آدرس DTA را مشخص می‌کند. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است :

```
char far *getdta (void)
```

به عنوان مثال، مجموعه دستورات زیر، آدرس DTA را مشخص می‌کند :

```
char far *ptr ;
ptr = getdta () ;
```

توابع getfat() و getfatd()

تابع getfat() اطلاعات گوناگونی در مورد دیسک موجود در یک درایو را در اختیار قرار می‌دهد. الگوی توابع فوق در فایل dos.h قرار داشته و به صورت زیر می‌باشد :

```
void getfat (int drive, struct fatinfo *fptr)
void getfatd (struct fatinfo *fptr)
```

در الگوهای فوق، fptr سگمنتی است که اطلاعات باید در آنجا ذخیره شود و drive مشخص‌کننده درایوی است که تابع getfat() باید بر روی آن عمل کند. ساختمان fatinfo دارای ساختار زیر است :

```
struct fatinfo {
        char fi_sclus ;
        char fi_farid ;
        int  fi_nclus ;
        int  fi_bysec ;
} ;
```

تابع getfatd() فقط بر روی درایو جاری عمل می‌کند .

مثال ۹-۱۹

برنامه‌ای که فضای دیسک موجود در درایو جاری را مشخص می‌کند.

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
int main()
{
   long total ;
   struct fatinfo p ;
   clrscr();
   getfat(0, &p) ;
   total = (long) p.fi_sclus*
     (long)p.fi_nclus*(long)p.fi_bysec ;
   printf("\n total storage capasity: %ld", total) ;
   getch();
```

```
    return 0;
}
//*****************
struct ftime {
        unsigned ft_tsec:5 ;
        unsigned ft_min:6 ;
        unsigned ft_hour:5 ;
        unsigned ft_day:5 ;
        unsigned ft_month:4 ;
        unsigned ft_year:7 ;
} ;
```

تابع getftime()

تابع getftime() زمان و تاریخ تشکیل یک فایل را مشخص می‌کند. الگوی این تابع در dos.h قرار داشته و به صورت زیر است:

int getftime (int handle, struct ftime *fprt)

در الگوی فوق، handle شماره فایل را مشخص می‌کند و fptr ساختمانی است که حاوی تاریخ و زمان تشکیل فایل خواهد بود. ساختمان ftime به صورت زیر تعریف شده است:

```
struct   ftime  {
         unsigned   ft_tsec:5  ;
         unsigned   ft_min:6   ;
         unsigned   ft_hour:5  ;
         unsigned   ft_day:5   ;
         unsigned   ft_month:4 ;
         unsigned   ft_year:7  ;
} ;
```

مثال ۱۰-۱۹

برنامه‌ای که تاریخ و زمان تشکیل فایل test.$$$ را مشخص می‌کند.

```
#include <stdio.h>
#include <io.h>
#include <conio.h>
int main(void)
{
   FILE *stream;
   struct ftime ft;
   clrscr();
   if ((stream = fopen("TEST.$$$", "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
```

```
        return 1;
    }
    getftime(fileno(stream), &ft);
    printf("File time: %u:%u:%u\n", ft.ft_hour, ft.ft_min, ft.ft_tsec * 2);
    printf("File date: %u/%u/%u\n",  ft.ft_month, ft.ft_day, ft.ft_year+1980);
    fclose(stream);
    getch();
    return 0;
}
```

تابع getpsp()

تابع getpsp() سگمنتی را که psp در آن جا قرار دارد مشخص می‌کند. الگوی این تابع، در فایل dos.h قرار داشته و به صورت زیر است :

```
unsigned getpsp (void)
```

توابع inport() و inportb()

تابع inport() کلمهٔ خوانده شده از پورت و تابع inportb() بایت خوانده شده از پورت را برمی‌گردانند. الگوی این توابع در فایل dos.h قرار داشته و به صورت زیر است :

```
int  inport (int port)
int  inportb (int port)
```

در الگوهای فوق، port شمارهٔ پورتی است که توابع فوق باید بر روی آن عمل کنند. به عنوان مثال، دستورات زیر، کلمه‌ای را از پورت شماره ۱ می‌خواند :

```
unsigned int  i;
i = inport (1)
```

توابع harderr()، hardresume() و hardretn()

تابع harderr() ، یک تابع پاسخگویی را به جای برنامهٔ پاسخگویی سیستم عامل DOS در موقع بروز اشکالات سخت‌افزاری، جایگزین می‌کند. این تابع در موقع صدور وقفه 24H اجرا خواهد شد. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است :

```
void harderr (int (*int_handler) ( ))
```

اگر نام تابع پاسخگویی به خطا err_handler در نظر گرفته شود این تابع باید دارای الگوی زیر باشد:

```
err_handler (int errnum, int ax, int bp, int si)
```

در الگوی فوق، errnum کد خطای DOS است و ax ، bp و si به ترتیب حاوی مقادیر ثباتهای AX ، BP و SI هستند. اگر ax مقداری مثبت باشد، بیانگر بروز خطایی در رابطه با دیسکها است. در این صورت اگر این مقدار با 0xFF ، AND شود، نتیجهٔ حاصل شماره درایوی است که خطا در آنجا رخ داده است. (۱ برای A و ...). اگر ax منفی

باشد، خطای حاصل در مورد دستگاهها است که آدرس کنترل کننده آن در ثباتهای BP و SI قرار خواهد گرفت.
در مورد نوشتن توابع پاسخگویی به خطای سخت‌افزاری باید نکات زیر را در نظر داشت:

۱. برنامه پاسخگویی به وقفه نباید از توابع استاندارد بورلند C استفاده نماید.
۲. از توابع ۱ تا ۱۲ وقفه 21H می‌توان استفاده نمود.

برای خروج از تابع پاسخگویی وقفه (سخت‌افزاری)، می‌توان به دو طریق عمل کرد:

۱. استفاده از تابع ()hardresume جهت برگشت به DOS. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

```
void hardresume (int code)
```

۲. استفاده از تابع ()hardretn جهت برگشت به برنامه، الگوی این تابع در فایل dos.h قرار دارد و به صورت زیر است:

```
void hardretn (code)
```

در الگوهای فوق، code مقداری است که در حین خروج، به برنامه و یا DOS فرستاده می‌شود.

تابع ()keep

تابع ()keep با اجرای وقفه 31H موجب خاتمه اجرای برنامه می‌گردد؛ ولی آن را به صورت مقیم نگهداری می‌کند.
الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

```
void keep (int status, int size)
```

در الگوی فوق، status کدی است که به DOS برگردانده می‌شود. size اندازه برنامه‌ای است که باید به صورت مقیم باقی بماند. چون درک مفهوم برنامه‌های مقیم، مطالب دیگری را می‌طلبد، ذکر مثالی در مورد این تابع، چندان ساده نیست.

توابع ()outport و ()outportb

تابع ()outport برای انتقال یک کلمه و تابع ()outportb برای انتقال یک بایت به یک پورت مورد استفاده قرار می‌گیرند. الگوی این توابع در فایل dos.h قرار داشته و به صورت زیر است:

```
void outport (int port, int word)
void outportb (int port, char byte)
```

در الگوی فوق، port شماره پورتی است که اطلاعات باید به آن منتقل شوند، word و byte حاوی این اطلاعات می‌باشند. به عنوان مثال، دستور ; (outport (0x10, oxFF موجب انتقال مقدار 0xFF به پورت شماره 0x10 می‌شود.

توابع peek()، poke()، peekb() و pokeb()

این توابع برای خواندن و نوشتن اطلاعات بر روی حافظه RAM مورد استفاده قرار می‌گیرند و دارای الگوی زیر هستند (این الگوها در فایل dos.h قرار دارند):

```
int  peek (int seg, unsigned off)
char peekb (int seg, unsigned off)
void poke (int seg, unsigned off, int word)
void pokeb (int seg, unsigned off, char byte)
```

تابع peek() محتویات ۱۶ بیت از حافظه را که در آدرس سگمنت seg و تفاوت مکان off قرار دارد می‌خواند. تابع peekb() محتویات یک بایت از حافظه را که در آدرس seg و تفاوت مکان off قرار دارد می‌خواند. تابع poke() اطلاعات موجود در کلمه word را در آدرس سگمنت seg و تفاوت مکان off می‌نویسد و بالاخره تابع pokeb() اطلاعات موجود در بایت byte را در آدرس سگمنت seg و تفاوت مکان off می‌نویسد.

مثال ۱۱-۱۹

برنامه‌ای که محتویات محل حافظه 000:0100 را نمایش می‌دهد.

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
int main() {
  clrscr();
  printf("%d", peekb(0, 0x0100)) ;
  getch();
  return 0;
}
```

توابع randbrd() و randbwr()

تابع randbrd() تعدادی از رکوردها را از آدرس انتقال دیسک (DTA) خوانده و به حافظه منتقل می‌کند و تابع randbwr() تعدادی از رکوردهای موجود در حافظه را به فایل منتقل می‌کند. الگوی این توابع در فایل dos.h قرار داشته و به صورت زیر است:

```
int randbrd (struct fcb  *fcbptr, int count)
int randbwr (struct fcb  *fcbptr, int count)
```

در الگوهای فوق، count تعداد رکوردهایی است که باید خوانده یا نوشته شوند و fcbptr ساختمان رکورد (طول رکورد) را مشخص می‌کند. مقادیری که توسط دو تابع randbrd() و randbwr() برگردانده می‌شوند عبارتنداز:

مقدار	مفهوم
۰	کلیه رکوردها با موفقیت خوانده یا نوشته شدند
۱	EOF اتفاق افتاده است ولی آخرین رکورد نوشته شده است
۲	تعداد رکوردها زیاد است
۳	EOF اتفاق افتاده ولی آخرین رکورد نوشته نشده است

تابع segread()

تابع segread() محتویات فعلی ثباتهای سگمنت را در ساختمانی از نوع REGS کپی می‌کند و برای توابع intdosx() و int86x() مفید است. الگوی تابع segread() در فایل dos.h قرار داشته و به صورت زیر است :

```
void segred (struct REGS  *regs)
```

توابع setdate() و settime()

تابع setdate() برای اعلام تاریخ جدید به سیستم و تابع settime() برای اعلام زمان جدید به سیستم استفاده می‌شود. الگوی این توابع در فایل dos.h قرار داشته و به صورت زیر است :

```
void setdate (struct date *d)
void settime (struct time *t)
```

مثال ۱۲-۱۹

برنامه‌ای که زمان جاری سیستم را 10:10:10.0 تعیین می‌کند.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
int main() {
   struct time t ;
   clrscr();
   t.ti_hour = 10 ;
   t.ti_min = 10 ;
   t.ti_sec = 10 ;
   t.ti_hund = 0 ;
   settime(&t) ;
   getch();
   return 0;
}
```

تابع setdta()

تابع setdta() برای تعیین آدرس DTA به کار می‌رود. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است :

```
void  setdta (char far *dta)
```

در الگوی فوق، dta حاوی آدرسی است که محل DTA را تعیین می‌کند. به عنوان مثال، دستورات زیر، موجب می‌شود تا محل 0000:A000 به عنوان آدرس DTA انتخاب شود.

```
char  far  *p ;
p = MK_FP (0xA000,0) ;
setdta (P) ;
```

تابع setverify()

تابع setverify() برای دستکاری فلگ بازبینی مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل dos.h قرار دارد و به صورت زیر است:

void setverify (int value)

در الگوی فوق، اگر value برابر با ۱ باشد فلگ بازبینی فعال و اگر برابر با ۰ باشد، فلگ بازبینی غیرفعال خواهد شد.

مثال ۱۳-۱۹

برنامه‌ای که موجب فعال شدن فلگ بازبینی می‌شود.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
int main() {
  clrscr();
  printf("turning the verify flag on.") ;
  setverify(1) ;
  getch();
  return 0;
}
```

تابع sleep()

تابع sleep() موجب به تعویق انداختن اجرای برنامه به مدت زمان معینی می‌شود. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

void sleep (unsigned time)

در الگوی فوق، time مدت زمانی (به ثانیه) است که اجرای برنامه باید به تعویق افتد.

مثال ۱۴-۱۹

برنامه‌ای که به مدت ۱۰ ثانیه متوقف می‌شود.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
int main() {
  clrscr();
  printf("\n sleep 10 second.") ;
  sleep(10) ;
  printf("\n good morning ") ;
  getch() ;
  return 0;
}
```

توابع تخصیص حافظه پویا

در فصل ۹ تعدادی از توابع تخصیص حافظه پویا مورد بررسی قرار گرفته‌اند. چون تشریح بعضی از توابع تخصیص حافظه به اطلاعات بیشتری نیاز داشتند و اکنون این امر حاصل شده است، به تشریح آنها می‌پردازیم.

تابع allocmem()

تابع allocmem() با استفاده از تابع 0x48 از وقفه 21H حافظه‌ای را که در مرز پاراگراف قرار دارد از سیستم اخذ می‌کند. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

```
int allocmem (unsigned size, unsigned  *seg)
```

در الگوی فوق، size تعداد پاراگرافهایی است که باید از سیستم اخذ شود (هر پاراگراف ۱۶ بایت است) و seg حاوی آدرس سگمنتی است که حافظه از آنجا اخذ شده است.

مثال ۱۵-۱۹

مجموعه دستوراتی که حافظه‌ای به میزان ۱۰۰ پاراگراف را از سیستم اخذ می‌کند.

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
int main()
{
   unsigned int i = 0 ;
   int j;
   clrscr();
   if((j = allocmem(100, &i)) == -1)
       printf("\n allocate successfull.") ;
   else  {
      printf("\n allocate failure, only  %u paragraphs available.", j);
   }
   getch();
   return 0;
}
```

تابع brk()

تابع brk() میزان حافظه لازم برای سگمنت داده یک برنامه را تغییر می‌دهد. الگوی این تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
int brk (void   *ends)
```

اگر عمل تابع با موفقیت انجام شود، انتهای سگمنت داده در ends قرار می‌گیرد و مقدار صفر توسط تابع برگردانده می‌شود و در غیر این صورت مقدار ۱- توسط تابع برگردانده خواهد شد.

تابع coreleft()

تابع coreleft() میزان حافظهٔ موجود در سمت چپ heap (جایی که حافظه خالی نگهداری می‌شود) را مشخص می‌کند. الگوی این تابع در فایل alloc.h قرار دارد و بر حسب این که در مدل حافظه small یا large به کار گرفته شود، الگوهای آن به صورت زیر است:

```
unsigned coreleft (void)  /* small data model */
unsigned long coreleft (void)  /* large data model */
```

مثال ۱۶-۱۹

برنامه‌ای که میزان حافظه heap را در مدل حافظه small مشخص می‌کند.

```
#include <alloc.h>
#include <conio.h>
#include <stdio.h>
int main() {
  clrscr();
  printf("the size of the heap is: %u", coreleft()) ;
  getch();
  return 0;
}
```

تابع faralloc()

تابع faralloc() مانند تابع calloc() است، با این تفاوت که حافظه تخصیص یافته، خارج از سگمنت داده جاری خواهد بود. الگوی این تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
void far  *faralloc (unsigned long num, unsigned long size)
```

تابع farcoreleft()

تابع farcoreleft() میزان حافظه قابل استفاده در far heap را مشخص می‌کند. الگوی این تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
long far coreleft (void)
```

مثال ۱۷-۱۹

برنامه‌ای که میزان حافظه far heap را مشخص می‌کند.

```
#include <alloc.h>
#include <conio.h>
#include <stdio.h>
int main() {
  clrscr();
  printf("far heap free memory is: %ld", farcoreleft()) ;
  getch();
  return 0;
}
```

توابع farmalloc() و farfree()

تابع farmalloc() همانند تابع malloc() است، با این تفاوت که حافظه لازم را از far heap تخصیص می‌دهد. تابع farfree() نیز همانند تابع free() است، با این تفاوت که بر روی حافظه‌ای که از far heap گرفته شده است عمل می‌کند. الگوی این دو تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
void far *farmalloc (unsigned long size)
void farfree (void far *ptr)
```

مثال ۱۸-۱۹

برنامه‌ای که میزان حافظه‌ای را از far heap اخذ نموده مجدداً برمی‌گرداند.

```
#include <alloc.h>
#include <conio.h>
#include <stdio.h>
int main()
{
  void far *p ;
  clrscr();
  p = farmalloc(100) ;
  if(p)
     farfree(p) ;
  getch();
  return 0;
}
```

تابع farrealloc()

تابع farrealloc() همانند تابع realloc() عمل می‌کند، با این تفاوت که عمل آن بر روی حافظه تخصیص یافته از far heap می‌باشد. الگوی این تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
void far *farrealloc (void *ptr, unsigned long newsize)
```

تابع freemem()

تابع freemem() برای برگرداندن حافظه‌ای که توسط تابع allocmem() تخصیص یافته است به کار می‌رود. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

```
int freemem (unsigned seg)
```

در الگوی فوق، seg آدرس شروع حافظه‌ای است که باید توسط تابع به سیستم برگردانده شود.

مثال ۱۹-۱۹

مجموعه دستوراتی که موجب اخذ حافظه از سیستم و سپس برگشت آن به سیستم می‌شود.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
int main()
{
   unsigned i, size ;
   clrscr();
   if(allocmem(size, &i) != -1)
      printf("\n allocation error.") ;
   else  freemem(i) ;
   getch();
   return 0;
}
```

تابع sbrk()

تابع sbrk() برای افزایش یا کاهش میزان حافظه تخصیص یافته جهت سگمنت داده به کار می‌رود. الگوی این تابع در فایل alloc.h قرار داشته و به صورت زیر است:

```
char *sbrk (int amount)
```

در الگوی فوق، amount میزان حافظه‌ای است که باید به سگمنت داده اضافه (در صورت مثبت بودن) و یا از سگمنت داده کم (در صورت منفی بودن) شود.

تابع setblock()

تابع setblock() برای تغییر میزان حافظهٔ تخصیص یافته توسط تابع allocmem() به کار می‌رود. الگوی این تابع در فایل dos.h قرار داشته و به صورت زیر است:

```
int setblock (int seg, int size)
```

در الگوی فوق، seg شروع قسمتی از حافظه است که توسط allocmem() اخذ شده است و size میزان جدید حافظه می‌باشد. اگر size مقدار مناسبی نباشد، بزرگترین مقدار حافظه ممکن، توسط این تابع تخصیص خواهد یافت. اگر عمل تابع با موفقیت انجام شود مقدار ۱- توسط آن برگردانده خواهد شد. به عنوان مثال، دستور زیر سعی می‌کند تا ۱۰۰ پاراگراف از سیستم اخذ نماید. در صورت عدم موفقیت، پیام مناسبی صادر خواهد شد.

```
if(setblock(seg, 100) != -1)
      printf("resize error.") ;
```

توابع کنترلی

منظور از توابع کنترلی، آن دسته از توابعی هستند که چگونگی روند اجرای برنامه، خاتمه اجرای برنامه و فراخوانی برنامه‌ای در داخل برنامه دیگر را مشخص می‌کنند. الگوی بعضی از این توابع در فایل process.h قرار دارند.

تابع abort()

تابع abort() موجب قطع اجرای فوری برنامه می‌شود. محتویات هیچگونه فایل داده بر روی دیسک قرار نخواهند گرفت و مقدار ۳ را به تابع فراخواننده برمی‌گرداند. الگوی این تابع به صورت زیر است:

```
void abort ()
```

مثال ۲۰-۱۹

برنامه‌ای که با واردنمودن حرف A اجرای آن خاتمه پیدا می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
int main()
{
  clrscr();
  for(;;)
    if(getch() == 'A')
      abort() ;
}
```

تابع atexit()

تابع atexit() تابعی را به برنامه معرفی می‌کند که در صورت خاتمه عادی اجرای برنامه، آن تابع اجرا می‌شود. الگوی این تابع در فایل stdlib.h قرار دارد و به صورت زیر است:

```
int atexit (func) void (*func) ( )
```

در الگوی فوق، func به تابعی اشاره می‌کند که در صورت خاتمه عادی اجرای برنامه، باید اجرا گردد. اگر عمل تابع با موفقیت انجام شود عدد صفر وگرنه مقداری غیر از صفر برگردانده می‌شود.

مثال ۲۱-۱۹

برنامه‌ای که نحوه کاربرد تابع atexit() را نشان می‌دهد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
void done();
int main()
{
  clrscr();
  if(atexit(done))
    printf("\n error in atexit()") ;
```

```
    getch();
    return 0;
}
//*****************
void done()
{
    printf("\n hello there.") ;
}
```

توابع ورودی - خروجی

بعضی از توابع ورودی - خروجی در فصلهای گذشته بنا به ضرورت مورد بررسی قرار گرفته‌اند. و بقیه توابع ورودی - خروجی در این فصل بررسی خواهند شد.

تابع ()_chmod

تابع ()chmod_ صفت یک فایل را تشخیص می‌دهد و یا صفت جدیدی به آن اعمال می‌کند. اگر عمل تابع با موفقیت انجام نشود عدد ۱- توسط تابع برگردانده خواهد شد.الگوی این تابع در فایل io.h قرار داشته و به صورت زیر است:

```
int _chmod (char  *filename, int get_set, int attrib)
```

در الگوی فوق، filename به فایلی اشاره می‌کند که صفت آن باید مشخص گردد. اگر get_set برابر با صفر باشد، صفت فعلی فایل در پارامتر attrib قرار می‌گیرد و اگر برابر با یک باشد صفت موجود در پارامتر attrib به فایل اعمال خواهد شد. پارامتر attrib می‌تواند یکی از ماکروهای زیر باشد:

ماکرو	مفهوم
FA_RDONLY	فایل فقط خواندنی
FA_HIDDEN	فایل مخفی
FA_SYSTEM	فایل سیستم

به عنوان مثال، دستور زیر موجب می‌شود تا فایل test.tst به صورت "فقط خواندنی" درآید.

```
if ( _chmod ("test.tst", 1, FA_RDONLY) == FA_RDONLY )
    printf ("file set to read_only mode") ;
```

تابع ()chmod

تابع ()chmod وضعیت فایل را از جهت خواندن یا نوشتن مشخص می‌کند. الگوی این تابع در io.h قرار داشته و به صورت زیر است:

```
int chmod (char  *filename, int mode)
```

در الگوی فوق، filename به نام فایلی اشاره می‌کند که تابع باید بر روی آن عمل کند و mode حالت فایل را مشخص می‌کند. اگر mode برابر با S_IWRITE باشد فایل به صورت نوشتنی خواهد بود؛ اگر برابر با S_IREAD باشد فایل به صورت خواندنی خواهد بود؛ و اگر ترکیبی از هر دو ماکرو باشد، فایل به صورت خواندنی - نوشتنی خواهد بود. به عنوان مثال، دستور زیر موجب می‌شود تا فایل test.tst به صورت خواندنی و نوشتنی قابل استفاده باشد.

```
if (! chmod ("test.tst", S_IWRITE | S_IREAD))
        printf ("file set to read / write") ;
```

تابع clearerr()

تابع clearerr() موجب می‌شود تا فلگ‌های (flags) خطای فایل، reset شوند تا برنامه بتواند ادامه پیدا کند. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر است:

```
void  clearerr (FILE  *fp)
```

در الگوی فوق fp به فایلی اشاره می‌کند که تابع باید بر روی آن عمل نماید.

مثال ۲۲-۱۹

برنامه‌ای که فایلی را بر روی فایل دیگر کپی کرده، در صورت بروز خطا، پیامی صادر می‌کند و سپس فلگ خطا پاک می‌شود.

```
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
void main(int argc , char *argv[])
 {
   FILE *in , *out;
   int tab , i ;
   char ch ;
   clrscr();
   if(argc != 3)  {
    printf("\n incorrect number of parameters.") ;
    getch() ;
    exit(1) ;
   }
   in = fopen(argv[1] , "rb") ;
   if (in == NULL)  {
      printf("\n can not open input file.") ;
      getch() ;
      exit(1) ;
    }
   out = fopen(argv[2] , "wb") ;
   if (out == NULL)  {
        printf("\n can not open output file.") ;
```

```
            getch() ;
            exit(1) ;
        }
        while(!feof(in))  {
          ch = getc(in) ;
          if (ferror(in))  {
             printf("error in input file.") ;
             clearerr(in) ;
          }
          else {
             putc(ch , out) ;
             if (ferror(out))  {
                printf("\n error in write.");
                clearerr(out) ;
             }//end of if
          }//end of else
        }//end of while
        fcloseall() ;
}
```

توابع filelength() و fileno()

تابع filelength() برای تعیین طول فایل (به بایت) مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل io.h قرار داشته و به صورت زیر است:

long filelength (int fd)

در الگوی فوق، fd شماره(handle) فایلی است که این تابع بر روی آن عمل می‌کند. برای تعیین شماره فایل از تابع fileno() استفاده می‌شود این تابع دارای الگوی زیر است:

int fileno (FILE *fp)

در الگوی فوق، fp اشاره‌گر فایلی است که شماره آن باید تعیین شود.

مثال ۲۳-۱۹

برنامه‌ای که طول فایلی را تشخیص می‌دهد.

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
int main(void)
{
   int handle;
   char buf[11] = "0123456789";
   /* create a file containing 10 bytes */
```

```
    handle = open("DUMMY.FIL", O_CREAT);
    write(handle, buf, strlen(buf));
    /* display the size of the file */
    printf("file length in bytes: %ld\n", filelength(handle));
    close(handle);
    return 0;
}
```

تابع ftell()

تابع ftell() "موقعیت‌سنج" فایل را مشخص می‌کند. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر است:

long ftell (FILE *fp)

در الگوی فوق، fp مشخص کننده فایلی است که "موقعیت سنج" آن باید مشخص گردد. اگر عمل تابع با موفقیت انجام نشود عدد ۱- توسط آن برگردانده خواهد شد. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
long int ;
if ((i = ftell (fp)) == -1L) printf ("\n error occured") ;
```

تابع isatty()

تابع isatty() مشخص می‌کند که آیا شماره فایل، به یکی از دستگاههای: ترمینال، صفحه‌نمایش، چاپگر و یا پورت موازی نسبت داده شده است یا خیر؛ اگر چنین نباشد مقدار صفر وگرنه مقداری غیر از صفر را برمی‌گرداند. الگوی این تابع در فایل io.h قرار داشته و به صورت زیر است:

int isatty (int fd)

fd شماره فایلی است که تابع باید بر روی آن عمل نماید. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
if (isatty (fd)) printf ("is a character device.") ;
    else
        printf ("is not a character device.") ;
```

تابع lock()

تابع lock() برای قفل کردن یک فایل مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل io.h قرار داشته و به صورت زیر است:

int lock (int fd, long offset, long length)

در الگوی فوق، fd شماره فایلی است که باید قفل گردد و قسمتی از فایل که باید قفل شود توسط دو پارامتر offset و length مشخص می‌شود. offset نقطهٔ شروع از ابتدای فایل و length طول قسمتی از فایل را که باید قفل شود تعیین

می‌کند. اگر عمل تابع با موفقیت انجام شود عدد صفر وگرنه عدد ۱- توسط تابع برگردانده می‌شود. به عنوان مثال، دستور ; lock (fd, 0, 128) موجب قفل شدن اولین ۱۲۸ بایت از فایلی با شماره fd می‌گردد.

تابع ()unlock

تابع ()unlock فایلی را که توسط تابع ()lock قفل شده است آزاد می‌کند و در شبکه‌های کامپیوتری مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل io.h قرار داشته و به صورت زیر است:

```
int onlock (int handle, long offset, long length)
```

در الگوی فوق، handle شماره فایلی است که این تابع باید بر روی آن عمل کند، offset محلی از فایل را مشخص می‌کند که عمل بازکردن قفل باید از آن نقطه شروع شود و length قسمتی از فایل را (به بایت) مشخص می‌کند (با شروع از محل offset) که باید قفل آن باز شود. اگر عمل تابع باموفقیت انجام شود مقدار صفر وگرنه مقدار ۱- توسط آن برگردانده می‌شود. به عنوان مثال، دستور ; unlock (fp, 0,100) موجب بازکردن قفل فایل از ابتدای آن و به تعداد ۱۰۰ بایت می‌شود.

تابع ()rename

تابع ()rename برای تغییر نام فایل مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر می‌باشد:

```
int rename (char  *old, char  *new)
```

در الگوی فوق، old نام فایلی است که باید به new تغییر نام یابد. اگر عمل تابع با موفقیت انجام شود عدد صفر وگرنه عددی غیر از صفر را برمی‌گرداند.

مثال ۲۴-۱۹

برنامه‌ای که نام دو فایل را به عنوان ورودی پذیرفته و عمل تغییر نام را انجام می‌دهد.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  clrscr();
  if(rename(argv[1], argv[2]) != 0)
      printf("\n rename error.") ;
  getch();
  return 0;
}
```

تابع setmode()

تابع setmode() برای تغییر حالت فایلی که اکنون باز است به کار می‌رود. الگوی این تابع در فایل io.h قرار داشته و به صورت زیر می‌باشد:

```
int  setmode (int fd, unsigned mode)
```

در الگوی فوق، fd شماره فایلی است که تابع باید بر روی آن عمل نماید و mode حالتی را مشخص می‌کند که فایل باید به آن حالت درآید. اگر mode برابر با O_BINARY باشد فایل باینری خواهد بود و اگر برابر با O_TEXT باشد فایل به صورت متن(text) خواهد بود. به عنوان مثال، دستور ; (setmode (fd, O_TEXT موجب می‌شود تا فایلی با شماره fd به صورت متن درآید.

تابع sprintf()

تابع sprintf() همانند تابع printf() عمل می‌کند. با این تفاوت که خروجی آن به جای ظاهر شدن در صفحه‌نمایش در یک آرایه قرار خواهد گرفت. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر می‌باشد:

```
int sprintf (char  *buf, char  *format, arg)
```

در الگوی فوق، format و arg همانند تابع printf() می‌باشند و پارامتر buf آرایه‌ای است که خروجی تابع به آن منتقل می‌شوند. به عنوان مثال، دستورات زیر موجب انتقال مقادیر 3، 2، و one به رشته str می‌شوند.

```
char str (80) ;
sprintf (str, "%s , %d %c" , "one", 2 , '3');
printf (str) ;
```

تابع sscanf()

تابع sscanf() همانند تابع scanf() عمل می‌کند، با این تفاوت که در تابع scanf() اطلاعات از صفحه‌کلید خوانده می‌شوند ولی در تابع sscanf() اطلاعات از آرایه خوانده می‌شوند. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر است:

```
int sscanf (char  *buf, char  *format, arg)
```

در الگوی فوق، پارامترهای arg و format همانند تابع scanf() عمل می‌کنند و پارامتر buf حاوی اطلاعاتی است که باید توسط تابع خوانده شوند.

مثال ۲۵-۱۹

برنامه‌ای که چگونگی عملکرد تابع sscanf() را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char str[80] ;
  int i ;
  sscanf("hello 1 2 3 4 5", "%s %d", str, &i);
  printf("\n %s %d", str, i) ;
  getch();
  return 0;
}
```

تابع tmpfile()

تابع tmpfile() یک فایل موقت به‌صورت باینری و در حالت ورودی-خروجی باز می‌کند. فایل را طوری ایجاد می‌کند که با هیچکدام از فایلهای موجود روی دیسک همنام نباشد. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر می‌باشد:

```
FILE  *tmpfile (void)
```

اگر عمل تابع با موفقیت انجام نشود، نتیجهٔ آن NULL خواهد بود. فایل موقت، پس از بسته‌شدن و یا خاتمهٔ اجرای برنامه، به طور اتوماتیک حذف خواهد شد. به عنوان مثال، مجموعه دستورات زیر را در نظر بگیرید:

```
FILE *temp ;
if(!(temp == tmpfile()))
{
    printf("\n cannot open temp file.");
    exit(1) ;
}
```

تابع tmpnam()

تابع tmpnam() برای تولید نام فایلی که مشابه آن در درایو جاری وجود نداشته باشد به کار می‌رود. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر است :

```
char  *tmpnam (char * name)
```

در الگوی فوق، name آرایه‌ای است که نام تولید شده در آن قرار خواهد گرفت. تابع tmpnam() می‌تواند حداکثر به تعداد TMP_MAX (ماکرویی در فایل stdio.h) بار فراخوانی شود و در هر بار فراخوانی نام فایل جدیدی را تولید خواهد کرد. اگر عمل تابع با موفقیت انجام نشود نتیجهٔ آن NULL خواهد بود.

مثال ۲۶-۱۹

برنامه‌ای که ۳ فایل منحصر بفرد را تولید می‌کند.

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
int main()
{
   char name[40] ;
   int i ;
   clrscr();
   for(i = 0; i < 3; i++) {
       tmpnam(name) ;
       printf("\n %s", name) ;
   }
   getch();
   return 0;
}
```

تابع ()ungetc

تابع ()ungetc کاراکتری را به یک فایل برمی‌گرداند. الگوی این تابع در فایل stdio.h قرار داشته و به صورت زیر است:

int ungetc (int ch, FILE *fp)

در الگوی فوق، fp به فایلی اشاره می‌کند که تابع باید بر روی آن عمل نماید و ch مقداری است که ۸ بیت کم‌ارزش آن باید به فایل برگردانده شود. کاراکتر EOF قابل برگشت به فایل نیست. اگر عمل تابع با موفقیت انجام شود نتیجه حاصل از تابع برابر با ch وگرنه برابر با EOF خواهد بود.

مثال ۲۷-۱۹

برنامه‌ای که عمل تابع ()ungetc را نشان می‌دهد.

```
#include <stdio.h>
#include <ctype.h>
int main( void )
{
   int i=0;
   char ch;
   puts("Input an integer followed by a char:");
   /* read chars until non digit or EOF */
   while((ch = getchar()) != EOF && isdigit(ch))
      i = 10 * i + ch - 48; /* convert ASCII into int value */
   /* if non digit char was read, push it back into input buffer */
   if (ch != EOF)
      ungetc(ch, stdin);
   printf("i = %d, next char in buffer = %c\n", i, getchar());
   return 0;
}
```

توابع متفرقه

بعضی از توابع در C وجود دارند که در هیچکدام از دسته‌بندیهایی که تاکنون انجام شده‌اند نمی‌گنجد،به همین دلیل تحت نام **توابع متفرقه** مورد بررسی قرار می‌گیرند. الگوی بسیاری از این توابع در فایل stdlib.h قرار دارد. در این فایل دو نوع به نامهای div_t و idiv_t تعریف شده‌اند که معادل نوعهایی هستند که توسط دو تابع ()div و ()idiv برگردانده می‌شوند.

توابع ()atof و ()atol

تابع ()atof یک رشتهٔ عددی را به یک عدد از نوع float و تابع ()atol یک رشته عددی را به یک مقدار عددی صحیح بزرگ تبدیل می‌کند. اگر عمل این توابع با موفقیت انجام نشود عدد صفر را برمی‌گردانند. الگوی این توابع در فایل stdlib.h قرار دارد. به عنوان مثال دستورات زیر را در نظر بگیرید:

```
char num1[80], num2[80] ;
float f;
long int p;
gets(num1) ;
gets(num2) ;
f =  atof(num1);
p =  atol(num2)) ;
```

تابع ()bsearch

تابع ()bsearch برای انجام جستجوی دودویی در یک آرایه به کار می‌رود. الگوی این تابع در فایل stdlib.h قرار داشته و به صورت زیر می‌باشد:

```
void *bsearch (const void *key, const void *base, size_t num, size_t size, int
(*compare)(const void * ,const void *))
```

در الگوی فوق، base به آرایهٔ مرتبی اشاره می‌کند که عنصر key باید در آن جستجو شود. size_t در فایل stdlib.h به صورت unsigned int تعریف شده است و num تعداد عناصر آرایه را مشخص می‌کند و اندازه هر عنصر آرایه توسط پارامتر size مشخص می‌شود. تابعی که پارامتر compare به آن اشاره می‌کند، برای مقایسه دو عنصر مورد استفاده قرار می‌گیرد که شکل کلی آن باید به صورت زیر باشد :

```
func_name (void *arg1, void *arg2)
```

این تابع باید مقادیر زیر را برگرداند:

۱. اگر arg1 < arg2 مقداری کمتر از صفر.
۲. اگر arg1 = arg2 مقدار صفر.
۳. اگر arg1 > arg2 مقداری بزرگتر از صفر.

آرایه‌ای که base به آن اشاره می‌کند باید به طور صعودی مرتب شده باشد. اگر عمل تابع با موفقیت انجام شود، به عنصر مورد نظر اشاره خواهد کرد وگرنه نتیجهٔ آن NULL خواهد بود.

مثال ۲۸-۱۹

برنامه‌ای که مقداری را در آرایه‌ای جستجو می‌کند.

```
#include <stdlib.h>
#include <stdio.h>
typedef int (*fptr)(const void*, const void*);
#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))
int numarray[] = {123, 145, 512, 627, 800, 933};
int numeric (const int *p1, const int *p2)
{
   return(*p1 - *p2);
}
//*****************************
#pragma argsused
int lookup(int key)
{
   int  *itemptr;
   /* The cast of (int(*)(const void *,const void*))
      is needed to avoid a type mismatch error at
      compile time */
      itemptr = (int *) bsearch (&key, numarray, NELEMS(numarray),
        sizeof(int), (fptr)numeric);
       return (itemptr != NULL);
}
//*************************
int main(void) {
   if (lookup(512))
      printf("512 is in the table.\n");
   else
      printf("512 isn't in the table.\n");
   return 0;
}
```

تابع div() و idiv()

تابع div() خارج قسمت و باقیمانده تقسیم دو عدد صحیح و idiv() باقیمانده و خارج قسمت دو عدد صحیح بزرگ را محاسبه می‌کند. الگوی این توابع در فایل stdlib.h قرار داشته و به صورت زیر است:

```
div_t div (int number, int denom)
idiv_t idiv (long int number, long int denom)
```

حاصل number/denom محاسبه شده، خارج قسمت در qoute و باقیمانده در rem قرار می‌گیرد. در الگوی فوق، div_t و idiv_t ساختمانهایی هستند که در فایل stdlib.h تعریف شده، به صورت زیر هستند.

```
typedef struct {
  long int quot;    /* quotient */
  long int rem;     /* remainder */
} div_t;
//**********************
typedef struct {
  long int quot;    /* quotient */
  long int rem;     /* remainder */
} ldiv_t;
```

مثال ۲۹-۱۹

برنامه‌ای که عملکرد تابع ()div را نشان می‌دهد.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
   div_t x;
   clrscr();
   x = div(10, 3);
   printf("10 div 3 = %d remainder %d\n" ,x.quot, x.rem);
   return 0;
}
```

توابع ()ecvt و ()fcvt

این توابع، عددی را به رشته‌ای با طول معین تبدیل می‌کنند. الگوی آنها در فایل stdlib.h قرار دارد و به صورت زیر است:

```
char *ecvt (double value, int ndigit, int *dec, int *sign)
char *fcvt (double value, int ndigit, int *dec, int *sign)
```

در الگوی فوق، value مقداری عددی است که باید به رشته‌ای به طول ndigit تبدیل شود. پس از عمل تابع، dec محل نقطهٔ اعشار را مشخص می‌کند. اگر نقطهٔ اعشار در سمت چپ عدد باشد، dec منفی خواهد بود و همچنین اگر متغیری که sign به آن اشاره می‌کند، منفی باشد، عدد منفی خواهد بود. تفاوت این دو تابع این‌است‌که تابع ()fcvt نتیجه عمل را گرد می‌کند. به عنوان مثال دستورات زیر را در نظر بگیرید:

```
int decpnt, sign ;
char  *out ;
out = ecvt (10.12, 5, &decpnt, &sign)
```

تابع itoa()

تابع itoa() یک مقدار عددی صحیح را به رشته تبدیل می‌کند. الگوی این تابع در فایل stdlib.h قرار دارد و به صورت زیر است:

```
char  *itoa (int num, char  *str, int radix)
```

در الگوی فوق، num عدد صحیح است که پس از تبدیل به رشته، در str قرار خواهد گرفت. مبنایی که رشتهٔ عددی باید به آن تبدیل شود توسط radix مشخص می‌گردد که در بازهٔ ۲ تا ۳۶است.

مثال ۳۰-۱۹

برنامه‌ای که عدد ۱۴۲۳ را به مبنای ۱۶ تبدیل می‌کند.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void main() {
  char p[17] ;
  clrscr();
  itoa(1423, p, 16) ;
  printf(p) ;
  getch();
}
```

تابع labs()

این تابع قدرمطلق اعداد صحیح بزرگ را محاسبه می‌کند. الگوی آن در فایل stlib.h قرار دارد و به‌صورت زیر است:

```
long  labs (long num)
```

در الگوی فوق، num عددی است که باید قدر مطلق آن محاسبه شود.

توابع lfind() و lsearch()

توابع lfind() و lsearch() برای جستجوی خطی در یک آرایه مورد استفاده قرار می‌گیرند. الگوی این توابع در فایل stdlib.h قرار دارد و به صورت زیر است:

```
void *lfind (const void *key, const void *base, size_t *num, size_t size,
int (*compare) (const void * , const void *))
void lsearch (const void *key, const void *base, size_t *num, size_t size,
int (compare) (const void * , const void *))
```

پارامترهای توابع فوق همانند تابع bsearch() می‌باشند ولی آرایه‌ای که base به آن اشاره می‌کند لازم نیست که مرتب باشد. تفاوت lsearch() و lfind() در این است که، اگر کلید مورد جستجو در آرایه وجود نداشته باشد، تابع lsearch() آن را به انتهای آرایه اضافه می‌کند ولی lfind() این کار را انجام نمی‌دهد.

مثال ۳۱-۱۹

برنامه‌ای که عددی را در آرایه‌ای جستجو می‌کند.

```
#include <stdio.h>
#include <stdlib.h>
int compare(int *x, int *y) {
   return( *x - *y );
}
//*******************
int main(void)
{
   int array[5] = {35, 87, 46, 99, 12};
   size_t nelem = 5;
   int key;
   int *result;
   key = 99;
   result = (int *) lfind(&key, array, &nelem,
      sizeof(int), (int(*)(const void *,const void *))compare);
   if (result)
      printf("Number %d found\n", key);
   else
      printf("Number %d not found\n", key);
   return 0;
}
```

توابع setjmp() و longjmp()

تابع setjmp() محتویات پشتهٔ سیستم را در یک بافر نگهداری می‌کند تا بعداً توسط تابع longjmp() استفاده گردد. الگوی تابع setjmp() در فایل stdlib.h و الگوی تابع longjmp() در فایل setjmp.h قرار دارد و به صورت زیر است:

```
void longjmp (jmp_buf envbuf, int val)
int setjmp (jmp_buf envbuf)
```

تابع longjmp() موجب می‌شود تا اجرای برنامه از نقطه آخرین فراخوانی تابع setjmp() ادامه پیدا کند. این دو تابع امکان انتقال کنترل از تابعی به تابع دیگر را فراهم می‌کنند. در الگوهای فوق، jmp_buf نوعی است که در فایل setjmp.h تعریف شده است. envbuf بافری است که در تابع setjmp()، محتویات پشته به آن منتقل می‌شوند و در تابع longjmp() محتویات آن به پشته منتقل شده سپس تابع عمل می‌کند. پارامتر val مقدار برگردانده شده توسط تابع setjmp() را نگهداری می‌کند و بیانگر محلی است که تابع longjmp() از آنجا عمل کرده است. مقدار آن نمی‌تواند صفر باشد.

مثال ۳۲-۱۹

برنامه‌ای که چگونگی عملکرد توابع setjmp() و longjmp() را نشان می‌دهد.

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
#include <conio.h>
void subroutine(jmp_buf);
int main(void) {
   int value;
   jmp_buf jumper;
   value = setjmp(jumper);
   if (value != 0) {
      printf("Longjmp with value %d\n", value);
      exit(value);
   }
   printf("About to call subroutine ... \n");
   subroutine(jumper);
   return 0;
}
//**********************
void subroutine(jmp_buf jumper)
{
   longjmp(jumper,1);
}
```

تابع ltoa()

تابع ltoa() یک عدد صحیح بزرگ را به رشته تبدیل می‌کند. الگوی این تابع در فایل stdlib.h قرار داشته و به صورت زیر است:

char *ltoa (long num, char *str, int radix)

در این الگو، num عددی‌است‌که باید به رشته تبدیل شود و در str قرار گیرد. مبنای رشته عددی توسط radix مشخص می‌شود که می‌تواند در بازه ۲ تا ۳۶ باشد. طول رشته‌ای که نتیجهٔ عمل تابع در آن قرار می‌گیرد، باید ۳۳ بایت باشد.

مثال ۱۹-۳۳

برنامه‌ای که چگونگی عملکرد تابع ltoa() را نشان می‌دهد.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
   char string[25];
   long value = 123456789L;
   clrscr();
   ltoa(value, string, 10);
   printf("number = %ld  string = %s\n", value, string);
   return 0;
}
```

تابع ()qsort

تابع ()qsort برای مرتب کردن یک آرایه به طور صعودی و به روش quicksort استفاده می‌شود. الگوی این تابع در فایل stdlib.h قرار دارد و به صورت زیر است :

```
void qsort (void *base, int num, int size, int (*compare) ( ))
```

در الگوی فوق، base به آرایه‌ای اشاره می‌کند که باید مرتب شود. تعداد عناصر آن برابر با num بوده و اندازه هر عنصر آن با size مشخص می‌شود. compare به تابعی اشاره می‌کند که باید عمل مقایسه را انجام دهد و شکل کلی آن به صورت زیر است :

```
func_name (void *arg1, void *arg2)
```

مقادیری که توسط تابع مقایسه کننده باید برگردانده شود عبارتنداز :

۱. اگر arg1 < arg2 یک عدد منفی
۲. اگر arg1 = arg2 عدد صفر
۳. اگر arg1 > arg2 یک عدد مثبت

مثال ۳۴-۱۹

برنامه‌ای که آرایه‌ای از رشته‌ها را مرتب می‌کند.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
int sort_function( const void *a, const void *b);
char list[5][4] = { "cat", "car", "cab", "cap", "can" };
int main(void)
{
   int x;
   qsort((void *)list, 5, sizeof(list[0]), sort_function);
   for (x = 0; x < 5; x++)
      printf("%s\n", list[x]);
   return 0;
}
//**********************
int sort_function( const void *a, const void *b)
{
   return( strcmp((char *)a,(char *)b) );
}
```

توابع random() و randomize()

تابع random() موجب می‌شود تا عدد تصادفی تولید شده، در یک بازه خاص باشد و تابع randomize() عددی را که اعداد تصادفی از آن تولید می‌شوند مشخص می‌کند. الگوی این توابع در stdlib.h قرار داشته و به صورت زیر است:

```
int random (int num)
void randomize (void)
```

عدد تصادفی تولید شده باید بین صفر تا num در این الگو باشد.

مثال ۱۹-۳۵

برنامه‌ای که ۱۰ عدد تصادفی بین صفر تا ۲۵ تولید می‌کند. چون تابع randomize() با استفاده از تابع time()، مولد اعداد تصادفی را ایجاد می‌کند، فایل time.h باید به برنامه ضمیمه گردد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
   int i ;
   clrscr();
   randomize() ;
   for(i = 0; i < 10; i++)
       printf("%3d", random(25)) ;
   getch();
   return 0;
}
```

تابع srand()

تابع srand() برای تعیین عددی(seed) که اعداد تصادفی از آن تولید می‌شوند به کار می‌رود. الگوی این تابع در فایل stdlib.h قرار دارد و به صورت زیر است:

```
void srand (unsigned seed)
```

در الگوی فوق، seed عددی است که اعداد تصادفی باید از آن تولید شوند.

مثال ۱۹-۳۶

برنامه‌ای که چگونگی عملکرد تابع srand() را نشان می‌دهد.

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main(void)
{
```

```
    int i;
    time_t t;
    clrscr();
    srand((unsigned) time(&t));
    printf("Ten random numbers from 0 to 99\n\n");
    for(i = 0; i < 10; i++)
        printf("%d\n", rand() % 100);
    return 0;
}
```

توابع ()strtod و ()strtol و ()strtoul

تابع ()strtod رشتهٔ عددی را به یک مقدار عددی از نوع double، تابع ()strtol رشتهٔ عددی را به یک مقدار عددی صحیح بزرگ و تابع ()strtoul رشتهٔ عددی را به یک مقدار صحیح بزرگِ بدون علامت تبدیل می‌کند. الگوی این توابع در فایل stdlib.h قرار دارد و به صورت زیر است:

```
double strtod (char *start, char **end)
long int strtol (char *start, char  **end, int radix)
unsigned long strtoul(const char *start, char **end, int radix)
```

در الگوهای فوق، start به رشتهٔ عددی اشاره می‌کند. نحوهٔ عمل توابع به این صورت است که: از ابتدای رشته شروع کرده، blankهای ابتدای آن را حذف می‌کنند و کاراکتر به کاراکتر در طول رشته به پیش می‌روند. هرگاه به کاراکتری غیرمجاز (blank ، ; ، کاما و ...) رسیدند، عمل تابع خاتمه پیدا می‌کند و آدرس اولین کاراکتر غیرمجاز در end قرار می‌گیرد. به عنوان مثال، اگر رشتهٔ 100.00ABX به آنها داده شود، تابع ()strtod عدد 100.00 و تابع ()strtol مقدار 100 را برمی‌گرداند و end به کاراکتر A اشاره خواهد کرد. radix در توابع ()strtol و ()strtoul مبنای عدد را تعیین می‌نماید که اگر صفر باشد، مبنای عدد از خودِ عدد نتیجه می‌شود وگرنه می‌تواند در بازه ۲ تا ۳۶ باشد.

مثال ۳۷-۱۹

برنامه‌ای که کاربرد توابع ()strtod، ()strtol و ()strtoul را نشان می‌دهد.

```
#include <ctype.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char *string = "87654321", *endptr;
  long lnumber;
  char *end;
  char *start = "100.00test 200.00 exist";
  clrscr();
```

```
    end = start ;
    while(*start) {
      printf(" %f", strtod(start, &end)) ;
      printf("\t remainder:%s\n", end) ;
      start = end ;  /* move past the non_digit */
      while(!isdigit(*start) && *start)
        start ++ ;
    }
    lnumber = strtol(string, &endptr, 10);
    printf("string = %s  long = %ld\n", string, lnumber);
    return 0;
}
```

تابع swab()

این تابع قسمتی از رشته را در رشتهٔ دیگر کپی می‌کند. الگوی آن در فایل stdlib.h قرار دارد و به‌صورت زیر است:

void swab (char *source, char *dest, int num)

در الگوی فوق، source رشته‌ای است که به تعداد num بایت از ابتدای آن، در رشتهٔ dest کپی می‌شود. به عنوان مثال، مجموعه دستورات زیر، رشتهٔ "in the name" را در dest کپی می‌کند.

```
char dest[20] ;
swab ("in the name of god") ;
printf ("%s", dest, 11);
```

تابع system()

تابع system() برای اجرای فرمانهای سیستم عامل مورد استفاده قرار می‌گیرد. الگوی این تابع در فایل stdlib.h قرار دارد و به صورت زیر است:

Int sysytem (char *str)

در الگوی فوق، str یکی از فرمانهای سیستم عامل است.

مثال ۳۸-۱۹

برنامه‌ای که دستور dir را اجرا می‌کند.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
   printf("spawn command interpreter and run a DOS command\n");
   system("dir");
   return 0;
}
```

توابع va_start()، va_end() و va_arg()

این توابع با هم مورد استفاده قرار گرفته، امکان فراخوانی توابعی با تعداد متغیری از آرگومانها را فراهم می‌کنند. الگوی آنها در فایل stdarg.h قرار دارد و به صورت زیر می‌باشد:

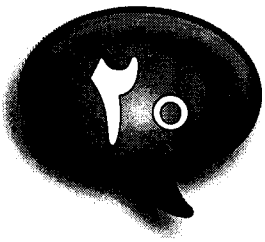
```
void va_start(va_list argptr, last_parm)
void va_end(va_list argptr)
void va_arg(va_list argptr, type)
```

شرط ایجاد تابعی با تعداد متغیری از آرگومانها این است که حداقل یک پارامتر داشته باشد. آخرین پارامتر تابع با last_parm مشخص می‌شود، نوع va_list در فایل stdarg.h تعریف شده است. پارامتر argptr به لیست آرگومانها اشاره می‌کند که قبل از استفاده باید توسط تابع va_start() ارزش‌دهی گردد. تابع va_arg() برای جداکردن آرگومانهایی با نوع type از argprt استفاده می‌شود و فراخوانی va_end() جهت حصول اطمینان از بازیابی اطلاعات از پشته، انجام می‌گیرد.

مثال ۳۹-۱۹

برنامه‌ای که عناصر آرایه‌ای را که آخرین عنصر آن صفر است با هم جمع می‌کند.

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>
void sum(char *msg, ...)
{
   int arg, total = 0;
   va_list ap;
   va_start(ap, msg);
   while ((arg = va_arg(ap,int)) != 0) {
      total += arg;
   }
   printf(msg, total);
   va_end(ap);
}
//**********************
int main(void) {
   clrscr();
   sum("The total of 1+2+3+4 is %d\n", 1,2,3,4,0);
   return 0;
}
```



مدیریت منوها

یکی از وظایف طراحان نرم‌افزار این‌است‌که سیستمها را طوری طراحی کنند که بهره‌برداری از آنها آسان باشد. داشتن منوهای مناسب و سپردن حداقل کار به کاربر، از جمله مسایلی است که به این امر کمک می‌کند. در برنامه‌هایی که تاکنون نوشته شده‌اند منوهای ساده‌ای طراحی شده‌اند که در همهٔ زبانهای برنامه‌سازی مرسوم می‌باشند. در این فصل چگونگی ایجاد دو نوع منو به نامهای popup و pulldown مورد بررسی قرار می‌گیرند. لازمه ایجاد این دو نوع منو، استفاده از وقفه‌های بایوس (BIOS) و یا دسترسی مستقیم به حافظه نمایش است.

تفاوت منوهای معمولی با منوهای popup و pulldown در این است که منوهای معمولی، صفحه‌نمایش را پاک می‌کنند و سپس در صفحه‌نمایش ظاهر می‌گردند و پس از عمل انتخاب نیز مجدداً صفحه‌نمایش پاک می‌شود. ولی دو منوی دیگر، بر روی آنچه که اکنون در صفحه‌نمایش وجود دارند نوشته می‌شوند و پس از عمل انتخاب، صفحه‌نمایش به حالت قبلی برمی‌گردد. عمل انتخاب در این منوها به دو روش امکان‌پذیر است :

۱. وارد نمودن یک کلید خاص (معمولاً اولین حرف از گزینه)
۲. انتقال مکان‌نما به یک گزینه و سپس فشاردادن کلید enter

تفاوت منوی popup با pulldown در این است که در آنِ واحد فقط یک منوی popup در صفحه نمایش قابل ظاهر شدن است، و در مواقعی مورد استفاده قرار می‌گیرد که گزینه‌های یک منو، خودشان شامل گزینه‌های دیگری نباشند. ولی چند منوی pulldown می‌توانند در آنِ واحد در صفحه‌نمایش وجود داشته باشند و در مواقعی مورد استفاده قرار می‌گیرند که هر یک از گزینه‌های منو دارای چند گزینه دیگر نیز باشند.

ذخیره و بازیابی قسمتی از صفحه‌نمایش

قبل از ظاهر شدن منوهای popup و pulldown در صفحه‌نمایش، باید محتویات آن قسمت از صفحه‌نمایش که این منوها در آنجا ظاهر می‌شوند ذخیره شود تا پس از عمل انتخاب، محتویات قبلی صفحه‌نمایش ظاهر گردند. لذا باید توابعی برای این منظور نوشته شوند. لازمهٔ نوشتن چنین توابعی استفاده از تابعی جهت انتقال مکان‌نما به یک محل مناسب از صفحه‌نمایش است.

مثال ۱-۲۰

تابعی برای انتقال مکان‌نما در صفحه‌نمایش .

```
#include <dos.h>
void goto_xy(int x, int y)
{
    union  REGS r ;
    r.h.ah = 2 ;
```

```
    r.h.dl = y ; /* column */
    r.h.dh = x ; /* row */
    r.h.bh = 0 ;
    int86(0x10, &r, &r) ;
}
```

مثال ۲-۲۰

تابعی برای ذخیره کردن بخشی از صفحه‌نمایش در یک بافر.

```
void save_video(int startx, int endx, int starty,
       int endy , unsigned int *buf_ptr)
{
     register int i , j ;
     union REGS r ;
     for(i = starty ; i < endy; i++)
       for(j = startx; j < endx; j++) {
          goto_xy(j, i) ;
          r.h.ah = 8 ;
          r.h.bh = 0 ;
          *buf_ptr++ = int86(0x10, &r, &r) ;
          putchar(' ') ;
       }
}
```

مثال ۳-۲۰

تابعی برای بازیابی بخشی از صفحه‌نمایش که در بافری قرار دارد.

```
void restore_video(int startx, int endx, int starty,
   int endy, unsigned char *buf_ptr)  {
     register int i , j ;
     union REGS r ;
     for(i = starty ; i < endy; i++)
     for(j = startx; j < endx; j++) {
        goto_xy(j, i) ;
        r.h.ah = 9 ;
        r.h.bh = 0 ;
        r.x.cx = 1 ;
        r.h.al = *buf_ptr++ ;
        r.h.bl = *buf_ptr++ ;
        int86(0x10, &r, &r) ;
     }
}
```

ایجاد منوی popup

برای نوشتن تابعی که بتواند منوی popup را ایجاد نماید باید اطلاعاتی را به آن تابع منتقل کرد:

۱. گزینه‌های منو. چون گزینه‌ها به صورت رشته‌ای می‌باشند، بهترین راه برای انتقال لیست گزینه‌ها، نوشتن آنها در یک آرایه دوبعدی و استفاده از یک اشاره‌گر برای دسترسی به آن است.

۲. همانطور که قبلاً گفته شد، برای انتخاب هر یک از گزینه‌های منو به دو طریق می‌توان عمل کرد (فشاردادن یک کلید خاص و یا انتقال مکان‌نما به آن و سپس فشاردادن کلید enter). برای اینکه تابع تولید منو بداند که چه کلیدهایی می‌توانند وارد شوند و هر کلید چه کاری باید انجام دهد، اسامی این کلیدها باید به تابع منتقل شوند. بهترین راه برای انجام این عمل، استفاده از یک رشته جهت ذخیره کردن این کلیدها و تعریف یک اشاره‌گر جهت دسترسی به آنها است.

۳. تعداد گزینه‌های منو

۴. محلی که گزینه‌های منو باید در آنجا نوشته شوند.

۵. رسم یا عدم رسم حاشیه برای منو. برای این منظور می‌توان از یک متغیر استفاده کرد، چنانچه مقدار این متغیر یک باشد، حاشیه رسم می‌شود وگرنه رسم نخواهد شد.

تابعی که در این فصل برای ایجاد منوی popup نوشته می‌شود ()popup نام دارد که پارامترهای آن به صورت زیر می‌باشند:

```
int popup(char *menu[] , char *keys , int count ,int x, int y, int border)
```

در الگوی کلی فوق، اشاره‌گر menu به رشته حاوی منو اشاره می‌کند، اشاره‌گر keys به رشته‌ای که حاوی کلیدهای معتبر جهت انتخاب گزینه‌های منو است اشاره می‌نماید، count تعداد گزینه‌های منو، x و y محل ظاهر شدن منو و border رسم یا عدم رسم حاشیه را مشخص می‌کند (اگر صفر باشد حاشیه رسم نخواهد شد). تابع ()popup اعمال زیر را انجام می‌دهد:

۱. ذخیره کردن قسمتی از صفحه‌نمایش که منو در آنجا ظاهر می‌شود، در یک بافر.
۲. رسم حاشیه برای منو در صورت لزوم.
۳. نمایش منو در صفحه.
۴. اخذ انتخاب کاربر.
۵. بازیابی قسمتی از صفحه‌نمایش که در بافر ذخیره شده است.

چگونگی ذخیره و بازیابی قسمتی از صفحه‌نمایش توسط دو تابع ()save_video و ()restor_video انجام می‌شوند که لیست آنها قبلاً مشاهده شده است.

برای ظاهر نمودن گزینه‌های منوها در صفحه‌نمایش، تابع ()popup توسط یک اشاره‌گر، به آرایه‌ای که حاوی این گزینه‌ها است اشاره می‌نماید. این اشاره‌گر در واقع، آرایه‌ای از اشاره‌گرها است که هر عنصر آن به یک گزینه منو اشاره می‌نماید.

مثال ۴-۲۰

تابعی برای نمایش گزینه‌های منو.

```
void display_menu(char *menu[], int x, int y, int count)
{
    register int i ;
    for(i = 0 ; i < count ; i++ , x++) {
       goto_xy(x, y) ;
       printf(menu[i]) ;
    }
}
```

مثال ۵-۲۰

تابع draw_border() برای رسم حاشیه منو. برای رسم حاشیه منو کافی است نقطهٔ بالای سمت چپ و نقطهٔ پایین سمت راست حاشیه مشخص باشد. **لیست این تابع را در مثال ۱۰-۲۰ ببینید.**

مثال ۶-۲۰

تابع get_resp() برای دریافت انتخاب کاربر. این تابع طوری طراحی شده است که با کلیدهای مکان‌نما و SPACE می‌توان از گزینه‌ای به گزینه دیگر رفت و سپس با کلید enter آن را انتخاب نمود. **لیست این تابع را در مثال ۱۰-۲۰ ببینید.**

پس از اجرای تابع get_resp() ، اولین گزینه منو رنگی خواهد بود. این عمل توسط دو ماکروی REV_VID و NORM_VID انجام می‌شود. کلید ESC برای خروج از منو مورد استفاده قرار می‌گیرد. تابع در یک حلقه تکرار جهت اخذ انتخاب کاربر باقی می‌ماند. این عمل توسط تابع کتابخانه‌ای bioskey() انجام می‌شود. علت استفاده از تابع bioskey() به جای توابعی مثل getchar() این است که برنامه باید هر ۱۶ بیت تولید شده توسط کلید را بخواند. اگر یک کلید معمولی فشار داده شود، ۸ بیت کم‌ارزش حاوی کد این کلید و ۸ بیت باارزش، صفر خواهد بود. اگر کلید فشار داده شده یکی از کلیدهای مکان‌نما و یا کلیدهای مشابه با آنها باشد، ۸ بیت کم‌ارزش، صفر و ۸ بیت باارزش کد آن کلید خواهد بود. کد کلید UP ARROW برابر با ۷۲ و کد کلید DOWN ARROW برابر با ۸۰ است، توابعی مثل getchar() فقط کد کلیدهای معمولی را می‌خوانند.

وقتی یکی از کلیدهای مکان‌نما وارد شد، گزینه‌ای از منو که اکنون رنگی است، از حالت رنگی خارج شده، گزینه بعدی رنگی می‌شود (مکان‌نما به آن منتقل می‌گردد). تابع is_in() توسط تابع get_resp() مورد استفاده قرار می‌گیرد. این تابع، محل کلید فشار داده شده را در رشتهٔ حاوی کلیدهای معتبر، مشخص می‌کند. **لیست این تابع را در مثال ۱۰-۲۰ ببینید.**

مثال ۷-۲۰

تابعی برای نوشتن گزینه‌ای از منو (رشته) در نقطه‌ای از صفحه‌نمایش.

```
void write_video(int x, int y, char *p, int attrib)
 {
      register int i,j ;
      union REGS r ;
      for(i = y ; *p ; i++)   {
         goto_xy(x,i) ;
         r.h.ah=9 ;
         r.h.bh=0 ;
         r.x.cx=1 ;
         r.h.al=*p++ ;
         r.h.bl=attrib ;
         int86(0x10,&r,&r) ;
       }
 }
```

مثال ۸-۲۰

تابع is_in() که کاراکتری را در رشته جستجو می‌کند. در صورت وجود کاراکتر در رشته محل وجود آن، وگرنه صفر را بر می‌گرداند.

```
int is_in(char *s , char c)
 {
    register int i ;
    for(i = 0 ; *s ; i++)
          if(*s++ == c)
             return i + 1 ;
    return 0 ;
 }
```

مثال ۹-۲۰

تابع popup() اسکلت اصلی برنامه منوی popup است. **لیست این تابع را در مثال ۱۰-۲۰ مشاهده کنید.**

توضیح

همانطور که در تابع popup() مشاهده می‌گردد. چنانچه گزینه‌های منو بخواهند از صفحه‌نمایش خارج شوند (مختصات نقطهٔ (x,y) مناسب انتخاب نگردد) عدد ۲- به عنوان نتیجه عمل برگردانده می‌شود. توجه دارید که در صورت فشاردادن کلید ESC توسط کاربر، تابع get_resp() عدد ۱- را برمی‌گرداند. چنانچه این مقدار توسط تابع popup() به برنامه اصلی برگردانده شود، به منزلهٔ صرفنظر از منو است. تابع popup() برای ذخیره کردن صفحه‌نمایش در بافر، حافظه لازم را به طور پویا از سیستم اخذ می‌کند.

مثال ۱۰-۲۰

برنامه تولید منوی popup با استفاده از وقفه‌های بایوس. در ادامه این فصل، برنامه تولید منوی popup را که بدون استفاده از این وقفه‌ها نوشته می‌شود (دسترسی مستقیم به حافظه نمایش) مشاهده خواهیم کرد.

```
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#define  BORDER      1
#define  ESC         27
#define  REV_VID     0x70
#define  NORM_VID    7
int popup(char *menu[] , char *keys , int count ,int x, int y, int border);
void save_video(int startx, int endx, int starty,
        int endy , unsigned int *buf_ptr);
void draw_border(int startx, int starty, int endx, int endy);
void display_menu(char *menu[], int x, int y, int count);
int get_resp(int x, int y, int count, char *menu[] , char *keys);
void restore_video(int startx, int endx, int starty,
        int endy, unsigned char *buf_ptr);
int is_in(char *s , char c);
void write_video(int x, int y, char *p, int attrib);
void goto_xy(int , int) , cls() ;
char *fruit[] = { "apple      ", "orange     ", "pear       ",
                  "grape      ", "raspberry ", "strawberry" } ;
char *color [] = { "red",  "yellow",  "orange",  "green" } ;
char *apple_type[] = { "red delicious" , "jonthan", "winesap", "rome"};
int main()
 {
       int i ;
       cls() ;
       goto_xy(0, 0) ;
       popup(fruit," aoprts", 6, 1, 3, BORDER) ;
       popup(color , "ryog", 4, 5, 10, BORDER) ;
       popup(apple_type, "rjw", 4, 10, 18, BORDER) ;
       getch();
       return 0;
  }
/* displ a pop-up menu and return
  selection
  return -2 if menu cannot be constructed
  return -1 if user hits escape key
  otherwise the item number is returned
  starting with 0 as the first(top most) entry
```

```
 */
int popup(char *menu[] , char *keys , int count ,int x, int y, int border)
/* displ a pop-up menu and return selection
 return -2 if menu cannot be constructed
 return -1 if user hits escape key
 otherwise the item number is returned
 starting with 0 as the first(top most) entry
*/
 {
    register int i , len ;
    int endx , endy , choice  ;
    unsigned int *p ;
    if((x > 24) || (x < 0) || (y > 79) || (y < 0)) {
         printf("\n range error") ;
         return -2 ;
    }
    len = 0 ;
    for(i = 0 ; i < count ; i++)
         if(strlen(menu[i]) > len)
    len = strlen(menu[i]) ;
    endy = len+2+y ;
    endx = count + 1 + x ;
    if((endx + 1 > 24) || (endy + 1 > 79)) {
        printf("\n menu won't fit") ;
        return -2 ;
    }
    p = (unsigned int *) malloc((endx - x + 1) * (endy - y + 1)) ;
    if(!p)
          exit(1) ;
    save_video(x, endx + 1, y, endy + 1, p);
    if(border)
        draw_border(x, y, endx, endy) ;
     display_menu(menu , x + 1 , y + 1 , count) ;
     choice = get_resp(x + 1, y, count, menu , keys) ;
     restore_video(x, endx + 1 , y , endy + 1 ,(char *) p) ;
     free(p) ;
     return choice ;
 }
//*****************
void display_menu(char *menu[], int x, int y, int count)
 {
      register int i ;
      for(i = 0 ; i < count ; i++ , x++)  {
```

```
            goto_xy(x, y) ;
            printf(menu[i]) ;
        }
    }
//*******************
void draw_border(int startx, int starty, int endx, int endy)
    {
     register int i ;
     for(i = startx + 1 ; i < endx; i++)  {
         goto_xy(i, starty) ;
         putchar(179) ;
         goto_xy(i,endy) ;
         putchar(179) ;
     }
     for(i = starty + 1 ; i < endy; i++)  {
        goto_xy(startx, i) ;
        putchar(196) ;
        goto_xy(endx, i) ;
        putchar(196) ;
     }
     goto_xy(startx, starty) ;
     putchar(218) ;
     goto_xy(startx, endy) ;
     putchar(191) ;
     goto_xy(endx, starty) ;
     putchar(192) ;
     goto_xy(endx, endy) ;
     putchar(217) ;
 }
 //************************
int get_resp(int x, int y, int count, char *menu[] , char *keys)
  {
       union inkey {
           char ch[2] ;
          int i ;
          } c ;
     int arrow_choice = 0 , key_choice ;
     y++ ;
     goto_xy(x,y) ;
     write_video(x, y, menu[0], REV_VID) ;
     for(;;)  {
        while(!bioskey(1));
             c.i = bioskey(0) ;
```

```
        goto_xy(x + arrow_choice, y) ;
        write_video(x + arrow_choice, y, menu[arrow_choice], NORM_VID);
        if(c.ch[0]) { /* normal key */
            key_choice = is_in(keys, tolower(c.ch[0])) ;
            if(key_choice)
                return key_choice - 1 ;
             switch(c.ch[0])  {
                case '\r' : return arrow_choice ;
                case ' '  : arrow_choice++ ;  break ;
                case ESC  : return -1 ; /* cancel */
             }//end of switch
        }//end of if
        else {
            switch(c.ch[1]) {
              case 72 : arrow_choice -- ;  break ;
              case 80 : arrow_choice ++ ;  break ;
            }//end of switch
        } // end of else
        if(arrow_choice == count)
             arrow_choice = 0 ;
        if(arrow_choice < 0)
            arrow_choice = count - 1 ;
        goto_xy(x+arrow_choice,y) ;
         write_video(x+arrow_choice,y,menu[arrow_choice],REV_VID);
    }//end of for(;;)
 }
 //*************************
void write_video(int x, int y, char *p, int attrib)
 {
      register int i,j ;
      union REGS r ;
      for(i = y ; *p ; i++)   {
          goto_xy(x,i) ;
          r.h.ah=9 ;
          r.h.bh=0 ;
          r.x.cx=1 ;
          r.h.al=*p++ ;
          r.h.bl=attrib ;
          int86(0x10,&r,&r) ;
      }
 }
 //*************************
void save_video(int startx, int endx, int starty, int endy , unsigned int *buf_ptr)
```

```
{
      register int i , j ;
      union REGS r ;
      for(i = starty ; i < endy; i++)
        for(j = startx; j < endx; j++) {
             goto_xy(j, i) ;
             r.h.ah = 8 ;
             r.h.bh = 0 ;
            *buf_ptr++ = int86(0x10, &r, &r) ;
             putchar(' ') ;
        }
}
//*********************
void restore_video(int startx, int endx, int starty, int endy, unsigned char
*buf_ptr)
 {
      register int i , j ;
      union REGS r ;
      for(i = starty ; i < endy; i++)
        for(j = startx; j < endx; j++) {
             goto_xy(j, i) ;
             r.h.ah = 9 ;
             r.h.bh = 0 ;
             r.x.cx = 1 ;
             r.h.al = *buf_ptr++ ;
             r.h.bl = *buf_ptr++ ;
             int86(0x10, &r, &r) ;
         }
 }
 //***************************
void cls()
 {
     union REGS r ;
     r.h.ah = 6 ;
     r.h.al = 0 ;
     r.h.ch = 0 ;
     r.h.cl = 0 ;
     r.h.dh = 24 ;
     r.h.dl = 79 ;
     r.h.bh = 7 ;
     int86(0x10, &r, &r) ;
 }
 //*******************************
```

```
void goto_xy(int x, int y)
 {
     union  REGS r ;
     r.h.ah = 2 ;
     r.h.dl = y ; /* column */
     r.h.dh = x ;
     r.h.bh = 0 ;
     int86(0x10, &r, &r) ;
 }
//**********************
int ls_in(char *s , char c)
 {
    register int i ;
    for(i = 0 ; *s ; i++)
      if(*s++ == c)
          return i + 1 ;
    return 0 ;
 }
```

ایجاد منوی popup بدون وقفه بایوس

در فصل ۱۶ چگونگی دسترسی مستقیم به حافظه نمایش مورد بررسی قرار گرفت که جهت یادآوری، می‌توانید به این فصل مراجعه نمایید. برای دسترسی به حافظه نمایش، باید نوع بورد گرافیکی مشخص گردد (چون آدرس حافظه نمایش، در بوردهای مختلف، متفاوت است). برای تشخیص بورد گرافیکی کامپیوتر، تابع video_mode() به صورت زیر نوشته شده است :

```
video_mode()
{
     union REGS r ;
     r.h.ah = 15 ;
     return int86 (0x10,&r,&r) & 255 ;
}
```

پس از تعیین بورد گرافیکی، دستورات زیر جهت تعیین آدرس باید نوشته شوند:

```
vmode = video_mode() ;
if (vmode!=2) && (vmode!=3) && (vmode!=7))
{
     printf("\n video must be 80  column text mode.") ;
     exit (1) ;
}
if (vmode == 7)
     vid_mem = (char far *) 0xB0000000 ;
else
     vid_mem = (char far *) 0xB8000000 ;
```

در این مجموعه دستورات، متغیر vid_mem یک اشاره‌گر از نوع char و به صورت far می‌باشد که به عنوان یک متغیر عمومی تعریف شده است.

در منوسازی به روش مستقیم، در دو تابع ()save_video و ()restore_video تغییراتی حاصل می‌شود که لیست آنها را در سطور بعدی مشاهده خواهیم کرد. در دو تابع ()save_video و ()restore_video مشاهده می‌کنید که کاراکترها و صفات آنها از طریق آدرس بورد گرافیکی خوانده و یا نوشته می‌شوند که در سایر توابع نیز باید تغییرات مناسبی اعمال شود.

همانطور که می‌دانید برای هر کاراکتر دو بایت از حافظه نمایش، اختصاص می‌یابد (بایت اول برای کاراکتر و بایت دوم برای صفت آن). لذا هر سطر از اطلاعات به ۱۶۰ بایت حافظه نیاز دارد که برای پیداکردن آدرس یک کاراکتر از رابطه زیر استفاده می‌شود:

آدرس = آدرس بورد گرافیکی + X * 160 + Y * 2

مثال ۲۰-۱۱

لیست تابع ()save_video در دسترسی مستقیم به حافظه.

```
void save_video(int startx, int endx, int starty,
          int endy, unsigned char *buf_ptr)  {
      register int i , j ;
      char far *v , far *t ;
      v=vid_mem ;
      for(i=starty ; i<endy;i++)
        for(j=startx;j<endx;j++)  {
          t=v+(j*160)+i*2 ;
          *buf_ptr++=*t++ ;
          *buf_ptr++=*t ;
          *(t-1)=' ' ;
        }
   }
```

مثال ۲۰-۱۲

لیست تابع ()restore_video در دسترسی مستقیم به حافظه.

```
void restore_video(int startx, int endx, int starty,
    int endy, unsigned char *buf_ptr)    {
      register int i , j ;
      char far *v , far *t ;
      v=vid_mem ;
      t=v ;
      for(i=starty ; i<endy;i++)
      for(j=startx;j<endx;j++)  {
         v=t ;
         v+=(j*160) + i*2 ;
         *v++=*buf_ptr++ ;
         *v=*buf_ptr++ ;
      }
   }
```

مثال ۲۰-۱۳

در روش مستقیم، به تابع دیگری جهت نوشتن کاراکتر و یا صفت آن در یک آدرس خاص نیاز است. نام این تابع write_char() انتخاب شد که لیست آن در زیر مشاهده می‌گردد.

```
void write_char(int x, int y, char ch, int attrib)
  {
     register int i ;
     char far *v ;
     v = vid_mem ;
     v += (x * 160) + y * 2 ;
     *v++ = ch ;
     *v = attrib ;
  }
```

مثال ۲۰-۱۴

برنامه کاملِ منوی popup در روش دسترسی مستقیم به حافظه.

```
#include <string.h>
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#define BORDER   1
#define ESC      27
#define REV_VID  0x60
#define NORM_VID 2
int popup(char *menu[], char *keys, int count, int x, int y, int border);
void display_menu(char *menu[] , int x, int y, int count);
void draw_border(int startx, int starty, int endx, int endy);
int get_resp(int x, int y, int count, char *menu[], char *keys);
void write_string(int x, int y, char *p, int attrib);
void write_char(int x, int y, char ch, int attrib);
void save_video(int startx, int endx, int starty,
     int endy, unsigned char *buf_ptr);
void restore_video(int startx, int endx, int starty,
     int endy, unsigned char *buf_ptr);
int video_mode();
int is_in(char *s , char c);
char far *vid_mem ;
char *fruit[]={"apple     ", "orange    ", "pear      ",
               "grape     ", "raspberry ", "strawberry" } ;
```

```
char *color []={ "red",  "yellow",  "orange", "green",} ;
char *apple_type[]={ "red delicious" ,  "jonthan",  "winesap",   "rome"};
int main()
 {
     int i,choice ;
     clrscr() ;
      choice=popup(fruit,"aoprts",6,1,3,BORDER) ;
      choice=popup(color , "ryog",4,5,10,BORDER) ;
      choice=popup(apple_type,"rjw",4,10,18,BORDER) ;
     if(choice == -2)
         printf("\n range error.");
     return 0;
 }
//**********************
int popup(char *menu[], char *keys, int count, int x, int y, int border)
 {
    register int i , len ;
    int endx , endy , choice , vmode ;
    unsigned char *p ;
    if((x>24) || (x<0) || (y>79) || (y<0))
    {
        printf("range") ;
        return -2 ;
    }
    vmode = video_mode() ;
    if((vmode!=2) && (vmode!=3) && (vmode!=7))
    {
        printf("video must be in 80 column text mode") ;
        exit(1) ;
    }
    if(vmode==7)
          vid_mem=(char far *) 0xB0000000 ;
    else
          vid_mem=(char far *) 0xB8000000 ;
    len=0 ;
    for(i=0 ; i<count ;i++)
     if(strlen(menu[i]) > len) len=strlen(menu[i]) ;
        endy=len+2+y ;
      endx=count+1+x ;
      if((endx+1 > 24) || (endy+1 >79))   {
               printf("menu won't fit") ;
               return -2 ;
      }
```

```
        p=(unsigned char *)  malloc(2*(endx-x+1)*(endy-y+1)) ;
        if(!p)
                exit(1) ;
        save_video(x,endx+1,y,endy+1,p);
        if(border)
                draw_border(x,y,endx,endy) ;
        display_menu(menu , x+1 , y+1 , count) ;
        choice=get_resp(x+1 , y , count , menu ,keys) ;
        restore_video(x,endx+1 , y , endy+1 , p) ;
        free(p) ;
             return choice ;
   }
 //************************
void display_menu(char *menu[] , int x, int y, int count)
   {
        register int i ;
        for(i=0 ; i < count ; i++ , x++)
            write_string(x , y , menu[i] , NORM_VID) ;
   }
 //*************************
void draw_border(int startx, int starty, int endx, int endy)
     {
      register int i ;
      char far *v , far *t ;
      v=vid_mem ;
      t=v ;
      for(i=startx+1 ; i<endx;i++)   {
          v+=(i*160)+starty*2 ;
          *v++=179 ;
          *v=NORM_VID ;
          v=t ;
          v+=(i*160)+endy*2 ;
          *v++=179 ;
          *v=NORM_VID ;
          v=t ;
      }
      for(i=starty+1 ; i<endy;i++)   {
           v+=(startx*160)+i*2 ;
           *v++=196 ;
           *v=NORM_VID ;
           v=t ;
           v+=(endx*160)+i*2 ;
           *v++=196 ;
```

```
            *v=NORM_VID ;
            v=t ;
        }
        write_char(startx , starty , 218 , NORM_VID) ;
        write_char(startx , endy , 191, NORM_VID) ;
        write_char(endx , starty , 192 , NORM_VID) ;
        write_char(endx , endy , 217 , NORM_VID) ;
    }
//************************
int get_resp(int x, int y, int count, char *menu[], char *keys)
    {
        union inkey {
            char ch[2] ;
            int i ;
        } c ;
        int arrow_choice=0 , key_choice ;
        y++ ;
        gotoxy(x,y) ;
        write_string(x,y,menu[0],REV_VID) ;
        for(;;)    {
            while(!bioskey(1));
                c.i=bioskey(0) ;
            gotoxy(x+arrow_choice,y) ;
            write_string(x+arrow_choice,y,menu[arrow_choice],NORM_VID);
            if(c.ch[0]) {
                key_choice=is_in(keys,tolower(c.ch[0])) ;
                if(key_choice)
                    return key_choice-1 ;
                switch(c.ch[0])  {
                  case '\r' : return arrow_choice ;
                  case ' '  : arrow_choice++ ;  break ;
                  case ESC  : return -1 ;
                }
            }//end of if
            else {
                switch(c.ch[1])   {
                  case 72 : arrow_choice -- ;   break ;
                  case 80 : arrow_choice ++ ;   break ;
                }
            }
            if(arrow_choice==count)
                arrow_choice=0 ;
            if(arrow_choice<0)
```

```
                arrow_choice=count-1 ;
            gotoxy(x+arrow_choice,y) ;
             write_string(x+arrow_choice,y,menu[arrow_choice],REV_VID);
        }//end of for
  }
 //*************************
 void write_string(int x, int y, char *p, int attrib)
  {
       register int i ;
       char far *v;
       v = vid_mem ;
       v += (x*160) + y*2 ;
       for (i=y ;*p;i++)   {
            *v++=*p++ ;
            *v++=attrib ;
       }
  }
 //**********************
 void write_char(int x, int y, char ch, int attrib)
   {
       register int i ;
       char far *v ;
       v = vid_mem ;
       v += (x * 160) + y * 2 ;
       *v++ = ch ;
       *v = attrib ;
   }
 //*************
 void save_video(int startx, int endx, int starty, int endy, unsigned
    char *buf_ptr)
  {
       register int i , j ;
       char far *v , far *t ;
       v=vid_mem ;
       for(i=starty ; i<endy;i++)
          for(j=startx;j<endx;j++) {
              t=v+(j*160)+i*2 ;
              *buf_ptr++=*t++ ;
              *buf_ptr++=*t ;
              *(t-1)=' ' ;
          }
 }
 //************************
```

```
void restore_video(int startx, int endx, int starty, int endy, unsigned
    char *buf_ptr)
  {
     register int i , j ;
     char far *v , far *t ;
     v = vid_mem ;
     t = v ;
     for(i = starty ; i< endy;i++)
        for(j = startx; j < endx;j++) {
          v=t ;
          v+=(j*160) + i*2 ;
          *v++=*buf_ptr++ ;
          *v=*buf_ptr++ ;
        }
}
//*******************
int video_mode()
   {
     union REGS  r ;
     r.h.ah=15 ;
     return int86(0x10 , &r , &r) & 255 ;
   }
//***********************
int is_in(char *s , char c)
{
     register int i ;
     for(i=0 ; *s ; i++)
        if(*s++==c)
           return i+1 ;
        return 0 ;
}
```

ایجاد منوی pulldown

منوی pulldown به صورت پنجره‌های مختلفی است که برخلاف منوی popup ، امکان نمایش چند پنجره در آنِ واحد در صفحه‌نمایش وجود دارد. در منوی pulldown در حین ظاهر شدن منو، چنانچه لازم باشد، صفحه‌نمایش ذخیره شده پس از عمل انتخاب، بازیابی می‌گردد.

در منوی pulldown هر پنجره با چارچوب (menu frame) خاصی مشخص می‌شود که برای فعال شدن هر پنجره از شماره آن پنجره استفاده می‌گردد و سایر اطلاعات موردنیاز، توسط توابع دیگر در دسترس قرار می‌گیرند. بهترین راه برای پشتیبانی چارچوب منو، تعریف آرایه‌ای از ساختمان است که حاوی کلیه اطلاعات مربوط به آن منو می‌باشد. این ساختمان به صورت زیر تعریف می‌شود:

```
struct menu_frame {
        int startx, endx, starty, endy ;
        unsigned char *P ;
        char **menu ;
        char *keys ;
        int border, count ;
        int active ;
} frame [MAX_FRAME] ;
```

ماکروی MAX_FRAME حداکثر تعداد منوها را مشخص می‌کند. تنها چیزی که در منوی pulldown وجود دارد ولی در منوی popup مشاهده نمی‌گردد، متغیری است که فعال، یا غیرفعال بودن منو را در صفحه‌نمایش مشخص می‌کند (عنصر active از ساختمان menu_frame).

مثال ۱۵-۲۰

تابع ()make_menu که برای تعریف چارچوب منو مورد استفاده قرار می‌گیرد. پارامترهایی که در فراخوانی تابع ()make_menu استفاده می‌شوند همان پارامترهای تابع ()popup می‌باشند، با این تفاوت که تعداد منوها باید اولین پارامتر تابع باشد. **لیست این تابع را در مثال ۱۸-۲۰ ببینید.**

مثال ۱۶-۲۰

تابع ()pulldown که برای ایجاد منوی pulldown نوشته شده است. برای استفاده از تابع ()pulldown کافی است تعداد منوهای موردنیاز، به آن وارد شود.

```
int pulldown(int num)
/* display pulldown menu and return selection*/ {
        int vmode, choice ;
        vmode = video_mode() ;
        if((vmode != 2) && (vmode != 3) && (vmode != 7))
        {
           printf("\n video must be in 80 column text.");
           exit(1) ;
        }
        if(vmode==7)
             vid_mem=(char far  *) 0xB0000000;
        else
             vid_mem=(char far  *) 0xB8000000;
        /* get active window */
        if(!frame[num].active) /* not currently in use */
        {
           save_video(num) ; /* save the current screen */
           frame[num].active = 1 ; /* set active flag */
        }
        if(frame[num].border) drow_border(num) ;
        display_menu(num) ;
        return get_resp(num) ;
}
```

مثال ۲۰-۱۷

برای بازیابی محتویات صفحه‌نمایش که در بافری ذخیره شده است، از تابع restore_video() استفاده می‌شود که در موقع مناسبی باید از آن استفاده نمود. سعی شود که تعدادی از منوهای pulldown در صفحه‌نمایش وجود داشته باشند تا کاربر کلیه انتخابهای خود را انجام دهد. تابع restore_video() که در منوی pulldown مورد استفاده قرار می‌گیرد، تغییراتی نسبت به منوی popup پیدا می‌کند که لیست آن در زیر مشاهده می‌گردد.

```
void restore_video(int num)
    {
      register int i , j ;
      char far *v , far *t , far *buf_ptr ;
      buf_ptr = frame[num].p ;
      v = vid_mem ;
      t = v ;
      for(i = frame[num].starty ; i < frame[num].endy + 1; i++)
        for(j = frame[num].startx; j < frame[num].endx+1;j++)
        {
          v = t ;
          v += (j * 160) + i * 2 ;
          *v++ = *buf_ptr++ ;
          *v = *buf_ptr++ ;
        }
        frame[num].active=0 ; /* deactivate */
}
```

مثال ۲۰-۱۸

برنامه کامل منوی pulldown.

```
#include <string.h>
#include <dos.h>
#include <ctype.h>
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
#define BORDER      1
#define ESC         27
#define REV_VID  0x70
#define NORM_VID    2
#define MAX_FRAME  10
int make_menu(int num, char *menu[], char *keys, int count, int x, int y, int
border);
void  goto_xy(int, int);
int   pulldown(int num);
void  restore_video(int num);
void  save_video(int num);
```

```
int   video_mode();
void  drow_border(int num);
void  display_menu(int num);
int   get_resp(int num);
void  write_string(int x, int y, char *p, int attrib);
void  write_char(int x, int y, char ch, int attrib);
void  pd_driver() ,cls() ;
int   is_in(char *s , char c);
char far *vid_mem ;
struct menu_frame {
      int startx,endx,starty,endy ;
      unsigned char *p ;
      char **menu ;
      char *keys ;
      int border,count ;
      int active ;
} frame[MAX_FRAME], i ;
char *fruit[]={"apple      ", "orange     ", "pear       ",
               "grape      ", "raspberry ", "strawberry" } ;
char *color []={"red       ",  "yellow   ", "orange     ", "green    } ;
char *apple_type[]={ "red delicious" ,  "jonthan",  "winesap",   "rome"};
char *grape_type[]= { "Concord " ,  "cAnadice " ,   "Thompson" ,  "Red flame" } ;
int main()
 {
      int i ;
      cls() ;
      goto_xy(0, 0);
       make_menu(0,fruit,"aopgrs",6,5,20,BORDER) ;
       make_menu(1,color , "ryog",4,9,28,BORDER) ;
       make_menu(2,apple_type,"rjwr",4,12,32,BORDER) ;
       make_menu(3,grape_type,"catr",4,9,10,BORDER) ;
      printf("\n select your fruit:") ;
      pd_driver() ;/* active the menu system */
      return 0;
 }
//************************
void pd_driver()
{
  int  choice1,choice2,selection ;
  /* active as need */
  while((choice1 = pulldown(0)) != -1)  {
     switch(choice1) {
         case 0 : /* want an apple */
             while((choice2 = pulldown(1)) != -1)  {
                 if(choice2 == 0)  selection=pulldown(2) ; /* rea */
```

```
                restore_video(2) ;
            }//end of while
            restore_video(1) ;
            break ;
        case 1 :
        case 2 :
            goto_xy(1, 0);
            printf("\n out of that seection ") ;
            break ;
        case 3: /* want a grape */
            selection=pulldown(3) ;
            restore_video(3) ;
            break ;
        case 4 :
        case 5 :
            goto_xy(1,0) ;
            printf("\n out of that selection ") ;
            break ;
    }//end of switch
  }//end of while
  restore_video(0) ;
}
//************************
int pulldown(int num)
/* display pulldown menu and return selection*/
 {
     int vmode, choice ;
     vmode = video_mode() ;
     if((vmode != 2) && (vmode != 3) && (vmode != 7))
     {
         printf("\n video must be in 80 column text.");
         exit(1) ;
     }
     if(vmode==7)
        vid_mem=(char far  *) 0xB0000000;
     else
        vid_mem=(char far  *) 0xB8000000;
     /* get active window */
     if(!frame[num].active) /* not currently in use */
     {
         save_video(num) ; /* save the current screen */
         frame[num].active = 1 ; /* set active flag */
     }
     if(frame[num].border)
          drow_border(num) ;
```

```
        display_menu(num) ;
        return get_resp(num) ;
    }
//****************
int make_menu(int num, char *menu[], char *keys,
        int count, int x, int y, int border)
 {
    register int i , len ;
    int endx , endy , choice , vmode ;
    unsigned char *p ;
    if(num > MAX_FRAME) {
          printf("\n too many menus.") ;
          return 0 ;
     }
     if((x > 24) || (x < 0) || (y > 79) || (y < 0)) {
        printf("range error") ;
        return 0 ;
     }
     len=0 ;
     for(i = 0 ; i < count ;i++)
         if(strlen(menu[i]) > len)
              len=strlen(menu[i]) ;
     endy=len+2+y ;
     endx=count+1+x ;
     if((endx + 1 > 24) || (endy + 1 > 79)) {
           printf("menu won't fit") ;
           return 0 ;
     }
     p = (unsigned char *)  malloc(2*(endx-x+1)*(endy-y+1)) ;
     if(!p)  exit(1) ;
     frame[num].startx = x ;
     frame[num].endx = endx ;
     frame[num].starty = y ;
     frame[num].endy = endy ;
     frame[num].p = p ;
     frame[num].menu = (char * *)menu ;
     frame[num].border = border ;
     frame[num].keys = keys ;
     frame[num].count = count ;
     frame[num].active = 0 ;
     return 1 ;
 }
//*********************
void display_menu(int num)
   {
```

```
        register int i,x ;
        char **m ;
        x = frame[num].startx+1 ;
        m = frame[num].menu ;
        for(i = 0 ; i < frame[num].count ; i++ , x++)
            write_string(x ,frame[num].starty+1 , m[i] , NORM_VID) ;
      }
//******************
void drow_border(int num)
     {
      register int i ;
      char far *v , far *t ;
      v = vid_mem ;
      t = v ;
      for(i = frame[num].startx+1 ; i < frame[num].endx; i++)
      {
        v += (i * 160) + frame[num].starty * 2 ;
        *v++ = 179 ;
        *v = NORM_VID ;
        v = t ;
        v += (i*160)+frame[num].endy*2 ;
        *v++ = 179 ;
        *v = NORM_VID ;
        v = t ;
      }
      for(i = frame[num].starty + 1 ; i < frame[num].endy; i++)
      {
          v += (frame[num].startx*160) + i * 2 ;
          *v++ = 196 ;
          *v = NORM_VID ;
          v = t ;
          v += (frame[num].endx * 160) + i * 2 ;
          *v++ = 196 ;
          *v = NORM_VID ;
          v = t ;
      }
       write_char(frame[num].startx , frame[num].starty , 218 , NORM_VID) ;
       write_char(frame[num].startx , frame[num].endy , 191, NORM_VID) ;
       write_char(frame[num].endx , frame[num].starty , 192 , NORM_VID) ;
       write_char(frame[num].endx , frame[num].endy , 217 , NORM_VID) ;
 }
 //**********************
int get_resp(int num)
  {
       union inkey {
```

```
            char ch[2] ;
            int i ;
        } c ;
        int arrow_choice = 0 , key_choice ;
        int x, y ;
        x = frame[num].startx + 1 ;
        y = frame[num].starty + 1 ;
       /*highligth the first selection*/
        goto_xy(x, y) ;
        write_string(x, y, frame[num].menu[0], REV_VID) ;
        for(;;)  {
           while(!bioskey(1)); /* wait for key stroke*/
           c.i=bioskey(0) ; /* read the key */
           goto_xy(x + arrow_choice, y) ;
write_string(x+arrow_choice, y, frame[num].menu[arrow_choice],
NORM_VID);
           if(c.ch[0])  {
              key_choice =  is_in(frame[num].keys,tolower(c.ch[0])) ;
              if(key_choice)
                  return key_choice - 1 ;
              switch(c.ch[0])  /*check for Enter or space key */
              {
                 case '\r' : return arrow_choice ;
                 case ' '  : arrow_choice++ ; break ;
                 case ESC  : return -1 ;
              }
           } //end of if
           else  {
              switch(c.ch[1])    {
                 case 72 : arrow_choice -- ;    break ;
                 case 80 : arrow_choice ++ ;    break ;
              }//end of switch
           }//end of else
           if(arrow_choice==frame[num].count)
               arrow_choice = 0 ;
           if(arrow_choice < 0)
               arrow_choice = frame[num].count - 1 ;
           goto_xy(x + arrow_choice, y) ;
            write_string(x+arrow_choice,y,frame[num].menu[arrow_choice],REV_VID);
       }//end of for(;;)
  }
/*******************/
void write_string(int x, int y, char *p, int attrib)
  {
      register int i ;
```

```
        char far *v;
        v = vid_mem ;
        v += (x * 160) + y * 2 ;
        for (i = y ; *p; i++) {
            *v++ = *p++ ;
            *v++ = attrib ;
        }//end of for
    }
/*********************/
void write_char(int x, int y, char ch, int attrib)
    {
        register int i ;
        char far *v ;
        v = vid_mem ;
        v += (x * 160) + y * 2 ;
        *v++ = ch ;
        *v = attrib ;
    }
/*****************/
void save_video(int num)
{
        register int i , j ;
        char far *buf_ptr, far *v , far *t ;
        buf_ptr = frame[num].p ;
        v = vid_mem ;
        for(i = frame[num].starty ; i < frame[num].endy + 1; i++)
          for(j = frame[num].startx; j < frame[num].endx + 1; j++)
          {
            t = v + (j * 160) + i * 2 ;
            *buf_ptr++ = *t++ ;
            *buf_ptr++ = *t ;
            *(t - 1) = ' ' ;
          }
}
void restore_video(int num)
    {
        register int i , j ;
        char far *v , far *t , far *buf_ptr ;
        buf_ptr = frame[num].p ;
        v = vid_mem ;
        t = v ;
        for(i = frame[num].starty ; i < frame[num].endy + 1; i++)
          for(j = frame[num].startx; j < frame[num].endx+1;j++)
          {
            v = t ;
```

```
            v += (j * 160) + i * 2 ;
            *v++ = *buf_ptr++ ;
            *v = *buf_ptr++ ;
          }
          frame[num].active=0 ; /* deactivate */
 }
 /*****************/
 void cls()
    {
     union REGS r ;
     r.h.ah = 6 ;
     r.h.al = 0 ;
     r.h.ch = 0 ;
     r.h.cl = 0 ;
     r.h.dh = 24 ;
     r.h.dl = 79 ;
     r.h.bh = 7 ;
     int86(0x10, &r, &r) ;
   }
 /*******************/
 void goto_xy(int x, int y)
    {
      union  REGS r ;
      r.h.ah = 2 ;
      r.h.dl = y ; /* column */
      r.h.dh = x ;
      r.h.bh = 0 ;
      int86(0x10, &r, &r) ;
  }
 //************************
 int is_in(char *s , char c)
  {
     register int i ;
     for(i = 0 ; *s ; i++)
       if(*s++ == c)
           return i + 1 ;
     return 0 ;
 }
 //*************
 int video_mode()
 {
       union REGS  r ;
       r.h.ah = 15 ;
       return int86(0x10, &r, &r) & 255 ;
  }
```

ارتباط زبان C با اسمبلی

یکی از ویژگیهای جالب زبان C این‌است‌که می‌تواند با زبان اسمبلی ارتباط برقرار کند. این ارتباط به دو صورت امکان‌پذیر است. در روش اول، می‌توان در هر مکانی از برنامه زبان C از یک یا چند دستور اسمبلی استفاده کرد. در روش دوم، برنامه اسمبلی به‌عنوان یک زیربرنامه نوشته شده، در برنامه C فراخوانی می‌شود.

دستورات اسمبلی در زبان C

برای نوشتن دستورات اسمبلی در زبان C از دستور asm به صورت زیر استفاده می‌شود:

```
asm {
        دستورات اسمبلی
}
```

مثال ۱-۲۱

برنامه‌ای که کاراکتر ♥ را ۵ بار در صفحه‌نمایش چاپ می‌کند.

```
#include <conio.h>
int main()
{
   clrscr();
   asm {
      mov    ah, 9h   // function call to print
      mov    bh, 0    // page number
      mov    bl, 7    // attribute
      mov    cx, 5    // number of print
      mov    al, 03h  //heart character
      int    10h      //interrupt call
   }
   getch();
   return 0;
}
```

استفاده از زیربرنامه‌های اسمبلی در برنامه C

برنامه‌نویسان ترجیح می‌دهند روالهای اسمبلی را به صورت مجزا نوشته، به obj تبدیل کرده آنها را در زبان C فراخوانی کنند. برای برقرارکردن ارتباط بین C و اسمبلی باید سه نکته را رعایت نمود.

۱. نوع و ترتیب استفاده از سگمنت‌ها در توربو اسمبلر باید با نوع و ترتیب سگمنت‌ها در توربو C تطبیق کامل داشته باشد.

۲. برنامه‌های TC و TASM شناسه‌ها و توابع خود را به اشتراک گذارند.

۳. استفاده از TLINK برای برقراری ارتباط.

کوچک و بزرگ بودن حروف و متغیرها

در زبان C، بین حروف بزرگ و کوچک تفاوت است ولی توربو اسمبلر بین حروف بزرگ و کوچک فرقی قائل نمی‌شود. لذا در ارتباط بین اسمبلی و C به این موضوع توجه داشته باشید.

پیش فرض سگمنت

در برخی از زیربرنامه‌های اسمبلی، نیاز داریم انواع سگمنت داده را تعریف نماییم. هنگام ورود از برنامه C به برنامه اسمبلی، توربو C ثباتهای CS و DS را با پیش فرض معینی مقداردهی می‌کند.

در تمام مدلهای حافظه tiny ، small و medium ثباتهای SS و DS مقادیر یکسانی را به خود می‌گیرند. جدول ۱-۲۱ مدل سگمنت را در توربو C و توربو اسمبلر نمایش می‌دهد:

ارتباط شناسه‌های extern و public در توربو C و توربو اسمبلر

توربو اسمبلر می‌تواند به شناسه‌های عمومی و توابع خارجی توربو C دسترسی پیدا کند و توربو C نیز همینطور. در این صورت TC انتظار دارد تمام شناسه‌ها و توابع عمومی در اسمبلی با خط (_) شروع شود و ما باید تمام شناسه‌ها و توابع خارجی C را در اسمبلی با خط ربط شروع کنیم. اگر شناسه‌ای برای یک برنامه، خارجی باشد در اسمبلی باید به صورت extrn تعریف شود. اگر برنامه اسمبلی در TC فراخوانی می‌گردد این برنامه در TC باید به صورت extern تعریف شود و در اسمبلی باید به صورت public تعریف گردد.

جدول ۱-۲۱ جدول مدل سگمنت در توربو C و توربو اسمبلر.

MODEL	CS	DS
ting	_TEXT	DGROUP
small	_TEXT	DGROUP
compact	_TEXT	DGROUP
medium	FILENAME_TEXT	DGROUP
large	FILENAME_TEXT	DGROUP
huge	FILENAME_TEXT	CALLING_FILENAME_DATA

مثال ۲-۲۱

برنامه‌ای که متغیر i را در برنامه C تعریف‌می‌کند و در برنامه اسمبلی یک واحد به آن اضافه می‌کند و در برنامه C مقدار i را چاپ می‌کند.

توضیح

زیربرنامه inci در C به صورت extern تعریف شد و در برنامه اسمبلی در جلوی آن خط ربط قرار گرفت و از نوع far تعریف گردید. متغیر i در C به صورت متغیر عمومی (خارج از ()main) و در اسمبلی به صورت extrn ، با خط ربط و از نوع word تعریف شد (چون نوع متغیر i در C، از نوع int تعریف شده است). در برنامه C، مقدار i برابر با صفر است که در برنامه اسمبلی یک واحد به آن اضافه شده است، در نتیجه، خروجی برنامه C به صورت i = 1 است. برای اجرای برنامه کافی است در خط فرمان دستور زیر را تایپ کنید:

TCC - MC C نام برنامه اسمبلی نام برنامه ASM.

پس از اجرای این دستور، یک فایل EXE همنام با فایل برنامه C ایجاد می‌شود که با تایپ نام برنامه اجرایی، برنامه اجرا می‌شود.

برنامهٔ C

```
#include <stdio.h>
extern void far inci();
extern void far cls(void);
int i=0;
void main(void) {
        inci();
        printf("\n  i=%i",i);
}
```

برنامهٔ اسمبلی

```
.model small
.data
extrn _i:word
.code
public  _inci
_inci       proc       far
        inc  _i
        retf
_inci     endp
          end
```

ترجمه چند فایل C و اسمبلی

برای ترجمه چند فایل C و اسمبلی، دو روش وجود دارد. در روش اول از فرمان TCC به صورت زیر استفاده می‌شود:

نام فایل اسمبلی n ... نام فایل۲ اسمبلی نام فایل۱ اسمبلی نام فایل Cn ... نام فایل C2 نام فایل C1 TC -MS

در این روش ابتدا نام فایلهای C را بدون پسوند در جلوی TCC قرار می‌دهیم و سپس نام فایل اسمبلی را با پسوند ASM، بعد از آنها ذکر می‌کنیم. به‌این‌ترتیب یک فایل EXE به نام اولین فایل C ایجاد می‌گردد.

در روش دوم، فایلها را در TC در فایل پروژه قرار می‌دهیم و فایلهای اسمبلی باید قبلاً توسط TASM یا MASM به فایلهای obj ترجمه گردند. برای آشنایی با شیوه تشکیل فایلهای پروژه در C، به فصل ۱۷ کتاب مراجعه کنید.

تبادل پارامترها بین اسمبلی و TC

در اینجا دو موضوع به شرح ذیل مورد بحث قرار می‌گیرند:

۱. انتقال اطلاعات از برنامه C به اسمبلی
۲. بازگرداندن مقادیر بازگشتی از برنامه اسمبلی به برنامه C

ارسال پارامترها از برنامه C به اسمبلی

برای ارسال پارامترها از برنامه C به برنامه اسمبلی، دو روش وجود دارد، در روش اول از ثباتها و در روش دوم از پشته استفاده می‌شود.

مثال ۳-۲۱

برنامه‌ای که یک نام را از ورودی می‌خواند و ناحیه‌ای از صفحه‌نمایش را پاک می‌کند و نام خوانده شده را وسط این ناحیه چاپ می‌کند. هدف از این برنامه آشنایی با انتقال پارامترها از طریق ثباتها به اسمبلی است.

برنامهٔ C

```
#include <stdio.h>
#include <conio.h>
extern void far cls(void);
void main(void)
{
  char fname[21];
  printf ( "\nplease enter name :");
  gets(fname);
  clrscr();
  asm {
      mov    al, 10
      mov    ch, 5
      mov    cl, 10
      mov    dh, 15
      mov    dl, 40
      mov    bh, 17h
  }
```

```
    cls(); // call assembly procedure
    asm {
        mov     ah, 02   //cursor move
        mov     dh, 10   //row
        mov     dl, 25   //column
        mov     bh, 0    //page number
       int     10h      //interrupt call
    }
    printf("%s",fname);
    getch();
    clrscr();
}
```

برنامهٔ اسمبلی

```
IGROUP    group  _TEXT                    ;program segment
DGROUP    group  _BSS,_DATA                ;data segment
assume    cs:IGROUP , ds:DGROUP , es:DGROUP , ss:DGROUP
_BSS      segment    word   public  'BSS'
_BSS      ends
_DATA     segment    word   public   'DATA'
_DATA     ends
_TEXT     segment    byte   public   'CODE'
 public  _cls
_cls      proc     far
          mov      ah, 06
          int    10h
          ret
_cls      endp
_TEXT      ends
          end
```

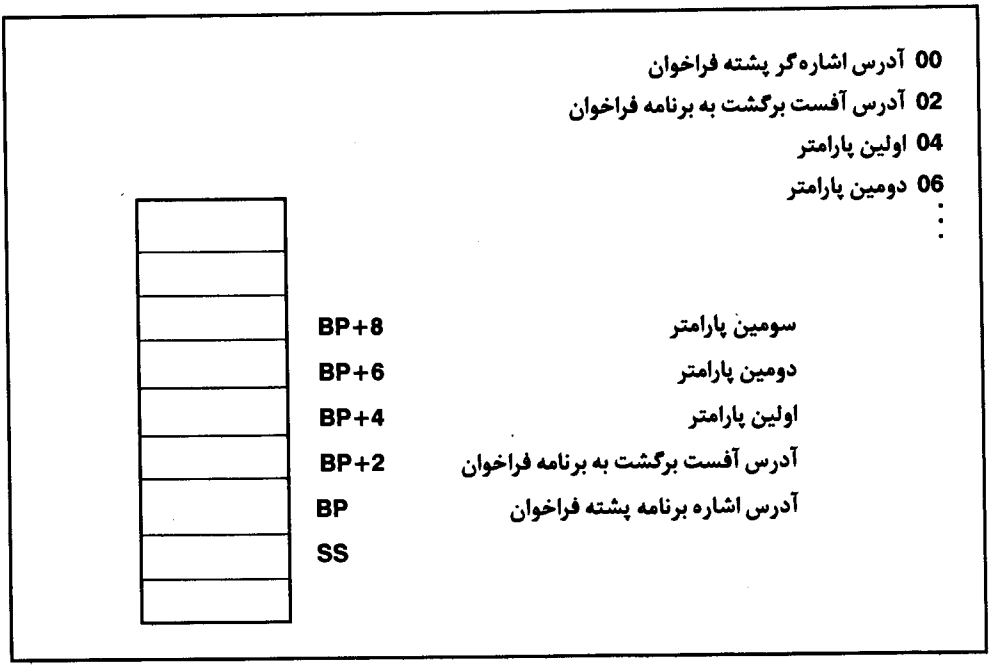
استفاده از پشته برای انتقال پارامترها

در زبان C برخلاف بقیه زبانها پارامترها از سمت راست به چپ در پشته ذخیره می‌شوند (به طوری که ابتدا سمت راست‌ترین پارامتر در پشته ذخیره می‌گردد). دستور فراخوانیِ زیر را درنظر بگیرید:

```
locatexy (row,col);
```

این دستور، کدهای زیر را ایجاد می‌کند.

```
push   word ptr    dgroup :_col
push   word ptr    dgroup :_row
call   locatexy
add    sp, 4
```



شکل ۱-۲۱ ذخیره پارامترهای زیربرنامه C در برنامه اسمبلی نوع near.

در ابتدای برنامه اسمبلی، باید BP را در پشته ذخیره کرد تا بتوان از برنامه اسمبلی به برنامه C برگشت نمود و سپس SP را در BP قرار داد. در این صورت برای دسترسی به پارامترها در برنامه اسمبلی، دو حالت پیش می‌آید. در حالتی که برنامه اسمبلی near است، پارامترها مانند شکل ۱-۲۱ در پشته ذخیره می‌شوند.

مثال ۴-۲۱

یک کاراکتر و تعداد دفعات چاپ آن، در برنامه C از ورودی خوانده می‌شود و با استفاده از یک برنامه اسمبلی، کاراکتر خوانده شده، به این تعداد نمایش داده می‌شود(برنامه اسمبلی از نوع near است).

توضیح

برنامه اسمبلی disp، در C به صورت extern و در برنامه اسمبلی به صورت public و از نوع near تعریف شد. برنامه C با یک پیام، کاراکتر را می‌خواند و در ch قرار می‌دهد و بعد از آن، تعداد دفعاتی که کاراکتر باید چاپ شود خوانده شده در متغیر count قرار می‌گیرد. سپس برنامه اسمبلی disp با دو پارامتر ch و count فراخوانی می‌شود. چون در برنامه اسمبلی، disp از نوع near است، برای دسترسی به پارامتر اول محتویات [BP+04] به ثبات AL (کاراکتری که باید نمایش داده شود) و محتویات پارامتر دوم، [BP+06]، به ثبات CX منتقل شده است (تعداد دفعاتی که کاراکتر باید نمایش داده شود).

با استفاده از تابع شماره 0eH، وقفه 10H کاراکتر موجود در ثبات AL در صفحه‌نمایش چاپ شده است. دستور INT 10H در حلقه تکرار، کاراکتر موجود در ثبات AL را به تعداد CX بار در صفحه‌نمایش چاپ می‌کند.

برنامهٔ C

```
#include <stdio.h>
#include <conio.h>
extern void    disp(char,int);
void main(void)
{
  char  ch;
  int count;
  printf ( "\nPlease  enter a char:");
  scanf("%c",&ch);
  printf ( "\nPlease  count for display char:");
  scanf("%i",&count);
  disp(ch,count);
}
```

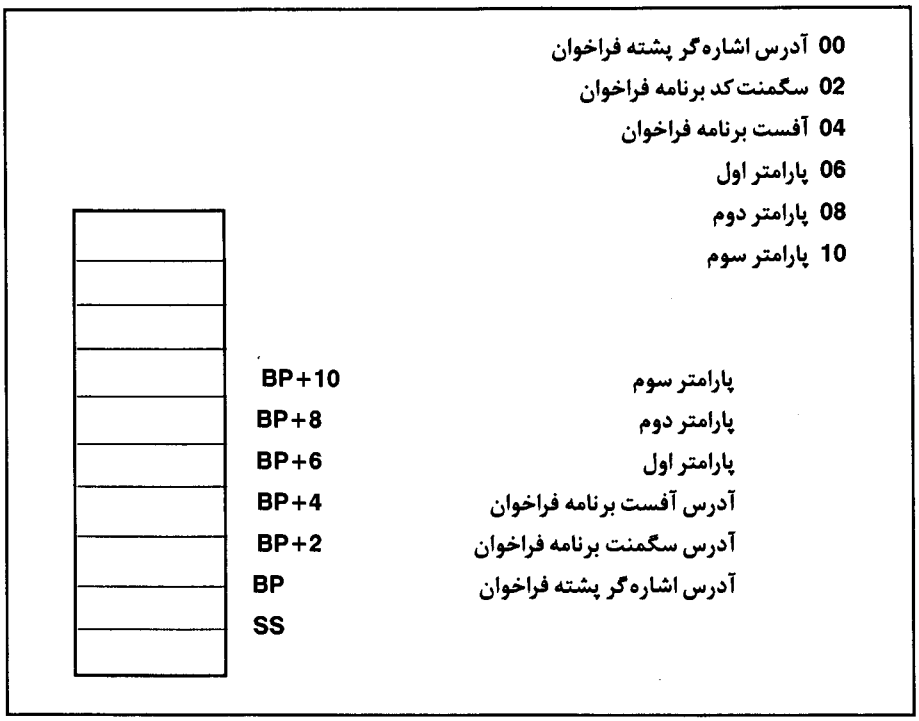
برنامهٔ اسمبلی

```
.model small
.data
.code
public   _disp
_disp    proc      near
         push      bp
         mov       bp, sp
         mov       al, [bp+04]
         mov       cx, [bp+06]
         mov       ah, 0eh
         loop1:
         int 10h
         loop  loop1
         pop   bp
         ret  4
_disp    endp
end
```

خروجی برنامه

```
please enter a char : B
please enter count for display char : 12
BBBBBBBBBBBB
```

در حالتی که نوع برنامه اسمبلی far است، پارامترها به صورت شکل ۲-۲۱ در پشته قرار می‌گیرند.



شکل ۲-۲۱ ذخیره شدن پارامترهای C در برنامه اسمبلی نوع far.

مثال ۵-۲۱

برنامه‌ای که سطر و ستون را در C از ورودی خوانده با استفاده از یک برنامه اسمبلی به نام locatexy مکان‌نما را به این سطر و ستون انتقال می‌دهد و یک خروجی را در آن سطر و ستون چاپ می‌کند. زیربرنامه locatexy از نوع far می‌باشد.

برنامهٔ C

```
#include <stdio.h>
#include <conio.h>
extern void far locatexy(int,int);
void main(void) {
  int row,col;
  clrscr();
  printf ( "\nPlease  enter row (1-24):");
  scanf("%i",&row);
  printf ( "\nPlease enter row (1-79):");
  scanf("%i",&col);
  clrscr();
  locatexy(row,col);
  printf("link  a moudle c with assambly (far) ");
}
```

برنامه اسمبلی Locatexy

```
IGROUP    group  _TEXT          ;program segment
DGROUP    group  _BSS,_DATA  ;data segment
assume    cs:IGROUP , ds:DGROUP , es:DGROUP , ss:DGROUP
_BSS      segment    word   public  'BSS'
_BSS      ends
_DATA     segment    word   public   'DATA'
_DATA     ends
_TEXT     segment    byte   public   'CODE'
 public   _locatexy
_locatexy     proc     far
      push     bp
      mov      bp, sp
      mov      dh, [bp+06]
      mov      dl, [bp+08]
      mov      ah, 02
      xor      bh, bh
      int      10h
      pop      bp
      retf   4
_locatexy   endp
_TEXT   ends
       end
```

همانطور که در برنامه اسمبلی ملاحظه می‌شود برای دسترسی به پارامتر اول، محتویات [BP+06] را در ثبات dh و برای دسترسی به پارامتر دوم، محتویات [BP+08] را در dl قرار می‌دهیم. با اجرای تابع شماره ۲ وقفه 10H مکان‌نما به سطری که مقدار آن در dh و ستونی که مقدار آن در ثبات dl قرار دارد، انتقال می‌یابد.

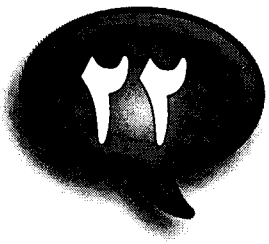
برای برگشت از برنامه اسمبلی به برنامه C، دستور retf 4 در انتهای برنامه اسمبلی قرار دارد (4 به معنی این است که چهار بایت از پشته بازیابی می‌شود). زیرا دو پارامتر دو بایتی از نوع int، در هنگام فراخوانی برنامه اسمبلی، در پشته قرار گرفتند. قبل از بازگشت از برنامه اسمبلی، این دو پارامتر باید از پشته بازیابی شوند تا اختلال در خاتمه برنامه پیش نیاید.

بازگرداندن مقادیر از اسمبلی به C

برای بازگرداندن مقادیر یک بایتی و دو بایتی از ثبات AX و چهار بایتی از ثبات DX:AX استفاده می‌شود و مقادیر بیش از چهار بایت را به صورت static ذخیره کرده، اشاره‌گری به آنها را برگشت می‌دهیم. نوع اطلاعات و نحوه برگشت مقادیر در جدول ۲-۲۱ آورده شده است.

جدول ۲-۲۱ نوع اطلاعات و نحوه بازگشت مقادیر بین اسمبلی و C .

نوع مقدار بازگشتی	ثبات
unsingned char	AX
char	AX
enum	AX
unsigned short int	AX
short int	AX
int	AX
unsigned int	AX
long	DX:AX
float ، double, long double	8087 top of stack
near *	AX
far *	DX:AX



دستورات پیش پردازنده

پیش پردازنده[1]، نوعی مترجم است که دستورات توسعه‌یافته‌ای از یک زبان برنامه‌سازی را گرفته، به دستورات قابل فهم برای کامپایلر همان زبان تبدیل می‌کند. به عبارت دیگر، ورودی پیش پردازنده، برنامه‌ای با دستورات توسعه‌یافته است و خروجی آن، برنامه دیگری است که توسط پردازنده(CPU) قابل ترجمه و اجرا است. دستورات توسعه‌یافته‌ای در زبان C وجود دارند که توسط پردازنده قابل ترجمه و اجرا نیست، بلکه فقط توسط پیش پردازنده قابل فهم هستند و به دستورات **پیش پردازنده** معروفند. دستورات پیش پردازنده در زبان C از اهمیت ویژه‌ای برخوردار اند.

تعریف ماکرو

ماکرو[2]، نامی برای یک رشته است که این رشته می‌تواند ترکیبی از حروف، ارقام، مقادیر ثابت، توابع و غیره باشد. دستور پیش‌پردازندهٔ #define برای تعریف ماکرو استفاده می‌شود. این دستور قبل از تابع ()main به دو صورت به کار می‌رود که یک روش آن به صورت زیر است:

#define <نام ماکرو> <رشته>

نام ماکرو، همانند نام یک متغیر در C است که حداقل باید با رشتهٔ تعریف کننده ماکرو، یک فاصله داشته باشد و بهتر است جهت مشخص بودن در برنامه، با حروف بزرگ انتخاب شود. به عنوان مثال، دستور #define MAX 20 موجب می‌شود تا مقدار ماکروی MAX در سرتاسر برنامه برابر با ۲۰ فرض شود. یعنی در هر جای برنامه که از ماکروی MAX استفاده شود، مثل این است که از عدد ۲۰ استفاده شده است.

در مورد دستورات پیش پردازنده در زبان C باید این نکته را در نظر داشت که آنها علیرغم سایر دستورات زبان C، به ; ختم نمی‌شوند.

مثال ۱-۲۲

برنامه‌ای که چگونگی استفاده از ماکروها را نشان می‌دهد. این برنامه تعدادی عدد را از ورودی خوانده و مجموع آنها را محاسبه می‌کند (اگر عددی که وارد می‌شود، منفی باشد، برنامه متوقف می‌شود). در این مثال به‌جای تابع ()main از PROGRAM، به جای { از BEGIN، به جای { از END و ... استفاده شده است. بدیهی است که انتخاب این نامها به نظر برنامه‌نویس بستگی دارد و می‌تواند هر چیز دیگری را به جای آنها انتخاب نماید.

```
#include <stdio.h>
#include <conio.h>
#define PROGRAM main()
```

1. preprocessor 2. macro

```
#define FLOAT   float
#define WHILE while(
#define DO    )
#define BEGIN {
#define END   }
#define FORMATOUT "the sum is: %6.2f\n"
#define FORMATIN "%f"
#define MESSAGE  printf("enter numbers(negative to end):");
PROGRAM
 BEGIN
  FLOAT data, sum=0.0;
  clrscr();
  MESSAGE
  scanf(FORMATIN, &data);
  WHILE data >= 0.0 DO
  BEGIN
    sum += data;
    scanf(FORMATIN, &data);
  END
  printf(FORMATOUT, sum);
  getch();
  return 0;
 END
```

خروجی

```
enter numbers (negative to end) : 4  9  11  -2
th sum is : 24.00
```

مثال ۲-۲۲

برنامه‌ای که با استفاده از علامت * ، مثلثی را در صفحه‌نمایش رسم می‌کند.

```
#include <stdio.h>
#include <conio.h>
#define NUMLINE  5
#define BLANK    ' '
#define ASTRISK '*'
#define NEWLINE '\n'
int main()
{
  int l, le, s ;
  l = 1;
  clrscr();
  while (l <= NUMLINE) {
      le = 1 ;
      while (le++ <= NUMLINE - l)
```

```
            printf("%c", BLANK);
        s = 1;
        while(s++ <= 2 * I - 1)
          printf("%c", ASTRISK);
        printf("%c", NEWLINE);
        I ++;
    }
    getch();
    return 0;
}
```

خروجی

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

کاربرد دیگر دستور #define در تعریف ماکروهایی است که دارای پارامتر باشند. در این مورد از دستور #define به صورت زیر استفاده می‌شود.

#define نام ماکرو (اسامی پارامترها) تعریف ماکرو

تعریف ماکرو، مشخص می‌کند که چه عملی باید توسط ماکرو انجام گیرد. **اسامی پارامترها** متغیرهایی هستند که در حین اجرای ماکرو به آن منتقل می‌شوند؛ اگر تعداد آنها بیش از یکی باشد، با کاما از یکدیگر جدا می‌گردند. برخلاف توابع، در ماکروها نیازی به تعریف نوع پارامترها نیست. به عنوان مثال دستور #define READINT(I) scanf("%d",&I) ماکرویی به نام READINT تعریف می‌کند که دارای پارامتر I است. در هنگام فراخوانی این ماکرو (که توسط نام آن انجام می‌شود) باید آرگومانی را ذکر کنیم؛ این آرگومان در عبارت scanf("%d",&I) به جای متغیر I منظور شده و تابع scanf عددی را از ورودی خوانده در متغیری که به عنوان آرگومان ذکر شده است قرار می‌دهد. به عنوان مثال اگر بخواهیم متغیری مثل distance را از ورودی بخوانیم کافی است ماکروی فوق را به صورت زیر فراخوانی کنیم:

```
READINT (distance) ;
```

اگر تعریف ماکرو شامل عبارتی طولانی باشد (به طوری که در یک خط قابل بیان نباشد) می‌توان آن را در دو یا چند خط ادامه داد. بدین منظور باید در انتهای هر خط ناتمام، از کاراکتر \ (back slash) استفاده نمود. دستور زیر را در نظر بگیرید:

```
#define IS_EQU (y)    y%4 == 0&&y % 100 != 0 || \
                      y%400 = 0
```

همانطور که در دستور فوق مشاهده می‌گردد در انتهای اولین خط، از کاراکتر \ استفاده شد تا بتوان بقیه عبارت ماکرو را در خط بعدی تایپ نمود.

اگر ماکروی IS_EQU با پارامتری به نام year فراخوانی گردد، این متغیر به جای y قرار خواهد گرفت؛ یعنی فراخوانی:

```
if (IS_EQU (year))
```

معادل دستور زیر است:

```
if  (year %4 = 0 && year % 100 != 0 || year %400 == 0)
```

در فراخوانی ماکروها باید دقت کافی به خرج داد تا از اجرای آنها نتیجه مطلوبی به دست آید. به عنوان مثال، ماکروی زیر را که عمل توان را انجام می‌دهد در نظر بگیرید:

```
#define  SQUARE (x)   x*x
```

اکنون به فراخوانی ماکروی SQUARE توجه نمایید:

```
s = SQUARE (b)          (1)
```

با این فراخوانی، حاصل b*b در متغیر x قرار می‌گیرد که منظور برنامه‌نویس برآورده می‌شود. اکنون فراخوانی دیگر را در نظر بگیرید:

```
s = SQUARE (x +1)       (2)
```

در اثر این فراخوانی، عبارت x+1*x+1 ارزیابی شده و نتیجهٔ آن در s قرار می‌گیرد که با توجه به تقدم عملگرها (که تقدم عمل ضرب (*) بیشتر از تقدم عمل جمع (+) است)، نتیجهٔ ارزیابی، با توان عدد x+1 برابر نیست. برای رفع این مشکل دو راه حل وجود دارد: **راه حل اول:** x+1 را در یک متغیر قرار داده سپس آن متغیر را به عنوان پارامتر به ماکرو منتقل کرد: **راه حل دوم:** ماکرو را طوری تعریف نمود که از ایجاد این اشتباه جلوگیری شود:

```
# define SQUARE ((x) * (x))
```

اکنون اگر این ماکرو با x+1 فراخوانی گردد، عبارت زیر حاصل می‌شود که نتیجهٔ ارزیابی آن، صحیح است:

```
s = ((x+1) * (x+1))
```

مثال ۳-۲۲

برنامه‌ای که با استفاده از ماکرو، مساحت دایره‌ای را محاسبه می‌نماید.

توضیح

در این برنامه از دو ماکرو استفاده شده است، PI و AREA که AREA دارای یک پارامتر است.

```
#include <stdio.h>
#include <conio.h>
#define PI 3.14
#define AREA(r)  r * r * PI
int main()
{
  float r, s;
  clrscr();
  printf("enter radius:");
```

```
    scanf("%f", &r);
    s = AREA(r);
    printf("area is :%6.2f", s);
    getch();
    return 0;
}
```

خروجی

```
enter radius : 12.4
area is : 482.81
```

ضمیمه کردن فایلها

در زبان C، می‌توان در حین ترجمه برنامه، یک یا چند فایل را به آن ضمیمه کرد. فایلهایی که به‌این طریق به برنامه ضمیمه می‌شوند، فایلهای سرآیند نام دارند. ضمیمه کردن فایلها، توسط دستور پیش‌پردازنده #include انجام می‌شود. این دستور به صورتهای زیر مورد استفاده قرار می‌گیرد:

```
#include  "اسم فایل"
#include  <اسم فایل>
```

فایلهای سرآیند به دو دسته تقسیم می‌شوند:

۱. فایلهایی که همراه کامپایلر C وجود دارند و پسوند همهٔ آنها h. است.
۲. فایلهایی که توسط برنامه‌نویس نوشته می‌شوند.

نحوه کاربرد اولِ دستور #include معمولاً برای ضمیمه فایلهایی استفاده می‌شود که توسط برنامه‌نویس نوشته شده‌اند و روش دوم، برای ضمیمه فایلهایی به کار گرفته می‌شود که همراه کامپایلر C وجود دارند و معمولاً در فهرستی به نام include قرار دارند.

فایلهای سرآیند از اهمیت ویژه‌ای برخوردارند زیرا:

۱. بسیاری از توابع مثل ()getchar و ()putchar در فایلهای سرآیند مربوط به سیستم، به صورت ماکرو تعریف شده‌اند.

۲. با فایلهای سرآیند که برنامه‌نویس می‌نویسد، علاوه بر تعریف ماکروها، می‌توان از بسیاری از تعاریف تکراری جلوگیری کرد.

به عنوان مثال، دستور زیر، موجب ضمیمه شدن فایل stdio.h به برنامه می‌گردد:

```
#include   <stdio.h>
```

فرض کنید کار ما طوری است که همواره برنامه‌هایی می‌نویسیم که نیاز به محاسبهٔ مساحت‌های اشکال مختلفی مثل دایره، بیضی و غیره دارد. بهتر است برای سهولت کار، مساحت کلیهٔ اشکال را به صورت ماکرو تعریف کرده در یک فایل header قرار دهیم؛ سپس در ابتدای برنامهٔ موردنظر، آن را توسط دستور #include معرفی کنیم تا به برنامه ضمیمه شود.

مثال ۴-۲۲

ایجاد یک فایل سرآیند به نام ar.h برای محاسبه مساحت اشکال.

```
#define PI  #.14159
#define  AREA_CIRCLE(radius)  (PI * radius * radius)
#define  AREA_SQUARE(length, width)   (length * width)
#define  AREA_TRIANGLE(base, height)   (base * height / 2)
#define  AREA_ELLIPS(radius1, radius2)   (PI * radius1 * radius2)
#define  AREA_TRAPEZOID(heigth, side1, side2)  (heigth*(side1+side2)/2)
```

پس از تشکیل فایل ar.h، هر برنامه‌ای که نیاز به محاسبهٔ مساحت این اشکال هندسی دارد، می‌تواند با ضمیمه کردن این فایل، از ماکروهای تعریف شده در آن استفاده کند.

مثال ۵-۲۲

برنامه‌ای که با استفاده از فایل ar.h مساحت مثلثی را محاسبه می‌کند.

```
#include<stdio.h>
#include <conio.h>
#include "ar.h"
int main()
{
   int base, height;
   float s;
   clrscr();
   base = 10;
   height = 15;
   s = AREA_TRIANGLE(base, height);
   printf("%6.2f",s);
   getch();
   return 0;
}
```

خروجی

the area of triangle is : 75.00

دستورات پیش پردازندهٔ شرطی

در حالت معمولی، دستور if برای تصمیم‌گیری در نقاط مختلف به کار می‌رود. شرط‌هایی که در دستور if ذکر می‌شوند در حین اجرای برنامه ارزشیابی می‌شوند. یعنی اگر شرط ذکر شده در دستور if چه درست باشد یا نادرست، این دستور و کلیه دستورات دیگر که در بلاک if قرار دارند ترجمه می‌شوند؛ ولی در دستورات **پیش‌پردازندهٔ شرطی**، شرطی که در آن ذکر می‌شود در حین ترجمه ارزشیابی می‌شود.

دستورات پیش پردازندهٔ شرطی عبارتنداز :

#if , #else , #elif , #endif , #ifdef , #ifndef

دستور if# به صورت زیر به کار می‌رود:

```
#if عبارت شرطی
  مجموعهٔ دستورات
#endif
```

عبارت شرطی که در دستور if# ذکر می‌شود، عبارتی است که مقدار آن در زمان ترجمه معلوم است. این عبارت در حین ترجمه ارزیابی می‌شود؛ چنانچه دارای ارزش "درستی" باشد مجموعهٔ دستورات موجود در بین if# (ابتدای بلاک) و endif# (انتهای بلاک) ترجمه خواهد شد وگرنه این مجموعه دستورات ترجمه نمی‌گردد.

مثال ۶-۲۲

کاربرد دستور if# در برنامه.

```
#include "stdio.h"
#include <conio.h>
#define MAX 100
int main()
{
   #if MAX > 90
     printf("compiled for array greater than 99\n");
   #endif
   getch();
   return 0;
}
```

خروجی

```
compiled for array greater than 90
```

وقتی برنامهٔ مثال ۶-۲۲ ترجمه می‌گردد شرط MAX > 90 تست می‌شود، چون این شرط برقرار است، دستور بعدی ترجمه شده و آمادهٔ اجرا می‌گردد که نتیجهٔ اجرای آن را نیز مشاهده نمودید.

نحوهٔ عمل else# همانند else در دستور if است (مثال ۷-۲۲).

مثال ۷-۲۲

کاربرد if# و else# در برنامه.

```
#include <stdio.h>
#define MAX 10
int main() {
    #if MAX > 99
     printf("compiled for array...\n");
    #else
     printf("compiled for small array");
    #endif
    return 0;
}
```

خروجی

compiled for small array

در برنامهٔ مثال ۷-۲۲ چون شرط 99 < MAX برقرار نیست، دستور بعد از آن ترجمه نمی‌شود؛ اما دستور بعد از #else ترجمه شده، پس از اجرا نتیجهٔ زیر حاصل می‌گردد:

"compiled for small array"

#elif مشابه دستور else if رفتار می‌کند و به صورت زیر استفاده می‌شود:

```
# if    عبارت شرطی ۱
        مجموعهٔ دستورات ۱
# elif  عبارت شرطی ۲
        مجموعهٔ دستورات ۲
# elif  عبارت شرطی ۳
        مجموعهٔ دستورات ۳
        .
        .
        .
# elif  عبارت شرطیn
        مجموعهٔ دستورات n
# endif
```

همانطور که ملاحظه می‌شود، هر #elif به همراه یک عبارت شرطی است که اگر نتیجهٔ ارزیابی آن ارزش "درستی" داشته باشد مجموعهٔ دستورات مربوط به آن ترجمه شده کنترل به دستور بعد از #endif برمی‌گردد. وگرنه شرط‌های موجود در #elif‌های بعدی مورد بررسی قرار می‌گیرند. اگر هیچکدام از عبارات شرطی ذکر شده در این ساختار دارای ارزش "درستی" نباشند، هیچیک از مجموعهٔ دستورات ۱ تا n ترجمه نخواهد شد.

مثال ۸-۲۲

برنامه‌ای برای نمایش چگونگی استفاده از دستورات پیش پردازنده شرطی.

```
#define US 0
#define ENGLAND 1
#define IRAN g
#define ACTIVE_COUNTRY US
#include <stdio.h>
#include <conio.h>
int main()
{
  clrscr();
  #if ACTIVE_COUNTRY==US
     char currency[]="dolar";
  #elif ACTIVE_COUNTRY==ENDGLAND
     char curreney[]="pound",
  #else
```

```
    char currency[]="rials";
  #endif
  printf("\n currency is: %s", currency);
  getch();
  return 0;
}
```

خروجی

currency is : dolar

استفاده از #ifdef و #ifndef روش دیگری برای پیاده‌سازی ترجمهٔ شرطی دستورات است. (#ifdef یعنی "if defined" و #ifndef یعنی "if not define").

دستور #idef به صورت زیر به کار می‌رود:

```
# ifdef  نام ماکرو
       مجموعهٔ دستورات
# endif
```

اگر ماکرویی که نام آن در جلوی #ifdef ذکر شده است در یک دستور #define تعریف شده باشد، مجموعه دستورات ذکر شده ترجمه خواهند شد وگرنه این دستورات ترجمه نمی‌شوند.

دستور #ifndef که عکس دستور #ifedf عمل می‌کند به صورت زیر استفاده می‌شود:

```
#ifndef  نام ماکرو
       مجموعهٔ دستورات
#endif
```

اگر ماکرویی که نام آن در جلوی #ifndef ذکر شده است در یک دستور #define تعریف نشده باشد مجموعهٔ دستورات ذکر شده، ترجمه می‌گردند وگرنه ترجمه نخواهند شد.

مثال ۹-۲۲

برنامه‌ای که چگونگی استفاده از #ifndef را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#define TEN 10
int main() {
    clrscr();
    #ifdef TEN
      printf("\n TEN defined.");
    #else
      printf("\n TEN not defined.");
    #endif
    #ifndef ALPHA
     printf("\n ALPHA is not defined.");
    #endif
    getch();
    return 0;
}
```

در مثال ۹-۲۲، دستور ifdef# تست می‌کند که آیا ماکروی TEN تعریف شده است یا خیر. چون این ماکرو در این برنامه تعریف شده است، دستور بعد از آن ترجمه شده پس از اجرا نتیجهٔ زیر حاصل می‌شود:

TEN defined.

دستور ifndef # تعریف یا عدم تعریف ماکروی ALPHA را تست می‌کند، چون این ماکرو تعریف نشده است موجب خروجی زیر می‌شود:

ALPHA is not defined.

حذف ماکروی تعریف شده

اگر در برنامه، ماکرویی توسط دستور define# تعریف گردد، دستور undef# می‌تواند از یک نقطهٔ دلخواه برنامه (از نقطه‌ای که این دستور قرار می‌گیرد) به بعد، تعریف ماکرو را بی‌اثر سازد. این دستور به صورت زیر به کار می‌رود.

#undef <نام ماکرو>

تعریف ماکرویی که نام آن در دستور undef # آمده است از جایی که این دستور در برنامه ظاهر می‌گردد به بعد، منتفی می‌شود.

مثال ۱۰-۲۲

برنامه‌ای که طریقه حذف ماکروی تعریف شده را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#define LEN 100
#define WIDTH 100
int main()
{
    char array[LEN][WIDTH];
    clrscr();
    #undef LEN
    #undef WIDTH
    #ifndef LEN
      printf("LEN deleted.");
    #endif
    getch();
    return 0;
}
```

خروجی

LEN deleted.

توجه داشته باشید که چون دستور #undef LEN ، ماکروی LEN را حذف کرد، شرط موجود در برنامه ارزش درستی پیدا کرده پیام فوق را چاپ نمود.

اسامی ماکروهای از پیش تعریف شده

اسامی تعداد ۵ ماکرو که به همراه کامپایلر استاندارد زبان C وجود دارند عبارتند از:

۱. _LINE_ ۲. _FILE_ ۳. _DATE_ ۴. _TIME_ ۵. _STDC_

ماکروی _LINE_ حاوی شماره دستوری از فایل برنامه است که در حال ترجمه شدن می‌باشد.
ماکروی _FILE_ حاوی نام فایلی است که اکنون در حال ترجمه شدن می‌باشد.
ماکروی _DATE_ حاوی رشته‌ای به صورت mm/dd/yy است و حاوی تاریخی است که فایل برنامه، ترجمه شده است (mm برای بیان ماه، dd برای بیان روز و yy برای بیان سال است).
ماکروی _TIME_ حاوی رشته‌ای به صورت hh:mm:ss است و شامل زمانی است که فایل برنامه، ترجمه شده است (hh برای بیان ساعت، mm برای بیان دقیقه و ss برای بیان ثانیه است).
ماکروی _STDC_ مشخص می‌کند که کامپایلر مورد استفاده، استاندارد است یا خیر. اگر محتویات این ماکرو برابر با یک (۱) باشد، کامپایلر استاندارد و در غیر این موارد استاندارد نیست.

دستور پیش پردازندهٔ #line

دستور #line محتویات دو ماکروی _LINE_ و _FILE_ را تغییر می‌دهد. دستور #line به صورت زیر به کار می‌رود:

#line <شماره> "نام فایل"

شماره، یک عدد صحیح مثبت است که مقدار جدید ماکروی _LINE_ را مشخص می‌کند و "نام فایل" که اختیاری نیز هست، محتویات جدید ماکروی _FILE_ را تعیین می‌کند. ماکروی #line برای اشکالزدایی برنامه مفید است.

مثال ۱۱-۲۲

برنامه‌ای که کاربرد _LINE_ را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
#line 100
int main()
{
   clrscr();
   printf("\n printf start at address: %d\n", __LINE__);
   getch();
   return 0;
}
```

خروجی

```
printf start at address : 103
```

در مثال ۱۱-۲۲، مقدار اولیه‌ای که در ماکروی _LINE_ قرار می‌گیرد برابر با ۱۰۰ است. لذا شمارهٔ دستور printf برابر با ۱۰۳ خواهد بود. این مطلب از خروجی برنامه که چاپ محتویات ماکروی _LINE_ است مشخص می‌گردد.

دستور پیش پردازندهٔ #error

دستور #error موجب جلوگیری از ادامهٔ ترجمهٔ برنامه توسط کامپایلر شده، به صورت زیر به کار می‌رود:

#error پیام خطا

پیام خطا، جمله‌ای است که کامپایلر پس از رسیدن به این دستور، آن را به صورت زیر در صفحه‌نمایش ظاهر می‌کند:

Error : نام فایل شماره خط Error directive : پیام خطا

مثال ۱۲-۲۲

برنامه‌ای که کاربرد #error را نشان می‌دهد.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int arr[4] , i ;
    clrscr();
    for (i=0 ; i < 100 ; i++) {
      if ( i > 4 )
         #error enter less than 20 ;
      else
         scanf("%d",&arr[i]) ;
      getch();
      return 0;
}
```

چند نکته برنامه‌نویسی

برنامه خوب برنامه‌ای است که دارای ۳ خصیصه مهم زیر باشد:

۱. از منابع سیستم (دیسک، حافظه و ...) به نحو خوبی استفاده نماید.
۲. تحت هیچ شرایطی با اشکال مواجه نشود.
۳. از کامپیوتری به کامپیوتر دیگر قابل استفاده باشد.

در این ضمیمه سعی می‌گردد مواردی که برای برآوردن ۳ خصیصه فوق مفید هستند ذکر شده و توصیه می‌شود که برنامه‌نویس آنها را به کار گیرد:

۱. برای افزایش و کاهش یک واحد از متغیر، از عملگرهای ++ و -- استفاده شود.
۲. شمارنده‌های حلقه‌های تکرار با کلاس حافظه register معرفی شوند زیرا:
   ۱. این نوع متغیرها در ثبات ذخیره می‌شوند و زمان دسترسی به آنها کم است.
   ۲. در افزایش سرعت اجرای برنامه مؤثر است.
۳. برای دسترسی به عناصر آرایه از اشاره‌گر استفاده شود:

```
int array[10], *p;
  .
  .
  .
p = array ;
for (;;)
{
  a = *p (++) ;
  .
  .
  .
}
```

۴. برنامه به صورت چند تابع نوشته شود و حتی‌الامکان هر تابع از متغیرهای محلی استفاده نماید.

۵. استفاده از #define جهت تعریف مقادیر، کلمات کلیدی و ... که موجب تقویت مساله "قابل حمل بودن" می‌شود.

۶. استفاده از عملگر sizeof جهت تعیین طول نوعهای مختلف در توابعی که با این مساله سروکار دارند (مثل fwrite()، fread() و ...).

۷. توجه لازم به تقدم عملگرها، برای روشن شدن مطلب به دو مجموعه دستورات زیر توجه کنید:

```
y = 10 ;          (۱)
x = y ++ ;
y = 10 ;
x = ++ y ;        (۴)
```

در دستور (۲) مقدار ۱۰ در متغیر x، و در دستورات (۴) مقدار ۱۱ در متغیر x قرار می‌گیرد.

۸. در استفاده از اشاره‌گرها دقت کافی به خرج داد (در این مورد می‌توانید به فصل ۶ مراجعه نمایید).

۹. هر چند که انتخاب اسامی توابع همنام با توابع کتابخانه‌ای امکان‌پذیر است ولی باید از این کار صرفنظر کرد.

۱۰. عناصر آرایه از صفر شروع می‌شوند که در حین مقداردهی به آرایه‌ها و دسترسی به عناصر آن باید توجه لازم را مبذول داشت.

۱۱. الگوی توابع قبل از تابع ()main تعریف شوند.

۱۲. در فراخوانی توابع باید دقت داشت که پارامترها و آرگومانها همنوع باشند.

۱۳. در نگهداری برنامه دقت کافی به خرج داده شود. برای این منظور باید نسخه‌هایی از آن تهیه گردد.

۱۴. در ورود اطلاعات حتی‌الامکان از تابع ()scanf استفاده نگردد. چون یکی از اشکالاتی که این تابع ایجاد می‌کند این‌است‌که کاراکتر انتهای خط را از بافر صفحه‌کلید نمی‌خواند. این امر موجب می‌شود که توابعی مثل ()gets که باید رشته‌ای از ورودی بخوانند، به محض دریافت کاراکتر انتهای خط، به کار خود خاتمه دهند، چون فرض می‌کنند که کاربر به‌جای واردکردن رشته، فقط کلید enter را وارد کرده است.

۱۵. استفاده از volatile در تعریف متغیرهایی که تحت کنترل برنامه نیستند. به عنوان مثال، متغیری که توسط پالس سیستم تغیر می‌کند، از کنترل برنامه خارج است:

```
volatile int clck ;
int time1 ;
time1 = clck ;
if (time1 == clck)
{
   .
   .
   .
}
```

ارتباط با دستگاه‌های جانبی

یکی از ویژگی‌های زبان C این‌است‌که به‌راحتی می‌تواند با دستگاههای جانبی ارتباط برقرار کند، به‌طوری که اطلاعاتی را از پورتی بخواند و اطلاعاتی را در پورتی بنویسد. در این پیوست، توابعی را مطالعه خواهیم کرد که این امکانات را فراهم می‌کنند. یکی از ضروریات کارکردن با این توابع این‌است‌که آدرس پورتهای دستگاه خود را بدانید. مثال‌های این فصل بر روی دیسک، در شاخه **P** قرار دارند.

توابع خواندن از پورت

تعدادی از توابع وجود دارند که بایت یا کلماتی را از پورت سخت‌افزاری می‌خوانند. در این بخش به بررسی این توابع می‌پردازیم.

تابع inp()

این تابع یک بایت را از پورت سخت‌افزاری می‌خواند. این تابع به صورت یک ماکرو پیاده‌سازی شده است و در فایل conio.h قرار دارد:

```
int inp(unsigned portid);
```

پارامتر این تابع یک مقدار صحیح بدون علامت است که آدرس پورت را مشخص می‌کند. مثال ۱ را ببینید.

تابع inportb()

این تابع بایتی را از پورت سخت‌افزاری می‌خواند. این تابع به صورت ماکرو پیاده‌سازی شده است و در فایل conio.h قرار دارد.

```
unsigned char inportb(unsigned portid);
```

پارامتر این تابع مقدار صحیح بدون علامت است که آدرس پورت را مشخص می‌کند. مثال ۱ را ببینید.

تابع inpw()

این تابع یک کلمه را از پورت سخت‌افزاری می‌خواند. این تابع در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
unsigned inpw(unsigned portid);
```

پارامتر این تابع شماره پورتی است که اطلاعات باید از آنجا خوانده شود. مثال ۱ را ببینید.

تابع inport()

این تابع یک کلمه را از پورت سخت‌افزاری می‌خواند. این تابع در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
unsigned inport(unsigned portid);
```

پارامتر این تابع شماره پورتی است که اطلاعات باید از آنجا خوانده شود. مثال ۱ را ببینید.

مثال ۱

برنامه‌ای که یک بایت و سپس یک کلمه را از پورت سریال شماره ۰ می‌خواند (خروجی برنامه را ببینید).

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int port = 0;
  int result;
  clrscr();
  result = inp(port);
  printf("The value 0x%X read from port %d\n", result, port);
  result = inportb(port);
  printf("The value 0x%X  read from port %d\n", result, port);
  result = inpw(port);
  printf("The value 0x%X read from port %d\n", result, port);
  result = inport(port);
  printf("The value 0x%X read from port %d\n", result, port);
  getch();
  return 0;
}
```

خروجی

```
The value 0x8 read from port 0
The value 0x8 read from port 0
The value 0x808 read from port 0
The value 0x808 read from port 0
```

توابع نوشتن در پورت

برای نوشتن اطلاعات در پورت سخت‌افزاری از چهار تابع استفاده می‌شود. این توابع را در این بخش بررسی خواهیم کرد.

تابع ()outp

این تابع یک بایت را در پورت سخت‌افزاری می‌نویسد، در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
int outp(unsigned portid, int value);
```

portid شماره پورتی است که مقدار value باید در آن نوشته شود (بایت کم‌ارزش value نوشته می‌شود). این تابع مقدار value را نیز برمی‌گرداند.

تابع ()outpw

این تابع یک کلمه را در پورت سخت‌افزاری می‌نویسد، در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
unsigned outpw(unsigned portid, unsigned value);
```

value مقداری است که باید در پورت portid نوشته شود. value به عنوان نتیجه تابع نیز برگردانده می‌شود.

تابع ()outportb

این تابع یک بایت را در پورت سخت‌افزاری می‌نویسد، در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
void outportb(unsigned portid, unsigned char value);
```

این تابع مقدار value را در پورت portid می‌نویسد.

تابع ()outport

این تابع یک کلمه را در پورت سخت‌افزاری می‌نویسد، در فایل conio.h قرار دارد و به صورت زیر به کار می‌رود:

```
void outport(unsigned portid, unsigned value);
```

این تابع مقدار value را در پورت portid می‌نویسد.

مثال ۲

برنامه‌ای که بایت و کلمه‌ای را در پورت سخت‌افزاری می‌نویسد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int port = 0;
   int value = 'C';
   int value1 = 30231;
   clrscr();
   outp(port, value);
```

```
    printf("Value %d sent to port number %d\n", value, port);
    outportb(port, value);
    printf("Value %d sent to port number %d\n", value, port);
    outport(port, value1);
    printf("Value %d sent to port number %d\n", value1, port);
    outpw(port, value1);
    printf("Value %d sent to port number %d\n", value1, port);
    getch();
    return 0;
}
```

خروجی

Value 67 sent to port number 0
Value 67 sent to port number 0
Value 30231 sent to port number 0
Value 30231 sent to port number 0

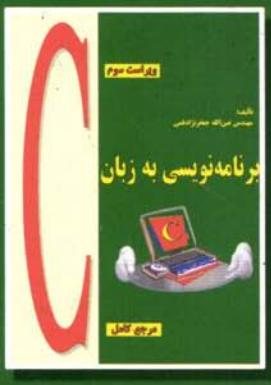
منابع و مأخذ

1. Herbert schildt, C the complete reference(C++, ANCI)
2. Herbert schildt, Turbo C the complete reference
3. Herbert schildt, C power user's guide
4. Kent A. Barclay, C problem solving and programming
5. Robert Lafor, Turbo C programming for IBM
6. Herbert schildt, Advanced Turbo C
7. Abacus, PC system programming
8. Herbert schildt, Teach yourself C
9. Kris Jamsa, Turbo C programmer's Library
10. Eliot B. Koffman & Jeri R. Hanly, problem solving and program design in C
11. Herbert schildt, Advanced C
12. Peter Aitken & Bradley Jones, Teas yourself C in 21 days

ایندکس

علوم رایانه منتشر کرده است:

- آموزش گام‌به‌گام اینترنت
- برنامه‌نویسی به زبان جاوا
- مهندسی نرم‌افزار
- اصول طراحی کامپایلرهای نوین
- اصول طراحی سیستم‌های عامل
- آموزش گام‌به‌گام اکسل
- تحلیل و طراحی سیستم‌ها
- ساخت صفحات وب
- زبان‌های برنامه‌سازی
- ویندوز ۹۸
- شبکه‌های کامپیوتری
- آموزش گام‌به‌گام ویژوال‌بیسیک
- برنامه‌نویسی به زبان دلفی
- کارور رایانه درجه ۱ و ۲
- مبانی کامپیوتر و الگوریتم‌ها
- اصول طراحی مدارهای منطقی
- برنامه‌نویسی به زبان پاسکال
- پرسش‌های چهارگزینه‌ای پاسکال
- برنامه‌نویسی به زبان C++
- راهنمای جامع HTML
- برنامه‌نویسی به زبان ویژوال C++
- رهیافت حل مسئله در C++
- ویندوزXP (مرجع کامل)
- خودآموز ویندوز XP تصویری
- سیستمهای خبره
- جستجو در اینترنت



علوم رایانه
بابل: خیابان مدرس، مجتمع تجاری نیما، واحد ۵۴
تلفن: ۳۲۶۰۷۷۲-۰۱۱۱
jghomim@yahoo.com
شابک: ۵-۲۹-۶۸۶۴-۹۶۴