فهرست

فصل ۳: توابع ورودی و خروجی

فصل ۴: عبارت، دستور، عملگر

فصل ۵: دستورهای کلی

فصل ۶: توابع و کلاس حافظه

فصل ۷: آرایه‌ها

فصل ۸: اشاره‌گرها

فصل ۹: نوعهای تعریف‌شده

فصل ۱۰: فایلها

فصل ۱۱: مطالب تکمیلی

پاسخ خودآزماییها

آزمونهای کلی

پیشگفتار

این کتاب با توجه به سر فصل تعیین شده برای دانشجویان دانشگاه پیام‌نور در رشته‌های ریاضی و آمار تهیه و تنظیم شده است. تلاش کرده‌ایم به طور مناسب، تواناییهای زبان C و کاربردهای مختلف آن را معرفی کنیم. مطالب، ساده و روان و به شکل خودآموزند و در هر قسمت همراه با مثالهای متعدد بیان شده‌اند. این مجموعهٔ آموزشی در ۱۱ فصل تدوین شده است که از مفاهیم اولیهٔ زبان C آغاز و به تدریج مطالب پیچیده‌تر مطرح می‌شود. برای هر فصل، خودآزمایی و پاسخ تشریحی تهیه شده که غالباً به شکل مسائل برنامه‌نویسی مطرح شده‌اند. همچنین در انتهای کتاب تعدادی آزمون جامع چهارگزینه‌ای و نیز شماری مسائل برنامه‌نویسی عنوان شده که در یادگیری بهتر مطالب، قطعاً مؤثر خواهند بود.

استفاده از مفاهیم ریاضی در مثالها و تمرینهای متعدد و نیز بهره‌گیری از جدول و تصویر در قسمتهای مختلف، از ویژگیهای بارز کتاب به شمار می‌رود.

از آنجایی که درس مبانی کامپیوتر و برنامه‌سازی، پیش نیاز این درس است، در قسمتهایی از متن کتاب، مقایسه‌ای بین زبان پاسکال و زبان C صورت گرفته و نکاتی یادآوری می‌شود.

در پایان از آقای مهندس حبیب الهی که در تنظیم و تدوین کتاب همکاری شایانی داشته‌اند و خانم زهرا جلال‌زاده، ویراستار کتاب، سپاسگزاری می‌کنم.

داود کریم‌زادگان مقدم

تابستان ۱۳۸۳

فصل ۱

کلیات و مفاهیم

هدف کلی

آشنایی با مفاهیم اولیه و ویژگیهای کلی زبان C

هدفهای رفتاری

پس از مطالعهٔ این فصل انتظار می‌رود دانشجو بتواند:

۱. تاریخچهٔ پیدایش زبان C را بگوید.

۲. ویژگیهای اصلی این زبان را شرح دهد.

۳. کاراکترهای این زبان را تشخیص دهد.

۴. ویژگی شناسه‌های C را بیان کند.

۵. متغیرها را شناسایی کند.

۶. قسمتهای مختلف برنامهٔ نوشته شده به این زبان را شرح دهد.

مقدمه

در اوایل دههٔ ۱۹۷۰ میلادی، «دنیس ریچی» در آزمایشگاه کمپانی بل زبان C را برای برنامه‌نویسی سیستمها طراحی کرد. این زبان از دو زبان پیشین به نامهای BCPL و B منتج شده است که این دو نیز در همین آزمایشگاه نوشته شدند. زبان C تا سال ۱۹۷۸ منحصر به

استفاده در همین آزمایشگاه بود تا اینکه «ریچی» و «کرنیه» نسخهٔ نهایی این زبان را منتشر کردند. به سرعت مفسرهای متعددی از C تهیه شد، لیکن برای جلوگیری از ناسازگاریهای ایجاد شده و نیز حفظ قابلیت حمل زبان، استاندارد ANSI تعاریف متحدالشکلی مطرح کرد. آنچه در این کتاب مطرح می‌شود بر اساس همین استاندارد است. مفسر خود برنامه‌ای کامپیوتری است که برنامهٔ سطح بالا، دادهٔ ورودی آن و برنامهٔ ایجاد شده به زبان ماشین، خروجی آن را تشکیل می‌دهد.

به طور کلی ویژگیهای مهم زبان C به اختصار به شرح زیر است:

- زبان C به طور گسترده‌ای در دسترس است. مفسرهای تجارتی آن در بیشتر کامپیوترهای شخصی، مینی‌کامپیوترها و نیز در کامپیوترهای بزرگ قابل استفاده‌اند.
- C زبانی است همه منظوره، ساخت‌یافته سطح بالا (مانند زبان پاسکال و فورترن) و انعطاف‌پذیر که برخی از خصوصیات زبانهای سطح پایین را نیز که معمولاً در اسمبلی یا زبان ماشین موجود است داراست. در عین حال این زبان برای کاربردهای ویژه طراحی نشده و می‌توان از آن در همهٔ زمینه‌ها، بخصوص به دلیل نزدیکی آن به زبان ماشین در برنامه‌نویسی سیستم، استفاده کرد. بنابراین C بین زبانهای سطح بالا و سطح پایین قرار دارد و در نتیجه اجازه می‌دهد که برنامه‌نویس خصوصیات هر دو گروه زبان را به کار برد. از این رو در بسیاری از کاربردهای مهندسی به طور انحصاری زبان C به کار می‌برند. (زبانهای سطح بالا، دستورالعملهایی شبیه زبان انسان و پردازش فکری او دارند، همچنین یک دستورالعمل زبان سطح بالا معادل چند دستورالعمل به زبان ماشین است.)
- برنامه‌های نوشته شده به زبان C به طور کلی مستقل از ماشین یا نوع کامپیوتر است و تقریباً تحت کنترل هر سیستم عاملی اجرا می‌شود.
- مفسرهای C معمولاً فشرده و کم حجم‌اند و برنامه‌های هدف ایجاد شده با آنها در مقایسه با سایر زبانهای برنامه‌سازی سطح بالا، خیلی کوچک و

کارآمدند.

- برنامه‌های C در مقایسه با سایر زبانهای برنامه‌سازی سطح بالا، به راحتی قابل انتقال‌اند. دلیل آن این است که C خیلی از ویژگیهای وابسته به نوع کامپیوتر را در توابع کتابخانه‌ای خود منظور داشته است. بنابراین هر نسخه از C با مجموعه‌ای از توابع کتابخانه‌ای مخصوص به خود همراه است که براساس ویژگیهای کامپیوتر میزبان مربوط نوشته شده است. این توابع کتابخانه‌ای تا حدودی استاندارد است و معمولاً هر تابع کتابخانه‌ای در نسخه‌های متعدد C به شکل یکسان در دسترس است.
- C روش برنامه‌سازی ماژولار را پشتیبانی می‌کند. همچنین از نظر عملگرها نیز زبانی قوی است که عملگرهای گوناگونی برای دستکاری روی داده‌ها در سطح بیت داراست.
- به طور کلی جامعیت، عمومیت، خوانایی، سادگی، کارآیی و پیمانه‌ای بودن که همگی از مشخصات برنامه‌ای ایده‌آل‌اند در زبان C پیاده‌سازی می‌شوند.

ویژگیهای فوق موجب شده زبان C یکی از قوی‌ترین و محبوب‌ترین زبانهای برنامه‌سازی در دنیا مطرح شود.

کاراکترها

زبان برنامه‌نویسی C مجموعه‌ای خاص از کاراکترها را شناسایی می‌کند. این مجموعه که در حکم مصالح اولیه جهت شکل دادن به اجزای اصلی برنامه‌اند عبارت‌اند از:

- **حروف بزرگ و حروف کوچک.** زبان C برخلاف زبان پاسکال بین حروف بزرگ و کوچک فرق می‌گذارد. مثلاً FOR با for یکسان نیست.
- **ارقام دهدهی.** شامل ۰ تا ۹
- **جای خالی یا Blank**
- **کاراکترهای مخصوص.** شامل ? * / \ = - + ! @ # $ % ^ ( ) ; ~ [ ] { } < > . , &
- **کاراکترهای فرمت‌بندی.** که برای بیان حالتهای ویژه‌ای به کار می‌روند و عبارت‌اند از

کاراکترHorizontal Tab یا \t، کاراکترVertical Tab یا \v، کاراکتر خط جدید یا \n، کاراکتر برگشت به عقب (Back Space) یا \b، کاراکتر تغذیهٔ فرم (Form Feed) یا \f، کاراکتر ابتدای سطر یا \r، کاراکتر تهی یا \0، و جز آن که آنها را کاراکترهای فرمان نیز می‌نامند. لازم به توضیح است که هر کاراکتر فرمان بیان‌کنندهٔ تنها یک کاراکتر است هر چند که با دو یا چند کاراکتر نوشته می‌شود.

شناسه‌ها

شناسه‌ها نشانه‌های سمبولیکی‌اند که برای مراجعه به انواع داده‌ها مانند مقادیر ثابت، متغیرها، نوعها و توابع به کار می‌روند. به عبارتی دیگر شناسه نامی است که به عناصر مختلف برنامه مانند متغیرها، توابع، آرایه‌ها، اشاره‌گرها و جز آن اختصاص می‌یابد. یک شناسهٔ C دنباله‌ای است از حروف، ارقام یا علامت زیرخط که با هر ترتیبی می‌توانند قرار گیرند. اما اولین کاراکتر باید حرفی باشد. در شناسه‌ها هر دو شکل حروف بزرگ و کوچک مجاز شناخته می‌شوند. برحسب قرارداد شناسه‌هایی که با علامت زیرخط شروع می‌شوند فقط در برنامه‌های سیستم کاربرد دارند و در برنامه‌های کاربردی قابل استفاده نیستند. در مورد تعداد کاراکترها در اسامی از نظر C محدودیتی وجود ندارد گرچه هر مفسر ویژگیها و محدودیتهای خاص خودش را به کار می‌برد. طول اسامی در زبان C استاندارد تا ۳۱ کاراکتر مجاز است.

ـ **مثال ۱ـ۱** اسامی زیر شناسه‌های معتبرند.

```
x1 , sum , payam_noor , maximum
```

ـ **مثال ۱ـ۲** اسامی زیر شناسه‌های غیرمعتبرند.

```
book-5 , 4s , $tax , "p" , number one
```

متغیرها

متغیرها در زبان C شناسه‌هایی‌اند که محلهایی از حافظه را به خود اختصاص می‌دهند. به عبارت دیگر متغیر شناسه‌ای است که در میان بخشی از برنامه برای نسبت دادن نوع تعیین

شده‌ای از اطلاعات مورد استفاده قرار می‌گیرد. در ساده‌ترین حالت هر متغیر جانشین یک قلم داده می‌شود. متغیر ترکیبی است از ارقام، حروف و علامت زیرخط (_). لازم به ذکر است متغیرهایی که با علامت زیرخط شروع می‌شوند برای متغیرهای داخلی سیستم رزرو شده‌اند. طول هر متغیر در استاندارد ANSI تا ۳۱ کاراکتر است، ولی در بعضی از مفسرهای قدیمی محدود به ۸ کاراکتر است. تمامی متغیرهایی که در برنامه‌های C به کار می‌روند باید تعریف یا اعلان شوند. به یک متغیر می‌توان داده‌های مختلفی در محلهای گوناگون برنامه نسبت داد. بنابراین مقدار متغیر در طول اجرای برنامه ممکن است تغییر کند.

بعضی از شناسه‌های زبان C کلمات رزرو شده یا کلیدی‌اند. یعنی معنی و مفهوم آن از قبل در زبان تعریف و پیش‌بینی شده است. بنابراین شناسه‌های تعریف شدهٔ برنامه‌نویس نیستند. متداول‌ترین کلمات کلیدی زبان C در زیر نشان داده شده است.

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

البته در بعضی از مفسرهای زبان C ممکن است کلمات کلیدی دیگری نیز وجود داشته باشد که باید به کتاب راهنمای مربوط مراجعه کرد. همچنین توجه داشته باشید که همهٔ کلمات کلیدی با حروف کوچک نوشته می‌شوند. پس main کلمهٔ کلیدی است درحالی که Main کلمهٔ کلیدی نیست زیرا حرف اول آن بزرگ است. همین طور void کلمهٔ کلیدی است اما VOID کلمهٔ کلیدی نیست.

علامت توضیح

در زبان C هر عبارتی که بین دو علامت /* و */ قرار گیرد صرفاً توضیح محسوب می‌شود. مثلاً اگر بخواهیم در مورد دستوری توضیح دهیم که چه کاری انجام می‌دهد در هر جای برنامه که فضای خالی مجاز باشد می‌توان برای توضیح از علامت فوق استفاده کرد. (در اغلب نسخه‌ها علامت // هم مجاز است.)

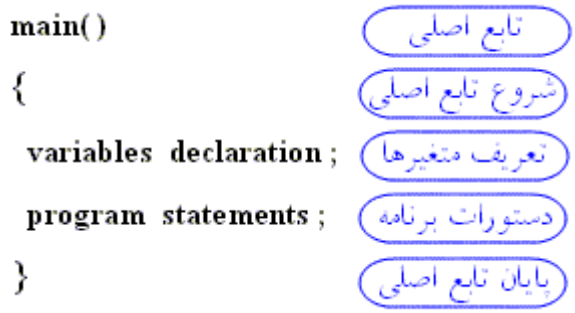
ـ **مثال ۱ـ۳** در برنامهٔ زیر از علامت توضیح استفاده شده است.

```
#include<stdio.h>
```

```
main ()
 {
    int j , k ;
    for (j =1 ; j <= 10 ; j+ +)    /* outer loop */
     {
       printf("%5d ", j) ;
       for (k=1; k<=10; k+ +)    /* inner loop */
          printf("%5d", j * k) ;
       printf("\n") ;
     }
 }
```

ساختار برنامه‌های C

در زبان C برنامه‌ها با فرمت خاصی نوشته می‌شوند. همهٔ برنامه‌های C شامل یک یا چندین تابع‌اند که فقط یکی از آنها تابع اصلی یا main نامیده می‌شود. بنابراین هر برنامه فقط یک تابع اصلی دارد. برنامه همیشه با اجرای تابع اصلی آغاز می‌گردد. معرفی توابع دیگر ممکن است قبل و یا بعد از تابع اصلی قرار گیرد. به طور کلی می‌توان گفت که هر برنامه به زبان C حداقل دارای اجزای مقدماتی به ترتیب زیر است.



دستورالعملهای اجرایی

در هر برنامه دستورالعملهای اجرایی باید بعد از تعریف متغیرها درج شوند. دستوری قابلیت

اجرا دارد که در پایان آن دستور، علامت سمیکولون (;) نوشته شود. برای فهم بهتر این موضوع به مثال زیر توجه کنید.

ـ **مثال ۱ـ۴** برنامه‌ای بنویسید که مساحت مستطیلی به طول ۶ و عرض ۳ را محاسبه و چاپ کند.

```
#include<stdio.h>
main()
{
    int length , width , S ; /* variable declaration */
    length = 6 ;
    width = 3 ;
    S = length * width ;
    printf ("area = %d", S) ;
}
```

پس از اجرای برنامه، خروجی برنامه به صورت زیر نمایش داده می‌شود.

```
area = 18
```

توضیح. برنامه با حروف کوچک تایپ شده است. توضیحات نیز معمولاً با حروف کوچک تایپ می‌شوند. در زبان C حروف بزرگ و کوچک معادل یکدیگر نیستند. خط اول برنامه اعلان می‌کند که کتابخانهٔ مربوط به توابع ورودی و خروجی برای دستیابی به توابع آن آماده شود. در واقع ارجاع به فایلی است که شامل اطلاعاتی است که هنگام تفسیر باید در برنامه قرار گیرند. C یکی از زبانهایی است که به لحاظ داشتن توابع توکار یا از پیش‌فرض شده بسیار غنی است. هر مجموعه از توابع که عملیات ویژه‌ای را انجام می‌دهد در مجموعه‌ای با عنوان کتابخانه یا library قرار می‌گیرد. توابعی که عملیات ورودی و خروجی را انجام می‌دهند در کتابخانه‌ای به نام stdio.h قرار دارند که در آن stdio به معنی standard input output (ورودی و خروجی استاندارد) و h نیز معرف header یا عنوان است. تابع printf نیز یکی از توابع خروجی است.

حال به توضیح #include می‌پردازیم. برنامه‌های نوشته شده به زبان C قبل از اینکه به‌وسیلهٔ مفسر ترجمه شوند در اختیار برنامهٔ دیگری با عنوان پیش‌پردازنده یا preprocessor قرار می‌گیرند. یکی از کاربردهای اصلی این برنامه آن است که کتابخانه‌های مورد نیاز برنامهٔ

منبع را یعنی کتابخانه‌هایی را که توابع به کار رفته در برنامهٔ منبع را شامل است برای استفاده آماده می‌کند. این کار با دستور include که در ابتدای آن علامت # و به دنبال آن نام کتابخانه در داخل علامت <> یا " " می‌آید انجام می‌گیرد. اولین عبارت در برنامهٔ بالا همین کار را برای ما انجام می‌دهد.

در خط بعد تابع اصلی تعریف شده است. پرانتز خالی به دنبال نام تابع بیان می‌کند که این تابع آرگومانی ندارد. چند خط بعدی، متن برنامهٔ اصلی را تشکیل می‌دهند که از پنج دستور ساده تشکیل شده است. پایان هر دستور را علامت سمیکولون (;) مشخص می‌نماید. چون متن برنامه بیش از یک دستور است مجموع آنها که دستور مرکب یا بلاک را تشکیل می‌دهند در داخل یک زوج آکولاد قرار می‌گیرند. در واقع هر آکولاد چپ برای مفسر C به معنی شروع بلاک و هر آکولاد راست معرف پایان آن است. البته در بعضی مواقع برحسب مورد آکولادها را می‌توان به صورت تودرتو نیز به کار برد.

اولین دستور در متن برنامه یا همان شروع آکولاد، توصیف متغیرهاست. سه دستور بعدی دستورات محاسباتی و جایگزینی‌اند و در آخر نیز دستور خروجی است که در آن تابع printf برای چاپ فرمت‌دار است. اولین آرگومان تابع مذکور متن داخل گیومه است که تابع آن را به همان صورت در خروجی چاپ می‌کند. البته بعضی از قسمتهای متن که شامل علامت % باشد مانند %d به مفسر اطلاع می‌دهد که اولین متغیر بعد از بسته شدن گیومه که در این مثال S است مقادیر صحیح می‌پذیرد. در اینجا d معرف decimal است و %d فرمت متغیر در خروجی را تعریف می‌کند. در فصلهای بعدی به طور کامل به بحث فرمت متغیرها خواهیم پرداخت.

ـ **مثال ۱ـ۵** برنامهٔ زیر طول و عرض مستطیلی را از طریق ورودی استاندارد می‌خواند و با فراخواندن تابعی به نام Rectangle مساحت آن را محاسبه می‌کند و سپس طول و عرض و مساحت را در دستگاه استاندارد خروجی نمایش می‌دهد. طول با a و عرض با b و مساحت با area مشخص می‌شود.

```
# include <stdio. h>
main ()
```

```
{
    int a , b , area ;
    int Rectangle (int a , int b) ;
    scanf ("%d %d" , &a , &b) ;
    area = Rectangle (a , b);
    printf ("\n length = %d width = %d area = %d" , a , b , area) ;
}
int Rectangle (int a , int b) ;
{
    int s ;
    s = a * b ;
    return (s) ;
}
```

اگر a = ۵ و b = ۴، خروجی برنامهٔ مذکور به صورت زیر خواهد بود.

length = 5 width = 4 area = 20

توضیح. خط اول ارجاع به فایل کتابخانه‌ای، خط دوم تعریف تابع اصلی، خط سوم شروع تابع اصلی، و خط چهارم اعلان متغیرهاست. در خط پنجم تابع فرعی Rectangle اعلان شده است که مقدار صحیح برمی‌گرداند و آرگومانهای آن نیز a و bاند که مقادیر صحیح‌اند. در خط ششم، تابع ورودی استاندارد scanf به کار رفته است. این تابع که جزء کتابخانهٔ stdio.h است، اطلاعات را از طریق ورودی استاندارد که صفحه کلید است دریافت می‌کند. فرمت و ساز و کار این گونه توابع را در فصلهای بعد بررسی می‌کنیم. در اینجا یادآور می‌شویم که فرم کلی تابع مزبور به صورت زیر است.

scanf (control string , argl , arg2 , … , arg*n*) ;

که در آن رشتهٔ کنترل که در داخل گیومه (" ") می‌آید، اطلاعات مورد نیاز دربارهٔ فرمت اقلام داده‌های ورودی را شامل می‌شود و عناصر arg*n* ..., arg2 , arg1 نیز آرگومانهایی‌اند که اقلام داده‌های ورودی را معرفی می‌کنند. در این دستور کاراکتر "&" عملگر آدرس است. در رشتهٔ کنترلی نیز که در داخل گیومه قرار دارد، از چپ به راست %d اول معرف فرمت اولین آرگومان یا دادهٔ ورودی به عنوان عدد صحیح است و به دنبال آن %d دوم نیز معرف فرمت دومین آرگومان به عنوان عدد صحیح است. علامت \n در رشتهٔ کنترلی تابع printf، موجب انتقال به سطر جدید می‌گردد. بنابراین اطلاعات بعدی در سطر جدید چاپ خواهد شد.

خودآزمایی ۱

۱. کدام یک از اسامی زیر مجاز است به عنوان نام متغیردر برنامه به کار رود؟

Integer , -19, Lesson four , Unit_25, define , Loop2, Star565void , Please?, Payam_noor , C+ +, S#, Five$

۲. برنامهٔ زیر مساحت مربعی به ضلع ۵ سانتی‌متر را محاسبه و چاپ می‌کند. قسمتهای مختلف آن را شرح دهید.

```
#include<stdio.h>
main()
  {
      int x , S ;
      x = 5 ;
      S = x * x ;
      printf ("area = %d", S) ;
  }
```

۳. ویژگیهای برنامهٔ ایده‌آل را شرح دهید.

۴. چند نمونه از زبانهای سطح بالا، سطح پایین و سطح میانی را نام ببرید.

فصل ۲

انواع داده‌ها

هدف کلی

آشنایی با انواع داده‌های زبان برنامه‌نویسی C و کاربردها و شیوه‌های معرفی آنها

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از خواندن این فصل:

۱. انواع داده‌های زبان C را نام برد.

۲. مقادیر متغیر و ثابت داده‌ها را بشناسد.

۳. انواع مقادیر ثابت را نام برد و تشریح کند.

۴. داده‌های اسکالر و مجموعه‌ای را بشناسد.

۵. چگونگی اعلان متغیرها در زبان C را بداند.

۶. ویژگیهای داده‌های صحیح و شیوه‌های معرفی آنها را بداند.

۷. ویژگیهای مقادیر ثابت صحیح بر مبنای ۸، ۱۰ و ۱۶ و چگونگی معرفی آنها را بداند.

۸. در داده‌های‌اعشاری، روشهای‌نوشتن‌ثابتهای‌با ممیزشناور را شرح‌دهدوکاربرد هریک‌را بداند.

۹. در داده‌های کاراکتری کد اسکی و ebedic را تعریف کند.

۱۰. تفاوت بین عدد و کاراکتر را در زبان C بداند.

۱۱. طریقهٔ شناساندن ثابتهای حرفی به مفسر را بداند.

۱۲. طریقهٔ شناسایی حروف کوچک و بزرگ را در کد اسکی بداند.

۱۳. رشته یا ثابت رشته‌ای و طریقهٔ معرفی آن را به مفسر بداند.

۱۴. تفاوت ثابت حرفی و ثابت رشته‌ای تک‌حرفی را بداند.
۱۵. مقداردهی اولیهٔ متغیرها را بداند.
۱۶. وظیفهٔ عملگر cast را شرح دهد.
۱۷. داده‌های تهی و void را بشناسد.
۱۸. پیش‌پردازنده و شیوهٔ معرفی آن را بشناسد.
۱۹. وظیفهٔ فرمان include# و چگونگی تعریف آن را بداند.
۲۰. وظیفهٔ فرمان define# و فواید آن را بداند.

مقدمه

دسته‌بندی داده‌ها به انواع مختلف، یکی از تواناییهای جدید زبانهای برنامه‌نویسی است. زبان C مجموعهٔ کاملی از انواع داده‌ها را پشتیبانی می‌کند که عبارت‌اند از:

- داده‌های صحیح (integer)
- داده‌های اعشاری (floating point)
- داده‌های کاراکتری (character).

همچنین داده‌ها در زبانهای برنامه‌نویسی به صورت مقادیر ثابت و مقادیر متغیر به کار می‌روند. متغیرها در طول اجرای برنامه، مقادیر مختلفی از داده‌ها را می‌پذیرند. اما مقادیر ثابت مقادیری‌اند که در طول برنامه تغییر نمی‌کنند. در زبان C چهار نوع ثابت وجود دارد که عبارت‌اند از ثابتهای صحیح، ثابتهای با ممیز شناور، ثابتهای کاراکتری و ثابتهای رشته‌ای.

مقادیر صحیح ثابت را می‌توان علاوه بر روش معمول دهدهی، در مبناهای هشت و شانزده نیز نوشت. مجموعه داده‌های از نوع صحیح و اعشاری را داده‌های از نوع محاسباتی یا arithmetic می‌نامند. دو نوع دیگر از داده‌ها، نوع اشاره‌گر یا pointer و نوع شمارشی یا enumerated است، که همراه با نوع محاسباتی، داده‌های نوع اسکالر نامیده می‌شود. این نوع داده‌ها را از این لحاظ اسکالر می‌نامند که قابل مقایسه یا قابل سنجش با همنوع خودند. علاوه بر داده‌هایی از نوع اسکالر، داده‌هایی از نوع مجموعه‌ای وجود دارند از جمله آرایه، رکورد، ساختار و اجتماع که در سازماندهی متغیرهایی مفیدند که به طور منطقی به یکدیگر مرتبط‌اند. این نوع داده‌ها را نیز در فصلهای بعدی بررسی می‌کنیم.

اعلان متغیرها

اعلان، گروهی از متغیرها را به نوع داده خاصی مربوط می‌سازد. در زبان C هر متغیر، پیش از آنکه در دستوری از برنامه به کار رود، باید تعریف شود. دستورهای مربوط به تعریف متغیرها، اطلاعات لازم در مورد نوع داده‌هایی را که متغیرهای مورد نظر می‌پذیرند و اینکه چند بایت حافظه اشغال می‌کنند و چگونگی تفسیر آنها را در اختیار کامپایلر قرار می‌دهند. اعلان شامل نوع داده و به همراه آن نام یک یا چند متغیر است و به سمیکولون ختم می‌شود. برای اعلان یا تعریف متغیرهایی از نوع صحیح (integer) کلمهٔ کلیدی int و به دنبال آن اسامی متغیرهای مورد نظر را که با کاما از یکدیگر تفکیک می‌شوند می‌نویسیم.

```
int a , b , c ;
```

البته می‌توان هریک از متغیرها را در دستوری جداگانه و یا در سطری جداگانه معرفی کرد.

```
int a ;
int b ; int c ;
```

که در سطر اول یک متغیر اعلان شده و در سطر دوم با دو دستور جداگانه متغیر دوم و سوم اعلان شده است. واضح است روش اول که در آن هر سه متغیر در یک سطر و با یک دستور اعلان شده ساده‌تر است. کلمات کلیدی برای اعلان داده‌هایی از نوع اسکالر در جدول ۲ـ۱ نشان داده شده است.

جدول ۲ـ۱ کلمات کلیدی در اعلان متغیرها

اصلی	اصلاح/توصیف‌کننده
int float char double enum	Short long signed unsigned

پنج کلمهٔ ستون اول نوع اصلی یا پایه‌ای‌اند. چهار کلمهٔ ستون دوم را اصلاح‌کننده[1] یا توصیف‌کننده[2] نامند که به طریقی پنج نوع اصلی را توصیف می‌کنند. به عبارت دیگر می‌توان

1. modifier
2. qualifier

پنج نوع اصلی را اسم و چهار نوع توصیف‌کننده را صفت برای آن اسامی تصور کرد.

هرگونه اعلان متغیرها در داخل بلاک باید قبل از اولین دستور اجرایی قرار گیرد. اما ترتیب اعلان آنها فرق نمی‌کند. برای مثال دو روش اعلان زیر از نظر نتیجه یکسان‌اند.

روش اول	روش دوم
float x , y; float z ; int a ; int b ;	int a , b ; float x , y , z ;

داده‌های صحیح

زبان C از لحاظ بزرگی عناصر و همچنین از نظر نمایش داخلی آنها استاندارد ویژه‌ای به کار نمی‌برد. به طور کلی اعداد صحیح مثبت، منفی و صفر و نیز متغیرهایی که مقادیر صحیح را می‌پذیرند، ۱۶ یا ۳۲ بیت حافظه اشغال می‌کنند. فرم اولیهٔ داده‌هایی از نوع صحیح، یا همان int مقدار صحیح در نظر گرفته می‌شود. اما اندازه یا بزرگی آن برحسب نوع ماشین و کامپایلر فرق می‌کند.

در هنگام تعریف متغیرهای از نوع int توصیف‌کننده‌های short، long، signed، unsigned و یا ترکیبی از آنها نیز ممکن است به کار رود.

داده‌هایی که با این کلمات توصیف می‌شوند، ممکن است از کامپایلری به کامپایلر دیگر تفسیر متفاوت داشته باشند، ولی اساس آنها یکسان است. اگر مقادیر صحیح در کامپایلری در حالت عادی ۲ بایت باشد، بین short int و int فرقی نخواهد بود و هر دو ۱٦ بیت یا ۲ بایت حافظه اشغال می‌کنند. در ضمن short int را می‌توان فقط به صورت short نیز به کار برد (یعنی پیش فرض آن است که short همان short int است). در چنین حالتی long int نیز ٤ بایت حافظه اشغال خواهد کرد که آن را هم می‌توان فقط به صورت long به کار برد (یعنی در اینجا نیز پیش فرض آن است که long همان long int است). ولی چنانچه مقادیر صحیح در حالت عادی ٤ بایت حافظه اشغال کنند، short int یا فقط short ۲ بایت حافظه به کار خواهد برد. اما بین long int و int (یا فقط long) تفاوتی وجود نخواهد داشت و هر دو ٤ بایت حافظه اشغال خواهند کرد.

در مواردی متغیرها، فقط دارای مقادیر غیرمنفی خواهند بود. مثلاً متغیری که برای شمارش به کار می‌رود، یکی از این موارد و همیشه مثبت است. زبان C اجازه می‌دهد که‌این گونه متغیرها را با به کار بردن توصیف‌کنندهٔ unsigned، بدون علامت اعلان کنیم. یک مقدار صحیح بدون علامت از نظر میزان حافظهٔ اشغالی با مقدار صحیح معمولی فرقی ندارد. تفاوت میان آنها در بیت سمت چپ است که بیت علامت نامیده می‌شود و در مورد مقادیر صحیح بدون علامت، این بیت نیز مثل سایر بیتها برای نمایش مقدار عدد به کار می‌رود و در نتیجه مقادیر صحیح بدون علامت همیشه غیرمنفی است و بزرگی آن ممکن است تقریباً تا دو برابر مقدار صحیح معمولی باشد. برای مثال عدد صحیح معمولی از ۳۲۷۶۸- تا ۳۲۷۶۷+ (در مورد مقادیر صحیح دو بایتی) تغییر می‌کند، بنابراین مقدار صحیح بدون علامت از صفر تا ۶۵۵۳۵ تغییر خواهد کرد.

در استاندارد ریچی توصیف‌کنندهٔ signed وجود ندارد. ولی استاندارد ANSI آن را پشتیبانی می‌کند. در اغلب موارد به صورت پیش‌فرض، متغیرها signedاند. بنابراین، به استفاده از توصیف‌کنندهٔ signed در آنها نیازی نخواهد بود. در اغلب مفسرها پیش‌فرض برای داده‌های کاراکتری signed char است.

متغیرهایی که معرف اعداد صحیح‌اند به صورتهای زیر توصیف می‌شوند.

```
short int
int
unsigned int
signed int
long int
unsigned long int
unsigned short int
```

ـ **مثال ۲ـ۱** در زیر نمونه‌هایی از نحوهٔ معرفی متغیرهایی از نوع مقادیر صحیح نمایش داده شده است.

```
1) long int temp , Pnoor ;
2) short int y1 , y2 , y3 ;
3) unsigned int m , n ;
4) unsigned long sum , average ;
5) unsigned short pt , qt ;
```

تعریف متغیرها از نوع long int و long هم‌ارز است، بنابراین مثال ۱ را می‌توان به صورت

long temp , Pnoor ; نوشت. همچنین short int و short نیز معادل هم‌اند. پس مثال ۲ را می‌توان به صورت short y1 , y2 , y3 ; نوشت. مثال ۳ را نیز می‌توان این طور نوشت: unsigned m , n ;.

مقادیر ثابت صحیح

در زبان C یکی دیگر از انواع داده‌ها، مقادیر ثابت صحیح است. یک مقدار ثابت صحیح عدد و یا دنباله‌ای از ارقام است که در مبنای ۸، ۱۰ و یا ۱۶ تعریف شده باشد. اعداد زیر نمونه‌هایی از اعداد با مقادیر ثابت صحیح در مبنای ۱۰ اند.

36925 , 9999 , +835 , 512 , 0

در C به طور پیش فرض اعداد صحیح در مبنای ۱۰ تعریف شده‌اند. اما مبناهای ۸ و ۱۶ نیز کاربرد زیادی دارند، زیرا ۸ و ۱۶ توانهایی از مبنای ۲ اند و این گونه سیستمهای عددنویسی برای کامپیوترها مناسب‌تر است. برای مثال عدد ۶۵۵۳۶ در یک ماشین ۱۶ بیت همان عدد ۱۰۰۰۰ در مبنای ۱۶ است.

حال ببینیم کامپیوتر چگونه تشخیص می‌دهد که عددی در مبنای ۸ یا ۱۰ یا ۱۶ تعریف شده؟ برای مشخص ساختن آن از پیشوندهای 0 برای مبنای ۸ و 0x برای مبنای ۱۶ استفاده می‌شود. مبنای ۱۰ هم که پیش فرض است و پیشوند ندارد.بنابراین در مورد اعداد

0751 , 0666 +04163 , -0326

صفر سمت چپ به معنای آن است که اعداد مزبور در مبنای ۸ اند، لذا اگر عدد در مبنای ۱۰ باشد، اولین رقم سمت چپ آن نمی‌تواند صفر باشد. بدیهی است در مبنای ۸ فقط هشت نشانهٔ صفر تا ۷ در ارقام به کار می‌روند. همچنین در مبنای ۱۶ نیز، شانزده نشانهٔ مختلف به کار می‌رود که ده نشانهٔ آن همان نشانه‌های متداول در مبنای ۱۰ یعنی صفر تا ۹ است و شش نشانهٔ دیگر حروف A , B , C , D , E , F است که به ترتیب معادل 10 , 11 , 12 , 13 , 14 , 15 در مبنای ۱۰ اند. مثالهای زیر نمونه‌ای از اعداد مبنای ۱۶ اند.

0xF1E6 , 0x5AB , 0x327 , 0x99

اگر طول هر کلمه در ماشین مورد نظر ۱۶ بیت باشد، طول آن از 32k- تا 32k+ یعنی از 32768- تا 32767+ تغییر خواهد کرد که معادل 2^{15}-1 و یا معادل 077777 مبنای ۸ و یا 7FF

مبنای ۱۶ است. ولی اگر طول هر کلمه ۳۲ بیت باشد، طول آن از 2G- تا 2G+ خواهد بود یعنی از 2, 147,483, 648 -2, تا 2, 147, 483,647 که معادل 2^{31}-1 است.

مقادیر ثابت صحیح بدون علامت یا unsigned integer constants با قرار دادن u، حرف اول کلمهٔ unsigned، و همین‌طور مقادیر ثابت صحیح طولانی یا long integer constants با قراردادن حرف l، حرف اول کلمه long، در سمت راست آنها مشخص می‌گردد که l و u را می‌توان به هر دو صورت بزرگ یا کوچک نوشت. همچنین اگر عددی هر دو صفت مذکور را داشته باشد (یعنی هم بدون علامت و هم به‌صورت طولانی باشد)، با دو حرف ul (u در سمت چپ، l در سمت راست آن) متمایز می‌شود.

ـ **مثال ۲ـ۲** جدول زیر مثالهایی از انواع مقادیر ثابت صحیح را نشان می‌دهد.

مقدار ثابت صحیح	سیستم عددی
546780u	مبنای 10 (بدون علامت)
1234567l	مبنای 10 (طولانی)
1234567ul	مبنای 10 (بدون علامت و طولانی)
0123456u	مبنای 8 (بدون علامت)
0123456ul	مبنای 8 (بدون علامت و طولانی)
0563214l	مبنای 8 (طولانی)
02677u	مبنای 8 (بدون علامت)
0x12545678l	مبنای 16 (طولانی)
0xABCDEFu	مبنای 16 (بدون علامت)
0xEF1A3Abul	مبنای 16 (بدون علامت و طولانی)

توابع scanf و printf در فرمت مربوط به خواندن و نوشتن مقادیر صحیح در مبنای ۸ و ۱۶ به ترتیب حروف o و x را مشخص‌کنندهٔ فرمت در رشته کنترل فرمت به کار می‌برند.

ـ **مثال ۲ـ۳** برنامهٔ زیر عددی در مبنای ۱۶ (با پیشوند 0x یا بدون آن) را از طریق ترمینال می‌خواند و معادل آن را در مبناهای ۱۰ و ۸ چاپ می‌کند.

```
# include<stdio.h>
main ()
   {
        int n ;
        printf ("Enter a hexadecimal constant:  \n") ;
        scanf ("%x",&n) ;
        printf ("The decimal equivalent of %x is:  %d ", n , n) ;
        printf ("\n The octal equivalent of %x is:  %o\n", n , n) ;
   }
```

داده‌های اعشاری

در زبان C اعداد اعشاری نیز قابل نمایش است. داده‌های صحیح در بسیاری موارد مناسب است. اما برای مقادیر خیلی بزرگ و برای مقادیر کسری کوچک که در اغلب زمینه‌های علمی کاربرد دارد، به داده‌های از نوع ممیز شناور یا floating point نیاز است. برای نوشتن ثابتهای با ممیز شناور دو روش به کار می‌رود.

روش اول که ساده‌ترین راه است، آن است که از علامت ممیز (که در انگلیسی "." است) استفاده کنیم. مثالهای زیر از این نوع است.

0.996 , 15.0 , 3.1415 , 7. ,.275

روش دوم که نمایش علمی[1] نیز نامیده می‌شود، روش کوتاه‌نویسی مفیدی است. در این روش هر مقدار شامل دو جزء است: یک قسمت عددی که آن را مانتیس نامند و به‌دنبال آن یک قسمت نما یا توان می‌آید که قسمت مانتیس باید در ۱۰ به توان نما ضرب شود. بین این دو قسمت حرف E یا e (که حرف اول exponent است) به مفهوم نما یا توان به کار می‌رود. برای مثال3E2 به مفهوم $3*10^2$ و 125.7E-3- به مفهوم $125.7*10^{-3}$- است. در واقع یک مقدار ثابت با ممیز شناور، عددی است در مبنای ۱۰ که شامل یک ممیز یا علامت اعشار یعنی "." یا شامل یک نما یا هر دو است؛ مانند مثالهای زیر

182.25 , -3.1415 , 5E-5 , 0.775E-3 0. , 0.0 , 1. , 0.92

البته قسمت نما نمی‌تواند یک عدد کسری باشد، پس 29.5E3.4 درست نیست. در بعضی نسخه‌های C برای اینکه مشخص کنند که مقادیر مورد نظر یک کلمه اشغال کرده است حرف

1. scientific notation

F (یا f) را به آخر آن اضافه می‌کنند، مانند 6.125E5F .

همچنین برای مشخص کردن اینکه مقادیر مورد نظر فضایی به طول دو کلمه را اشغال کرده است، حرف L (یا l) به آخر آن اضافه می‌شود مانند 0.123456789E-25L .

دقت مقادیر ثابت با ممیز شناور ممکن است برحسب نسخه‌های مختلف تغییر کند، ولی همهٔ آنها حداقل ۶ رقم با معنی را می‌پذیرند.

برای اعلان متغیرهایی از نوع floating point از دو کلمهٔ کلیدی "float" و "double" استفاده می‌شود، مانند

```
float a , b , c ;
double x , y , z ;
```

که در آن کلمه "double" به مفهوم دقت مضاعف یا double precesion است و در اغلب ماشینها طول فضایی که برای متغیرهای توصیف شده با آن رزرو می‌شود، دو برابر "float" است. یک متغیر توصیف شده با "float" به‌طور متعارف ۴ بایت در حافظه اشغال می‌کند، پس در "double" این فضا ۸ بایت خواهد شد و نمایش درونی مقادیر floating-point نیز از ویژگیهای معماری سخت‌افزار کامپیوتر است و هنوز کاملاً استاندارد نیست. در مورد میزان دقت float و double نیز باید به مستندات کامپایلر مربوط مراجعه کرد. در برخی کامپایلرها برای اعلان متغیرهای از نوع double نیز می‌توان آنها را به صورت long float تعریف کرد. بنابراین دو روش اعلان زیر معادل‌اند.

```
double a , b , c ;          long float a , b , c ;
```

به هرحال اگر کلمهٔ توصیف‌کنندهٔ long به تنهایی جلوی متغیری به کار رود، آن متغیر به صورت پیش‌فرض از نوع مقادیر صحیح خواهد بود. بنابراین با دستور long a , b , c ; سه متغیر a , b , c از نوع صحیح خواهند بود.

داده‌های کاراکتری

یکی از انواع داده‌هایی که در برنامه‌نویسی استفاده می‌شود داده‌های کاراکتری است. در بسیاری از زبانهای برنامه‌سازی داده‌های عددی و داده‌های کاراکتری با یکدیگر تفاوت دارند.

مثلاً عدد 2 داده‌ای عددی و حرف A داده‌ای کاراکتری است. در عمل هم، کاراکترها به صورت عدد در حافظهٔ کامپیوتر ذخیره می‌شوند، و هر کاراکتر دارای یک کد عددی است. کدهای مختلفی وجود دارد که دو نوع آن عبارت‌اند از اسکی (ascii)[1] یا کد استاندارد آمریکایی برای تبادل اطلاعات و دیگری ebcdic[2] یا کد توسعه‌یافتهٔ bcd که آی‌بی‌ام روی سیستم بزرگ خود به کار می‌برد و بیشتر متداول است. البته در زبان C، کد اسکی متداول است. در همهٔ کدگذاریها برای هر کاراکتر نماد عددی وابسته‌ای وجود دارد که در کدگذاری اسکی به آن ascii code گویند و مقدار آن از صفر تا ۲۵۵ است.

در زبان C تفاوت بین اعداد و کاراکتر ناچیز است. در این زبان یکی از انواع داده‌ها char نامیده می‌شود، اما در حقیقت کاراکتر مقدار صحیح یک بایتی است که هم برای نگهداری اعداد و هم برای نگهداری کاراکترها به کار می‌رود.

ثابتهای حرفی در داخل یک زوج گیومه قرار می‌گیرد. این گیومه‌ها به کامپایلر اعلان می‌کند که کد عددی کاراکتر مورد نظر را به دست آورد. برای مثال در دستورهای

```
char a , b ;
b = 5 ;
a = '5';
```

مقدار a برابر ۵۳، یعنی برابر کد اسکی کاراکتر '5' و مقدار b برابر ۵ خواهد بود.

ـ **مثال ۲ـ۴** برنامهٔ زیر کاراکتری را از ورودی می‌خواند و کد عددی آن را نمایش می‌دهد.

```
#include <stdio.h>
main()
  {
      char ch ;
      scanf ("%c", &ch) ;
      printf (" The numeric code is:  %d \n ", ch) ;
  }
```

همچنین در توابع printf و scanf نماد %c فرمت متغیرهای کاراکتری است مانند %d که برای متغیرهای مقادیر صحیح و %f که برای متغیرهای مقادیر اعشار به کار می‌رود (توابع ورودی و خروجی و فرمت متغیرها را در فصل ۳ بررسی می‌کنیم).

1. American standard code for information interchange
2. Extended binary-coded decimal interchange

در مجموعه کاراکتر اسکی، ترتیب کد کاراکترها براساس ترتیب کاراکترهاست. برای مثال، کد حرف 'A' برابر ٦٥، 'B' برابر ٦٦، ...، و 'Z' برابر ۹۰ است. کد حروف کوچک از ۹۷ تا ۱۲۲ است. با توجه به ارزش کد حروف، تابع زیر حروف بزرگ را به حروف کوچک تبدیل می‌کند (توابع را در فصل ۶ بررسی می‌کنیم).

```
char Up-to-low (ch)
char ch ;
  {
     return ch + 32 ;
  }
```

تابع مذکور به کد اسکی هر کاراکتر دریافتی ۳۲ واحد اضافه می‌کند که درنتیجه حروف بزرگ به حروف کوچک تبدیل می‌شود. مثلاً کد اسکی حرف 'A' (٦٥) ۳۲ واحد از کد اسکی حرف 'a' (۹۷) کوچک‌تر است. مشابه آن می‌توان تابعی برای تبدیل حروف کوچک به بزرگ تعریف کرد که در این حالت باید از کد اسکی حرف مورد نظر ۳۲ واحد کسر گردد تا به حرف بزرگ مشابه خود تبدیل شود.

در سیستم کدگذاری غیر اسکی، مانند ebcdic، تفاوت عددی کد حروف بزرگ و کوچک ۳۲ نیست. بنابراین در چنین حالتی تابع تعریف شدهٔ بالا نتیجهٔ مطلوب را نمی‌دهد. برای جلوگیری از چنین اشتباهی زبان C دارای توابع کتابخانه‌ای به اسامی toupper و tolower است که به ترتیب عمل تبدیل کاراکترها را از بزرگ به کوچک و از کوچک به بزرگ انجام می‌دهند.

ثابتهای رشته‌ای

رشته یا ثابت رشته‌ای از تعدادی کاراکتر متوالی تشکیل می‌شود که بین دو گیومه قرار می‌گیرند. به عبارت دیگر شامل دنباله‌ای از کاراکترهاست که در بین دو گیومه قرار دارند، مانند نمونه‌های زیر.

"university" , "256" , "payam noor" , "1380-02-06" , "five$" , "p4"

همچنین باید توجه داشت که " " نیز رشته‌ای تهی (empty) یا null است.

ـ **مثال ۲ـ۵** ثابت رشته‌ای زیر شامل سه کاراکتر مخصوص است که با escape

sequence متناظرشان نشان داده شده‌اند.

"\t to continue , press the \"RETURN\" KEY\n"

که در آن نشانه‌ها یا کاراکترهای مخصوص عبارت‌اند از

\t horizontal tab،

\" گیومه یا quotation mark دوبل که دو بار به کار رفته است،

\n خط جدید یا new line.

کامپایلر به طور خودکار یک کاراکتر null (\0) در پایان هر ثابت رشته‌ای قرار می‌دهد که آخرین کاراکتر در داخل رشته (قبل از بسته شدن گیومه) خواهد بود. این کاراکتر وقتی که رشته نمایش داده شود رؤیت‌پذیر نیست. به هرحال می‌توان هریک از کاراکترها را در داخل رشته امتحان کرد که آیا کاراکتر null است یا نه. در خیلی موارد مشخص ساختن پایان یک رشته با یک کاراکتر مخصوص مانند کاراکتر null نیاز به تعیین حداکثر طول برای رشته را از بین می‌برد. به عنوان مثال رشتهٔ فوق ۳۸ کاراکتر دارد که شامل پنج فضای خالی و چهار کاراکتر مخصوص است که با escape sequence معرفی شده‌اند و در پایان کاراکتر null است که انتهای رشته را مشخص می‌سازد.

یک ثابت حرفی مانند 'P' با یک ثابت رشته‌ای تک‌حرفی متناظر آن مانند "P" هم‌ارز نیست. همچنین به‌خاطر داشته باشید که در جدول کد اسکی، هر کاراکتر دارای یک مقدار عددی است، ولی یک رشتهٔ تک‌حرفی این‌طور نیست. در واقع یک رشتهٔ تک‌حرفی متشکل از دو کاراکتر است که کاراکتر دوم همان کاراکتر null است که پایان رشته را مشخص می‌سازد.

باز هم توجه داشته باشید که یک رشتهٔ n کاراکتری به آرایه n+۱ عنصری نیاز خواهد داشت، زیرا کاراکتر null نیز به طور خودکار به عنوان کاراکتر پایانی در آن قرار داده خواهد شد. برای مثال اگر رشتهٔ "COMPUTER" در یک آرایهٔ یک بعدی کاراکتری به نام book ذخیره گردد، خانه اول آن، یعنی book[0]، شامل کاراکتر C و خانهٔ آخر، یعنی book[8] شامل کاراکتر null خواهد بود که معرف پایان رشته است.

دربارهٔ رشته‌ها و آرایه‌ها و کاربرد آنها در فصل جداگانه‌ای به طور مشروح بحث خواهیم کرد.

مقداردهی اولیهٔ متغیرها

در صورتی که از پیش مقدار شروع متغیر را بدانیم می‌توانیم به هنگام تعریف، مقدار اولیهٔ مورد نظر را نیز به آن اختصاص دهیم. برای این کار در تعریف متغیرها، به دنبال نام آن، اپراتور جایگزینی '=' را همراه با مقدار اولیه به کار می‌بریم. برای مثال هر یک از روشهای زیر متغیرهای a , b , c را توصیف می‌کنند و به ترتیب مقادیر 25- , 13 , 12 را به آنها اختصاص می‌دهند.

روش اول	روش دوم	روش سوم
int a =12; int b =13 ; int c = -25 ;	int a =12 , b =13 ; int c = -25 ;	int a =12 , b =13 , c = -25 ;

اما در دستور زیر فقط به b مقدار اولیه داده شده است.

```
int a , b = 15 , c ;
```

در این گونه موارد برای جلوگیری از اشتباه بهتر است متغیرهایی را که مقدار اولیه می‌پذیرند جدا از سایر متغیرها توصیف کرد، مانند مثال زیر.

```
int a =12 , b =13 , c =25 ;
int d , e , f ;
```

ـ **مثال ۲ـ٦** چند نمونهٔ دیگر از مقداردهی اولیهٔ متغیرها در زیر نشان داده شده است.

```
int sum = 5 ;
char Str = ' # ' ;
float tmp = 10.2 ;
double p1 = 0.1234E-6 ;
```

←

عملگر cast

می‌توان تبدیل یک نوع به نوع دیگر را به صورت صریح انجام داد. این کار به کمک عملگر cast انجام می‌گیرد. پس ساختار cast نوع دیگر از تبدیل است. برای این کار کافی است نوع جدید داده مورد نظر را در داخل پرانتز مستقیماً جلوی عبارت قرار دهیم؛ برای مثال

```
k = (float)2 ;
```

مقدار صحیح 2 را قبل از اختصاص‌دادن به k به float تبدیل می‌کند و سپس آن را به k

اختصاص می‌دهد. بنابراین، اپراتور cast اپراتور یکانی است؛ یعنی فقط یک اپراند دارد.

در موارد متعددی روش casting خیلی مفید است. برای مثال حالت زیر را درنظر بگیرید.

```
int i =2 , k =3 ;
float h = k / i ;
```

در اینجا مقدار k / i (یعنی 3 / 2) برابر 1.5 خواهد شد. سپس نتیجه به float یعنی 1.0 تبدیل و به h نسبت داده می‌شود. حال می‌خواهیم مقدار 1.5 را که نتیجهٔ واقعی عبارت ریاضی 3/2 است به k و i یا هر دوی آنها را با cast به float تبدیل کنیم، مثلاً

```
(float) k / i ;
```

در اینجا به‌طور صریح k به float تبدیل می‌گردد، پس نتیجه برابر 1.5 خواهد شد. عبارت مزبور را می‌توان به صورت ; (float) k / (float) i یا ; k / (float) i نیز نوشت که نتیجه باز هم 1.5 می‌گردد، یعنی نتیجه سه روش مزبور هم‌ارز است.

از مثالهای بالا نتیجه می‌شود که به کمک casting می‌توان در وسط جمله نوع داده را به نوع دیگری تبدیل کرد. بنابراین اپراتور cast به‌عنوان نوع یا type عمل می‌کند؛ یعنی type conversion است و فرمت آن به‌این طریق است که نوع جدید متغیر یا عبارت مورد نظر جلوی آن متغیر یا عبارت در داخل پرانتز نوشته شود. برای مثال دستور (int)d1+d2 یعنی اول d1 به int تبدیل می‌شود بعد با d2 جمع می‌شود. درحالی که دستور (int)(d1+d2) یعنی نتیجهٔ d1+d2 به int تبدیل می‌شود.

بنابراین فرمت اپراتور cast به صورت زیر است.

```
(data type) expression
```

حال برای آنکه نقش اپراتور cast را بهتر متوجه شوید، به نتیجه و عملکرد دو مجموعه دستورهای زیر توجه کنید.

مثال اول

```
float x ;
printf ("%d\n",(int)x) ;
```

مثال دوم

```
short int x ;
printf ("%s\n",(char)x) ;
```

در مثال اول برای متغیر x که از نوع float اعلان شده است، ٤ بایت حافظه پیش‌بینی می‌شود. ولی در نتیجهٔ اجرای دستور printf سطر دوم، به دلیل دستور (int)x مقدار آن به نوع

int تبدیل می‌گردد و نمایش داده می‌شود. بنابراین، اگر برای مثال محتوای حافظه به‌صورت

0.26 E+7

باشد، مقدار 2600000 نمایش داده خواهد شد.

همین‌طور در مثال دوم برای متغیر x که از نوع short int اعلان شده است، ۲ بایت حافظه پیش‌بینی می‌شود، ولی درنتیجهٔ اجرای دستور printf سطر دوم، به دلیل (char)x مقدار آن به نوع کاراکتر تبدیل می‌گردد و محتوای دو بایت حافظهٔ مزبور به‌صورت یک رشتهٔ دوبایتی نمایش داده می‌شود. به همین دلیل است که در فرمت چاپ مقدار متغیر مزبور، از فرمت "%s" که برای رشته است استفاده شده است. حال اگر برای مثال محتوای حافظهٔ مربوط به متغیر x به صورت 4 3 2 1 باشد، موقع نوشتن به صورت رشتهٔ "4 3 2 1" چاپ می‌شود.

در اینجا به اختصار یادآور می‌شویم که فرمتهای "%d", "%f", "%c", "%s" به ترتیب برای متغیرهای از نوع مقادیر صحیح، اعشار، کاراکتر و رشته به کار می‌روند.

نوع void

داده از نوع void (تهی) در استاندارد ریچی وجود نداشت و بعد به استاندارد ANSI افزوده شد. از این نوع داده هدف مهمی مورد نظر است و آن در مورد معرفی توابعی است که مقداری را برنمی‌گردانند، بلکه فقط عمل خاصی را انجام می‌دهند.

ـ **مثال ۲ـ۷** تابعی به نام FF1 که دارای دو آرگون x و y است به صورت زیر تعریف شده است.

```
void FF1(x , y)
int x , y ;
   {
        --------------
        --------------
        --------------
   }
```

قرار گرفتن void در جلوی نام تابع مزبور به‌این دلیل است که این تابع چیزی را برنمی‌گرداند.

پیش‌پردازنده

پیش‌پردازنده را می‌توان برنامهٔ جداگانه‌ای در نظر گرفت که قبل از کامپایلر واقعی اجرا می‌گردد. هنگامی که برنامه‌ای را کامپایل می‌کنید، پیش‌پردازنده به طور خودکار اجرا می‌گردد. تمام فرامین پیش‌پردازنده با علامت " # " شروع می‌گردند که باید اولین کاراکتر خط باشد. وظیفهٔ اصلی و مهم پیش‌پردازنده آن است که فایل درخواستی را آماده سازد و وارد برنامه کند. برخلاف دستورهای C که به سمیکولون ختم می‌شوند، پایان جمله‌های آن با خط جدید مشخص می‌گردد. دو دستور متداول از پیش‌پردازنده‌ها را در ادامه بررسی می‌کنیم.

فرمان #include

فرمان #include موجب می‌گردد که کامپایلر همزمان با فایلی که ترجمه می‌کند، یک متن را نیز از فایل دیگر بخواند. این عمل شما را قادر می‌سازد که بتوانید قبل از شروع ترجمه، محتوای یک فایل را در فایل دیگر بریزید (درج کنید)، ضمن اینکه فایل اولیه تغییر نمی‌یابد. این عمل به ویژه در حالتی که بیش از یک فایل مبنا، اطلاعاتی یکسان را سهیم شوند و به کار ببرند مفید است. با این کار به جای اینکه اطلاعات مشترک دو فایل را به صورت دوبله در هر دو منظور کنید، آن را فقط در یک فایل قرار می‌دهید، سپس هر موقع آن اطلاعات در فایل دوم مورد نیاز باشد، به طریق مذکور آن را برای استفادهٔ فایل دوم نیز آماده می‌کنید.

تعریف فرمان #include به دو شکل زیر است.

شکل اول	شکل دوم
# include < filename >	# include "filename"

در شکل اول، پیش‌پردازنده فقط محل خاصی را که با عامل مشخص شده است نگاه می‌کند. این محل جایی است که include fileهای سیستم، مانند header files برای کتابخانهٔ زمان اجرا یا Runtime Library نگهداری می‌شوند. اگر شکل دوم به کار رود، پیش‌پردازنده فهرست یا دایرکتوری را که شامل فایل مبناست نگاه می‌کند. اگر فایل include file را در آنجا پیدا نکند، پس از آن مشابه شکل اول، محل خاص مورد نظر را جستجو می‌کند. اسامی include fileها بر حسب قرارداد، به پسوند "h" ختم می‌شوند.

حال ببینیم وقتی که پیش‌پردازنده با فرمان include<stdio.h> # مواجه می‌شود، چه پیش می‌آید؟ پیش‌پردازنده فهرست تعریف شده در سیستم را برای فایلی به نام stdio.h جستجو می‌کند. سپس فرمان include# را با محتوای فایل جایگزین می‌کند.

برای اینکه بدانید فرمان include# چگونه کار می‌کند، فرض کنید که فایلی به نام file1.h دارید که محتوای آن فقط دو دستور زیر است.

```
int st-no ;
char name ;
```

سپس در فایل مبنا، فرمان include# را به صورت زیر به کار می‌برید.

```
#include "file1.h"
main()
   {
         -----------
         -----------
   }
```

حال وقتی که برنامهٔ مزبور را ترجمه می‌کنید، پیش‌پردازنده به جای فرمان include# محتوای فایل مشخص شده را قرار می‌دهد. بنابراین فایل مبنا به صورت زیر درمی‌آید.

```
int st-no ;
char name ;
main()
   {
       ------------
       ------------
   }
```

فرمان define#

همان طور که می‌توان با توصیف یا اعلان متغیر، اسمی را به یک محل از حافظه وابسته کرد و به آن محل با آن نام (که همان متغیر مورد نظر است) مراجعه کرد، به همان طریق می‌توان اسمی را به یک مقدار ثابت وابسته کرد و آن را با همان اسم که ثابت سمبولیکی نامیده می‌شود مشخص و هنگام نیاز به آن مراجعه کرد. این گونه متغیرها را که معمولاً با حروف بزرگ معرفی می‌شوند ثابتهای سمبولیکی یا symbolic constants گویند و این عمل با دستور define# انجام می‌گیرد.

برای مثال با دستور define book 15 # می‌توان در هر جای برنامه به جای 15 از book

استفاده کرد. بنابراین دو دستور زیر هم‌ارزند.

```
k = 12 + 15 ;
k = 12 + book ;
```

هر دو دستور مقدار 27 (27 = 15+ 12) را به متغیر k نسبت می‌دهند.

براساس دستور define مقدارbook در حافظه به صورت مقدار ثابت 15 است که در طول برنامه تغییر نمی‌کند. انتخاب نام برای مقادیر ثابت چند فایدهٔ مهم دارد.

اول آنکه به بعضی مقادیر ثابت می‌توان اسم با معنی اختصاص داد. مثلاً می‌توان عدد معروف «پی» را که تا 4 رقم اعشار معادل 3.1415 است، در آغاز به صورت

```
#define Pi 3.1415
```

تعریف کرد و سپس در سرتاسر برنامه، به جای عدد مزبور در تمام محاسبات وابسته به آن Pi را به کار برد.

دوم آنکه اگر مجبور باشیم در برنامه‌ای یک مقدار ثابت طولانی نامأنوس را چندین بار به کار بریم، ساده‌تر آن خواهد بود که با دستور #define نامی مناسب برای آن انتخاب کنیم و به جای ثابت مزبور از آن نام استفاده کنیم. مثلاً اگر مجبور باشیم همان عدد پی را با 6 رقم اعشار در قسمتهای متعددی از برنامه به کار بریم، این عمل هم پردردسر و هم اشتباه‌زا خواهد بود. لذا بهتر است، مثل حالت قبل، یک اسم برای آن انتخاب کنیم.

سوم آنکه ممکن است طبیعت مقدار ثابت طوری باشد که در زمانهای مختلف تغییر کند، مثل نرخ مالیات، یا اجرت ساعت کار و یا درصدی که به عنوان سود در بانکها به پس‌اندازهای پولی یا سرمایه‌گذاری تعلق می‌گیرد که همیشه ثابت نیست و ممکن است برحسب مقررات، قوانین و سایر شرایط مقدار آن عوض شود. در چنین مواردی باید در سرتاسر برنامه مقدار ثابت موردنظر را عوض کنیم. درحالی که اگر با دستور #define اسمی برای آن انتخاب کرده باشیم کافی است فقط در همان یک دستور تغییر مورد نظر را اعمال کنیم. مثلاً اگر براساس قوانین، نرخ جدید مالیات بر درآمد برابر ۲ درصد باشد، کافی است به آن قبلاً نام TAX را اختصاص داده باشیم و با دستور #define TAX 0.02 مقدار جدید را جایگزین قبلی کنیم و دیگر نیازی نیست که در داخل برنامه تغییراتی انجام دهیم. از مثال بالا مشخص می‌گردد که یک ثابت سمبولیکی نامی است که جایگزین دنباله‌ای از کاراکترها

می‌گردد. کاراکترها ممکن است یک ثابت عددی، یک ثابت کاراکتری، و یک ثابت رشته‌ای باشند. بنابراین یک ثابت سمبولیکی اجازه می‌دهد که در یک برنامه به جای یک مقدار ثابت (عددی، حرفی یا رشته‌ای) یک اسم قرار گیرد. وقتی که برنامه ترجمه می‌گردد، در هر محلی از برنامه که ثابت سمبولیکی قرار گرفته باشد، دنبالهٔ کاراکترهای متناظر آن جایگزین می‌گردد. ثابتهای سمبولیکی معمولاً در آغاز برنامه تعریف می‌گردند و شکل آنها به صورت زیر است.

#define name text

که در آن name معرف نام سمبولیک است که معمولاً با حروف بزرگ نوشته می‌شود. text نیز دنباله‌ای از کاراکترها را که باید به نام سمبولیک اختصاص داده شود معرفی می‌نماید. توجه داشته باشید که text به سمیکولون ختم نمی‌گردد، زیرا تعریف ثابت سمبولیک دستور واقعی C نیست.

ـ **مثال ۲ـ۸** در زیر نمونه‌های دیگری از ثابتهای سمبولیکی نشان داده شده است.

```
#define     Temp    524
#define     Pi      3.1415
#define     true    1
#define     Name    " payam noor "
#define     f(x)    x + 1
```

خصیصهٔ #define که برای تعریف ثابتهای سمبولیک به کار می‌رود یکی از چندین خصیصه‌ای است که در پیش‌پردازنده وجود دارد.

ـ

خودآزمایی ۲

۱. برنامه‌ای بنویسید که مساحت دایره‌ای به شعاع R را محاسبه کند و نمایش دهد.

۲. برنامه‌ای بنویسید که کاراکتری را از ورودی بخواند و کد اسکی آن را چاپ کند.

۳. برنامه‌ای بنویسید که مجموع ۱۰۰ جملهٔ اول سری زیر را محاسبه و چاپ کند.

$y = 1 + 1/2 + 1/3 + 1/4 + \ldots$

٤. برنامه‌ای بنویسید که ضرایب معادله‌های درجه دوم را بخواند، جوابهای آن را به دست آورد و چاپ کند. اگر معادله ریشهٔ حقیقی نداشته باشد، پیغام مناسب نمایش دهد.

۵. خروجی برنامهٔ زیر چیست ؟

```
main()
```

```
{
        unsigned char ch ;
        ch = -12 ;
        printf ("%d" , ch) ;
}
```

٦. خروجی برنامهٔ زیرچیست ؟

```
main ()
  {
        unsigned char c ;
        c = 100 * 4 ;
        printf ("%d" , c) ;
}
```

فصل ۳

توابع ورودی و خروجی

هدف کلی

آشنایی با هشت تابع ورودی و خروجی زبان C

هدفهای رفتاری

انتظار می‌رود پس از مطالعهٔ این فصل دانشجو:

۱. با کاربرد و ویژگیهای تابع خروجی ()printf آشنا شود.

۲. فرامین فرمت را در تابع آرگومان‌دار ()printf بشناسد.

۳. با کاربرد و ویژگیهای تابع ورودی ()scanf آشنا شود.

۴. تفاوت و تشابه توابع ()printf و ()scanf را بداند.

۵. با کاربرد و ویژگیهای تابع ورودی ()getchar آشنا شود.

۶. با کاربرد و ویژگیهای تابع خروجی ()putchar آشنا شود.

۷. با کاربرد و ویژگیهای تابع ورودی ()getche آشنا شود.

۸. با کاربرد و ویژگیهای تابع ورودی ()getch آشنا شود.

۹. با کاربرد و ویژگیهای تابع ورودی ـ خروجی ()gets و ()puts آشنا شود.

مقدمه

زبان C با مجموعه‌ای از توابع کتابخانه‌ای همراه است که تعدادی از آنها تابع ورودی و خروجی‌اند. برخی از این توابع متداول را که اغلب در کتابخانه یا فایل stdio.h قرار دارند در

این فصل بررسی می‌کنیم.

توابع فرمت‌دار scanf و printf امکان انتقال اطلاعات میان کامپیوتر و دستگاههای ورودی و خروجی استاندارد را فراهم می‌کنند. در واقع این توابع فرمت‌دار روی داده‌هایی که به شکل پیش‌فرض برای کامپایلر شناخته شده‌اند عمل می‌کنند. دو تابع getchar وputchar موجب انتقال یک کاراکتر به حافظه و بالعکس می‌گردند. دو تابع gets و puts نیز ورود و خروج رشته‌ها را ساده‌تر می‌سازند. برخی از توابع ورودی و خروجی به آرگومان نیاز ندارند. بعضی از این توابع مقداری را برنمی‌گردانند. برخی به صورت مستقل به کار می‌روند و برخی ممکن است در داخل عبارات و دستورها به کار روند.

تابع printf()

داده‌های خروجی استاندارد در کامپیوتر با استفاده از این تابع نوشته می‌شود. این تابع به تعداد دلخواه آرگومان می‌پذیرد که آرگومان اول دارای مفهوم خاص است و رشتهٔ فرمت[1] یا رشتهٔ کنترل[2] نامیده می‌شود. رشتهٔ کنترل در داخل زوج گیومه قرار می‌گیرد و شامل اطلاعات قالب‌بندی است؛ یعنی، مشخص می‌کند که باید چند قلم آرگومان داده چاپ شود و فرمت آنها چگونه است. فرم کلی این تابع به صورت زیر است.

```
printf ("control string", arguments list) ;
```

یا

```
printf ("control string", arg1, arg2,…, argn) ;
```

رشتهٔ کنترل دربردارندهٔ دو نوع اقلام داده است. نوع اول شامل رشته یا کاراکترهایی است که به همان صورت روی صفحه نمایش داده خواهد شد. نوع دوم شامل فرامین فرمت است که با علامت "%" آغاز می‌گردد و به دنبال آن کد فرمت یا مشخص‌کنندهٔ فرمت می‌آید. فرمان فرمت، قالب یا فرمت و تعداد متغیرها یا آرگومانهایی را که باید چاپ شوند مشخص می‌سازد و به آن type conversion نیز می‌گویند و در آن برای مشخص ساختن فرمت هر آرگومان یک گروه از کاراکترها که با "%" آغاز می‌گردند به کار می‌رود.

برخی مؤلفان کد فرمت را فرامین فرمت نیز می‌گویند. فهرست کد یا فرامین فرمت که

1. format string
2. control string

در فرمان printf به کار می‌روند در جدول ۳ـ۱ نشان داده شده است.

جدول ۳ـ۱ فرامین فرمت در تابع printf ()

کد تبدیل	مفهوم فرمان فرمت
%c	داده به صورت تک‌کاراکتر نمایش داده می‌شود.
%d	داده به صورت عدد صحیح دهدهی علامت‌دار نمایش داده می‌شود.
%i	داده به صورت عدد صحیح علامت‌دار نمایش داده می‌شود.
%f	داده به صورت عدد اعشاری، ممیز شناور بدون نما یا توان نمایش داده می‌شود.
%e	داده به صورت floating-point ولی به فرم نمایی یا علمی نمایش داده می‌شود.
%g	داده به صورت floating-point به فرم %f یا %e (هرکدام کوتاه‌تر باشد) نمایش داده می‌شود.
%u	داده به صورت عدد صحیح دهدهی بدون علامت نمایش داده می‌شود.
%s	داده به صورت رشته نمایش داده می‌شود.
%o	داده به صورت عدد صحیح در مبنای ۸ نمایش داده می‌شود.
%x	داده به صورت عدد صحیح در مبنای ۱۶ و بدون علامت نمایش داده می‌شود.
L	پیشوندی که با %d , %u , %x , %o برای معرفی مقدار صحیح بلند یا طولانی به کار می‌رود (مثل ld).
%p	در مورد داده‌های از نوع اشاره‌گر به کار می‌رود.
%%	علامت % چاپ می‌کند.

ـ **مثال۳ـ۱** در برنامهٔ زیر مقادیر متغیرهایی از نوعهای مختلف در خروجی چاپ می‌شود.

```
#include<stdio.h>
main ()
  {
        int x =31 ;
        float y =148.5 ;
        char z[10] = {"PayamNoor"};
```

```
        printf(" %d %f %s " , x , y , z) ;
    }
```

اولی عدد صحیح دهدهی، دومی از نوع اعشاری و سومی رشته است که به ترتیب با فرمان فرمت %d ,%f ,%s مشخص شده‌اند. خروجی دستور مزبور به صورت زیر خواهد بود.

31 148.5 PayamNoor

در برنامهٔ مذکور به‌علت وجود یک محل خالی بین فرامین فرمت در رشتهٔ کنترل، بین مقادیر چاپ شده نیز فضای خالی ایجاد شده است.

ـ **مثال ۳ـ۲** به‌این برنامه توجه کنید.

```
#include<stdio.h>
main ()
    {
        float x=2.0 , y=3 ;
        printf("%f %f %f %f %d", x , y , x+y , x*y, x) ;
    }
```

خروجی برنامه به صورت زیر خواهد بود.

2.000000 3.000000 5.000000 6.000000 2

در اینجا آرگومانهای اول و دوم تک‌متغیرند. ولی آرگومانهای سوم و چهارم عبارات محاسباتی‌اند که اول مقدار آنها محاسبه می‌گردد و سپس نتیجه براساس کد فرمت مربوط چاپ می‌شود. آرگومان آخر به شکل عدد صحیح چاپ می‌شود.

ـ **مثال ۳ـ۳** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main ()
    {
        double x=50.0 , y=0.25 ;
        printf(" %f %f %f %f\n ", x , y , x*y , x / y) ;
        printf(" %e %e %e %e ", x , y , x*y , x / y) ;
    }
```

هر دو دستور printf دارای آرگومانهای یکسان‌اند ولی یکی با فرمان فرمت %f و در دیگری با فرمان فرمت %e چاپ شده‌اند. همچنین علامت "\n" موجب چاپ خروجی در سطر بعدی می‌گردد.

خروجی این برنامه به صورت زیر خواهد بود.

```
50.000000 0.250000 12.500000 200.000000
5.000000e+01 2.500000e-01 1.250000e+01 2.000000e+02
```

در سطر اول مقادیر x , y , x*y , x/y به فرم استاندارد floating point نشان داده شده، درحالی‌که در سطر دوم همان مقادیر به دلیل استفاده از فرمان فرمت "%e" به فرم نمایش علمی چاپ شده است.

ملاحظه می‌کنید که هر مقدار تا ۶ رقم دقت، پس از نقطهٔ اعشار نمایش داده شده است. به هرحال این تعداد ارقام را می‌توان با قرار دادن میزان دقت در رشتهٔ کنترلی تغییر داد.

در فرمان فرمت می‌توان حداقل پهنای میدان و تعداد ارقام اعشار، یعنی ارقام بعد از ممیز، را نیز مشخص کرد. همچنین می‌توان تعیین کرد که اطلاعات خروجی از سمت چپ میدان چاپ (یعنی فضای تعیین‌شده برای چاپ) تراز گردند (در حالت عادی اطلاعات رشته‌ای از طرف چپ و اطلاعات عددی از سمت راست میدان تراز می‌شوند). حداقل طول میدان را می‌توان با قرار دادن یک عدد صحیح (به عنوان مشخص‌کنندهٔ حداقل فضای لازم) بین علامت % و کد فرمت مشخص کرد. این کار موجب می‌گردد که اگر طول میدان بیشتر از طول مورد نیاز برای اطلاعات خروجی باشد، فضای اضافی خالی باقی بماند. ولی اگر طول رشته یا عدد بزرگ‌تر از طول پیش‌بینی‌شده برای آن باشد، طول پیش‌بینی‌شده نادیده گرفته می‌شود و اطلاعات به طور کامل نمایش می‌یابد. اگر بخواهید در مورد اطلاعات عددی فضای اضافی با صفر پر شود، سمت چپ عدد m رقم "0" را قرار دهید. برای مثال %04d موجب می‌گردد که اگر مقدار چاپ شده از چهار رقم کمتر باشد، سمت چپ آن به تعداد لازم صفر قرار گیرد به طوری که مقدار چاپ شده چهاررقمی باشد.

در مورد مقادیر floating point برای مشخص ساختن تعداد ارقام بعد از ممیز، باید پس از عدد مشخص‌کنندهٔ طول میدان، علامت ممیز "." و پس از آن نیز یک عدد که معرف تعداد ارقام اعشار خواهد بود قرار داد. برای مثال کد فرمت %10.4f عدد را با حداقل ده کاراکتر پهنای میدان و با چهار محل برای ارقام اعشار نمایش خواهد داد.

مثال ۳ـ۴ به برنامهٔ زیر دقت کنید.

```
#include<stdio.h>
main ()
```

```
{
        int x = 12345 ;
        float y =345.125 ;
        printf("3d %5d %8d\n\n", x , x , x) ;
        printf("%3f %10f %13f\n", y , y , y) ;
        printf("%3e %10e %13e\n", y , y , y) ;
        printf("%g %10g %13g \n", x , x , x) ;
        printf("%3g %10g %13g ", x , x , x) ;
}
```

خروجی این برنامه به صورت زیر خواهد بود.

```
12345 12345 12345
345.125000 345.125000 345.125000
3.45125e+02 3.45125e+02 3.45125e+02
345.125 345.125 345.125
345.125 345.125 345.125
```

در سطر اول، خروجی با استفاده از مینیمم پهنای فیلد (به طول سه کاراکتر، پنج کاراکتر و هشت کاراکتر) چاپ شده است. در هر فیلد تمامی عدد به طور کامل نمایش داده شده است، اگرچه طول میدان پیش‌بینی شده کمتر از عدد مورد نظر است (مانند فیلد اول که مینیمم طول پیش‌بینی شده سه کاراکتر است ولی عدد مورد نظر 5 رقمی است).

دومین خروجی در سطر اول با یک محل خالی شروع شده است. این محل خالی به‌علت وجود یک محل خالی بین فرمان فرمت فیلد اول و دوم در رشتهٔ کنترلی است.

سومین خروجی در سطر اول دارای چهار محل خالی یا blank در سمت چپ است که یک محل آن به سبب وجود یک محل خالی بین فرمان فرمت فیلد دوم و سوم در رشتهٔ کنترلی است. سه محل خالی دیگر نیز برای پرکردن حداقل فضای پیش‌بینی شده برای میدان مورد نظر است (که در اینجا هشت کاراکتر برای یک عدد پنج رقمی است).

شرایط مشابهی در سطر دوم و سوم خروجی وجود دارد. فقط باید توجه کرد که کد فرمت در سطر دوم از نوع f و در سطر سوم از نوع e است.

در سطر چهارم و پنجم همان مقادیر با استفاده از کد فرمت g% چاپ شده است. این فرم نتیجه را کوتاه‌تر نشان می‌دهد. فواصل بین مقادیر چاپ شده نیز براساس درنظر گرفتن حداقل پهنای پیش‌بینی شده در رشتهٔ کنترلی برای آرگومانهای مورد نظر است.

می‌توان ماکزیمم تعداد ارقام اعشار در مورد مقادیر floating point یا ماکزیمم تعداد

کاراکتر برای یک رشته را نیز مشخص کرد. مشخصهٔ مورد نظر برای این عمل، precision یا دقت نامیده می‌شود. دقت مورد نظر، یک عدد صحیح بدون علامت است که قبل از آن نیز یک علامت ممیز "." قرار می‌گیرد. اگر علاوه بر precision حداقل طول میدان نیز مشخص شده باشد (که معمولاً نیز همین طور است)، طول میدان قبل از precision قرار می‌گیرد و علامت ممیز "." بین آن دو درج می‌شود و کد فرمت پس از این مجموعه کاراکترها می‌آید.

در مورد مقادیر floating point اگر دقت پیش‌بینی شده برای ارقام اعشار کوچک‌تر از تعداد ارقام اعشار باشد، جزء اعشار گرد می‌شود، به طوری که تعداد ارقام آن با دقت یا precision پیش‌بینی شده مطابقت نماید.

ـ **مثال ۳ـ۵** به‌این برنامه توجه کنید.

```
#include<stdio.h>
main ()
  {
        float x =123.456 ;
        printf("%7f %7.3f %7.1f\n", x , x , x) ;
        printf("%12e %12.5e %12.3e", x , x , x) ;
  }
```

خروجی این برنامه به صورت زیر خواهد بود.

```
123.456000 123.456 123.5
1.234560e+02 1.23456e+02 1.235e+02
```

سطر اول با کد فرمت f ایجاد شده است. در اینجا عدد سوم به دلیل وجود مشخصهٔ دقت که یک رقم اعشار در نظر گرفته شده گرد شده است. همچنین با درنظر گرفتن حداقل طول میدان (هفت کاراکتر) در سمت چپ عدد سوم، دو محل فضای خالی به دلیل حداقل طول میدان و یک محل فضای خالی نیز به دلیل وجود یک محل خالی بین فرمان فرمت فیلدهای دوم و سوم ایجاد شده است.

سطر دوم با کد فرمت نوع e ایجاد شده و دارای مشخصه‌های مشابه سطر اول است. در اینجا هم عدد سوم گرد شده است تا با دقت پیش‌بینی‌شده، یعنی ۳ رقم اعشار، تطابق یابد. همچنین چهار فضای خالی نیز با در نظر گرفتن حداقل طول میدان (دوازده کاراکتر) و وجود یک محل خالی بین فرمان فرمت فیلدهای دوم و سوم در سمت چپ آن ایجاد شده است.

ضرورتی ندارد که مشخصهٔ دقت با حداقل طول میدان توأم به کار رود، بلکه می‌توان مشخصهٔ دقت را بدون ذکر حداقل طول میدان نیز به کار برد. ولی به هرحال نقطهٔ معرف اعشار باید در جلوی آن به کار رود، مانند دستور زیر.

```
printf(" %.4f ", x) ;
```

علاوه بر کاراکترهای تبدیل و پهنای میدان و شاخص دقت، رشته کنترل یک نشانه یا flag نیز دارد که در شکل ظاهری خروجی اثر می‌گذارد. flag بلافاصله بعد از علامت (%) قرار می‌گیرد. بعضی کامپایلرها اجازه می‌دهند که دو flag پشت سر هم درون یک مشخصهٔ تبدیل به کار روند. flagهای متداول در زبان C در جدول ۳ـ۲ نمایش داده شده‌اند.

جدول ۳ـ۲ Flag های متداول

Flag	مفهوم و کاربرد
–	داده‌ها در داخل میدان از سمت چپ تراز می‌شوند. فضاهای خالی لازم برای پرکردن حداقل پهنای میدان نیز از طرف راست یعنی بعد از داده افزوده می‌شود.
+	در مورد داده‌های عددی، علامت آن (+ یا –) نیز در جلوی آن ظاهر می‌گردد. بدون استفاده از این flag فقط در مورد مقادیر منفی علامت آنها در خروجی ظاهر می‌شود.
0	در مورد داده‌های عددی موجب می‌شود که فضای خالی سمت چپ به جای blank با صفر پرشود. البته فقط در مورد داده‌هایی به کار می‌رود که از سمت راست تراز می‌شوند.
‥ blankspace	در مورد هر داده عددی علامت‌دار و مثبت یک blank در جلوی آن ظاهر می‌شود. این نشانه اگر با نشانهٔ (+) به کار رود آن را لغو می‌کند.
# (با نوع تبدیل -0 و -x)	باعث می‌شود که در جلوی داده‌های اکتال 0 و داده‌های هگزادسیمال 0x ظاهر شود.
# (با نوع تبدیل -g, -f, -e)	باعث می‌شود که در تمام اعداد floating-point علامت ممیز "." ظاهر شود، حتی اگر عدد مورد نظر جزء اعشار نداشته باشد. همچنین در مورد نوع تبدیل g از نادیده گرفته شدن صفرهای بدون معنی سمت راست جلوگیری می‌کند.

ـ **مثال ۳ـ۶** کاربرد فلاگها در مورد مقادیر صحیح و اعشاری در برنامهٔ زیر نمایش داده شده است.

```
#include<stdio.h>
main ()
  {
        int x =123 ;
        float y =45.0 , z = -6.7 ;
        printf(": %5d %7.0f %10.1e:  \n", x , y , y) ;
        printf(": %-5d %-7.0f %-10.1e:  \n", x , y , z) ;
        printf(": %+5d %+7.0f %+10.1e:  \n", x , y , z) ;
        printf(": %7.0f %#7.0f %7g %#g: " , y , y , z , z) ;
  }
```

با اجرای برنامه خروجی زیر حاصل می‌شود که در آن علامت ":" آغاز ابتدای میدان و پایان انتهای میدان را نمایش می‌دهد.

```
: 123 45 -6.7+00:
: 123 45 -6.7e+00:
: +123 +45 -6.7e+00:
:+123 +45 -6.7e+00:
: 45 45. -6.7 -6.500000:
```

خط اول خروجی را بدون استفاده از فلاگ نمایش می‌دهد که در آن هر عدد درون میدان مشخص‌شده برای آن، از طرف راست تراز شده است. خط دوم همان اعداد را با استفاده از همان کاراکترهای تبدیل با پیش‌بینی یک فلاگ در هر گروه از رشته فرمت نمایش می‌دهد، و ملاحظه می‌کنید که‌این بار به علت وجود فلاگ "-" همهٔ اعداد از سمت چپ تراز شده‌اند. در خط سوم از فلاگ "+" استفاده شده است. در این حالت اعداد مانند خط اول از سمت راست تراز می‌شوند. اما علامت مربوط نیز (در هر دو حالت مثبت و منفی) در جلوی آنها ظاهر شده است. در خط چهارم ترکیب دو فلاگ "-" و "+" به کار رفته است. در اینجا ضمن اینکه اعداد به دلیل وجود فلاگ "-" از سمت چپ تراز شده‌اند، علامت مربوط نیز به‌علت وجود فلاگ "+" در جلوی آنها ظاهر شده است. در خط پنجم دو مقدار ممیز شناور را یکبار بدون وجود فلاگ و بار دیگر با استفاده از فلاگ "#" نمایش می‌دهد و همان طور که ملاحظه می‌شود نقش این فلاگ آن است که در مورد عدد 45 علامت ممیز را نیز منظور داشته است و در مورد عدد دوم صفرهای بدون ارزش بعد از ممیز را نیز در کد فرمت "g" نمایش می‌دهد.

مثال ۳ـ۷ برنامهٔ زیر چاپ اعداد را در مبنای ۸ و ۱۰ و ۱۶ نمایش می‌دهد.

```
#include<stdio.h>
```

```
main ()
  {
        int x = 1234 , y = 0155 , z = 0xa06b ;
        printf(": %6u %6o %6x:  \n", x , y , z) ;
        printf(": %-6u %-6o %-6x:  \n", x , y , z) ;
        printf(": %#6u %#6o %#6X:  \n" , x , y , z) ;
        printf(": %06u %06o %06X:  ", x , y , z) ;
  }
```

خروجی برنامه به صورت زیر خواهد بود که در اینجا نیز علامت ":" ابتدای میدان و پایان میدان را در هر خط نمایش می‌دهد.

```
: 1234 155 a06b:
:1234 155 a06b:
: 1234 0155 0XA06B:
:001234 000155 00A06B:
```

خط اول بدون استفاده از فلاگ، اعداد را بدون علامت و به ترتیب در مبناهای ۱۰، ۸ و ۱۶ در خروجی نمایش می‌دهد. خط دوم همان داده‌ها را با همان کاراکتر تبدیل و با استفاده از فلاگ "-" نشان می‌دهد که درنتیجه اعداد در فضای پیش‌بینی‌شده برای آنها از سمت چپ تراز شده‌اند. در خط سوم از فلاگ "#" استفاده شده است. این فلاگ موجب می‌گردد که در جلوی اعداد در مبنای ۸ و ۱۶ به‌ترتیب "0" و "0x" ظاهر شود. همچنین به سبب استفاده از حرف بزرگ "X" در کاراکتر تبدیل، حروف موجود در اعداد مبنای ۱۶، در خروجی به صورت حروف بزرگ (یعنی 0XA06B) ظاهر شده‌اند. خط آخر نقش استفاده از فلاگ "0" را نمایش می‌دهد. این فلاگ موجب می‌گردد که سمت چپ اعداد به تعداد لازم با صفر پر شود. در اینجا نیز به علت استفاده از حروف بزرگ "X" در کاراکتر تبدیل، حروف موجود در اعداد مبنای ۱۶، در خروجی به صورت حروف بزرگ ظاهر شده‌اند.

تابع scanf()

در زبان C داده‌های ورودی می‌توانند به کمک تابع کتابخانه‌ای scanf از طریق دستگاه ورودی استاندارد وارد کامپیوتر شوند. تابع scanf نیز تابع فرمت‌دار و مشابه تابع printf است ولی در جهت عکس عمل می‌کند. به کمک این تابع می‌توان داده‌های عددی، کاراکترها، رشته‌ها یا ترکیبی از آنها را وارد کامپیوتر کرد. فرمت این تابع مشابه فرمت تابع printf و فرم کلی آن به

صورت زیر است.

scanf ("control string", arguments list) ;

یا

scanf ("control string", argl , arg2 ,…, arg n) ;

در اینجا نقش رشتهٔ کنترل مشابه تابع printf و شامل اطلاعات قالب‌بندی خاص است. مشابه printf این تابع نیز می‌تواند هر تعداد آرگومان را دارا باشد، که در آن اولین آرگومان رشتهٔ فرمت یا رشتهٔ کنترل است. همچنین این تابع، اغلب همان کد فرمت تابع printf را به کار می‌برد؛ برای مثال کدهای فرمت %d , %f , %c ,%s که به ترتیب برای خواندن داده‌هایی از نوع مقادیر صحیح، اعشاری، کاراکتر و رشته به کار می‌روند. تفاوت مهم بین این دو تابع آن است که در جلوی آرگومانها، اپراتور آدرس یعنی "&" نیز قرار می‌گیرد.

البته اگر بخواهید مقداری را برای متغیر رشته‌ای بخوانید، نیازی به اپراتور "&" نخواهد بود زیرا رشته‌ها در زبان C به صورت آرایه‌ای از نوع کاراکتر معرفی می‌گردند و نام آرایه نیز معرف آدرس آرایه (یعنی آدرس اولین عنصر آن) است.

ـ **مثال ۳ـ۸** برنامهٔ زیر نحوهٔ کاربرد عملگر & را در تابع scanf نشان می‌دهد.

```
#include<stdio.h>
main ()
  {
      int x ;
      char name[6] ;
      scanf("%d" , &x) ;
      scanf("%s", name) ;
      printf("%d %s", x , name) ;
  }
```

دستور scanf اول سیستم را هدایت می‌کند که داده ورودی را به صورت عدد صحیح از طریق ترمینال دریافت کند و این مقدار را در متغیر x ذخیره کند. دستور scanf دوم به دلیل استفاده از آرایه، بدون عملگر & به کار می‌رود و اگر در این برنامه برای متغیر name رشتهٔ "book" را وارد کرده باشیم، خروجی آن کلمهٔ book خواهد بود.

ـ

جدول ۳ـ۳ فرامین یا کاراکترهای فرمت برای داده‌های ورودی را که کاراکترهای تبدیل نیز نامیده می‌شوند نشان می‌دهد.

جدول ۳ـ۳ کاراکترهای فرمت در تابع scanf ()

کد فرمت	شـــــرح
%c	داده ورودی به صورت تک‌کاراکتر تعبیر می‌شود.
%d	داده ورودی به صورت عدد صحیح علامت‌دار (در مبنای ۱۰) تعبیر می‌شود.
%i	داده ورودی به صورت عدد صحیح علامت‌دار تعبیر می‌شود.
%u	داده ورودی به صورت عدد صحیح بدون علامت دهدهی تعبیر می‌شود.
%f , %e, %g	داده ورودی به صورت عدد صحیح اعشاری با ممیز شناور (floating_point) تعبیر می‌شود.
%h	داده ورودی به صورت عدد صحیح کوتاه (short integer) تعبیر می‌شود.
%s	داده به صورت رشته تعبیر می‌شود. ورودی با یک کاراکتر non_white_space آغاز می‌گردد و با اولین کاراکتر white_space خاتمه می‌پذیرد (به پایان رشته به طور خودکار کاراکتر "\0" افزوده خواهد شد).
%0	داده ورودی به صورت عدد صحیح در مبنای ۸ تعبیر می‌شود.
%x , %X	داده ورودی به صورت عدد صحیح در مبنای ۱۶ تعبیر می‌شود.
%p	داده ورودی اشاره‌گر تعبیر می‌شود.

ـ **مثال ۳ـ۹** برنامهٔ زیر یک خط متن حداکثر به طول ۷۹ کاراکتر را می‌خواند و آن را به همان صورت چاپ می‌کند.

```
#include<stadio.h>
main () /* read a line of text */
  {
      char line[80] ;
      int count , k ;
      /* read in the line */
      for (k=0 ; line[k]=getchar ()!='\n' ; + +k)
      count = k ;
      for (k=0 ; k<count ; + +k)
      putchar(line[k]) ;
  }
```

در حلقهٔ for، شمارندهٔ k از صفر شروع می‌شود و مقدار آن در هر تکرار یک واحد افزایش می‌یابد و در هر تکرار یک کاراکتر با تابع getchar از طریق ورودی استاندارد دریافت می‌شود و به line[k] نسبت داده می‌شود و وقتی که کاراکتر خط جدید (یعنی \n) وارد شد، عمل ورود کاراکترهای رشته خاتمه می‌یابد که در این لحظه مقدار k برابر تعداد کاراکترهای واقعی رشته خواهد بود. سپس در حلقهٔ بعدی محتوای آرایهٔ line[ ] که دربردارندهٔ رشتهٔ

دریافت شده است چاپ می‌گردد (دو تابع getchar و putchar دوباره بررسی خواهد شد). راه دیگر برای ورود رشته‌ها به حافظهٔ کامپیوتر استفاده از تابع gets است که در مبحث رشته‌ها بحث می‌کنیم.

برای خواندن رشته‌هایی که در آنها فضای خالی (space یا blank) وجود داشته باشد، می‌توان به طریقی از تابع scanf نیز استفاده کرد. برای این کار می‌توان به جای کاراکتر تبدیل نوع s در رشتهٔ کنترلی، دنباله‌ای از کاراکترها را در داخل کروشه به صورت [...] قرار داد که در این صورت رشتهٔ مورد نظر هریک از کاراکترهای موجود در داخل کروشه ازجمله blank را شامل می‌شود.

با چنین روشی وقتی که برنامه اجرا می‌گردد، تا زمانی که کاراکترهای متوالی خوانده شده از طریق دستگاه ورودی با یکی از کاراکترهای موجود در درون کروشه‌ها یکسان باشد، عمل خواندن رشته‌ها ادامه می‌یابد. فضای خالی نیز در داخل رشته‌ها منظور می‌شود. به محض اینکه کاراکتری خوانده شود که در داخل کروشه‌ها وجود نداشته باشد، عمل خواندن خاتمه می‌پذیرد. درضمن یک کاراکتر null به طور خودکار به پایان رشته افزوده می‌شود.

ـ **مثال ۳ـ۱۰** برنامهٔ زیر کاربرد تابع scanf را برای خواندن رشته‌هایی که شامل حروف بزرگ و فضای خالی است نشان می‌دهد. طول این رشته با درنظر گرفتن کاراکتر پایان رشته ۸۰ کاراکتر خواهد بود.

```
#include<stdio.h>
main ()
  {
     char line[80] ;
     .....
     scanf("%[ ABCDEFGHIJKLMNOPQRSTUVWXYZ ]", line) ;
     .....
  }
```

حال اگر از طریق ورودی، رشتهٔ COMPUTER SCIENCE وارد شود، وقتی که برنامه اجرا می‌گردد، تمامی رشتهٔ مزبور به آرایهٔ line نسبت داده می‌شود. به هرحال اگر یکی از حروف رشتهٔ مزبور به حرف کوچک تایپ شود، ورود رشته در همان کاراکتر خاتمه می‌پذیرد.

مثلاً اگر در مثال بالا p به‌صورت کوچک تایپ شود، فقط سه حرف com به آرایهٔ line نسبت داده می‌شود و عمل خواندن در حرف چهارم (حرف p) خاتمه خواهد یافت.

راه دیگر آن است که به جای اینکه کاراکترهای مجاز در رشتهٔ مورد نظر را در داخل کروشه ذکر کنیم، فقط کاراکترهایی را که مجاز نیستیم در رشته‌ها به کار ببریم مشخص می‌کنیم. برای این کار کافی است کاراکترهای مورد نظر را به دنبال نماد "^" که circumflex نامیده می‌شود، در داخل کروشه قرار دهیم. یعنی در اینجا نقش کاراکترهای کروشه‌ای عکس حالت قبلی است و وجود هرکدام از آنها در داخل یک رشته موجب قطع ورود بقیهٔ کاراکترهای رشته می‌گردد و عمل خواندن رشته خاتمه می‌پذیرد.

اگر کاراکتر داخل کروشه‌ها که بعد از "^" می‌آید، فقط کاراکتر خط جدید "\n" باشد، رشته‌ای که از طریق دستگاه ورودی استاندارد وارد می‌شود هر کاراکتر اسکی به جز کاراکتر خط جدید را شامل می‌شود. بنابراین، کاربر می‌تواند هرچه خواست به‌عنوان کاراکترهای رشته وارد کند و در پایان کلید Enter را فشار دهد. این کلید کاراکتر خط جدید را صادر می‌کند و درنتیجه پایان رشته را اعلام خواهد کرد.

مثال ۳ـ۱۱ فرض کنید که یک برنامهٔ C شامل دستورهای زیر است.

```
#include<stdio.h>
main ()
  {
     char line[80] ;
     .....
     .....
     scanf("%[^\n]", line) ;
     .....
     .....
  }
```

وقتی که تابع scanf در برنامهٔ بالا اجرا می‌گردد، رشته‌ای به طول نامشخص (ولی حداکثر ۷۹ کاراکتر) از طریق دستگاه ورودی استاندارد وارد می‌گردد و به line نسبت داده می‌شود. هیچ گونه محدودیتی در مورد کاراکترهای تشکیل‌دهندهٔ رشته وجود نخواهد داشت، فقط باید در یک خط بگنجد. برای مثال رشتهٔ زیر از طریق صفحه‌کلید وارد و به line نسبت داده می‌شود.

WE LEARN MATHEMATICS.

تابع getchar()

برای خواندن یک کاراکتر از دستگاه ورودی، می‌توان علاوه بر تابع scanf از تابع getchar نیز استفاده کرد. تابع مزبور که جزء کتابخانهٔ I/O زبان استاندارد C است، کاراکتری از دستگاه ورودی استاندارد که معمولاً صفحه‌کلید است می‌خواند. این تابع آرگومان ندارد و به طور متعارف در یک دستور انتساب یا جایگذاری به کار می‌رود و کاراکتر دریافتی از ورودی را به متغیری که در سمت چپ دستور جایگذاری مورد نظر است اختصاص می‌دهد. شکل کلی آن به صورت زیر است.

```
character variable = getchar() ;
متغیر کاراکتری = getchar() ;
```

که در آن متغیر کاراکتری نام متغیری از نوع کاراکتر است که باید از قبل توصیف شده باشد.

ـ **مثال ۳ـ۱۲** به دستورهای زیر توجه کنید.

```
char ch ;
ch = getchar() ;
```

در عبارت اول، متغیر ch از نوع کاراکتر توصیف شده است. وقتی که اجرای برنامه به دستور دوم برسد، برنامه منتظر فشار دادن کلیدی از صفحه‌کلید می‌شود. حال کاراکتر کلید فشار داده شده، به متغیر ch اختصاص می‌یابد. چنانچه متغیر ch از نوع int معرفی گردد، کد اسکی کاراکتر مربوط به کلید فشار داده شده، به آن متغیر اختصاص می‌یابد.

اگر هنگام خواندن کاراکتر با تابع getchar، شرایط پایان فایل پیش آید مقدار سمبولیکی EOF به طور خودکار برگشت داده می‌شود (این مقدار در داخل فایل stdio.h اختصاص می‌یابد. به طور متعارف مقدار ۱- به EOF اختصاص داده می‌شود، اگرچه ممکن است این مقدار از کامپایلری به کامپایلر دیگر فرق کند). ظاهر شدن EOF به‌این طریق، راه ساده‌ای برای تشخیص پایان فایل در هنگام اجرای آن است (در این مورد، در مبحث فایلها بیشتر بحث خواهیم کرد. لذا به هیچ وجه نگران آن نباشید). می‌توان تابع getchar را نیز برای خواندن رشتهٔ چند کاراکتری به صورت حلقهٔ تکرار به کار برد که در هر تکرار یک کاراکتر را بخواند.

ـ

تابع putchar()

این تابع برای شمارش یک کاراکتر روی خروجی استاندارد که معمولاً صفحه نمایش است به کار می‌رود و نقش آن مشابه تابع getchar اما در جهت عکس است. طبیعی است کاراکتری که انتقال می‌یابد به صورت ثابت کاراکتر یا متغیری از نوع کاراکتر که آرگومان تابع مزبور است به کار می‌رود. شکل کلی این تابع به صورت زیر است.

putchar (character variable) ;
putchar (متغیر کاراکتری) ;

البته می‌توان ثابت کاراکتری را نیز به عنوان آرگومان تابع مزبور به کار برد. این تابع با استفاده از آرایه یک رشته را در خروجی چاپ می‌کند.

ـ **مثال ۳ـ۱۳** برنامهٔ زیر به تدریج در هر بار یک کاراکتر می‌خواند و سپس آن را در خروجی چاپ می‌کند.

```
#include<stdio.h>
main ()
  {
      char ch ;
      while (1)
      {
        ch = getchar() ;
        putchar(ch) ;
      }
  }
```

در این برنامه ch کاراکتر اعلان شده است. هر بار که یک کاراکتر از طریق دستگاه ورودی استاندارد خوانده می‌شود، به همان طریق به خروجی انتقال می‌یابد. اما به لحاظ اینکه عبارت مربوط به while، یعنی مقدار داخل پرانتز بعد از while، برابر "1" و همیشه غیرصفر است، براساس قوانین زبان C، این عبارت همیشه درست یا true است. بنابراین ساختار while(1) حلقه‌ای بی‌نهایت است و تنها راه متوقف ساختن برنامه وقفه‌ای است که با کلیدهای control-c عملی خواهد شد.

راه دیگر برای نوشتن برنامهٔ بالا به صورت زیر است.

```
#include<stdio.h>
main ()
  {
     int ch ;
     while ((ch=getchar()) != EOF)
     putchar(ch) ;
  }
```

این برنامه ترکیب دو عمل در یک دستور را در حلقه نشان می‌دهد. گفتیم EOF در برنامهٔ بالا علامت سمبولیک پایان فایل است. آنچه در واقعیت نشانهٔ پایان فایل را نشان می‌دهد تابع سیستم است. برای این کار اغلب عدد ۱- به کار می‌رود، ولی سیستمهای مختلف ممکن است مقادیر متفاوتی داشته باشند. با گنجانیدن فایل stdio.h و به کار بردن ثابت سمبولیکی EOF، برنامه را قابل حمل یا قابل اجرا ساخته‌ایم. یعنی، فایل مبنا روی سیستمهای مختلف بدون تغییر اجرا می‌شود.

ملاحظه می‌کنید که در روش اخیر، متغیر ch به جای char به صورت int معرفی شده است. هرچه برای نشان دادن پایان فایل به کار می‌رود، نمی‌تواند مقداری باشد که یک کاراکتر را معرفی نماید. حال چون C به‌صورت int معرفی شده است، می‌تواند مقادیر تمام کاراکترهای ممکن و همین طور مقدار ویژهٔ EOF را نگهداری کند.

همان طور که گفتیم، هر دو تابع getchar و putchar در فایل stdio.h تعریف شده‌اند و ممکن است در بعضی سیستمها در فایلهای دیگری نیز مانند فایل conio.h تعریف شده باشند.

ـ **مثال ۳ـ۱۴** برنامهٔ زیر یک خط متن را از ورودی با حروف کوچک دریافت و آن را به حروف بزرگ تبدیل می‌کند.

```
#include<stdio.h>
main ()
  {
     char line[80] ;
     int count , k ;
     /* read in the line */
     for (k=0 ; (line[k]=getchar())!='\n' ; + + k) ;
       count = k ;
     /* write out the line in upper-case */
     for(k=0 ; k<count ; + +k)
       putchar(toupper(line[k])) ;
```

```
}
```

در برنامهٔ بالا از حلقهٔ for و تابع کتابخانه‌ای toupper استفاده شده است. نقش این تابع آن است که حروف کوچک را به بزرگ تبدیل می‌کند. بنابراین اگر حروف ورودی هنگام تایپ حروف بزرگ یا ارقام و مشابه آن باشند، به شکل اولیهٔ خود نمایش داده خواهند شد. برای مثال اگر ورودی به صورت Advanced programming باشد، خروجی به صورت ADVANCED PROGRAMMING خواهد بود.

ـ **مثال ۳ـ۱۵** برنامهٔ زیر یک خط از متن را می‌خواند و در آن هر کاراکتر را (به غیر از کاراکتر فضای خالی یا space) به کاراکتر بعدی تبدیل می‌کند و نمایش می‌دهد (درواقع متن را به شکلی به صورت رمز در می‌آورد و نمایش می‌دهد).

```
#include<stdio.h>
#define space ' '
main ()
   {
        char ch ;
        ch = getchar () ; /* read a character from i/o */
        while(ch!='\n') /*while not end of line */
             {
               if (ch= =space) /* leave the space */
                  putchar(ch) ; /* character unchanged */
               else
                  putchar(ch+1) ; /* change other characters */
               ch = getchar() ; /* get next character */
             }
   }
```

برای مثال اگر computer science2 ورودی باشد، خروجی dpnqvufs tdjfodf3 خواهد بود.

با ترکیب دو دستور خواندن و تست کردن پایان متن در یک عبارت، برنامهٔ مزبور را می‌توان به صورت ساده و فشرده‌تر زیر نوشت.

```
#include<stdio.h>
#define space ' '
main ()
   {
        char ch ;
        while ((ch=getchar()) != '\n')
```

```
    {
      if (ch = = space) /* leave the space */
      putchar(ch) ; /* character unchanged */
      else
      putchar(ch+1) ; /* change other characters */
    }
}
```

که در این برنامه دستور while ((ch=getchar()) != '\n') ترکیب دو عمل در یک دستور را نشان می‌دهد که روشی متداول در زبان C است. این دو عمل آن است که اول به کمک تابع getchar مقداری به ch نسبت داده می‌شود و سپس مقدار ch با کاراکتر خط جدید مقایسه می‌شود. وجود پرانتز دور عبارت ch = getchar() آن را اپراند چپ اپراتور =! می‌سازد. اگر آن را حذف کنیم نتیجهٔ مطلوب به دست نمی‌آید زیرا اپراتور =! نسبت به اپراتور = تقدم بالاتری دارد. بنابراین اگر دستور مزبور را به صورت while (ch = getchar()!= '\n') بنویسیم، اول عبارت getchar()!= '\n' ارزیابی می‌شود که عبارتی رابطه‌ای است (بنابراین کاراکتر خط جدید برنمی‌گرداند، بلکه یک مقدار برمی‌گرداند) که ارزش آن یک یا صفر (درست یا نادرست یعنی true یا false) خواهد بود. سپس این مقدار به ch نسبت داده می‌شود که هدف مورد نظر ما را از دستور مزبور تأمین نمی‌کند.

ـ **مثال ۳ـ۱۶** برنامهٔ زیر کاراکترها را از طریق ورودی صفحه‌کلید دریافت می‌کند و آنها را به صفحهٔ نمایش می‌فرستد. این برنامه با دریافت کاراکتر $ از ورودی خاتمه می‌پذیرد.

```
#include<stdio.h>
main ()
  {
     char ch ;
     while ((ch=getchar()) != '$')
     putchar(ch) ;
  }
```

در این برنامه نیز از ترکیب دو دستور در یک دستور در درون حلقهٔ while استفاده شده است.

در زبان C علاوه بر تابع getchar دو تابع getch و getche نیز برای خواندن یک کاراکتر

از ورودی به کار می‌رود که در ادامه بررسی می‌کنیم.

تابع getche()

اگر بخواهیم کاراکتری به کمک تابع scanf یا تابع getchar خوانده شود، باید پس از تایپ کاراکتر مورد نظر، کلید Enter را نیز استفاده کنیم. یعنی، درواقع دو تابع مزبور تا موقعی که کلید برگشت (که به آن carriage return یا به اختصار CR گویند) فشرده نشود ورودی را در بافر نگه می‌دارند. پس از زدن کلید برگشت، دادهٔ تایپ شده در اختیار برنامه قرار می‌گیرد. حسن این روش آن است که اگر کلیدی را اشتباه وارد کرده باشیم، می‌توانیم آن را با backspace تصحیح کنیم. یعنی، قبلی را پاک کنیم و دوباره کاراکتر صحیح مورد نظر را تایپ کنیم. عیب این کار آن است که این عمل در محیط محاوره‌ای امروز وقت‌گیر و دردسرزاست. ازاین رو تابع getche به‌وجود آمد که در آن دیگر نیازی به تحریر کلید برگشت یا CR نیست. اشکال این تابع آن است که اگر کاراکتر اشتباه تحریر شود امکان تصحیح وجود ندارد. همچنین کاراکتر تحریر شده، روی صفحهٔ تصویر نمایش داده می‌شود که‌این عمل echoing نامیده می‌شود. در واقع حرف e در آخر نام تابع getche به مفهوم echo (عکس‌العمل) است.

تابع getch()

این تابع همانند تابع getche است با این تفاوت که کاراکتر تحریر شده در صفحهٔ تصویر ظاهر نمی‌گردد. در مورد هریک از این سه تابع وقتی که کنترل اجرای برنامه به این توابع می‌رسد، برنامه منتظر فشردن کلیدی در صفحه‌کلید می‌شود. اگر متغیر مورد نظر از نوع کاراکتری باشد، یعنی در برنامه با فرمت %c توصیف شده باشد، مقدار کاراکتری کلید فشرده شده به‌این متغیر نسبت داده می‌شود و درصورتی که‌این متغیر از نوع عددی باشد کد اسکی کاراکتر مربوط به کلید فشرده شده در این متغیر قرار می‌گیرد.

توابع puts() و gets()

این دو تابع این امکان را فراهم می‌سازند که بتوان رشته‌هایی از کاراکترها را از طریق کنسول خواند یا در خروجی نوشت (دستگاههای ورودی و خروجی استاندارد را کنسول نامند که در

مورد ریزکامپیوترها معمولاً صفحه‌کلید ورودی استاندارد و مانیتور خروجی استاندارد را تشکیل می‌دهند).

تابع gets() یک رشته از کاراکترها را که از طریق صفحه‌کلید وارد می‌شود، می‌خواند و آنها را در آدرسی قرار می‌دهد که با آرگومانهای آن تعیین شده است و اشاره‌گری کاراکتری است. کاراکترهای رشتهٔ مورد نظر را تایپ می‌کنید و در پایان، کلید Enter را می‌زنید. با این عمل به طور خودکار کاراکتر null یا '\0' نیز در پایان رشته قرار می‌گیرد. در اینجا اگر کاراکتری اشتباه تایپ شود، می‌توان آن را قبل از فشردن کلید Enter با استفاده از کلید backspace تصحیح کرد. در واقع در اینجا نیز کاراکترهای تایپ شده در بافر می‌ماند و تا موقعی که کلید برگشت فشرده نشده است در اختیار برنامه قرار نمی‌گیرد.

تابع puts() آرگومانهای رشته‌ای خود را به صفحه نمایش می‌فرستند و سپس قلم نوشتار به خط جدید انتقال می‌یابد.

ـ **مثال ۳ـ۱۷** برنامهٔ زیر رشته‌ای را از طریق صفحه‌کلید می‌خواند و در آرایهٔ line قرار می‌دهد. سپس آن را روی خروجی نمایش می‌دهد.

```
#include<stdio.h>
main()
  {
     char line[80] ;
     gets(line) ;
     puts(line) ;
  }
```

فراخوانی تابع puts در مقایسه با فراخوانی printf دارای overhead کمتری است و درنتیجه سریع‌تر از آن عمل می‌کند زیرا تابع puts فقط یک رشته از کاراکتر را به خروجی می‌فرستد و نمی‌تواند مشابه printf تبدیل فرمت انجام دهد. همچنین نمی‌تواند مقادیر عددی را به عنوان خروجی داشته باشد. بنابراین چون puts فضای کمتری می‌گیرد و سریع‌تر از printf اجرا می‌گردد، هنگامی که در برنامه‌سازی حالت خیلی بهینه مورد نظر باشد، از این تابع استفاده می‌شود.

خودآزمایی ۳

۱. برنامه‌ای بنویسید که با استفاده از تابع printf و تابع puts رشتهٔ زیر را در دو خط جداگانه چاپ کند.

"Payam noor university"

۲. برنامه‌ای بنویسید که کاراکتری از ورودی بخواند و کاراکتر بعدی آن را در خروجی چاپ کند.

۳. برنامه‌ای بنویسید که عدد صحیح m1 وعدد اعشاری m2، اطلاعات کاراکتری و آدرس متغیرm3 را در خروجی چاپ کند.

٤. خروجی برنامهٔ زیر چیست ؟

```
# include < stdio. h>
main ()
  {
      double x ;
      x = 2e + 004 ;
      printf ("\n x1 = %e" , x) ;
      printf ("\n x2 = %E" , x) ;
      printf ("\n x3 = %g" , x) ;
  }
```

۵. خروجی برنامهٔ زیر چیست ؟

```
# include < stdio. h>
main ()
  {
      float x = 123.456 ;
      printf ("%f %.3f %7.2f" , x , x , x) ;
  }
```

٦. برنامه‌ای بنویسید که سه عدد صحیح را از ورودی بخواند و میانگین آنها را چاپ کند.

۷. برنامه‌ای بنویسیدکه دو متغیر صحیح را از ورودی بخواند و محتویات آنها را بدون استفاده از متغیر کمکی عوض کند و نتیجه را در خروجی نمایش دهد.

۸. برنامه‌ای بنویسید که سن شما را برحسب روز از ورودی بخواند و مشخص کند که

چند سال، چند ماه، چند هفته و چند روز دارید.

فصل ۴

عبارت، دستور، عملگر

هدف کلی

آشنایی با عبارتها، دستورهای ساده و ساخت‌یافته، و عملگرهای مختلف

هدفهای رفتاری

انتظار می‌رود پس از خواندن این فصل دانشجو بتواند،

۱. مفهوم عبارت را تعریف کند.

۲. عبارتهای محاسباتی، قیاسی، و منطقی را بشناسد.

۳. مفهوم دستور را تعریف کند.

۴. انواع دستورهای ساده را بشناسد.

۵. انواع دستورهای ساخت‌یافته را بشناسد.

۶. مفهوم عملگر را تعریف کند.

۷. عملگرهای محاسباتی را بشناسد.

۸. عملگرهای انتساب را بشناسد.

۹. عملگرهای یکانی را بشناسد.

۱۰. چند عملگر تک‌عملوندی را نام ببرد.

۱۱. عملگرهای رابطه‌ای (مقایسه‌ای) را بشناسد.

۱۲. عملگر منطقی را بشناسد.

۱۳. عملگر شرطی را بشناسد.

۱۴. عملگر کاما را بشناسد.

۱۵. عملگرهای حافظه را بشناسد.

عبارت

در زبان برنامه‌نویسی، عبارت مجموعه‌ای معنی‌دار از داده‌ها (مقادیر عددی و متغیرها) است که با استفاده از نشانه‌ها یا عملگرهای محاسباتی، قیاسی و منطقی با یکدیگر ترکیب شده‌اند. در زبان C عبارات را به شکل زیر دسته‌بندی می‌کنند.

عبارت محاسباتی

ترکیبی از مقادیر ثابت، متغیرهای صحیح و اعشاری با استفاده از مجموعه عملگرهای محاسباتی است که با قاعدهٔ خاصی تشکیل می‌شود، مانند مثالهای زیر.

۱. a * x + b

۲. (a + b) / c

در مثال اول مقدار a در مقدار x ضرب و نتیجه با مقدار b جمع می‌شود.

در مثال دوم ابتدا مقدار b با مقدار a جمع و سپس بر مقدار c تقسیم می‌شود.

ـ **مثال ۴ـ۱** جدول ۴ـ۱ نمونه‌هایی از عبارات محاسباتی را نشان می‌دهد.

جدول ۴ـ۱ چند عبارت محاسباتی

عبارت ریاضی	معادل در زبان C
$\frac{ab}{c}$	a*b / c
$\frac{a}{bc}$	(a / (b*c))
$\frac{b^2 - 4ac}{2a}$	(b*b – 4*a*c)(2*a)

→

عبارات قیاسی

عبارات قیاسی ترکیبی است از عبارات محاسباتی با استفاده از عملگرهای قیاسی و با رعایت قوانین مربوط به نحوهٔ به کار بردن عملگرها. نتیجهٔ حاصل از اجرای عبارت قیاسی همیشه

درست یا نادرست است؛ یعنی، اگر شرط یا شرطهای به کار رفته در عبارت قیاسی برقرار باشد، نتیجهٔ عبارت مزبور درست است وگرنه نادرست می‌شود، که در اغلب زبانها (مانند پاسکال) آنها را به ترتیب true یا false نامند. ولی، در زبان C مقدار true برابر یک و مقدار false برابر صفر است. در عبارات قیاسی حق تقدم اجرا، اول با عبارات محاسباتی است و سپس عمل مقایسهٔ مورد نظر انجام می‌گیرد.

ـ **مثال ٤ـ۲** نمونه‌هایی از عبارات قیاسی در جدول ۴ـ۲ نشان داده شده است.

جدول ٤ـ۲ چند عبارت قیاسی

عبارت ریاضی	معادل در زبان C
$a > 15$	a >15
$a + b - c \leq 3.14$	a+ b - c <= 3.14
$3a + b \neq 2$	3*a + b != 2

→

عبارت منطقی

عبارت منطقی مجموعه‌ای از عبارات محاسباتی و قیاسی است که در آن حداقل یک عملگر منطقی نیز به کار رفته است. معمولاً این گونه عبارات از دو گروه قبلی پیچیده‌ترند. در اغلب زبانها، عبارات قیاسی که در طرفین عملگر منطقی قرار می‌گیرند باید در داخل پرانتز محصور شوند.

ـ **مثال ٤ـ۳** نمونه‌هایی از عبارات منطقی در جدول ۴ـ۳ نشان داده شده است.

جدول ۴ـ۳ چند عبارت منطقی

عبارت ریاضی	معادل در زبان C
$5 > a > 3$	(a>3) && (a<5) یا (a<5) && (a>3)
$(a \leq 3.14)$ $a \geq 15$	(a <= 3.14) | | (a>=15)

→

دستور

دستور حکمی است که سبب می‌شود کامپیوتر عملی انجام دهد. دو گروه دستور داریم: دستورهای ساده و دستورهای ساخت‌یافته.

دستورهای ساده

دستورهای ساده دستورهای غیرشرطی‌اند که متداول‌ترین آنها عبارت‌اند از:

- جایگزین کردن مقداری معین به یک متغیر که به آن دستور انتساب[1] می‌گویند.
- خواندن و نوشتن
- فراخوانی تابع
- انتقال کنترل به نقطه‌ای از برنامه[2]

ـ **مثال ۴ـ۴** نمونه‌هایی از دستورهای ساده در زیر نشان داده شده است.

```
A = B * C ;
scanf("%d%d ", &a , &b) ;
printf("%d%f",a , b) ;
fact(b) ;
goto a ;
```

دستورهای ساخت‌یافته

دستورهای ساخت‌یافته دستورهایی‌اند که از انواع ساختارهای الگوریتمی ساخته شده‌اند و متداول‌ترین آنها عبارت‌اند از:

- دستور مرکب که شامل دو یا چند دستور متوالی است و در داخل یک زوج آکولاد محصور است. در زبان C هر دستور ساده به یک سمیکولون ختم می‌شود.

```
{
    scanf("%d %d" , &a , &b) ;
    s = a * b ;
    p = 2 * (a+b) ;
    printf("%d %d", s , p) ;
}
```

1. assignment statement
2. go to statement

- دستور حلقهٔ تکرار

```
for (i =0 ; i<=10 ; i++)
    sum = sum + i ;
```

- دستور شرطی.

```
if (a>b)
    c = a + b ;
else
    c = a – b ;
```

عملگر

عملگر یا اپراتور نشانه‌هایی‌اند که در عبارات به کار می‌روند و به کمک آنها می‌توان اعمالی را روی انواع داده انجام داد. انواع عملگرها عبارت‌اند از: محاسباتی، انتساب، یکانی، رابطه‌ای (مقایسه‌ای)، منطقی، شرطی، کاما، و حافظه.

عملگرهای محاسباتی

فهرست عملگرهای محاسباتی در جدول ۴ـ۴ نشان داده شده است.

جدول ٤ـ٤ عملگرهای محاسباتی

نام عملگر	نشانه	شکل	نوع عمل
جمع	+	a + b	a جمع با b
تفریق	-	a - b	a منهای b
منهای یکانی	-	-a	منهای a
جمع یکانی	+	+a	مقدار عملوند a
ضرب	*	a * b	a ضرب در b
تقسیم	/	a / b	a تقسیم بر b
باقیماندهٔ تقسیم	%	a % b	باقیماندهٔ تقسیم a بر b
یک واحد افزایش	++	a++ , ++a	افزایش یک واحد به مقدار a
یک واحد کاهش	--	a-- , --a	کاهش یک واحد از مقدار a

چهار عملگر + , - , * , / تقریباً روی همه نوع داده‌های استاندارد موجود در زبان C به کار می‌رود. در صورتی که عملگر "/" روی مقادیر صحیح یا کاراکتر به کار رود، جزء اعشار حذف می‌شود. مثلاً مقدار 10/3برابر 3 خواهد بود یعنی فقط جزء صحیح آن درنظر گرفته خواهد شد و قسمت اعشار بریده می‌شود.

عملگر % باقیماندهٔ تقسیم را به دست می‌آورد و هر دو عملوند آن باید مقدار صحیح باشد. مثلاً مقدار 10%3 برابر یک خواهد بود. یعنی باقیماندهٔ تقسیم 10 بر 3 مساوی یک است.

دو عملگر ++ و -- در سایر زبانهای برنامه‌نویسی وجود ندارند. عملگر ++ یک واحد به عملوند خود اضافه می‌کند و عملگر -- یک واحد از عملوند خود کم می‌کند. هر دو عملگر تک‌عملوندی‌اند. درواقع دو دستور ; a+ + و ; + +a معادل این دستورند: ; a = a + 1.همچنین دو دستور ; a– – و ; – –a معادل این دستورند ; a = a – 1 .

توجه داشته باشید سرعت عمل دو عملگریکانی ++ و -- از سرعت عمل عملگر انتساب (یعنی عملگر =) بالاتر است.

ـ **مثال ٤ـ٥** به خروجی قطعه برنامهٔ زیر توجه کنید.

```
a = 5 ;
printf ("%d %d\n", a , a+ +) ;
printf ("%d", a) ;
```

خروجی برنامه

```
5    5
6
```

ـ **مثال ٤ـ٦** به خروجی قطعه برنامهٔ زیر توجه کنید.

```
a = 5 ;
printf ("%d %d\n", a , + +a) ;
printf ("%d", a) ;
```

خروجی برنامه

```
5    6
6
```

با توجه به دو مثال بالا ملاحظه می‌کنید که از نظر کاربرد دو دستور ; a+ + و ; + +a با

یکدیگر تفاوت دارند؛ یعنی، در مثال اول با اجرای دستور printf اولی مقدار a (که مساوی 5 است) چاپ می‌شود و سپس دوباره همان مقدار a چاپ می‌گردد و پس از انجام عمل چاپ مقدار آن یک واحد افزایش می‌یابد. با اجرای دستور printf دوم کنترل به آغاز خط جدید انتقال می‌یابد، سپس مقدار a که اکنون برابر 6 است چاپ می‌شود. در مثال دوم با اجرای دستور printf اولی مقدار a (که مساوی 5 است) چاپ و سپس دستور a+ + اجرا می‌شود؛ یعنی، به a یک واحد افزوده می‌شود بعد مقدار آن که برابر 6 شده است چاپ می‌شود. دستور printf دوم در اینجا نیز مشابه مثال اول عمل می‌کند. عملکرد دو دستور a-- و --a نیز به همین روش است.

ترتیب تقدم این گروه از عملگرها در جدول ۴ـ۵ نشان داده شده است.

جدول ۴ـ۵ ترتیب تقدم عملگرها

بالاترین تقدم	++ --
	-
	* / %
پایین ترین تقدم	+ -

در مورد عملگرهای هم‌تقدم ترتیب تقدم از چپ به راست است. درصورت وجود پرانتز، تقدم پرانتز از تقدم همهٔ عملگرها بالاتر است.

ـ **مثال ٤ـ۷** با توجه به مقادیر داده شده، چند عبارت محاسباتی همراه با مقادیر آنها در جدول ۴ـ۶ نشان داده شده است.

```
int a = 10 , b = 3 ;
float C = 12.5 , d = 2.0 ;
```

جدول ۴ـ۶

عبارت محاسباتی	مقدار	عبارت محاسباتی	مقدار
a + b	13	C + d	14.5
a - b	7	C - d	10.5
a * b	30	C * d	25.0
a / b	3	C / d	6.25
a % b	1	C % d	error

ـ **مثال ٤ـ٨** با توجه به اعلان داده شده، چند عبارت محاسباتی همراه با مقادیر آنها در جدول ۴ـ۷ نشان داده شده است.

```
char c1 = 'A' , c2 = 'E' ;
```

جدول ۴ـ۷

عبارت محاسباتی	مقدار
c1	65
c1 + c2	134
c1 + c2 + 5	139
c1 + c2 + '5'	187

در عبارات مورد نظر هر کجا c1 و c2 به کار رفته، به جای آنها کد اسکی معرف کاراکتر مربوط به آنها به کار برده شده است؛ یعنی، در مورد متغیر c1 که معرف کاراکتر A است عدد 65 (کد اسکی حرف A) و در مورد متغیر c2 نیز که معرف کاراکتر E است عدد 69 (کد اسکی حرف E) به کار برده شده است. همچنین ملاحظه می‌کنید که عدد 5، با کاراکتر 5 که کد اسکی آن 53 است تفاوت دارد.

ـ **مثال ٤ـ٩** با توجه به اعلان داده شده چند عبارت محاسباتی همراه با مقادیر آنها در جدول ۴ـ۸ نشان داده شده است.

```
int a = 11 , b = -3 ;
```

جدول ۴ـ۸

عبارت محاسباتی	مقدار
a + b	8
a - b	14
a * b	-33
a / b	-3
a % b	2

در مثال بالا اگر مقدار a برابر 11- و مقدار b برابر 3 باشد، مقدار a / b باز هم برابر 3- می‌شود، اما مقدار a % b برابر 2- خواهد شد. به طریق مشابه اگر a و b هر دو مقدار منفی (متناظراً 11- و 3-) داشتند، مقدار a / b برابر 3 می‌شد، ولی مقدار a % b باز هم برابر 2- باقی می‌ماند.

عملگرهای انتساب

در زبان C علامت '=' به مفهوم مساوی نیست، بلکه عملگر جایگذاری یا عملگر انتساب است. این عملگر موجب می‌گردد که مقدار عملوند سمت راست آن در محل حافظه‌ای که با عملوند سمت چپ مشخص شده است قرار گیرد. برای مثال دستور ; K = 123 مقدار 123 را به متغیر k اختصاص می‌دهد؛ یعنی، آنچه در سمت چپ علامت قرار دارد، نام یک شناسه یا متغیر و آنچه در سمت راست آن قرار دارد، مقدار یا value متغیر مزبور است. پس دستور بالا به مفهوم " k مساوی 123 " نیست بلکه یعنی "مقدار 123 به k اختصاص داده شود". پس باید به تمایز بین نام متغیر و مقدار متغیر توجه کرد.

حال دستور متعارف ; K = K+1 را درنظر بگیرید. از نظر ریاضی این دستور مفهوم نیست. اما از دیدگاه برنامه‌سازی، دستور مزبور دستور جایگذاری و بدین مفهوم است که متغیری را که نام آن K است پیدا کنید. سپس به مقدار آن یک واحد اضافه کنید و مقدار جدید را به متغیری که نام آن K است (درواقع به همان متغیر) اختصاص دهید.

به طور کلی در زبان C چندین عملگر مختلف انتساب یا جایگذاری وجود دارد که همهٔ آنها برای تشکیل یک عبارت انتساب یا عبارت جایگذاری به کار می‌روند و مقدار یک عبارت را به یک شناسه یا متغیر اختصاص یا نسبت می‌دهند. متداول‌ترین عملگر انتساب عملگر '=' است. فرم کلی دستور انتساب به صورت زیر است.

```
identifier = expression ;
variable = expression ;
```

توجه داشته باشید که عملگر انتساب، یعنی '='، کاملاً با عملگر مساوی که علامت ' = =' است فرق دارد. عملگر انتساب برای اختصاص و نسبت دادن یک مقدار به یک شناسه یا متغیر به کار می‌رود، درحالی که عملگر تساوی یا برابری، برای تعیین اینکه آیا دو عبارت مقدار یکسان دارند یا نه به کار می‌رود. پس این دو عملگر نمی‌توانند به جای یکدیگر به کار روند.

اگر دو عملوند عملگر انتساب از نظر نوع یکسان نباشند، مقدار عبارت یا عملوند طرف راست به طور خودکار به نوع شناسه یا متغیر طرف چپ عملگر تغییر می‌یابد. بنابراین اگر نتیجهٔ عبارت سمت راست عملگر مزبور از نوع float و عملوند سمت چپ آن از نوع int

باشد، جزء اعشاری آن حذف خواهد شد.

ـ **مثال ٤ـ١٠** با توجه به اعلان داده شده، نمونه‌هایی از دستور انتساب در جدول ٤ـ٩ نشان داده شده است.

```
int a , b = 5 ;
```

جدول ٤ـ٩

عبارت	مقدار	عبارت	مقدار
a = 5.56 ;	5	a = -25.9 ;	-25
a = b/2 ;	2	a = 2*b /2 ;	5
a = 'x' ;	120	a = 2*(b/2) ;	4
a = '0';	48	a = 'a'- '0'-52	-3
a = ''	32	a = 'E'-'B'/3	47

ملاحظه می‌کنید که در عبارات بالا در مورد کاراکترها، کد اسکی آنها جایگزین می‌شود. مثلاً در آخرین عبارت، به جای E و B به ترتیب مقادیر 69 و 66 قرار می‌گیرد و در نتیجه
69 - 66/3 = 69-22 = 47.

همچنین در عبارت ; a = 2 * (b/2) از آنجایی که پرانتز تقدم بالاتری دارد، اول مقدار b/2 محاسبه می‌شود که نتیجهٔ آن برابر 2 خواهد شد و سپس نتیجهٔ حاصل در 2 ضرب می‌شود و بنابراین نتیجهٔ نهایی برابر 4 خواهد شد.

در زبان C می‌توان دستورهای انتساب چندگانه به کار برد مانند ; a = b = 5. شکل کلی این گونه دستورهای انتساب به صورت زیر است.

```
V1 = V2 =... = Vn = expression ;
```

در چنین حالتی، تقدم عمل انتساب از راست به چپ است. بنابراین دستور ; a = b = c = 5 معادل

; a = (b = (c = 5)) است.

در زبان C، می‌توان عملگر انتساب را با عملگرهای محاسباتی ترکیب کرد و

عملگرهای = +، -= ، =* ، =/ ، =% را به دست آورد که آنها را عملگرهای محاسباتی انتساب[1] نامند و در جدول ۴ـ۱۰ نشان داده شده‌اند.

جدول ٤ـ۱۰ عملگرهای محاسباتی انتساب

عملگر	نام عملگر	دستور انتساب	معادل
+=	انتساب جمع	a += b ;	a = a+b ;
-=	انتساب تفریق	a -= b ;	a = a-b ;
*=	انتساب ضرب	a *= b ;	a = a*b ;
/=	انتساب تقسیم	a /= b ;	a = a/b ;
%=	انتساب باقیماندهٔ تقسیم	a %b ;	a = a%b ;

ـ **مثال ٤ـ۱۱** اگر متغیرهای a و b از نوع int باشند و مقدار آنها به ترتیب برابر 10 و 5 باشد، دستور a += b ; معادل دستور a = a + b ; است؛ یعنی، a = 15.

ـ

عملگرهای یکانی

زبان C مجموعه‌ای عملگر دارد که فقط روی یک عملوند عمل می‌کنند و آنها را عملگرهای یکانی یا تک‌عملوند نامند. این عملگرها اغلب در جلوی عملوند خود قرار می‌گیرند. متداول‌ترین عملگر تک‌عملوندی علامت منفی است که در جلوی یک مقدار ثابت عددی، یا یک متغیر و یا یک عبارت قرار می‌گیرد.

ـ **مثال ٤ـ۱۲** در زیر نمونه‌هایی از عملگر منهای یکانی نشان داده شده است.

```
–915     – 3.1415        – 10.25 –2e –4
–sum     – (x+y) –3 * (x + y)
```

اگر جلوی مقادیر علامتی قرار نگیرد، علامت آنها مثبت در نظر گرفته می‌شود، ولی به هرحال می‌توان جلوی آنها یک علامت "+" قرار داد که در این صورت "+" را که فقط دارای یک عملوند است جمع یکانی نامند.

ـ

1. arithmetic assignment operators

ـ **مثال ٤ـ١٣** در زیر نمونه‌هایی از عملگر جمع یکانی نشان داده شده است.

```
+915        +3.1415        +10.25        +2e – 4
+sum        +(x+y)         +3 * (x + y)
```

اگر عملگر مزبور به کار نمی‌رفت، مقدار اقلام بالا باز هم تغییر پیدا نمی‌کرد و به همین دلیل کاربرد این عملگر متداول نیست.

ـ

از عملگرهای تک‌عملوندی دیگر عملگر "cast" است که آن را عملگر تبدیل نوع نیز گویند و در خودآزمایی ۲ فصل ۳ بررسی کردیم.

یکی دیگر از عملگرهای تک‌علموندی عملگر sizeof است که بزرگی عملوند خود را برحسب بایت برمی‌گرداند. عملوند این عملگر معمولاً با نوع داده همراه است، مانند

int , unsignded int , long , float , unsigned

و مشابه آنها، که در این صورت عملگر مزبور بزرگی این نوع ساختمان داده‌های استاندارد را برمی‌گرداند، یا اینکه عملوند آن عبارت است که در این صورت بزرگی آن عبارت را برحسب بایت برمی‌گرداند. جدول ۴ـ۱۱ نحوهٔ کاربرد این عملگر را نشان می‌دهد.

جدول ٤ـ١١ کاربرد عملگر sizeof

نام عملگر	نشانه	شکل	نوع عمل
بزرگی (sizeof)	sizeof	sizeof (t) sizeof x	بزرگی نوع داده t یا عبارت x را برحسب بایت برمی‌گرداند.

ـ **مثال ٤ـ١٤** به دستورهای زیر توجه کنید.

```
k1 = sizeof (char) ;
k2 = sizeof (short) ;
k3 = sizeof (float) ;
k4 = sizeof (int);
k5 = sizeof (int *) ;
k6 = sizeof (double) ;
```

با اجرای دستورهای بالا مقادیر k1 , k2 , k3 به‌ترتیب برابر 1 , 2 , 4 خواهد بود که به‌ترتیب معرف طول نوع داده char و short و float است. در مورد k4 نیز اگر نوع int

گونه‌ای از زبان C به طول 2 بایت باشد، مقدار k4 برابر 2 خواهد بود و اگر به طول 4 بایت باشد، مقدار آن 4 خواهد بود. k5 نیز بزرگی یک اشاره‌گر به یک داده از نوع integer است که برحسب ماشین مورد نظر ممکن است 4 بایت و یا عدد دیگری باشد.

ـ **مثال ٤ـ١٥** درنتیجهٔ اجرای دو دستور زیر

```
float a ;
printf ("%d%d", sizeof a , sizeof (float)) ;
```

مقادیر 44 نمایش داده خواهد شد، که 4 اول معرف طول متغیر a است (که از نوع float است) و 4 دوم معرف طول نوع داده float است. درضمن ملاحظه می‌گردد که اگر عملوند این عملگر معرف نوع داده مانند int، float باشد، عملوند در داخل پرانتز محصور می‌گردد و اگر معرف متغیر (مانند a در مثال بالا) باشد، نیاز نیست که در درون زوج پرانتز قرار داده شود.

یکی دیگر از عملگرهای متداول تک‌عملوندی عملگر منطقی "!" به مفهوم نقیض است که در عملگرهای منطقی توضیح می‌دهیم.

عملگرهای رابطه‌ای (مقایسه‌ای)

عملگرهای رابطه‌ای، همان طور که از نامشان پیداست، رابطهٔ بین دو مقدار را تعیین می‌کنند. این عملگرها در جدول ۴ـ۱۲ نشان داده شده‌اند.

جدول ٤ـ١٢ عملگرهای رابطه‌ای

نام عملگر	نشانه	شکل	نتیجه
بزرگ‌تر از	>	a > b	اگر a بزرگ‌تر از b باشد، نتیجه 1 وگرنه 0 است.
کوچک‌تر از	<	a < b	اگر a کوچک‌تر از b باشد، نتیجه 1 وگرنه 0 است.
مساوی یا بزرگ‌تر از	>=	a >= b	اگر a مساوی یا بزرگ‌تر از b باشد، نتیجه 1 وگرنه 0 است.
مساوی یا کوچک‌تر از	=<	a<=b	اگر a مساوی یا کوچک‌تر از b باشد، نتیجه 1 وگرنه 0 است.
مساوی	= =	a = =b	اگر a مساوی b باشد، نتیجه 1 وگرنه 0 است.

مخالف	=!	a!=b	اگر a مخالف b باشد، نتیجه 1 وگرنه 0 است.

ایده و مفهوم اصلی در مورد عملگرهای رابطه‌ای وابسته به مفهوم مقدار true و false است. در زبان C، true هر مقدار غیر از صفر و false مقدار صفر است. عباراتی که عملگرهای رابطه‌ای یا منطقی را به کار می‌برند برای حالت نادرست یا false مقدار صفر و برای حالت درست یا true مقدار یک برمی‌گردانند.

تقدم عملگرهای رابطه‌ای پایین‌تر از تقدم عملگرهای محاسباتی است. بنابراین دو عبارت

```
15 > 14+7
15 > (14+7)
```

یکسان ارزیابی خواهند شد. همین طور عبارت a + b* c < d / h به صورت زیر ارزیابی می‌شود.

```
(a +(b* c)) < (d / h)
```

ترتیب تقدم بین خود عملگرهای رابطه‌ای به صورت جدول ۴ـ۱۳ است.

جدول ٤ـ١٣ ترتیب تقدم عملگرهای رابطه‌ای

> >= < <=	بالاترین تقدم
= = !=	پایین‌ترین تقدم

در اینجا نیز مشابه عملگرهای محاسباتی در مورد عملگرهای هم‌تقدم، عملیات از چپ به راست انجام می‌گیرد.

ـ **مثال ٤ـ١٦** به قطعه برنامهٔ زیر توجه کنید.

```
int a ;
a = 10 ;
printf (" %d " , a>5) ;
```

عبارات رابطه‌ای، نتیجه 0 یا 1 ایجاد می‌کنند. بنابراین قطعه برنامهٔ بالا درست است و مقدار 1 را روی صفحهٔ نمایش نشان خواهد داد.

ـ

ـ **مثال ٤ـ١٧** با توجه به اعلان داده شده چندین عبارت مقایسه‌ای همراه با مقدار آنها در زیر نمایش داده شده است.

```
int I = 1 , J = 2 , K = 3 ;
```

عبارت مقایسه‌ای	تفسیر	مقدار
I = = 1	true	1
J != 2	false	0
I < J	true	1
(I+J) >= k	true	1
(J+k) > (k+5)	false	0

استفاده از پرانتز در اینجا ضروری نیست. پرانتز فقط خوانایی برنامه را بیشتر می‌کند.

ـ **مثال ٤ـ١٨** با توجه به اعلان داده شده چندین عبارت مقایسه‌ای همراه با مقدار آنها در زیر نمایش داده شده است.

```
int A = 7 ;
float B = 5.5 ;
char C = 'w' ;
```

عبارت	تفسیر	مقدار
B > 5	True	1
(A + B) <= 10	False	0
C = =119	True	1
C != 'p'	True	1
C >= 10 * (A + B)	False	0

ـ **مثال ٤ـ١٩** با توجه به اعلان داده شده، در جدول ۴ـ۱۴ نمونه‌هایی از نحوهٔ عملکرد مفسر با عبارات مقایسه‌ای پیچیده نشان داده شده است.

```
int j = 0 , m = 1 , n = -1 ;
float x = 2.5 , y = 0.0 ;
```

جدول ۴ـ۱۴

عبارت مقایسه‌ای	عبارت معادل آن	مقدار
j > m	j > m	0
j <= m >= n	((j<m) >= n)	1
j <= x = = m	((j<x) = = m)	1

-x+j = = y>n>m	((-x)+j)= =((y>n)>=m)	0
x += (y>=n)	x =(x+(y>=n))	3.5
++j = = m! =y*2	((++j) = = m) != (y*2)	1
j <= x = = m	((j<=x) = = m	1
-x+y = = y>n>m	((-x)+y = =((y>n) > m	0
x -= (y>=n)	x = (x-(y>=n))	1.5
++y = = m != y*2	((++y) = = m != (y*2)	1

عملگرهای منطقی

عملگرهای منطقی به طور متعارف بر عملوندها یا عبارات منطقی‌ای عمل می‌کنند که دو ارزش درست یا true و نادرست یا false دارند. جدول ٤ـ١٥ عملگرهای منطقی را نشان می‌دهد. در بین عملگرهای منطقی، عملگر"!" بالاترین تقدم و عملگرد "||" پایین‌ترین تقدم را دارد.

جدول ٤ـ١٥ عملگرهای منطقی

عملگر	علامت مفعول	شکل	نتیجه
و (AND) منطقی	&&	a&&b	اگر a و b هر دو برابر یک یا غیرصفر (true) باشند، نتیجه برابر یک، در غیراین صورت برابر صفر خواهد بود.
یا (OR) منطقی	||	a || b	اگر b یا a یا هر دو غیرصفر باشند، نتیجه برابر با یک، در غیراین صورت برابر صفر خواهد بود.
نفی یا نقیض (NOT) منطقی	!	!a	اگر a برابر صفر (false) باشد، نتیجه برابر یک، در غیر این‌صورت صفر خواهد بود.

همان‌طور که در عملگرهای رابطه‌ای بیان شد در زبان C ارزش نادرست یا false با مقدار صفر و ارزش درست یا true با مقادیر غیرصفر مشخص می‌گردد. عملگرهای منطقی را در منطق ریاضی به ترتیب با علایم "∧"، "∨" و "~" نمایش می‌دهند و آنها را به ترتیب ترکیب عطفی، ترکیب فصلی، و نقیض یا نفی نامند. عملگرهای منطقی، بیشتر به صورت ترکیبی با

عملگرهای رابطه‌ای به کار می‌روند. در واقع عملگرهای رابطه‌ای برای مقایسهٔ ارزش دو عبارت هستند، درحالی که عملگرهای منطقی AND و OR برای اتصال دو عبارت ارزشی و در مورد NOT برای نفی آن به کار می‌روند.

ـ **مثال ٤ـ٢٠** با توجه به اعلان داده شده جدول ۴ـ۱۶ مثالهایی از عبارات منطقی را همراه با معادل هر عبارت نشان می‌دهد.

```
int j = 0 , m = 1 , n = -1 ;
float x = 2.5 , y = 0.0 ;
```

جدول ٤ـ١٦

عبارت منطقی	عبارت معادل	تفسیر	نتیجه
j && m	(j) && (m)	false	0
j<m && n<m	(j<m) && (n<m)	true	1
m+n || !j	(m+n) || (!j)	true	1
X*5 && 5 || m/n	((x*5)&&5) || (m/n)	true	1
J<=10 && x>=1 &&m	((j<=10)&&(x>=1))&&m	true	1
!x || !n || m+n	((!x) || (!n)) || (m+n)	false	0
x *y<j+m || n	((x*y)<(j+m)) || n	true	1
(x>y)+!j || n++	((x>y)+(!j)) || (n+ +)	true + true	2
(j || m) + (x || ++n)	(j || m) + (x || (+ +n))	true + true	2

معمولاً یک عبارت رابطه‌ای پیچیده، قسمت شرطی یک دستور حلقه‌ای یا if را تشکیل می‌دهد (دستور if را در فصل دستورهای کنترلی بررسی می‌کنیم). برای مثال عبارت

```
if ((a<b) && (b<c))
    state1 ;
```

از نظر عملکرد معادل دستورهای زیر است.

```
if (a<b)
    if (b<c)
        state1 ;
```

و این سازوکار تا جایی که else وجود نداشته باشد درست است. به هرحال مجموعه دستورهای

```
if ((a<b) && (b<c))
      state1 ;
else
      state2 ;
```

معادل دستورهای زیر نیست.

```
if (a<b)
      if (b<c)
            state1 ;
else
      state2 ;
```

و برای اینکه ازنظر عملکرد همان نتایج به دست آید، باید آن را به صورت زیر نوشت.

```
if (a<b)
      if(b<c)
        state1 ;
      else
        state2 ;
else
      state2 ;
```

عملگر شرطی (:?)

عملیات شرطی ساده را عملگر شرطی انجام می‌دهد. این عملگر سه عملوند دارد و شکل کلی آن در جدول ۴ـ۱۷ نشان داده شده است.

جدول ٤ـ۱۷

عملگر	علامت	شکل	نحوهٔ عمل
شرطی	?:	a ? b: c	اگر a غیرصفر باشد، نتیجه b و گرنه نتیجه c است.

به عبارت دیگر شکل کلی عبارت شرطی به صورت زیر است.

```
exp1 ? exp2:  exp3
```

که در آن ابتدا عبارت اول (exp1) ارزیابی می‌شود، اگر نتیجهٔ آن درست یا true (یعنی مقدار آن غیرصفر) باشد، عبارت دوم (exp2) ارزیابی می‌شود و مقدار عبارت شرطی برابر نتیجهٔ ارزیابی عبارت دوم خواهد بود. ولی چنانچه عبارت اول نادرست یا false (یعنی ارزش آن

برابر صفر) باشد، عبارت سوم (exp3) ارزیابی می‌شود و مقدار عبارت شرطی برابر نتیجهٔ ارزیابی عبارت سوم خواهد بود.

در واقع عملگر شرطی، شکل کوتاه دستور کنترلی if...else است. برای مثال عبارت

```
if (x<y)
    z = 2 ;
else
    z = 3 ;
```

را می‌توان به‌این صورت نوشت.

```
z = ((x<y) ? 2:  3) ;
```

اولین عملوند، بررسی شرط است که باید از نوع اسکالر باشد. عملوندهای دوم و سوم نتیجهٔ نهایی عبارت را نشان می‌دهند که برحسب مقدار عملوند اول، فقط یکی از آن دو انتخاب می‌گردد. عملوندهای دوم و سوم ممکن است از هر نوع داده‌ای باشند تا جایی که دو نوع، براساس قوانین طبیعی تبدیل، سازگار باشند. برای مثال اگر عملوند دوم int و عملوند سوم double باشد، نتیجه بدون توجه به اینکه کدام انتخاب خواهد شد double خواهد بود (یعنی اگر int انتخاب شود، به double تبدیل خواهد شد).

ـ **مثال ٤ـ۲۱** در عبارت شرطی زیر فرض کنید که متغیر i از نوع int باشد.

```
(i<0) ? 0:  100
```

ابتدا عبارت (i<0) ارزیابی می‌شود. اگر پاسخ true بود (یعنی مقدار متغیر i کوچک‌تر از صفر بود)، کل عبارت شرطی مقدار صفر را خواهد پذیرفت. در غیر این صورت (یعنی اگر مقدار i کوچک‌تر از صفر نباشد)، کل عبارت شرطی مقدار 100 را خواهد پذیرفت.

عبارات شرطی، اغلب در طرف راست دستور انتساب ظاهر می‌گردند، مانند مثال اول. در چنین مواردی، مقدار عبارت شرطی به متغیر واقع در طرف چپ دستور انتساب اختصاص می‌یابد.

ـ **مثال ٤ـ۲۲** به عبارت شرطی زیر توجه کنید.

```
min = (i<0) ? 0:  10 ;
```

در اینجا اگر متغیر i منفی باشد، به متغیر min مقدار صفر نسبت داده خواهد شد. وگرنه به متغیر min مقدار 10 اختصاص خواهد یافت. عملگر شرطی تقدمی بالاتر از عملگرهای

انتساب دارد و شرکت‌پذیری آن از راست به چپ است.

ـ **مثال ٤ـ۲۳** به قطعه برنامهٔ زیر توجه کنید.

```
int a =1 , b = 2 , c = 3 ;
c += (a > 0 && a<= 10) ? ++a:  a/b ;
```

در اینجا مقدار عبارت شرطی برابر 2 می‌شود و با توجه به عملگر انتساب جمع داریم

```
c = 3 + 2 = 5
```

معمولاً این عملگر خوانایی برنامه را کاهش می‌دهد. لذا به جای آن اغلب از دستور if else استفاده می‌شود.

عملگر کاما

عملگر کاما "," این امکان را می‌دهد که چندین عمل در یک دستور انجام شوند. نحوهٔ عملکرد این عملگر در جدول ۴ـ۱۸ نشان داده شده است.

جدول ٤ـ۱۸ شکل کلی عملگر کاما

عملگر	علامت	شکل	عمل
کاما	,	a , b ;	a ارزیابی می‌شود، b ارزیابی می‌شود، نتیجه b خواهد شد.

یک روش برای استفاده از عملگر کاما به صورت زیر است.

; (عبارت۲ , عبارت۱) = متغیر

در اینجا عملگر کاما موجب می‌گردد که ابتدا عبارت ۱ و سپس عبارت ۲ ارزیابی شود و نتیجهٔ ارزیابی عبارت ۲ به متغیر مورد نظر نسبت داده شود. در این گونه موارد معمولاً عبارت ۱ و عبارت ۲ با یکدیگر مرتبط‌اند.

ـ **مثال ٤ـ۲٤** عبارت زیر را درنظر بگیرید.

```
a = (b=5 , b+15) ;
```

در عبارت مزبور، ابتدا b برابر 5 قرار داده می‌شود و سپس عبارت b+15 محاسبه می‌گردد که نتیجهٔ آن برابر 20 خواهد بود. در پایان، این مقدار به متغیر a نسبت داده می‌شود،

یعنی پس از اجرای دستور مزبور مقدار a برابر 20 خواهد شد.

کاربرد دیگری از عملگر کاما در دستور for است که این نوع کاربرد بیشتر متداول است و در فصل دستورهای کنترلی بررسی می‌شود.

عملگرهای حافظه

در زبان C چند عملگر وجود دارد که اجازه می‌دهند به خانه‌های حافظه و محتوای آنها دستیابی داشته باشید. با بعضی از این عملگرها، مانند عملگر آدرس (یعنی &)، آشنایید. این گونه عملگرها در جدول ۴ـ۱۹ نشان داده شده‌اند.

جدول ٤ـ۱۹ عملگرهای حافظه

نام عملگر	علامت عملگر	مثال
آدرس	&	&a
محتوا	*	*a
عضو آرایه	[ ]	a[2]
نقطه	.	a. b
پیکان راست	->	p-> a

خودآزمایی ۴

۱. برنامه‌ای بنویسید که عددی را بخواند و قدرمطلق آن را چاپ کند.

۲. برنامه‌ای بنویسید که طبق فرمول زیر درجه حرارت برحسب فارنهایت (F) را به درجه حرارت برحسب سیلسیوس (C) تبدیل کند.

C = 5/9 * (F - 32)

۳. اگر b = 5 و a = 2، حاصل عبارت ; y = a*b + ++a + 10 چیست ؟

٤. برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و مشخص کند آیا این اعداد می‌توانند اضلاع یک مثلث باشند یا خیر.

۵. دستورهای زیر را به ترتیب اجرا کنید و در هر قسمت معادل بیتی و محتوای متغیر x را مشخص سازید.

```
x = 7 ;
x = x >> 1 ;
x = x >> 3 ;
x = x >> 2 ;
x = x << 1 ;
x = x << 2 ;
x ; ^ x = x
```

٦. خروجی قطعه برنامهٔ زیر چیست ؟

```
int x = 7 , y = 8 ;
printf ("%d" , x && y) ;
printf ("%d" , x & y) ;
```

فصل ۵

دستورهای کنترلی

هدف کلی

آشنایی با مهم‌ترین دستورهای کنترلی در زبان C و کاربرد آنها

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از خواند این فصل،

۱. با کاربرد دستورهای کنترلی آشنا شود.

۲. شکل کلی و کاربرد دستور while را بداند.

۳. شکل کلی و کاربرد دستور do-while را بداند و تفاوت آن را با while بیان کند.

۴. شکل کلی و کاربرد دستور for را بداند.

۵. شکل کلی و کاربرد دستور عملگر کاما را بداند.

۶. شکل کلی و کاربرد دستورهای if و if-else را بداند.

۷. شکل کلی و کاربرد دستور switch را بداند.

۸. شکل کلی و کاربرد دستور break را بداند.

۹. شکل کلی و کاربرد دستور continue را بداند.

۱۰. شکل کلی و کاربرد دستور goto را بداند.

۱۱. شکل کلی و کاربرد تابع exit را بداند.

مقدمه

یکی از امکانات زبانهای برنامه‌نویسی جدید، استفاده از دستورها و ساختارهای کنترلی است و در نتیجه این امکان را فراهم می‌سازند که قطعه‌ای از برنامه تا موقعی که شرط ویژه‌ای برقرار

است چندین بار اجرا شود.

زبان برنامه‌نویسی C دارای قلمرو گسترده‌ای از این نوع ساختارهاست. در حالت عادی دستورهای هر برنامه به طور متوالی اجرا می‌شود. اما اگر نیاز باشد که دستور یا مجموعه‌ای از دستورها در صورت وجود یا عدم وجود شرط یا شرایط خاصی اجرا گردند باید شیوهٔ دیگری به کار برد. ساختارهای کنترلی دستورهایی‌اند که چنین زمینه‌ای را در برنامه‌نویسی فراهم می‌کنند.

مهم‌ترین دستورهای کنترلی عبارت‌اند از while _ do و for _ while که ساختارهای حلقه‌های تکرار را تشکیل می‌دهند و دستورهای switch و if که دستورهای شرطی یا ساختارهای تصمیم‌گیری‌اند و بالاخره دستورهای goto، continue، break، و exit.

حال به دو مفهوم درست یا true و نادرست یا false توجه کنید. اغلب دستورهای کنترلی برنامه در زبان C بر نتیجهٔ وجود شرط تکیه می‌کند تا برحسب برقراری آن شرط عملی انجام گیرد یا انجام نگیرد. درواقع نتیجهٔ آزمایش این شرط مقدار درست یا نادرست است. اغلب زبانهای کامپیوتری مقادیری را به عنوان ارزش درستی یا نادرستی مشخص می‌کنند (مثلاً ۱ برای درست و ۱- برای نادرست)، اما در زبان C هر مقدار غیرصفر (مثبت یا منفی) درست یا true (یعنی شرط مورد نظر برقرار است) و مقدار صفر نیز نادرست یا false تلقی می‌گردد.

در این فصل، مهم‌ترین دستورها و ساختارهای کنترلی زبان C را بررسی می‌کنیم.

دستور while

این دستور یکی از دستورهای کنترلی زبان C است که برای انجام عملیات تکراری به کار می‌رود. با استفاده از این دستور، حلقه تا موقعی که شرط معینی برقرار باشد اجرا می‌گردد. شکل کلی این دستور به صورت زیر است.

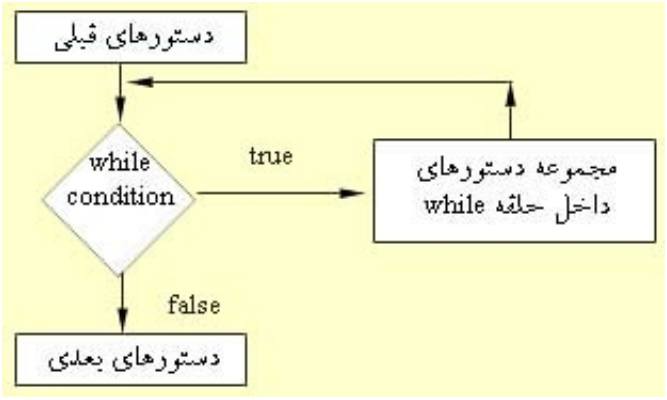
```
while (condition)
    statement ;
```

در اینجا پس از عبارت while فقط یک دستور به کار رفته است. اما می‌توان مجموعه‌ای از دستورها را نیز به کار برد. گفتیم که در زبان C، هر دستور به یک سمیکولون ختم می‌شود و مجموعه‌ای از دستورها (یعنی بیش از یک دستور) را نیز دستورهای مرکب یا بلاک نامند که

در زبان C در داخل یک زوج آکولاد قرار می‌گیرد. بنابراین در حالت کلی شکل دستور while به صورت زیر خواهد بود.

```
while (condition)
    {
        statements ;
    }
```

نمودار کلی آن را نیز در شکل ۵ـ۱ می‌بینید.



شکل ۵ـ۱ نمودار کلی دستور while

سازوکار و نحوهٔ عملکرد دستور در while به‌این طریق است که تا موقعی که شرط مورد نظر که پس از کلمهٔ کلیدی while در داخل پرانتز نوشته می‌شود برقرار باشد، مجموعه دستورهای داخل حلقهٔ while به صورت تکراری اجرا خواهد شد. شرط مورد نظر با استفاده از عملگرهای رابطه‌ای به صورت عبارات رابطه‌ای یا به صورت عبارت منطقی بیان می‌شود که در این صورت تا موقعی که عبارت مزبور ارزش درست یا true داشته باشد حلقه اجرا خواهد شد.

ـ **مثال ۵ـ۱** برنامهٔ زیر به دو روش اعداد صحیح صفر تا ۱۰ را در روی خطوط متوالی چاپ می‌کند.

روش اول

```
#include<stdio.h>
main()
   {
      int number = 0 ;
      while (number<=10)
           printf ("%d\n", number ++);
    }
```

روش دوم

```
#include<stdio.h>
main()
   {
     int number = 0 ;
     while (number<=10)
          { printf("%d\n", number);
            ++ number;
          }
    }
```

→

ـ **مثال ۵ـ۲** برنامه‌ای بنویسید که عدد صحیح n را بخواند و فاکتوریل آن را حساب و با خود عدد چاپ کند.

```
#include<stdio.h>
main ()
   {
     int n , i =1 , fact =1 ;
     scanf ("%d",&n) ;
     while (++ i <= n)
          fact *= i ;
     printf ("factorial of %d is %d", n , fact) ;
   }
```

→

ـ **مثال ۵ـ۳** برنامهٔ زیر یک خط متن با حروف کوچک را کاراکتر به کاراکتر از آرایهٔ text می‌خواند. سپس با استفاده از تابع کتابخانه‌ای toupper متن مزبور را به حروف بزرگ تبدیل و چاپ می‌کند.

```
#include<stdio.h>
#define eol '\n'
main ()
   {
     char text[80] ;
     int tag , count = 0 ;
                  /*read the text in lower case */
     text [count] = getchar() ;
     while (text[count]!eol)
        { count = count + 1 ;
```

```
        text[count]=getchar() ;
    }
    tag = count ;
    count = 0 ;
    while (count<tag)
      { putchar (toupper(text[count])) ;
          + + count ;
      }
}
```

در برنامهٔ بالا دو عبارت while جداگانه به کار رفته که یکی برای خواندن متن از حافظه و دیگری برای تبدیل حرف کوچک به بزرگ و چاپ متن تبدیل شده است. حال اگر mathematics and statistics وارد شود، عبارت MATHEMATICS AND STATISTICS در خروجی نمایش داده می‌شود.

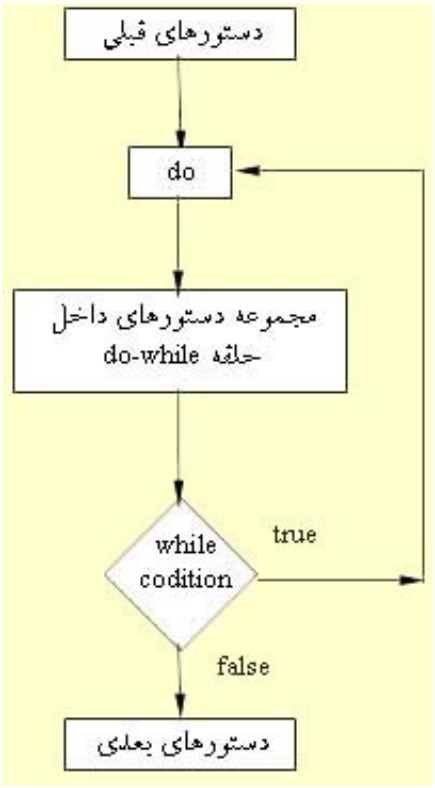
دستور do - while

در دستور while آزمایش شرط برای ادامهٔ حلقه در آغاز هر تکرار حلقه انجام می‌گیرد. گاهی مطلوب است که این آزمایش در پایان حلقه انجام شود. این کار با دستور کنترلی do-while امکان‌پذیر است. شکل کلی دستور do-while به صورت زیر است.

```
do
  {
     statements
  }while (condition) ;
```

در صورتی که حلقهٔ تکرار فقط شامل یک دستور باشد، نیازی به قراردادن زوج آکولاد نخواهد بود. در اینجا اول statements اجرا می‌گردد، سپس شرط داخل پرانتز، یعنی condition بررسی می‌شود. بنابراین، در این ساختار همیشه statements حداقل یک بار اجرا خواهد شد. در این حالت نیز عبارت داخل پرانتز معمولاً یک عبارت رابطه‌ای یا منطقی است که نتیجهٔ آن مشابه while است.

در اغلب کاربردها، آزمایش شرط ادامه برای اجرای حلقه، به طور طبیعی در آغاز حلقه صورت می‌گیرد. بدین لحاظ دستور do-while در مقایسه با دستور while کاربرد کمتری دارد. نحوهٔ عملکرد این دستور در شکل ۵ـ۲ نشان داده شده است.



شکل ۵ـ۲ نمودار کلی دستور do – while

ـ **مثال ۵ـ٤** برنامه‌ای بنویسید که با استفاده از دستور do - while اعداد صحیح صفر تا ۱۰ را در روی خطوط متوالی چاپ کند.

```
#include<stdio.h>
main ()
   {
     int number = 0 ;
     do
          printf ("%d\n", number + +) ;
     while (number<=10) ;
   }
```

ـ **مثال ۵ـ۵** برنامه‌ای بنویسید که با استفاده از دستور do - while عدد صحیح n را بخواند و فاکتوریل آن را حساب و با خود عدد چاپ کند.

```
#include<stdio.h>
main ()
   {
     int n , i =1 , fact =1 ;
     scanf ("%d",&n) ;
     do
          fact *= i ;
          while (+ +i<= n) ;
          printf ("factorial of %d is %d", n , fact) ;
   }
```

دستور for

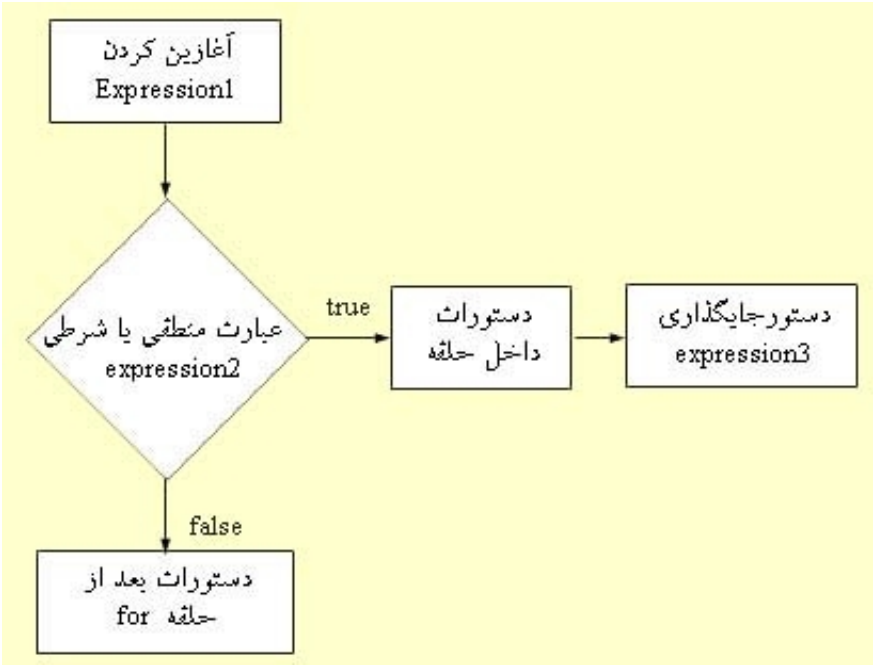
دستور for متداول‌ترین دستور حلقه در زبان C و شبیه به دستور while است. این دستور شامل یک عبارت است که مقدار نخستین یک شاخص را مشخص می‌سازد. عبارت دیگر چگونگی ادامه یا پایان حلقه را تعیین می‌کند و سومین عبارت شاخص را در پایان هر حلقه تغییر می‌دهد. نمودار آن را در شکل ۵ـ۳ می‌بینید.

شکل کلی این دستور نیز به صورت زیر است.

for (expression1; expression2; expression3) statement ;

در اینجا expression1 برای مقداردهی اولیهٔ پارامتری که حلقه را کنترل می‌کند (و شاخص یا index نامیده می‌شود) به کار می‌رود. expression2 یک شرط را معرفی می‌کند که باید برای ادامهٔ اجرای حلقه صادق باشد و expression3 نیز برای تغییر مقدار پارامتری که در آغاز به expression1 اختصاص داده شده به کار می‌رود. معمولاً expression1 عبارت جایگذاری، expression2 عبارت منطقی یا رابطه‌ای و expression3 عبارت جایگذاری یا unary expression است.

وقتی که دستور for اجرا می‌گردد قبل از هر گذر در داخل حلقه، expression2 ارزیابی و آزمایش می‌شود. اما expression3 در پایان هر گذر ارزیابی می‌گردد. بنابراین دستور for



شکل ۵ـ۳ نمودار دستور for

معادل دستورهای زیر است.

```
expression1 ;
while (expression2)
   {
     statements
     expression3 ;
   }
```

اجرای حلقه تا زمانی که مقدار expression2 مساوی صفر نباشد، یعنی تا هنگامی که شرط معرفی شده با expression2 برقرار و یا true باشد، ادامه خواهد داشت.

ـ **مثال ۵ـ٦** با استفاده از دستور for برنامه‌ای بنویسید که اعداد صحیح صفر تا ۱۰ را روی خطوط متوالی چاپ کند.

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

```
    for (number=0 ; number<=10 ; + + number)
      printf ("%d\n", number) ;
  }
```

در این برنامه اولین خط در دستور for شامل سه expression است که در داخل پرانتز قرار گرفته‌اند. اولین expression مقدار صفر را به متغیر number اختصاص می‌دهد. دومین expression بیان می‌کند که تکرار حلقه تا هنگامی که مقدار فعلی number از ۱۰ تجاوز نکرده باشد ادامه خواهد یافت. سومین expression پس از هر بار گذر حلقه، مقدار number را یک واحد افزایش می‌دهد.

از نظر دستوری نیاز نیست هر سه expression مربوط به دستور for به کار برده شود، اگرچه به کار بردن سمیکولون مربوط به هر expression الزامی است. نتیجهٔ حذف هر expression باید به طور کامل فهمیده شود. اگر آغازین کردن و یا تغییر شاخص یا هر دو به طریق دیگری انجام گیرد، ممکن است expression های اول و سوم حذف گردند. درصورتی که expression دوم حذف گردد، ارزش آن مقدار ثابت true درنظر گرفته خواهد شد و در نتیجه تکرار حلقه به صورت نامتناهی ادامه خواهد یافت مگر اینکه خروج از حلقه به طریق دیگری مثلاً با دستور break یا return پیش‌بینی گردد. در اغلب کاربردهای دستور for هر سه expression استفاده می‌شود.

ـ **مثال ۵ـ۷** با استفاده از دستور for برنامه‌ای بنویسید که عدد صحیح n را بخواند و فاکتوریل آن را حساب و با خود عدد چاپ کند.

```
#include<stdio.h>
main ()
  {
    int n , fact =1 ;
    scanf ("%d", &n) ;
    for (i=2 ; i<=n ; ++i)
      fact = fact * i ;
    printf ("factorial of %d is %d", n , fact) ;
  }
```

ـ **مثال ۵ـ۸** برنامه‌ای بنویسید که نمره‌های امتحانی دانشجویان کلاسهای مختلف را

بخواند و معدل هر کلاس را حساب و چاپ کند. تعداد کلاسها و تعداد دانشجویان هر کلاس از طریق دستگاه ورودی خوانده شود.

```
#include<stdio.h>
main ()
  {
    int n , count , loops , loopcount ;
    float x , average , sum ;
    scanf("%d",&loops) ;
                /*outer loop*/
    for (loopcount=1 ; loopcount<=loops ; ++ loopcount)
      {
        sum = 0 ;
        printf("\n class number %d \n how many numbers?", loopcount) ;
        scanf ("%d",&n) ;
        for (count=1 ; count<=n ; + +count)
                /* inner loop */
          {
            scanf ("%f",&x) ;
            sum += x ;
          }     /*end of inner loop*/
        average = sum / n ;
        printf("\n the average is %f\n", average) ;
      }         /*end of outer loop*/
  }
```

حلقه‌ها می‌توانند در یکدیگر قرار گیرند یعنی به صورت تودرتو[1] به کار روند که در این حالت حلقهٔ داخلی باید کاملاً توسط حلقهٔ بیرونی احاطه گردد. به عبارت دیگر دو حلقه نباید یکدیگر را قطع کنند. همچنین هر حلقه باید با شاخص جداگانه‌ای کنترل شود. برنامهٔ فوق از دو حلقه تشکیل شده است. در آغاز تعداد کلاسها (loops) خوانده می‌شود و بر اساس آن حلقهٔ بیرونی تکرار می‌گردد یعنی به ازای هر کلاس یک بار اجرا می‌شود. هر بار در داخل حلقهٔ بیرونی، متغیر sum مساوی صفر قرار داده می‌شود، سپس تعداد دانشجویان یا تعداد نمره‌های امتحانی n خوانده می‌شود. پس از آن حلقهٔ درونی به تعداد دانشجویان کلاس مورد نظر n اجرا می‌شود که در هر تکرار نمرهٔ امتحانی یک دانشجو خوانده و به sum افزوده می‌شود. هر بار پس از کامل شدن حلقهٔ درونی، جمع نمره‌های امتحانی کلاس به دست می‌آید

1. nested loops

که سپس معدل کلاس با نام average حساب و چاپ می‌شود.

عملگر کاما

در خصوص این عملگر در فصل ۴ مطالبی بیان شد. این عملگر عملگری باینری است و به دو عبارت مختلف اجازهٔ ظاهر شدن در موقعیتهایی را می‌دهد که در آن باید فقط یک عبارت به کار رود. همچنین عملگر کاما از چپ به راست عمل می‌کند و در مجموعه عملگرهای زبان C کمترین تقدم را دارد. عملوندهای آن عبارت‌اند از

expression1, expression2

در اینجا ابتدا expression1 و سپس expression2 محاسبه می‌شود.

کاما ممکن است در دستور for به کار رود و اجازهٔ پردازش چندین اندیس را به شکل یکجا می‌دهد. همچنین امکان مقداردهی اولیهٔ چند متغیر را به شکل همزمان فراهم می‌سازد. حالتهای مختلف دستور for با توجه به عملگر کاما به شکل زیر است.

```
for (expression1 ; expression2 ; expression3)
for (expression1A , expression1B ; expression2 ; expression3)
for (expression1 ; expression2 ; expression3A , expression3B)
```

ـ **مثال ۵ـ۹** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main ()
   {
     int i , n , sum ;
     scanf("%d", &n) ;
     for (sum = 0 , i = 1 ; i<n ; sum+= i , ++i) ;
       printf ("\n %d", sum) ;
     }
```

این برنامه مجموع اعداد صحیح ۱ تا n را محاسبه و چاپ می‌کند. همانطور که ملاحظه می‌کنید در اینجا دو بار از عملگر کاما در دستور for استفاده شده است. همچنین در انتهای دستور for سمیکولون به کار رفته است. یعنی پس از پایان حلقه که مقدار sum محاسبه شد، کنترل برنامه به دستور printf منتقل می‌شود.

لازم به توضیح است که برای تفکیک متغیرها، اعلان متغیرها و آرگومانهای توابع از

کاراکتر کاما استفاده می‌شود که نباید آن را با عملگر کاما اشتباه کرد.

دستور if و if-else

دستورهای شرطی برای انجام آزمون منطقی و برگزیدن یکی از دو حالت ممکن که به نتیجهٔ آزمون بستگی دارد استفاده می‌شود. به عبارت دیگر این دستورها موجب می‌گردند تا در صورت وجود شرط یا شرایطی، مجموعه‌ای از دستورها اجرا گردند و یا در صورت وجود شرط یا شرایطی، یک مجموعه از دستورها و در صورت عدم وجود آن، مجموعهٔ دیگری از دستورها اجرا شوند. این دستورها را ساختارهای تصمیم نیز می نامند.

دستور if به صورتهای if وif-else به کار می‌رود. ساده‌ترین شکل دستور if به صورت زیر است.

```
if (condition)
   statement ;
```

درصورتی که بیش از یک دستور مورد نظر باشد، شکل دستور if به صورت زیر خواهد بود.

```
if (condition)
   {
     statements ;
   }
```

در هر دو شکل if دستورها درصورتی اجرا می‌گردد که شرط مورد نظر، که پس از کلمهٔ کلیدی if در داخل یک زوج پرانتز بیان شده است، برقرار باشد. در هر حال پس از پایان اجرای ساختار if، دستورهای پس از ساختار if اجرا می‌گردند. شرط مورد نظر ممکن است به صورت عبارت منطقی بیان گردد که در این صورت اگر نتیجهٔ آن true (یا عدد غیرصفر) باشد، باز هم دستورهای داخل زوج آکولاد اجرا می‌گردد و سپس کنترل به اولین دستور بعد از ساختار if انتقال می‌یابد؛ و اگر نتیجهٔ آن نادرست یعنی false (یا عدد صفر) باشد، مشابه حالت شرطی کنترل مستقیماً به بعد از ساختار if انتقال می‌یابد.

شکل کلی دستور if-else به صورت زیر است.

```
if (condition)
   {
     statements1;
```

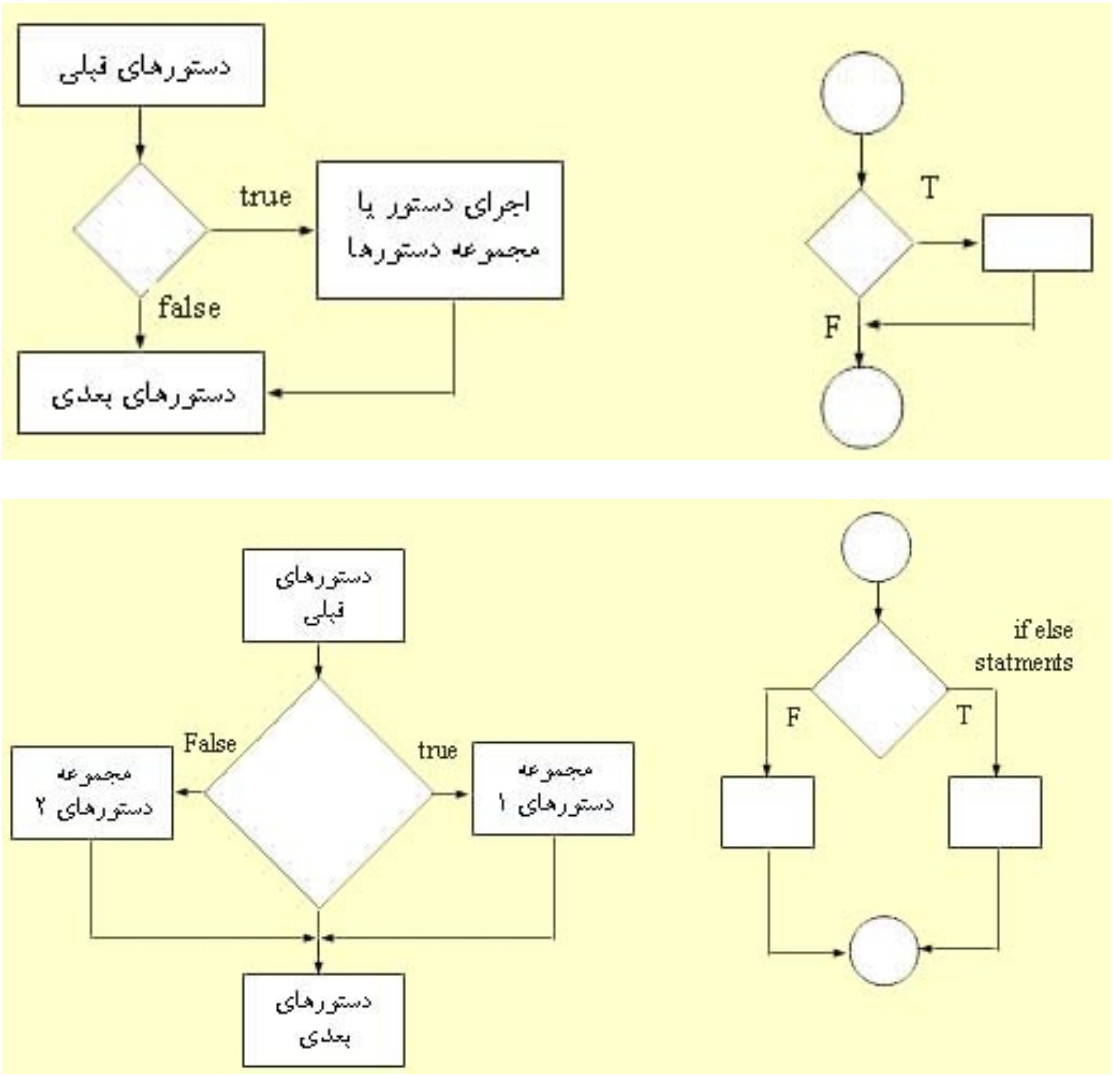
```
  }
 else{
        statements2 ;
    }
 next statement ;
```

در این حالت condition عبارت شرطی یا عبارت منطقی است که اگر شرط مورد نظر برقرار باشد یا نتیجهٔ عبارت منطقی درست باشد، دستورهای داخل زوج آکولاد اول اجرا می‌گردد و کنترل به next statement انتقال می‌یابد وگرنه دستورهای داخل زوج آکولاد دوم اجرا می‌شود و باز هم کنترل به next statement انتقال می‌یابد.

نمودار هر دو ساختار در شکل ۵ـ۴ نشان داده شده است. اگر دستورهای بعد از if یا else بیش از یک دستور نباشد، به استفاده از زوج آکولاد نیازی نخواهد بود. ساختار if را می‌توان به صورت تودرتو نیز به کار برد که در این صورت هم if و هم else به صورت تودرتو تا هرچند سطح که منطق برنامه نیاز داشته باشد تکرار می‌شوند.

ـ **مثال ۵ـ۱۰** برنامه‌ای بنویسید که متنی را بخواند و تعداد محلهای خالی، ارقام، حروف، پایان خط و جمع سایر نشانه‌های موجود در آن متن را بشمارد و چاپ کند.

```
#include<stdio.h>
main()
  {
    int c , blank_cnt , digit_cnt , letter_cnt , n1_cnt , other_cnt ;
    blank_cnt=digit_cnt=letter_cnt=n1_cnt=other_cnt= 0;
    while ((c=gethar())!=EOF)
    if (c= =' ')
      ++ blank_cnt ;
    else if ('0'<=c&&c<='9')
     ++ digit_cnt ;
    else if ('a'<=c && c<='z' || 'A'<=c && c<='Z')
      ++ letter_cnt ;
    else if (c= ='\n')
      ++n1_cnt ;
    else
      ++ other_cnt ;
    printf ("\n%12s %12s %12s %12s %12s %12s","blanks","digits",
          "letters","lines","others","totals") ;
    printf("\n\n %12d %12d %12d %12d %12d %12d
          \n\n" , blank_cnt , digit_cnt , letter_cnt , n1_cnt , other_cnt ,
          blank_cnt + digit_cnt + letter_cnt + n1_cnt + other_cnt) ;
  }
```



شکل ۵ـ۴ نمودار دو ساختار if و if – else

ـ **مثال ۵ـ۱۱** برنامه‌ای بنویسید که کد اسکی کاراکتری را بخواند و تشخیص دهد که کدام یک از کاراکترهای پایان فایل یا EOF، خط جدید یا new line، رقم یا digit، کنترل

کاراکتر (کاراکترهایی که کد اسکی آنها کوچک‌تر از ۳۲ است و قابل چاپ نیستند و ۳۲ که کد اسکی کاراکتر فضای خالی یا ’ ’ است، پس کد اسکی کنترل کاراکترها از کد اسکی ’ ’ کوچک‌تر است)، حروف بزرگ، حروف کوچک و یا سایر کاراکترهایند.

```
#include<stdio.h>
main ()
  {
      int ch ;
      printf("please enter a character?") ;
      ch = getchar() ;
      if (ch= = EOF)
          printf("\n end of file found\n") ;
      else
          if (ch= =’\n’)
            printf("\n new line character\n") ;
      else
          if (ch<’ ’)
            printf("\n control character %d \n", ch) ;
      else
          if ((ch>=’0’) && (ch<=’9’))
            printf("\n character %c is a digit\n", ch) ;
     else
          if ((ch>’A’) && (ch<=’Z’))
            printf("\ncharacter %c is upper-case\n", ch)) ;
     else
          if ((ch>=’a’) && (ch<=’z’))
            printf("\n character %c is lower-case\n", ch) ;
     else
        printf("\n other printable character %c \n", ch) ;
  }
```

دستور switch

این ساختار یا دستور موجب می‌گردد که گروه مشخصی از دستورها بین چندین گروه از دستورها انتخاب گردد. این انتخاب بستگی به مقدار فعلی یک عبارت خواهد داشت که در switch منظور شده است. قبلاً گفتیم که در دستور if-else، بر اثر وجود یا عدم وجود شرط (یا شرایطی)، یکی از دو مجموعه دستورها انتخاب و اجرا می‌گردد. به عبارت دیگر در اینجا

حداکثر دو انشعاب یا دو شاخه وجود دارد که باید فقط بر اثر نتیجهٔ بررسی شرط مورد نظر، یکی از آن دو شاخه یا مسیر انتخاب گردد. دستور switch این امکان را فراهم می‌سازد که از بین شاخه‌های متعدد (بیش از دو شاخه) فقط یکی انتخاب گردد که قانون انتخاب را نتیجهٔ ارزیابی عبارت ذکر شده در switch تعیین می‌کند. بنابراین، این دستور اجازهٔ چندین انشعاب می‌دهد.

شکل کلی ساختار switch به صورت زیر است.

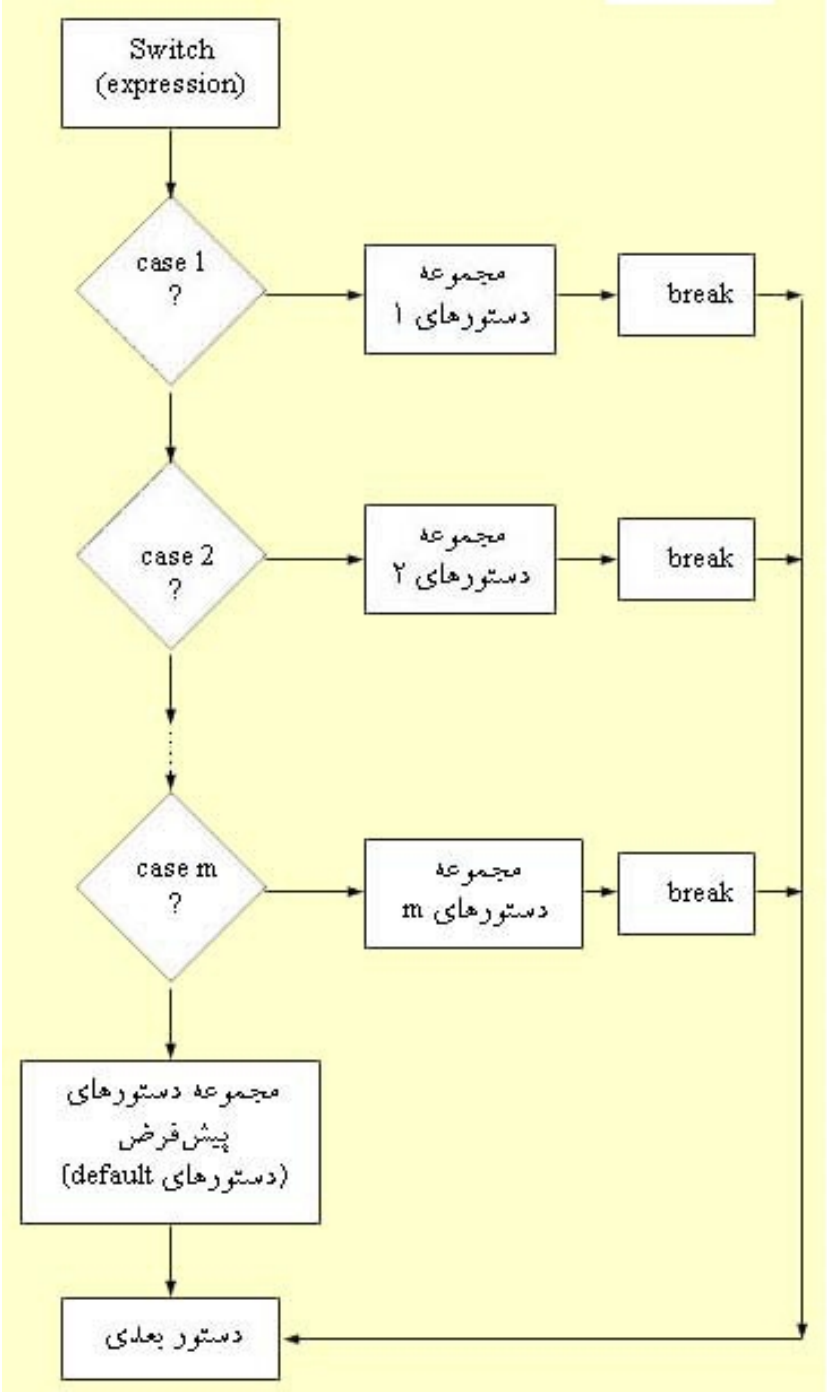
```
switch(expression)
{
     case exp1:  statements 1
                 break ;
     case exp2:  statements 2
                 break ;
    ....
    ....
     case exp n:  statements n
                 break ;

       default:  statements
                 break ;
}
```

در دستور switch نتیجهٔ عبارت یا expression که پس از کلمهٔ کلیدی switch در داخل یک زوج پرانتز قرار دارد یک عدد صحیح خواهد بود. در ضمن expression ممکن است کاراکتر نیز باشد زیرا هر کاراکتر نیز مقدار صحیح هم‌ارزی دارد که کد اسکی آن کاراکتر است. statement معمولاً دستوری مرکب است. break برای خروج از دستور است.

نمودار ساختار switch در شکل ۵ـ۵ نشان داده شده است.

نحوهٔ عملکرد ساختار switch بدین شکل است که وقتی این دستور اجرا می‌شود عبارت ذکر شده در switch محاسبه و ارزشیابی می‌شود. سپس نتیجهٔ آن از بالا به پایین به ترتیب با مقدار ذکر شده در case یعنی exp 1 , exp 2 ,... , exp n مقایسه می‌شود. اگر نتیجهٔ مقایسه با هرکدام از مقادیر ذکر شده در case های مزبور برابر باشد، دستورهای ذکر شده در آن case اجرا می‌گردد و کنترل اجرای برنامه به اولین دستور پس از ساختار switch انتقال می‌یابد. چنانچه مقدار ذکر شده در switch با هیچ یک از مقادیر ذکر شده در caseهای متوالی

یکسان نباشد، مجموعه دستورهای مربوط به default به عنوان پیش‌فرض اجرا می‌گردد.



شکل ۵ـ۵ نمودار ساختار switch

در دستور switch استفاده از حالت default دلخواه است. بنابراین چنانچه حالت default به کار برده نشود و مقدار عبارت switch در مقابله با مقادیر ذکر شده در caseهای متوالی مساوی (همسان) نباشد، دستور switch خاتمه می‌یابد و کنترل برنامه به اولین دستور بعد از switch انتقال می‌یابد.

یادآوری مهم. گفتیم، عبارت ذکر شده در switch فقط مقادیر صحیح یا کاراکتر را می‌پذیرد. بنابراین برخلاف دستور if نمی‌توان در آن از عبارت رابطه‌ای و منطقی استفاده کرد؛ به عبارت دیگر switch فقط تساوی را تست می‌کند، اما if عبارات رابطه‌ای و منطقی را نیز ارزیابی می‌کند. نتیجه‌ای که می‌توان گرفت آن است که در حالت کلی نمی‌توان دستور switch را حالت تعمیم‌یافتهٔ دستور if-else برای m انشعاب تصور کرد.

دستور switch برای پردازش فرامین صفحه کلید بسیار مناسب است و در واقع شکل تغییریافته‌ای از دستور if-else است.

ـ **مثال ۵ـ۱۲** برنامهٔ زیر با استفاده از دستور switch ارقام ۰ تا ۹ را از ورودی دریافت می‌کند و آنها را با حروف نمایش می‌دهد.

```
#include<stdio.h>
main ()
{
   char digit ;
   digit = getchar() ;
   switch (digit)
     {
       case 0:  printf ("zero") ;
                break ;
       case 1:  printf ("one") ;
                break ;
       case 2:  printf ("two") ;
                break ;
       case 3:  printf ("three") ;
                break ;
       case 4:  printf ("four") ;
                break ;
       case 5:  printf ("five") ;
                break ;
```

```
        case 6:  printf ("six") ;
                 break ;
        case 7:  printf ("seven") ;
                 break ;
        case 8:  printf ("eight") ;
                 break ;
        case 9:  printf ("nine") ;
                 break ;
      default:  printf("it is not digit");
    }
  }
```

ـ **مثال ۵ـ۱۳** برنامهٔ زیر با استفاده از دستور switch متغیر کاراکتری color را که معرف رنگ است از صفحه کلید دریافت می‌کند و رنگی را که با آن حرف آغاز می‌شود در خروجی چاپ می‌کند.

```
#include<stdio.h>
main ()
{
   char color ;
   color = getchar() ;
   switch (color)
     {
       case 'Y': printf ("Yellow") ;
                 break ;
       case 'B': printf ("Black") ;
                 break ;
       case 'W': printf ("White") ;
                 break ;
       case 'R': printf ("Red") ;
                 break ;
       case 'G': printf ("Green") ;
                 break ;
      default:  printf("error");
     }
}
```

در این برنامه می‌توان دستور switch را با دستور خواندن متغیر از ورودی به شکل زیر ترکیب کرد.

```
switch (color=getchar())
```

در مواردی برای چندین حالت مختلف یک نوع عمل انجام می‌گیرد. در این گونه موارد

می‌توانیم همان دستورها را با چندین ثابت case مشخص کنیم.

ـ **مثال ۵ـ۱۴** به قطعه برنامهٔ زیر توجه کنید.

```
switch (ch)
{
      case 'a':
      case 'b':  x = x + 1 ;
                 break ;
      case 'c':
      case 'd':
      case 'e':  x = x + 2 ;
                 break ;
      default:   x = x + 3 ;
                 break ;
}
```

دستور break

این دو دستور مسیر معمولی و ثابت کنترل را تغییر یا قطع می‌کنند. دستور break برای خارج شدن از دستور switch و یا پایان دادن حلقه‌ها استفاده می‌شود. در واقع کنترل را به پایان ساختاری که break در داخل آن به کار برده شده است انتقال می‌دهد. در صورتی که دستور مزبور در یکی از حلقه‌های for وwhile و do-while به کار رود، کنترل به محض رسیدن به break بلافاصله از حلقه خارج می‌شود. بنابراین، این روش برای پایان دادن به حلقه‌ها در صورت بروز خطا یا شرایط خاص مناسب است.

ـ **مثال ۵ـ۱۵** به قطعه برنامهٔ زیر توجه کنید.

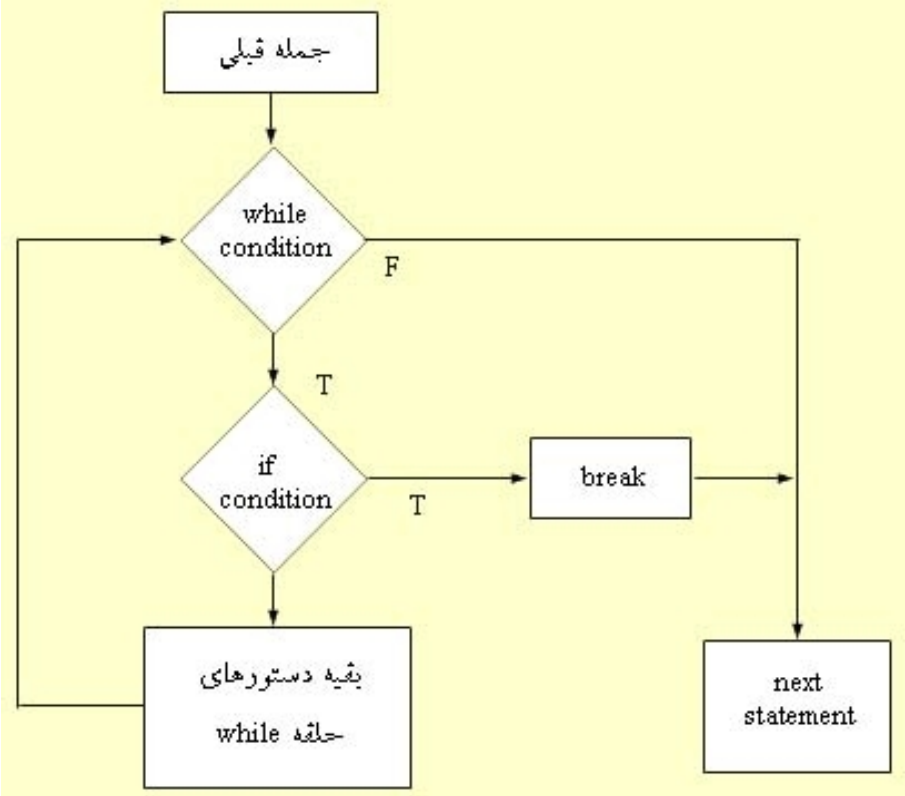
```
while(1)
      { ch = getchar() ;
        if (ch = = '@')
                  break ;          /* exit loop */
        printf ("%c ", toupper(ch)) ;
      }
```

در صورتی که کاراکتر @ از ورودی خوانده شود، دستور break موجب خروج از حلقهٔ بی‌نهایت while می‌گردد. در غیر این صورت حلقه ادامه خواهد یافت.

همان طور که ملاحظه می‌کنید شکل کلی دستور break به صورت زیر است.

```
break ;
```

نحوهٔ عملکرد این دستور در شکل ۵ـ۶ آمده است.



شکل ۵ـ۶ نمودار دستور break

لازم به ذکر است که در حلقه‌های تو در تو، دستور break برای انتقال کنترل به خارج از حلقهٔ داخلی به کار می‌رود.

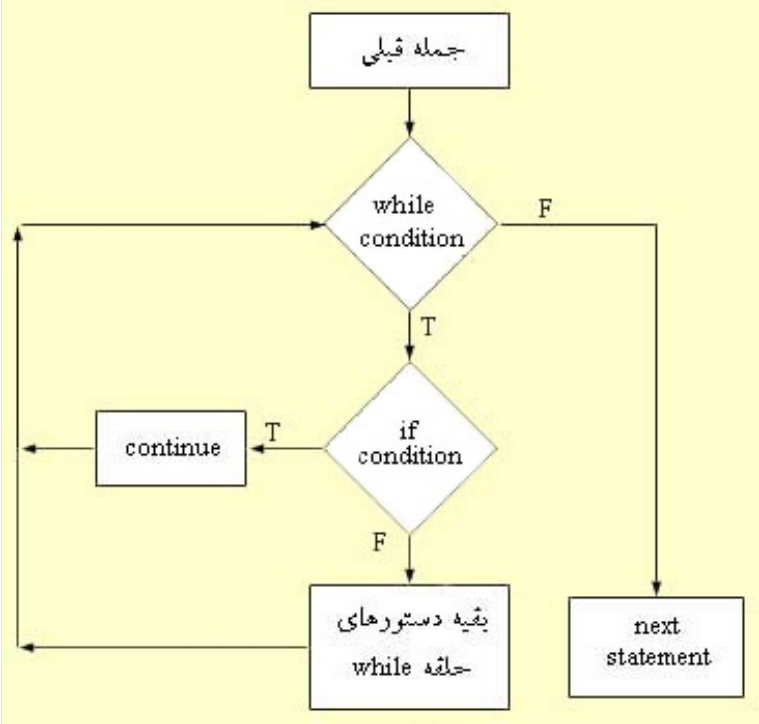
دستور continue

این دستور به منظور عبور جانبی از کنار سایر مراحل جاری حلقه به کار می‌رود. در واقع

باقیماندهٔ تکرار جاری حلقه نادیده گرفته می‌شود و بلافاصله تکرار بعدی حلقه آغاز می‌گردد. حلقه زمانی که به دستور continue می‌رسد خاتمه نمی‌یابد و از این جهت با دستور break تفاوت دارد. این دستور ممکن است با ساختارهای while , do-while , for به کار رود و شکل کلی آن به صورت سادهٔ زیر است.

continue ;

همچنین نحوهٔ عملکرد این دستور به صورت شکل ۵ـ۷ است.



شکل ۵ـ۷ نمودار دستور continue

ـ **مثال ۵ـ۱۶** برنامهٔ زیر با استفاده از دستور continue، n عدد از ورودی می‌خواند و مجموع اعداد مثبت را محاسبه می‌کند.

```
#include<stdio.h>
main ()
```

```
{
  int n , i , x , sum = 0 ;
  printf("how many numbers? ") ;
  scanf("%d", &n) ;
  for (i=1 ; i<n ; + + i)
      {
        scanf ("%d", &x) ;
        if (x<=0)
            continue ;
        sum += x ;
      }
  printf ("\n sum = %d \n", sum) ;
}
```

→

دستور goto

این دستور ترتیب طبیعی اجرای برنامه را تغییر می‌دهد و کنترل را به قسمت دیگری از برنامه منتقل می‌کند. در واقع انتقال کنترل بدون شرط است و اصول برنامه‌سازی ساخت‌یافته در زبان C را لغو می‌کند. از این رو استفاده از آن توصیه نمی‌شود. در برخی زبانهای برنامه‌نویسی قدیمی مثل فورترن و بیسیک کاربرد گسترده‌ای دارد. شکل کلی این دستور به صورت زیر است.

```
goto label ;
```

که در آن label شناسه‌ای است که محل انتقال یا محل دستور بعدی‌ای را که باید اجرا شود نشان می‌دهد. بنابراین، محل انتقال کنترل یا جملهٔ هدف باید label داشته باشد. پس از برچسب علامت دو نقطه به کار می‌رود. شکل آن به صورت زیر است.

```
label:  statement
```

ـ **مثال ۵ـ۱۷** به قطعه برنامهٔ زیر توجه کنید.

```
{
---------
---------
goto msg ;
---------
---------
}
msg:  printf("\n please try again") ;
```

اما گاهی استفاده از دستور goto مفید است و کار را ساده می‌کند. به عنوان مثال در صورتی که نیاز باشد کنترل برنامه از درون دو حلقهٔ تودرتو به خارج انتقال یابد معمولاً ساده‌ترین حالت استفاده از دستور goto خواهد بود.

ـ **مثال ۵ـ۱۸** برنامهٔ زیر عددی صحیح از ورودی می‌خواند. در صورتی که زوج باشد پیغام مناسب را نمایش می‌دهد، در غیر این صورت از برنامه خارج می‌شود.

```
#include<stdio.h>
main ()
  {
    int n ;
    scanf("%d", &n) ;
    if (n % 2 = =0)
      goto even ;
    else
      goto odd ;
    even:  printf("even number.\n") ;
    exit(1) ;
    odd:  exit (0);
  }
```

تابع exit

این تابع اجرای برنامه را به طور کامل قطع می‌کند و کل برنامه را خاتمه می‌بخشد. این تابع که در کتابخانهٔ استاندارد وجود دارد معمولاً با آرگومان صفر فراخوانی می‌شود و بر خاتمه پذیرفتن طبیعی برنامه دلالت می‌کند. اما آرگومانهای دیگری نیز دارد.

کاربرد متداول exit زمانی است که شرط خاصی برای اجرای برنامه برقرار نباشد و اجرای برنامه قطع شود.

ـ **مثال ۵ـ۱۹** به قطعه برنامهٔ زیر توجه کنید.

```
{
    ....
    if x<5
      exit (0);
    ....
```

```
}
```

خودآزمایی ۵

۱. برنامه‌ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را محاسبه و چاپ کند.

۲. برنامه‌ای بنویسید که یک سطر متن را بخواند و تعداد محلهای خالی (blank) موجود در آن را تعیین و چاپ کند.

۳. موجودی ۵۰۰۰۰۰ ریال با سود بانکی ۱۶ درصد در سال داریم. برنامه‌ای بنویسید که برای پنج سال متوالی موجودی را در پایان هر سال محاسبه و چاپ کند.

٤. برنامه‌ای بنویسید که کاراکتری را از طریق ورودی بخواند و تعیین کند که آیا کاراکتر مزبور یکی از کاراکترهای EOF (پایان فایل)، خط جدید، رقم، حروف کوچک، حروف بزرگ یا از کاراکترهای دیگر است.

۵. عددی که مساوی مجموع مقسوم‌علیه‌های خود باشد را عدد کامل نامند، مانند عدد ٦؛ یعنی $6 = 3 + 2 + 1$. برنامه‌ای بنویسید که عددی صحیح و مثبت را بخواند و تعیین کند که آیا این عدد کامل است یا نه.

۶. در ریاضی، سری فیبوناچی به صورت زیر تعریف شده است.

1 1 2 3 5 8 13 ...

که در آن دو جملهٔ اول و دوم برابر ۱ است. در مورد بقیهٔ جمله‌ها هر جمله برابر است با مجموع دو جملهٔ قبلی آن. برنامه‌ای بنویسید تا جمله‌های اول تا ده سری مزبور و مجموع آنها را محاسبه و چاپ کند.

۷. برنامه‌ای بنویسید که چهار عمل اصلی ماشین حساب (جمع، تفریق، ضرب و تقسیم) را انجام دهد.

۸. برنامه‌ای بنویسید که نمرات امتحانی n نفر دانشجو را بخواند و معدل کلاس، نمرهٔ امتحانی شاگرد اول و شاگرد آخر کلاس را تعیین و چاپ کند.

۹. برنامه‌ای بنویسید که مجموع اعداد زوج و نیز مجموع اعداد فرد ۱ تا n را محاسبه و چاپ کند.

فصل ۶

توابع و کلاس حافظه

هدف کلی

آشنایی با توابع، پارامترهای خط فرمان، آرگومانهای argc و argv و حافظه‌های C

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از مطالعهٔ این فصل،

۱. تابع و انواع آن را تعریف کند.

۲. مزایای تعریف تابع فرعی را بیان کند.

۳. عناصر تشکیل‌دهندهٔ هر تابع را بشناسد.

٤. کاربرد دستور return را بیان کند.

۵. نحوهٔ فراخوانی تابع را بشناسد.

۶. روش انتقال آرایه به تابع را بیان کند.

۷. روشهای تعریف پارامتری که اشاره‌گر آرایه دریافت می‌کند بیان کند.

۸. توابع بازگشتی یا خودگردی را بشناسد.

۹. پارامترهای خط فرمان را بشناسد.

۱۰. آرگومانهای argc و argv را بشناسد.

۱۱. تفاوت فراخوانی توابع در C و پاسکال را بیان کند.

۱۲. متغیرهای محلی یا خصوصی و متغیرهای عمومی را بشناسد.

۱۳. کلاس حافظه و کاربرد هریک را شرح دهد.

مقدمه

در حالت کلی و بدون توجه به زبان برنامه‌نویسی خاص، برنامهٔ مورد نظر برای پیاده‌سازی هر پیمانه یا ماژول را زیربرنامه گویند. در زبان پاسکال به آنها تابع یا رویه گویند. پیمانه یا ماژول را در زبان C تابع نامند؛ یعنی برای پیاده‌سازی هر ماژول می‌توان یک تابع نوشت. تابع قطعه برنامه‌ای کامل است که کار معینی انجام می‌دهد و موجب جلوگیری از برنامه‌نویسی تکراری در بین برنامه‌ها می‌شود.

گفتیم هر برنامه در زبان C، مجموعه‌ای از یک یا چندین تابع است که یکی و فقط یکی از آنها تابع اصلی یا main است و بقیه تابع فرعی‌اند. توابع به دو گروه دسته‌بندی می‌شوند. یک سری توابع از پیش تعریف شده‌اند و آنها را توابع کتابخانه‌ای[1] نامند. کتابخانهٔ استاندارد C، مجموعه‌ای غنی از این گونه توابع دارد که محاسبات ریاضی، انجام عملیات روی نوشته‌ها و کاراکترها، انجام عملیات ورودی و خروجی، انجام عملیات در زمینه‌های گرافیکی و جز آن را به عهده دارند که متداول‌ترین آنها را بررسی می‌کنیم. توربو C و همین طور گونه‌های دیگر C، از این لحاظ کتابخانه‌های گسترده‌تری دارند که آنها را نیز به اختصار بررسی می‌کنیم. وجود این گونه توابع از پیش تعریف شده کار برنامه‌نویسان را ساده‌تر می‌کند و توانمندی آنها را برای نوشتن برنامه‌های کارآمد بالا می‌برد.

گروه دیگر از توابع فرعی را برنامه‌نویس تعریف می‌کند. C، مشابه سایر زبانهای برنامه‌نویسی، اجازه می‌دهد که برنامه‌نویس، برحسب سلیقهٔ خود، زیربرنامه‌هایی را به عنوان تابع فرعی طراحی کند و آنها را برحسب نیاز در برنامه یا تابع اصلی و یا سایر توابع فراخواند. این شیوه، به برنامه‌نویس امکان می‌دهد که برنامه‌ای بزرگ را به صورت ماژول‌شده طراحی کند؛ یعنی کار یا پروژهٔ مورد نظر را به قسمتهای کوچک‌تری تجزیه کند، برای هر قسمت یک تابع بنویسد و آنها را به کمک تابع اصلی به یکدیگر پیوند دهد و برحسب لزوم آنها را فراخواند.

طراحی برنامه به صورت پیمانه‌ای یا ماژول‌شده، یعنی به صورت مجموعه‌ای از توابع، مزایایی دارد که مهم‌ترین آنها عبارت‌اند از:

1. library functions

ـ امکان استفادهٔ مجدد از تابع
ـ ساده شدن کنترل و خطایابی
ـ امکان انجام تغییرات در برنامه و اصلاح آن
ـ امکان همکاری برنامه‌نویسان متعدد در نوشتن یک برنامه.

در این فصل ایجاد و کاربرد توابع را بررسی می‌کنیم.

نحوهٔ تعریف تابع

چنانکه گفتیم هر برنامهٔ C از یک یا چندین تابع تشکیل می‌گردد که فقط یکی از آنها تابع اصلی است. به طور کلی هر تابع شامل عناصر زیر است:

– عنوان تابع[1] که شامل نام تابع، نوع تابع و آرگومانهای تابع است. تابع باید از لحاظ نوع توصیف شود که قبل از نام آورده می‌شود و مشخص می‌کند که تابع چه نوع مقداری را برمی‌گرداند. در صورتی که تابع آرگومان داشته باشد به دنبال نام تابع و درون یک جفت پرانتز، آرگومانهای تابع درج می‌شود. همچنین نوع هر آرگومان نیز باید اعلان شود. آرگومانها با ویرگول از هم جدا می‌شوند. اگر تابع مقدار صحیح برمی‌گرداند، تعریف نوع را می‌توان حذف کرد. در صورتی که تابع آرگومان نداشته باشد، یک جفت پرانتز خالی به دنبال نام آن می‌آید.

– بدنهٔ تابع که شامل موارد زیر است:

الف) اعلان متغیرهای محلی[2]. هر متغیری را که جزء آرگومانهای تابع نباشد و به عبارت دیگر مخصوص خود تابع باشد متغیر محلی نامند و باید در آغاز تابع پس از عنوان تابع اعلان گردد. متغیرهای محلی خارج از تابع شناخته نمی‌شوند.

ب) سایر دستورهای تابع. متن تابع و دستورهای اجرایی در این قسمت واقع می‌شود و شامل دستورهای محاسباتی، دستورهای کنترلی و حتی دستور فراخوانی خود تابع است. شکل ٦ـ١ عناصر یک تابع را نمایش می‌دهد.

وقتی که تابعی در داخل تابع اصلی فراخوانی می‌شود، اصطلاح آرگومان به کار می‌رود.

1. function heading
2. local variables declaration

شکل ۶ـ۱ عناصر اصلی تابع

ولی در خود تابع فرعی، یعنی در تعریف تابع فرعی، اصطلاح پارامتر به کار می‌رود.

دستور return

برای خروج از تابع از دستور return استفاده می‌شود. اگر نیاز باشد که تابع مقداری را برگرداند دستور return دارای آرگومان خواهد بود که اول مقدار آرگومان آن به نام تابع اختصاص می‌یابد، سپس کنترل از تابع فرعی (تابع فراخوانده شده) به تابع اصلی (تابع فراخوانندهٔ آن) برمی‌گردد.

در صورتی که تابع مقداری را برنگرداند و نقش آن فقط انجام عمل خاصی باشد، دستور return آرگومان نخواهد داشت و نقش آن فقط خروج از تابع و انتقال کنترل به تابع فراخوانندهٔ آن خواهد بود.

ـ **مثال ٦ـ۱** قطعه برنامهٔ زیر تابع فرعی F1 را نشان می‌دهد که مقداری را به تابع اصلی (برنامهٔ فراخوانندهٔ) آن برمی‌گرداند. در اینجا مقدار x که برابر 5 است به تابع اصلی برگردانده می‌شود و در آنجا به متغیر a اختصاص می‌یابد و سپس با دستور printf مقدار 5 در خروجی چاپ می‌شود.

تابع فرعی

```
F1()
  {.
     x = 5 ;
     retun (x) ;
  }
```

تابع اصلی

```
main()
    {....
       a = F1() ;
       printf("%d", a) ;
    }
```

→

دستور return تنها می‌تواند شامل یک عبارت باشد. یعنی می‌تواند فقط یک مقدار را به تابع فراخوانندهٔ آن برگرداند. چنانچه نیاز باشد که بیش از یک مقدار به تابع فراخواننده بازگردانده شود، باید از اشاره‌گرها استفاده کرد.

تعریف تابع شامل چند دستور return است که هر کدام یک عبارت متفاوت دارند. توابعی که دارای انشعابهای چندگانه‌اند اغلب به چند return احتیاج پیدا می‌کنند.

ـ **مثال ٦ـ٢** تابع زیر ماکزیمم دو عدد صحیح را به دست می‌آورد و در خروجی چاپ می‌کند.

```
Maximum(int x , int y)
{ int z ;
     z = (x>=y) ? x:  y ;
     printf("%d" , z) ;
     return ;
}
```

در این تابع می‌توان دستور return را حذف کرد اما در برنامه‌نویسی صحیح و اصولی این کار انجام نمی‌گیرد. اگر تابعی بدون آنکه به دستور return برخورد کند، به انتها برسد کنترل به سادگی به دستور فراخوانندهٔ برنامه برمی‌گردد، بدون آنکه هیچ اطلاعاتی برگرداند. در این گونه موارد توصیه می‌شود که از یک دستور return خالی استفاده شود تا هم منطق برنامه واضح باشد و هم اینکه با تغییرات بعدی در تابع تطبیق پیدا کند.

ـ **مثال ٦ـ٣** برنامهٔ زیر تابعی را نشان می‌دهد که کاراکتری را از ورودی می‌خواند و اگر از حروف بزرگ باشد به حرف کوچک برمی‌گرداند. در غیر این صورت، خود کاراکتر دریافت شده برگردانده می‌شود.

```
char UptoLow (void)
 {
    char ch ;
    ch = getchar() ;
    if (ch > 64 && ch < 91)
         return (ch + 32) ;
    else
         return (ch) ;
 }
```

دستور return فقط در پایان تابع به کار نمی‌رود بلکه هر کجا در منطق برنامه لازم باشد

به کار می‌رود. همان طور که ملاحظه می‌کنید در این تابع دو بار از دستور return استفاده شده، یکبار در انتهای تابع و یکبار هم در وسط برنامه. از طرفی می‌دانیم که کد اسکی حروف بزرگ از ٦٥ تا ۹۰ است (٦٥ برای A و ۹۰ برای Z) و کد اسکی حروف کوچک نیز از حروف همنام خود ۳۲ واحد بیشتر است. حال در این تابع پس از خواندن کاراکتر ورودی، ابتدا بررسی می‌شود که آیا کد اسکی حرف خوانده شده در فاصلهٔ باز ٦٥ تا ۹۱ است یا نه؟ درصورت مثبت بودن پاسخ مقدار ۳۲ واحد به کد اسکی آن حرف افزوده می‌گردد تا حرف مزبور به حرف کوچک همنام خود تبدیل شود و نتیجه به تابع فراخوانندهٔ آن برگردانده شود. در غیر این صورت، همان حرف بدون تغییر به تابع فراخواننده بازگردانده می‌شود.

ـ **مثال ٦ـ٤** برنامهٔ زیر عددی صحیح از ورودی می‌خواند، سپس با فراخواندن تابعی به نام fact، فاکتوریل عدد مزبور محاسبه و در خروجی چاپ می‌شود.

```
#include<stdio.h>
main ()
{
  printf("n=");
   scanf("%d , &n) ;
   fact(n) ;
}
void fact (int n)
{
   int f = 1 , i ;
   for (i =2 ; i<=n ; ++i)
      f = f * i ;
   printf ("fact (n)= %d" , f) ;
}
```

در اینجا که تابع مقداری را برنمی‌گرداند تابع void تعریف شده، همچنین در فراخوانی تابع، فقط نام تابع در یک دستور ساده به کار رفته است. در این تابع نیازی به دستور return نیست؛ یعنی با رسیدن به انتهای تابع، کنترل به طور خودکار به تابع فراخواننده که در اینجا تابع اصلی است برمی‌گردد، ولی می‌توان دستور return را نیز در انتهای تابع، یعنی قبل از آکولاد پایانی تابع، قرار داد که در این صورت دستور مزبور بدون آرگومان به کار می‌رود.

برنامهٔ فوق را می‌توان به طریقی نوشت که فاکتوریل عدد ورودی را تابع فرعی برگرداند

و در تابع اصلی چاپ شود. در این صورت برنامهٔ مورد نظر بدین صورت خواهد بود.

```
#include<stdio.h>
main ()
{
   int n ;
   long int factorial ;
   printf("n = ") ;
   scanf ("%d" , &n) ;
   factorial = fact(n) ;
   printf ("\n fact (n) = %d" , factorial) ;
}
int fact (int n)
{
   int i ;
   long int f = 1 ;
   if (n >1)
      for (i = 2 ; i<=n ; ++i)
         f = f * i ;
   return(f) ;
}
```

در اینجا تابع باید مقداری را برگرداند لذا نوع مقداری که باید تابع برگرداند مشخص شده است و در این مثال از نوع int است. در ضمن نام تابع در تابع اصلی، در طرف راست یک دستور جایگذاری به کار رفته است. در تابع فرعی نیز دستور return دارای آرگومان است و اول مقدار آرگومان دستور return به نام تابع نسبت داده می‌شود سپس کنترل به تابع اصلی برمی‌گردد.

در این برنامه می‌توان فراخوانی تابع را در دستور printf نیز به صورت زیر انجام داد.

```
printf (" fact(n) = %d" , fact(n)) ;
```

بنابراین سه صورت فراخوانی تابع، یعنی دستور ساده، دستور انتساب و نیز در درون دستور خروجی، را در این برنامه ملاحظه کردید.

در زبان C، اگر نوع مقداری که تابع برمی‌گرداند به طور صریح مشخص نشده باشد، پیش‌فرض آن است که تابع مزبور مقدار صحیح برمی‌گرداند. اگر نوع مقداری که تابع برمی‌گرداند غیر از مقدار صحیح باشد، باید دو عمل زیر انجام گیرد.

اول، نوع تابع (یعنی نوع مقداری را که تابع برمی‌گرداند) به طور صریح مشخص گردد.

دوم، قبل از هر فراخوانی تابع، باید نوع آن مانند متغیر معمولی در تابع فراخوانده شده توصیف گردد و یا اینکه نوع تابع به صورت سراسری یا عمومی قبل از همهٔ توابعی که آن را فراخواهند خواند و به طور متعارف قبل از تابع main مشخص و اعلان شود.

ـ **مثال ٦ـ٥** در برنامهٔ زیر تابعی تعریف شده که حاصل جمع دو عدد را به تابع اصلی بازمی‌گرداند.

```
main ()
 {
    float sum (float , float) ;
    float x , y ;
    scanf("%f %f" , x , y) ;
    printf ("%f , sum (x , y)) ;
}
float sum (a , b)
    float a , b ;
    {
       return (a + b) ;
    }
```

همان طور که ملاحظه می کنید تابع sum از نوع اعشاری و با آرگومانهای اعشاری تعریف شده است. همچنین اعلان تابع پس از main آمده است. اما تابع مزبور می‌تواند قبل از تابع فراخواننده main نیز توصیف گردد که در این صورت برنامهٔ فوق به صورت زیر خواهد بود.

```
float sum ()
main ()
 {
   float x , y ;
   scanf("%f %f" , x , y) ;
   printf ("%f , sum(x , y)) ;
}
float sum(a , b)
   float a , b ;
 {
   return (a + b) ;
 }
```

همان طور که می‌بینید در تعریف تابع sum قبل از تابع main، آرگومانهای آن مشخص نشده است. در این حالت در بیشتر نسخه‌های زبان C نیازی نیست که نوع آرگومانها را در

تعریف تابع تعیین کرد.

فراخوانی تابع

به طور کلی در زبان C فراخوانی یک تابع با نوشتن نام تابع و پارامترهای آن به صورت تک دستور و یا در دستور جایگذاری انجام می‌شود. در صورت نداشتن پارامتر زوج، پرانتز خالی به کار می‌رود. راه دیگر برای فراخوانی تابع آن است که نام تابع در عبارت محاسباتی یا دستور خروجی به کار رود. به طور کلی اگر تابعی به صورت void اعلان شود، به صورت عملوند در هر عبارت قابل قبول در زبان C به کار می‌رود. در مثال زیر نام تابع در دستور خروجی به کار رفته است.

ـ **مثال ٦ـ٦** فاکتوریل عدد صحیح n را به دست آورید.

```
#include<stdio.h>
main ()
{
  int n ;
  scanf ("%d" , &n) ;
  printf ("n = %d fact (n) = %d" , n , fact (n)) ;
}
int fact (int n)
{
    int f = 1 , i ;
    for (i =2 ; i<= n ; ++i)
        f = f * i ;
    return (f) ;
}
```

توابع، ازنظر نحوهٔ انتقال اطلاعات از یک تابع فراخواننده به تابع فراخوانده شده و یا به عبارت دیگر ازنظر نحوهٔ انتقال آرگومانها، به دو روش فراخوانی می‌شوند:

– فراخوانی با مقدار یا فراخوانی با ارزش[1]

1. call by value

– فراخوانی با آدرس[1] یا فراخوانی با ارجاع[2].

در روش اول خود متغیرها یا آرگومانها به تابع فراخوانده شده (تابع فرعی) انتقال نمی‌یابد، بلکه فقط مقدار آنها یا به عبارت دیگر کپی آنها به تابع فراخوانده‌شده انتقال می‌یابد. بنابراین هر عملی که تابع فرعی روی نسخهٔ دریافت شده از آرگومانها انجام دهد، در تابع فراخوانندهٔ آن منعکس نمی‌گردد؛ یعنی مقدار آن آرگومانها در تابع فراخواننده تغییر نخواهد کرد.

ـ **مثال ٦ـ٧** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main ()
{
   int n ;
   long int factorial ;
   printf("n = ") ;
   scanf ("%d" , &n) ;
   factorial = fact(n) ;
   printf ("\n fact (n) = %d" , factorial) ;
}
int fact (int n)
{
  long int f = 1 ;
   while (n >1)
      f *= n-- ;
   return(f) ;
}
```

fact را درنظر بگیرید. تابع مزبور عدد صحیح n را به عنوان آرگومان دریافت و فاکتوریل آن را محاسبه می‌کند و برمی‌گرداند. در اینجا درواقع دو حافظه با نام n در برنامه وجود دارد: یکی در تابع اصلی که مقدار n را از طریق دستور ورودی scanf دریافت می‌کند و دیگری در تابع فرعی که نسخه‌ای از همان مقدار را دارد. وقتی که تابع فرعی عملیاتی روی n انجام می‌دهد در واقع این عملیات در روی متغیر n در تابع فرعی و یا روی محتوای حافظه‌ای که با نام n در تابع فرعی پیش‌بینی شده است انجام می‌گیرد و هیچ‌گونه تأثیری در مقدار n در

1. call by address
2. call by reference

تابع اصلی ندارد. به همین دلیل این گونه فراخوانی را فراخوانی با مقدار گویند.

ـ

ـ **مثال ٦ـ٨** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main ()
{ int x = 2 ;
   printf (" %d" , x) ;
   calc (x) ;
   printf ("\n %d" , x) ;
}
void calc (int x)
{
   printf ("\n%d " , x);
   x ++ ;
   printf ("\n %d " , x) ;
}
```

خروجی برنامهٔ مزبور به صورت زیر خواهد بود.

```
2
2
3
2
```

نحوهٔ عملکرد برنامهٔ مزبور به این طریق است که با اجرای اولین دستور printf در تابع اصلی، مقدار x در تابع اصلی، یعنی 2، چاپ می‌گردد (سطر اول خروجی). سپس با فراخوانی تابع فرعی یک نسخه از مقدار x به عنوان آرگومان به تابع فرعی گذر داده می‌شود و با اجرای اولین دستور printf در تابع مزبور در سطر دوم خروجی 2 چاپ می‌گردد. حال با اجرای دستور ; x ++ در تابع مزبور، به مقدار x یک واحد افزوده می‌شود. در واقع اکنون مقدار جدید x در تابع فرعی 3 است که با اجرای دستور printf دوم در آن تابع، در سطر سوم خروجی بالا چاپ می‌شود. سپس کنترل به تابع اصلی برمی‌گردد و در تابع اصلی، دستور printf دوم اجرا می‌گردد. با اجرای دستور مزبور، خروجی سطر چهارم ظاهر خواهد شد.

در واقع دو حافظه با نام x وجود دارد که یکی در تابع اصلی و دیگری در تابع فرعی است. هر بار در تابع اصلی به x مراجعه شود، مقدار x (محتوای خانه‌هایی از حافظه که به نام x است) در تابع اصلی چاپ می‌شود و هر موقع در تابع فرعی به x مراجعه شود، محتوای

حافظه‌هایی که در تابع فرعی برای مقدار x درنظر گرفته شده است چاپ می‌گردد.

همچنین توجه داشته باشید آرگومانهایی که به تابع فرعی فرستاده می‌شوند، متغیرهای مجازی یا متغیرهای مستعار نامیده می‌شوند و اسامی آنها در تابع فرعی ممکن است غیر از آنچه باشد که در تابع اصلی نامگذاری شده است. بنابراین برنامهٔ مزبور را می‌توان به صورت زیر نیز نوشت.

```
#include<stdio.h>
main ()
{ int x = 2 ;
   printf (" %d" , x) ;
   calc (x) ;
   printf ("\n d" , x) ;
}
void calc (int k)
{
   printf ("\n%d" , k);
   x ++ ;
   printf ("\n %d " , k) ;
}
```

ملاحظه می‌کنید که در این روش فراخوانی تابع مقدار آرگومان، به پارامتر تابع فراخوانده شده، که آن را پارامتر رسمی یا فرمال نیز گویند، کپی می‌شود. چنانکه به یاد دارید، در فراخوانی تابع متغیرهای انتقالی در تابع فراخوانندهٔ آرگومان و در تابع فراخوانده شده پارامتر نامیده می‌شود.

روش فراخوانی با مقدار از انتقال اطلاعات توسط آرگومانها به جزء فراخوانندهٔ برنامه جلوگیری می‌کند. به این دلیل، انتقال با مقدار برای انتقال اطلاعات روش محدودی است.

در روش دوم فراخوانی تابع، آدرس آرگومان به درون پارامتر تابع فراخوانده شده کپی می‌شود و این آدرس در درون تابع فراخوانده شده، برای دسترسی به پارامترهای حقیقی، به کار می‌رود. از این رو انجام هرگونه عملیات یا تغییرات در روی این پارامترها، بر روی آرگومانهای متناظر آنها در تابع فراخواننده نیز همان اثر را خواهد داشت. پس، از آنجایی که در این روش آدرس آرگومانها به درون تابع فراخوانده شده عبور داده می‌شود، می‌توان آن آرگومان را خارج از تابع فراخواننده، یعنی در درون تابع فراخوانده شده، تغییر داد.

انتقال آرایه به تابع

انتقال آرایه به عنوان آرگومان به یک تابع، استثنا بر شکل استاندارد فراخوانی، با مقدار است، زیرا در اینجا فقط آدرس آرایه به تابع گذر می‌کند نه کپی تمام عناصر آرایه. وقتی که تابعی با آرگومانی از آرایه فراخوانده می‌شود اشاره‌گر به اولین عنصر در آرایه (یعنی آدرس اولین عنصر آرایه) به تابع گذر می‌کند. توجه داشته باشید که در C نام یک آرایه به تنهایی، یک اشاره‌گر به اولین عنصر آرایه است.

برای تعریف پارامتری که اشاره‌گر آرایه را دریافت می‌کند سه روش وجود دارد که به شرح زیر بیان می‌شود.

روش اول. در این روش، پارامتر مورد نظر به صورت آرایه تعریف می‌گردد، مانند مثال زیر.

```
#include<stdio.h>
void display (int num[10]) ;
main()
{
      int a[10] , i ;
      for (i = 0 ; i<10 ; + +i)
        a [i] = i ;
      display(a) ;
}
void noor(num)
int num[10] ;
{
      int i ;
      for (i=0 ; i<10 ; + +i)
        printf ("\n %d" , num[i]) ;
}
```

در اینجا پارامتر num به صورت آرایه‌ای ۱۰ عنصری از نوع int توصیف شده است اما C، به طور خودکار، آن را به اشاره‌گری با مقدار صحیح تبدیل می‌کند، زیرا هیچ پارامتری نمی‌تواند تمامی یک آرایه را دریافت کند. فقط یک اشاره‌گر به یک آرایه گذر داده می‌شود. بنابراین باید یک پارامتر اشاره‌گر آن را دریافت کند.

روش دوم. در این روش، برای توصیف یک پارامتر آرایه، آن را به عنوان آرایه‌ای معرفی می‌کنیم که اندازه یا تعداد خانه‌های آن مشخص نشده باشد، مانند تابع زیر.

```
void noor (num)
int num[ ] ;
{
     for (i= 0 ; i< 10 ; + +i)
       printf("\n%d" , num[i]) ;
}
```

که در این تابع num به صورت آرایه‌ای از نوع int و با اندازهٔ نامعلوم معرفی شده است. این روش در واقع num را به عنوان اشاره‌گر int تعریف می‌کند.

روش سوم. راه سوم آن است که num را به عنوان اشاره‌گر int تعریف کنیم که این روش مناسب‌ترین و حرفه‌ای‌ترین روش برای انجام این کار است. این روش در زیر نشان داده شده است.

```
void noor (num)
int *num ;
{
     int i ;
     for (i =0 ; i<10 ; + +i)
       printf ("%d" , num[ i ] );
}
```

که در آن علامت "*" عملگر اشاره‌گر است. (اشاره‌گرها را در فصل ۸ بررسی می‌کنیم.)

یک نکتهٔ مهم در اینجا آن است که وقتی آرایه‌ای آرگومان یک تابع باشد، آدرس آن به تابع گذر داده می‌شود. این حالت در زبان C، یک استثنا بر فراخوانی با مقدار، در رابطه با قرارداد گذردادن پارامتر، محسوب می‌گردد. بنابراین در مورد آرایه‌ها، عملیاتی که تابع فرعی روی آرایه انجام می‌دهد، در روی محتوای خود آرایه خواهد بود؛ یعنی در اینجا دیگر نسخه‌ای از آرایه به تابع انتقال نمی‌یابد. پس نتیجهٔ عملیات تابع روی آرایه‌ها، در تابع اصلی (تابع فراخواننده) نیز منعکس خواهد شد.

توابع بازگشتی

بازگشتی بودن فرآیندی است که در آن یک تابع به طور مکرر خودش را فراخوانی می‌کند تا آنکه به شرط خاصی برسد. استفاده از این روش جهت انجام محاسبات تکراری که در آن منطقی وجود دارد مناسب است. از نظر ریاضی، یک تابع بازگشتی یا خودگردی تابعی است

که برحسب خودش تعریف می‌شود. به عنوان مثال فاکتوریل عدد صحیح و مثبت m، که آن را با !m نمایش می‌دهند، به شکل متداول به صورت زیر تعریف می‌گردد.

m! = m(m-1)(m-2) …. 3. 2. 1

در ضمن فاکتوریل صفر برابر ۱ است.

راه دیگر برای تعریف فاکتوریل که روش بازگشتی است این است که اگر m = ۱، خروج؛ در غیر این‌صورت m(m-1)!.

ملاحظه می‌کنید که در روش دوم، در صورتی که m بزرگ‌تر از یک باشد، فاکتوریل آن به طور مستقیم محاسبه نمی‌شود، بلکه برحسب فاکتوریل عدد (m-1) تعریف می‌شود. مثلاً فاکتوریل عدد ۴ به صورت 4! = 4 * 3! تعریف می‌شود؛ یعنی تابع دوباره برحسب خودش تعریف می‌شود. به عبارت دیگر تعریف تابع روی خود تابع برمی‌گردد. بنابراین فاکتوریل ۴ دوباره به صورت 4! = 4 * 3! تعریف می‌شود. این روش ادامه پیدا می‌کند تا اینکه آرگومان تابع به ۱ برسد که در این حالت پاسخ آن برابر ۱ خواهد بود. در اینجاً باید به عقب برگشت و مقادیر ایجاد شده را در یکدیگر ضرب کرد تا در پایان فاکتوریل عدد ۴ به دست آید.

از نظر برنامه‌نویسی، در صورتی تابع را بازگشتی نامند که به طور مستقیم یا غیرمستقیم تابع خود را فراخوانی کند. در بعضی زبانهای برنامه‌سازی مانند بیسیک و فورترن، روش تعریف تابع به صورت بازگشتی پیش‌بینی نشده است؛ یعنی امکان خودفراخوانی تابع وجود ندارد. در بعضی از زبانها این امکان پیش‌بینی شده، ولی هنگام تعریف تابع، باید به طور صریح مشخص گردد که تابع به صورت بازگشتی تعریف می‌گردد تا کامپایلر تابع مزبور را تابع بازگشتی بشناسد. زبان برنامه‌نویسی PL/I از این گروه است. در زبان پاسکال هم استفاده از توابع بازگشتی امکان پذیر است. در زبان C، همهٔ توابع می‌توانند به صورت بازگشتی به کار برده شوند و نیاز نیست که‌این عمل به طور صریح بیان گردد؛ یعنی تعریف یا اعلان خاصی نیاز نیست. ایدهٔ تابع بازگشتی در شکل اولیه‌اش ساده است.

ـ **مثال ٦ـ٩** به‌این برنامه توجه کنید.

```
main()
  {
    printf("\n HELLO ") ;
    main()
  }
```

ملاحظه می‌کنید که این برنامه پایان‌ناپذیر است و بی‌نهایت بار اجرا می‌شود؛ یعنی تابع اصلی مرتب خودش را فراخوانی می‌کند و در هر فراخوانی عبارت " HELLO " چاپ می‌گردد. واضح است که‌این نوع خودفراخوانی توابع مطلوب نیست؛ یعنی باید تعداد دفعات خودفراخوانی متناهی باشد تا مانند مثال بالا با حلقهٔ بدون پایان مواجه نشویم.

ـ **مثال ٦ـ۱۰** تابع زیر مجموع اعداد طبیعی ۱ تا n را به شکل بازگشتی محاسبه می‌کند.

```
int sum (int n)
 {
   if (n<=1)
        return (n) ;
   else
        return (n+sum (n-1)) ;
 }
```

نحوهٔ عملکرد این تابع در بعضی مقادیر n در جدول زیر تجزیه، تحلیل و نمایش داده شده است.

فراخوانی تابع	مقدار بازگشتی
sum(1)	1
sum(2)	2+sum(1) ➔ 2+1
sum(3)	3+sum(2) ➔ 3+2+1
sum(4)	4+sum(3) ➔ 4+3+2+1

ـ **مثال ٦ـ۱۱** برنامهٔ زیر، با استفاده از تابع بازگشتی، فاکتوریل اعداد صفر تا ۱۰ را محاسبه می‌کند و در خروجی نشان می‌دهد.

```
#include <stdio.h>
long fact (long) ;
main()
  {
    int i ;
    for (i=0 ; i<=10; i + +)
       printf(" %d! = %ld \n" , i , fact (i)) ;
  }
long fact (long n)
  {
    if (n < =1)
```

```
    return 1 ;
  else
    return (n * fact (n - 1));
}
```

خروجی برنامه

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

پارامترهای خط فرمان

در زبان C بعضی مواقع ضرورت دارد که هنگام اجرای برنامه اطلاعاتی را به برنامه انتقال دهیم. روش کلی آن است که اطلاعات مورد نظر را با به کار بردن آرگومانهای خط فرمان به تابع main انتقال دهیم. یک آرگومان خط فرمان اطلاعاتی است که به دنبال نام تابع روی خط فرمان سیستم‌عامل می‌آید. برای مثال وقتی که برنامه‌ای با استفاده از خط فرمان توربو C ترجمه می‌شود دستورهای زیر تایپ می‌گردد.

```
>tcc program-name
```

که در آن program-name نام برنامه‌ای است که باید ترجمه شود. نام برنامه به عنوان یک آرگومان به توربو C انتقال داده می‌شود.

حال اگر بخواهید برنامه را با به کار بردن کامپایلر خط فرمان بورلند C ترجمه کنید، دستورها به‌این شکل تایپ می‌شوند:

```
>bcc program-name
```

برای دریافت آرگومان خط فرمان، دو آرگومان مخصوص از پیش تعریف شده به اسامی argc و argv به کار می‌رود. تا به حال در تمام توابعmain که به کار برده شد، چیزی در داخل زوج پرانتز که پس از نام تابع می‌آید به کار برده نشد؛ یعنی، پرانتزها توخالی بودند، به

صورت main () .

ممکن است این پرانتزها شامل آرگومانهای خاصی نیز باشند که اجازه می‌دهند پارامترهایی از سیستم عامل به تابع اصلی انتقال یابد. اغلب گونه‌های C اجازه می‌دهد که دو آرگومان که به صورت سنتی argc و argv نامیده می‌شوند به کار برده شوند. اولین آرگومان، یعنی argc، باید متغیری از نوع صحیح باشد. درحالی که دومی، یعنی argv، آرایه‌ای از اشاره‌گرهایی به کاراکترهاست. در واقع آرایه‌ای از رشته‌هاست. هر رشته در این آرایه به یک پارامتر دلالت خواهد داشت که به تابع main انتقال یافته است. مقدار argc، تعداد پارامترهایی را نشان می‌دهد که به تابع main انتقال یافته است.

ـ **مثال ٦ـ١٢** برنامهٔ زیر نحوهٔ تعریف پارامترهای argc و argv را در تابع main نشان می‌دهد.

```
main (argc , argv)
int argc ;
char *argv[ ] ;
{
     …..
}
```

در نسخه های جدید C می‌توان آن را به شکل فشرده‌تر زیر نوشت.

```
main (int argc , char *argv[ ])
 {
   …..
 }
```

اجرای یک برنامه معمولاً با مشخص ساختن نام برنامه (در واقع نام فایلی که دربردارندهٔ برنامهٔ هدف، ترجمه شده است) در سطح سیستم عامل آغاز می‌گردد. نام برنامه به عنوان فرمان سیستم‌عامل تفسیر می‌گردد. بنابراین سطری که نام برنامه در آن ظاهر می‌شود، معمولاً خط فرمان است.

برای اینکه بتوان هنگام آغاز اجرای برنامه، یک یا چندین پارامتر را به آن انتقال داد، باید در خط فرمان، پارامترها نیز به دنبال نام برنامه بیایند، مانند دستورهای زیر.

program-name parameter1 parameter2……parameter m

پارامترها باید با فضای خالی یا tab از یکدیگر جدا گردند. بعضی سیستمهای عامل

اجازه می‌دهند که فضای خالی نیز در یک پارامتر به کار رود که در چنین حالتی تمامی پارامترها درون گیومه قرار می‌گیرند.

نام برنامه اولین پارامتر در argv ذخیره می‌شود که به دنبال آن سایر پارامترها می‌آیند. بنابراین اگر به دنبال نام برنامه m پارامتر بیاید، در argv به تعداد (m+1) آرایه وجود خواهد داشت که از argc[0] شروع و به argv[m] خاتمه می‌یابد. به argc نیز به طور خودکار مقدار (m+1) نسبت داده می‌شود. توجه داشته باشید که مقدار argc به طور صریح در خط فرمان تعیین نمی‌گردد.

ـ **مثال ٦ـ١٣** برنامهٔ زیر را درنظر بگیرید.

```
#include<stdio.h>
main (int argc , char *argv[ ])
 {
    int count ;
    printf("argc = %d\n" , argc):
    for (count = 0 ; count<argc ; + +count)
      printf ("argv[%d] = %s\n" , count , argv[count]) ;
 }
```

این برنامه اجازه می‌دهد که بتوان به تعداد دلخواه پارامتر (یعنی بدون نیاز به تعیین تعداد) از خط فرمان وارد کرد. وقتی که برنامه اجرا می‌گردد، مقدار جاری argc و عناصر (مقادیر) argv در خطوط جداگانه‌ای در خروجی ظاهر خواهد شد.

ـ **مثال ٦ـ١٤** فرض کنید که نام برنامه sample و خط فرمان که اجرای برنامه را شروع می‌کند به صورت زیر باشد.

```
sample red white blue
```

اجرای برنامهٔ مزبور خروجی زیر را نتیجه می‌دهد.

```
argc = 4
argv [0] = sample.exe
argv [1] = red
argv [2] = white
argv [3] = blue
```

در این خروجی چهار پارامتر جداگانه از طریق خط فرمان وارد شده است. پارامتر اول، نام برنامه است؛ یعنی sample.exe که به دنبال آن سه پارامتر red، white و blue آمده است.

هرکدام از عناصر مزبور یکی از عناصر آرایه argv است (توجه داشته باشید که sample.exe اسم فایل هدف است که از ترجمهٔ برنامهٔ مبنای sample.c# به دست آمده است.)

به روش مشابه، اگر خط فرمان به صورت sample red "white blue" باشد، خروجی حاصل به صورت زیر خواهد بود.

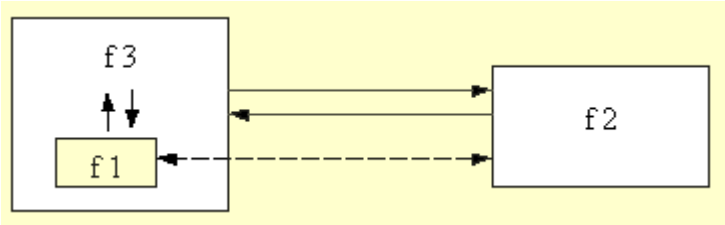
```
argc = 3
argv [0] = sample.exe
argv [1] = red
argv [2] = white
```

در این حالت، رشتهٔ "white blue" به علت وجود گیومه، پارامتر ساده تعبیر خواهد شد؛ یعنی هرکدام از دو رشتهٔ white و blue پارامتر مستقل درنظر گرفته نخواهند شد.

همین که پارامترها به طریق بالا وارد شدند، می‌توانند برحسب نیاز، در برنامه به کار روند. یکی از کاربردهای متداول آن است که اسامی فایلهای داده‌ها را به عنوان پارامترهای خط فرمان مشخص کرد.

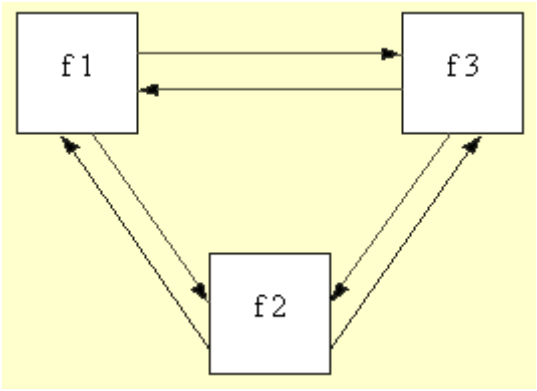
استفاده از چند تابع

در زبان C، می‌توان در هر برنامه به هر تعداد که نیاز باشد تابع تعریف کرد و به کار برد و هر تابع می‌تواند تابع دیگری را فراخوانی کند. در اغلب زبانهای برنامه‌سازی مانند زبان پاسکال توابع به صورت تودرتو تعریف می‌شوند و تابعی که در درون یک تابع تعریف شده نمی‌تواند با تابع دیگر فراخوانی شود. در واقع تابعی را که در درون یک تابع تعریف شده نمی‌توان با توابع دیگر رؤیت کرد. زبان C این طور نیست، یعنی توابع به صورت تودرتو تعریف نمی‌شوند و دستیابی به همهٔ توابع به صورت یکسان است و هر تابع می‌تواند تابع دیگر را فراخوانی کند. برای اینکه مفهوم بالا روشن شود، به شکل ٦ـ١ و ٦ـ٢ توجه کنید که ارتباط توابع در C و پاسکال را نمایش می‌دهد و مفهوم توابع تودرتو را روشن می‌سازد.



شکل ٦ـ۱ ارتباط توابع در پاسکال

در شکل ٦ـ۱ دو تابع f1 و f2 نمی‌توانند با یکدیگر ارتباط برقرار کنند. دو تابع f1 و f3 می‌توانند با یکدیگر ارتباط برقرار کنند. دو تابع f2 و f3 نیز می‌توانند با یکدیگر ارتباط برقرار کنند.



شکل ٦ـ۲ ارتباط توابع در C

در شکل ٦ـ۲ همهٔ توابع می‌توانند با یکدیگر ارتباط برقرار کنند. بنابراین در زبان C، توابع تودرتو یعنی امکان تعریف دو تابع به صورت تودرتو وجود ندارد.

قلمرو متغیرها

حال می‌خواهیم قلمرو متغیرها را بررسی کنیم؛ یعنی تعیین نماییم که هر متغیر در چه قسمتی از برنامه یا در کدام یک از توابع شناخته شده است.

متغیرهایی را که در داخل یک تابع توصیف گردند متغیرهای محلی[1] یا متغیرهای

1. local variables

خصوصی[1] نامند. این گونه متغیرها فقط در درون همان تابع شناخته می‌شوند و سایر توابع یا تابع اصلی آنها را نمی‌شناسد و به همین دلیل به آنها متغیرهای محلی یا خصوصی می‌گویند؛ یعنی خاص آن تابع مورد نظر است. در C، این گونه متغیرها را متغیرهای خودکار می‌شناسند. به هرحال این گونه متغیرها فقط باید در داخل بلوکی که توصیف شده‌اند به کار روند و در خارج بلوک خاص خود شناخته شده نیستند. یادآوری می‌شود که هر بلوک در زبان C با آکولاد باز آغاز و به آکولاد بسته خاتمه می‌یابد.

نکتهٔ مهم دیگر دربارهٔ این گونه متغیرها آن است که آنها فقط در موقعی که اجرای برنامه وارد بلوک مربوط به آنها می‌شود وجود دارند و با خروج از بلوک مربوط، متغیرهای مزبور دیگر وجود ندارند. به عبارت دیگر، هنگام ورود به بلوک حافظه‌های مورد نظر به این گونه متغیرها اختصاص داده می‌شوند و با خارج شدن از بلوک آن حافظه‌ها آزاد می‌گردند و دیگر در اختیار آن متغیرها نیستند.

متداول‌ترین بلوکی که متغیرهای محلی توصیف می‌گردند، بلوک مربوط به یک تابع است.

ـ **مثال ٦ـ١٥** دو تابع زیر را در نظر بگیرید.

```
func1()
   {
     int n ;
     n = 5 ;
     printf("%d" , n) ;
   }
func2()
   {
     int n ;
     n = 22 ;
     printf("%d" , n) ;
   }
```

ملاحظه می‌کنید که متغیر n دوبار توصیف شده است؛ یکبار در تابع اول، بار دیگر در تابع دوم. با اینکه نام متغیر در هر دو تابع یکسان است، هیچ ارتباطی بین این دو وجود ندارد. متغیر n اولی فقط در درون تابع func1 و متغیر n دومی نیز فقط در درون تابع func2 شناخته

1. private variables

می‌شود.

متداول است که نام متغیرهای یک تابع در آغاز تابع توصیف شود. ولی به هرحال این کار الزامی نیست، بلکه می‌توان متغیرهای محلی را در درون هر بلوک که مجموعه‌ای دستورند، توصیف کرد.

ـ **مثال ٦ـ١٦** به تابع زیر توجه کنید.

```
func1()
{
  int x ;
  scanf ("%d" , &x) ;
  if (x <=1)
     {
       char str[20] ;
       printf ("Enter City:  ") ;
       gets(str) ;
       puts(str) ;
     }
}
```

در اینجا متغیر محلی str هنگام ورود به بلوک مربوط به if ایجاد و هنگام خروج از آن تخریب می‌گردد. به علاوه str فقط در درون بلوک if شناخته شده است و در جای دیگر نمی‌توان به آن مراجعه کرد، حتی در درون همان تابع، ولی خارج از بلوکی که توصیف شده است.

ملاحظه می‌کنید که در اینجا در هر بار ورود به بلوک، متغیرهای محلی ایجاد و هنگام خروج تخریب می‌شوند. در فراخوانی تابع باید به‌این خاصیت توجه کرد؛ یعنی وقتی که تابع فراخوانی می‌شود، متغیرهای محلی آن ایجاد و هنگام خروج یا برگشت از آن تابع تخریب می‌شوند. به عبارت دیگر، متغیرهای محلی مقادیر خودشان را در بین فراخوانیهای متوالی نگه نمی‌دارند. در مقابل متغیرهای محلی متغیرهای دیگری به نام متغیرهای عمومی یا متغیرهای سراسری با ویژگیهای خودشان قرار دارد.

متغیرهای عمومی[1] متغیرهایی‌اند که در طول برنامه شناخته شده‌اند و می‌توانند در هر

1. global variables

قسمت از برنامه به کار روند. همچنین این گونه متغیرها مقادیر خود را در تمامی زمان اجرای برنامه نگه می‌دارند. متغیرهای عمومی با توصیف آنها در خارج هر تابع ایجاد می‌شوند. بنابراین معمولاً قبل از تابع main توصیف می‌گردند و با هر عبارتی در درون هریک از توابع در دسترس‌اند.

ـ **مثال ٦ـ۱۷** به برنامهٔ زیر توجه کنید.

```
int count ;          /* global variable */
main()
  {
     count = 100 ;
     func1() ;
  }

func1()
  {
     int temp ;
     temp = count ;
     func2() ;
     printf("count is %d" , count) ;
  }

func2()
  {
     int count ;
     for (count = 1 ; count < 10 ; count + +)
     putchar (" ; ") ;
  }
```

در این برنامه متغیر count خارج از توابع توصیف شده و در اینجا این کار قبل از تابع main انجام گرفته است. به هرحال می‌توانست در هر جای دیگر نیز توصیف شده باشد. ولی باید خارج از تابعی که آن را به کار می‌برد توصیف گردد. روش متعارف آن است که متغیرهای عمومی در بالای برنامه توصیف گردند. با توجه به قطعه برنامهٔ فوق ملاحظه می‌گردد که با اینکه متغیر count در main و همین طور در func1 توصیف نشده است، هر دوی آنها می‌توانند آن را به کار برند. اما به هرحال در func2 یک متغیر محلی به نام count توصیف شده است. بنابراین هرکجا در این تابع متغیر مزبور به کار برده شود، همان متغیر محلی تابع مزبور درنظر گرفته خواهد شد و متغیر عمومی یا سراسری که قبل از برنامه توصیف شده است

منظور نخواهد شد. پس باید توجه کرد که اگر یک متغیر عمومی با یک متغیر محلی همنام باشد، هر موقع در داخل تابعی که متغیر در آن به صورت محلی توصیف شده مراجعه شود، هیچ تأثیری در مورد متغیر عمومی حاصل نخواهد شد.

حافظهٔ مربوط به متغیرهای عمومی در ناحیهٔ ثابتی از حافظه که توسط کامپایلر برای همین منظور کنار گذاشته شده درنظر گرفته می‌شود. متغیرهای عمومی در مواقعی که داده‌های یکسان در بسیاری از توابع مربوط به برنامه به کار برده می‌شوند کاربرد زیادی دارند. به هرحال باید به دلایل زیر از به کار بردن غیرضروری متغیرهای عمومی خودداری کرد.

– در تمام مدت اجرای برنامه حافظه را اشغال می‌کنند.

– استفاده از متغیر عمومی درحالی که متغیر محلی بتواند همان نقش را به شکل مناسب ایفا نماید از عمومیت و کلیت تابع می‌کاهد، زیرا تابع باید به متغیری تکیه کند که در خارج از آن تابع تعریف شده است.

– استفاده از این متغیرها در سطح گسترده ممکن است به لحاظ آثار جنبی ناخواسته و ناشناخته منجر به بروز اشتباهاتی در برنامه گردد. یک مشکل عمده در طراحی برنامه‌های بزرگ، تغییر ناگهانی مقدار یک متغیر است، زیرا متغیر در جایی دیگر از برنامه به کار برده می‌شود.

کلاس حافظه

سطح ذخیره‌سازی اطلاعات یا کلاس حافظه قلمرو متغیر و نیز زمان حیات یک متغیر را در برنامه مشخص می‌کند؛ یعنی به آن بخش از برنامه که متغیر در آن شناخته می‌شود اشاره می‌کند.

به طور کلی در زبان C، چهار کلاس حافظه وجود دارد که عبارت‌اند از:

- حافظهٔ خودکار
- حافظهٔ ایستا
- حافظهٔ ثبات
- حافظهٔ خارجی

که در برنامه‌نویسی، این چهار کلاس حافظه به ترتیب با چهار کلمهٔ کلیدی auto، static، register، و extern شناخته و استفاده می‌شوند.

ـ **مثال ٦ـ۱۸** چند نمونه از اعلان متغیرها با مشخص ساختن کلاس حافظهٔ آنها در زیر بیان شده است.

```
auto int a , b , c ;
static int sum = 5 ;
extern float R1 , R2 ;
register int tax ;
```

ملاحظه می‌کنید که در نمونهٔ دوم، علاوه بر اعلان متغیر sum و نمایش نوع کلاس آن از نظر حافظه، مقدار اولیه نیز اختصاص داده شده است.

ـ

حافظهٔ خودکار

متغیرهای در حافظهٔ خودکار همیشه در درون یک تابع توصیف می‌شوند و نسبت به آن تابع متغیر محلی‌اند؛ یعنی حوزه و قلمرو کاربرد آنها به همان تابع محدود است. بنابراین متغیرهایی که در توابع گوناگون در این کلاس حافظه تعریف شوند، از یکدیگر مستقل خواهند بود حتی اگر اسامی یکسانی داشته باشند.

هر متغیری که در درون یک تابع اعلان شود، از لحاظ کلاس حافظه، به طور پیش‌فرض خودکار در نظر گرفته می‌شود؛ یعنی قرار دادن کلمهٔ کلیدی auto در جلوی آنها ضروری نیست. پس اگر کلاس حافظهٔ متغیری غیر از خودکار باشد، باید حتماً کلمهٔ کلیدی مربوط به آن کلاس حافظه، به طور صریح مشخص شود. می‌توان هنگام توصیف این گونه متغیرها به آنها مقدار اولیه نیز اختصاص داد.

ـ **مثال ٦ـ۱۹** در زیر، متغیرهایی با استفاده از توصیف auto و هم بدون آن نمایش داده شده‌اند که هم ارزند.

```
int a , b , c ;                    auto int a , b , c ;
float x1 , x2 ;                    auto float x1 , x2 ;
char ch ;                          auto char ch ;
int a[10] ;                        auto int a[10] ;
float p = 3.1415 ;                 auto float p = 3.1415 ;
char ch ='A' ;                     auto char ch ='A' ;
char str[5] = "noor" ;             auto char str[5] = "noor" ;
```

همان طور که گفتیم، متغیرهای خودکار با خروج از تابع یا بلوکی که در درون آن تعریف شده‌اند، مقادیر خود را از دست می‌دهند؛ یعنی مقادیر خود را نگهداری نمی‌کنند. اگر در منطق برنامه‌ای لازم باشد که به یک متغیر خودکار مقدار خاصی اختصاص داده شود، هر زمان که تابع مربوط به آن (یعنی تابعی که متغیر مزبور متغیر محلی آن است) اجرا می‌گردد، باید هنگام ورود مجدد به تابع، آن متغیر مورد نظر نیز دوباره مقداردهی شود.

حافظهٔ خارجی

زبان برنامه‌نویسی C، برای کمک به مدیریت پروژه‌های بزرگ و سرعت بخشیدن به عمل ترجمه، اجازه می‌دهد که ماژولهای جداگانه ترجمه شدهٔ برنامه‌ای بزرگ با یکدیگر پیوند داده شوند. لذا باید راهی وجود داشته باشد که فایلهای مربوط به متغیرهای عمومی مورد نیاز، به برنامه گفته شود. برای این کار، همهٔ متغیرهای عمومی در یک فایل توصیف می‌گردند و در سایر فایلها، آن متغیرها با استفاده از کلمهٔ کلیدی extern توصیف می‌گردند.

به طور کلی اگر حجم برنامه‌ای بزرگ باشد، می‌توان آن را به قسمتهای منطقی کوچک‌تر به نام ماژول یا واحد تجزیه کرد و هر واحد را در یک فایل جداگانه قرار داد و ترجمه یا تفسیر کرد و سپس آنها را با یکدیگر به اجرا درآورد. در این روش، اگر متغیرهایی را در واحد اصلی تعریف کنیم و بخواهیم از آنها در واحدهای فرعی استفاده کنیم، بدون اینکه در این واحدهای فرعی حافظه‌ای به آنها اختصاص یابد، باید آنها را در این واحدها با کلمهٔ کلیدی extern معرفی کنیم. با این عمل به کامپایلر می‌گوییم که این متغیرها در جای دیگری، یعنی در واقع در واحد اصلی، تعریف شده‌اند؛ یعنی اینها متغیرهای خارجی‌اند.

ـ **مثال ٦ـ۲۰** در زیر، دو ماژول که جداگانه ترجمه می‌شوند با استفاده از متغیرهای کلی

نشان داده شده‌اند.

File1

```
int x , y ;
char ch ;
main()
    {....
        ....
    }
func1()
    {
        x = 123 ;
    }
```

File2

```
extern int x , y ;
extern char ch ;
func2()
    {
      x = y / 10 ;
    }
func3()
    {
      y = 10 ;
    }
```

ملاحظه می‌کنید که در فایل دوم، لیست متغیرهای عمومی به طور دقیق از فایل اول نسخه‌برداری شده‌اند، ولی کلمهٔ کلیدی extern در توصیف آنها افزوده شده است. توصیف‌کنندهٔ extern به کامپایلر می‌گوید که نوع و اسامی این متغیرها در جای دیگری توصیف شده‌اند، به عبارت دیگر، extern اجازه می‌دهد که کامپایلر نوع و اسامی این متغیرهای عمومی را بدون ایجاد دوبارهٔ حافظهٔ واقعی برای آنها بشناسد.

وقتی که متغیر عمومی را در درون تابعی در همان فایلی که توصیف متغیرهای عمومی نیز در آن صورت گرفته است به کار می‌برید، به استفاده از extern نیازی نیست. مانند متغیر x در func1 و File1 و متغیرهای x , y در func2 و y در func3 از File2.

البته انتخاب extern کمتر متداول است. برای مثال قطعه برنامهٔ زیر کاربرد این گزینش را نشان می‌دهد.

```
int first , last ;        /* global definition of first and last */
main()
  {
    extern int first ;        /* optional use of the extern declaration */
  }
```

با اینکه توصیف متغیر extern می‌تواند در داخل همان فایلی که متغیرهای عمومی توصیف شده‌اند انجام پذیرد، این کار ضرورتی ندارد، زیرا کامپایلر C وقتی که با متغیری

برخورد می‌کند که توصیف نشده است، کنترل می‌کند که آیا این متغیر با یکی از متغیرهای عمومی همنام است یا نه و درصورت همنام بودن، آن را متغیر extern درنظر می‌گیرد.

به هرحال چون متغیرهای خارجی به صورت عمومی تعریف شده‌اند، قلمرو آنها از همان محل یا نقطهٔ تعریفشان تا انتهای برنامه است و در بقیهٔ برنامه و همهٔ توابعی که بعد از آن می‌آیند شناخته شده‌اند. بنابراین توسط همهٔ آن توابع در دسترس‌اند. حال چنانچه در هرکدام از این توابع، مقادیری به‌این‌گونه متغیرها اختصاص داده شود، پس از خروج از آن تابع نیز مقادیر اختصاص داده شده به آن متغیرها باقی خواهد ماند. با استفاده از این خاصیت می‌توانیم اطلاعاتی را بدون استفاده از آرگومان به توابع انتقال دهیم. این روش، بویژه در مواردی که تابع مورد نظر به اقلام داده ورودی متعددی نیاز دارد، راهی ساده در اختیار ما قرار می‌دهد.

به هرحال اگر تعریف تابع قبل از تعریف متغیرهای خارجی بیاید، باید آن متغیرها در تابع مزبور نیز متغیرهای خارجی توصیف گردند. همچنین باید توجه داشت که در توصیف متغیرهای خارجی نمی‌توان به آنها مقدار اولیه اختصاص داد. این، یک تفاوت اساسی بین تعریف و توصیف متغیرهای خارجی است؛ یعنی در تعریف متغیرهای خارجی به عنوان متغیرهای عمومی یا global می‌توان به آن مقدار اولیه نیز نسبت داد، ولی در توصیف آنها در جای دیگر یا تابع دیگر، به‌عنوان متغیر خارجی، اختصاص مقدار اولیه مجاز نیست، ولی بعد می‌توان در همان تابع با دستور انتساب یا خواندن، مقادیر دلخواه دیگری به آنها اختصاص داد.

حافظهٔ ایستا

در برنامه‌ای تک‌فایل، هرکدام از متغیرهای ایستا در همان فایلی که مربوط به آن‌اند تعریف می‌شوند. بنابراین قلمرو آنها مشابه متغیرهای خودکار است؛ یعنی نسبت به تابعی که در درون آن تعریف شده‌اند محلی‌اند. اما به هرحال این گونه متغیرها، برخلاف متغیرهای خودکار، در تمامی مدت اجرای برنامه مقدار خود را نگهداری می‌کنند؛ یعنی با خروج از تابع، مقدار قبلی خود را حفظ می‌کنند و با ورود مجدد به تابع همان مقادیری را دارایند که در موقع خروج از تابع داشتند. این ویژگی، این اجازه را به تابع می‌دهد که متغیرهای مورد نظر در تمامی مدت اجرای برنامه مقدار خود را از دست ندهند.

متغیرهای ایستا در درون تابع به همان شیوه‌ای که در مورد متغیرهای خودکار بیان شد

تعریف می‌شوند، با این تفاوت که توصیف متغیر باید با مشخص ساختن کلاس حافظه به کمک کلمهٔ کلیدی static مشخص گردد. این گونه متغیرها مشابه همان متغیرهای خودکار، در درون تابع به کار می‌روند. به هرحال امکان دستیابی به این متغیرها خارج از تابعی که در درون آن تعریف شده‌اند ممکن نیست.

می‌توان متغیرهای خودکار و ایستا را همنام با متغیرهای خارجی تعریف کرد. بنابراین، در چنین مواردی هر گونه تغییر در متغیرهای خودکار و ایستا هیچ‌گونه نقشی در مورد متغیرهای عمومی همنام با آنها نخواهد داشت.

ـ **مثال ٦ـ۲۱** برنامهٔ زیر را درنظر بگیرید.

```
float a , b , c ;
main()
   {
     static float a ;
     void dummy(void) ;
   }
void site (void) ;
   {
     static int a ;
     int b ;
   }
```

در این برنامه، متغیرهای a، b، c به صورت اعشاری و خارجی معرفی شده‌اند، اما متغیر a مجدداً در درون تابع main به صورت اعشاری و ایستا تعریف شده است. بنابراین در این تابع فقط دو متغیر c و b متغیرهای خارجی‌اند و متغیر محلی a مستقل از متغیر خارجی a خواهد بود.

به طریق مشابه متغیرهای a و b که در درون تابع site مجدد از نوع صحیح تعریف شده‌اند، متغیرهای محلی آن تابع خواهند بود که البته a از کلاس حافظهٔ ایستاست، ولی b حافظهٔ خودکار است. بنابراین هنگام خروج از تابع، متغیر a مقدار قبلی خود را حفظ خواهد کرد، اما b آن را از دست خواهد داد. متغیر c نیز در این تابع متغیر خارجی است، ولی a و b که محلی‌اند از متغیرهای a و b خارجی مستقل خواهند بود.

ـ

هنگام توصیف متغیرهای محلی ایستا، می‌توان آنها را مقداردهی اولیه کرد؛ یعنی به آنها

مقدار اولیه نیز اختصاص داد که اگر به‌این‌گونه متغیرها مقدار اولیه اختصاص داده نشود، کامپایلر مقدار آنها را اغلب صفر درنظر می‌گیرد. همچنین مقدار اولیه باید به شکل ثابت بیان گردند نه عبارت.

حافظهٔ ثبات

کلاس دیگری از حافظه که فقط در مورد متغیرهایی از نوع int و char اعمال‌پذیر است، مشخصهٔ ثبات یا register است. این مشخصه موجب می‌گردد که این نوع متغیرها حافظه را به جای حافظهٔ اصلی کامپیوتر (که به طور متعارف همهٔ متغیرها در آن ذخیره می‌گردند) ثبات CPU درنظر گیرند که در این صورت هر نوع عملیات روی این گونه متغیرها خیلی سریع انجام می‌گیرد. این شیوه معمولاً در مورد متغیرهای کنترل‌کنندهٔ حلقه به کار می‌رود که موجب افزایش محسوس سرعت عملیات می‌گردد. به هرحال این کار فقط برای متغیرهای محلی است. همچنین به لحاظ محدود بودن ثباتهای CPU، تعداد متغیرهایی که می‌توانند در هر برنامه با این شیوه به کار روند محدود است.

ـ **مثال ٦ـ۲۲** به تابع زیر توجه کنید.

```
int noor(m , n)
int m ;
register int n ;
   {
     register int k ;
     k = 1 ;
     for (; n>0 ; n--)
          k = k + m ;
     return k ;
   }
```

در این تابع هر دو متغیر n و k به صورت متغیری با کلاس حافظهٔ register توصیف شده‌اند، زیرا هر دو در درون حلقه به کار می‌روند.

ـ

خودآزمایی ۶

۱. تابعی بنویسید که میانگین اعداد طبیعی ۱ تا n را محاسبه و چاپ کند.

۲. تابعی بنویسید که توانهای اعشاری را محاسبه کند.

۳. برنامه‌ای بنویسید که دو عدد صحیح m و n را از طریق ورودی بخواند و سپس بزرگ‌ترین مقسوم علیه مشترک آن دو را به دست آورد و چاپ کند.

٤. برنامهٔ تمرین قبل را به کمک تابع فرعی بنویسید.

۵. برنامهٔ تمرین قبل را به کمک تابع بازگشتی بنویسید.

٦. برنامه‌ای بنویسید که عددی را در مبنای ۱۰ بخواند، معادل آن را در مبنای ۲ به دست آورد و چاپ کند.

۷. برنامهٔ تمرین قبل را به صورت تابع فرعی بنویسید. درضمن مبنای جدید را نیز d درنظر بگیرید که به فرض d کوچک‌تر از ۱۰ است.

۸. برنامهٔ تمرین ٦ را بدون استفاده از آرایه بنویسید.

۹. برنامهٔ تمرین قبل را با فراخوانی تابع فرعی به صورت بازگشتی بنویسید.

۱۰. تابعی بنویسید که عدد صحیح m در مبنای ۲ را دریافت کند و معادل آن را در مبنای ۱۰ به دست آورد و برگرداند.

۱۱. برنامه‌ای بنویسید که عدد صحیح n را دریافت و تعیین کند که آیا عدد اول است یا نه؟ و برحسب مورد پیغام مناسبی چاپ کند.

۱۲. ازنظر ریاضی عددی را که برابر مجموع مقسوم‌علیه‌های خودش (به غیر از خودش) باشد، عدد کامل یا complete number نامند. مانند عدد ٦ که مقسوم‌علیه‌های آن ۱، ۲، ۳ است و داریم:

$$6 = 1 + 2 + 3$$

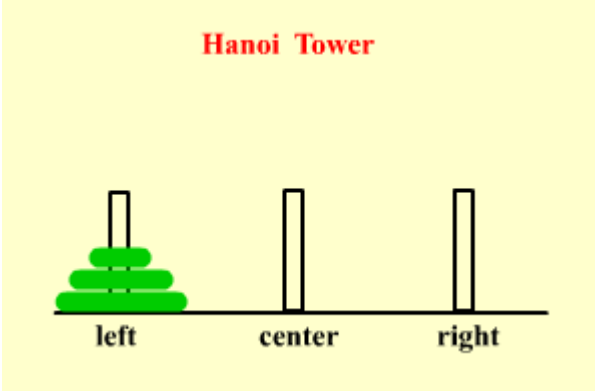
برنامه‌ای بنویسید که عدد صحیح m را دریافت و تعیین کند که آیا عدد کامل است یا نه و برحسب مورد، پیغام مناسبی چاپ کند.

۱۳. برنامه‌ای بنویسید که عدد صحیح m را دریافت کند و سپس با فراخوانی تابعی، کلیهٔ اعداد کامل را که مقدار آنها مساوی یا کوچک‌تر از m است تعیین و چاپ کند.

۱٤. اعداد دوقلو (twin numbers)، دو عدد اول را گویند که تفاوت یا تفاضل آنها برابر ۲ باشد، مانند دو عدد ۲ و ۵ یا دو عدد ۱۱ و ۱۳. برنامه‌ای بنویسید که عدد صحیح m را دریافت کند و کلیهٔ اعداد دوقلوی مساوی یا کوچک‌تر از آن را تعیین و چاپ کند.

۱۵. تابعی بنویسید که عدد صحیح m را دریافت کند. اگر m عدد زوج بود مقدار یک وگرنه مقدار صفر برگرداند.

۱٦. برج هانوی[1] یکی از بازیهای معروف بچه‌هاست. در این بازی سه میله و تعدادی حلقه با قطرهای مختلف وجود دارد که در میله‌ها قرار می‌گیرند. بازی به‌این طریق انجام می‌گیرد که در آغاز کار همهٔ حلقه‌ها در میلهٔ سمت چپ به‌صورت نزولی قرار می‌گیرند (یعنی حلقهٔ بزرگ‌تر در زیر و حلقهٔ کوچک‌تر در بالا).



این حلقه‌ها باید با رعایت دو شرط زیر از میلهٔ سمت چپ (میلهٔ مبدأ) به میلهٔ سمت راست (میلهٔ مقصد) انتقال یابند.

ـ هر بار فقط مجازیم یک حلقه از روی میله‌ای به روی میلهٔ دیگر انتقال دهیم (می‌توان میلهٔ وسط را میلهٔ واسطه و کمکی قرار داد).

ـ هیچ‌وقت حلقه‌ای با قطر بزرگ‌تر روی حلقه‌ای با قطر کوچک‌تر قرار نگیرد.

این بازی را طراحی کنید.

راهنمایی. برنامه‌نویسی برای حل این مسئله به روش عادی یا روش تکراری یا

1. Tower of Hanoi

Iteration method بسیار طولانی و مشکل است. ولی می‌توان آن را به سادگی با روش بازگشتی برنامه‌نویسی کرد. روش استدلال ریاضی را در مورد اثبات قضیه که آن را استقراء ریاضی یا Mathematical Induction نامند یادآور می‌شویم. در این روش، مسئله را برای حالت n-1 اثبات شده فرض می‌کنند. سپس برای حالت n دلیل می‌آورند. در اینجا نیز از همین روش استفاده می‌کنیم؛ یعنی فرض می‌کنیم که برای انتقال n-1 حلقه از روی میلهٔ دلخواهی به روی میلهٔ دلخواه دیگر، با رعایت دو شرط مذکور در بالا، باید راهی وجود داشته باشد. با قبول این فرض می‌توان از الگوریتم زیر برای انتقال n حلقه از روی میلهٔ مبدأ به روی میله مقصد استفاده کرد.

الف) n-1 حلقه‌های رویی را (با استفاده از روشی که فرض کردیم وجود دارد) از روی میلهٔ چپ (میلهٔ مبدأ) به میلهٔ وسطی (میلهٔ واسطه) انتقال دهید.

ب) حلقهٔ n اُم را (که بزرگ‌ترین حلقه است و اکنون آزاد شده است) از میلهٔ مبدأ (میلهٔ چپ) به میلهٔ مقصد (میلهٔ راست) انتقال دهید.

ج) n-1 حلقه را با استفاده از روشی که فرض کردیم وجود دارد از میلهٔ وسطی (میلهٔ واسطه) به میلهٔ راست (میلهٔ مقصد) انتقال دهید.

نکتهٔ ۱: واضح است که اگر n = 0، نیازی به هیچ‌گونه نقل و انتقال نخواهد بود و درواقع این حالت قانون توقف الگوریتم را نشان می‌دهد.

نکتهٔ ۲: مراحل اولی و آخری (یعنی دو مرحلهٔ ۱ و ۳) الگوریتم مزبور حالت بازگشتی دارد؛ یعنی نقل و انتقال n-1 حلقه نیز برحسب n-2 بیان خواهد شد تا اینکه n = 0 گردد.

۱۷. برنامه‌ای بنویسید که مستطیل توپری دور عبارت "COMPUTER SCIENCE" رسم کند.

۱۸. چاپ کردن کاراکتر مخصوص ' \X7 ' که در کد اسکی استاندارد bell نام دارد، صدای کوتاهی به نام beep در بلندگوی دستگاه ایجاد می‌کند. برنامه‌ای بنویسید که با فراخوانی تابع twobeep دو بار صدای کوتاه (منقطع) در بلندگوی دستگاه ایجاد کند.

۱۹. تابع زیر که در آن n عدد صحیح غیرمنفی است چه عملی انجام می‌دهد ؟

```
int sum(int n)
  {
    int n ;
    if (n<=1)
      return n
    else
      return (n + sum(n-1)) ;
  }
```

فصل ۷

آرایه‌ها

هدف کلی

آشنایی با قابلیتهای متعدد آرایه‌ها

هدفهای رفتاری

از دانشجو انتظار می‌رود، پس از خواندن این فصل،

۱. با مفهوم آرایه آشنا شود.

۲. بردار و ماتریس را بشناسد.

۳. نحوهٔ تعریف آرایه‌های یک‌بعدی را بیان کند.

۴. کلاس حافظه در آرایه‌ها و نحوهٔ مقداردهی اولیهٔ آنها را بشناسد.

۵. چگونگی تعریف آرایه‌های چندبعدی را بداند.

۶. با نحوهٔ انتقال آرایه به تابع آشنا شود.

۷. با رشته‌ها، ثابت رشته‌ای، و متغیر رشته‌ای آشنایی پیدا کند.

۸. هدف از به کاربردن آرایه‌ها در مرتب‌سازی و روشهای آن را بداند.

۹. انواع رشته‌های جستجو را نام ببرد.

۱۰. با توابع کتابخانه‌ای strcpy، strcat، strlen، strcmp، و strcmoi آشنا شود.

مقدمه

آرایه مجموعه عناصری است که ویژگیها و صفات یکسانی دارند. به عبارت دیگر آرایه فضای پیوسته‌ای از حافظهٔ اصلی کامپیوتر است که می‌تواند چندین مقدار را در خود جای دهد. همهٔ

عناصر یک آرایه از یک نوع‌اند و با اندیس مشخص می‌شوند.

از نظر ریاضی معمولاً آرایه‌های یک‌بعدی را بردار و آرایه‌های دوبعدی را ماتریس نامند. همچنین به طریق مشابه می‌توان آرایه‌های چندبعدی را تعریف کرد.

تعریف آرایه‌ها

در زبان C، آرایه‌ها به شکل متغیرهای معمولی تعریف می‌شوند با این تفاوت که نام آرایه باید با مشخصهٔ اندازه همراه باشد.

آرایهٔ یک‌بعدی

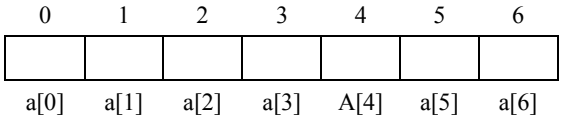
آرایهٔ یک بعدی به صورت زیر تعریف می‌شود.

```
type array-name [array-size] ;
```

که در آن array-name نام آرایه است که از قانون نامگذاری متغیرها پیروی می‌کند، array-size بزرگی و یا تعداد عناصر آرایه است و type نیز نوع عناصر آن را مشخص می‌کند. برای مثال اگر آرایهٔ a دارای ۷ عنصر از نوع int باشد، به‌این صورت معرفی می‌شود.

```
int a[7] ;
```

و خانه‌های اختصاص داده شده به آن به صورت متوالی و به شکل زیر خواهد بود.



همان طور که می‌بینید شمارهٔ خانه‌ها از صفر تا شش است. به عبارت دیگر حد پایین آن برابر صفر و حد بالای آن یک واحد از طول یا بزرگی آرایه کمتر خواهد بود که در مثال مزبور، حد بالای آن برابر ۶ است.

روش برنامه‌نویسی خوب آن است که اندازهٔ آرایه به صورت ثابت سمبولیک تعریف شود. از آنجا که با تغییر مقدار ثابت سمبولیک اندازهٔ آرایه به راحتی تغییر می‌کند، این عمل تغییر برنامه‌ای را که از آرایه سود می‌برد ساده‌تر می‌سازد.

ـ **مثال ۷ـ۱** به دستورهای زیر توجه کنید.

```
#define size 50
int A[size] ;
```

در اینجا طول آرایه به صورت غیرمستقیم و با استفاده از دستور define مشخص شده است. کلاس حافظه ممکن است خودکار، ایستا، یا خارجی باشد، اما نمی‌تواند ثبات تعریف شود. بنابراین در حالت کلی می‌توان آرایه‌ای یک‌بعدی را به صورت زیر تعریف کرد.

storage-class data-type array-name [expression] ;

که در آن expression ممکن است عدد صحیح یا متغیر از نوع int و یا عبارت سادهٔ محاسباتی باشد که از ترکیب مقادیر عددی صحیح و متغیرهایی از نوع int با استفاده از عملگرهای محاسباتی مانند + و – تشکیل شده است. نتیجهٔ این عبارت یک عدد صحیح خواهد بود که معرف بزرگی آرایه است. متعارف آن است که بزرگی آرایه به صورت عدد صحیح و یا متغیری از نوع عدد صحیح تعیین گردد. واضح است که اگر بزرگی آرایه به صورت متغیر از نوع int بیان گردد، باید مقدار آن هنگام تعریف عناصر آن تشخیص داده شود. برای آرایه‌هایی که درون یک تابع یا بلوک تعریف می‌گردند سطح ذخیره‌سازی خودکار و برای آرایه‌هایی که بیرون از تابع تعریف می‌گردند سطح ذخیره‌سازی خارجی پیش فرض خواهد بود.

ـ **مثال ۷ـ۲** به نمونه‌هایی از تعریف چند آرایهٔ یک بعدی توجه کنید.

```
int A[5] ;
float B[25] ;
static float C[15] ;
double x1[10] ;
char str[80] ;
```

در این اعلان آرایهٔ A از نوع int با ۵ عنصر و آرایهٔ B از نوع float با ۲۵ عنصر و آرایه C از نوع float با ۱۵ عنصر و از لحاظ کلاس حافظه نیز ایستا تعریف شده است. همچنین آرایهٔ x1 از نوع double و آرایهٔ str از نوع char با ۸۰ عنصر تعریف شده است. (آرایه‌های کاراکتری معمولاً برای نمایش رشته‌ها به کار می روند.)

مراجعه به عناصر آرایه

وقتی که آرایه‌ای را تعریف می‌کنیم، کامپایلر مجموعه‌ای حافظه به صورت پیوسته یا متوالی برای آن در نظر می‌گیرد. وقتی که آرایه‌ای ایجاد شد، برای مراجعه به هر عنصر آن کافی است

پس از نام آرایه، شمارهٔ عنصر مورد نظر را در درون یک زوج کروشه قرار دهیم. همچنین در زبان C، اندیس آرایه از صفر آغاز می‌گردد.

ـ **مثال ۷ـ۳** اگر A نام آرایه‌ای باشد که از پیش تعریف شده و مقادیری نیز به آن اختصاص داده شده باشد در این صورت دستور ; k = A[2] یک کپی از محتوای خانهٔ شمارهٔ 2 آرایه (یعنی سومین عنصر آرایه) را به متغیر k نسبت می‌دهد.

یادآور می‌شویم که نامگذاری آرایه‌ها از قانون نامگذاری متغیرها تبعیت می‌کند و نوع عناصر آن نیز مانند متغیرهای معمولی ممکن است int، float، char و جز آن باشد.

ـ

کلاسهای حافظه درآرایه (و نحوهٔ مقداردهی اولیهٔ آنها)

گفتیم که آرایه‌ها از نظر کلاس حافظه ممکن است خودکار، ایستا یا خارجی تعریف شوند، ولی نمی‌توانند ثبات تعریف شوند. آرایه‌های از نوع خودکار، برخلاف متغیرهای خودکار ممکن است هنگام تعریف، مقدار اولیه بپذیرند.

حال باتوجه به این توضیح‌ها می‌توان شکل کلی مقداردهی اولیه به آرایه‌ها را به صورت زیر بیان کرد.

storage-class data-type array-name [size] = {value1 , value2 , … valuem} ;

که در آن value1، value2...valuem به ترتیب مقدار اولیهٔ اولین، دومین،... و m اُمین عنصر آرایه را مشخص می‌کنند.

فرض کنید که آرایه‌های a، b و c به صورت زیر تعریف شده‌اند.

```
int a[7] = {1 , 2 , 3 , 4 , 5 , 6 , 7} ;
static float b[5] = {2.5 , -3.5 , 1.25 , 12.5 , 3.14} ;
char c[3] = {'a' , 'b' ,'c'} ;
```

ملاحظه می‌کنید که آرایهٔ b ازنظر کلاس حافظهٔ ایستا معرفی شده است، ولی کلاس حافظهٔ دو آرایهٔ a و c به طور صریح بیان نشده است. بنابراین باید فرض کرد که هر دوی آنها خارجی‌اند. پس مقادیر عناصر سه آرایهٔ مزبور به صورت زیر خواهد بود.

```
a[0] = 1b [0] = 2.5      c (0) = a
a[1] = 2b [1] = 3.5      c (1) = b
a[2] = 3b [2] = 1.25     c (2) = c
a[3] = 4b [3] = 12.5
a[4] = 5b [4] = 3.14
```

a[5] = 6
a[6] = 7

اگر آرایه‌ای به صورت مقداردهی اولیه تعریف گردد، نیاز نیست بزرگی یا اندازهٔ آن را مشخص کنیم.

ـ **مثال ۷ـ٤** به اعلان آرایهٔ زیر توجه کنید.

```
int digit[ ] = {1 , 2 , 3 , 4 , 5} ;
```

که عناصر آن به این صورت خواهد بود.

digit[0] = 1
digit[1] = 2
digit[2] = 3
digit[3] = 4
digit[4] = 5

در مثالهای ذکر شده، آرایه‌هایی که مقادیر اولیه گرفته‌اند و کلاس حافظهٔ آنها به طور صریح مشخص نشده از نوع خارجی فرض شده‌اند (یعنی از نوع خودکار نیستند).

ـ **مثال ۷ـ۵** برنامهٔ زیر نمرهٔ امتحانی ۲۵ نفر دانشجویان کلاسی را در درس ریاضی می‌خواند و تعداد دانشجویان مردود و همچنین تعداد دانشجویانی را که نمرهٔ امتحانی آنان کمتر از معدل کلاس است تعیین و در خروجی چاپ می‌کند.

```
# include<stdio.h>
main ()
  {
    int i , f = 0 , p = 0 ;
    float a[25] , average , sum = 0 ;
    for (i = 0 ; i<25 ; ++ i)
      {
        scanf ("%f " , &a[i]) ;
        sum = sum + a[i] ;
        if (a[i]<10)
          f = f + 1 ;
      }
    average = sum / 25 ;
    for (i = 0 ; i<25 ; ++i)
      if (a[i] < average)
        p = p + 1 ;
    printf ("%f %d %d" , average , f , p) ;
  }
```

در این برنامه f و p به‌ترتیب معرف تعداد دانشجویان مردود و دانشجویانی است که نمرهٔ آنان کمتر از معدل کلاسی است و در آغاز مقدار اولیهٔ صفر داده شده است. در ضمن یادآور می‌شویم که این برنامه را به طور متعارف نمی‌توانیم بدون استفاده از آرایه بنویسیم، زیرا قبل از اینکه نمرهٔ امتحانی همهٔ دانشجویان خوانده شود و معدل کلاس محاسبه گردد، نمی‌توان تعیین کرد که آیا نمرهٔ دانشجوی مورد نظر از معدل کلاس کمتر است یا نه. بنابراین چنانچه نمره‌های دانشجویان را با متغیری ساده معرفی کنیم، هر بار که نمرهٔ دانشجوی جدیدی خوانده می‌شود، در همان حافظه می‌نشیند. لذا نمرهٔ دانشجوی قبلی پاک می‌گردد. پس اگر معدل کلاس را بدون استفاده از آرایه حساب کنیم، در پایان خواندن نمرهٔ دانشجویان به حافظهٔ کامپیوتر، فقط نمرهٔ نفر آخر را در حافظه خواهیم داشت و درنتیجه در مورد دانشجویان اول تا بیست و چهارم نمی‌توانیم تعیین کنیم که نمرهٔ کدام یک از آنها کمتر از معدل کلاس است. پس در این مثال استفاده از آرایه الزامی است. درنتیجهٔ آن ۲۵ نمره به ۲۵ آدرس مختلف خوانده می‌شود. همچنین باید توجه کنیم که چون اندیس از صفر شروع می‌شود، شمارهٔ اندیسها از صفر تا بیست و چهار خواهد بود.

آرایه‌های چند بعدی

آرایه‌های چندبعدی نیز مشابه آرایه‌های یک‌بعدی تعریف می‌گردند، با این تفاوت که طول یا بزرگی هر بعد آرایه در داخل یک زوج کروشه مشخص می‌گردد. از این رو آرایهٔ دوبعدی دو جفت کروشه و آرایهٔ سه بعدی سه جفت کروشه دارد.

بنابراین شکل کلی تعریف آرایهٔ n بعدی به صورت زیر خواهد بود.

```
storage-class data-type array-name[size1][size2]… [sizen] ;
```

که در آن sizen ,..., size2 , size 1 به‌ترتیب بزرگی بعد یکم تا n اُم آرایه است.

برای مثال یک آرایهٔ دوبعدی 4 × 3 از نوع int را می‌توان به صورت زیر معرفی کرد.

```
int a[3][4] ;
```

در شکل ۷ـ۱ آرایهٔ دوبعدی m × n (شامل m سطر و n ستون) را می‌بینید.

به طریق مشابه می‌توان آرایه‌های ۳ بعدی یا بیشتر را تعریف کرد، مانند مثالهای زیر.

```
int a[3][4][5] ;
```

float b[2][3][8][5] ;
char page[2][5][10] ;

	ستون ۱	ستون ۲	ستون ۳		ستون n
سطر ۱	a[0][0]	a[0][1]	a[0][2]	...	a[0][n -1]
سطر ۲	a[1][0]	a[1][1]	a[1][2]	...	a[1][n -1]
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.
سطر m	a[m -1][0]	a[m -1][1]	a[m - 1][2]	...	a[m -1][n -1]

شکل ۷ـ۱ آرایهٔ دوبعدی m × n

همچنین می‌توان هنگام تعریف آرایه‌های دوبعدی یا بالاتر به آنها مقدار اولیه نسبت داد که البته باید آرایهٔ مورد نظر از کلاس حافظهٔ ایستا یا خارجی باشد.

ـ **مثال ۷ـ۶** دستور زیر را در نظر بگیرید.

int array[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12} ;

در اینجا فرض بر این است که آرایه از کلاس حافظهٔ خارجی است. آرایهٔ مزبور را می‌توان جدولی تصور کرد که دارای ۳ سطر و ۴ ستون (۴ عنصر در هر سطر) است. مقادیر اولیه نیز به ترتیب به عناصر سطرها اختصاص می‌یابد (یعنی اندیس یا زیرنویس سمت راست سریع‌تر حرکت می‌کند). بنابراین مقادیر عناصر آرایهٔ مزبور به صورت زیر خواهد بود.

array[0][0] = 1 array[0][1] = 2 array[0][2] = 3 array[0][3] = 4
array[1][0] = 5 array[1][1] = 6 array[1][2] = 7 array[1][3] = 8
array[2][0] = 9 array[2][1] = 10 array[2][2] = 11 array[2][3] = 12

توجه داشته باشید که عملکرد یا محدودهٔ تغییرات اندیس اول (اندیس سمت چپ) از صفر تا ۲ و اندیس دوم (اندیس سمت راست) از صفر تا ۳ است.

ـ

نحوهٔ اختصاص مقادیر اولیه به عناصر آرایه را می‌توان با دسته‌بندی کردن آنها در داخل زوج آکولاد تغییر داد. برای مثال در آرایهٔ دوبعدی مقادیر داخل هر زوج آکولاد درونی به ترتیب به عناصر یکی از سطرها نسبت داده خواهد شد. اگر تعداد مقادیر موجود در درون هر

زوج آکولاد درونی کمتر از تعداد عناصر سطر متناظر آن باشد، به بقیهٔ عناصر سطر مزبور مقدار صفر نسبت داده خواهد شد. برای مثال، نتیجهٔ عملکرد دستور زیر

```
int array[3][4] = {
 {1 , 2 , 3 , 4} ,
 {5 , 6 , 7 , 8} ,
 {9 , 10 , 11 , 12}
 } ;
```

با مثال ۷ـ۶ یکسان خواهد بود.

حال دستور زیر را درنظر بگیرید.

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

به عناصر ستون چهارم مقادیری نسبت داده نشده است. لذا مقادیر آنها برابر صفر خواهد شد؛ یعنی مقادیر عناصر آرایهٔ مورد نظر به صورت زیر خواهد بود.

```
array[0][0] = 1  array[0][1] = 2  array[0][2] = 3  array[0][3] = 0
array[1][0] = 4  array[1][1] = 5  array[1][2] = 6  array[1][3] = 0
array[2][0] = 7  array[2][1] = 8  array[2][2] = 9  array[2][3] = 0
```

درحالی که اگر آرایهٔ مزبور را به صورت `int array[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9) ;` تعریف کنیم، مقادیر نسبت داده شده به عناصر آرایهٔ مورد نظر به صورت خواهد بود.

```
array[0][0] = 1  array[0][1] = 2  array[0][2] = 3  array[0][3] = 4
array[1][0] = 5  array[1][1] = 6  array[1][2] = 7  array[1][3] = 8
array[2][0] = 9  array[2][1] = 0  array[2][2] = 0  array[2][3] = 0
```

انتقال آرایه به تابع

در زبان C، وقتی که نام آرایه به عنوان آرگومان تابع ظاهر می‌شود، آدرس اولین عنصر آرایه تعبیر می‌گردد. بنابراین هنگامی که آرایه‌ای را به عنوان آرگومان به تابع گذر یا انتقال می‌دهیم، تمامی آرایه به آن تابع انتقال می‌یابد. این روش انتقال آرگومان به تابع، با آنچه در مورد انتقال متغیرهای معمولی به تابع در فصل مربوط به توابع بیان شد و فراخوانی با مقدار نامیدیم متفاوت است. روش فراخوانی جدید را فراخوانی با آدرس یا فراخوانی با ارجاع نامند. در این روش به جای کپی داده‌ها، آدرس آنها به تابع انتقال می‌یابد. تشریح کامل این روش را در فصل

۸ بیان می‌کنیم.

برای گذر دادن آرایه‌ای به تابع باید فقط نام آن بدون کروشه و بدون اندیس، به عنوان آرگومان واقعی، در فراخوانی تابع ظاهر گردد. در تعریف تابع نیز باید آرگومان فرمال متناظر آن به همان طریق نوشته شود و به عنوان آرایه تعریف گردد. بدین طریق که نام آرایه همراه با یک زوج کروشهٔ بدون اندیس نوشته شود و نوع داده آن نیز مشخص گردد.

ـ **مثال ۷ـ۷** قطعه برنامهٔ زیر نحوهٔ فرستادن یا گذردادن آرایه را به تابع نشان می‌دهد.

```
main()
  {
    int n ;        /* variable declaration */
    float avg ;  /* variable declaration */
    float list [100] ;      /* array definition */
    float average () ;     /* function declaration */
    ....
    avg = average (n , list) ;
    ....
  }
    float average (a , x)          /* function definition */
    int a ;        /* formal argument declaration */
    float x[ ] ;  /* formal argument (array) declaration */
      {
        ....
      }
```

ملاحظه می‌کنید که تابع فرعی average در داخل تابع اصلی فراخوانده شده است. این تابع دارای دو آرگومان است: یکی متغیر n که نوع آن int و معرف تعداد عناصر آرایه است. دیگری آرایهٔ یک‌بعدی list که نوع عناصر آن float است. همچنین می‌بینید که در فراخوانی تابع، آرایهٔ list به صورت متغیر ساده ظاهر شده است.

در تعریف تابع نیز در سطر اول دو آرگومان فرمال را که a و x نامیده شده‌اند مشاهده می‌کنید. سپس در دو سطر بعدی دو آرگومان مزبور به ترتیب به‌صورت int و آرایهٔ یک‌بعدی از نوع float توصیف شده‌اند. ملاحظه می‌کنید که بین آرگومان واقعی n و آرگومان فرمال a تناظر وجود دارد. به طریق مشابه تناظری بین آرگومان واقعی list و آرگومان فرمال x وجود دارد. در واقع لازم نیست که آرگومانهای واقعی با آرگومانهای فرمال همنام باشند و به همین دلیل آرگومانهای فرمال را متغیرهای مجازی یا ساختگی نیز می‌نامند. همچنین ملاحظه می‌کنید

که بزرگی آرایه x، در توصیف آرگومان فرمال، مشخص نشده است. همین طور مشاهده می‌کنید که در تعریف تابع به صورت پیش‌نمونه یا prototype در تابع اصلی، پس از نام تابع فقط یک زوج پرانتز تهی به کار رفته است.

حال اگر در تعریف تابع فرعی، توصیف آرگومانهای آن نیز در همان خط اول تابع، پس از ذکر نام تابع در درون زوج پرانتز انجام گیرد، باید در تعریف پیش‌نمونه در تابع اصلی نیز این تناظر محفوظ بماند؛ یعنی باید پس از نام تابع آرگومانهای آن نیز در درون زوج پرانتز توصیف گردند (برخلاف حالت قبل که فقط یک زوج پرانتز تهی به کار می‌رفت). قطعه برنامهٔ زیر همان مثال قبلی را با این شیوه نشان می‌دهد.

```
main
  {
    int n ;        /* variable declaration */
    float avg ;  /* variable declaration */
    float list [100] ;      /* array definiation */
    float average (int , float[ ]) ;          /* function declaration */
   ....
    avg = average (n , list) ;
   ....
  }
float average (int a , float x [ ])          /* function definition */
  {
   ....
   ....
  }
```

در انتقال نام آرایه به تابع، آدرس اولین عنصر آرایه به تابع منتقل می‌شود؛ یعنی نام آرایه، آدرس اولین عنصر آرایه تفسیر می‌شود. وقتی که تابع فراخوانی می‌شود، این آدرس به آرگومان فرمال متناظر آن اختصاص می‌یابد. پس آرگومان مزبور به عنوان اشاره‌گر به اولین عنصر آرایه برمی‌گردد. بنابراین هر عملی که در تابع فرعی روی آرایه انجام گیرد، نتیجهٔ آن در تابع اصلی (یا تابع فراخواننده) نیز منعکس می‌شود. به طوری که بیان شد، این شیوهٔ فراخوانی تابع را فراخوانی با آدرس نامند.

وقتی که در درون تابع فراخوانده شده به عنصر آرایه مراجعه می‌شود، اندیس عنصر مورد نظر به مقدار اشاره‌گر افزوده می‌گردد تا به آدرس آن عنصر در درون آرایه دلالت نماید.

پس هر عنصری از آرایه به سادگی در درون تابع در دسترس قرار می‌گیرد و به طوری که گفتیم هر عنصر آرایه که در درون تابع تغییر یابد، اثر این تغییر در تابع فراخواننده نیز منعکس خواهد شد.

ـ **مثال ۷ـ۸** برنامهٔ زیر برنامهٔ ساده‌ای را نشان می‌دهد که آرایه‌ای ۳ عنصری را به تابع گذر می‌دهد و مقادیر عنصر آرایه در درون آن تابع تغییر می‌یابد. مقادیر عناصر آرایه در سه موقعیت از برنامه نوشته شده است تا اثر این تغییرات را نشان دهد.

```
# include<stdio.h>
main ()
  {
    int count , a [3] ;     /* array definition */
    void modify (int a[ ]) ;        / * function declaration */
    printf("\n from main , before calling the function: \n") ;
    for (count = 0 ; count<=2 ; + + count)
      {
        a[count] = count + 1 ;
        printf ("a[%d] = %d\n" , count , a[count]) ;
      }
    modify(a) ;
    printf ("\n from main , after calling the function:  \n") ;
    for (count = 0 ; count<=2 ; ++ count)
        printf ("a[%d] = %d\n" , count , a[count]) ;
  }
void modify (int a[ ])   /* function definition */
  {
    int count ;
    printf ("\n from the function , after modifying the values: \n") ;
     for (count = 0 ; count< = 2 ; ++ count)
      {
        a[count] = -9 ;
        printf ("a[%d] = %d" , count , a [count]) ;
      }
    return ;
  }
```

در اولین حلقه‌ای که در درون تابع اصلی ظاهرمی‌گردد، مقادیر a [0] = 1 , a [1] = 2 , a [2] = 3 به عناصر آرایه نسبت داده می‌شود و همین مقادیر با دستورprintf نمایش می‌یابد. سپس، آرایهٔ مزبور به تابع modify گذر داده می‌شود که در درون تابع مزبور به هریک از

عناصر آرایه مقدار ۹- نسبت داده می‌شود. سپس همین مقادیر جدید در آن تابع چاپ می‌گردد. بالاخره پس از برگشت کنترل از تابع فرعی به تابع اصلی، دوباره مقادیر عناصر آرایه نمایش می‌یابد.

اگر برنامهٔ مزبور اجرا گردد، خروجی زیر را خواهیم داشت.

```
from main , before calling the function:
a [0] = 1
a [1] = 2
a [2] = 3
from the function , after modifying the values:
a [0] = -9
a [1] = -9
a [2] = -9
from main , after calling the function:
a [0] = -9
a [1] = -9
a [2] = -9
```

این نتایج نشان می‌دهد که تغییرات اعمال شده با تابع modify روی آرایه، در تابع اصلی نیز منعکس شده است.

آرایه‌ها و رشته‌ها

در زبان C رشته‌ها، آرایه‌ای از کاراکترها تعریف می‌شوند به طوری که هر کاراکتر رشته، درون یک عنصر از آرایه ذخیره می‌گردد. هر رشته به کاراکتر null که نشانهٔ پایان است خاتمه می‌یابد. ثابت رشته‌ای در داخل دوبل گیومه قرار می‌گیرد و وقتی که ذخیره می‌گردد، کاراکتر null به طور خودکار به انتهای آن افزوده می‌شود. در داخل یک برنامه، کاراکتر null در فرم escape sequence با '\0' نشان داده می‌شود. ثابت رشته‌ای آرایه‌ای را معرفی می‌کند که محدوده یا اندیس پایین آن صفر و محدوده یا اندیس بالای آن تعداد کاراکترهایی است که در رشته وجود دارد.

ـ **مثال ۷ـ۹** به رشتهٔ زیر توجه کنید.

```
" payam noor university "
```

این رشته آرایه‌ای ۲۲ کاراکتری است (دو فضای خالی بین کلمات و یک 0\ نیز کاراکتر

شمرده می‌شوند). تعریف این رشته ممکن است در آرایه‌ای کاراکتری به شکل زیر باشد.

```
char str[ ] = " payam noor university "
```

متغیرهای رشته‌ای متغیرهایی‌اند که مقادیر آنها ممکن است یک رشته باشد. در واقع متغیرهای رشته‌ای، آرایه‌هایی از نوع char اند.

ـ **مثال ۷ـ۱۰** برنامهٔ زیر ۱۵ رشته (مثلاً اسامی ۱۵ نفر) را می‌خواند و آنها را در سطرهای متوالی چاپ می‌کند. فرض بر این است که طول هر رشته با درنظر گرفتن null character پایان رشته حداکثر ۱۲ کاراکتر است.

```
# include<stdio.h>
main ()
  {
     char name[12] ;
     int i ;
     for (i = 1 , i < = 15 ; ++i)
       {
         scanf ("%s" , name) ;
         printf ("\n %s" , name) ;
       }
  }
```

ملاحظه می‌کنید که متغیر رشته‌ای name به صورت آرایه معرفی شده است. در ضمن در خواندن اطلاعات به کمک تابع scanf، فقط نام متغیر، بدون اپراتور آدرس (یعنی بدون علامت '&') و بدون ذکر اندیس آرایه، به کار می‌رود، زیرا به طوری که گفتیم، نام آرایه آدرس اولین خانه یا اولین عنصر آن است. همچنین هم در موقع خواندن مقدار برای name و همین طور در چاپ مقدار آن از کد فرمت %s استفاده شده است.

می‌توان مجموعه‌ای از رشته‌ها را به صورت آرایه‌ای از رشته‌ها تعریف کرد. در اینجا چون خود رشته به تنهایی یک آرایه است، هر مجموعه از رشته‌ها می‌توانند آرایه‌ای دوبعدی را تشکیل دهند که اگر آنها را به صورت جدول مستطیل شکل تصور کنیم، هر سطر این جدول معرف یکی از رشته‌ها خواهد بود.

ـ **مثال ۷ـ۱۱** قطعه برنامهٔ زیر اسامی ۱۵ نفر را به آرایهٔ رشته‌ای name می‌خواند و آنها

را در سطرهای متوالی چاپ می‌کند.

```
# include<stdio.h>
main ()
  {
    char name[15][12] ;
    int i ;
    for (i = 0 ; i<15 ; ++i)
      scanf ("%s" , &name[i]) ;
    for (i = 0 ; i<15 ; ++i)
      printf ("\n%s" , name[i]);
  }
```

ملاحظه کنید که در خواندن اطلاعات رشته‌ای به آرایه و در چاپ کردن آن فقط بعد اول آرایه به کار برده شده است؛ یعنی در واقع اگر آرایهٔ دو بعدی در این مجموعه از رشته‌ها را، همان طور که گفتیم، جدولی مستطیل شکل فرض کنیم که در هر سطر آن یک رشته قرار دارد، در خواندن اطلاعات هر بار سطری خوانده می‌شود که آدرس آن سطر و در نتیجه اپراتور آدرس نیز به کار رفته است. به این طریق مشابه در چاپ کردن آن نیز هر بار اطلاعات یک سطر که شامل یک رشته است چاپ می‌گردد.

روشهای مرتب سازی

یکی از کاربردهای متداول آرایه‌ها، استفاده از آنها در روشهای مرتب سازی است. چنانچه مجموعه‌ای از داده‌ها یا اطلاعات، براساس ویژگی یا نظم خاصی سازمان‌دهی شوند، این عمل را مرتب‌سازی گویند. در این صورت پیدا کردن عنصری دلخواه در درون آن مجموعه، ساده‌تر و سریع‌تر انجام می‌گیرد. بنابراین عمل مرتب کردن، به منظور سرعت بخشیدن به عمل جستجوست.

تعداد مقایسه‌ها و نیز تعداد جابه‌جایی عناصر، از عوامل اساسی در مورد سرعت مرتب‌سازیهایند. طبیعی است که هر روش مرتب‌سازی که سرعت بالا و منطق ساده‌تری داشته باشد و همچنین حافظهٔ کمتری اشغال کند مطلوب‌تر است. بر این اساس روشهای متعددی مطرح شده که در ادامه چند نمونه را بررسی می‌کنیم.

روش مرتب‌سازی حبابی

این روش از نظر منطق، ساده‌ترین و از نظر کارآیی پایین‌ترین روش مرتب‌سازی اطلاعات به شمار می‌رود. در این روش ابتدا عنصر اول با عنصر دوم مقایسه می‌شود. اگر عنصر اول بزرگ‌تر از دومی باشد، جای آن دو تعویض می‌شود. سپس این عمل در مورد عنصر دوم با سوم و همین‌طور برای بقیهٔ عناصر به صورت متوالی انجام می‌گیرد؛ یعنی، در پایان با عنصر n-1 اُم مقایسه می‌شود که اگر بزرگ‌تر از آن بود جایشان تعویض می‌شود. وقتی که این عمل یک بار کامل شد، بزرگ‌ترین عنصر آرایه به آخر آرایه منتقل می‌شود. حال اگر خانهٔ آخر را نادیده بگیریم و این عمل را دوباره برای خانه‌ها یا عناصر ۱ تا n-1 تکرار کنیم، این بار بزرگ‌ترین عنصر خانه‌ها ۱ تا n-1 (که دومین عنصر بزرگ آرایه خواهد بود) به خانهٔ n-1 آرایه منتقل می‌شود. اگر این عمل را به صورت تکراری برای n-1 بار انجام دهیم و برای سرعت عمل بیشتر، در هر بار تکرار نیز خانهٔ آخر محدودهٔ جاری آرایه را نادیده بگیریم، یعنی هر بار از طول ناحیهٔ مورد مقایسه یک واحد کاسته شود، در پایان، عناصر آرایه به صورت صعودی مرتب می‌گردد. اگر بخواهیم که عناصر آرایهٔ مورد نظر به‌صورت نزولی مرتب شود، باید در مقایسهٔ دو عنصر متوالی a_i با a_{i+1}، چنانچه a_i کوچک‌تر از a_{i+1} باشد، جای آن دو با یکدیگر تعویض شود.

از آنجایی که در این روش مرتب سازی در هر بار تکرار، بزرگترین عنصر آرایه به سمت بالا یا انتهای آرایه حرکت می‌کند (مانند حبابهای هوا که در آب جوش موجود است)، آن را مرتب سازی حبابی نامند.

ـ **مثال ۷ـ۱۲** تابع زیر آرایهٔ n عنصری A را به روش حبابی مرتب می‌کند.

```
void BubbleSort (int A[ ] , int n)
  {
    int i , j , temp ;
    for (i =1 ; i<n ; + +i)
       for (j =0 ; j<n-i ; + +j)
          if (A[ j] > A[ j+1])
             {
               temp = A [ j] ;
               A[ j] = A[ j+1] ;
               A[ j+1] = temp ;
             }
```

```
  }
```

روش مرتب سازی انتخابی

در این روش مرتب سازی، شیوهٔ کار بدین صورت است که محل بزرگ‌ترین عنصر را در درون آرایهٔ n عنصری مورد نظر می‌یابیم و جای آن را با عنصر آخر یعنی a_n عوض می‌کنیم؛ سپس بین عناصر a_1 تا a_{n-1} محل بزرگ‌ترین عنصر را (که در واقع دومین عنصر بزرگ آرایهٔ اصلی خواهد بود) به‌دست می‌آوریم و جای آن را با عنصر a_{n-1} عوض می‌کنیم. این کار را n-1 بار تکرار می‌کنیم. در این صورت در تکرار آخر فقط دو عنصر a_1 و a_2 را خواهیم داشت که باید بزرگ‌ترین آن دو در خانهٔ a_2 قرار گیرد.

برتری این روش نسبت به روش مرتب‌سازی حبابی آن است که تعداد تعویض عناصر کمتر می‌شود، زیرا در این روش، در هر تکرار حداکثر یکبار جابه‌جایی انجام می‌گیرد.

ـ **مثال ۷ـ۱۳** تابع زیر آرایهٔ n عنصری A را به روش انتخابی مرتب می‌کند.

```
void SelectionSort (int a[ ] , int n)
  {
    int i , max , temp ;
    for (i = 1 ; i<n ; + + i)
       {
         max = 0 ;
         for (j =1 ; j<= n-i+1 ; + + j)
           if (A[ j] > A[max])
              max = j ;
           if (max != n-i)
              {
                temp = A[max] ;
                A[max] = A[n-i] ;
                A[n-i] = temp ;
              }
       }
  }
```

روشهای جستجو

یکی دیگراز کاربردهای متداول آرایه‌ها، استفاده از آنها در روشهای جستجوست. هر گاه در

داخل مجموعه‌ای از عناصر، دنبال عنصر خاصی بگردیم، این عمل را جستجو کردن نامند. جستجو، در اغلب زمینه‌ها، بـویژه در مورد بانکهای اطلاعاتی و سیستمهای تجاری، کاربرد زیادی دارند.

شیوه‌های متعددی برای جستجو وجود دارد که دو روش متداول آن را در ادامه بررسی می‌کنیم.

جستجو به روش خطی

این روش ساده‌ترین راه برای جستجو در آرایه یا جدول نامرتب است. برای این کار عنصر مورد جستجو را به طور متوالی با عناصر اول تا n اُم آن جدول مقایسه می‌کنیم. چنانچه در حین مقایسه، عنصر مزبور پیدا شد، شمارهٔ آن یادداشت می‌شود و عمل جستجو خاتمه می‌یابد. در غیر این صورت پیغام مناسب صادر می‌شود. در این روش رابطهٔ بین تعداد عناصر جدول و تعداد مقایسه‌ها به صورت معادلهٔ درجه اول خواهد بود.

ـ **مثال ۷ـ۱٤** تابع زیر عنصر x را در آرایهٔ n عنصری A به روش جستجوی خطی جستجو می‌کند. اگر پیدا شد، اندیس آن، و در غیر این صورت مقدار صفر را برمی‌گرداند.

```
int LinearSearch (int A[ ] , int n , int x)
  {
    int i ;
    for (i = 0 ; i < n ; + + i)
      if (x = = A[i])
        return (i +1) ;
    return (0) ;
  }
```

جستجو به روش دودویی

روش دیگر برای جستجوی عنصری در داخل جدول یا آرایه روش دودویی است. این روش در صورتی امکان‌پذیر است که عناصر جدول مورد نظر مرتب شده باشد. همچنین، در مقایسه با روش قبلی، از سرعت بیشتری برخوردار است.

در این روش، عنصر اول و عنصر آخر جدول را با دو متغیر مانند L و H نمایش می‌دهیم و سپس عنصر مورد جستجو را با عنصر وسطی جدول یا m = (L + H) / 2 مقایسه

می‌کنیم. اگر مساوی بود، عنصر مورد جستجو پیدا شده است. در غیر این صورت، چنانچه عنصر مورد جستجو از عنصر وسطی بزرگ‌تر باشد، L را مساوی m+1 وگرنه H را مساوی m-1 قرار می‌دهیم. سپس این عملیات را باز هم تکرار می‌کنیم؛ یعنی باز هم عنصر وسطی جدول حاصل را به‌دست می‌آوریم و عنصر مورد جستجو را با مقدار آن مقایسه می‌کنیم. این عمل تا موقعی که شرط $L \leq H$ برقرار نباشد، ادامه می‌یابد و پیغامی مبنی بر عدم وجود عنصر مورد جستجو در داخل جدول مورد نظر چاپ می‌گردد.

ـ **مثال ۷ـ۱۵** مراحل الگوریتم جستجو به روش دودویی برای ده عدد زیر در جدول نمایش داده شده است.

65 , 141 , 192 , 205 , 218 , 389 , 424 , 500 , 538 , 567

جستجوی عدد ۲۰۵				جستجوی عدد ۵۶۷			
X	l	h	m	X	l	h	m
218	1	10	5	218	1	10	5
141	1	4	2	500	6	10	۸
192	3	4	3	538	9	10	9
205	4	4	4	567	10	10	10

ـ **مثال ۷ـ۱۶** تابع زیر در آرایهٔ مرتب شدهٔ n عنصری، به روش دودویی عنصر x را جستجو می‌کند. اگر پیدا شد، اندیس آن و در غیر این صورت مقدار صفر را برمی‌گرداند.

```
int BinarySearch (int A[ ] , int n , int x)
  {
   int middle , L , H ;
   L = 0 ;
   H = n-1 ;
   while (L <= H)
      {
        middle = (L+H)/2 ;
        if (x = = A[middle])
          return (middle +1) ;
        if (x >A[middle])
          L = middle +1 ;
        else
          H = middle -1 ;
      }
   return (0) ;
  }
```

توابع کتابخانه‌ای (در مورد رشته‌ها)

اغلب نسخه‌های زبان C، مجموعهٔ وسیعی از توابع کتابخانه‌ای در مورد عملیات روی رشته‌ها را پشتیبانی می‌کنند که در مبحث توابع کتابخانه‌ای بحث کردیم. چند تابع بسیار متداول که در نوشتن برنامه‌ها کاربرد زیادی دارند در جدول ۷ـ۱ آمده است.

جدول ۷ـ۱ چند تابع متداول در برنامه‌های کاربردی

نام تابع	عمل تابع
strcpy (s1 , s2)	رشتهٔ s2 را روی رشتهٔ s1 کپی می‌کند.
strcat (s1 , s2)	رشتهٔ s2 را به دنبال رشتهٔ s1 ملحق (ضمیمه) می‌کند.
strlen (s)	طول رشتهٔ s را برمی‌گرداند.
strcmp (s1, s2)	رشتهٔ s1 را با رشتهٔ s2 مقایسه می‌کند.

اگر بخواهیم درمورد مقایسهٔ دو رشته، تمایز بین حروف بزرگ و کوچک نادیده گرفته شود، تابع strcmp را به صورت strcmpi به کار می‌بریم.

در ضمن یادآور می‌شویم که توابع کتابخانه‌ای مربوط به رشته‌ها در فایل string.h قرار دارند. پس اگر بخواهیم در برنامه‌ای از این تابع استفاده کنیم، باید دستور # include<string.h> را نیز قبل از تابع main به کار بریم.

حال به چند مثال در مورد رشته‌ها توجه کنید.

ـ **مثال ۷ـ۱۷** برنامهٔ زیر نحوهٔ استفاده و کاربرد چند تابع کتابخانه‌ای را نشان می‌دهد.

```
# include<string.h>
# include<stdio.h>
main ()
  {
    char s1[80] , s2[80] , s3[80] ;
    gets (s1) ;
    gets (s2) ;
    printf ("\n Lengths:  %d %d\n" , strlen (s1) , strlen (s2)) ;
    if (!strcmp (s1 , s2))
      printf ("\n The strings are equal\n") ;
    strcpy (s1 , s3) ;
    strcat (s1 , s2) ;
```

```
    printf ("\n %s" , s3) ;
    printf ("\n %s" , s1) ;
  }
```

این برنامه دو رشته از ورودی می‌خواند و آن دو را به هم متصل می‌کند.اگر این برنامه را اجرا و برای دو رشتهٔ s1 و s2 کلمهٔ "computer science" را وارد کنیم، خروجی برنامهٔ مزبور به صورت زیر خواهد بود.

```
Lengths:  16     16
The strings are equal
computer science
computer sciencecomputer science
```

ـ **مثال ۷ـ۱۸** برنامه‌ای بنویسید که اسامی n نفر دانشجو را بخواند، سپس آنها را به ترتیب الفبا مرتب و چاپ کند. n، حداکثر ۱۵ است که با اولین دستور ورودی خوانده می‌شود.

```
# include<stdio.h>
# include<string.h>
main ()
  {
   int i , n , j ;
   char name [15][12] , temp [12] ;
   scanf ("%d" , &n) ;
   for (i=0 ; a<n ; ++i)
      scanf ("%s" , &name[i]) ;
    for (i=1 ; i<n ; ++i)
      for (j=0 ; j<n-i ; ++j)
        if (strcmp (name[j] , name [j+1]) > 0)
          {
            strcpy (temp , name [j]) ;
            strcpy (name[j] , name [j+1]) ;
            strcpy (name[j+1] , temp) ;
          }
    for (i=0 ; i<n ; ++i)
      printf ("\n %s" , name[i]) ;
  }
```

در این برنامه از آرایه‌های دوبعدی، روش مرتب سازی حبابی و چند تابع کتابخانه‌ای رشته‌ای استفاده شده است.

خودآزمایی ۷

۱. با استفاده از آرایه برنامه‌ای بنویسید که جمله‌های اول تا دهم سری فیبوناچی را محاسبه و چاپ کند.

۲. برنامه‌ای بنویسید که اعداد زوج ۲ تا ۲۰ را به عناصر آرایه‌ای ۱۰ عنصری نسبت دهد و سپس شمارهٔ هر عنصر و مقدار متناظر آن را در دو ستون نشان دهد و چاپ کند.

۳. برنامهٔ زیر آرایه‌ای ۱۰ عنصری را، هنگام تعریف آن، مقداردهی اولیه می‌کند. سپس مجموع مقادیر آن را محاسبه و چاپ می‌کند. خروجی این برنامه چیست؟

```
# include<stdio.h>
# define size 10
main()
  {
    int i , total = 0 ;
    int A[size] = {10 , 9 , 8 , 7 , 6 , 5 , -5 , -6 , -7 , -8} ;
    for (i = 0 ; i<size ; i++)
      total += A[i] ;
    printf (" Sum = %d" , total) ;
  }
```

۴. برنامه‌ای بنویسید که عددی صحیح را دریافت کند و معادل آن را به شکل باینری (در مبنای ۲) چاپ کند.

۵. برنامه‌ای بنویسید که متنی را از ورودی بخواند وحروف کوچک را به حروف بزرگ تبدیل کند.

٦. برنامه‌ای بنویسید که با استفاده از تابع کتابخانه‌ای rand که اعداد تصادفی ایجاد می‌کند، ٦۰۰۰ بار پرتاب تاس تخته نرد را شبیه‌سازی کند.

۷. تفاوت بین فراخوانی با مقدار و فراخوانی با آدرس (یا فراخوانی با ارجاع) در این فصل به اختصار بیان شد. برنامه‌ای بنویسید که تفاوت انتقال (ارسال) تمامی آرایه به یک تابع (یعنی درواقع ارسال آدرس آرایه به تابع) یا فراخوانی با آدرس را با انتقال عنصری از آرایه به تابع، یعنی فراخوانی با مقدار، نشان می‌دهد، به‌طوری که در حلقهٔ for اول مقادیر آرایه چاپ گردد. سپس تابع modifyarray(a) فراخوانی شود. در این فراخوانی، نام آرایه (یعنی آدرس آغاز آرایه) به تابع انتقال یابد. تابع مزبور مقدار هریک از عناصر آرایه را دو برابر کند. پس از

برگشت کنترل به تابع اصلی، مقادیر آرایه چاپ گردد. نتیجهٔ عملکرد تابع فرعی روی آرایه در تابع اصلی نیز منعکس شود. سپس با فراخوانی تابع modifyelement یک عنصر آرایه، به آن تابع انتقال یابد (که درواقع فراخوانی با مقدار است). تابع مزبور مقدار عنصر دریافتی را دوبرابر کند و نتیجه چاپ گردد. پس از برگشت مجدد کنترل به تابع اصلی، مقدار عنصر مزبور دوباره چاپ شود و نتیجهٔ عملکرد تابع modifyelement در تابع اصلی منعکس نشود.

۸. در علم آمار، سه پارامتر میانگین، میانه و مد مجموعه‌ای از مقادیر به شکل زیر تعریف می‌شوند.

الف) میانگین (mean) n مقدار a_1 , a_2 ,... , a_n که آن را میانگین حسابی نیز نامند، برابر است با:

$$\text{mean} = \frac{a_1 + a_2 + \ldots + a_n}{n} = \frac{\sum_{i=1}^{n} a_i}{n}$$

ب) اگر عناصر را به صورت صعودی مرتب کنیم، چنانچه تعداد عناصر فرد باشد، مقدار عنصر وسطی را میانه (media) نامند و چنانچه تعداد عناصر زوج باشد، نصف مجموع دو مقدار وسطی را میانه نامند.

ج) مد (mode) عبارت از عنصری است که بیشتر از بقیه تکرار شده باشد.

نتیجه عبارت است از یک پرسش از ۹۹ نفر که مقادیر عددی صحیح بین ۱ تا ۱۰ است. برنامه‌ای بنویسید که ۳ پارامتر میانگین، میانه و مد را محاسبه و چاپ کند و همچنین نمودار میله‌ای یا هیستوگرام پاسخها را به صورت ستاره رسم کند.

۹. برنامه‌ای بنویسید که ۱۰۰ عدد را به حافظه بخواند. سپس عددی را از طریق ورودی دریافت کند و با فراخوانده شدن تابعی، عدد دریافتی در درون مجموعه اعداد جستجو شود. اگر پیدا شد، تابع مزبور شمارهٔ آن را برگرداند. در غیر این صورت مقدار ۱- برگرداند. سپس در تابع اصلی پیغام مناسبی مبنی بر اینکه عدد مورد نظر پیدا شده است یا نه چاپ شود.

۱۰. برنامه‌ای بنویسید که عناصر دو ماتریس a و b را که دارای m سطر و n ستون باشند، به حافظه بخواند و سپس مجموع آن دو را براساس قانون جمع ماتریسها به دست آورد و چاپ کند. m و n حداکثر ۸ اند که با اولین دستور ورودی خوانده می‌شوند.

۱۱. برنامه‌ای بنویسید که عناصر دو ماتریس a و b را به حافظه بخواند و سپس حاصل ضرب آن دو را براساس قانون ضرب ماتریسها به دست آورد و نتیجه را به فرم ماتریس (آرایهٔ دوبعدی) چاپ کند. ماتریس a دارای m سطر و n ستون و ماتریس b، دارای n سطر و h ستون است. m و n و h حداکثر ۷ اند که از اولین دستور ورودی خوانده می‌شوند.

۱۲. برنامه‌ای بنویسید که یک سطر متن را با استفاده از تابع getchar و هر بار یک کاراکتر به یک آرایهٔ کاراکتری بخواند. سپس آرایهٔ کاراکتری مزبور را به صورت رشته چاپ کند.

۱۳. تابعی بنویسید که طول رشته‌ای را به دست آورد و برگرداند.

۱٤. تابعی بنویسید که بدون استفاده از تابع کتابخانه‌ای strcat، دو رشته را به هم ملحق کند (یعنی رشتهٔ دوم را به آخر رشتهٔ اول ضمیمه کند).

۱۵. تابعی بنویسید که در درون یک رشته، زیر رشتهٔ دیگری را جستجو کند. اگر وجود داشت، محل آن (یعنی از چندمین کاراکتر رشته اول شروع شده است) را برگرداند، در غیر این صورت مقدار ۱- برگرداند.

۱٦. تابعی بنویسید که در درون رشتهٔ S1، از کاراکتر i اُم آن به طول j کاراکتر در رشته S2 کپی کند.

۱۷. برنامه‌ای بنویسید که مجموعه‌ای از رشته‌ها را به حافظه بخواند و سپس با فراخوانده شدن تابعی، رشته‌های مزبور به ترتیب الفبا مرتب کند. تعداد رشته‌ها حداکثر ۱۰ رشته است که پایان آن با کلمهٔ "END" مشخص شده است.

فصل ۸

اشاره‌گرها

هدف کلی

آشنایی با کاربردهای متعدد اشاره‌گرها در زبان C

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از مطالعهٔ این فصل،

۱. با تعریف و کاربردهای اشاره‌گرها آشنا شود.

۲. با نحوهٔ معرفی اشاره‌گرها در برنامه آشنا شود.

۳. کاربرد اپراتور یکانی & و عملگر * را بشناسد.

۴. با نحوهٔ آدرس‌دهی داده‌ها آشنا شود.

۵. نحوهٔ مقداردهی اولیه به اشاره‌گرها را بداند.

۶. کاربرد اشاره‌گر تهی را بشناسد.

۷. با سه عملیات انتساب، محاسبه و مقایسه بر روی اشاره‌گرها آشنا شود.

۸. نحوهٔ گذردادن آرگومانها با آدرس و با مرجع را بداند.

۹. با کاربرد انتقال دوطرفهٔ اطلاعات در اشاره‌گرها آشنا شود.

۱۰. رابطهٔ بین اشاره‌گرها و آرایه‌های تک‌بعدی و دوبعدی را بداند.

۱۱. با تعریف آرایه به صورت قراردادی آشنا شود.

۱۲. با نحوهٔ انتقال آرایه به تابع آشنا شود.

۱۳. نحوهٔ تعریف آرایه‌ای از اشاره‌گرها را بداند.

۱۴. مفهوم اشاره‌گر به اشاره‌گر با بشناسد.
۱۵. نحوهٔ ارسال اشاره‌گرها به آرگومان به توابع را بداند.

مقدمه

در اغلب زبانهای برنامه‌نویسی قدیمی، مانند فورترن و کوبول، مفهومی به نام اشاره‌گر وجود ندارد. اما یکی از ویژگیهای بارز زبان C، کاربرد متعدد اشاره‌گرها و انجام عملیات محاسباتی روی آنهاست. اشاره‌گر متغیری است که آدرس متغیر دیگری را در خود نگه می‌دارد؛ یعنی به آدرس متغیر دیگر اشاره می‌کند. به عبارت دیگر مقدار آن، آدرس یک خانه از حافظه است. اشاره‌گر روش غیرمستقیم دسترسی به داده‌هاست و کاربردهای زیادی در C دارد که از آن جمله می‌توان موارد زیر را عنوان کرد.

- انتقال آدرس متغیرها به تابع فرعی
- برگرداندن چندین مقدار از تابع فرعی
- دستیابی به عناصر آرایه‌ها
- تشکیل ساختارهای پیچیده‌تر مانند فهرستهای پیوندی، درختها و نمودارها
- عمل تخصیص حافظه به صورت پویا.

نحوهٔ معرفی اشاره‌گر

برای استفاده از اشاره‌گر در برنامه، ابتدا باید اشاره‌گر تعریف شود. روش کلی تعریف متغیری از نوع اشاره‌گر به صورت زیر است.

data-type * ptvar ;

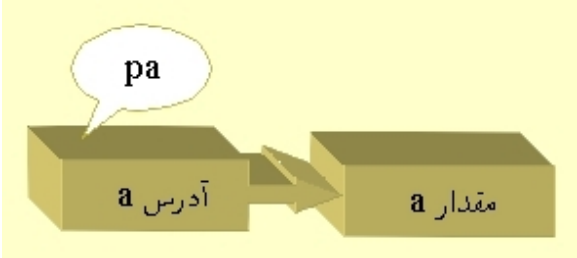
که در آن ptvar نام متغیر مورد نظر و data-type نوع متغیری است که آدرس آن در متغیر اشاره‌گر ptvar قرار می‌گیرد. نماد `*` نیز اپراتور اشاره‌گر است.

متغیرهای اشاره‌گر ممکن است به متغیرهای عددی، کاراکتری، آرایه‌ها، توابع، ساختارها یا دیگر متغیرهای اشاره‌گر اشاره کند. در حالت کلی هر نوع داده ذخیره شده در حافظهٔ کامپیوتر یک یا چند بایت متوالی از خانه‌های حافظه را اشغال می‌کند. در صورتی می‌توان به داده دسترسی داشت که آدرس اولین خانه یا اولین بایت آن را در حافظه بدانیم. آدرس محل

متغیر a در حافظه با عبارت &a تعیین می‌گردد که در آن & اپراتور یکانی یا تک‌اپراندی است و اپراتور آدرس نامیده می‌شود و آدرس اپراند یا عملوند خود را به دست می‌دهد. حال فرض کنید که متغیر a از نوع int و متغیر pa نیز متغیر اشاره‌گر باشد و به صورت زیر توصیف کرده باشیم.

```
int *pa ;
```

در این صورت با دستور جایگذاری ; pa = &a آدرس متغیر a به اشاره‌گر pa نسبت داده می‌شود. pa را اشاره‌گر a می‌نامند، زیرا به محلی از حافظه اشاره می‌کند که مقدار متغیر a در آن ذخیره شده است. به هرحال به‌خاطر بسپارید که pa مقدار a را معرفی نمی‌کند، بلکه آدرس a را معرفی می‌کند و به همین لحاظ آن را متغیر اشاره‌گر نامند. شکل ۸ـ۱ رابطهٔ بین pa و a را نشان می‌دهد.



شکل ۸ـ۱ رابطهٔ بین اشاره گر و متغیر

داده‌ای که با a معرفی می‌گردد (یعنی داده‌ای که در خانهٔ a از حافظه ذخیره شده است) با عبارت *pa در دسترس قرار می‌گیرد که در آن * اپراتور یکانی یا تک‌اپراندی است که فقط روی متغیرهایی از نوع اشاره‌گر عمل می‌کند. بنابراین a و *pa هر دو همان قلم داده (یعنی هر دو محتوای خانه‌های یکسان از حافظه) را معرفی می‌کنند. پس با اجرای دو دستور

```
pa = &a ;
k = *pa ;
```

k و a هر دو یک مقدار را معرفی خواهند کرد؛ یعنی مقدار a به طور غیرمستقیم به k نسبت داده خواهد شد. به عبارت دیگر، نتیجهٔ دو دستور مزبور مشابه نتیجهٔ دستور k = a است.

بنابراین عملگر * در مورد *pa محتوای محلی را برمی‌گرداند که آدرس آن در pa قرار

دارد و به همین لحاظ به آن عملگر غیرمستقیم نیز گویند.

آدرس داده‌ها

هر متغیری دارای آدرس منحصر به فردی است که محل آن متغیر را در حافظه مشخص می‌کند. در بعضی کاربردها بهتر است که برای دستیابی به متغیر به جای نام آن متغیر، از آدرس آن استفاده کرد. برای به دست آوردن آدرس متغیر، اپراتور ampersand یا & به کار می‌رود. برای مثال، فرض کنید که متغیر k از نوع long int و آدرس آن 1004 باشد، دستور Ptr = &A; مقدار 1004 (آدرس متغیر A) را در متغیر Ptr ذخیر می‌کند که البته باید Ptr از نوع اشاره‌گر توصیف شده باشد که به متغیری از نوع long int اشاره می‌کند.

ـ **مثال ۸ـ۱** برنامهٔ سادهٔ زیر مقدار و آدرس متغیر A را چاپ می‌کند.

```
# include <stdio.h>
main ()
   {
     int A = 5 ;
     printf (" The value of A is:  %d\n" , A) ;
     printf (" The address of A is:  %p\n" , &A) ;
   }
```

خروجی برنامه

```
The value of A is:  5
The address of A is:  1004
```

یادآور می‌شویم که در تابع printf برای چاپ آدرس متغیر از کد فرمت %p استفاده شده است. این کد فرمت ممکن است در کامپایلرهای قدیمی وجود نداشته باشد.

همچنین می‌توان قطعه برنامهٔ بالا را به صورت زیر نیز نوشت.

```
# include <stdio.h>
main ()
   {
     int A = 5 ;
     int *pA ;
     pk = &A ;
     printf ("the address of A is:  %p\n" , pA) ;
   }
```

خروجی این برنامه نیز مشابه قبلی خواهد بود.

از مطالب مطرح شده می‌توان نتیجه گرفت که عملگر ستاره یعنی * به دو مفهوم جداگانه به کار می‌رود.

الف) در معرفی متغیرهای عملگر، اشاره‌گر در سمت چپ متغیرهای مورد نظر قرار می‌گیرد، مانند مثالهای زیر.

```
int *p1 , *p2 , *p3 ;
float *p4 , *p5 ;
char *p6 , *p7 ;
```

ب) برای دستیابی به مقدار متغیری که آدرس آن در متغیر اشاره‌گر قرار دارد، مانند

```
p1 = &a ;
*p1 = a ;
```

ـ **مثال ۸ـ۲** به برنامهٔ زیر توجه کنید.

```
# include <stdio.h>
main ()
   {
     char * pch ;
     char ch1 = `Z´ , ch2 ;
     printf ("the address of pch is %p" , &pch) ;
     pch = &ch1 ;
     printf ("the value stored at pch is %p\n" , pch) ;
     printf ("the value stored at the address pointed by pch is %c\n" , *pch) ;
     ch2 = *pch ;
     printf ("the value stored at ch2 is %c\n" , ch2) ;
   }
```

خروجی برنامه

```
the address of pch is 1004
the value stored at pch is 2001
the value stored at the address pointed by pch is Z
the value stored at ch2 is Z
```

در این برنامه متغیر pch اشاره‌گری به متغیرهایی از نوع کاراکتر توصیف شده است. متغیرهای ch1 و ch2 نیز از نوع کاراکتر اعلان شده‌اند که به متغیر ch1 مقدار اولیهٔ کاراکتر `a´

نسبت داده شده است. در دستور printf اول آدرس متغیر pch چاپ می‌گردد که به فرض ۱۰۰٤ است. سپس آدرس متغیر ch1 به pch نسبت داده می‌شود. در دستور printf دوم، مقدار pch (آدرس متغیر ch1) که به فرض ۲۰۰۱ است چاپ می‌گردد. در دستور printf سوم، محتوای خانه‌ای از حافظه که آدرس آن در متغیر pch قرار دارد (یعنی مقدار متغیر ch1) و کاراکتر `a´ است چاپ می‌شود. سپس در خط بعدی همین مقدار (یعنی حرف `a´) به متغیر ch2 نسبت داده می‌شود. بالاخره با دستور printf آخری مقدار متغیر ch2 (که همان حرف `a´ است) چاپ می‌گردد.

نکته. عبارت *pch در سمت چپ دستور جایگذاری نیز ظاهر می‌شود. مثلاً در همان برنامهٔ بالا پس از اجرای دستور ; pch = & ch1 با دستور ; `b´ = *pch کاراکتر b به متغیر ch1 نسبت داده می‌شود.

مقداردهی اولیه به اشاره‌گر

به هر نوع متغیر از نوع اشاره‌گر می‌توان هنگام اعلان آنها، مشابه سایر متغیرها، مقدار اولیه نیز نسبت داد. در این صورت مقدار اولیه مورد نظر باید یک آدرس باشد. پس اشاره‌گر NULL یا یک آدرس را به عنوان مقدار اولیه می‌پذیرد. برای مثال می‌توان دستورهایی به صورت زیر نوشت.

```
int x ;
int *px = &x ;
```

اما نمی‌توان متغیری را قبل از اینکه توصیف یا اعلان گردد در دستوری به کار برد. بنابراین مجموعه دستورهای زیر قابل قبول نیست.

```
int *px = &x ;
int x ;
```

همچنین می‌توان اشاره‌گر را به صورت ; int *ptr = 0 مقداردهی اولیه کرد که برای مشخص ساختن بعضی شرایط خاص به کار برده می‌شود.

در حالت کلی، نسبت دادن مقدار صحیح به متغیر اشاره‌گر مفهوم ندارد. به هرحال، مثال

اخیر حالت استثنایی در این مورد است که همان طور که گفتیم، برای مشخص ساختن بعضی شرایط خاص به کار می‌رود. در چنین مواردی توصیه می‌گردد که ثابت سمبولیکی مانند NULL را که معرف صفر باشد تعریف کرد و آن را به اشاره‌گر اختصاص داد. این روش تأکید می‌کند که اختصاص دادن صفر، معرف شرطی ویژه است.

ـ **مثال ۸ـ۳** برنامه‌ای به زبان C ممکن است تعاریف و عبارات زیر را شامل باشد.

```
# define NULL 0
float x , y ;
float *pr = NULL ;
```

در این مثال متغیرهای x و y به صورت متغیرهایی از نوع ممیز شناور و pr به صورت متغیر اشاره‌گر اعلان شده که مقداری ویژه به‌عنوان مقدار اولیه به آن نسبت داده شده است. بنابراین استفاده از ثابت سمبولیک NULL نشان می‌دهد که این اختصاص مقدار اولیه، چیزی به غیر از اختصاص مقدار صحیح معمولی است. به هرحال در اغلب کامپایلرهای C ثابت سمبولیک NULL در چندین header file و بویژه در <stdio.h> تعریف شده است. پس اختصاص مقدار اولیهٔ صفر یا NULL به یک اشاره‌گر هم‌ارز است، ولی NULL ترجیح داده می‌شود.

ـ

اشاره‌گر تهی

زبان C مفهوم اشاره‌گر NULL را پشتیبانی می‌کند و آن اشاره‌گری است که به هیچ شئی قابل‌قبول یا معتبر اشاره نمی‌کند. اشاره گر NULL هر اشاره‌گری است که مقدار صحیح صفر به آن نسبت داده شده باشد.

ـ **مثال ۸ـ٤** در مثال زیر اشاره‌گر p مقدار صفر دارد.

```
char *p ;
p = 0 ;
```

ـ

اشاره‌گر NULL بویژه در دستورهای مربوط به کنترل جریان مفید است، زیرا اشاره‌گرهایی با مقدار صفر false درنظر گرفته می‌شوند، درحالی که متغیرهای اشاره‌گر با سایر مقادیر true منظور می‌گردند.

ـ **مثال ۸ـ۵** در برنامهٔ زیر حلقهٔ while تا موقعی که p اشاره‌گر NULL نباشد، عمل تکرار را ادامه می‌دهد.

```
char *p ;
 ....
 ....
while (p)
 {
 ....
 ....
 }
```

این گونه کاربرد اشاره‌گرها، بویژه در کاربردهایی آشکار می‌گردد که آرایه‌هایی از اشاره‌گرها را به کار می‌برد، و در همین فصل بررسی می‌کنیم.

ـ

عملیات روی اشاره‌گرها

عملیات متداولی که روی اشاره‌گرها انجام می‌شوند عبارت‌اند از انتساب، محاسبه، و مقایسه که به شرح هر یک می‌پردازیم.

الف) انتساب

در مورد اشاره‌گرها نیز مشابه سایر انواع متغیرها، می‌توان مقداری را به متغیر اشاره‌گر نسبت داد. اما به طوری که گفتیم، این عمل مانند سایر متغیرها گسترده نیست. به متغیر اشاره‌گر می‌توان آدرس متغیر یا مقدار صفر نسبت داد.

ب) محاسبه

زبان C اجازه می‌دهد که مقدار صحیحی را به اشاره‌گر اضافه و یا از آن کسر کنید. برای مثال اگر p متغیر اشاره‌گر باشد، عباراتی مشابه p+5 و p-5 معتبر است. ولی باید به مفهوم آن دقت کافی شود. برای مثال مفهوم p+5 آن است که به پنج شئی بعد از شئی که p به آن اشاره می‌کند اشاره خواهد کرد. بنابراین اگر p آدرس متغیری از نوع short int p ; را داشته باشد که دو بایت حافظه اشغال می‌کند، عبارت p+5 به (۱۰ = ۲×۵) ۱۰ بایت بعد از آدرسی که p معرف

آن است اشاره خواهد کرد. حال اگر p آدرس متغیری از نوع ; float p را در خود داشته باشد، عبارت p+5 به (۲۰ = ٤×٥) ۲۰ بایت بعد اشاره خواهد کرد. بنابراین p+5 همیشه به مفهوم به اندازهٔ ۵ شئی بعد از آنکه p اشاره می‌کند خواهد بود. پس کامپایلر ۵ را در بزرگی یا طول شئی مورد نظر برحسب بایت ضرب می‌کند و آن را بر محتوای p اضافه می‌کند تا آدرس جدید به دست آید. بنابراین عملیات محاسباتی روی اشاره‌گرها، چهار عمل افزودن، کاستن، ++ و -- است.

اگر دو متغیر اشاره‌گر از یک نوع باشند، یعنی اشیایی که اشاره‌گرهای مزبور به آن اشاره می‌کنند یکسان باشند، می‌توان مقادیر آن دو اشاره‌گر را از هم تفریق کرد. برای مثال مقدار &a[3] - &a[0] برابر ۳ خواهد بود. در واقع این تفاضل تعداد اشیاء بین این دو اشاره‌گر را معرفی می‌کند.

ـ **مثال ۸ـ٦** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main ()
  {
     int *px , *py ;
     static int A[6] = {1 , 2 , 3 ,4 , 5 , 6};
     px = &A[0] ;
     py = &A[5] ;
     printf("px=%x py=%x" , px , py) ;
     printf("\n py – px =%x" , py - px) ;
  }
```

خروجی خط اول برنامه آدرس دو متغیر px و py بر حسب هگزادسیمال خواهد بود و خروجی خط دوم مقدار 5 است.

ـ

ج) مقایسه

متغیرهای اشاره‌گر را، که هر دو به داده‌هایی از یک نوع مشابه اشاره داشته باشند، می‌توان با هم مقایسه کرد. به عبارت دیگر دو اشاره‌گر را می‌توان در یک عبارت رابطه‌ای با یکدیگر مقایسه کرد.

ـ **مثال ۸ـ۷** فرض کنید px و py اشاره‌گرهایی باشند که به عناصری از یک آرایه اشاره می‌کنند. چند عبارت منطقی حاصل از این دو متغیر به این شکل خواهد بود.

```
px < py
px >= py
px = = py
px != py
px = = null
```

همچنین دستورهای زیر درست است.

```
if (px<py)
   printf ("px points to lower memory than py") ;
else
   printf ("px points to upper memory than py") ;
```

ـ

انتقال مقادیر به تابع

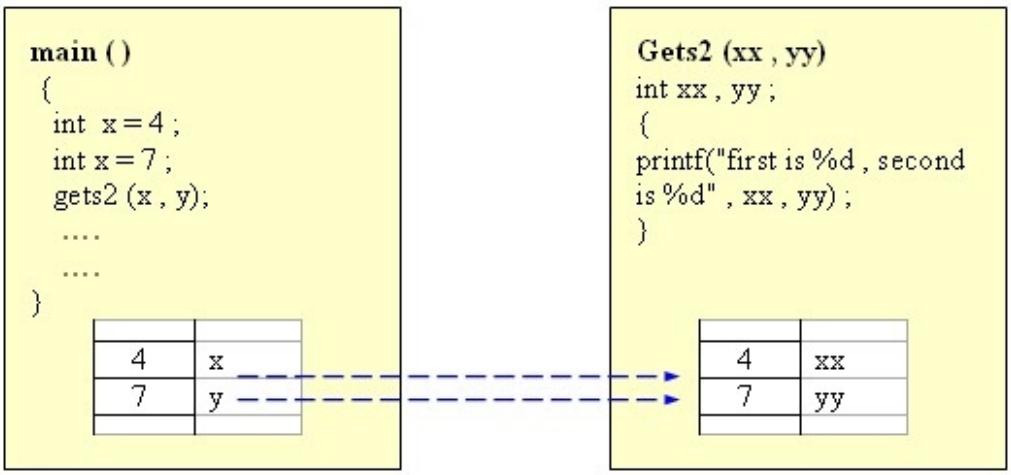
حال ببینیم وقتی مقادیری را به تابع گذر می‌دهیم چه اتفاقی رخ می‌دهد. در اینجا برنامهٔ ساده‌ای را ملاحظه می‌کنید که دو مقدار صحیح ٤ و ۷ را به تابعی به نام gets2 می‌فرستد.

```
main ()
   {
     int x = 4 , y = 7 ;
     gets2 (x , y) ;
   }
void gets2(xx , yy)        /* print out values of two arguments */
int xx , yy ;
   {
     printf ("first is %d , second is %d" , xx , yy) ;
   }
```

این تابع عمل خاصی انجام نمی‌دهد، فقط دو مقداری را که به آن گذر داده شده است چاپ می‌کند. اما تابع مزبور نکتهٔ مهمی را نشان می‌دهد و آن اینکه تابع دو مقدار از برنامهٔ فراخوانندهٔ آن دریافت می‌کند و آنها را به طور جداگانه، یعنی به‌صورت دوبله، در فضای حافظهٔ خاص خودش ذخیره می‌کند. حتی تابع می‌تواند به آن دو مقدار، اسامی متفاوتی (مانند مثال مورد نظر ما) که فقط در تابع مزبور شناخته شده است اختصاص دهد که در اینجا این اسامی جدید نیز xx و yy است و به جای x و y به کار رفته است (البته می‌توانست همان x و

y نیز به کار برده شود).

شکل ۸ـ۲ این سازوکار را نمایش می‌دهد. حال این تابع می‌تواند روی متغیرهای جدید xx و yy، بدون اینکه تأثیری روی x و y داشته باشد، هر عملی را انجام دهد (اگر اسامی یکسان انتخاب می‌شد، باز هم در شیوهٔ کار تغییری حاصل نمی‌شد).



شکل ۸ـ۲ انتقال مقادیر به تابع

انتقال اشاره‌گر به تابع

اشاره‌گرها اغلب به عنوان آرگومان به یک تابع فرستاده می‌شود. این امر اجازه می‌دهد که عناصر داده‌های برنامهٔ فراخواننده، که معمولاً تابع main است، با تابع فراخوانده شده قابل دستیابی باشند و در داخل تابع فراخوانده شده تغییر یابند و نتیجه در تابع یا برنامهٔ فراخواننده نیز اعمال گردد. این گونه کاربرد اشاره‌گر، گذر دادن آرگومانها با آدرس و یا گذر دادن آرگومانها با مرجع نامیده می‌شود.

هنگامی که آرگومانها با مقدار گذر داده می‌شوند، عناصر داده (مقدار داده) به تابع کپی می‌شوند. بنابراین هر گونه تغییرات اعمال شده در روی آنها در درون تابع یا روتین فراخواننده اثر نمی‌گذارد. اما وقتی که آرگومان به صورت آدرس انتقال می‌یابد (یعنی وقتی که اشاره‌گر به تابع گذر داده می‌شود)، در واقع آدرس آن قلم از داده به تابع فرستاده می‌شود. حال محتوای آن آدرس به راحتی هم در درون تابع مزبور و هم در تابع فراخواننده قابل دستیابی است.

به‌علاوه هر تغییراتی که روی آن قلم داده انجام پذیرد (یعنی هر گونه تغییراتی که در محتوای آدرس مورد نظر انجام گیرد) هم در تابع فراخوانده شده و هم در برنامه یا تابع فراخواننده تأثیر می‌گذارد و تشخیص داده می‌شود.

دو برنامهٔ نمایش داده شده در مثالهای ۸ـ۸ و ۸ـ۹ دو گونه از تابعی به نام add را نشان می‌دهد که آرگومان خود را یک واحد افزایش می‌دهد. مثال ۸ـ۸ متغیر count را با روش فراخوانی با مقدار، به تابع گذر می‌دهد. تابع add آرگومان خود را یک واحد افزایش می‌دهد و مقدار جدید را با دستور return به تابع main برمی‌گرداند. مقدار جدید در تابع main به count نسبت داده می‌شود. در مثال ۸ـ۹، برنامهٔ مورد نظر متغیر count را با فراخوانی آدرس گذر می‌دهد؛ یعنی آدرس count (نه مقدار آن) به تابع add گذر داده می‌شود. تابع add1 اشاره‌گری به مقدار صحیح را که در اینجا countptr نامیده شده است به عنوان آرگومان دریافت می‌کند. تابع add مقداری را که با countptr به آن اشاره شده است (یعنی محتوای محلی را که آدرس آن در اشاره‌گر countptr قرار دارد) یک واحد افزایش می‌دهد. این عمل همچنین مقدار count را در main تغییر می‌دهد.

ـ **مثال ۸ـ۸** مقدار متغیر با به کار بردن فراخوانی با مقدار افزایش می‌یابد.

```
#include <stdio.h>
main ()
   {
     int count = 7 ;
     int add(int) ;
     printf ("the original value of count is %d\n" , count) ;
     count = add (count) ;
     printf ("the new value of count is %d\n" , count) ;
   }
int add(int c)
   {
     return ++c ; \* incrementsl variable c */
   }
```

خروجی برنامه

```
the original value of count is 7
the new value  of count is 8
```

→

ـ **مثال ۸ـ۹** مقدار متغیر با فراخوانی آدرس افزایش می‌یابد.

```
# include <stdio.h>
main ()
   {
     int count = 7 ;
     int add (int *) ;
     printf("the original value of count is %d\n" , count) ;
     add (& count) ;
     printf ("the new value of count is %d/n" , count) ;
   }
void add (int *countptr)
   {
     + + (*countptr) ; /* increments count in main */
   }
```

خروجی برنامه

```
the original value of count is 7
the new value of count is 8
```

پارامتر متناظر تابعی که آدرسی را به عنوان آرگومان دریافت می‌کند، باید اشاره‌گر باشد. مثلاً عنوان تابع add در این مثال به صورت زیر است.

```
void add (int *countptr)
```

و بیان می‌کند که add آدرس متغیری از نوع int را دریافت و در countptr ذخیره خواهد کرد.

ـ **مثال ۸ـ۱۰** برنامهٔ زیر تفاوت بین آرگومانهای معمولی را که با مقدارعبور داده می‌شوند و آرگومانهای اشاره گر را که با مرجع عبور داده می‌شوند روشن می‌سازد.

```
#include<stdio.h>
main ()
  {
     int u = 1 ;
     int v = 3 ;
     void func1 (int u , int v) ;
     void func2 (int *pu , int *pv) ;
     printf (" Before calling func1:  u=%d v=%d " , u , v) ;
     func1(u , v) ;
     printf (" After calling func1:  u=%d v=%d " , u , v) ;
     printf (" Before calling func2:  u=%d v=%d " , u , v) ;
     func2(u , v) ;
     printf (" After calling func2:  u=%d v=%d " , u , v) ;
  }
```

```
void func1 (int u , int v)
  {
     u = 0 ;
     v = 0 ;
     printf (" Within func1:  u=%d v=%d " , u , v) ;
     return ;
  }
void func2 (int *pu , int *pv)
  {
     *pu = 0 ;
     *pv = 0 ;
     printf (" Within func2:  *pu=%d *pv=%d " , *pu , *pv) ;
     return ;
  }
```

خروجی برنامه به این شکل خواهد بود.

```
Before calling func1:  u = 1 v = 3
Within func1:  u = 0 v = 0
After calling func1:  u = 1 v = 3
Before calling func2:  u = 1 v = 3
Within func2:  *pu = 0 *pv = 0
After calling func2:  u = 0 v = 0
```

ملاحظه می‌کنید که تابع func1 دو متغیر از نوع صحیح را به عنوان آرگومان می‌پذیرد. تابع func2 دو اشاره‌گر به متغیرهای صحیح را به عنوان آرگومان دریافت می‌دارد.

انتقال دوطرفهٔ اطلاعات

وقتی که تابعی آدرس متغیری را در برنامهٔ فراخوانندهٔ آن بداند، می‌تواند هم مقادیری را در این متغیرها قرار دهد (یعنی مقادیر آنها را تغییر دهد) و هم مقادیر آن متغیرها را به کار برد. بنابراین به کمک اشاره‌گرها می‌توان مقادیر را هم از برنامهٔ فراخواننده به تابع فراخوانده شده و هم از تابع فراخوانده شده به برنامهٔ فراخوانندهٔ آن (درواقع در هر دو جهت) گذر داد. البته در مبحث توابع، انتقال مقادیر به روش فراخوانی با مقدار بررسی شد، ولی روش مذکور ما را قادر می‌سازد که بیش از این مقدار را به برنامه یا تابع فراخواننده برگردانیم و بدین طریق محدودیتی که در برگرداندن نتایج تابع فرعی به کمک نام آن تابع وجود دارد و در آن فقط می‌توان یک مقدار را با نام تابع برگرداند از بین برد.

ـ **مثال ۸ـ۱۱** برنامهٔ زیر ضرایب معادلهٔ درجه دومی را می‌خواند و سپس با فراخوانده شدن تابع فرعی‌ای به نام root ریشه‌های معادلهٔ مزبور را محاسبه می‌کند و به تابع اصلی برمی‌گرداند. اگر معادله ریشهٔ حقیقی نداشته باشد، تابع فرعی هر دو ریشه را صفر برمی‌گرداند. در ضمن اگر معادله ریشه داشته باشد، ضرایب معادله همراه با ریشه‌های آن در تابع اصلی چاپ می‌شود. در غیر این صورت، ضرایب آن همراه با پیغام مناسب چاپ می‌شود.

```
#include <stdio.h>
# include <math.h>
main ()
   {
     float a , b , c , x1 , x2 ;
     scanf ("%f %f %f" , &a , &b , &c) ;
     root (a , b , &x1 , &x2) ;
     if (x1 = = 0 && x2 = = 0)
       printf ("\n %f %f %f no real solution" , a , b , c) ;
     else
       printf ("\n %f %f %f %f %f" , a , b , cx1 , x2) ;
   }
void root (a , b , c , px1 , px2)
float a , b , c , *px1 , *px2 ;
  {
     float d , delta ;
     delta = b*b - 4*a*c ;
     if (delta < 0)
       { *px1 = *px2 = 0 ;
         return ;
       }
     else
       { d = sqrt (delta) ;
         *px1 = (-b+d) / (2*a) ;
         *px2 = (-b-d) / (2*a) ;
         return ;
       }
  }
```

در فراخوانی تابع root آدرس متغیرهای x2 و x1 (که باید ریشه‌های معادله را بپذیرد) به تابع مذکور گذر داده می‌شود و سپس در تابع root، اگر معادله دارای ریشه‌های حقیقی باشد، مقادیر آنها به متغیرهای x2 و x1 نسبت داده می‌شود (یعنی در محلهایی که آدرس آنها در متغیر اشاره‌گر px1 و px2 است قرار می‌گیرد). در غیر این صورت به هر دو متغیر مقدار صفر

نسبت داده می‌شود. بدین طریق بیش از یک مقدار از تابع فرعی به اصلی برگردانده می‌شود.

اشاره‌گرها و آرایه‌ها

بین اشاره‌گرها و آرایه‌ها رابطهٔ نزدیکی وجود دارد. نام آرایه در واقع اشاره‌گری به اولین عنصر آن است. قطعه برنامهٔ زیر را درنظر بگیرید.

```
char str[80] , *p ;
p = str ;
```

می‌دانیم که نام آرایه همان آدرس اولین عنصر آرایه است. بنابراین دو دستور

```
p = &str[0] ;
p = str ;
```

هم‌ارزند. پس در قطعه برنامهٔ بالا در اشاره‌گر p، آدرس آرایه (یعنی آدرس اولین عنصر آرایه) قرار داده شده است. حال اگر بخواهیم به پنجمین عنصر در str دسترسی داشته باشیم، می‌توانیم این کار را به دو روش زیر انجام دهیم.

```
str[4]
```

یا

```
*(p+4)
```

هر دو دستور، پنجمین عنصر را برمی‌گردانند.

همین طور اگر داشته باشیم

```
int a[15] , *p ;
p = &a[0] ;
```

دو عبارت a[3] و *(p+3) هم‌ارزند و هر دو چهارمین عنصر از ۱۵ عنصر را برمی‌گردانند؛ یعنی، به طور کلی عنصر a[k] هم‌ارز با *(p + k) خواهد بود و هر دو محتوای خانه k اُم آرایهٔ a یا (k+1) اُمین عنصر از مجموعه عناصر مزبور را برمی‌گردانند. همین طور عبارت مزبور هم‌ارز *(a + k) است، زیرا a نام آرایه، معرف آدرس آغاز آن است و (a + k) آدرس خانهٔ k اُم آرایه خواهد بود و در نتیجه عنصر *(a + k) محتوای خانهٔ k اُم از آرایهٔ مزبور را برمی‌گرداند. بنابراین C، سه روش برای دستیابی به عناصر آرایه در اختیار ما قرار می‌دهد. اما مهم است بدانیم که دستیابی به عناصر آرایه از طریق اشاره‌گر، سریع‌تر از روش استفاده از اندیس است. لذا روش *(p + k) و همین طور *(a + k) سریع‌تر از a[k] عمل می‌کند. بدین لحاظ استفاده از

اشاره‌گرها برای دستیابی به عناصر آرایه، روش بسیار متداولی در زبان C است.

حال برای تفسیر k در عبارتهای *(p+k) و *(p+k) به دو برنامهٔ زیر توجه کنید.

```
main ()
   {
     static int nums[ ] = {92 , 81 , 70 , 69 , 58} ;
     int k ;
     for (k = 0 ; k<5 ; k+ +)
     printf (" %d\n " , nums[k]) ;
   }
```

خروجی برنامه

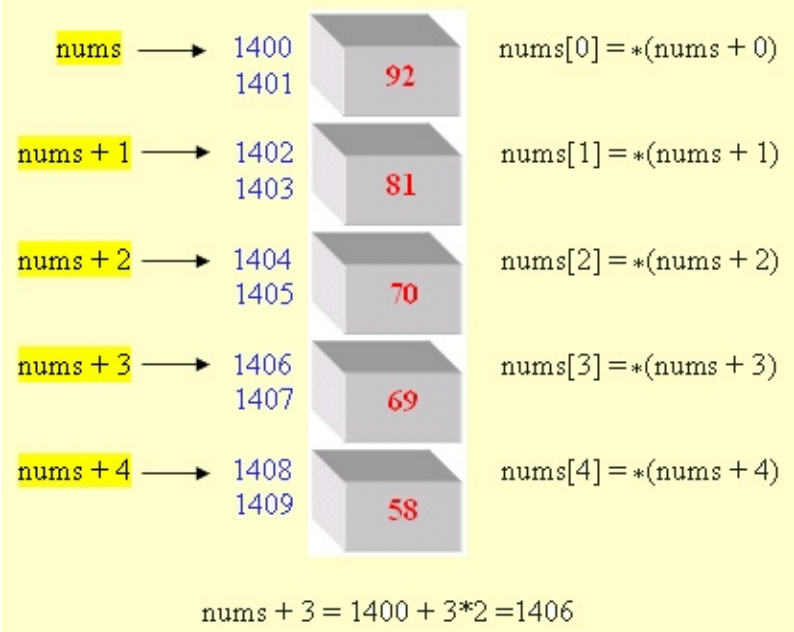
```
92
81
70
69
58
```

برنامهٔ مزبور برنامهٔ ساده‌ای است که در آن برای دستیابی به عناصر آرایه از روش متعارف علامتگذاری آرایه استفاده شده است.

حال همان برنامه با روش به کارگیری از اشاره‌گر به صورت زیر خواهد بود.

```
/* uses pointers to print out values from array */
main ()
   {
     static int nums[ ]= {92 , 81 , 70 , 69 , 58}
     int k ;
     for (k=0 ; k<5 ; k++)
     printf (" %d\n ", *(nums + k)) ;
   }
```

شکل ۸ـ۳ نشان می‌دهد که منظور از *(nums+k) محتوای k خانه، بعد از nums است که در آن بزرگی هر خانه مساوی بزرگی داده یا شئی مورد نظر در آرایه برحسب بایت است که در مثال بالا ۲ بایت است. همچنین در عبارت *(nums + k) نقش اپراتور ستاره را به‌عنوان عملگر غیرمستقیم ملاحظه می‌کنید که محتوای خانه یا آدرس nums+k را در اختیار قرار می‌دهد.



شکل ۸ـ۳ جمع اشاره‌گرها (آدرسها و مقادیر)

از موارد بالا نتیجه می‌شود که array[index] با *(array + index) یکسان است.

همچنین دو راه برای مراجعه به آدرس عنصری از آرایه وجود دارد. یکی به صورت nums+k در فرم اشاره‌گر و دیگری به صورت nums[k] در فرم آرایه.

حال اجازه دهید با برنامه‌ای ساده، رابطهٔ بین عناصر آرایه و آدرس آنها را بررسی کنیم.

ـ **مثال ۸ـ۱۲** برنامهٔ زیر را درنظر بگیرید.

```
# include <stdio.h>
main ()
  {
    static int x[6] = {10 , 11 , 12 , 13 , 14 , 15};
    int i ;
    for (i=0 ; i<6 ; + +i)
      printf("\n i =%d x[i] = %d *(x+i) = %d &x[i] = %x x+i =%x", i , x[i] ,
          *(x+i) , &x[i] , x+i) ;
  }
```

(فرض بر این است که آدرس شروع آرایه، ۷۲ در مبنای ۱۶ است.)

خروجی برنامه

i = 0	x[i] = 10	*(x+i) = 10	&x[i] = 72	x+i = 72
i = 1	x[i] = 11	*(x+i) = 11	&x[i] = 74	x+i = 74
i = 2	x[i] = 12	*(x+i) = 12	&x[i] = 76	x+i = 76
i = 3	x[i] = 13	*(x+i) = 13	&x[i] = 78	x+i = 78
i = 4	x[i] = 14	*(x+i) = 14	&x[i] = 7a	x+i = 7a
i = 5	x[i] = 15	*(x+i) = 15	&x[i] = 7c	x+i = 7c

از خروجی بالا فرق بین x[i] که معرف i اُمین عنصر آرایه است، با &x[i] که آدرس آن را نمایش می‌دهد، مشخص می‌گردد. در ضمن مشاهده می‌شود که مقدار i اُمین عنصر آرایه را می‌توان با x[i] یا *(x + i) معرفی کرد. در ضمن می‌توان تفاوت بین x + i و *(x + i) را ملاحظه کرد که اولی معرف آدرس و دومی نشان دهندهٔ محتوای آن آدرس است. همچنین نتیجه گرفته می‌شود که اگر x[i] در سمت چپ یک دستور جایگذاری باشد، می‌توان به جای آن *(x + i) را به کار برد. اصولاً همه جا می‌توانیم به جای x[i] هم‌ارز آن *(x+i) را به کار ببریم. به هرحال عباراتی مشابه x و x + i و &x[i] که معرف آدرس‌اند، نمی‌توانند در سمت چپ دستور جایگذاری به کار روند. همچنین آدرس آرایه نمی‌تواند به طور دلخواه تغییر یابد. بنابراین عبارتی مشابه x++ مجاز نیست.

قبلاً گفتیم که نام آرایه، به طور واقعی اشاره‌گری به اولین عنصر آرایه است. در نتیجه باید امکان داشته باشد که یک آرایه را، به جای روش قراردادی متداول، متغیر اشاره‌گر تعریف کرد.

به هرحال تعریف آرایه به روش قراردادی موجب می‌گردد که بلوک ثابت از حافظه، در آغاز اجرای برنامه رزرو شود. ولی اگر آرایه برحسب متغیر اشاره‌گر توصیف شود، با این عمل رزرو کردن جا اتفاق نمی‌افتد. درنتیجه موقع استفاده از اشاره‌گر برای معرفی آرایه، نیاز است که به طریقی قبل از اینکه عناصر آرایه مورد پردازش قرار گیرد، به عناصر آرایه، حافظه اختصاص داده شود. در حالت کلی اختصاص اولیهٔ حافظه در چنین مواردی، با استفاده از تابع کتابخانه‌ای malloc انجام می‌گیرد. گرچه شیوهٔ انجام این کار از کاربردی به کاربرد دیگر فرق خواهد کرد.

اگر آرایه به صورت متغیر اشاره‌گر تعریف گردد، نمی‌توان به عناصر آرایه‌های عددی

مقدار اولیه نسبت داد. در نتیجه این گونه موارد، به تعریف آرایه به صورت روش عادی و قراردادی نیاز دارد.

ـ **مثال ۸ـ۱۳** اگر بخواهیم a را به صورت آرایهٔ ۱۰ عنصری از مقادیر صحیح تعریف کنیم، می‌توان آن را به جای ; int a[10] به صورت ; int *a نوشت؛ یعنی a را متغیر اشاره‌گر تعریف کرد.

به هرحال در روش دوم، به a که آرایه‌ای ۱۰ عنصری است یک بلوک حافظه اختصاص داده نمی‌شود، بلکه a به صورت متغیر اشاره‌گر تعریف شده است.

برای اختصاص حافظهٔ مورد نیاز به a جهت معرفی آرایه‌ای ۱۰ عنصری، می‌توان از تابع malloc به صورت زیر استفاده کرد.

```
a = malloc (10 * sizeof(int)) ;
```

این تابع بلوک حافظه‌ای برای ذخیره کردن ۱۰ مقادیر صحیح رزرو می‌کند. در دستور بالا اپراتور sizeof بزرگی نوع داده int را برحسب بایت برمی‌گرداند. این مقدار در ۱۰ (تعداد عناصر آرایه) ضرب می‌شود تا فضای مورد نیاز برحسب بایت تعیین و رزرو شود. a به صورت اشاره‌گر به مقدار صحیح تعریف شده است و تابع malloc یک اشاره‌گر به کاراکتر برمی‌گرداند و می‌دانیم که در زبان C مقادیر صحیح و کاراکترها معادل‌اند. لذا دستور بالا قابل قبول است. اگرچه از لحاظ اطمینان کامل می‌توان از تبدیل نوع cast استفاده کرد و آن را به صورت زیر به کار برد.

```
a = (int *) malloc (10 * sizeof (int)) ;
```

این گونه اختصاص حافظه به روش اختصاص حافظه به صورت پویا موسوم است.

به هرحال اگر قرار باشد به عناصر آرایه، مقدار اولیه نیز اختصاص یابد، باید a به جای متغیر اشاره‌گر به صورت آرایه توصیف گردد، مشابه زیر.

```
int a[10] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10} ;
```

یا

```
int a[ ] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10}
```

در برنامه‌نویسی با C ممکن است برای مراجعه به عناصر آرایه، به جای روش معمول، عباراتی برحسب اشاره‌گرها به کار ببریم. این روش در آغاز کار کمی غیرطبیعی به نظر می‌آید،

ولی می‌توان با کمی تمرین به سادگی تجربهٔ لازم را در این مورد به دست آورد. در مثال ۸ـ۱٤ ضمن اینکه روش استفاده از تابع malloc نشان داده می‌شود، عناصر آرایه نیز با استفاده از این روش در دسترس قرار می‌گیرند.

ـ **مثال ۸ـ۱٤** برنامهٔ زیر مجموعه‌ای از اعداد را از ورودی می‌خواند و با استفاده از اشاره‌گرها مرتب می‌کند.

```
# include <stdio.h>
main ()
  {
    int k , m , *a ;
    void sort (int k , int *a) ;
    scanf ("%d" , &m) ;            /* read in a value for m */
    a = (int*) malloc (m * sizeof(int)) ;   /* allocate memory */
    for (k = 0 ; k<m ; + +k)       /* read in the list of numbers */
      scanf ("%d" , a + k) ;
    sort (m , a) ;
    for (k=0 ; k<m ; + +k)         /* display sorted list , elements */
      printf ("\n k=%d a =%d" , k+1 , *(a+k)) ;
  }
void sort (int m , int *a)         /* sort array in ascending order */
  {
    int i , j , temp ;
    for (i=1 ; i<m ; + +i)
      for (j=0 ; j<m-i ; + +j)
        if (*(a+j) < *(a+j+1))
          {
            temp = *(a+j) ;
            *(a+j) = *(a+j+1) ;
            *(a+j+1) = temp;
          }
    return ;
  }
```

در این برنامه، آرایه‌ای با عناصری از نوع مقادیر صحیح، به صورت اشاره‌گر به مقداری صحیح تعریف شده است. در آغاز به کمک تابع کتابخانه‌ای malloc به متغیر اشاره‌گر، حافظه اختصاص داده شده است (یعنی حافظهٔ مورد نیاز از سیستم گرفته شده و آدرس اولین بایت آن در a قرار داده شده است). در هر دو تابع اصلی و فرعی برای پردازش هر عنصر از روش مراجعه به اشاره‌گر استفاده شده است. ملاحظه می‌کنیم که در تابع scanf نیز برای آدرس

عنصر k اُم به جای &a[k] از a + k استفاده شده است. به طریق مشابه، در تابع printf برای معرفی مقدار k اُمین عنصر، به جای a[k] از *(a + k) استفاده شده است. ملاحظه می‌کنید که در تابع فرعی نیز آرگومان آن، به جای آرایه، متغیر اشاره تعریف شده است.

اشاره‌گرها و آرایه‌های چند بعدی

دیدیم که عناصر آرایه‌ای یک بعدی می‌تواند برحسب اشاره‌گر (نام آرایه) و مقدار به عنوان آفست به منظور جبران کردن مقدار اندیس عنصر مورد نظر آرایه نمایش داده شود. مثلاً اگر نام آرایه a و عنصر مورد نظر ما a[5] باشد، می‌توان به آن به صورت *(a + 5) مراجعه کرد که در آن مقدار آفست همان 5 است که به نام آرایه افزوده شده است و به کمک عملگر * به مقدار آن دسترسی پیدا می‌کنیم.

حال می‌توان گفت که آرایه‌ای دوبعدی نیز مجموعه‌ای از آرایه‌های یک‌بعدی است. بنابراین می‌توان آرایه‌ای دو بعدی را به صورت اشاره‌گر به گروه پیوسته و مجاور هم از آرایه‌های یک بعدی تعریف کرد. درنتیجه می‌توان توصیف آرایه‌ای دو بعدی را به جای data-type array[d1][d2] ; (که در آن منظور از d1 و d2 به‌ترتیب بزرگی ابعاد اول و دوم آرایه است) به صورت زیر نوشت.

```
data-type (*ptvar)[d2] ;
```

این ایده را می‌توان به آرایه‌های n بعدی تعمیم داد و آن را به جای

```
data-type array [d1][d2]…[dn] ;
```

به صورت زیر نوشت.

```
data-type (*ptvar)[d2][d3]…[dn] ;
```

که در آنها data-type نوع عناصر آرایه و array نیز نام آرایه است. عناصر d1 , d2 ,..., dn نیز به ترتیب ماکزیمم عناصر هر اندیس یا هر بعد آرایه‌اند. درضمن توجه کنید که ptvar نیز نام متغیر اشاره‌گر است.

انتقال آرایه به تابع

در زبان C، نام هر آرایه‌ای که به عنوان آرگومان تابع به کار رود، آدرس اولین عنصر آرایه

تفسیر می‌گردد. برنامهٔ زیر را درنظر بگیرید.

```
main ()
  {
    float func() ;
    float x , array[15] ;
    .......
    .......
    x = func(array) ; /* same as func (&array [0]) */
    .......
    .......
  }
```

حال در تابع فرعی نیاز است که آرگومان را به عنوان اشاره‌گر به اولین عنصر آرایه توصیف کنیم. برای این کار، دو راه به صورت زیر وجود دارد.

راه اول

```
func(ar)
float ar[ ] ;
   {
     .......
     .......
   }
```

راه دوم

```
func(ar)
float *ar ;
   {
     .......
     .......
   }
```

راه اول، ar را آرایه‌ای با اندازهٔ (یا بزرگی) نامشخص توصیف می‌کند. آرایه هم‌اکنون در تابع اصلی ایجاد شده است. آنچه گذر داده می‌شود، اشاره‌گری به اولین عنصر از آرایه است، چون کامپایلر می‌داند که عبارت آرایهٔ حاصل، به اشاره‌گر به اولین عنصر آرایه برمی‌گردد، پس ar را مشابه توصیف ar در روش دوم، به اشاره‌گر از نوع float تبدیل می‌کند. بنابراین هر دو گونه ازنظر نحوهٔ عملکرد، معادل و هم‌ارز یکدیگرند.

به هرحال از نظر وضوح، ممکن است روش اول ترجیح داده شود، زیرا این روش تأکید می‌کند که آنچه باید گذر داده شود آدرس پایه یا آدرس اولین عنصر آرایه است. در روش دوم، راهی وجود ندارد تا بتوان تشخیص داد آیا ar به آغاز آرایه‌ای از نوع float و یا تنها به یک عنصر از نوع float اشاره می‌کند یا نه.

آرایه‌هایی از اشاره‌گرها

در زبان C می‌توان آرایه‌ای از اشاره‌گرها تعریف کرد؛ یعنی آرایه‌ای که عناصر آن اشاره‌گر باشند. دستور زیر آرایه‌ای ۱۰ عنصری از اشاره‌گرها را توصیف می‌کند.

```
int *x[10] ;
```

اینها اشاره‌گرهایی‌اند که می‌توانند آدرس متغیرهایی از نوع مقادیر صحیح را در خود داشته باشند. به عنوان مثال برای اختصاص دادن آدرس متغیری به نام z به عنصر سوم آرایهٔ مزبور می‌نویسیم

```
*x[2] = &z ;
```

همین طور برای به دست آوردن مقدار z از دستور [x]** استفاده می کنیم.

آرایه‌ای از اشاره‌گرها را نیز می‌توان مشابه آرایه‌های معمولی به یک تابع انتقال داد؛ یعنی به سادگی، نام آرایه را بدون اندیس یا زیرنویس آن به عنوان آرگومان تابع قرار می‌دهیم.

ـ **مثال ۸ـ۱۵** تابع FF1 می‌تواند آرایهٔ x را به صورت زیر دریافت کند.

```
void FF1 (int *a[ ])
  {
    int k ;
    for (k=0 ; k<10 ; k+ +)
      printf (" %p" , *a[k]) ;
  }
```

توجه داشته باشید که در این مثال a اشاره‌گری به مقادیر صحیح نیست بلکه اشاره‌گری به آرایه‌ای از اشاره‌گرهایی به مقادیر صحیح است. بنابراین نیاز است که پارامتر a آرایه‌ای از اشاره‌گرهایی به مقادیر صحیح، به همان طریق که نشان دادیم، توصیف شود.

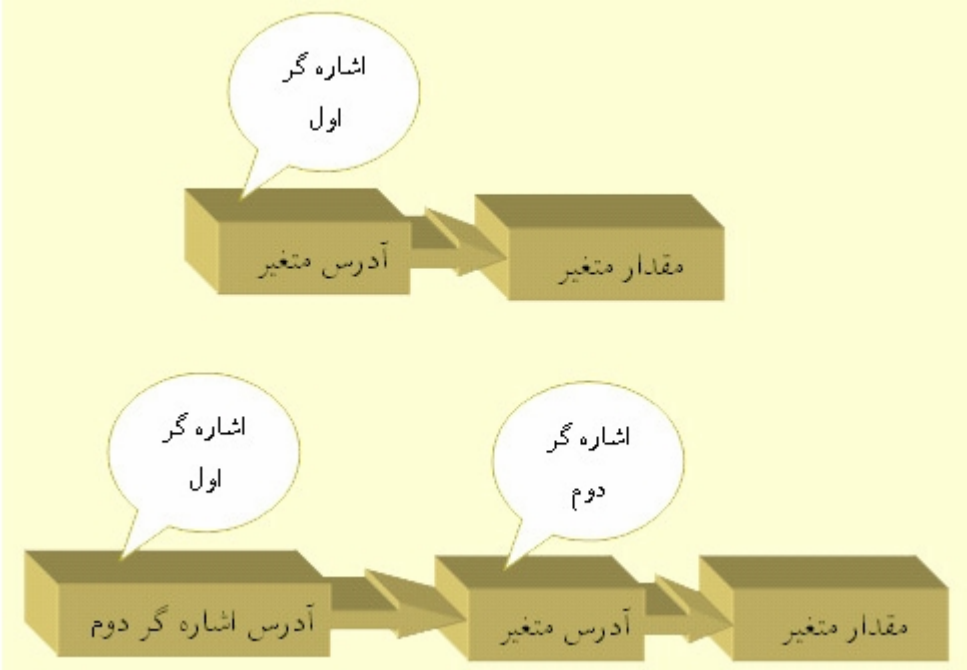
آرایه‌های اشاره‌گر اغلب برای نگهداری اشاره‌گرهایی به رشته‌ها به کار می‌روند.

ـ

اشاره‌گر به اشاره‌گر

گفتیم اگر متغیری آدرس متغیر دیگری را در خود نگه دارد آن را اشاره‌گر نامند. حال اگر متغیر دوم نیز از نوع اشاره‌گر باشد، متغیر اول اشاره‌گر به اشاره‌گر است. چنین موقعیتی را اشاره‌گر به اشاره‌گر نامند. ممکن است تصور و فهم اشاره‌گر به اشاره‌گر ایجاد اشکال کند. شکل ۸ـ٤

مفهوم اشاره‌گر به متغیر عادی و اشاره‌گر به اشاره‌گر را روشن می‌کند.



شکل ۸ـ٤ نمایش اشاره‌گر به متغیر عادی و اشاره‌گر به اشاره‌گر

همان طور که ملاحظه می‌کنید، مقدار اشاره‌گر معمولی، آدرس متغیر است اما در مورد اشاره‌گر به اشاره‌گر، اولین اشاره‌گر آدرس اشاره‌گر دوم را دارد که آن هم به نوبهٔ خود آدرس متغیر دیگری را در خود دارد. این روش می‌تواند (به‌صورت تودرتو) تا هرچند بار که نیاز باشد، تکرار گردد. اما در عمل کمتر به بیش از یک بار نیاز پیش می‌آید و داشتن تصور صحیح از آن نیز برای اغلب برنامه‌نویسان مشکل است.

برای توصیف متغیرهایی از نوع اشاره‌گر به اشاره‌گر باید دو ستاره در جلوی آن قرار داد. برای مثال توصیف زیر به کامپایلر می‌گوید که متغیر z اشاره‌گر به اشاره‌گری از نوع float است.

```
float **z ;
```

به هرحال باید توجه داشته باشید که z اشاره‌گری به یک مقدار اعشار نیست، بلکه اشاره‌گری به اشاره‌گر است که اشاره‌گر دوم می‌تواند آدرس متغیری از نوع float را داشته

باشد.

برای دستیابی به مقدار متغیر هدف، که آدرس آن در اشاره‌گر دوم است، باید اپراتور ستاره را دوباره به کار ببرید، مانند مثال زیر.

```
# include <stdio.h>
main ()
  {
    int x , *p , **q ;
    x = 10 ;
    p = &x ;
    q = &p ;
    printf ("%d" , **q) ; / * print the value of x */
  }
```

در این مثال، p اشاره‌گر به متغیر int و q نیز اشاره‌گری به اشاره‌گر توصیف شده است که ممکن است آدرس متغیری از نوع int را داشته باشد. حال نتیجهٔ اجرای printf این خواهد شد که ۱۰ (مقدار متغیر x) روی صفحه نمایش نشان داده شود.

یادآوری. آرایه‌ای از اشاره‌گرها نوعی از اشاره‌گر به اشاره‌گر است.

ارسال تابعی به تابع دیگر

در زبان C هنگامی که تابعی درون تابع دیگری اعلان می‌شود آن تابع را تبدیل اشاره‌گری به خودش نامند. چنین اشاره‌گرهایی را می‌توان به عنوان آرگومان به توابع دیگر فرستاد که در واقع باعث می‌شود تابع متناظر با آنها به تابع دیگری ارسال شود و درون آن بتوان بدان دست یافت. از آنجایی که آرگومان حقیقی مورد استفاده همواره متغیر است، بنابراین در فراخوانیهای مختلف تابع دوم می‌توان اشاره‌گرهای گوناگونی (توابع متفاوتی) به آن فرستاد. هر گاه تابعی از طریق آرگومان محلی خود تابعی دیگر را بپذیرد، باید اعلان آن به گونه‌ای باشد که مشخص سازد آرگومان محلی مورد نظر اشاره‌گری است به تابع. در ساده‌ترین شکل این آرگومان را می‌توان به صورت زیر اعلان کرد.

```
data_type (*function_name) () ;
```

که در آن data_type نوع داده کمیت بازگشتی تابع ارسالی است. به دنبال این اعلان می‌توان به تابع مزبور با عملگر غیرمستقیم دست یافت.

نتیجه‌گیری

باتوجه به مطالب این فصل می‌توان نتایج زیر را در مورد اشاره‌گرها بیان کرد.

۱. دو عملگر یا اپراتوری که در اشاره‌گرها استفاده می‌شوند عبارت‌اند از * و &. عملگر & عملگری یکانی است که آدرس عملوند یا اپراند خود را مشخص می‌کند. عملگر * نیز عملگری یکانی است که محتویات آدرس حافظه را مشخص می‌کند. بنابراین عملگر * مکمل عملگر & است.

۲. اعمال متداول روی اشاره‌گرها عبارت‌اند از:

الف) عمل انتساب یا جایگذاری

ب) اعمال محاسباتی شامل جمع، تفریق، + + و - -

ج) مقایسهٔ اشاره‌گرها.

۳. اشاره‌گرها دارای ویژگیهای زیرند.

الف) عمل تخصیص حافظه به‌صورت پویا را امکان‌پذیر می‌کنند.

ب) کار با آرایه‌ها و رشته‌ها را آسان می‌کنند.

ج) موجب بهبود کارآیی بسیاری از توابع می‌گردند.

د) فراخوانی با آدرس را در مورد توابع امکان‌پذیر می‌سازند. درنتیجه برگردان بیش از یک مقدار، از یک تابع، میسر می‌گردد.

نکتهٔ ۱. کلمهٔ کلیدی far برای تعریف اشاره‌گرهایی به کار می‌رود که به ذخیرهٔ آدرسهایی بیش از دو بایت نیاز دارند و قبل از نام متغیر اشاره‌گر و بعد از تعیین نوع اصلی ذکر می‌شود، مانند اعلان زیر.

```
char far *ptr ;
```

نکتهٔ ۲. کلمهٔ کلیدی huge برای تعریف اشاره‌گرهایی به کار می‌رود که آدرسهایی بیش از شانزده بیت را در خود ذخیره می‌کنند.

خودآزمایی ۸

۱. تابعی بنویسید که با استفاده از اشاره‌گر، طول رشته‌ای را به دست آورد.

۲. برنامه‌ای بنویسید که یک خط متن از ورودی دریافت کند و تعداد حروف صدادار، حروف بی صدا، فاصله‌ها و تعداد ارقام را در آن مشخص کند.

۳. یکی از کاربردهای مهم اشاره‌گرها در مورد آرایه‌های کاراکتری است. اغلب عملیات روی رشته‌ها معمولاً با استفاده از عملیات محاسباتی روی اشاره‌گرها انجام می‌گیرد. تابع زیر دو رشته را از لحاظ یکسان بودن با یکدیگر مقایسه می‌کند. اگر یکسان نبودند true و در غیر این صورت false برمی‌گرداند. در واقع نقش تابع کتابخانه‌ای strcmp را انجام می‌دهد.

٤. تابعی بنویسید که با استفاده از اشاره‌گر، دو رشته را به هم متصل کند و رشتهٔ سومی تشکیل دهد.

۵. برنامه‌ای بنویسید که با استفاده از اشاره‌گر و روش اختصاص حافظه به صورت پویا، حافظه‌ای برای آرایهٔ n عنصری اختصاص دهد و عناصر آرایه را به حافظه بخواند. سپس با فراخوانده شدن یک تابع، عناصر آرایهٔ مزبور با به کارگیری اشاره‌گر، به صورت صعودی مرتب و نتیجه در تابع اصلی چاپ شود.

٦. برنامه‌ای بنویسید که عناصر دو ماتریس (آرایهٔ دوبعدی) را به حافظه بخواند و مجموع آن دو را براساس قانون جمع ماتریسها در آرایهٔ C قرار دهد و نتیجه را به صورت ماتریس چاپ کند.

۷. یک راه برای مرتب کردن رشته‌ها با استفاده از اشاره‌گرها آن است که آدرس رشته‌ها را در آرایه‌ای از اشاره‌گرها قرار دهیم. سپس در مقایسهٔ دو رشته، اگر نیاز به جابه‌جایی آن دو با یکدیگر باشد، آدرس دو رشته را در درون آرایهٔ اشاره‌گرها که آدرس رشته‌ها را دارد با یکدیگر عوض کنیم.

۸. تابعی بنویسید که مقادیر دو متغیر را با استفاده از اشاره گر تعویض کند.

۹. برنامه‌ای بنویسید که رشته و کاراکتری را از ورودی بخواند. سپس با فراخواندن تابع فرعی، تعداد دفعاتی را که کاراکتر مورد نظر در رشتهٔ مزبور وجود دارد بشمارد و چاپ کند.

۱۰. نحوهٔ اعلان یا توصیف متغیرهای زیر در اشاره گرها را شرح دهید.

```
1. int *p ;
2. int *p[10] ;
3. int (*p)[10] ;
4. int *p(void) ;
5. int p(char *a) ;
6. int *p(char *a) ;
7. int (*p) (char *a) ;
11. int *p(char a[ ]) ;
12. int *p(char (*a)[ ]) ;
13. int *p(char *a[ ]) ;
14. int (*p)(char (*a)[ ]) ;
15. int *(*p)(char (*a)[ ]) ;
16. int *(*p)(char *a[ ]) ;
17. int (*p[10])(void) ;
```

8. int (*p(char *a))[10] ;
9. int p(char(*a)[ ]) ;
10. int p(char *a[ ]) ;

18. int (*p[10])(char a) ;
19. int *(*p[10])(char a) ;
20. int *(*p[10])(char *a) ;

فصل ۹

نوعهای تعریف شده

هدف کلی

آشنایی با ساختار، typedef، اجتماع، و شمارش به منظور استفاده از مجموعه‌ای از داده‌هایی که ویژگیهای یکسان ندارند.

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از خواندن این فصل،

۱. با ساختار در نوع داده‌ای با صفات مختلف و تفاوت آن با رکورد آشنا شود.

۲. با متداول‌ترین کاربرد ساختار، یعنی آرایه‌ای از ساختارها آشنایی بیابد.

۳. نحوهٔ پردازش در ساختار را بشناسد.

۴. نحوهٔ انتقال ساختار به تابع را بشناسد.

۵. شکل کلی و کاربرد دستور typedef و مزیت آن را بر ساختار درک کند.

۶. ساختار داده‌ها و اشاره‌گرها را بشناسد.

۷. متغیر اشاره‌گر در عضو ساختار را بشناسد.

۸. با اجتماع در نوع داده‌ای با صفات مختلف آشنا شود.

۹. با شمارشی در نوع داده‌ای با صفات مختلف آشنا شود.

مقدمه

اغلب در کاربردهای مختلف برنامه نویسی، با مجموعه‌ای از داده‌ها که ویژگیهای یکسان

ندارند مواجهیم و در این حالت، آرایه‌ها قابل استفاده نیستند. در زبان C این مشکل رفع شده است و اجازه می‌دهد که کاربر چند نوع داده با توجه به نیاز خود ایجاد کند که عبارت‌اند از:

الف) ساختار. عبارت است از دسته‌بندی متغیرهایی با صفات مختلف تحت یک نام.

ب) typedef. نام جدیدی برای نوع (type) موجود ایجاد می‌کند.

ج) اجتماع. با آن می‌توان قسمتی از حافظه را برای استقرار دو یا چندین نوع متفاوت از داده‌ها تعریف کرد.

د) شمارشی. فهرستی از نشانه‌ها یا سمبولهاست که می‌توان با مقادیر صحیح شمارشی ۱، ۲، ۳ و... به آنها مراجعه کرد.

ساختار[1]

ساختار مجموعه‌ای از متغیرهاست که با یک نام به آنها مراجعه می‌شود. در واقع ساختار داده جدیدی است که هر عنصر آن از نوع داده متفاوت است. این شیوه وسیلهٔ ساده‌ای برای یکپارچه ساختن مجموعه‌ای از داده‌ها یا اطلاعات مربوط به هم در اختیار برنامه نویس قرار می‌دهد. برای مثال فرض کنید که از هر دانشجویی شمارهٔ دانشجویی، نام و نمرهٔ امتحانی وی با فرمت زیر مورد نیاز باشد.

st-no	name	grade

در اینجا نمی‌توان این ۳ فیلد را در آرایه‌ای ۳ عنصری قرار داد، زیرا فیلد اول از نوع int، فیلد دوم از نوع رشته (آرایهٔ کاراکتری) و فیلد سوم از نوع float است؛ یعنی صفات آنها یکسان نیستند. حال خواسته آن است که بتوان به این مجموعهٔ ۳ عنصری، تحت یک نام مراجعه کرد و اگر نیاز باشد بتوان به عناصر یا فیلدهای آن نیز مراجعه کرد. این گونه مجموعه را در زبان پاسکال و کوبول رکورد و در زبان C ساختار تعریف می‌کنند. اگر این مجموع را rec بنامیم، نحوهٔ تعریف آن در زبان C به صورت زیر خواهد بود.

```
struct rec
  {
    int st-no ;
```

1. structure

```
    char name[15] ;
    float grade ;
  } ;
```

در اینجا کلمهٔ کلیدی struct برای تعریف داده از نوع ساختار به کار رفته که پس از آن نام انتخابی برای مجموعهٔ مورد نظر به کار برده شده و سپس در داخل یک زوج آکولاد، عناصر یا اعضای مجموعهٔ مورد نظر تعریف شده‌اند و در پایان نیز سمیکولون ';' به عنوان پایان تعریف ساختار به کار رفته است. بنابراین در این مثال کلمهٔ rec این ساختار ویژه از داده‌ها را مشخص می‌سازد.

حال می‌توان متغیرهایی مانند x و z را از نوع ساختار مزبور تعریف کرد. برای این کار کافی است بنویسیم

```
struct rec x , z ;
```

در واقع با این تعریف، به کامپایلر می‌گوییم که متغیرهای x و z از نوع struct است که با نام rec تعریف شده است.

وقتی که ساختاری را تعریف می‌کنیم، در واقع نوع پیچیده‌ای مرکب از عناصر ساختار را تعریف می‌کنیم. بنابراین در مثال بالا کلمهٔ rec متغیر نیست بلکه معرف نوع داده است، ولی کلمات x و z متغیرهایی‌اند که نوع آنها از نوع rec است.

می‌توان x و z را به یکی از دو صورت زیر نیز تعریف کرد.

روش اول

```
struct rec
  {
   int st-no ;
   char name[15] ;
   float grade ;
  } x , z ;
```

روش دوم

```
struct
  {
    int st-no ;
    char name[15] ;
    float grade ;
  } x , z ;
```

در هر دو روش بالا، اسامی متغیرهای مورد نظر بلافاصله پس از تعریف ساختار و قبل از بستن آن با علامت ';' که پایان تعریف struct است آمده است. لذا دیگر نیازی به معرفی آنها با دستور struct rec نیست. در تعریف اول، نامی برای نوع داده انتخاب شده است. پس بعد از این نیز هر جای دیگر از برنامه می‌توان متغیرهای دیگری را از نوع struct rec تعریف کرد. اما در تعریف دوم نمی‌توان متغیری از این نوع تعریف کرد. به هر حال شیوهٔ بهتر آن

است که برای تعریف متغیر دیگری از نوع ساختار مورد نظر در جای دیگری از برنامه با محدودیت مواجه نشویم.

حال باتوجه به توضیحات بالا، چنانچه در حالت کلی نام ساختار ۱ را با tag و اسامی اعضای آن را با: member*m* , ... , member2 , member1 نشان دهیم، فرم کلی ترکیب ساختار به صورت زیر خواهد بود.

```
struct tag                          struct structure-name
  {                                   {
     member1 ;                           type variable1-name ;
     member2 ;                           type variable2-name ;
     ....                                ....
     ....                                ....
     member m ;                          type variablem-name ;
  } ;                                 } ;
```

که در آن struct کلمهٔ کلیدی برای تعریف داده از نوع ساختار و tag نام این ساختار داده‌هاست. همچنین، member*m* , … , member2 , member1 نیز معرف توصیف اعضای ساختار مزبورند.

هر یک از اعضای ساختار می‌تواند متغیر معمولی، اشاره‌گر، آرایه، یا حتی ساختار دیگری باشد. با اینکه ممکن است نام اعضای یک ساختار با متغیرهای دیگری که در جایی دیگر از برنامه تعریف شده‌اند همنام باشد، گویاتر آن است که در این گونه موارد اسامی همنام به کار نبریم.

به اعضای یک ساختار نمی‌توان کلاس حافظه اختصاص داد و همچنین نمی‌توان هنگام تعریف ساختار به اعضای آن مقدار اولیه نسبت داد. وقتی ترکیب ساختار یک بار تعریف گردید، می‌توان متغیرهایی از نوع ساختار مزبور به صورت زیر توصیف کرد.

storage-class struct tag variable1 , variable 2 , … , variable *m* ;

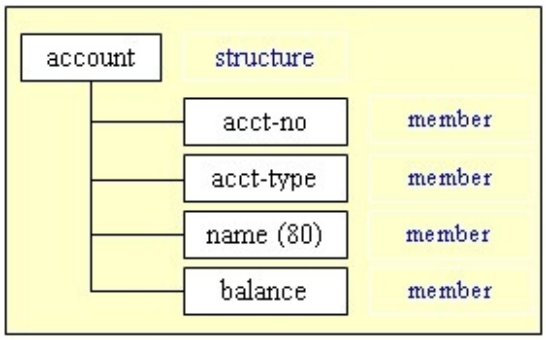
که در آن storage-class مشخص‌کنندهٔ کلاس حافظه است و به کار بردن آن اختیاری است، ولی کلمهٔ کلیدی struct الزامی است و tag نیز نام ساختار است و متغیرهای

variable1 , variable 2 , … variable *m*

نیز متغیرهایی از نوع ساختار tag اند.

ـ **مثال ۹ـ۱** نمونه‌ای از تعریف ساختار، همراه با نمای ظاهری آن نشان داده شده است.

```
struct account {
   int acct-no ;
   char acct-type ;
   char name[80] ;
   float balance ;
}
```



نام ساختار، در این مثال، account است و ۴ فیلد یا عنصر دارد که عبارت‌اند از:

- مقدار صحیح acct-no برای شماره حساب؛
- تک‌کاراکتر acct-type برای نوع حساب؛
- آرایه‌ای کاراکتری یا رشته‌ای name[80] برای نام صاحب حساب؛ و
- کمیت یا مقدار balance برای مبلغ موجودی صاحب حساب.

حال می‌توان متغیرهای oldcustomer و newcustomer را از نوع ساختار مزبور به صورت زیر تعریف کرد.

```
struct account oldcustomer , newcustomer ;
```

بنابراین، oldcustomer و newcustomer متغیرهایی از نوع ساختارند که ترکیب عناصر آنها با ساختار account مشخص شده است.

به طوری که گفتیم می‌توان توصیف ساختار را با متغیرهایی از نوع ساختار مزبور ترکیب و آنها را یکجا به صورت زیر تعریف کرد.

```
storage-class struct tag {
                          member1 ;
                          member2 ;
                          ...
                          memberm ;
                        } variable1 , variable2 ,... , variablem ;
```

در چنین موقعیتی، انتخاب نام برای ساختار، یعنی به کار بردن tag، اختیاری است.

→

ـ **مثال ۹ـ۲** توصیف زیر، معادل دو توصیف مثال ۹ـ۱ است.

```
struct account {
                int acct-no ;
                char acct-type ;
                char name[80] ;
                float balance ;
               } oldcustomer , newcustomer ;
```

حال چون تعریف متغیرها با تعریف نوع ساختار ترکیب شده است، می‌توان نام ساختار، یعنی account، را نیز به کار نبرد. البته این شیوه توصیه نمی‌گردد.

در یک ساختار ممکن است عضوی از ساختار دیگری تعریف گردد. در چنین حالتی باید تعریف ساختار درونی (یعنی ساختار به کار رفته در درون ساختار دیگر) قبل از تعریف ساختار بیرونی ظاهر گردد.

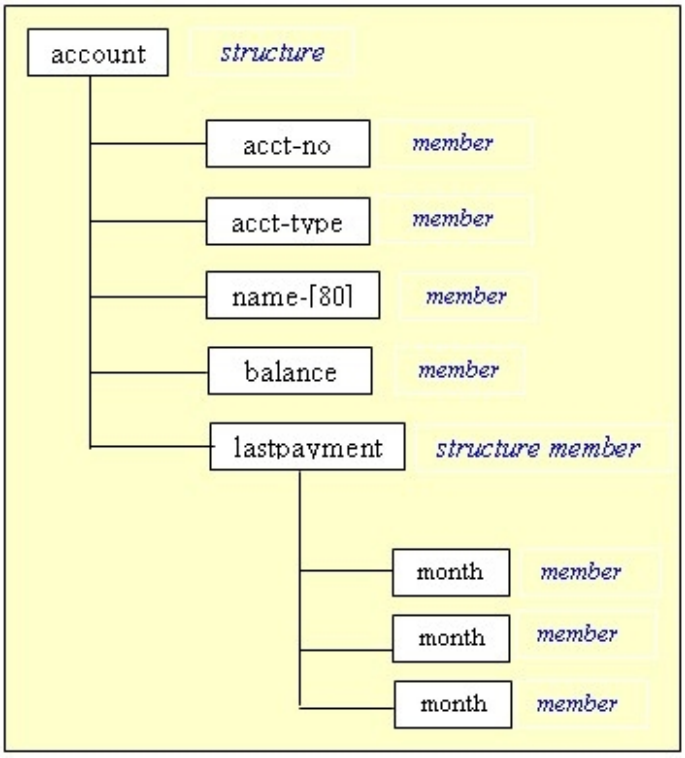
ـ **مثال ۹ـ۳** در برنامهٔ C تعریفهای زیرمفروض است که به همراه نمای ظاهری آن نشان داده شده است.

```
struct date {
        int month ;
        int day ;
        int year ;
} ;
struct account {
        int acct-no ;
        char acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} oldcustomer , newcustomer;
```

ملاحظه می‌کنید که ساختار account شامل ساختار دیگری، یعنی date، است که یکی از اعضا یا فیلدهای خود است. در چنین موقعیتی باید تعریف date قبل از تعریف account بیاید.

اختصاص مقادیر اولیه

به اعضا یا فیلدهای متغیر در ساختار می‌توان، مشابه روشی که در مورد عناصر آرایه ملاحظه



شکل ۹ـ۱ نمای ظاهری مثال ۹ـ۳

کردید، مقادیر اولیه نسبت داد. مقادیر اولیهٔ مورد نظر، باید مشابه همان ترتیب تناظر اختصاص آنها به عناصر آرایه ظاهر گردند و در داخل زوج آکولاد قرار گیرند و با کاما از یکدیگر مجزا گردند. شکل کلی آن به صورت زیر است.

storage-class struct tag variable = {value1 , value2 , … , valuem} ;

که در آن value1 , value2 ,...valuem معرف مقادیری است که باید به ترتیب به عناصر اول، دوم،... و m اُم متغیر ساختار اختصاص یابد.

ـ **مثال ۹ـ٤** قطعه برنامهٔ زیر نحوهٔ اختصاص مقادیر اولیه به اعضای متغیر ساختار را نشان می‌دهد.

```
struct date {
        int month ;
        int day ;
        int year ;
    } ;
```

```
struct account {
                int acct-no ;
                char acct-type ;
                char name[80] ;
                float balance ;
                struct date lastpayment ;
                } ;
static struct account customer = {12746 , `R´ , "payam noor" , 2986.50 , 5 , 24 , 75} ;
```

بنابراین، customer متغیر ساختاری ایستا از نوع account است که به عناصر آن مقادیر اولیه اختصاص داده شده است. به اولین عنصر عدد صحیح 12746، به دومین عنصر کاراکتر `R´، به سومین عنصر رشتهٔ " payam noor " و به چهارمین عنصر یا balance مقدار اعشاری 2986.50 اختصاص داده شده است. آخرین عنصر آن ساختاری است که شامل سه مقدار صحیح (از چپ به راست، معرف ماه، روز و سال) است. بنابراین به آخرین عضو customer مقادیر صحیح 75 , 24 , 5 اختصاص داده شده است.

آرایه‌ای از ساختارها

در زبان C می‌توان آرایه‌ای از ساختارها تعریف کرد؛ یعنی آرایه‌ای تعریف کرد که هر عنصر آن یک ساختار باشد. این شیوه، متداول‌ترین کاربرد ساختارهاست. برای توصیف آرایه‌ای از ساختارها، باید اول ساختار مورد نظر را توصیف کرد. سپس متغیری از نوع آرایه توصیف کرد که عناصر آن از نوع ساختار مزبور باشد.

ـ **مثال ۹ـ۵** آرایه‌ای ۱۰۰ عنصری به نام customer تعریف کنید که عناصر آن ساختاری از نوع مثال ۹ـ۴ یعنی از نوع account باشد.

روش اول

```
struct date {
        int month
        int day ;
        int year ;
} ;
struct account {
```

روش دوم

```
struct date {
        int month ;
        int day ;
        int year ;
} ;
struct account {
```

```
        int acct-no ;
        int acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} customer[100] ;
```

```
        int acct-no ;
        int acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} ;
struct account customer[100] ;
```

در این توصیف، customer آرایه‌ای ۱۰۰ عنصری است که هر عنصر آن ساختاری از نوع account است. درضمن هر دو روش توصیف بالا هم‌ارزند.

به آرایه‌ای از ساختارها نیز می‌توان مقادیر اولیه اختصاص داد. به‌خاطر داشته باشید که در اینجا عنصر آرایه ساختاری است که باید مقادیر اولیهٔ متناظر با هر عنصر به اعضای آن اختصاص داده شود. مثال ۹ـ٦ نحوهٔ عمل را در این مورد نشان می‌دهد.

ـ **مثال ۹ـ٦** برنامه‌ای شامل توصیف زیر است.

```
struct date {
            char name[80] ;
            int month ;
            int day ;
            int year ;
} ;
static struct date birthday[ ] = { "Sara" , 12 , 30 , 73 ,
                                   "Hassan" , 5 , 13 , 66 ,
                                   "Dara" , 7 , 15 , 72 ,
                                   "Iraj" , 11 , 29 , 70 ,
                                   "Arash" , 2 , 4 , 77 ,
                                   "Susan" , 12 , 29 , 63 ,
                                   "Ahmad" , 4 , 12 , 69 } ;
```

در این مثال birthday آرایه‌ای از ساختارهاست که اندازهٔ آن مشخص نشده است. مقادیر اولیه اندازهٔ آرایه و حافظهٔ لازم را تعریف می‌کند. لذا اندازهٔ آن برابر ۷ خواهد بود، زیرا ۷ سطر از مقادیر اولیه منظور شده است که از صفر تا ٦ شماره‌گذاری خواهد شد.

ممکن است بعضی برنامه‌نویسان ترجیح دهند که هر مجموعه از ثابتها را با رعایت ترتیب آنها در داخل یک زوج آکولاد جداگانه قرار دهند. توصیف آرایهٔ مزبور با این شیوه، به صورت زیر خواهد بود.

```
static struct date birthday[ ] = {
                                {"Sara" , 12 , 30 , 73} ;
                                {"Hassan" , 5 , 13 , 66} ;
                                {"Dara" , 7 , 15 , 72} ;
                                {"Iraj" , 11 , 29 , 70} ;
                                {"Arash" , 2 , 4 , 77} ;
                                {"Susan" , 12 , 29 , 63} ;
                                {"Ahmad" , 4 , 12 , 69}
} ;
```

پردازش ساختار

اعضای ساختار معمولاً به صورت مستقل و جدا از هم پردازش می‌شوند، یعنی هر عضو ساختار، به عنوان هویت مستقل، به صورت جداگانه پردازش می‌شود. به هر عضو یا عنصر ساختار با استفاده از عملگر `.` یا عملگر عضویت (که گاهی آن را عملگر period یا dot نیز نامند) دستیابی یا رجوع می‌شود. با استفاده از این عملگر می‌توان به هر عنصر ساختار به صورت زیر دست یافت.

structure-name. element-name

یا

variable. member

اپراتور نقطه نسبت به سایر اپراتورها، حتی اپراتورهای یکانی، تقدم بالایی دارد. برای مثال دو عبارت

++ variable. member و ++ (variable. member)

معادل هم‌اند؛ یعنی اپراتور ++ به عضو ساختار عمل خواهد کرد نه به تمام متغیر ساختار. به طریق مشابه، دو عبارت

&variable. member و &(variable. member)

معادل یکدیگرند. بنابراین هر دو عبارت مزبور به آدرس عضو ساختار دسترسی می‌یابند، نه به آدرس متغیر ساختار.

ـ **مثال ۹ـ۷** برنامه‌ای بنویسید که عملیات زیر را انجام دهد.

۱. مشخصات n دانشجو را که فرمت آن به صورت زیر است، به آرایه‌ای از ساختار

بخواند و سپس سابقهٔ هر دانشجو را در سطری جدا چاپ کند. n حداکثر ۲۵ است که از دستگاه ورودی استاندارد دریافت می‌شود.

۲. شمارهٔ دانشجویی را از ورودی بخواند و در لیست دانشجویان جستجو کند. اگر وجود داشت بقیهٔ مشخصات وی چاپ شود، وگرنه پیغام "not found" چاپ گردد.

۳. سوابق دانشجویان برحسب شمارهٔ دانشجویی به صورت صعودی مرتب شود و سپس سابقهٔ هر دانشجو در سطری جدا چاپ گردد.

st-no	name		grade
	l-name	f-name	

متغیرهای st-no، l-name، f-name و grade به ترتیب معرف شمارهٔ دانشجویی، نام خانوادگی، اسم کوچک و نمرهٔ امتحانی دانشجوست. درضمن ملاحظه می‌کنید که سابقهٔ هر دانشجو ساختاری شامل ۳ فیلد دارد: st-no، name ، grade که در آن فیلد یا عضو name خودش ساختاری دوعضوی است. پس در اینجا با ساختارهای تودرتو مواجهیم. اگر اسم ساختار اصلی را با strec نمایش دهیم، به هرکدام از اعضای آن می‌توان به‌ترتیب با strec.st-no , strec.name , strec.grade رجوع کرد. همچنین به عناصر یا اعضای name می‌توان با strec.name.f-name , strec.name.l-name رجوع کرد. ملاحظه می‌کنید که برای دستیابی به اعضای درونی ساختار، با اپراتور نقطه به صورت تکراری یا تودرتو مواجهیم. اگر اسم ساختار اصلی را با strec نمایش دهیم به هرکدام از اعضای آن می‌توان به‌ترتیب با strec.st-no , strec.name , strec.grade رجوع کرد. همچنین به عناصر یا اعضای name می‌توان با strec.name.f-name , strec.name.l-name رجوع کرد. ملاحظه می‌کنید که برای دستیابی به اعضای درونی ساختار، اپراتور نقطه به‌صورت تکراری یا تودرتو به کار رفته است.

برنامهٔ مورد نظر به صورت زیر است.

```
# include <stdio.h>
main ()
{ struct rec
        {
            int st-no ;
            struct name ;
```

```
                    {
                        char l-name[15] ;
                        char f-name[15] ;
                    } name1 ;
                float grade ;
            } ;
int i , j , n , s-n ;
struct rec strec[25] , temp ;
printf("enter the number of students \n") ;
scanf ("%d" , &n) ;
printf ("\n\n enter your data: \n") ;
for (i=0 ; i<n ; + + i)
{
    scanf ("%d" , &strec[i]. st-no) ;
    scanf ("%s %s" , strec[i].name.l-name , strec[i].name.f-name) ;
    scanf ("%f" , &strec[i].grade) ;
}
printf("display the records of students") ;
for (i=0 ; i<n ; + + i)
    printf("\n %d %s %s %f" , strec[i].st-no ,
        strec[i].name.l-name , strec[i].name.f-name , strec[i].grade) ;
printf ("\n enter a student number for search") ;
scanf ("%d" , &s-no) ;
printf ("\n search through list for indicated student") ;
for (i=0 ; i<n ; + + i)
    if (s-no = = strec [i].st-no)
        break ;
    if (i < n)
        printf ("\n %s %s %f" , strec[i].name.l-name ,strec.name.f-name ,
                                                    strec[i].grade) ;
    else
        printf("\n sort the student records in ascending order);
printf ("\n sort the student records in ascending order with respect to the student
                                                    number") ;
for (i=0 ; i<n ; + +i)
    for (j=i+1 ; j<n ; + + j)
        if (strec[i]. st-no>strec[i]. st-no)
            {
                temp = strec[i] ;
                strec[i] = strec[j] ;
                strec[j] = temp ;
            }
printf ("\n display the sorted records of students") ;
for (i=0 ; i<n ; + + i)
    printf ("\n %d %s %s %f" , strec[i].st-no ,
    strec[i].name.l-name , strec[i].name.f-name , strec[i].grade) ;
```

}

انتقال ساختار به تابع

می‌توان اعضای ساختار یا تمامی ساختار را به تابع گذر داد. راههای مختلفی برای انتقال یا گذر اطلاعات از نوع ساختار به تابع و بالعکس وجود دارد. سازوکار انتقال برحسب اینکه بعضی اعضا یا تمامی ساختار را انتقال دهیم و همچنین برحسب گونه‌های مختلف C متفاوت است.

اعضا یا عناصر ساختار را می‌توان هنگام فراخوانی تابع، به عنوان آرگومان، به آن انتقال داد یا اینکه عضو خاصی از ساختار را با دستور return با تابعی به تابع فراخوانندهٔ آن برگرداند. برای انجام این کار، با هریک از اعضای ساختار مشابه متغیری معمولی و تک‌مقدار برخورد می‌شود.

ـ **مثال ۹ـ۸** شکل کلی برنامهٔ C در زیر نشان داده شده است.

```
main ()
{
        typedef struct { /* structure declaration */
         int month ;
         int day ;
         int year ;
 }date ;
        suruct {/* structure declaration */
         int acct-no ;
         char acct-type ;
         char name[80] ;
         float balance ;
         date lastpayment ;
 }customer ;
 float adjust (char name[ ] , int acct-no , float balance) ;
        ...
customer.balance = adjust (customer.name , customer.acct-no , customer.balance) ;
        ...
}
float adjust (char name[ ] , int acct-no , float balance)
{
        float newbalance ;      /* local variable declaration */
        ...
        newbalance = … ;        /* adjust value of balance */
        ...
```

```
        return (newbalance) ;
}
```

این برنامه شیوهٔ انتقال اعضای ساختار به تابع را نمایش می‌دهد. در اینجا مقادیر

customer.acct-no , customer.name , customer.balance

به تابع adjust گذر داده شده‌اند. در داخل تابع مزبور، مقداری که به newbalance اختصاص می‌یابد احتمالاً از انجام عملیات روی اطلاعات گذر داده شده به دست می‌آید. سپس این مقدار به تابع اصلی برگردانده می‌شود که در آن به عضو customer.balance از متغیر ساختار customer اختصاص می‌یابد.

یادآوری. می‌توان تابع پیش‌نمونهٔ adjust در تابع main() را به صورت زیر نیز توصیف کرد.

```
float adjust (char[ ] , int , float) ;
```

به کمک اشاره‌گر به نوع ساختار می‌توان تمامی ساختار را به عنوان آرگومان به تابع انتقال داد. این شیوه، مشابه انتقال آرایه به تابع است. بدیهی است که این روش روش انتقال با آدرس است. بنابراین نتیجهٔ هر عضو ساختار که در درون تابع فراخوانده شده تغییر یابد، در تابع فراخواننده نیز تشخیص داده خواهد شد. لذا باز هم تشابه مستقیم بین این انتقال و آنچه در مورد انتقال آرایه به تابع گفتیم مشاهده می‌کنید.

مثال ۹ـ۹ برنامهٔ سادهٔ زیر را در نظر بگیرید.

```
typedef struct {
        char *name ;
        int acct-no ;
        char acct-type ;
        float balance ;
} record ;
main () / * transfer a structure-type pointer to a function */
 {
 void adjust (record *pt) ;
 static record customer = {"Nader , 3333 , `c´ , 33.33} ;
 printf (" %s %d %c %.2f \n" , customer.name , customer.acct-no ,
        customer.acct-type , customer.balance) ;
 adjust (&customer) ;
         printf (" %s %d %c %.2f \n" , customer.name , customer.acct-no ,
         customer.acct-type , customer.balance) ;
```

```
}
void adjust (record *pt)
{
        pt-> name = "Payam" ;
        pt-> acct-no = 9999 ;
        pt-> acct-type = `r´ ;
pt-> balance = 99.99
        return ;
}
```

این برنامه نحوهٔ انتقال ساختار به تابع را با گذر دادن آدرس آن (یعنی اشاره‌گر به ساختار) به تابع نمایش می‌دهد.

customer از نوع ساختار است و از لحاظ کلاس حافظه نیز ایستاست ، که به اعضای آن مقادیر اولیه اختصاص داده شده است. ابتدا در تابع اصلی، این مقادیر اولیهٔ اعضای customer با دستور printf نمایش داده می‌شود. سپس ضمن فراخوانی تابع adjust، آدرس ساختار به آن تابع گذر داده می‌شود. در تابع مزبور به اعضای customer مقادیر دیگری نسبت داده می‌شود. ملاحظه می‌کنید که در این تابع، متغیر pt اشاره‌گر به ساختار تعریف شده است که آدرس ساختار customer را دریافت می‌کند. پس از برگشت کنترل به تابع اصلی، برای بار دوم مقدار اعضای customer نمایش داده می‌شود.

اگر برنامهٔ مزبور اجرا شود، خروجی زیر را مشاهده خواهید کرد.

```
Nader 3333 c 33.33
Payam 9999 r 99.99
_
```

بازگشت اشاره‌گر به ساختار (با تابع)

تابع ممکن است اشاره‌گری به ساختار را به توابع فراخوانندهٔ آن برگرداند. مثال ۹ـ۱۰ نحوهٔ عمل را نمایش می‌دهد.

ـ **مثال ۹ـ۱۰** برنامهٔ زیر با فراخواندن تابعی به نام search، آرایه‌ای از ساختارها که شامل سابقهٔ مشتری بانک است و همچنین شماره حساب مشتری را در اختیار تابع قرار می‌دهد. تابع مزبور شماره حساب مورد نظر را در مجموعهٔ فهرست مشتریان جستجو می‌کند. اگر چنین رکوردی وجود داشت آدرس آن وگرنه، NULL (صفر) برمی‌گرداند. سپس در تابع

اصلی، درصورت وجود داشتن چنین رکوردی، بقیهٔ مشخصات صاحب شمارهٔ حساب مذکور نمایش داده می‌شود، در غیر این صورت پیغام مناسبی چاپ می‌شود. عمل فراخوانی تابع برای پیدا کردن رکورد خاص تا موقعی که شمارهٔ حساب ارسالی به تابع مخالف صفر باشد ادامه می‌یابد. سوابق دارندگان حساب به صورت مقدار اولیه به آرایه‌ای از ساختارها نسبت داده شده است.

```
# include<stdio.h>
# define n 3
# define NULL 0
typedef stuct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                   float balance ;
              } record ;
main ()
{
   static record customer[ ] = {
                                {"Nader" , 3333 ,'c' 33.33} ,
                                {"Payam" , 6666 ,'0' , 66.66} ,
                                {"Amir" , 9999 ,'d' , 99.99) ,
                                } ;                    /* array of structures */
   int acctn ;
   record *pt ;   /* pointer declaration */
   record *search (record table[ ] , int acctn) ;      /* function declaration */
   printf ("Customer Account Locator \n") ;
   printf ("To End , Enter 0 for the account number \n") ;
   printf ("\n Account no.: ") ;    /* enter first account number */
   scanf ("%d" , & acctn) ;
   while (acctn !=0)
        {
         pt = search (customer , acctn) ;  /* found a match */
         if (pt != NULL)
            {
              printf ("\n Name:  %s \n" , pt -> name) ;
              printf ("Accont no.:  %d \n" , pt -> acct-no) ;
              printf ("Account type:  %c \n" , pt -> acct-type) ;
              printf ("Balance:  %.2f \n" , pt -> balance) ;
            }
          else
            printf ("\n error-please try again\n") ;
```

```
            printf ("\n Account no.: ") ;       /* enter next account number*/
            scanf ("%d" , & acctn) ;
        }
 }
record *search (record table[ ] , int acctn)          / * function definition */
                    /* accept an array of structures and an account number ,
                    return a pointer to a particular structure (an array element)
                    if the account number matches a member of that structure */
{
   int count ;
   for (count = 0 ; count<n ; + + count)
     if (table [count].acct-no = = acctn)   /* found a match */
        return (&table[count]) ;   /* return pointer to array element */
   return (NULL) ;
}
```

اندازهٔ آرایهٔ مورد نظر با ثابت سمبولیکی n بیان شده و مقدار آن ۳ است؛ یعنی برای سادگی کار، فقط ۳ رکورد، برای نمونه ذخیره شده است. بدیهی است که در عمل، مقدار n خیلی بزرگ‌تر خواهد بود.

می‌توان تمامی ساختار را به طور مستقیم به عنوان آرگومان به تابع انتقال داد یا با دستور return ساختار را به طور مستقیم از تابع برگرداند (برخلاف آرایه که با تابع برگردانده نمی‌شود). چنین انتقال ساختاری به‌صورت مقدار خواهد بود. بنابراین اگر با تابع تغییراتی در اعضای ساختار داده شود، نتیجهٔ آن در تابع فراخواننده یا در خارج از تابع مذکور اعمال نخواهد شد. به هرحال اگر ساختار تغییریافته به نقطهٔ فراخوانی از برنامه برگردانده شود، تغییرات حاصل در اینجا شناخته خواهد شد.

مثال ۹ـ۱۱ برنامهٔ زیر را ملاحظه کنید.

```
# include<stdio.h>
typedef struct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                    float balance ;
                  } record ;
main ()           /* transfer a structure to a function */
{
  void adjust (record customer) ;          /* function declaration */
```

```
  static record customer = {"Nader" , 3333 , 'c' , 33.33} ;
  printf ("%s %d %c %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
  abjust (customer) ;
  printf ("%s %d %c %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
}
void adjust (record cust)        /* function definition */
 {
   cust.name = "Payam" ;
   cust.acct-no = 9999 ;
   cust.acct-type ='r' ;
   cust.balance = 99.99 ;
   return ;
}
```

برنامهٔ مزبور به جای اشاره‌گر به نوع ساختار، تمامی ساختار (در واقع کپی ساختار) را به تابع می‌فرستند. حال تابع adjust ساختاری را به عنوان آرگومان دریافت می‌کند (برخلاف مثال ۹ـ۱۰ که اشاره‌گر به ساختار را دریافت می‌کرد). در اینجا چیزی از تابع به تابع main برگردانده نشده است.

اگر برنامهٔ مزبور اجرا شود، خروجی زیر حاصل خواهد شد.

```
Nader 3333 c 33.33
Nader 3333 c 33.33
```

در اینجا، نتیجهٔ اجرای دستورهای جایگذاری، که در تابع adjust به کار برده شده است، در تابع main شناخته نمی‌شود (یعنی نتیجه در تابع اصلی منعکس نمی‌شود). دلیل این کار نیز واضح است، زیرا انتقال ساختار customer از تابع main به تابع adjust با مقدار صورت گرفته است.

حال فرض کنید که این برنامه را به طریقی اصلاح کنیم که تغییرات انجام شده در تابع adjust به تابع main برگردانده شود.

```
#include<stdio.h>
typedef struct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                    float balance ;
                  } record ;
main ()          /* transfer a structure to a function and return the structure */
```

```
{
  record adjust (record customer) ;        /* function declaration */
  static record customer = {"Nader" , 3333 ,'C ' , 33.33} ;
  printf (" %s %d %s %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
  customer = adjust (customer) ;
  printf ("%s %d %.2f \n" , custo6mer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
}
record adjust (record cust)      /* function definition */
 {
  cust.name = "Payam" ;
  cust.acct-no = 9999 ;
  cust.acct-type ='r' ;
  cust.balance = 99.99 ;
  return (cust) ;
 }
```

در این برنامه، برخلاف مثال ۹ـ۱۰، تابع adjust ساختاری را که همان ساختار تغییریافته در تابع مزبور است به تابع main برمی‌گرداند.

اجرای این برنامه خروجی زیر را ایجاد می‌کند.

```
Nader 3333 c 33.33
Payam 9999 r 99.99
```

بنابراین تغییرات اعمال شده در تابع adjust در تابع main نیز منعکس شده است، زیرا این بار ساختار تغییریافته به طور مستقیم به قسمت فراخوانی برنامه برگردانده شده است.

نوع داده کاربر

زبان برنامه‌نویسی C، دستور ویژه‌ای را معرفی می‌کند که اجازه می‌دهد تا کاربران بتوانند نام جدیدی برای نوع داده تعریف کنند. نام جدید معادل نوع داده مورد نظر خواهد بود. این کار به کمک کلمهٔ کلیدی typedef انجام می‌گیرد. در واقع در اینجا، کلاس جدیدی از داده‌ها ایجاد نمی‌گردد، بلکه برای نوع داده موجود نام جدیدی تعریف می‌گردد. پس از آنکه نام جدید برای نوع داده مورد نظر تعریف گردید، می‌توان متغیرها، آرایه‌ها، ساختارها و غیره را برحسب نوع داده جدید توصیف کرد.

شکل کلی دستور typedef به صورت زیر است.

```
typedef type name ;
```

یا

```
typedef type new-type ;
```

که در آن type هریک از نوع داده‌های مجاز است و name یا new-type نیز نام جدید برای این نوع است. در واقع type یا یکی از نوع داده‌های استاندارد (مانند int، float، char و...) یا نوع داده تعریف شدهٔ کاربر است که قبلاً تعریف کردیم. به هرحال باید توجه کرد که نوع داده جدید فقط از نظر نام آن جدید است وگرنه همان طور که گفتیم، کلاس جدیدی از داده‌ها نیست.

ـ **مثال ۹ـ۱۲** به دستور زیر توجه کنید.

```
typedef int age ;
```

در این دستور، برای نوع داده int، نام جدید age انتخاب شده است. بنابراین از این لحظه به بعد هر کجا نیاز باشد که داده‌ای به صورت int تعریف شود، می‌توان آن را با نوع age تعریف کرد. بنابراین دو توصیف زیر هم‌ارزند.

```
age a , b ;
int a , b ;
```

همین طور در دستورهای

```
typedef float height[100] ;
height a , b ;
```

دستور اول، height را آرایه‌ای ۱۰۰ عنصری از نوع float تعریف می‌کند. دستور دوم، a و b را آرایه‌های ۱۰۰ عنصری از نوع float توصیف می‌کنند.

راه هم‌ارز دیگری برای بیان دو دستور بالا عبارت است از

```
typedef float height ;
height a[100] , b[100] ;
```

گرچه روش قبلی ساده‌تر است.

ویژگی typedef، خصوصاً در تعریف ساختارها، کار را ساده‌تر می‌کند و کاربر را از نوشتن تکراری کلمهٔ کلیدی struct خلاص می‌کند. در چند مثال قبل در مورد ساختارها نیز از این دستور استفاده شد.

در شکل کلی، نوع ساختاری که کاربر تعریف می‌کند به صورت زیر است.

```
typedef struct {
        member 1 ;
```

```
        member 2 ;
        ...
        member m ;
} new-type ;
```

که در آن new-type نوع ساختار تعریف شدهٔ کاربر است. حال متغیرهای ساختار برحسب این نوع داده جدید (درواقع نام جدید برای نوع داده) تعریف می‌شوند.

ـ **مثال ۹ـ۱۳** دستورهای زیر برای تعریف متغیرهای ساختار به کار می‌رود. در اینجا برای شرح ساختار از نوع داده تعریف شدهٔ کاربر استفاده شده‌است.

```
typedef struct {
                    int acct_no ;
                    char acct_type ;
                    char name[80] ;
                    float balance ;
                  } Trecord ;
Trecord oldcustomer , newcustomer ;
```

Trecord از نوع دادهٔ تعریف شدهٔ کاربر توصیف شده است و متغیرهای oldcustomer و newcustomer که ساختارند از نوع Trecord تعریف شده‌اند.

ـ **مثال ۹ـ۱٤** دستورهای زیر نمونهٔ دیگری از کاربرد دستور typedef را نمایش می‌دهد.

```
typedef struct {
                    int day ;
                    int month ;
                    int year ;
                  } date ;
typedef struct {
                    int st_no ;
                    char name[20] ;
                    date test ;
                  } student ;
student Pnoor [100];
```

ساختار داده‌ها و اشاره‌گرها

می‌توان با به کار بردن عملگر آدرس، یعنی ´&`، به آدرس ابتدای ساختار دسترسی داشت

(مشابه دستیابی به سایر آدرسها). برای مثال اگر variable معرف متغیری از نوع ساختار باشد، &variable آدرس ابتدای آن متغیر را نشان خواهد داد. همچنین می‌توان متغیر اشاره‌گری به ساختار به صورت زیر تعریف کرد.

```
type *ptvar ;
```

که در آن type نوع داده است که دلالت بر ترکیب ساختار می‌کند و ptvar نیز معرف نام متغیر اشاره‌گر است. حال می‌توان آدرس آغاز متغیر ساختار را به صورت زیر به این اشاره‌گر نسبت داد.

```
ptvar = &variable ;
```

ـ **مثال ۹ـ۱۵** تعریف زیر را در مورد ساختار در نظر بگیرید.

```
typedef struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
        float balance ;
} account ;
account customer , *pc ;
```

در این مثال، customer متغیر ساختار از نوع account و pc نیز متغیر اشاره‌گر است که آدرس متغیر ساختار از نوع account را در خود دارد. حال با دستور زیر، آدرس آغاز customer به pc نسبت داده می‌شود.

```
pc = &customer ;
```

توصیف متغیر اشاره‌گر را می‌توان به صورت زیر با توصیف ساختار ترکیب کرد.

```
struct {
        member 1 ;
        member 2 ;
        ...
        member m ;
} variable , *ptvar ;
```

که در آن، variable، متغیر از نوع ساختار و ptvar نیز نام متغیر اشاره‌گر را معرفی می‌کند.

ـ **مثال ۹ـ۱۶** توصیف زیر معادل دو توصیف معرفی شده در مثال ۹ـ۱۵ است.

```
struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
```

```
        float balance ;
} customer , *pc ;
```

حال می‌توان، مشابه مثال قبل، آدرس آغاز customer را با دستور زیر به متغیر اشاره‌گر pc نسبت داد.

```
pc = &customer ;
```

به عضو ساختار می‌توان به صورت ptvar -> member دسترسی داشت که در آن ptvar متغیر اشاره‌گر به نوع ساختار مورد نظر است و می‌توان آن را با عملگر نقطه، یعنی `.`، قیاس کرد. بنابراین عبارت ptvar -> member هم‌ارز عبارت variable. member است که در آن variable متغیری از نوع ساختار است که دربارهٔ آن بحث کردیم.

عملگر -> نیز مانند عملگر `.` بالاترین تقدم را دارد. عملگر -> با عملگر نقطه ترکیب می‌شود و بدین طریق عمل دستیابی به زیرعضو یا submember را در درون ساختار (یعنی دستیابی به عضوی از ساختار که خودش عضو ساختار دیگر است) فراهم می‌سازد. بنابراین می‌توان به زیرعضوی به صورت زیر دسترسی داشت.

```
ptvar ->member. submember
```

به چند روش مشابه، عملگر -> برای دستیابی به عنصری از آرایه که خودش عضوی از ساختار است به کار می‌رود. این عمل با دستوری مشابه زیر میسر است.

```
ptvar -> member [expression]
```

که در آن expression مقدار صحیح غیرمنفی است و بر عنصر آرایه دلالت می‌کند (یعنی عنصر آرایه را مشخص می‌سازد).

مثال ۹ـ۱۷ تعریف ساختار و متغیرهایی را به صورت زیر درنظر بگیرید.

```
typedef struct {
        int month ;
        int day ;
        int year ;
} date ;
struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
        float balance ;
        date lastpayment ;
```

} customer , *pc = &customer ;

توجه داشته باشید که به متغیر اشاره‌گر pc، آدرس آغاز متغیر customer (که از نوع ساختار است) به صورت مقدار اولیه نسبت داده شده است. به عبارت دیگر، متغیر pc به customer اشاره می‌کند.

حال اگر بخواهیم به شمارهٔ حساب مشتری دسترسی پیدا کنیم، این کار به یکی از سه روش زیر انجام می‌گیرد.

```
customer. acct_no
pc-> acct_no
(*pc). acct_no
```

وجود پرانتز در مورد عبارت آخری ضروری است، زیرا عملگر نقطه نسبت به عملگر `*` تقدم بالاتری دارد و چنانچه پرانتز به کار نرود، کامپایلر نتیجهٔ اشتباه ایجاد می‌کند، زیرا pc که اشاره‌گر است، به صورت مستقیم با عملگر نقطه سازگار نیست.

به طریق مشابه می‌توان به موجودی مشتری به یکی از سه روش زیر دسترسی یافت.

```
customer. balance
pc-> balance
(*pc). balance
```

و به ماه آخرین پرداخت به یکی از دو روش زیر.

```
customer. lastpayment. month
pc-> lastpayment. month
```

و بالاخره به نام مشتری به یکی از سه روش زیر می‌توان دسترسی داشت.

```
(*pc). lastpayment. month
pc->name
(*pc). name
```

همین طور به سومین کاراکتر نام مشتری به یکی از روشهای زیر دسترسی می‌یابیم.

```
customer. name[2]
*(customer. name + 2)
pc->name[2]
pc->(name+2)
(*pc). name[2]
*((*pc). name + 2)
```

یادآور می‌شویم که اعضای ساختار می‌تواند اشاره‌گر نیز باشد. این روش در ساختارهایی مانند لیستهای پیوندی، درختها، نمودارها و مشابه آن کاربرد زیادی دارد.

عضو ساختار

متغیر اشاره‌گر می‌تواند عضو ساختار باشد. برای مثال مشخصات نفری با فرمت

نام	نام خانوادگی

را می‌توان به صورت ساختار زیر تعریف کرد.

```
struct names {
        char *lastname ;
        char *firstname ;
};
```

که در اینجا، lastname و firstname اشاره‌گرهایی‌اند که در واقع معرف دو رشته (یا دو آرایهٔ کاراکتری) اند.

همچنین عضو ساختار ممکن است اشاره‌گری باشد که آدرس عناصر دیگر یا آدرس ساختار دیگری را در خود داشته باشد یا آدرس ساختاری را از نوع خودش (یعنی ساختاری که در درون آن تعریف شده است) نگهداری کند.

در حالت کلی می‌توان این گونه ساختارها را به صورت زیر تعریف کرد.

```
struct tag {
        member 1 ;
        member 2 ;
        ....
        ....
        ....
         struct tag &name ;
        } ;
```

که در آن name متغیری از نوع اشاره‌گر است که آدرس متغیر دیگری از نوع ساختار به شکل tag در آن قرار می‌گیرد. یکی از مهم‌ترین کاربردهای روش بالا، در ایجاد ساختارهایی به نام لیست پیوندی و انجام عملیات یا پردازش روی آن است.

اجتماع[1]

اجتماع مانند ساختار داده‌هایی است که از چند عضو تشکیل می‌شود و هر عضو آن نوع

1. union

داده‌ای منحصر به خود دارد. اجتماع محلی از حافظه است که چندین متغیر در آن قرار می‌گیرند که ممکن است نوع آنها نیز یکسان نباشد. در واقع اجتماع متغیری است که امکان ذخیره کردن انواع مختلف داده در مکان مشترکی از حافظه را فراهم می‌کند.

تعریف اجتماع مشابه تعریف ساختار است با این تفاوت که در اجتماع تمام اعضای تشکیل‌دهندهٔ آن فضایی را به صورت اشتراکی اشغال می‌کنند (برخلاف ساختار که هر عضو آن محل حافظه یا مکان خاص خودش را داراست)، از این رو به منظور صرفه‌جویی در حافظه به کار گرفته می‌شود. این گونه ساختمان داده‌ها در کاربردهایی مفیدند که اعضای چندگانه از نوع مختلف دارند، ولی در هر زمان باید فقط به یکی از این اعضا مقداری نسبت داد. به هرحال کاربر باید بداند که هر لحظه چه نوع داده در حافظهٔ مشترک مورد نظر ذخیره شده است.

در حالت کلی ترکیب اجتماع ممکن است به صورت زیر تعریف شود.

```
union tag {
        member 1 ;
        member 2 ;
        …
        …
        …
        member m ;
} ;
```

که در آن union کلمه‌ای کلیدی است.

ـ **مثال ۹ـ۱۸** به تعریف زیر توجه کنید.

```
union union_type {
        int i ;
        char ch ;
        } ;
```

این تعریف نیز مشابه تعریف ساختار، موجب توصیف یا اعلان متغیر نمی‌گردد، بلکه فقط تعریف نوع داده است. برای اعلان هر متغیری از نوع اجتماع مورد نظر، باید نام آن را به دنبال تعریف union به کار برد یا به طور جداگانه آن را اعلان کرد. پس روش اعلان متغیرهایی از نوع اجتماع مشابه اعلان از نوع ساختار است.

ـ

با توجه به مطالب بالا، متغیر x به دو روش اعلان شده است که نتیجهٔ نهایی هر دو هم‌ارز است.

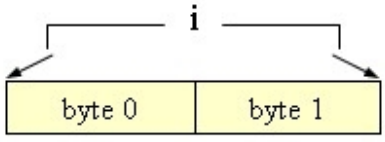
روش اول

```
union tag {
    int i ;
    char ch ;
} ;
union tag x ;
```

روش دوم

```
union tag {
    int i ;
    char ch ;
} x ;
```

در اینجا، متغیرهای i و ch که به ترتیب از نوع int و char اند، حافظهٔ مشترک دارند که البته متغیر i دو بایت و متغیر ch یک بایت حافظه اشغال می‌کند، ولی آدرس شروع آنها یکسان است (شکل ۹ـ۲).



شکل ۹ـ۲ نحوهٔ اشغال حافظهٔ مشترک با دو عضو اجتماع

وقتی که اجتماع اعلام می‌گردد، کامپایلر به طور خودکار متغیری که بتواند بزرگ‌ترین عضو اجتماع را در خود جای دهد ایجاد می‌کند که در مثال بالا بزرگی آن ۲ بایت است.

باتوجه به مطالبی که گفتیم، شکل کلی اعلان متغیرهایی از نوع اجتماع به صورت زیر است.

storage _ class union tag variable1 ,variable 2 , … , variable n ;

که در آن storage_class گزینهٔ اختیاری است که اگر به کار نبریم حافظهٔ مورد نظر از کلاس خودکار خواهد بود.

به طوری که گفتیم، می‌توان اعلان متغیرهایی از نوع اجتماع را به دنبال تعریف اجتماع به کار برد. پس می‌توان صورت بالا را چنین تعریف کرد.

```
storage_class union tag {
        member 1 ;
        member 2 ;
```

```
        ...
        ...
        ...
        member m ;
} variable 1 , variable 2 , … , variable n ;
```

ـ **مثال ۹ـ۱۹** به اعلان زیر توجه کنید.

```
union id {
        char color [12] ;
        int size ;
} shirt , blouse ;
```

در این مثال، دو متغیر shirt و blouse از نوع اجتماع‌اند که با نام id تعریف شده است. هرکدام از این دو متغیر در هر لحظه یا معرف رشتهٔ ۱۲ کاراکتری color یا معرف مقدار صحیح size خواهد بود. بدیهی است چون رشتهٔ مزبور ۱۲ کاراکتری است و به حافظهٔ بیشتری در مقایسه با size نیاز دارد، کامپایلر، به طور خودکار، ۱۲ بایت حافظه به هر متغیر اجتماع اختصاص خواهد داد.

ـ

اجتماع ممکن است عضوی از ساختار باشد. همچنین ساختار ممکن است عضوی از اجتماع باشد.

ـ **مثال ۹ـ۲۰** تعریف زیر را در نظر بگیرید.

```
union id {
        char color[12] ;
        int size ;
        struct clothes {
        char manufacturer[20] ;
        float cost ;
        union id description ;
        } shirt , blouse ;
}
```

در این مثال دو متغیر shirt و blouse متغیرهایی از نوع ساختارند که با نام clothes تعریف شده‌اند و هرکدام از آنها شامل اعضای زیر است: رشته‌ای به نام manufacturer (نام تولیدکنندهٔ لباس)، مقداری از نوع اعشار به نام cost (معرف بهای لباس) و اجتماعی به نام description (نوع لباس). در اینجا ممکن است که اجتماع معرف رشته یا color یا معرف مقدار صحیح یا size باشد. راه دیگری برای مشخص ساختن (اعلان) متغیرهای ساختار shirt

و blouse آن است که دو نوع اعلان بالا را با یکدیگر ترکیب کنیم و به‌صورت فشرده‌تر زیر بنویسیم.

```
struct clothes {
        char manufacturer[20] ;
        float cost ;
        union {
         char color[12] ;
         int size ;
        } description ;
} shirt , blouse ;
```

نحوهٔ دستیابی به عناصر یا اعضای اجتماع مشابه همان است که در مورد ساختارها گفتیم؛ یعنی این کار با استفاده از دو عملگر '.` و'->` انجام می‌گیرد. بنابراین اگر variable متغیر اجتماع باشد دستور variable. member یک عضو آن را معرفی و مشخص می‌کند. به طریق مشابه، اگر ptvar متغیر اشاره‌گر باشد که به اجتماع اشاره می‌کند، (یعنی آدرس متغیری از نوع اجتماع را در خود دارد) در این صورت ptvar->member به یک عضو آن اجتماع اشاره خواهد کرد.

ـ **مثال ۹ـ۲۱** برنامهٔ سادهٔ زیر را درنظر بگیرید.

```
# include < stdio. h>
main ()
{
  union id {
             char color ;
             int size ;
           } ;
   struct {
            char manufacturer[20] ;
            float cost ;
            union id description ;
          } shirt , blouse ;
  printf ("%d\n" , sizeof (union id)) ;
  shirt. description. color = `w' ;          /* assign a value to color */
  printf ("%c %d \n" , shirt. description. color , shirt. description. size) ;
  shirt. description. size = 12 ;    /* assign a value to size */
  printf ("%c %d\n" , shirt. description. color , shirt. description. size) ;
}
```

در این مثال، اولین عضو اجتماع تک‌کاراکتری است، برخلاف مثال قبلی که آرایه‌ای ۱۲ کاراکتری بود. انجام این تغییرات بدین لحاظ صورت گرفته که نسبت دادن مقادیر به اعضای اجتماع آسان‌تر باشد.

ملاحظه می‌کنید که به عضو shirt. description. Color مقدار 'w`نسبت داده شده است. پس باید توجه داشته باشید که عضو دیگر اجتماع، یعنی shirt. Description مقدار معنی نخواهد داشت. سپس با دستور printf، مقدار هر دو عضو اجتماع نمایش داده شده است. سپس به عضو shirt. description. size مقدار ۱۲ نسبت داده شده است. بدیهی است که این مقدار، روی مقدار حافظهٔ مشترک که برای دو عضو مزبور پیش‌بینی شده است خواهد نشست. سپس بار دیگر با دستور printf مقدار هر عضو نمایش داده شده است.

خروجی برنامهٔ مزبور به شکل زیر است.

```
2
w-24713
@ 12
```

سطر اول نشان می‌دهد که ۲ بایت حافظه به اجتماع اختصاص داده شده است تا بتواند بزرگ‌ترین عضو خود را که مقدار صحیح است در خود جای دهد. در سطر دوم، داده اولی 'w` است که معنی و مفهوم دارد. اما داده دوم، یعنی ۲۴۷۱۳–، معنی و مفهومی ندارد. در سطر سوم، داده اولی @ است که بدون معنی است. اما داده دوم که ۱۲ است معنی و مفهوم دارد.

شمارشی[1]

یکی از انواع داده‌های اسکالر نوع شمارشی است. بعضی زبانهای دیگر مانند زبان پاسکال نیز این نوع داده‌ها را در میان انواع داده‌های استاندارد پشتیبانی می‌کنند. داده‌ای از نوع شمارشی، مشابه ساختار یا اجتماع است و اعضای آن ثابتهایی‌اند که به عنوان شناسه نوشته می‌شوند، اگرچه مقدار یا ارزش آنها از نوع مقدار صحیح علامت‌دار است. این ثابتها مقادیری را معرفی می‌کنند که می‌توان به متغیرهای شمارشی متناظر با آنها نسبت داد.

در حالت کلی نوع شمارشی به صورت زیر تعریف می‌شود.

```
enum tag {member 1 , member 2 , … , member m} ;
```

1. enum

که در آن enum کلمه‌ای کلیدی است و tag نیز اسمی است که داده شمارشی را مشخص می‌کند و دارای این ترکیب است و عناصر member 1 , member 2 , … , member m نیز شناسه‌هایی را مشخص می‌کنند که ممکن است به متغیرهایی از نوع tag نسبت داده شود. اسامی اعضا باید متفاوت و متمایز از یکدیگر باشند.

وقتی که نوع شمارشی تعریف شد، می‌توان متغیرهای شمارشی متناظر با آن را به صورت زیر اعلان کرد.

storage_class enum tag variable1 , variable 2 , … , variable n ;

که در آن storage_class گزینهٔ اختیاری است و کلاس حافظه را مشخص می‌کند. enum نیز کلمه‌ای کلیدی است که باید به کار برده شود. tag اسمی است که در تعریف نوع شمارشی ظاهر می‌گردد و بالاخره variable1 , variable 2 , … , variable n متغیرهای شمارشی از نوع tag اند.

تعریف شمارشی را می‌توان با اعلان متغیرها ترکیب کرد و به صورت زیر به کار برد.

storage_class enum tag {member 1 , member 2 , … , member m}variable 1 , variable 2 , … , variable n ;

در این حالت، انتخاب نام برای مراجعه به آن اختیاری است و می‌توان آن را به کار نبرد. صورت بالا را می‌توان به اختصار چنین نوشت.

enum enum_type_name {enumeration list} variable_list ;

که در صورت مزبور به کار بردن نام نوع شمارشی، یعنی enum_type_name، اختیاری است. نام نوع شمارشی برای اعلان متغیرهایی از آن نوع است.

ـ **مثال ۹ـ۲۲** قطعه برنامهٔ زیر نوع شمارشی‌ای به نام coin تعریف و نوع متغیر money را از این نوع اعلان می‌کند.

```
enum coin {penny , nickel , dime , quarter , half_dollar , dollar} ;
enum coin money ;
```

با داشتن این تعریف و اعلان، دستورهای زیر کاملاً درست و معتبر است.

```
money = dime ;
if (money = = quarter) prinf ("is a quarter") ;
```

ـ

نکتهٔ مهمی که در اینجا باید در مورد نوع شمارشی توجه داشت آن است که هر سمبول معرف یک مقدار صحیح است و در هر عبارت از نوع مقادیر صحیح به کار می‌رود. برای مثال

عبارت

```
printf ("the number of nickels in a quarter is %d" , quarter + 2) ;
```

کاملاً صحیح و معتبر است.

مقدار اولین سمبول شمارشی برابر صفر، مقدار دومین سمبول شمارشی برابر ۱، و بالاخره مقدار n اُمین سمبول شمارشی برابر n – 1 است، مگر اینکه به طریق دیگری مقداردهی اولیه شده باشند. بنابراین عبارت ; ("%d %d" , penny , dime) printf مقادیر 2 0 را در صفحهٔ تصویر نمایش خواهد داد.

همچنین می‌توان به هریک از سمبولها مقدار اولیه نسبت داد. برای این کار باید پس از سمبول مورد نظر علامت '=` و سپس مقدار صحیح مطلوب را به کار ببریم. وقتی که به یکی از سمبولها به طریق مزبور مقدار اولیه نسبت داده شد، سمبول بعدی مقدار بعدی را خواهد داشت؛ یعنی یک واحد از آن بزرگ‌تر خواهد بود. برای مثال دستور زیر مقدار ۱۰۰ را به quarter نسبت می‌دهد.

```
enum coin {penny , nickel , dime , quarter = 100 , half_dollar , dollar} ;
```

و اکنون مقادیر سمبولها به صورت زیر خواهند بود.

```
penny   0
nickel  1
dime    2
quarter 100
half_dollar     101
dollar  102
```

به طور متعارف این طور فرض می‌شود که سمبولهای یک نوع شمارشی می‌توانند به طور مستقیم ورودی یا خروجی باشند. ولی این طور نیست. برای مثال قطعه برنامهٔ زیر آنچه را انتظار داریم انجام نمی‌دهد.

```
money = dollar ;
printf ("%d" , money) ;
```

به خاطر داشته باشید سمبول dollar به‌طور ساده مقدار رشته نیست، بلکه اسمی برای مقدار صحیح است. بنابراین تابع printf نمی‌تواند رشتهٔ "dollar" را نمایش دهد. به طریق مشابه نمی‌توانید با به کار بردن رشتهٔ هم‌ارز، به متغیر شمارشی مقدار نسبت بدهید. برای مثال دستور زیر درست کار نمی‌کند.

```
money = "penny" ;
```

به هرحال به طریق دیگری می‌توان مقدار رشته‌ای سمبولهای مورد نظر را ایجاد کرد.

ـ **مثال ۹ـ۲۳** قطعه برنامهٔ زیر نوع سکه‌هایی را که متغیر money شامل است نمایش خواهد داد.

```
switch money {
        case penny:  printf ("penny") ;
                break ;
        case nickel:  printf ("nickel") ;
                break ;
        case dime:  printf ("dime") ;
                break ;
        case quarter:  printf ("quarter") ;
                break ;
        case half_dollar:  printf ("half_dollar") ;
                break ;
        case dollar:  printf ("dollar") ;
        }
```

←

همچنین ممکن است آرایه‌ای از رشته‌ها تعریف کرد و مقدار متغیر شمارشی را شاخص یا index آن آرایه به کار برد تا مقدار شمارشی را به رشتهٔ متناظر آن ترجمه کند.

ـ **مثال ۹ـ۲٤** قطعه برنامهٔ زیر رشتهٔ مورد نظر را به عنوان خروجی تولید خواهد کرد.

```
char name[ ] = {
         "penny" ,
         "nickel" ,
         "dime" ,
         "quarter" ,
         "half_dollar" ,
         "dollar"
        } ;
        printf ("%c" , name [(int) money]) ;
```

البته این روش در صورتی درست کار می‌کند که به هیچ یک از سمبولها مقدار اولیه نسبت داده نشده باشد، زیرا شاخص آرایهٔ رشته‌ها همیشه از صفر شروع می‌شود.

←

ـ **مثال ۹ـ۲۵** فرض کنید که برنامهٔ C شامل دستورهای زیر باشد.

```
enum colors { black , blue , cyan , green , magenta , red , white , yellow} ;
colors foreground , background ;
```

در سطر اول، نوع شمارشی به نام colors تعریف شده که شامل هشت ثابت است و اسامی آنها عبارت‌اند از black , blue , cyan , green , magenta , red , white , yellow. در سطر دوم متغیرهای foreground و background به عنوان شمارشی، از نوع colors اعلان شده‌اند. بنابراین به هریک از این متغیرها می‌توان هر یک از هشت ثابت black , blue , cyan , yellow , ... را نسبت داد.

می‌توان دو دستور بالا را با یکدیگر ترکیب کرد و به صورت زیر نوشت.

```
enum colors {black , blue , cyan , green , magenta , red , white , yellow}foreground
, background ;
```

در این مثال مقادیر صحیح هشت ثابت بالا به صورت زیر خواهد بود.

black	0
blue	1
cyan	2
green	3
magenta	4
red	5
white	6
yellow	7

به هرحال به طوری که گفتیم، اگر به برخی از ثابتها مقدار اولیه نسبت داده شود، مقادیر مزبور تغییر خواهد یافت. برای مثال اگر نوع شمارشی مزبور را به صورت

```
enum colors {black= -1 , blue , cyan , green , magenta , red = 2 , white ,
                                                          yellow} ;
```

تعریف کنیم، هشت ثابت مورد نظر مقادیر زیر را خواهند داشت.

black	-1
blue	0
cyan	1
green	2
magenta	3
red	2
white	3
yellow	4

متغیرهای شمارشی نمی‌توانند به طور کامل مشابه متغیرهای از نوع مقدار صحیح مورد پردازش قرار گیرند. مثلاً نمی‌توان به آنها مقدار جدید نسبت داد یا آنها را مقایسه کرد. همین طور نمی‌توان آنها را به عنوان ورودی به درون کامپیوتر خواند و به متغیر شمارشی دیگر نسبت داد. اما می‌توان مقدار صحیح را از ورودی دریافت کرد و آن را به متغیر شمارشی نسبت

داد. گرچه این شیوه متداول نیست. همچنین فقط می‌توان مقدار صحیح متغیر شمارشی را به عنوان خروجی کامپیوتر نوشت.

حال دوباره دو دستور مذکور در آغاز این مثال را درنظر بگیرید. با درنظر گرفتن دو دستور مزبور می‌توان دستورهای زیر را به عنوان نمونه‌هایی از عملیات روی متغیرهای شمارشی به کار برد.

```
foreground = white ;
background = blue ;
if (background = = blue)
  foreground = yellow ;
else
  foreground = white ;
if (foreground = = background)
   foreground = (enum colors) (+ + background % 8) ;
switch (background)
   {
     case black:  foreground = white ;
     break ;
     case blue:
     case cyan:
     case green:
     case magenta:
     case red:  foreground = yellow ;
          break ;
     case white:  foreground = black ;
          break ;
     case yellow:  foreground = blue ;
          break ;
     case default:  printf ("Error in selection of background color ") ;
 }
```

خودآزمایی ۹

۱. ساختاری برای مشخصات دانشجویی زیر بنویسید.

شمارهٔ دانشجو	نام	تاریخ تولد

		ماه	روز	سال
123456	Amiri	11	5	1980

۲. با استفاده از ساختار برنامه‌ای بنویسید که نام n نفر همراه با شماره تلفنشان را ذخیره کند. سپس شمار تلفن نام فردی را که گرفته نمایش دهید.

۳. خروجی این برنامه چیست ؟

```
union un{
   int i ;
   char c ;
} s ;
main ()
 {
   union un *p ;
   s.c = 'A' ;
   p = &s ;
   printf (" %d %c %d %c" , s.i , s.c , p-> i , p-> c) ;
}
```

٤. برنامه‌ای بنویسید که با استفاده از اجتماعها توان اعشاری اعداد ($y = x^n$) را محاسبه و چاپ کند.

فصل ۱۰

فایلها

هدف کلی

آشنایی با انواع فایل و توابع مربوط به آنها

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از خواندن این فصل،

۱. فایل را تعریف کند و انواع آن را بشناسد.

۲. تفاوت فایلهای متنی و باینری و کاربرد هر یک را بداند.

۳. با نحوهٔ ذخیره و بازیابی داده‌ها آشنا شود.

٤. با بازکردن فایل و اطلاعاتی که همزمان با بازشدن فایل مشخص می‌شود آشنایی یابد.

۵. فایل ورودی، خروجی، و ورودی–خروجی را بشناسد.

٦. با توابع fopen و fclose آشنایی یابد.

۷. توابع putc و getc را بشناسد.

۸. کاربرد توابع getw و putw را بداند.

۹. کاربرد توابع fget و fputs را بداند.

۱۰. نقش فایل را به عنوان وسیله‌ای هم ورودی و هم خروجی بداند و تابع rewind را بشناسد.

۱۱. با کاربرد تابع ferror آشنایی بیابد.

۱۲. با کاربرد تابع remove آشنایی بیابد.

۱۳. توابع fscanf و fprintf را بشناسد.
۱۴. توابع fwrite و fread را بشناسد.
۱۵. با کاربرد تابع fseek آشنایی بیابد.
۱۶. دستگاههای ورودی – خروجی استاندارد را بشناسد.

مقدمه

متغیرهای معمولی، آرایه‌ها و ساختمانها همگی در حافظهٔ RAM قرار دارند. لذا پس از خاموش شدن کامپیوتر یا خروج از برنامه داده‌هایی که در آنها ذخیره شده‌اند از بین می‌روند و برای استفادهٔ مجدد باید دوباره آنها را وارد کرد که قطعاً این کار مقرون به صرفه نیست، زیرا نه‌تنها مستلزم صرف وقت زیادی است، بلکه حوصلهٔ انجام کار را نیز از برنامه‌نویس سلب می‌کند. برای رفع این مشکل از نوعی ساختمان دادهٔ دیگر به نام فایل استفاده می‌شود. این نوع ساختمان داده روی حافظهٔ جانبی مثل دیسک، نوار و جزآن تشکیل می‌گردد. چون اطلاعات موجود در روی حافظهٔ جانبی با قطع جریان برق، قطع اجرای برنامه 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 کلیهٔ دانشجویان دانشگاه. داده‌های مربوط به هر یک از اجزای فایل رکورد نام دارد. برای مثال، در دانشگاهی 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 بپردازیم ، مشاهده می‌کنیم که هر دانشجو ممکن است چند قلم داده داشته باشد، مثل نام دانشجو، تعداد واحدهایی که گذرانده، نمرهٔ هر درس و جز آن. به هریک از اجزای یک رکورد فیلد گویند. لذا می‌توان گفت که هر رکورد مجموعه‌ای از چند فیلد است.

در زبان C فایل داده ممکن است هر دستگاهی مثل صفحهٔ نمایش، صفحه کلید، چاپگر، ترمینال، دیسک، نوار و جز آن باشد. داده‌ها ممکن است به چهار روش در فایل ذخیره و سپس بازیابی شوند:

– داده‌ها کاراکتر به کاراکتر در فایل نوشته و سپس کاراکتر به کاراکتر از فایل خوانده

شوند.

– داده‌ها به صورت رشته‌ای از کاراکترها در فایل نوشته شوند و سپس به صورت رشته‌ای از کاراکترها در دسترس قرار گیرند.

– داده‌ها در حین نوشتن بر روی فایل با فرمت خاصی نوشته و سپس با همان فرمت خوانده شوند.

– داده‌ها به شکل ساختمان (رکورد) روی فایل نوشته و سپس به صورت ساختمان از فایل خوانده شوند.

برای هریک از موارد فوق توابع خاصی در زبان C منظور شده‌اند که در این فصل بررسی می‌کنیم.

انواع فایل

داده‌ها ممکن است در فایل به دو صورت متنی و باینری وجود داشته باشند. این دو روش ذخیره شدن داده‌ها در موارد زیر با یکدیگر تفاوت دارند.

– تعیین انتهای خط؛

– تعیین انتهای فایل؛

– نحوهٔ ذخیره شدن اعداد روی دیسک.

در فایل متنی اعداد به صورت رشته‌ای از کاراکترها ذخیره می‌شوند، ولی در فایل باینری اعداد با همان صورتی که در حافظه قرار می‌گیرند روی دیسک ذخیره می‌شوند. برای مثال، در فایل متنی عدد ۵۲۶، سه بایت را اشغال می‌کند، زیرا هر رقم آن به صورت کاراکتر در نظر گرفته می‌شود، ولی در فایل باینری این عدد در ۲ بایت ذخیره می‌گردد (چون عدد ۵۲۶ عدد صحیح است و اعداد صحیح در حافظهٔ کامپیوتر در دو بایت ذخیره می‌شوند).

در فایل متنی، کاراکتری که پایان خط را مشخص می‌کند، در حین ذخیره شدن روی دیسک باید به کاراکترهای CR/LF، carriage،line feed تبدیل شود و در حین خوانده شدن عکس این عمل باید صورت گیرد؛ یعنی کاراکترهای CR/LF باید به کاراکتر تعیین‌کنندهٔ پایان خط تبدیل شوند. بدیهی است که این تبدیلها 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 را سیستم نگهداری می‌کند و انتهای فایل با توجه به این طول مشخص می‌گردد. در حالت متنی کاراکتر 1A (در مبنای ١٦) یا ٢٦ (در مبنای ١٠) مشخص‌کنندهٔ انتهای فایل است (کلید Ctrl + z). 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 ١٦) یا ٢٦ (در مبنای ١٠) جزئی از اطلاعات باشند و بیانگر انتهای فایل نباشند. لذا نحوهٔ تشخیص انتهای فایل در فایل باینری با فایل متنی متفاوت است.

از نظر نحوهٔ ذخیره و بازیابی داده‌ها در فایل دو روش وجود دارد:

- سازمان فایل ترتیبی
- سازمان فایل تصادفی.

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 در دسترس قرار می‌گیرند.

در سازمان فایل تصادفی، به هر رکورد یک شماره اختصاص می‌یابد. لذا اگر فایل دارای n رکورد باشد، رکوردها از ١ تا n شماره‌گذاری خواهند شد. وقتی که رکورد در یک فایل با سازمان تصادفی قرار گرفت، محل آن توسط 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.

بازکردن و بستن فایل

هر فایل قبل از اینکه بتواند مورد استفاده قرار گیرد باید باز گردد. مواردی که در حین بازکردن فایل مشخص می‌شوند عبارت‌اند از:

- نام فایل

- نوع فایل ازنظر ذخیرهٔ اطلاعات (متنی یا باینری)

- نوع فایل ازنظر ورودی ـ خروجی (آیا فایل فقط ورودی است، آیا فقط خروجی است یا هم ورودی است و هم خروجی).

یک فایل ممکن است طوری باز شود که فقط عمل نوشتن اطلاعات روی آن مجاز باشد. به چنین فایلی فایل خروجی گویند. اگر فایل طوری باز گردد که فقط عمل خواندن اطلاعات از آن امکان‌پذیر باشد به چنین فایلی فایل ورودی گویند. اگر فایل طوری باز شود که هم عمل نوشتن اطلاعات روی آن مجاز باشد و هم عمل خواندن اطلاعات از آن، به چنین فایلی فایل ورودی - خروجی گویند. اگر فایلی قبلاً وجود نداشته باشد، در حین بازشدن باید فایل خروجی باز شود. اگر فایلی قبلاً وجود داشته باشد و به‌عنوان خروجی بازگردد، اطلاعات قبلی آن از بین می‌رود.

تابع fopen برای باز کردن فایل مورد استفاده قرار می‌گیرد و دارای الگوی زیر است.

```
FILE *fopen (char *filename , *mode)
```

در این الگو کلمهٔ کلیدی FILE با حروف بزرگ نوشته می‌شود. 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 قسمت نام و انشعاب است که بهتر است انشعاب فایل داده dat انتخاب گردد. محل تشکیل یا وجود فایل شامل نام درایو و یا هر مسیر موجود روی دیسک است. mode مشخص می‌کند که فایل چگونه باید باز شود (ورودی، خروجی یا...). مقادیری که می‌توانند به جای mode در تابع fopen قرار گیرند، همراه با مفاهیم آنها در جدول ۱۰ـ۱ آمده است.

برای باز کردن فایل باید اشاره‌گری از نوع فایل تعریف کرد تا به فایلی که با تابع fopen باز می‌شود اشاره کند. اگر فایل به دلایلی باز نشود، این اشاره‌گر برابر با NULL خواهد بود.

ـ **مثال ۱۰ـ۱** دستورهای زیر را درنظر بگیرید.

```
FILE *fp ;
fp = fopen ("A:  test" , "w") ;
```

دستور اول متغیر fp را از نوع اشاره‌گر فایل تعریف می‌کند و دستور دوم فایلی به نام test را در درایو A ایجاد می‌کند چون حالت "w" فایل را به صورت خروجی باز می‌کند.

جدول ۱۰ـ۱ مقادیر معتبر mode در تابع fopen()

mode	مفهوم
r (rt)	فایلی از نوع متنی را به عنوان ورودی باز می‌کند.
w (wt)	فایلی از نوع متنی را به عنوان خروجی باز می‌کند.
a (at)	فایل را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه کرد.
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 اطلاعاتی را به انتهای آن اضافه کرد.
r + (r+t)	فایل موجود از نوع متنی را به عنوان ورودی و خروجی باز می‌کند.
w + (w+t)	فایلی از نوع متنی را به عنوان ورودی و خروجی باز می‌کند.
a + (a+t)	فایل موجود از نوع متنی را به عنوان ورودی و خروجی باز می‌کند.
r + b	فایل موجود از نوع باینری را به عنوان ورودی و خروجی باز می‌کند.
a + b	فایل احتمالاً 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.

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 ("cannot open file \ n") ;
    exit (0) ;
  }
```

پس از اینکه برنامه‌نویس کارش را با فایل تمام کرد، باید آن را ببندد. بستن فایل با تابع fclose انجام می‌شود که دارای الگوی زیر است.

```
int fclose (FILE *fp)
```

در این الگو fp به فایلی اشاره می‌کند که باید با تابع fclose بسته شود. به عنوان مثال دستور ; (fclose (p موجب بستن فایلی می‌شود که p به آن اشاره می‌کند.

توابع putc و getc

برای نوشتن کاراکتر در فایلی که قبلاً باز شده است، از توابع putc و fputc استفاده می‌شود. طریقهٔ استفاده از این دو تابع یکسان است. تابعputc در نسخه‌های جدید C و نیز fputc در نسخه‌های قدیمی C وجود داشته است. چون تابع putc به صورت ماکرو تعریف شده است، سرعت عمل آن بالاست. الگوی تابع putc به صورت زیر است.

```
int putc (int ch , FILE *fp)
```

در این الگو، ch کاراکتری است که باید در فایل نوشته شود و fp اشاره‌گری از نوع فایل است که مشخص می‌کند کاراکتر مورد نظر باید در چه فایلی نوشته شود.

برای خواندن کاراکترها از فایل می‌توان از دو تابع getc و fgetc استفاده کرد. نحوهٔ به کارگیری این دو تابع یکسان است. تابع fgetc در گونه‌های قدیمی C و نیز getc در گونه‌های جدید C وجود دارد. چون تابعgetc به صورت ماکرو پیاده‌سازی شده است، از سرعت بیشتری برخوردار است. الگوی این تابع به صورت زیر است.

```
int getc (FILE *fp)
```

در این الگو، fp اشاره‌گری است که مشخص می‌کند کاراکتر مورد نظر از کدام فایل باید خوانده شود. در مورد خواندن و نوشتن داده‌ها روی فایل باید به چند نکته توجه داشت.

اول اینکه، وقتی کاراکترهایی روی فایل نوشته می‌شوند باید مکان بعدی‌ای که کاراکتر بعدی 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 برآوردن این هدف، سیستم از متغیری به نام موقعیت‌سنج فایل استفاده می‌کند که با هر دستور خواندن یا نوشتن روی فایل مقدار این متغیر به طور خودکار تغییر می‌کند تا موقعیت فعلی فایل را مشخص نماید. لذا عمل نوشتن روی فایل و عمل خواندن از روی آن از جایی شروع می‌شود که این متغیر نشان می‌دهد.

در هنگام خواندن داده‌ها از فایل باید بتوان انتهای فایل را بررسی کرد؛ یعنی در برنامه باید بتوان این تست را انجام داد که اگر در حین خواندن داده‌ها از فایل موقعیت‌سنج فایل به انتهای فایل رسید دستور خواندن بعدی صادر نگردد، چرا که 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 به عمل خواندن ادامه داد تا اینکه کاراکتر خوانده شده برابر با EOF گردد. در فایل باینری برای تست کردن انتهای فایل از تابع feof استفاده می‌گردد. الگوی این تابع به‌صورت زیر است.

int feof (FILE *fp)

در این الگو fp اشاره‌گری است که مشخص می‌کند این تابع باید روی چه فایلی عمل کند. تابع fopen علاوه بر تشخیص انتهای فایلهای باینری برای تشخیص انتهای فایلهای متنی نیز استفاده می‌شود.

ـ **مثال ۱۰ـ۲** برنامهٔ زیر کاراکترهایی را از ورودی می‌خواند و در فایل متنی قرار می‌دهد. سپس داده‌های موجود در این فایل را می‌خواند و به فایل دیگری منتقل می‌کند. آخرین کاراکتر ورودی نقطه در نظر گرفته شده است.

```
# include <stdio. h>
# include <stdlib. h>
void main (void)
  {
    FILE *in , *out ;
    char ch ;
    in = fopen ("F1.txt" , "w") ;
    if (in = = NULL)
      { printf ("cannot open F1.txt \n") ;
        exit(1) ;
      }
     do {
           ch = getchar() ;
           putc (ch , in) ;
     }while (ch ! = '.') ;
     fclose(in) ;
     out = fopen ("F2.txt" , "w") ;
     if (out = = NULL)
       { printf ("cannot open F2.txt ") ;
         exit(1) ;
       }
     int = fopen ("F1.txt " , "r") ;
     if (in = = NULL)
        { printf ("can not open F1.txt ") ;
          exit(1) ;
```

```
    }
  ch = getc(in) ;
  while (ch! = EOF)
    { putc(ch , out) ;
      ch = getc (in) ;
    }
  fclose(in) ;
  fclose(out) ;
}
```

ـ **مثال ۱۰ـ۳** برنامه‌ای بنویسید که کاراکترهایی را از صفحه کلید بگیرد و در فایل باینری قرار دهد. سپس کاراکترهای موجود در این فایل را بخواند و به فایل باینری دیگر منتقل کند. اسامی فایلهای ورودی و خروجی به عنوان آرگومان تابع اصلی به برنامه وارد می‌شوند.

```
# include <stdio. h>
# include <stdlib. h>
void main (int argc , char *argv[ ])
  {
    FILE *in , out ;
    char ch ;
    clrscr() ;
    if (argc!=3)
      { printf ("you forget enter file name \n ") ;
        exit(1) ;
      }
    in = fopen (argv[1] , "wb") ;
    if (in = = NULL)
      { printf ("cannot open (first) output file\n ");
        exit (1) ;
      }
    do { ch = getchar() ;
         putc(ch , in) ;
       } while (ch ! = '.') ;
    fclose(in) ;
    in = fopen (argv[1] , "rb") ;
    if (in = = NULL)
      { printf ("cannot open input file \n") ;
        exit(1) ;
      }
    out = fopen (argv[2] , "wb") ;
    if (out = = NULL)
      { printf ("cannot open output file \n ") ;
```

```
        exit(1) ;
      }
    ch = getc(in) ;
    while (!feof (in))
      {
        putc(ch , out) ;
        ch = getc(in) ;
      }
    fclose(in) ;
    fclose(out) ;
  }
```

توابع putw و getw

این دو تابع مشابه getc و putc اند، با این تفاوت که برای خواندن و نوشتن مقادیر صحیح از یک فایل به یک فایل دیگر به کار می‌روند. برای مثال دستور putw(50 , fp) ; عدد صحیح ۵۰ را در فایلی که fp به آن اشاره می‌کند می‌نویسد.

ـ **مثال ۱۰ـ٤** برنامهٔ زیر مقادیر صحیح را از فایلی می‌خواند و مجموع آنها را در خروجی چاپ می‌کند.

```
#include<stdio.h>
#include<stdlib.h>
main()
  {
    FILE *fp ;
    int sum ;
    if (fp = fopen("sample" , "r") = = NULL)
      {
        printf(" can not open this file \n") ;
        exit(1) ;
      }
    while (!feof(fp))
      sum = sum + getw(fp) ;
    printf(" sum = %d" , sum) ;
    fclose(fp) ;
  }
```

در اینجا تابع getw مقادیر صحیح را از فایل sample می‌خواند و شاخص موقعیت فایل را پیش می‌برد. برای مشخص ساختن اینکه به پایان فایل رسیده است یا نه، درون حلقه از

دستور feof استفاده شده است.

توابع fputs و fgets

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 زیرند.

```
int fputs (const char *str , FILE *fp)
char *fgets (char *str , int length , FILE *fp)
```

در الگوهای فوق، fp اشاره‌گری است که مشخص می‌کند این توابع باید 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 اشاره می‌کند که اطلاعات خوانده شده از فایل در آن قرار می‌گیرند. length طول رشته‌ای را که باید از فایل خوانده شود مشخص می‌کند. نحوهٔ عمل تابع fgets به‌این صورت است که از ابتدای فایل شروع به خواندن می‌کند تا به انتهای خط برسد 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.

ـ **مثال ۱۰ـ۵** برنامهٔ زیر رشته‌هایی را از ورودی (صفحه کلید) می‌خواند و در فایل قرار می‌دهد. از آنجایی که تابع gets کاراکتری که پایان خط را مشخص می‌کند به رشته اضافه نمی‌کند، در حین نوشتن روی فایل این کاراکتر به رشتهٔ خوانده شده اضافه می‌شود. برای خاتمهٔ برنامه کافی است به جای رشته فقط کلید enter وارد شود.

```
# include "stdio. h"
# include "stdlib. h"
void main (void)
  {
    FILE *fp ;
    char str [80] ;
    if ((fp = fopen ("test" , "w")) = = NULL)
       { printf ("cannot open file \n") ;
         exit(1) ;
       }
    printf ("enter a string") ;
```

```
    printf ("ENTER to quit. \n") ;
    while(1)
      { gets (str) ;
        if (str[0]) break ;
          strcat (str , "\n") ;
        fputs (str , fp) ;
      }
    fclose (fp) ;
  }
```

فایل وسیلهٔ ورودی ـ خروجی

می‌توان 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 برای باز کردن فایل متنی موجود به عنوان ورودی و خروجی استفاده کرد. از یکی از عبارات w+ یا w + t برای ایجاد فایل متنی به عنوان ورودی و خروجی استفاده کرد. و نیز از یکی از عبارات a+ یا a + t برای ایجاد فایل متنی یا باز کردن فایل متنی موجود، به عنوان ورودی و خروجی استفاده کرد.

همچنین از عبارت r + b برای باز کردن فایل باینری موجود، به عنوان ورودی و خروجی استفاده کرد. از عبارت w + b برای ایجاد فایل باینری به عنوان ورودی و خروجی استفاده کرد. از عبارت a + b برای ایجاد یا بازکردن فایل موجود باینری به عنوان ورودی و خروجی استفاده کرد.

ـ **مثال ۱۰ـ٦** دستورهای زیر را در نظر بگیرید.

```
fp1 = fopen ("test. dat" , "w+b") ;
fp2 = fopen ("sample. dat" , "r+b") ;
fp3 = fopen ("test2. dat" , "a+t") ;
```

دستور اول، 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 این فایل قبلاً وجود داشته باشد، محتویات قبلی آن از بین خواهند رفت.

دستور دوم، فایلی به نام sample. dat را که اکنون در درایو جاری وجود دارد از نوع باینری و به‌صورت ورودی و خروجی باز می‌کند. اگر این فایل بر روی درایو جاری وجود

نداشته باشد، پیام خطایی صادر خواهد شد.

دستور سوم، فایلی به نام test 2. 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 خواهد ماند و اطلاعات جدید به انتهای آن اضافه خواهد شد.

باتوجه به مطالبی که تاکنون در مورد فایلها گفتیم ، در حین کار با فایلها (نوشتن اطلاعات بر روی آنها و یا خواندن اطلاعات از آنها) برای برگشت به ابتدای فایل (تغییر موقعیت‌سنج فایل طوری که به ابتدای فایل اشاره کند) باید فایل را بست و مجدداً آن را باز کرد. اصولاً شاید در فایلهایی که فقط به عنوان خروجی یا فقط به عنوان ورودی باز می‌شوند، نیاز به برگشت به ابتدای فایل (بدون بستن و باز کردن مجدد آن) احساس نشود، ولی این امر در مورد فایلهای ورودی و خروجی ضروری است. برای این منظور از تابعی به نام rewind استفاده می‌گردد. الگوی این تابع در فایل stdio.h قرار دارد و به صورت زیر است.

void rewind (FILE *fp)

در این الگو fp به فایلی اشاره می‌کند که موقعیت‌سنج آن باید به ابتدای فایل اشاره کند.

ـ **مثال ۱۰ـ۷** برنامهٔ زیر رشته‌هایی را از ورودی می‌خواند و در فایل test قرار می‌دهد. سپس محتویات این فایل را می‌خواند و به صفحهٔ نمایش منتقل می‌کند.

```
# include <stdio.h>
# include <stdlib.h>
void main (void)
  {
    FILE *fp ;
    char str [80] ;
    if ((fp=fopen ("test" , "w+")) = NULL)
       {
         printf ("cannot open file\n") ;
         exit (1) ;
       }
     printf ("enter a string") ;
     printf ("Enter to quit \n") ;
     while (1)
       { gets (str) ;
         if (!str [0]) break ;
           strcat (str , "\n") ;
```

```
        fputs (str , fp) ;
      }
   printf ("\n the content of file is:  \n ") ;
   rewind (fp) ;
   fgets (str , 79 , fp) ;
   while (!feof (fp))
      { printf ("%s" , str) ;
        fclose (str , 79 , fp) ;
      }
    fclose (fp) ;
 }
```

تابع ferror

در حین انجام کار با فایلها ممکن است خطایی رخ دهد. برای مثال، عدم وجود فضای کافی برای ایجاد فایل، آماده نبودن دستگاهی که فایل باید در آنجا تشکیل گردد یا مواردی از این قبیل منجر به بروز خطا می‌شوند. با استفاده از تابع ferror می‌توان از بروز چنین خطایی مطلع شد. الگوی تابع ferror در فایل stdio.h قرار دارد و به صورت زیر است.

```
int ferror (FILE *fp)
```

در الگوی فوق fp اشاره‌گری است که مشخص می‌کند این تابع باید روی چه فایلی عمل کند. این تابع تابعی منطقی است؛ بدین معنی که اگر خطایی در رابطه با فایلها رخ داده باشد این تابع ارزش درست و در غیر این صورت ارزش نادرست را برمی‌گرداند. برای تشخیص خطا در کار با فایلها، بلافاصله پس از هر عملی که روی فایل انجام می‌شود باید از این تابع استفاده کرد.

ـ **مثال ۱۰_۸** برنامهٔ زیر کاراکترهای tab را از فایل حذف می‌کند و به جای آن به تعداد کافی فضای خالی یا blank قرار می‌هد. اسامی فایلهای ورودی و خروجی از طریق آرگومان به برنامه وارد می‌شود.

```
# include "stdio. h"
# include "stdlib. h"
# define TAB_SIZE 8
# define OUT 1
# define IN 1
void err (int) ;
```

```
void main (int argc , char *argv[ ])
  {
    FILE *in , *out ;
    int tab , i ;
    char ch ;
    if (argc! = 3)
       {
         printf ("\n incorrect number of parameters ") ;
         printf ("\n\t press any key …") ;
         getch () ;
         exit (1) ;
       }
    in = fopen (argv[2] , "wb") ;
    if (in = = NULL)
      {
        printf ("\n cannot open output file ") ;
        printf ("\n\t press a key …") ;
        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) ;
              }
              tab = 0 ;
            }
     else
       { putc(ch , out) ;
          if (ferror (out))
            err (OUT) ;
          tab + + ;
          if (tab = = TAB_SIAE || ch = = '\n' || ch = = '\r')
            tab = 0 ;
       }
    }while (!feof (in)) ;
    fclose (in) ;
    fclose (out) ;
  }
```

```
void err (int error)
  {
    if (erro = = IN)
      printf ("\n error on input file. ")
    else
      printf ("\n press any key …") ;
    getch () ;
    exit (1) ;
  }
```

تابع remove

برای حذف فایلهای غیرضروری می‌توان از تابع remove استفاده کرد. الگوی این تابع در فایل stdio.h قرار دارد و به صورت زیر است.

```
int remove (char *filename)
```

در این الگو filename به نام فایلی که باید حذف شود اشاره می‌کند. اگر عمل تابع با موفقیت انجام شود، مقدار صفر و در غیر این صورت مقداری غیر از صفر برگردانده خواهد شد.

ـ **مثال ۱۰_۹** برنامهٔ زیر نام فایلی را به عنوان آرگومان می‌پذیرد و آن را حذف می‌کند.

```
# include "stdio. h"
# include "stdlib. h"
# include "ctype. h"
main (int argc , char *argv[ ])
  {
    char str [80] ;
    if (argc!=2)
       {
         printf ("\n you must type a file name \n") ;
         exit (1) ;
       }
    printf ("Delete %s (y/n):  " , argv[1]);
    gets (str) ;
    if (toupper (*str) = = 'y')
       if (remove (argv[1]))
          {
            printf ("cannot delete file \n") ;
            exit (1) ;
```

```
        }
  }
```

ملاحظه می‌کنید که در این برنامه، برای حذف فایل مورد نظر از تابع remove استفاده شده است.

تابع fprintf , fscanf

اگر لازم باشد که داده‌ها با فرمت خاصی در فایل نوشته یا از آن خوانده شوند می‌توان از دو تابع fprintf و fscanf استفاده کرد. این دو تابع دقیقاً کار توابع printf و scanf را در ورودی – خروجی معمولی (غیر از فایل) 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 این الگو fp اشاره‌گری است که مشخص می‌کند اعمال این توابع باید روی چه فایلی انجام شود. control_string مشخص می‌کند که داده‌ها یا args باید با چه فرمتی نوشته یا خوانده شوند.

ـ **مثال ۱۰_۱۰** برنامهٔ زیر یک رشته و یک عدد صحیح را از ورودی می‌خواند و آن را در فایل می‌نویسد. سپس از این فایل می‌خواند و در صفحه نمایش چاپ می‌کند.

```
# include <stdio. h>
# include <stdlib. h>
# include <io. h>
void main (void)
  {
    FILE *fp ;
    char str[80] , number [10] ;
    int t ;
    if ((fp = fopen ("test" , "w")) = = NULL)
      {
        printf ("cannot open file\n") ;
        exit (1) ;
      }
    printf ("\n enter string:  ") ;
    gets (str) ;
    strcat (str , "\n") ;
```

```
    printf ("\n enter a number:  ") ;
    gets (number) ;
    t = atoi (number) ;
    fprintf (fp , "%s%d , str , t) ;
    fclose (fp) ;
    if ((fp = fopen ("test" , "r")) = = NULL)
       {
        printf ("cannot open file \n") ;
        exit (1) ;
       }
    fscanf (fp , "%s%d" , &str , &t) ;
    printf ("\nstring = %s , digit = %d" , str , t) ;
  }
```

در مورد توابع fprintf و fscanf باید توجه داشت که علی‌رغم اینکه ورودی ـ خروجی با این دو تابع آسان است، اطلاعات به همان صورتی که در صفحه نمایش ظاهر می‌شوند در فایل ذخیره می‌گردند. برای مثال، عدد ۲۶۷ که در صفحهٔ نمایش ۳ بایت را اشغال می‌کند، اگر با تابع fprintf روی فایل نوشته شود نیز ۳ بایت را اشغال خواهد کرد (توجه داریم که عدد ۲۶۷ عدد صحیح است و در دو بایت ذخیره می‌شود). این بدین معنی است که هر رقم به صورت کاراکتر تلقی می‌گردد. اگر این عدد با تابع fscanf از روی فایل خوانده شود، باید عمل تبدیل کاراکتر به عدد صورت گیرد که مستلزم صرف وقت است. برای جلوگیری از بروز این مشکل از دو تابع fread و fwrite که در ادامه بررسی خواهند شد استفاده می‌شود.

توابع fwrite و fread

توابع متعددی 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 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 کرد که از سرعت بالایی برخوردارند. الگوی این تابع در فایل stdio.h قرار دارد و به صورتهای زیرند.

```
int fread (void *buffer , int num_byte , int count , FILE *fp)
int fwrite (void *buffer , int num_byte , int count , FILE *fp)
```

در این دو الگو پارامتر buffer در مورد تابع fread 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 که باید خوانده یا نوشته شود را مشخص می‌کند. پارامتر count تعداد عناصری است که طول آن با num_byte مشخص گردید و باید در فایل نوشته یا از فایل خوانده شوند. اشاره‌گر fp به فایلی اشاره می‌کند که توابع fread و fwrite باید روی آنها عمل کنند.

ـ **مثال ۱۰_۱۱** مجموعه دستورهای زیر را درنظر بگیرید.

```
char student [20] ;
char str [10] ;
fwrite (student , sizeof (char) , 20 , fp) ;
fread (str , sizeof (char) , 10 , fp) ;
```

دستور اول و دوم رشته‌هایی به طولهای ۲۰ و ۱۰ را تعریف می‌کنند. دستور سوم، تعداد ۲۰ بایت از اطلاعات موجود در آرایهٔ student را در فایلی که fp به آن اشاره می‌کند می‌نویسد. دستور چهارم تعداد ۱۰ بایت از اطلاعات را از فایلی که fp به آن اشاره می‌کند می‌خواند و در متغیر str قرار می‌دهد. توابع fread و fwrite بیشتر در ورودی - خروجی رکورد استفاده می‌شوند.

ـ **مثال ۱۰_۱۲** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
main()
  {
    FILE *fp ;
    float x = 3.14 ;
    if ((fp = fopen("f1" , "wb")) = = NULL)
      {
        printf("can not open file \n") ;
        return ;
      }
    fwrite(&x , sizeof(float) , 1 , fp) ;
    fclose(fp) ;
  }
```

این برنامه با استفاده از تابع fwrite متغیر float را در فایل می‌نویسد. در اینجا بافر تابع fwrite متغیری ساده است.

تابع fseek

برای خواندن و نوشتن داده‌ها به صورت تصادفی از این تابع استفاده می‌شود. این تابع اجازه می‌دهد که برنامه‌نویس روی اشاره‌گر موقعیت فایل، کنترل داشته باشد. از این رو با استفاده از این تابع، برای دستیابی به رکوردی از فایل، اشاره‌گر موقعیت فایل را به ابتدای رکورد مورد نظر انتقال می‌دهیم.

الگوی این تابع به صورت زیر است.

```
int fseek(FILE *fp , long int num_bytes , int origin) ;
```

در این الگو fp اشاره‌گر فایل است. پارامتر دوم، تعداد بایتهای مورد جستجو از مبدأ را مشخص می‌کند و پارامتر سوم یا origin محل جستجو در فایل را مشخص می‌کند که ممکن است یکی از ماکروهای زیر باشد.

مبدأ	نام ماکرو	مقدار ماکرو
شروع از ابتدای فایل	SEEK_SET	0
از موقعیت جاری	SEEK_CUR	1
انتهای فایل	SEEK_END	2

کاربرد این تابع در مورد فایلهای باینری است، زیرا ترجمهٔ کاراکترها در فایلهای متنی موجب بروز اشتباه در مکانها می‌شود.

ـ **مثال ۱۰_۱۳** تابع زیر بایت شمارهٔ ٥٤ از فایلی به نام sample را می‌خواند.

```
readByte()
  {
    FILE *fp ;
    if ((fp = fopen("sample" , "rb")) = = NULL)
      {
        printf(" can not open this file \n") ;
```

```
        exit(1) ;
      }
    fseek(fp , 54L , 0) ;
    return getc(fp) ;
  }
```

این تابع اگر با موفقیت عمل کند مقدار صفر را برمی‌گرداند در غیر این صورت مقداری غیر از صفر را برمی‌گرداند. همان طور که ملاحظه می‌کنید از توصیف‌کنندهٔ L برای نشان دادن مقدار long int استفاده شده است. دستور fseek اشاره‌گر فایل را در بایت شمارهٔ ۵۴ قرار می‌دهد. سپس در دستور خط بعد کاراکتر موجود در این محل از فایل با دستور getc خوانده و به تابع فراخواننده بازگردانده می‌شود. مقدار صفر استفاده شده در دستور fseek نشان‌دهندهٔ ماکروی FSEEK_SET است.

دستگاههای ورودی ـ خروجی استاندارد

وقتی اجرای برنامه به زبان C آغاز می‌شود، پنج فایل به طور خودکار باز می‌شوند. اشاره‌گرهای آنها را در جدول ۱۰_۲ مشاهده می‌کنید.

جدول ۱۰_۲ دستگاههای ورودی - خروجی استاندارد

اشاره‌گر فایل	نام دستگاه (فایل)
stdin	دستگاه ورودی استاندارد (صفحه کلید)
stdout	دستگاه خروجی استاندارد (صفحه نمایش)
stderr	دستگاه استاندارد جهت ثبت پیامهای خطا (صفحه نمایش)
stdprn	دستگاه استاندارد چاپ (چاپگر موازی)
stdaux	پورت سری (serial port)

ـ **مثال ۱۰_۱۴** مجموعه دستورهای زیر را در نظر بگیرید.

```
putc (ch , stdout) ;
printf (stdout , "%d , %d" , a , b) ;
fscanf (stdin , "%d , %d" , &x , &y) ;
```

دستور اول موجب می‌شود تا کاراکتر ch در صفحه نمایش نوشته شود. دستور دوم

موجب می‌شود تا متغیرهای a و b در صفحه نمایش نوشته شوند. دستور سوم موجب می‌شود تا متغیرهای x و y از صفحه کلید خوانده شوند.

دستگاههای استاندارد ورودی ـ خروجی همان طور که به طور خودکار باز می‌شوند، به طور خودکار نیز بسته خواهند شد و لازم نیست برنامه‌نویس آنها را ببندد.

خودآزمایی ۱۰

۱. برنامه‌ای بنویسید که عددی از ورودی بخواند و فاکتوریل آن را در فایل بنویسد.

۲. برنامه‌ای بنویسید که رشته‌ای از ورودی دریافت کند، سپس رشتهٔ ورودی را به همراه معکوس آن رشته در فایلی درج کند.

۳. برنامه‌ای بنویسید که فایلی متنی به حجم ۱۰ بایت ایجاد کند.

٤. برنامه‌ای بنویسید که برنامهٔ موجود درفایل دیگری را فایل ورودی بپذیرد و تعداد پرانتزهای باز و بسته و همچنین تعداد آکولادهای باز و بستهٔ آن را شمارش کند. نام فایل ورودی به عنوان آرگومان تابع اصلی به برنامه وارد شود.

۵. برنامه‌ای بنویسید که مشخصات شغلی کارمندان سازمان را، که شامل نام، تعداد ساعت کار و کارمزد ساعتی است، دریافت کند و در فایل قرار دهد. سپس با استفاده از این اطلاعات، حقوق دریافتی آنها را محاسبه و چاپ کند.

٦. برنامه‌ای بنویسید که بتواند از فایل دلخواهی کپی تهیه کند.

فصل ۱۱

مطالب تکمیلی

هدف کلی

آشنایی با برخی مطالب تکمیلی در زبان برنامه‌نویسی C

هدفهای رفتاری

از دانشجو انتظار می‌رود پس از مطالعهٔ این فصل،

۱. ماکرو و تفاوت آن را با تابع بشناسد.

۲. با کدهای توسعه‌یافته آشنایی یابد.

۳. مراحل اجرای برنامه را بشناسد.

۴. با نحوهٔ نصب نرم‌افزار توربو C آشنایی یابد.

۵. با منوهای File، Run، Window، و Help آشنایی یابد.

۶. مراحل ایجاد برنامه در IDE را بداند.

۷. توابع کتابخانه‌ای ()clrscr، توابع تبدیل نوع، توابع ریاضی، توابع کاراکتری، توابع رشته‌ای، توابع تخصیص حافظهٔ پویا، و توابع گرافیکی را بشناسد.

ماکرو

در زبان C می‌توان برای تعریف ثابتهای سمبولیک از دستور #define استفاده کرد. در ابتدای زمان کامپایل، این ثابتهای سمبولیک با نمادهای معادلشان جایگزین می‌شوند. بنابراین ثابتهای سمبولیک شکلی از خلاصه‌نویسی را در اختیار برنامه‌نویس قرار می‌دهند که موجب ساده‌تر

شدن ساختمان برنامه می‌گردد.

اما از دستور #define می‌توان برای اموری به غیر از تعریف سادهٔ ثابتهای سمبولیک نیز استفاده کرد و آن تعریف ماکروهاست. ماکرو شناسه‌ای است که معادل با عبارت، دستور یا گروهی از دستورهای تعریف شده باشد. در صورتی که ماکرو شامل گروهی از دستورها باشد، مشابه تابع عمل می‌کند. اما هم روش تعریف کردن ماکروها با توابع فرق دارد و هم آنکه در حین عمل کامپایل به گونه‌ای متفاوت رفتار می‌کند.

ـ **مثال ۱۱_۱** برنامهٔ زیر را در نظر بگیرید.

```
#include<stdio.h>
#define area length*width
main()
  {
    int length , width ;
    scanf("%d%d" , length, width) ;
    printf("area = %d" , area) ;
  }
```

این برنامه ماکرویی به نام area دارد که عبارت length*width (حاصل ضرب طول و عرض) را بیان می‌کند. هنگامی که این برنامه در حال کامپایل شدن باشد، در دستور printf عبارت length*width جایگزین شناسهٔ area می‌گردد به طوری که دستور printf به شکل زیر در خواهد آمد.

```
printf("area = %d" , length*width) ;
```

پس از اجرای برنامه خروجی مورد نظر چاپ خواهد شد.

ـ

به شکل متداول، تعریف ماکروها در ابتدای برنامه و پیش از تعریف اولین تابع قرار داده می‌شود. حوزهٔ تعریف ماکرو از محل تعریف شدن تا انتهای آن فایل است. ماکرویی که در فایل تعریف شده باشد در فایل دیگری قابل شناسایی نیست.

ماکروهای چند خطی را می‌توان با آوردن کاراکتر \ در انتهای هر خط تعریف کرد. این ویژگی موجب می‌شود که بتوان با ماکرویی واحد دستوری مرکب را بیان کرد.

ـ **مثال ۱۱_۲** به برنامهٔ زیر توجه کنید.

```
#include<stdio.h>
#define multiply        for(i = 1 ; i < n ; i++) \
```

```
                    for(j = 1 ; j < n ; j++) \
                      printf("%d" , i*j) ;
main()
  {
    int i , j , n ;
    scanf("%d" , n) ;
    multiply
  }
```

همان طور که ملاحظه می‌کنید این برنامه شامل ماکرویی چندخطی است به نام multiply که بیانگر دستور مرکب است. این دستور مرکب از دو حلقهٔ for و یک دستور printf تشکیل شده. در انتهای هر خط این ماکرو، به جز آخرین خط، علامت \ آورده شده است. همچنین دستور فراخوانی ماکرو سمیکولون ندارد. زمانی که این برنامه کامپایل شود، هر ارجاع به این ماکرو در حقیقت با دستورهایی که در تعریف آن آورده شده جایگزین می‌گردد؛ یعنی برنامهٔ بالا تبدیل به این برنامه می‌شود.

```
#include<stdio.h>
main()
  {
    int i , j , n ;
    scanf("%d" , n) ;
    for(i = 1 ; i < n ; i++)
      for(j = 1 ; j < n ; j++)
    printf("%d" , i*j) ;
  }
```

در تعریف ماکرو می‌توان آرگومانهایی نیز قرار داد. این آرگومانها بین دو پرانتز جای داده می‌شود. پرانتز سمت چپ باید حتماً بلافاصله پس از نام ماکرو و بدون هیچ گونه فاصله‌ای آورده شود. هر ماکرویی که بدین صورت تعریف شود، حضور آن در هر جای برنامه همانند فراخوانی تابع خواهد بود.

ـ **مثال ۱۱_۳** برنامهٔ مثال ۱۱_۲ با در نظر گرفتن آرگومان در تعریف ماکرو به صورت زیر خواهد بود.

```
#include<stdio.h>
#define multiply(n)      for(i = 1 ; i < n ; i++) \
                           for(j = 1 ; j < n ; j++) \
                             printf("%d" , i*j) ;
```

```
main()
  {
    int i , j , n ;
    scanf("%d" , n) ;
    multiply(n)
  }
```

این برنامه آرگومان n را به گونه‌ای به ماکرو می‌فرستد که گویی آرگومان حقیقی برای فراخوانی تابع است.

گاهی اوقات در برنامه‌ها به جای توابع از ماکروها استفاده می‌شود. استفاده از ماکرو به جای تابع این فایده را دارد که در زمان صرف شده برای فراخوانی آن تابع صرفه‌جویی می‌کند. اگر برنامه‌ای دارای فراخوانیهای تکراری بسیاری برای تابع باشد، زمان ذخیره شده از این طریق قابل توجه خواهد بود.

از طرف دیگر هر جا که ارجاعی به ماکرو شده باشد، تعریف آن ماکرو جایگزین آن ارجاع خواهد شد. از این رو به حجم برنامه‌ای که دارای چند ارجاع به ماکرویی واحد باشد به شکل نامناسبی افزوده می‌شود. بنابراین در شرایطی قرار می‌گیریم که باید بین سرعت اجرا و اندازهٔ برنامهٔ کامپایل شده یکی را انتخاب کنیم. به عبارت دیگر ماکروها سرعت اجرای برنامه را افزایش می‌دهند و در عوض اندازهٔ برنامهٔ اجرایی را نیز بزرگ می‌کنند، اما استفاده از توابع گرچه سرعت اجرا را کاهش می‌دهد، از حجم برنامه نیز کم می‌کند. در هر حال، بهره گرفتن از ماکرو هنگامی سودبخش است که تعداد فراخوانیهای تابع نسبتاً کم باشد اما آن موارد پشت سر هم تکرار شوند.

در حالت کلی هنگام استفاده از ماکروها، توجه به نکات زیر ضروری است:

– هنگام ارسال آرگومانها به ماکرو، تعداد آنها کنترل می‌شود ولی نوع داده آنها کنترل نمی‌شود. در این مورد امکان خطایابی کمتری نسبت به فراخوانی تابع وجود دارد.

– شناسهٔ ماکرو آدرس‌پذیر نیست، بنابراین نمی‌توان از آن ماکرو با اشاره‌گر استفاده کرد؛ یعنی نمی‌توان شناسهٔ ماکرو را به عنوان آرگومان به تابعی فرستاد در حالی که می‌توان تابعی را به صورت آرگومان به تابعی دیگر ارسال داشت.

– همچنین ماکرو نمی‌تواند به شکل بازگشتی خود را فراخوانی کند، در حالی که تابع بازگشتی این ویژگی را دارد.

کدهای توسعه‌یافته

صفحه کلید برای حروف، اعداد و علایم مخصوص کدی یک بایتی تولید می‌کند. اما برای بسیاری از کلیدهای دیگر یا ترکیبی از چند کلید، کدی دو بایتی تولید می‌گردد که به آن کد توسعه‌یافته گویند. برای مثال، کلیدهای تابع F1 تا F10 و کلیدهای مکان نما را می‌توان نام برد. بایت اول کد توسعه‌یافته برابر با صفر و بایت دوم آن کد کلید است. وقتی که یکی از این کلیدها فشار داده شود، دو بایت از بافر صفحه کلید اشغال می‌شود. برای اینکه برنامه بتواند فشار دادن این کلیدها را تشخیص دهد باید بایت اول را بخواند و با صفر مقایسه کند (هیچ کلیدی وجود ندارد که کد صفر را تولید نماید). اگر برابر با صفر باشد، بایت بعدی کد کلید توسعه‌یافته است. در غیر این صورت کلید فشار داده شده کلیدی معمولی است. در جدول ۱۱_۱ برخی از کدهای توسعه‌یافته نشان داده شده است.

جدول ۱۱_۱ کدهای توسعه یافته

نام کلید	بایت دوم کد توسعه‌یافته
F1	59
F2	60
F3	61
F4	62
F5	63
F6	64
F7	65
F8	66
F9	67
F10	68
Home	71
Up arrrow	72
pgup	73
Left arrow	75
Right arrow	77
End	79
Down arrow	80
Pgdn	81
ins	82
del	83
shift + tab	15
ctrl + left arrow	115
ctrl + right arrow	116
ctrl + End	117
ctrl + pgdn	118
ctrl + Home	119
ctrl + pgup	132

ـ **مثال ۱۱ـ۴** برنامهٔ زیر کدی را از صفحه کلید می‌خواند و مشخص می‌کند که آیا کد توسعه‌یافته است یا نه.

```
main()
  {
    char key1 , key2 ;
    while ((key1=getch()) != '*')
      if (key1 = = 0)
        {
          key2 = getch() ;
          printf("%3d %3d \n" , key1 , key2) ;
        }
      else
    printf("%d \n" , key1) ;
  }
```

در این برنامه حلقهٔ while تا زمان استفاده از کلید * ادامه پیدا می‌کند. اگر کلید فشار داده شده دارای کد توسعه‌یافته باشد، بایت دوم نیز خوانده می‌شود و در متغیر key2 قرار می‌گیرد.

ـ

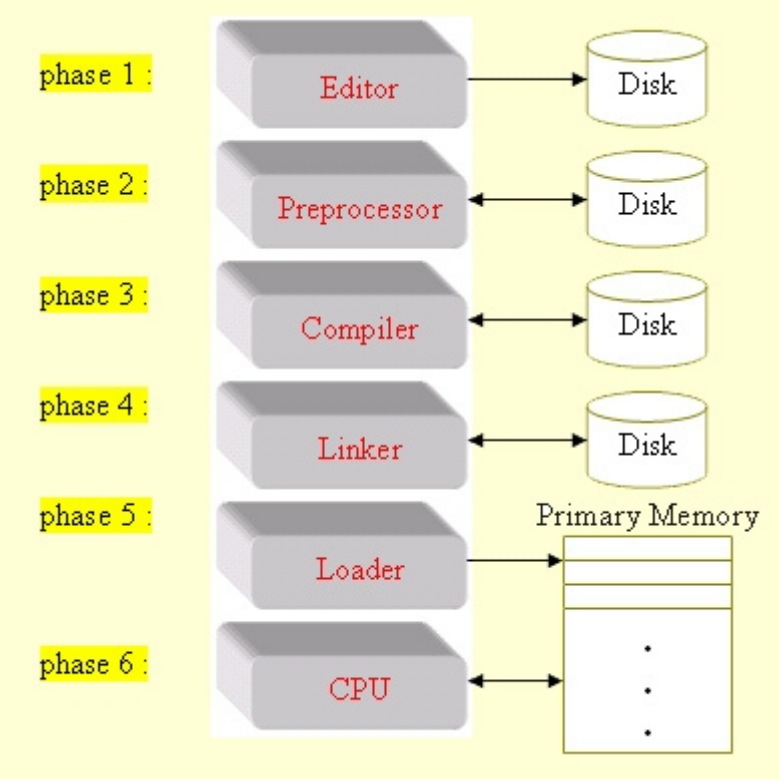
پیاده‌سازی و اجرای برنامه‌ها

برنامهٔ C، شش فاز یا مرحله را طی می‌کند تا به طور کامل اجرا شود و خروجی خود را ایجاد کند. نقش هر مرحله یا فاز در شکل ۱۱ـ۱ به اختصار بیان شده است.

در مرحلهٔ اول، یعنی ویرایش، به کمک ویرایشگر، برنامهٔ مورد نظر که برنامهٔ منبع[1] نامند تایپ می‌گردد و غلطهای دستوری آن به کمک ویراستار تصحیح می‌شود. نتیجهٔ نهایی به صورت فایل روی حافظهٔ جانبی که معمولاً دیسک است ذخیره می‌گردد. پسوند این فایل برحسب محیطهای مختلف برنامه‌سازی ممکن است c یا cpp باشد.

در مرحلهٔ دوم، فرمانی برای ترجمهٔ برنامه صادر می‌شود و کامپایلر، برنامهٔ مورد نظر را به زبان ماشین ترجمه می‌کند که نتیجهٔ حاصل را برنامهٔ هدف[2] نامند و پسوند فایل مورد نظر در این مرحله نیز معمولاً obj است.

1. source program
2. object code



شکل ۱۱ـ۱ مراحل ایجاد برنامه

در روند برنامه‌سازی C، قبل از اینکه فاز ترجمه شروع شود برنامه‌ای به نام پیش‌پردازنده[1] به شکل خودکار اجرا می‌شود. پیش‌پردازندهٔ C فرامین خاصی را می‌پذیرد. این فرامین مشخص می‌کند که قبل از ترجمه باید روی برنامهٔ منبع، بعضی دستکاری و عملیات خاصی انجام گیرد. این عملیات معمولاً ضمیمه کردن فایلهای دیگری در برنامه یا فایلی است که باید ترجمه شود و جایگزین کردن بعضی سمبولهای خاص با متن برنامه است. قبل از تبدیل برنامه به زبان ماشین کامپایلر پیش‌پردازنده را به طور خودکار احضار می‌کند.

مرحلهٔ چهارم مرحلهٔ پیوند دادن[2] است. به طور متعارف برنامه‌های C شامل مراجعه به

1. preprocessor
2. linking

توابعی است که در جای دیگری تعریف شده‌اند، مانند کتابخانه‌های استاندارد، یا کتابخانه‌های برنامه‌سازان دیگر که روی پروژه‌های خاص کار می‌کنند. برنامهٔ هدف[1] ایجاد شده با کامپایلر، به طور نمونه به‌علت نبودن یا عدم وجود این گونه توابع قسمتهای خالی دارند که در برنامهٔ مورد نظر به آنها مراجعه شده است. نرم افزار linker یا پیونددهنده کد توابع ارجاع داده شده به برنامهٔ هدف در برنامهٔ مورد نظر ما را در محلهای مربوط الحاق و ضمیمه می‌کند تا برنامه‌ای اجرایی ایجاد شود. برنامه یا فایل حاصل پسوند exe خواهد داشت.

مرحلهٔ پنجم، بار کردن[2] قابل اجرایی از روی حافظهٔ جانبی به حافظهٔ کامپیوتر است که این کار به کمک نرم‌افزاری به نام loader انجام می‌گیرد. loader کپی برنامهٔ اجرا را از روی دیسک برمی‌دارد و آن را در حافظهٔ اصلی قرار می‌دهد.

در مرحلهٔ آخر کامپیوتر تحت کنترل cpu برنامه را اجرا می‌کند که در این مرحله باید داده‌های مورد نیاز نیز در اختیار آن قرار گیرد.

نصب نرم افزار توربو C

نسخه‌های متعددی از محیط برنامه‌نویسی زبان C وجود دارد مانند Turbo C و Borland C. در اینجا محیط برنامه نویسی Turbo C (Version 3.0) را بررسی می‌کنیم. برای نصب این نرم افزار، از فایل install.exe که همراه توربو C عرضه شده استفاده می‌شود. روش کار بسیار ساده است. فایل install.exe را اجرا می‌کنید. در حین اجرا به تعدادی سؤال که از شما پرسیده می‌شود (مانند تعیین مسیر نصب نرم افزار) پاسخ می‌دهید و سپس منتظر می‌مانید تا راه‌اندازی سیستم با این نرم‌افزار تکمیل شود. پس از نصب چند زیرفهرست به نامهای Bgi، Bin، Lib، Classlib، و Include در فهرست اصلی ساخته می‌شوند. فایل اجرایی نرم افزار در زیرفهرست Bin و به نام tc.exe وجود دارد.

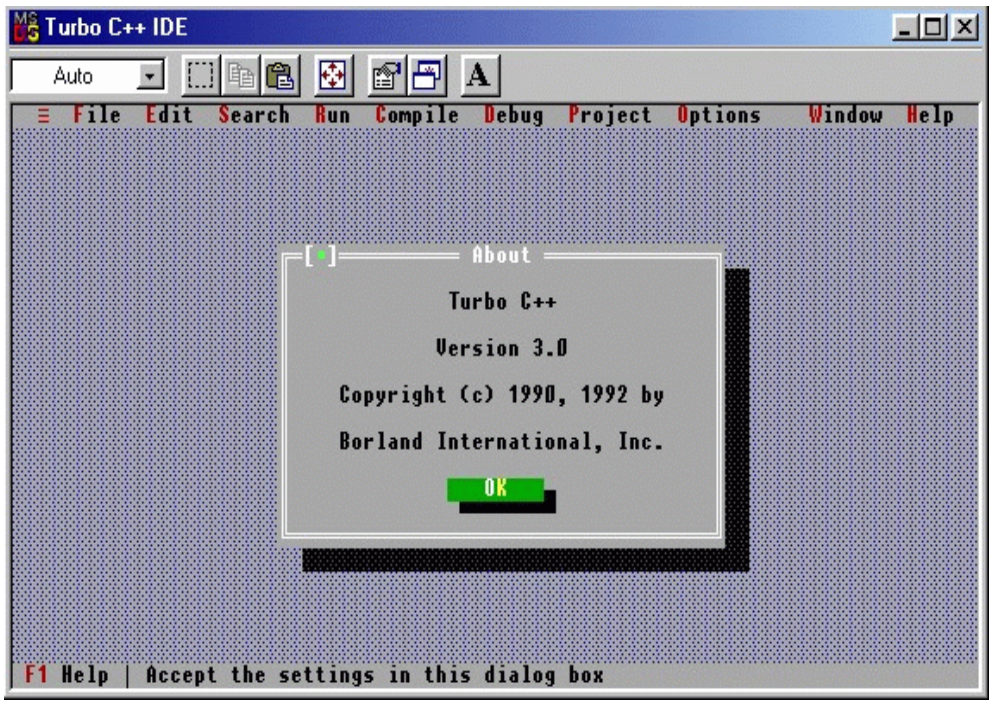
روش استفاده از نرم افزار

برای ایجاد برنامه‌ها از Integrated Development Environment یا به اختصار IDE موجود

1. object code
2. load

در محیط توربو C استفاده می‌کنیم که بسیار شبیه محیط توربو پاسکال است. در این محیط، تمام عملیات لازم برای ایجاد و تکمیل برنامه به صورت یکنواخت و روی فرمی در صفحهٔ نمایش از طریق منوها و پنجره‌ها در دسترس است. IDE تقریباً محیط ایده‌آلی برای آموختن و فراگیری عملی زبان C فراهم می‌کند. IDE این امکان را فراهم می‌سازد که بدون ترک محیط توربو C، بتوان برنامه‌ای را ویرایش و ترجمه کرد، فایل اجرایی ساخت، سپس آن را اجرا کرد. برای فعال کردن IDE به زیرفهرستی که توربو C در آن قرار دارد تغییر مسیر دهید، سپس فایل tc. exe را اجرا کنید. برای مثال اگر فهرست نصب شده TC باشد، جلوی خط فرمان Dos تایپ کنید D: \ TC \ BIN \ tc.exe. سپس کلید Enter را فشار دهید. پس از نصب وقتی که توربو C برای اولین بار اجرا می‌شود، صفحهٔ نمایش را به صورت شکل ۱۱ـ۲ مشاهده خواهید کرد.

وقتی که برای اولین بار IDE را احضار می‌کنید منوبار فعال می‌شود. اگر این طور نبود،



شکل ۱۱ـ۲ نمایش IDE

می‌توانید با فشردن کلید [F10] آن را فعال کنید. وقتی که منوبار فعال شد، یکی از منوهای موجود در آن پررنگ یا highlight خواهد شد. با مکان‌نما می‌توانید روی منوبار به طرف راست یا چپ حرکت کنید و منوهای دیگر را پررنگ کنید. در واقع آن منو را انتخاب کنید. هرکدام از منوها را که انتخاب کرده باشید برای اینکه ببینید در آن منو چه وجود دارد یا برای اینکه آن منو را فعال کنید کافی است کلید Enter را فشار دهید که در این صورت منوی مورد نظر باز می‌شود و محتوای آن را ملاحظه خواهید کرد. در واقع هر یک از منوهای اصلی یک یا چند منوی فرعی دارند. برای خروج از منو (درواقع بستن منو) و برگشت به منوبار کلید [F10] را فشار دهید.

می‌توانید با استفاده از کلیدها روی منوها حرکت و آنها را بررسی کنید و تا موقعی که کلید Enter را فشار نداده‌اید، هیچ اتفاقی نمی‌افتد. برای فعال کردن منوی پررنگ شده باید کلید Enter را فشار دهید. برای مثال برای انتقال کنترل از منوبار به پنجرهٔ ویرایش، با کلیدهای جهت‌دار، مکان‌نما را روی کلمه یا منوی Edit از منوبار ببرید (درواقع آن را پررنگ کنید)، سپس کلید Enter را فشار دهید. برای برگشت مجدد به منوبار، کلید [F10] را فشار دهید. به طور کلی برای انتخاب منوی اصلی دو روش وجود دارد:

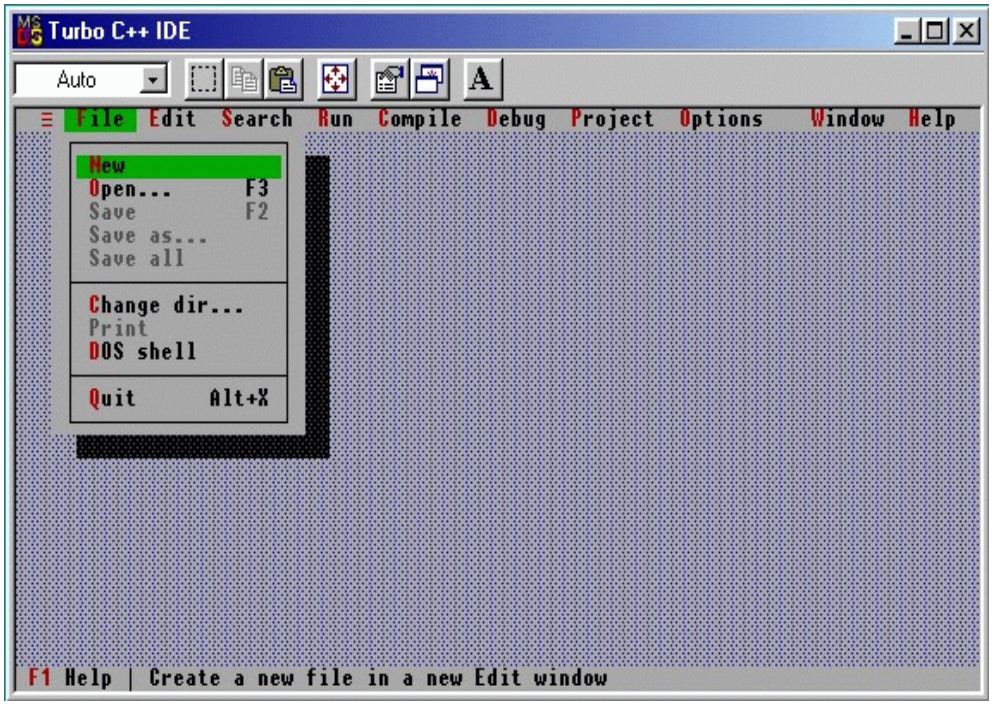
روش اول. با کلیدهای جهت‌دار حرکت کنید و منوی مورد نظر را انتخاب یا به عبارت دیگر پررنگ کنید. سپس کلید Enter را فشار دهید.

روش دوم. اولین حرف نام منو را (که معمولاً به رنگ قرمز است) تایپ کنید. برای مثال برای انتخاب File حرف F را تایپ کنید.

در ادامه، بعضی از منوها که کاربرد بیشتری دارند به اختصار شرح می‌دهیم.

منوی File

در این منو می‌توان اعمالی مانند ایجاد فایل جدید، باز کردن فایل، ذخیره‌سازی فایل، چاپ فایل و خروج از نرم افزار را انجام داد. این منو شامل چند گزینه یا فرمان است که در شکل ۳ـ۱۱ نشان داده شده است.

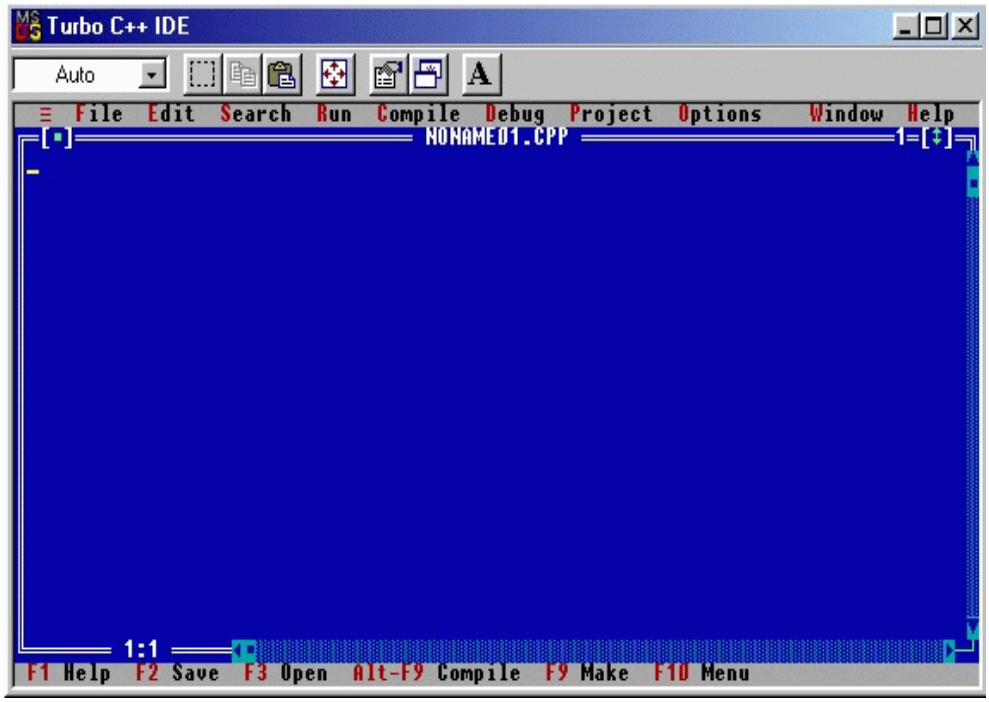


شکل ۱۱ـ۳ گزینه‌های منوی File

ـ **فرمان New**. این فرمان محتوای ویراستار را پاک و آن را برای ایجاد و ویرایش فایل جدید آماده می‌کند. فایلی با عنوان noname01.cpp باز می‌شود. اگر در فایل قبلی تغییراتی انجام گرفته ولی ثبت نشده باشد، این فرمان از شما سؤال می‌کند که آیا قبل از پاک شدن، آن را روی دیسک ذخیره کند یا نه. با اجرای این دستور صفحه‌ای مطابق شکل ۱۱ـ۴ نمایش داده می‌شود.

ـ **فرمان Open**. به کمک این فرمان می‌توان فایلی را از روی دیسک به محیط توربو C بارگذاری کرد. برای این کار مکان‌نما را روی این فرمان انتقال و کلید Enter را فشار می‌دهیم. پس از انجام این کار پیامی مبنی بر وارد کردن نام فایل صادر می‌شود. حال نام فایل را تایپ یا از لیست ارائه شده انتخاب می‌کنیم. سپس کلید Enter را فشار می‌دهیم. کلید میانبر این فرمان [F3] است.

ـ **فرمان Save**. این فرمان فایل موجود در محیط توربو C را روی دیسک ذخیره می‌کند.



شکل ۱۱ـ۴ نمایش فایل جدید

اگر فایل نامی نداشته باشد (یعنی نام آن noname.cpp باشد) نام جدید فایل را از شما می‌پرسد. کلید میانبر این فرمان [F2] است.

ـ **فرمان Save as**. با اجرای این فرمان می‌توان فایل موجود را با اسم جدید ذخیره کرد.

ـ **فرمان Print**. برای چاپ فایل جاری با چاپگر استفاده می‌شود.

ـ **فرمان Quit**. این فرمان موجب خروج از محیط توربو C می‌گردد. درضمن می‌توان با استفاده از کلیدهای ترکیبی ALT+X نیز از محیط توربو C خارج شد.

منوی Run

این منو برای ترجمه، اتصال و اجرای برنامه‌ای که هم‌اکنون در محیط توربو C قرار دارد به کار می‌رود. در این منو چند فرمان وجود دارد.

ـ **فرمان Run**. این فرمان موجب اجرای کامل برنامه می‌شود. کلید میانبر آن Ctrl + F9 است. در صورتی که برنامه ایراد داشته باشد، پیغام خطا یا Error مربوط نمایش داده می‌شود.

ـ **فرمان Goto cursor**. این فرمان موجب می‌شود که برنامه از محل استقرار مکان‌نما به بعد اجرا شود.

ـ **فرمان Trace into**. این فرمان موجب اجرای خط به خط برنامه می‌گردد. برای اشکال‌زدایی خطاهای موجود در منطق برنامه مناسب است. کلید میانبر آن [F7] است.

منوی Window

ـ **فرمان Close**. برای بستن پنجرهٔ فایل جاری استفاده می‌شود. کلید میانبر آن Alt + F3 است.

ـ **فرمان User screen**. این فرمان برای نمایش خروجی برنامهٔ اجرا شده مناسب است. کلید میانبر آن Alt + F5 است.

منوی Help

ـ **فرمان Index**. فهرست دستورهای موجود در توربو C را نمایش می‌دهد. با انتخاب هر دستور، راهنمای مربوط به آن دستور مشخص می‌شود. کلید میانبر این فرمان Shift + F1 است.

ـ **فرمان Topic search**. چنانچه مکان‌نما زیرِ دستوری قرار داشته باشد، با انتخاب این گزینه راهنمای مربوط به آن دستور مستقیماً نمایش داده می‌شود. کلید میانبر آن Ctrl + F1 است.

مراحل ایجاد برنامه در IDE

برای ایجاد برنامه‌ای ساده در محیط توربو C به ترتیب زیر عمل می‌کنیم.

- ایجاد فایل جدید با فرمان New
- نوشتن کد برنامهٔ مورد نظر
- نامگذاری و ذخیره کردن فایل با فرمان Save
- اجرای برنامه با فرمان Ctrl + F9

- در صورت نمایش پیغام خطا برطرف کردن خطا و مجدداً اجرای برنامه
- پس از اجرای کامل برنامه دیدن خروجی با فرمان Alt + F5
- بازگشت به محیط برنامه با کلید Esc

در صورتی که فایل مورد نظر را با نام pnoor ذخیره کرده باشید در پایان مراحل فوق سه فایل به اسامی pnoor.cpp، pnoor.obj، و pnoor.exe روی دیسک خواهید داشت که اولی فایل محتوی کد برنامه، دومی فایل کامپایل شده و سومی فایل اجرایی برنامه خواهد بود.

توابع کتابخانه‌ای

زبان C با یکسری توابع کتابخانه‌ای که عملیات و محاسبات پر کاربرد را انجام می‌دهند کامل می‌شود. این توابع کتابخانه‌ای به خودی خود قسمتی از زبان نیستند، هر چند که همهٔ مکملهای زبان آنها را شامل می‌شوند. بعضی توابع مقداری را باز می‌گردانند. برخی دیگر با بازگرداندن مقدار یک یا صفر نشان می‌دهند که شرط درست است یا نه. برخی دیگر نیز عملیات خاصی را روی داده‌ها انجام می‌دهند. معمولاً عملیاتی که وابسته به نوع کامپیوترند به صورت توابع کتابخانه‌ای نوشته می‌شوند.

به طور کلی می‌توان گفت که توابع در زبان C از تنوع زیادی برخوردارند. جهت انجام محاسبات ریاضی، اعمال کاراکتری، مقایسه و تغییرات رشته‌ای، تبدیل نوعهای مختلف به یکدیگر، ترسیمهای گرافیکی، کار روی فایلها و غیره مورد استفاده قرار می‌گیرند. هر تابع الگویی دارد که نوع تابع و نوع پارامترهای آن را مشخص می‌کند. الگوی هر تابع در فایل header جای دارد. تابع کتابخانه‌ای به سادگی با نوشتن اسم تابع به همراه فهرست آرگومانهایی که بیان‌کنندهٔ اطلاعات منتقل شونده به تابع‌اند در دسترس قرار می‌گیرد. آرگومانها باید در داخل پرانتزها قرار گیرند و با کاما از هم جدا شوند. در ادامه ضمن بررسی برخی توابع، الگو و نیز فایل header آن معرفی می‌گردد.

تابع clrscr()

این تابع یکی از پرکاربردترین توابع زبان C است و برای پاک کردن صفحهٔ نمایش در خروجی در نوع متنی به کار می‌رود و آرگومان ندارد. الگوی آن به شکل زیر است و در

فایل conio.h قرار دارد.

void clrscr (void)

۱. توابع تبدیل نوع

این توابع در زبان C برای انجام تبدیل نوع داده‌ها به یکدیگر به کار می‌روند و در فایل stdlib.h قرار دارند.

تابع atoi()

این تابع برای تبدیل نوع رشته به نوع integer به کار می‌رود. الگوی آن به شکل زیر است.

int atoi (const char *s)

تابع atof()

این تابع برای تبدیل نوع رشته به نوع ممیز شناور به کار می‌رود. الگوی آن به شکل زیر است.

double atof (const char *s)

تابع atol()

این تابع برای تبدیل نوع رشته به نوع long به کار می‌رود. الگوی آن به شکل زیر است.

long atol (const char *s)

۲. توابع ریاضی

این توابع در زبان C برای انجام برخی اعمال ریاضی مانند محاسبات مثلثاتی و عددی مورد استفاده قرار می‌گیرند. الگوی اغلب آنها در فایل math.h قرار دارد.

تابع sqrt ()

این تابع جذر عدد مثبت را محاسبه می‌کند و الگوی آن به صورت زیر است.

double sqrt (double x)

تابع pow ()

این تابع توانهای یک مبنا را محاسبه می‌کند و الگوی آن به صورت زیر است.

double pow (double x , double y)

نتیجهٔ تابع، عبارت x^y است. اگر مبنا صفر یا توان منفی یا صفر باشد، این تابع عمل نخواهد کرد.

تابع abs ()

این تابع برای محاسبهٔ قدرمطلق اعداد صحیح به کار می‌رود. اگر آرگومان این تابع عددی منفی باشد، نتیجهٔ حاصل از تابع عددی مثبت است و اگر آرگومان تابع مثبت یا صفر باشد، نتیجه عددی مثبت یا صفر خواهد بود. الگوی این تابع در فایل stdlib. h و نیز فایل math.h وجود دارد و به صورت زیر است.

int abs (int x)

تابع fabs ()

این تابع برای محاسبهٔ قدرمطلق اعداد اعشاری مورد استفاده قرار می‌گیرد و الگوی آن به صورت زیر است.

double fabs (double x)

تابع cabs ()

این تابع برای محاسبهٔ قدرمطلق اعداد موهومی به کار می‌رود. الگوی این تابع در فایل math. h قرار دارد. ساختمان اعداد موهومی به صورت زیر تعریف شده است.

```
struct complex {
                  double x ;
                  double y ;
                } ;
```

الگوی این تابع به صورت زیر است.

double cabs (struct complex num)

تابع sin ()

این تابع برای محاسبهٔ سینوس زاویه برحسب رادیان به کار می‌رود و دارای الگوی زیر است.

double sinh (double arg)

تابع cos ()

این تابع برای محاسبهٔ کسینوس زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است.

double cos (double arg)

تابع tan ()

این تابع برای محاسبهٔ تانژانت زاویه‌ای که برحسب رادیان است به کار می‌رود و الگوی آن به صورت زیر است.

double tan (double arg)

تابع asin ()

این تابع برای محاسبهٔ آرک‌سینوس یک عدد به کار می‌رود و الگوی آن به صورت زیر است.

double asin (double arg)

مقادیری که آرگومان این تابع قبول می‌کند در بازه ۱- تا ۱ است و نتیجهٔ حاصل به صورت رادیان و در بازه π/۲ و π/۲ است.

تابع acos ()

این تابع برای محاسبهٔ آرک‌کسینوس یک عدد استفاده می‌شود و الگوی آن به صورت زیر است.

double acos (double arg)

مقادیری را که آرگومان این تابع می‌پذیرد در بازه ۱- تا ۱ است و نتیجهٔ حاصل به صورت رادیان و در بازه صفر و π است.

تابع atan ()

این تابع برای محاسبهٔ آرک‌تانژانت یک عدد به کار می‌رود و الگوی آن به صورت زیر تعریف شده است.

double atan (double arg)

مقادیری که با این تابع محاسبه می‌شوند به صورت رادیان و در بازه π/2- و π/2 است.

تابع atan2 ()

این تابع دو آرگومان را دریافت، اولین آرگومان را بر دومین آرگومان تقسیم و آرک‌تانژانت نتیجهٔ حاصل را محاسبه می‌کند. الگوی تابع به صورت زیر تعریف شده است.

double atan2 (double y , double x)

این تابع دو آرگومان، به نامهای y و x دارد که آرک‌تانژانت y/x را محاسبه می‌کند.

تابع sinh ()

این تابع برای محاسبهٔ سینوس هیپربولیک زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است.

double sinh (double arg)

تابع cosh ()

این تابع برای محاسبهٔ کسینوس هیپربولیک عدد به کار می‌رود و الگوی آن به صورت زیر است.

double cosh (double arg)

تابع tanh ()

این تابع برای محاسبهٔ تانژانت هیپربولیک زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است.

double tanh (double arg)

تابع ceil ()

این تابع کوچک‌ترین عدد صحیح بزرگ‌تر یا مساوی با یک عدد را که آرگومان آن است محاسبه می‌کند. الگوی آن به صورت زیر است.

double ceil (double x)

تابع log ()

این تابع لگاریتم طبیعی یک عدد مثبت را محاسبه می‌کند (پایهٔ لگاریتم طبیعی عدد e = ۲/۷۱۸۲۰۰۰ است). الگوی این تابع به صورت زیر است.

double log (double x)

تابع log10 ()

این تابع لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می‌کند. الگوی این تابع به صورت زیر است.

double log10 (double x)

تابع exp ()

این تابع برای محاسبهٔ توانی از e (پایهٔ لگاریتم طبیعی) استفاده می‌شود و الگوی آن به صورت زیر است.

double exp (double arg)

تابع frexp ()

این تابع دارای دو آرگومان است. اولین آرگومان را به دو قسمت تبدیل می‌کند که قسمت اول به صورت کسری و در بازه ۰٫۵ تا کمتر از یک (۱)، 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 2^{exp} * کسر = num درمی‌آورد که قسمت کسر به عنوان نتیجهٔ عمل توسط تابع برگردانده می‌شود و قسمت توان در متغیر exp قرار می‌گیرد.

تابع ldexp ()

این تابع دو آرگومان دارد و الگوی آن به صورت زیر است.

double ldexp (double num , int exp)

نتیجهٔ تابع فوق عبارت $num *2^{exp}$ است.

تابع floor ()

این تابع بزرگ‌ترین مقدار صحیح کوچک‌تر یا مساوی یک عدد را که به‌صورت double نمایش داده می‌شود محاسبه می‌کند و الگوی آن به صورت زیر است.

double floor (double x)

تابع fmod ()

این تابع دو آرگومان دارد که باقیماندهٔ تقسیم اولین آرگومان را بر آرگومان دوم محاسبه می‌کند و به‌عنوان نتیجهٔ عمل برمی‌گرداند. الگوی تابع به صورت زیر است.

double fmod (double x , double y)

تابع modf ()

این تابع عددی را به دو قسمت صحیح و اعشاری تجزیه می‌کند و الگوی آن به صورت زیر است.

double modf (double x , int *y)

قسمت اعشاری عدد x نتیجهٔ عمل تابع را برمی‌گرداند و قسمت صحیح آن در متغیر y قرار می‌گیرد.

تابع hypot ()

این تابع با داشتن 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 قائم‌الزاویه‌اند.

تابع poly ()

این تابع برای ارزیابی چندجمله‌ای به کار می‌رود و دارای الگوی زیر است.

double poly (double x , int n , double C[ ])

در این الگو، n تعداد جمله و C آرایه‌ای است که ضرایب چندجمله‌ای را نگهداری می‌کند. x درجهٔ چندجمله‌ای را مشخص می‌نماید. به عنوان مثال، اگر n = 3، چندجمله‌ایی که ارزیابی می‌شود به صورت زیر است.

C[3]x + C[2]x + C[1]x + C[0]

۳. توابع کاراکتری

توابع کاراکتری برای ورودی ـ خروجی کاراکترها و مقایسهٔ آنها با یکدیگر، و تبدیل حروف کوچک و بزرگ به یکدیگر استفاده می‌شود. الگوی این توابع در فایل ctype. h قرار دارد.

تابع tolower ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و آن را به حرف کوچک انگلیسی تبدیل می‌کند. الگوی تابع به صورت زیر است.

int tolower (int ch)

تابع toupper ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و آن را به حرف بزرگ انگلیسی تبدیل می‌کند. اگر کاراکتر مورد نظر یکی از حروف بزرگ باشد، تغییری در آن ایجاد نمی‌شود. الگوی این تابع به صورت زیر است.

int toupper (int ch)

تابع isalnum ()

این تابع کاراکتری را به عنوان آرگومان می‌گیرد. اگر این کاراکتر هیچ کدام از حروف a تا z (A تا Z) یا ارقام ۰ تا ۹ نباشد، صفر برمی‌گرداند؛ در غیر این صورت نتیجهٔ برگردانده شده از

این تابع غیر از صفر خواهد بود. الگوی این تابع به صورت زیر است.

int isalnum (int ch)

تابع isalpha ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و تشخیص می‌دهد که این کاراکتر از حروف a تا z یا A تا Z هست یا نه. اگر این کاراکتر یکی از این حروف نباشد، نتیجهٔ حاصل از تابع عدد صفر است؛ در غیر این صورت صفر نیست. الگوی این تابع به صورت زیر است.

int isalpha (int ch)

تابع isascii ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و تشخیص می‌دهد که آیا این کاراکتر در بازهٔ صفر تا Ox7f هست یا خیر. اگر چنین نباشد نتیجهٔ حاصل از تابع برابر با صفر وگرنه غیر از صفر خواهد بود. الگوی این تابع به صورت زیر است.

int isascii (int ch)

تابع isdigit ()

این تابع کاراکتری را به‌عنوان آرگومان می‌پذیرد و 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)

تابع islower ()

این تابع کاراکتری را از ورودی می‌خواند و تشخیص می‌دهد که آیا این کاراکتر یکی از حروف کوچک انگلیسی هست یا خیر. اگر کاراکتر مورد نظر 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)

تابع ispunct ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و 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)

تابع isspace ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و مشخص می‌کند که آیا این کاراکتر یکی از کاراکترهای فضای خالی یا horizontal tab، vertical tab، form feed، carriage return، new line هست یا خیر و اگر نباشد، حاصل کار تابع صفر است. در غیر این صورت مخالف صفر خواهد بود. الگوی این تابع به صورت زیر است.

int isspace (int ch)

تابع isupper ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و تشخیص می‌دهد که آیا این کاراکتر از حروف بزرگ انگلیسی (A تا Z) هست یا خیر. اگر نباشد حاصل کار تابع صفر است و در غیر این صورت مخالف صفر خواهد بود. الگوی تابع به صورت زیر است.

int isupper (int ch)

۴. توابع رشته‌ای

این توابع معمولاً برای مقایسه و جستجوی کاراکترها یا رشته‌ای در رشتهٔ دیگر به کار می‌روند. الگوی توابع رشته‌ای در فایل "string. h" قرار دارد.

تابع strset ()

این تابع محتویات رشته را با کاراکتر پر می‌کند. الگوی این تابع به صورت زیر است.

strset (str , 'x') ;

این تابع رشتهٔ str را با x پر می‌کند.

تابع strnset ()

این تابع کاراکتری را به تعداد دفعات مشخصی در رشته کپی می‌کند. الگوی این تابع به صورت زیر است.

char *strnset (char *str , char ch , singned 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 رشتهٔ مورد نظر کپی شود.

تابع memchr ()

این تابع کاراکتری را در آرایه جستجو و محل اولین وقوع آن را مشخص می‌کند. اگر این کاراکتر پیدا شد، شمارهٔ آن محل را در اشاره‌گر کاراکتری قرار می‌دهد وگرنه کاراکتر تهی 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 جستجو شود. count محلی را در آرایه مشخص می‌کند که عمل جستجو باید از ابتدای آرایه تا آن محل انجام شود.

تابع memcpy ()

این تابع قسمتی از آرایه را در آرایهٔ 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 آرایهٔ منبع (آرایه‌ای که کاراکترهای مورد نظر در آنجا قرار دارند) و to اشاره‌گری است که به آرایهٔ مقصد (آرایه‌ای که کاراکترها باید به آنجا کپی شوند) اشاره می‌کند. تعداد کاراکترهایی که باید کپی شوند با آرگومان count مشخص می‌شود. برای مثال، اگر count برابر با ۱۰ ، یعنی اولین ۱۰ کاراکتر آرایه، باشد، from باید به ده محل اول آرایهٔ to کپی شوند. همان طور که از الگوی این تابع پیداست این تابع اشاره‌گری است که پس از مقایسه به آرایه‌ای اشاره می‌کند که to به آن اشاره می‌کرده است.

تابع memmove ()

این تابع تعدادی از کاراکترها را از آرایه‌ای به آرایهٔ دیگر کپی می‌کند. تفاوت آن با تابع memcpy در این است که اگر در تابع memcpy دو آرایه روی هم قرار گرفته باشند، 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 آرایهٔ مقصد است. count مشخص می‌کند که چه تعداد کاراکتر از آرایهٔ منبع به آرایهٔ مقصد

کپی شوند.

تابع memset ()

این تابع 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. count مشخص می‌کند که کاراکتر ch باید در چند عنصر آرایه (با شروع از اولین محل) کپی گردد. این تابع معمولاً برای ارزش‌دهی اولیهٔ آرایه‌ها استفاده می‌شود.

تابع strcspn ()

این تابع رشته‌ای را در رشتهٔ دیگر جستجو می‌کند و اولین محل از این رشته را که حتی یکی از کاراکترهای رشتهٔ مورد جستجو در آن محل باشند، به عنوان نتیجهٔ عمل برمی‌گرداند. الگوی این تابع به صورت زیر است.

int strcspn (const *str1 const *str2)

که در آن str2 در str1 جستجو می‌شود و محل اولین کاراکتر در رشتهٔ str1 که با هرکدام از کاراکترهای str2 برابر باشد، نتیجهٔ عمل تابع را برمی‌گرداند.

تابع strerror ()

این تابع موجب می‌شود تا بتوان در اثر بروز خطایی در برنامه پیام خطایی صادر کرد. نوع پیام خطا در اختیار برنامه‌نویس است. الگوی این تابع به صورت زیر است.

char *strerror (char *str)

تابع strlwr ()

این تابع رشته‌ای را به عنوان آرگومان می‌پذیرد و کلیهٔ حروف بزرگ آن را به کوچک تبدیل می‌کند. الگوی آن به صورت 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.

تابع strncat ()

این تابع قسمتی از رشته را به انتهای رشتهٔ دیگر الحاق می‌کند و سپس رشتهٔ نتیجه را به

کاراکتر تهی ختم می‌کند. الگوی تابع به صورت زیر است.

char * strncat (char *str1, const char *str2 , unsigned count)

که در الگوی فوق، count تعداد کاراکترهایی را مشخص می‌کند که از رشتهٔ str2 باید به رشتهٔ str1 الحاق شود.

تابع strncmp ()

این تابع تعداد مشخصی از کاراکترهای دو رشته را با یکدیگر مقایسه می‌کند (رشته‌ای از یک رشته را با زیررشته‌ای از رشتهٔ دیگر مقایسه می‌کند). الگوی این تابع به صورت زیر است.

int strncmp (const *str , const char *str , unsigned count)

در الگوی فوق به تعداد count کاراکتر، از دو رشتهٔ str1 و str2 با شروع از ابتدای رشته‌ها با یکدیگر مقایسه می‌شوند و یک عدد صحیح به عنوان نتیجهٔ عمل برمی‌گرداند.

تابع strncpy ()

این تابع تعداد مشخصی از کاراکترهای رشته را در رشته‌ای 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 دارند کمتر از مقدار count باشند، به تعداد لازم کاراکتر تهی در انتهای str1 کپی می‌شوند.

تابع strrchr ()

این تابع کاراکتری را در رشته جستجو و محل آخرین وقوع این کاراکتر را در آن رشته پیدا می‌کند و به اشاره‌گر نسبت می‌دهد. اگر این کاراکتر پیدا نشود، کاراکتر تهی در اشاره‌گر قرار خواهد گرفت. الگوی این تابع به صورت زیر است.

char *strrchr (const char *str , int ch)

تابع strspn ()

این تابع رشته‌ای را در رشتهٔ دیگر جستجو می‌کند و باعث برگشت طول زیررشتهٔ حاوی همهٔ کاراکترهای رشتهٔ مورد جستجو می‌شود. الگوی این تابع به صورت زیر است.

unsigned strspn (const char *str1 , const char *str2)

در الگوی فوق، str1 رشته‌ای است که عمل جستجو در آن انجام می‌شود و str2 رشته‌ای که باید در str1 جستجو گردد.

تابع strrev ()

این تابع کاراکترهای رشته را معکوس می‌کند؛ یعنی کاراکتر ابتدا را به انتهای آن منتقل می‌کند و این عمل را برای کلیهٔ کاراکترها انجام می‌دهد.

```
char *strrev (char *str)
```

تابع struper ()

این تابع در 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 شود.

۵. توابع تخصیص حافظهٔ پویا

متغیرهای معمولی و آرایه‌ها در حین ورود به بلوک ایجاد شده و تا خاتمهٔ اجرای بلوک باقی می‌مانند. 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 حافظهٔ پویا گویند. در C توابعی برای این منظور فراهم شده است.

تابع free ()

این تابع موجب می‌شود تا حافظهٔ اخذ شده از سیستم به آن بازگردانده شود. الگوی تابع در فایل "stdlib. h" قرار دارد و به شکل زیر است.

```
void free (void *)
```

برای بازگشت حافظه به سیستم کافی است اشاره‌گر آن محل از حافظه به این تابع داده شود.

تابع calloc ()

این تابع برای اخذ حافظه از سیستم در حین اجرای برنامه به کار می‌رود و دارای الگوی زیر

است.

void *calloc (unsigned unm , unsigned size)

الگوی این تابع در فایل stdlib. h قرار دارد. مقدار حافظه‌ای که این تابع در اختیار استفاده‌کننده قرار می‌دهد، برابر با size * num است. لذا از این تابع معمولاً جهت اخذ حافظهٔ لازم برای آرایه (شامل num عنصر که هر عنصر آن دارای طولی برابر با size باشد) استفاده می‌گردد. آدرس اولین بایت حافظه‌ای که با این تابع در اختیار استفاده‌کننده قرار می‌گیرد در اشاره‌گر قرار داده می‌شود. اگر این اشاره‌گر تهی باشد به معنی تخصیص نیافتن این حافظه است (عدم تخصیص حافظه ممکن است به دلیل نبودن حافظهٔ خالی به اندازهٔ مورد نیاز باشد). توصیه می‌شود که پس از استفاده از تابع calloc حتماً اشاره‌گر فوق تست شود تا از تخصیص حافظه اطمینان حاصل گردد.

تابع malloc ()

این تابع برای اخذ حافظه از سیستم، در حین اجرای برنامه، به کار می‌رود و دارای الگوی زیر است.

void *malloc (unsigned size)

الگوی این تابع در فایل "stdio. h" قرار دارد. میزان حافظه‌ای که این تابع از سیستم اخذ می‌کند با آرگومان size مشخص می‌گردد. آدرس حافظهٔ گرفته شده از سیستم با تابع malloc در اشاره‌گر قرار می‌گیرد. اگر این اشاره‌گر تهی (\0) باشد، بدین معنی است که حافظهٔ لازم تخصیص نیافته است.

تابع realloc ()

این تابع برای تغییر میزان حافظهٔ اختصاص یافته با توابع تخصیص حافظه، مثل malloc، به کار می‌رود. الگوی این تابع به صورت زیر است و در فایل "stdlib. h" قرار دارد.

*realloc (void *ptr , unsigned size)

برای تغییر میزان حافظهٔ تخصیص یافته، تنها ذکر اشاره‌گر به آن حافظه و همچنین اندازهٔ جدید این حافظه به عنوان آرگومان این تابع کافی است. میزان جدید حافظه ممکن است کمتر و یا بیشتر از حافظهٔ تخصیص یافته باشد. اطلاعات موجود در حافظهٔ تخصیص یافتهٔ قبلی از بین نخواهند رفت. تابع realloc پس از اخذ حافظه از سیستم، آدرس آن را در اشاره‌گر قرار

می‌دهد. اگر این اشاره‌گر تهی باشد، بدین معنی است که حافظه تخصیص نیافته است.

۶. توابع گرافیکی

در زبان C استاندارد ansi توابع گرافیکی در نظر گرفته نشده است. اما با توجه به کاربردهای متداول آنها، تعدادی از توابع گرافیکی مهم‌تر در نرم افزار توربو C را، که الگوی آنها در کتابخانه graphics.h قرار دارند، بررسی می‌کنیم.

تابع clreol()

این تابع موجب می‌شود که اطلاعات خط جاری از محل فعلی مکان‌نما تا انتهای خط جاری از چپ به راست پاک شود. الگوی آن به شکل زیر است.

```
void  clreol()
```

تابع deline()

این تابع موجب می‌شود که یک خط حذف شود. الگوی آن به شکل زیر است.

```
void delline()
```

تابع insline()

این تابع موجب می‌شود که یک خط خالی زیر خطی که مکان نما روی آن قرار دارد ایجاد شود و بقیهٔ خطوط نیز یک سطر به سمت پایین حرکت کنند. الگوی آن به شکل زیر است.

```
void insline()
```

تابع gotoxy()

این تابع موجب می‌شود که در محیط متن بتوان مکان‌نما را در محل دیگری از پنجرهٔ فعال قرار داد. الگوی آن به شکل زیر است که در آن x و y مختصات جدید مکان‌نما را تعیین می‌کند.

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

تابع moveto()

این تابع در محیط گرافیک، مکان‌نما را به محلی با مختصات x و y انتقال می‌دهد و الگوی آن به شکل زیر است.

```
void far moveto(int x , int y)
```

تابع getx() و gety()

این دو تابع مختصات جاری مکان‌نما را روی صفحهٔ گرافیکی برمی‌گردانند و الگوی آنها به شکل زیر است.

```
int far getx(void)
int far gety(void)
```

تابع wherex() و wherey()

این دو تابع در محیط متن، مختصات جاری مکان‌نما را نسبت به پنجرهٔ موجود برمی‌گرداند. الگوی آنها به شکل زیر است.

```
int wherex(void)
int wherey(void)
```

تابع gettext()

این تابع برای کپی کردن متن از صفحه نمایش به بافر به کار می‌رود. الگوی آن به شکل زیر است.

```
int gettext(int left , int top , int right , int bottom , void *buffer)
```

در این تابع مختصات گوشهٔ سمت چپ بالا و سمت راست پایین باید مشخص گردد. بافر اشاره‌گری است که به ناحیه‌ای از حافظه اشاره می‌کند و برای نگهداری متن استفاده می‌شود.

تابع puttext()

این تابع برای کپی کردن متن از بافر به صفحه نمایش به کار می‌رود. الگوی آن به شکل زیر است.

```
int puttext(int left , int top , int right , int bottom , void *buffer)
```

تابع movetext()

این تابع برای کپی کردن متن از قسمت صفحهٔ تصویر به قسمت دیگری استفاده می‌شود. الگوی آن به شکل زیر است.

```
int pettext(int left , int top , int right , int bottom , int newleft , newtop)
```

تابع window()

این تابع پنجرهٔ متن با ابعاد مشخص شده را فعال می‌کند. الگوی آن به شکل زیر است.

void window(int left , int top , int right , int bottom)

تابع textcolor()

این تابع رنگ متن را مشخص می‌کند. الگوی آن به شکل زیر است.

void textcolor(int color)

آرگومان این تابع مقادیر صفر تا ۱۵ را داراست.

تابع textbackground()

این تابع برای تعیین رنگ زمینهٔ متن به کار برده می‌شود. الگوی آن به شکل زیر است.

void textbackground(int color)

آرگومان این تابع مقادیر صفر تا ۶ را داراست.

تابع textmode()

این تابع برای تغییر حالت صفحه تصویر به کار می‌رود. الگوی آن به شکل زیر است.

void textmode(int mode)

آرگومان این تابع مقادیر 7,3,2,1,0 را داراست.

تابع initgraph()

این تابع برای تعیین تطبیق‌دهندهٔ حالت گرافیکی به کار می‌رود و راه‌انداز گرافیکی را فعال می‌کند. در واقع پیش از اجرای این دستور هیچ تابع گرافیکی نمی‌تواند اجرا شود. الگوی آن به شکل زیر است.

void far initgraph(int far *driver , int far *mode , char far *path)

اگر آرگومان driver عدد صفر منظور شود، به طور خودکار عمل می‌کند. آرگومان mode حالت تصویر را تعیین می‌کند. path نیز مسیر راه انداز را مشخص می‌کند.

تابع setbkcolor()

این تابع برای تعیین رنگ زمینه در حالت گرافیک به کار می‌رود و الگوی آن به شکل زیر است.

void far setbkcolor(int color)

تابع setpalette()

این تابع برای تغییر جعبه رنگ استفاده می‌شود و الگوی آن به شکل زیر است.

void far setpalette(int index , int color)

تابع closegraph()

این تابع برای پایان دادن به حالت تصویری گرافیکی استفاده می‌شود و الگوی آن به شکل زیر است.

void far closegraph()

تابع restorecrmode()

این تابع برنامه را خاتمه می‌دهد و تطبیق‌دهندهٔ گرافیکی را به حالت اول برمی‌گرداند. الگوی آن به شکل زیر است.

void far restorecrmode()

تابع putpixel()

این تابع رنگ مشخص شده را به محل تعیین شده با x و y می نویسد. الگوی آن به شکل زیر است.

void far putpixel(int x , int y , int color)

تابع line()

این تابع خطی از نقطه‌ای به نقطهٔ دیگر رسم می‌کند. الگوی آن به شکل زیر است.

void far line(int startx , int starty , int endx , int endy)

تابع lineto()

این تابع خطی از مکان جاری به نقطهٔ دیگر رسم می‌کند. الگوی آن به شکل زیر است.

void far lineto(int x , int y)

تابع circle()

این تابع دایره‌ای با مرکز و شعاع مشخص رسم می‌کند. الگوی آن به شکل زیر است.

void circle(int x , int y , int radius)

تابع ellips()

این تابع بیضی‌ای به مرکز x و y و دو شعاع xradius و yradius با رنگ جاری رسم می‌کند. الگوی آن به شکل زیر است.

void far ellipse(int x , int y , int start , int end , int xradius , int yradius)

در این الگو start و end آغاز و پایان بیضی را بر حسب درجه نشان می‌دهد. در صورتی که اولی صفر و دومی 360 باشد، بیضی به شکل کامل نمایش داده می‌شود.

تابع drawpoly()

این تابع با استفاده از رنگ جاری چندضلعی رسم می‌کند و الگوی آن به شکل زیر است.

void far drawpoly(int numpoints , int far *points)

در این الگو آرگومان numpoints تعداد گوشه‌های چندضلعی را تعیین می‌کند. مختصات گوشه‌ها در آرایه‌ای در نظر گرفته می‌شود که اشاره گر points به آن اشاره می‌کند.

تابع floodfill()

برای پر کردن شکلی بسته با رنگ مشخص از این تابع استفاده می‌شود. الگوی آن به شکل زیر است.

void floodfill(int x , int y , int bordercolor)

در این تابع x و y مختصات نقطه‌ای در درون شکل مورد نظر است.

تابع setfillstyle()

این تابع شیوهٔ پر کردن شکل را تغییر می‌دهد. الگوی آن به شکل زیر است.

void far setfillstyle(int pattern , int color)

در این الگو pattern مقادیر صفر تا ۱۲ را داراست.

تابع outtext()

این تابع برای نمایش متن در حالت گرافیکی استفاده می‌شود. الگوی آن به شکل زیر است.

void far outtext(char *str)

این تابع رشته‌ای را که با str مشخص شده در محل جاری مکان نما نمایش می‌دهد.

تابع outtextxy()

برای نشان دادن متن در حالت گرافیکی در محل خاصی با مختصات x و y از این تابع استفاده می‌شود. الگوی آن به شکل زیر است.

void far outtext(int x , int y , char *str)

تابع settextstyle()

این تابع برای انتخاب نوع فونت، اندازهٔ فونت و نوع نگارش افقی و عمودی استفاده می‌شود و الگوی آن به شکل زیر است.

void far settextstyle(int font , int direction , int charsize)

در این الگو font مقادیر صفر تا ۴، direction مقادیر صفر و ۱ و charsize مقادیر صفر

تا ۱۰ را دارایند.

تابع getimage()

این تابع برای کپی کردن یک ناحیه از پنجرهٔ گرافیک با مختصات مشخص به بافر استفاده می‌شود. الگوی آن به شکل زیر است.

void far getimage(int left , int top, int right , int bottom , void far *buf)

تابع putimage()

این تابع برای نمایش دادن محتوای یک بافر از داده‌های گرافیکی که قبلاً با تابع getimage در آن ذخیره شده به کار می‌رود. الگوی آن به شکل زیر است.

void far getimage(int left , int top, void far *buf , int op)

در این الگو، آرگومان op نحوهٔ ظاهر شدن تصویر را روی صفحهٔ نمایش تعیین می‌کند و مقادیر صفر تا ۴ را داراست.

تابع imagesize()

اندازهٔ بافر برحسب بایت برای ناحیهٔ داده شده را این تابع برمی‌گرداند. الگوی آن به شکل زیر است.

void far getimage(int left , int top, int right , int bottom)

تابع arc()

این تابع کمانی از start تا end را در امتداد دایرهٔ فرضی با مختصات مرکز x و y و نیز شعاع radius رسم می‌کند و الگوی آن به شکل زیر است.

void far arc(int x , int y , int star , int end , int radius)

تابع bar()

این تابع میله‌ای مستطیل شکل رسم و با رنگ و الگوی جاری پر می‌کند. الگوی آن به شکل زیر است.

void far bar(int left , int top , int right , int bottom)

تابع bar3d()

این تابع میله‌ای مستطیل شکل و سه بعدی به عمق آن با اندازهٔ depth پیکسل رسم و با رنگ و الگوی جاری پر می‌کند. الگوی آن به شکل زیر است.

void far bar3d(int left , int top , int right , int bottom , int depth , int topflag)

تابع getcolor()

این تابع رنگ جاری برای رسم را برمی‌گرداند و الگوی آن به شکل زیر است.

```
int far getcolor(void)
```

پاسخ خودآزماییها

خودآزمایی ۱

۱.

Integer	مجاز است.
-19	مجاز نیست، به دلیل وجود کاراکتر غیرمجاز '-'.
Lesson four	مجاز نیست، به دلیل وجود کاراکتر غیرمجاز فاصله.
Unit_25	مجاز است.
define	مجاز نیست، به دلیل کلمهٔ کلیدی بودن.
Loop2	مجاز است.
Star565void	مجاز است.
Please?	مجاز نیست، به دلیل وجود کاراکتر غیر مجاز '؟'.
Payam_noor	مجاز است.
C+ +	مجاز نیست، به دلیل وجود کاراکتر غیر مجاز '+'.
S#	مجاز نیست، به دلیل وجود کاراکتر غیر مجاز '#'.
Five$	مجاز نیست، به دلیل وجود کاراکتر غیر مجاز '$'.

۲. خط اول برنامه اعلان می‌کند که کتابخانه مربوط به توابع ورودی و خروجی زبان C برای دستیابی به توابع آن آماده شود. خط بعد نام تابع اصلی است. آکولاد باز شروع تابع است. در خط بعد متغیرهای x و S برنامه از نوع عدد صحیح تعریف شده‌اند. سه خط بعد دستورهای اصلی برنامه است که مقداردهی اولیهٔ متغیرx، محاسبهٔ مساحت درمتغیر S و چاپ آن در خروجی را شامل می‌شود. آکولاد بسته پایان تابع اصلی است.

۳. **تمامیت یا جامعیت**، به معنای درستی محاسبه. باید مشخص باشد که هر چیز دیگری که به برنامه افزوده می‌شود اگر در درستی اجرای برنامه نقشی نداشته باشد بی‌معنی است.

وضوح، به معنای خوانا بودن برنامه با تأکید بر اهمیت منطقی آن است. در این صورت تعقیب منطق برنامه بدون نیاز به تشریح برای هر برنامه‌نویس دیگر امکان‌پذیر خواهد بود.

سادگی، وضوح و درستی برنامه، که معمولاً با آسان نوشتن آن حفظ می‌شود و از جمله موضوعهای مرتبط با شکل برنامه‌اند.

کارآیی، که سرعت اجرا و بهره‌وری کار حافظه را بالا می‌برد.

ماژولار یا پیمانه‌ای بودن. بسیاری از برنامه‌ها به مجموعه‌ای از زیربرنامه‌ها تجزیه می‌شوند. این کار امکان تغییرات آتی برنامه را افزایش می‌دهد.

عمومیت. در برنامه از پارامترهایی استفاده می‌شود که با مقادیر مختلف داده‌ها به درستی کار کند.

٤. زبان سطح بالا: پاسکال، فورترن، کوبول، بیسیک
زبان سطح پایین: زبان ماشین و اسمبلی
زبان سطح میانی: زبان C.

خودآزمایی ۲

۱.

```
#include <stdio.h>
#define p 3.1415
main()
  {
     float R , s ;
     scanf("%f ", &R);
     s = p * R * R ;
     printf("%f ", s) ;
  }
```

توضیح. در این برنامه p از ثابتهای سمبولیک است و مقدار آن با دستور define معرفی

می‌شود. شعاع دایره و نیز مساحت آن از نوع اعشاری است. سپس شعاع دایره از ورودی خوانده می‌شود و دو بار در عدد p ضرب می‌شود. در نهایت مساحت s به شکل اعشاری چاپ می‌شود.

.۲

```
#include <stdio.h>
main()
  {
     char ch ;
     scanf("%c ", ch);
     printf("%d ", ch) ;
  }
```

توضیح. در این برنامه از کاراکترهای فرمت %c برای خواندن کاراکتر و %d برای نوشتن اعداد صحیح استفاده شده است. متغیر ch به شکل کاراکتر از ورودی خوانده و به صورت کد اسکی در خروجی چاپ می‌شود.

.۳

```
include <stdio.h>
main()
  {
     unsigned int k ;
     float y = 1 ;
     for(k = 2 ; k<= 100 ; + + k)
     y = y + 1/k ;
     printf("%f ", y) ;
  }
```

توضیح. در این برنامه، متغیر شمارنده است و مقادیر صحیح مثبت و کوچک‌تر از ۱۰۰ را شامل می‌شود. لذا از نوع بدون علامت یا unsigned تعریف شده است (استفاده از نوع int و نیز short هم مجاز است).

.٤

```
#include<math.h>
```

```
#include<stdio.h>
main()
  {
   float a , b , c , d , x1 , x2 , delta ;
   scanf("%f %f %f ",&a , &b , &c);
   delta = b * b – 4 * a * c ;
   if (delta<0)
      printf(" no root) ;
   else
      {
       d = sqrt(delta) ;
       x1 = (-b+d) / (2*a) ;
       x2 =(-b-d) / (2*a) ;
       printf("\n%f %f ", x1 , x2) ;
      }
  }
```

توضیح. در برنامهٔ بالا sqrt تابع از پیش فرض شده در زبان c است که جذر آرگومان خود را (که در اینجا متغیر delta است) برمی‌گرداند. این تابع جزء کتابخانهٔ "math.h" است، لذا در خط اول با دستور #include قرار گرفته است.

۵. خروجی برنامه عدد 244 می‌شود. ابتدا باینری معادل 12- در متغیر یک بایتی ch ریخته می‌شود.

(12) 10 = (00001100)2

برای به دست آوردن 12- از باینری 12+ متمم 2 می‌گیریم.

$(-12) = (11110100)_2$

چون ch متغیری بدون علامت است پس رشتهٔ 1110100 عددی بدون علامت فرض می‌شود.

(11110100) = (128 + 64 + 32 + 16 + 4) = 244

در دستور printf نیز محتوای متغیر ch به صورت عدد صحیح (به دلیل %d) چاپ می‌شود.

٦. خروجی برنامه عدد 144 می‌شود. مقدار 4*100 برابر 400 می‌شود که حاصل در یک بایت متغیر C جانمی‌گیرد و حداقل دو بایت جا نیاز دارد. معادل عدد 400 در دو بایت به

شکل زیر است.

0000 0001	1001 0000

چون متغیر C تنها یک بایت است فقط بایت سمت راست 400 یعنی 1001 0000 ذخیره می‌گردد و داریم

1001 0000 → بدون علامت → 128 + 16 = 144

خودآزمایی ۳

.۱

```
# include < stdio. h>
main ()
  {
    printf ("Payam noor university \n") ;
    puts ("Payam noor university ") ;
  }
```

توضیح. برای چاپ رشته‌ها در خروجی، سرعت تابع puts بیشتر است.

.۲

```
# include < stdio. h>
main ()
  {
     char ch ;
     ch = getchar() ;
     putchar(ch + 1) ;
  }
```

.۳

```
# include < stdio. h>
main ()
  {
    int m1 = 54 ;
    float m2 = 29. 4 ;
    char m3 = 'h' ;
    printf ("\n m1 = %d , m2 = %f , m3 = %c , m1 , m2 , m3) ;
    printf ("\n address of m3 is:  %p" , &m3) ;
  }
```

خروجی برنامه

```
m1 = 54 , m2 = 29.400000 , m3 = h
address of m3 is:  B8F4
```

توضیح. این برنامه جهت آشنایی با کاراکترهای فرمت است. %d عدد صحیح علامتدار، %f عدد اعشاری با شش رقم اعشار، %c کاراکتر و %p آدرس متغیر را چاپ می‌کند.

٤.

خروجی برنامه

```
x1 = 2.000000e + 04
x2 = 2.000000E + 04
x3 = 2e + 04
```

توضیح. این برنامه شکلهای مختلف نمایی یا علمی را نمایش می‌دهد.

٥.

خروجی برنامه

```
123.456000
123.456
123.46
```

توضیح. این برنامه شکلهای مختلف اعشاری با در نظر گرفتن طول میدان را نمایش می‌دهد. اولین داده به شکل کامل و با شش رقم اعشار، دومین داده با سه رقم اعشار و سومین داده به دلیل طول میدان کوچک‌تر یعنی ۲ گرد شده است.

٦. چون می‌خواهیم حاصل تقسیم به صورت اعشاری باشد باید تبدیل نوع انجام دهیم. بنابراین قبل از تقسیم، نوع اعشاری یعنی float را در داخل پرانتز قرار می‌دهیم (استفاده از cast). در این برنامه سه متغیر صحیح a , b , c از ورودی خوانده می‌شوند و average میانگین سه عدد است.

```
# include < stdio. H>
main ()
```

```
{
  int a , b , c ;
  float average ;
  printf (“\n Enter three integer numbers:  “) ;
  scanf (“%d%d%d” , &a , &b , &c) ;
  average = (float) (a + b + c)/3 ;
  printf (“\n average = %f” , average) ;
}
```

۷.

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

خروجی برنامه

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

توضیح. برای جابه‌جا کردن محتویات دو متغیر بدون استفاده از متغیر سوم، آن دو متغیر را جمع می‌کنیم و در یکی از آنها قرار می‌دهیم. سپس با عمل تفریق، این جابه‌جایی صورت می‌گیرد. به‌عنوان 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 یعنی ۱۲ را در y قرار می‌دهیم. سپس y را از x کم می‌کنیم (۱۲–۲۷) و حاصل آن یعنی ۱۵ را در x قرار می‌دهیم.

۸.

```
# include < stdio. h>
main ()
```

```
{
  int days , years , months , weeks ;
  printf (" Enter your age in days:  ") ;
  scanf ("%d" , &days) ;
  years = days / 365 ;
  days % = 365 ;
  months = days /30 ;
  weeks = days / 7 ;
  days % = 7 ;
  printf ("\nyour age is:  %d years , %d months , %d weeks , %d days." years ,
                                                    months , weeks , days) ;
}
```

توضیح. در این برنامه، از تقسیم صحیح و باقیماندهٔ تقسیم استفاده شده است. برای به‌دست آوردن سال، باید تعداد روزها را بر ۳٦۵ تقسیم کرد. برای به‌دست آوردن روزهای باقیمانده (پس از تبدیل به سال)، باید باقیماندهٔ تعداد کل روزها را نسبت به ۳٦۵ به‌دست آورد. این روند برای محاسبهٔ تعداد ماه، هفته و روز نیز به کار می‌رود. در این برنامه، days سن برحسب روز، years تعداد سال، months تعداد ماه و weeks تعداد هفته است. روزهای باقیمانده نیز در days قرار می‌گیرد.

خودآزمایی ۴

۱.

```
# include <stdio. h>
main ()
  {
    float x ;
    scanf ("%f" , &x) ;
    if (x<0) x = -x ;
      printf ("%f" , x) ;
  }
```

۲. در این برنامه، اگر قبل از 5 عبارت (float) نوشته نشود، تقسیم به صورت صحیح انجام می‌شود و حاصل تقسیم ۵ بر ۹ برابر صفر می‌گردد، یعنی خروجی همواره صفر خواهد شد.

```
# include < stdio.h >
main ()
  {
    int k = 32 ;
    float far , cel ;
    printf ("Enter farenheit: ") ;
    scanf ("%f" , & far) ;
    cel = ((float) 5 / 9) * (far - k) ;
    printf ("Celsius = %f" , cel) ;
    return 0 ;
  }
```

خروجی برنامه

```
Enter farenheit:  65
Celsius = 18. 333334
```

۳. y برابر 28 می‌شود. در واقع دستور فوق معادل دو دستور زیر است.

$++a ; \rightarrow a = 3$

$y = a * b + a + 10 ; \rightarrow y = 3 * 5 + 3 + 10 = 28$

٤.

```
# include <stdio. h >
main ()
  {
    float a , b , c ;
    scanf ("%f %f %f" , &a , &b , &c) ;
    if (a < b+c && b<a+c && c<a+b)
       printf ("yes") ;
    else
       printf ("n\ No") ;
  }
```

توضیح. در مثلث هر ضلع از مجموع دو ضلع دیگر کوچک‌تر است. در این برنامه با استفاده از دستور if این شرط بررسی می‌شود.

۵.

دستورها	معادل بیتی	مقدار 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
x = x ^ x ;	00000000	0

در این تمرین عملگرهای بیتی بررسی شده است.

نکتهٔ۱. هر شیفت به چپ عدد را دوبرابر می‌کند البته به شرطی که از حداکثر محدودهٔ متغیر خارج نشود یا به‌عبارتی دیگر هنگام شیفت به چپ دادن ارقام "1" از بین نرود. نتیجه آنکه اگر عددی n بیت به سمت چپ شیفت داده شود ضرب در 2^n می‌شود.

نکتهٔ۲. هر شیفت به راست عدد را نصف می‌کند به عبارتی دیگر اگر عددی n بیت به سمت راست شیفت داده شود، تقسیم صحیح بر 2^n می‌شود. بدیهی است هر عددی پس از تعدادی شیفت به راست تبدیل به صفر می‌شود.

نکتهٔ۳. xor هر عدد با خودش (توسط عملگر ^) برابرصفر می‌شود.

نکتهٔ٤. عملگرهای بیتی بیشتر برای برنامه‌نویسی سطح پایین و کار با وسایل I/O مثل پورتها استفاده می‌گردند. عملگر & معمولاً برای خاموش کردن بیت خاصی و برعکس عملگر | برای روشن کردن بیت خاصی استفاده می‌شوند.

نکتهٔ۵. باید به تفاوت بین عملگرهای منطقی و عملگرهای بیتی توجه داشته باشید. دو سمت عملگرهای منطقی شرط و دو سمت عملگرهای بیتی عدد می‌آید. اگر دو سمت عملگرهای منطقی عدد بیاید این اعداد به صورت True یا False تعبیر می‌شوند (صفر یعنی نادرست؛ غیرصفر یعنی درست).

٦. در دستور چاپ اول 8 && 7 عدد 8 درست و 7 درست تعبیر می‌شود و با توجه به

عملگر منطقی && ترکیب دو مقدار درست True یا 1 خواهد بود.

در دستور چاپ دوم با توجه به عملگر بیتی & داریم

7 →	0 0 0 0	0 1 1 1	AND
8 →	0 0 0 0	1 0 0 0	
	0 0 0 0	0 0 0 0	0عدد →

خودآزمایی ۵

.۱

```
#include<stdio.h>
main ()
   {
    int j , k ;
    for (j =1 ; j <= 10 ; j+ +) /* outer loop */
       {
        printf("%5d ", j) ;
        for (k=1; k<=10; k+ +) /* inner loop */
          printf("%5d", j * k) ;
        printf("\n") ;
       }
   }
```

ملاحظه می‌کنید که در این برنامه از دو حلقهٔ for تو در تو استفاده شده است.

خروجی برنامه

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

.۲

```
#include<stdio.h>
```

```
main ()
  {
    int ch , count = 0 ;
    printf("please enter a text: \n");
    ch = getchar() ;
    while (ch!='\n')
      {
        if (ch = =' ')
            count + + ;
        ch = getchar()
      }
    printf("number of spaces is %d\n", count) ;
  }
```

توضیح. با استفاده از تابع getchar از متن مورد نظر هربار یک کاراکتر خوانده می‌شود و کاراکتر خوانده شده به متغیر ch نسبت داده می‌شود. عمل خواندن کاراکترهای متن تا موقعی که به\n ،به‌عنوان پایان خط، نرسیده است ادامه می‌یابد و آزمایش رسیدن به پایان خط با دستور موجود در حلقهٔ while انجام می‌گیرد. هر بار در درون حلقهٔ while کاراکتر خوانده شده با کاراکتر معرف فضای خالی مقایسه می‌گردد تا درصورت تطابق، یک واحد به شمارندهٔ count که تعداد محل خالی در متن را می‌شمارد، افزوده شود. به‌طوری که گفتیم، در حلقهٔ while شرط مورد نظر برای اجرای بدنهٔ حلقه در آغاز تست می‌شود. لذا یک بار در خارج حلقه، مقداری برای متغیر ch خوانده می‌شود، سپس تست می‌گردد تا کلید برگشت تحریر شود که ’\n’ معرف آن است. پس از اتمام حلقه، مقدار متغیر count به عنوان تعداد محلهای خالی موجود در متن مورد نظر چاپ می‌شود.

.۳

```
#include<stdio.h>
main ()
  {
    int i ;
    long int x = 500000 ;
    for (i =1 ; i <= 5 ; ++i)
      {
        x += x * 0.16 ;
        printf("\n year %2d = %ld", i , x) ;
      }
  }
```

خروجی برنامه
year 1 = 580000 year 2 = 672800 year 3 = 780448 year 4 = 905319 year 5 = 1050170

.٤

```
#include<stdio.h>
main ()
  {
     int ch ;
     printf("please enter a character ? ");
     ch = getchar() ;
     if (ch = = EOF)
        printf("\n end of line found\n") ;
     else if (ch = ='\n')
        printf("\n new line character\n") ;
     else if (ch<' ')
        printf("\n control character %d\n", ch) ;
     else if ((ch>='0') && (ch<='9'))
        printf("\n character %c is a digit\n", ch):
     else if((ch>='A') && (ch<='Z'))
        printf("\n character %c is upper_case \n",ch)';
    else if((ch>='a') && (ch<='z'))
        printf("\n character %c is lower_case \n",ch)';
     else
        printf("\n other printable character %c \n", ch) ;
  }
```

.٥

```
#include<stdio.h>
main ()
  {
    int i , k , n sum =1 ;
    printf("enter number") ;
    scanf("%d",&n) ;
    k = n / 2 ;
    for(i = 2 ; i <= k ; + + i)
```

```
        if (n%i = = 0) sum = sum + i ;
          if (sum = = n)
            printf ("%d YES", n) ;
          else
             printf ("%d NO", n) ;
  }
```

توضیح. در این برنامه در ابتدا، k برابر نصف عدد n قرار داده می‌شود، سپس در حلقهٔ for هر بار باقیماندهٔ تقسیم n بر اعداد ۲ تا k به‌دست می‌آید. چنانچه باقیمانده برابر صفر باشد، یعنی n بر i بخش‌پذیر باشد، مقدار i یکی دیگر از مقسوم‌علیه‌های n بر sum که در آغاز برابر ۱ (اولین مقسوم‌علیه n) قرار داده شده است افزوده می‌شود. پس از خروج از حلقه، متغیر sum مجموع مقسوم علیه‌های n را خواهد داشت. سپس بررسی می‌شود که آیا این مقدار برابر خود n است یا نه، که برحسب مورد پیغام مناسب چاپ می‌شود.

.٦

```
#include<stdio.h>
main ()
  {
     int i;
     unsigned long int a1 , a0 ;
     a0 = 1 ;
     al = 1 ;
     printf("%d ", a0);
     printf("%d ",a1);
     for (i = 3 ; i <=10 ; + +i)
        {
          a2 = a0 + a1 ;
          a0 = a1 ;
          a1 = a2 ;
          printf("%d ", a2) ;
        }
  }
```

خروجی برنامه

1 1 2 3 5 8 13 21 34 55

۷. فرض کنید num1 , op , num2 به ترتیب معرف عملوند اول، عملگر و عملوند دوم در مورد یکی از چهار عمل اصلی باشد که مقادیر آنها به ترتیب از طریق ورودی خوانده

می‌شود و برحسب اینکه مقدار عملگر، برابر '+', '-', '*' یا '/' باشد، عمل ریاضی مربوط روی مقادیر عملوندها انجام گیرد. برنامهٔ مورد نظر و خروجی آن برای بعضی مقادیر نمونه، در زیر نشان داده شده است.

```
main ()
  {
    float num1, num2 ;
    char op ;
    while (1)
      {
        printf("Type number, operator, number:  \n");
        scanf("%f %c %f , &num1, &op, &num2);
        switch (op)
          {
            case '+':
             printf("=%f", num1 + num2) ;
             break ;
            case '-':
             printf("=%f", num1 - num2) ;
             break ;
            case '*':
             printf("=%f", num1 * num2) ;
             break ;
            case '/':
             printf("=%f", num1 / num2) ;
             break ;
            default:
             printf ("Error") ;
          }
    printf("\n") ;
  }
```

۸

```
#include<stdio.h>
main ()
   {
     int n , count =1 ;
     float x , avg , sum = 0 , max = 0 , min = 20 ;
     printf ("how many numbers?") ;
     scanf ("%d",&n) ;
     while (count <= n)
```

```
    {
      scanf ("%f",&x) ;
      sum + =x ;
      if (max<x)
        max = x ;
      if (max>x)
        min = x ;
      + + count ;
    }
  avg = sum / n ;
  printf ("\n average= %f , max= %f , min= %f ", avg , max , min) ;
}
```

توجه دارید که در این برنامه از حلقهٔ while و دستور کنترلی if استفاده شده است.

۹.

```
main ()
  {
    int i , j , k ;
    int even = odd = 0 ;
    for (i = 0 , j =2 , k = 1 ; i<n ; i +=2 , j += 2 , k += 2)
      {
        even += j ;
        odd += k ;
      }
    printf("even = %d odd = %d", even , odd) ;
  }
```

همان طور که ملاحظه می‌کنید در دستور for از عملگر کاما استفاده شده است.

خودآزمایی ۶

۱.

```
int average(int n)
  {
    int i , n ;
    float sum = 0 ;
    for(i =1 ; i<=n ; i++)
    sum += i ;
    printf("%f" , sum / n) ;
    return 0;
  }
```

۲.

```
double power(double x , double n)
  {
    return exp(n * log(x)) ;
  }
```

در این برنامه از توابع کتابخانه‌ای log برای لگاریتم و exp برای تابع‌نمایی استفاده شده که هر دو در فایل کتابخانه‌ای math. h قرار دارند. این تابع، توان اعشاری از هر عدد اعشاری (یعنی x^n) را محاسبه می‌کند.

۳. براساس تعاریف ریاضی، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد a و b، بزرگ‌ترین عددی است که بر هر دو عامل a و b بخش‌پذیر باشد. یکی از راههای ساده برای تعیین آن، روش پلکانی یا روش اقلیدس است که روش تقسیمات متوالی نیز نامیده می‌شود. در این روش عدد اولی را بر دومی تقسیم می‌کنیم و اگر باقیمانده برابر صفر بود، عدد دوم بزرگ‌ترین مقسوم‌علیه مشترک است. در غیر این صورت به جای عدد اولی، عدد دومی را قرار می‌دهیم و به جای عدد دومی نیز باقیمانده را قرار می‌دهیم و مجدد باقیماندهٔ تقسیم عدد اولی بر دومی را به دست می‌آوریم. این عمل را آنقدر ادامه می‌دهیم تا باقیمانده برابر صفر گردد.

```
#include<stdio.h>
main ()
  {
    int m , n , r ;
    scanf("%d%d" , &n , &m) ;
    r = m % n ;
    while (r!=0)
      {
        m = n ;
        n = r ;
        r = m % n ;
      }
    printf(" The BMM is = %d" , n) ;
  }
```

٤.

```
int BMM(int a , int b)
  {
    int r ;
    r = a % b ;
    while (r!=0)
      {
        a = b ;
        b = r ;
        r = a % b ;
      }
    return(b) ;
  }
```

۵. در این برنامه تابع فرعی به صورت آرگومان تابع printf در تابع اصلی فراخوانی شده است.

```
#include<stdio.h>
main ()
  {
    int x , y ;
    int BMM(int x , int y) ;
    scanf("%d%d" , &x , &y) ;
    printf("_____ \n") ;
    printf("x=%d | y=%d | BMM=%d |\n" , x , y , BMM(x , y)) ;
    printf("-----------------------\n") ;
  }
   int BMM(int x , int y)
      {
        int rem ;
        rem = x % y ;
        if (rem= =0) return(y) ;
          return(BMM(y , rem)) ;
      }
```

خروجی برنامه

X = 637 | Y = 60 | BMM = 3 |
X = 544 | Y = 98 | BMM = 2 |
X = 16 | Y = 12 | BMM = 4 |
X = 81 | Y = 45 | BMM = 9 |

٦. ساده‌ترین راه برای به دست آوردن معادل یک عدد در مبنای دیگری مانند d آن است که عدد مورد نظر را بر مبنای مزبور تقسیم کنیم. باقیماندهٔ حاصل، مرتبهٔ اول عدد حاصل در مبنای جدید را تشکیل خواهد داد. سپس خارج قسمت را بر مبنای جدید تقسیم می‌کنیم که این بار باقیماندهٔ حاصل، مرتبهٔ دوم عدد حاصل در مبنای جدید را به دست می‌دهد. این عمل را ادامه می‌دهیم تا خارج قسمت برابر صفر گردد. برنامهٔ زیر براساس این روش نوشته شده است که در آن باقیمانده‌ها در آرایه‌ای قرار داده می‌شود و در پایان کار، محتوای آرایهٔ مورد نظر که ارقام عدد مطلوب را در مبنای جدید خواهد داشت چاپ می‌گردد (البته باید توجه داشت که آخرین باقیمانده که آخرین یا چپ‌ترین رقم عدد حاصل در مبنای ۲ را تشکیل می‌دهد، در خانهٔ آخر آرایه قرار دارد، لذا محتوای آرایه باید با شروع از خانهٔ آخر به طرف خانهٔ اول چاپ گردد). خروجی مزبور برای عدد ۲۳ نشان داده شده است.

```
#include<stdio.h)
main()
  {
    int i , j , n , a[10] ;
    scanf("%d" , &n) ;
    printf("The decimal no:  %d\nlts binary value: " , n) ;
    i = 0 ;
    do
      {
        a[i] =n%2 ;
        n = n/2 ;
        i++ ;
      } while (n!=0) ;
          for(j=i-1 ; j>=0 ; j--)
             printf("%d" , a[j] );
        printf("\n") ;
  }
```

خروجی برنامه

```
The decimal no:  23
Its binary value:  10111
```

۷. در این برنامه m عدد مورد نظر در مبنای ۱۰ و d نیز مبنای جدید است.

```
void binary (int m , int d)
```

```
{
  int i , k , a [15] ;
  i = 0 ;
  do
    {
      a[i] = m % d ;
      m = m / d ;
      i++ ;
    }while (m! = 0) ;
  for (k=i-1 ; k> = 0 ; k--)
    printf("%d" , a[k]) ;
}
```

۸. در اینجا در هر تقسیم متوالی عدد مورد نظر بر مبنای جدید، باقیماندهٔ حاصل با ضرب‌شدن در توان مناسبی از ۱۰ به سمت چپ انتقال پیدا می‌کند. سپس با مقدار ایجاد شده تا این لحظه جمع می‌گردد، به طریقی که نتیجهٔ حاصل درست درآید.

```
#include<stdio.h>
main()
  {
    int base ;
    unsigned long dec , a=0 , b=1 , c ;
    printf("\n Enter Base=") ;
    scanf("%ld" , & base) ;
    printf("\nEnter a number=") ;
    scanf("%ld" , &dec) ;
    c = dec ;
    while (dec)
        {
          a + = dec % base * b ;
          b = 10 ;
          dec = dec / base ;
        }
    printf ("%ld (dec)= %ld(base %d)" , c , a , base) ;
    return 0 ;
  }
```

خروجی برنامه

```
Enter Base = 2
Enter a number = 1112
1112(dec) = 10001011000(base 2)
```

۹.

```
#include<stdio.h>
main()
   {
     int n ;
     printf("enter your number:  ") ;
     scanf("%d" , &n) ;
     printf("\n the binary equivalent of %d , is:  " , n) ;
     binary (n) ;
     printf("\n") ;
   }
binary(n)
int n ;
   {
     int r ;
     r = n % 2 ;
     if (n > =2)
       binary(n/2) ;
     printf("%d" , r) ;
     return ;
   }
```

خروجی برنامه

```
enter your number:  19
the binary equivalent of 19 is:  10011
```

۱۰. برای این کار باید عدد مزبور را مرتب بر ۱۰ تقسیم کنیم و باقیماندهٔ حاصل را در توانهای متوالی از ۲ یعنی در اعداد ... , 8 , 4 , 2 , 1 ضرب و این نتایج را با یکدیگر جمع کنیم. براساس این منطق، تابع مورد نظر در زیر نشان داده شده است.

```
int decimal (int bin)
   {
     int sum = 0 , k = 1 ;
     while (bin != 0)
       {
         sum = sum + bin %10 * k ;
         k = k * 2 ;
       }
     return(k) ;
   }
```

۱۱. می‌دانیم که اعداد اول اعدادی‌اند که جز بر یک و خودشان بخش‌پذیر نیستند. باز هم می‌دانیم کلیهٔ اعداد اول به غیر از عدد ۲ اعداد فردند. بنابراین در این برنامه اول تست می‌شود که آیا عدد دریافتی مساوی یا کوچک‌تر از ۳ است. در صورت مثبت بودن پاسخ، برنامه خاتمه می‌پذیرد. در غیر این‌صورت تست می‌شود که آیا عدد مورد نظر عدد زوج است. در صورت مثبت بودن پاسخ، پیغامی مبنی بر اینکه عدد دریافتی عدد اول نیست چاپ می‌گردد و برنامه خاتمه می‌پذیرد. درصورتی که هیچ یک از دو حالت بالا مصداق پیدا نکند، با استفاده از تابع کتابخانه‌ای sqrt، جذر عدد مزبور محاسبه می‌گردد. سپس در حلقهٔ for، عدد مزبور بر اعداد فرد بین ۳ تا جذر عدد مورد نظر تقسیم می‌گردد. اگر عدد مورد نظر بر یکی از این اعداد بخش‌پذیر باشد، پیغامی مربوط به اینکه عدد مزبور عدد اول نیست چاپ می‌گردد و برنامه پایان می‌یابد. چنانچه حلقهٔ for کامل گردد، عدد دریافتی عدد اول است. بنابراین پس از چاپ پیغام مناسبی مبنی بر اینکه عدد دریافتی عدد اول است، برنامه به پایان می‌رسد.

```
#include<stdio.h>
#include<math.h>
main()
  {
     int n , i , j , t ;
     printf("\n input your number:  ") ;
     scanf(%d" , &n) ;
     printf("%d" , n) ;
     if (n<=3)
       {
         printf("\n %d is a prime number" , n) ;
         exit() ;
       }
     if ((n%d) = =0)
       {
         printf("\n %d is not a prime number" , n) ;
         exit() ;
       }
     j = sqrt(n) + 1 ;
     for (i=3 ; i<j ; i=i+2)
       if (n%i= =0)
         {
           printf("\n %d is not a prime number" , n) ;
           exit() ;
         }
     printf("\n %d is a prime number" , n) ;
```

```
    exit() ;
}
```

خروجی برنامه

```
input your number:  149
149 is a prime number
input your number:  1597
1597 is a prime number
input your number:  1321
1321 is a prime number
input your number:  124
124 is not a prime number
```

۱۲. برنامهٔ مورد نظر با نمونه‌ای از خروجی آن در زیر نشان داده شده است. در برنامهٔ مزبور کلیهٔ مقسوم‌علیه‌های آن عدد که مساوی یا کوچک‌تر از نصف عدد مزبور است به دست می‌آیند و با یکدیگر جمع می‌گردند. سپس نتیجهٔ حاصل با خود عدد مقایسه می‌گردد. اگر با یکدیگر مساوی بودند، عدد مورد نظر یک عدد کامل است؛ در غیر این صورت عدد کامل نخواهد بود.

```
#include<stdio.h>
main()
  {
    int n , r , i , z , k = 1 ;
    scanf("%d" , &n) ;
    z = n/2 ;
    for (i=2 ; i<=z ; i++)
      {
        r = n % i ;
        if (r= =0) k = k + i ;
      }
  if (k= =n)
    printf("%d is a complete number" , n) ;
  else
    printf("%d is not a complete number" , n) ;
  }
```

خروجی برنامه

```
6 is a complete number
40 is not a complete number
```

۱۳. برنامهٔ مورد نظر با خروجی آن برابر حالتی که m = 10000 باشد در زیر نشان داده شده است.

```
#include<stdio.h>
main()
  {
     int m , k ;
     void complete (int x) ;
     printf ("\nEnter your number: ") ;
     scanf ("%d" , m) ;
     printf ("%d\n" , m) ;
     for (k=1 ; k=m ; k++)
       complete(k) ;
  }
void complete (int x)
   {
     int i , sum = 0 , z = x / 2 ;
     for (i=1 ; i<=z ; i++)
       if (x%i = = 0)
         sum=sum + i ;
       if (sum= =x)
         printf("%d" , x) ;
   }
```

خروجی برنامه

```
Enter your number:  10000
                    6
                    28
                    496
                    8128
```

۱٤. برنامهٔ مورد نظر با خروجی آن برابر m = 100 در زیر نشان داده شده است. در این برنامه دو حلقهٔ تودرتو به کار برده شده که حلقهٔ بیرونی اعداد ۲ تا m را درنظر می‌گیرد و حلقهٔ درونی تعیین می‌کند که آیا عدد مزبور با دو واحد بزرگ‌تر از خود تشکیل اعداد دوقلو می‌دهند یا نه. برای این کار باید دو عدد مزبور اعداد اول باشند، یعنی بر تمام اعداد بین ۲ تا آن بخش‌پذیر نباشند. به عبارت دیگر، باقیماندهٔ تقسیم آنها بر تمامی اعداد کوچک‌تر از آنها غیرصفر باشد.

```
#include<stdio.h>
main()
  {
     int a , n , i , p ;
     printf("\n Enter your number: ") ;
     scanf("%d" , &n) ;
     for (a = 3 ; a<n ; a = a + 2)
       {
         for (i = 3 ; i<a ; i = i + 2)
           {
             if (a%i !=0 && (a+2) % i != 0)
               p = 0 ;
             else
               {
                 p = 1 ;
                 break ;
               }
           }
         if (p= =0)
           printf("%d & %d\n" , a , a+2) ;
       }
  }
```

خروجی برنامه

```
Enter your number:  100
                    3 & 5
                    5 & 7
                    11 & 13
                    17 & 19
                    29 & 31
                    41 & 43
                    59 & 61
                    71 & 73
```

۱۵.

```
int even (int m)
 {
   if (m%2 = = 0)
     return(1) ;
   else
     return(0) ;
   }
```

.۱٦

```
#include<stdio.h>
main()
  {
    void transfer(int , char , char , char) ;
    int n ;
    printf("welcome to the towers of Hanoi\n\n") ;
    printf("how many disks ?") ;
    scanf("%d" , &n) ;
    printf("\n") ;
    transfer(n ,'l' ,'r' , 'c') ;
  }
void transfer (int n , char from , char to , char temp)
            /* transfer n disks from one pole to anuther */
            /* n=number of disks from = origin
               to = destination
               temp = temporary storage */
   {
    if (n>0)
       { /* move nth disk from origin to destination */
         printf("move disk %d from %c to %c\n" , n , from , to) ;
          /* move n-1 disks from temporary to destination */
          ransfer (n-1 , temp , to , from) ;
       }
    return ;
   }
```

خروجی این برنامه برای n = 3 به صورت زیر خواهد بود.

```
Welcome to the towers of hanoi
how many disks ? 3
move disk 1 from l to r
move disk 2 from l to c
move disk 1 from r to c
move disk 3 from l to r
move disk 1 from c to l
move disk 2 from c to r
move disk 1 from l to r
```

۱۷. با فراخوانی تابعی به نام line، خط بالایی و پایینی مستطیل و همین‌طور کناره‌های آن با چاپ کردن حروف گرافیکی تشکیل می‌گردد.

```
main()
   {
     line() ;
     printf ("\XDB COMPUTER SCIENCE \ XDB \n") ;
     line();
   }
 void line() /* draws solid line on screen , 20 chars ling */
   {
     int i ;
     for (i=1 ; i<=20 ; + +i)
        printf("\XDB") ;
     printf("\n") ;
   }
```

.۱۸

```
main()
   {
    twobeep() ;
    printf("type any character:  ") ;
    getche() ;
    twobeep() ;
   }
twobeep()
   {
    int k ;
    printf("\x7") ; /* first beep */
    for (k=1 ; k<5000 ; k+ +) ; /* delay */
      printf("\x7") ; /* second beep */
   }
```

برنامه در آغاز تابع twobeep را فرا می‌خواند که در نتیجهٔ اجرای آن، دوبار صدای کوتاه یا beep شنیده می‌شود. سپس برنامه از شما می‌خواهد که کلیدی را فشار دهید. هنگامی که این کار را انجام دادید دوباره مانند قبل، دوبار صدای مقطع یا بوق متوالی با فاصلهٔ کوتاه شنیده می‌شود. حلقهٔ for در بین دوبار دستور printf موجب تأخیر زمانی یا delay بین دو صدای متوالی می‌گردد. ملاحظه می‌کنید که در بدنهٔ این حلقه، هیچ دستوری وجود ندارد. بلکه فقط در انتها شامل علامت سمیکولون است که نمایانگر دستور تهی یا null statement است.

۱۹. تابع مزبور تابعی بازگشتی است و مجموع اعداد 1 تا n را محاسبه می‌کند و به تابع

فراخواننده برمی‌گرداند.

خودآزمایی ۷

.۱

```
#include<stdio.h>
main ()
  {
    unsigned long int a[10] ;
    int i ;
    a[0] = a[1] =1 ;
    printf ("%d %d", a[0] , a[1]) ;
      for (i = 2 ; i<10 ; + + i)
        {
          a[i] = a[i-1] + a[i-2] ;
          printf("%d ", a[i]) ;
        }
  }
```

خروجی برنامه

```
1 2 3 5 8 13 21 34 55
```

.۲

```
# include<stdio.h>
# define size 10
main ()
  {
    int s[size] , j ;
    for (j=0 ; j<size ; i ++)
       s[j] = 2 +2*j ;
    printf ("%s% 13s\n" , "Element" , "Value") ;
    for (j=0 ; j<=size ; j++)
       printf("%7d% 13d\n" , j , s[j]) ;
  }
```

خروجی برنامه

Element	Value
0	2
1	4

2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

۳.

خروجی برنامه

```
Sum = 19
```

٤.

```
#include<stdio.h>
main ()
   {
    int m , k , a[10] , i = 0 ;
    printf("enter a number:  ") ;
    scanf("%d",&m) ;
    printf("\n binary form = %d", m) ;
    while (m>=1)
       {
        a[i] = m % 2 ;
        m / = 2 ;
        + + i ;
       }
    for (; i >= 0 ; --i)
      printf("%d" , a[i]) ;
   }
```

توضیح. برای تبدیل عددی از مبنای ۱۰ به مبنای دلخواه دیگری مانند d باید آن عدد را بر d تقسیم کرد، خارج‌قسمت و باقیمانده را به دست آورد که باقیمانده مقدار مرتبهٔ اول عدد در مبنای جدید خواهد بود. سپس باید دوباره خارج‌قسمت را بر مبنای d تقسیم کرد که باقیماندهٔ آن مقدار مرتبهٔ دوم عدد در مبنای جدید خواهد بود. این عمل را باید به ترتیب ادامه داد تا خارج‌قسمت مساوی صفر گردد. در این برنامه d= 2 است. بنابراین اولین باقیماندهٔ تقسیم عدد دریافتی در اولین عضو آرایه یعنی a[0] قرار داده می‌شود. سپس خارج‌قسمت عدد

مزبور بر ۲ به دست می‌آید. باقیماندهٔ عدد حاصل بر ۲ در a[1] قرار داده می‌شود و خارج قسمت جدید محاسبه می‌شود. این عمل ادامه می‌یابد تا خارج‌قسمت برابر صفر گردد. حال ارقام عدد مورد نظر در مبنای ۲ در خانه‌های متوالی آرایهٔ a قرار دارد که در پایان محتوای آرایهٔ مزبور به عنوان معادل عدد m در مبنای ۲ چاپ می‌شود.

.۵

```
# include<stdio.h>
# define EOL '\n'
main ()
   {
     char text[80] ;
     int tag , count ;
     do
     ++count ;
     while ((text[count] = getchar()) != EOL) ;
        tag = count ;
     count = 0 ;
     do {
            putchar(toupper (text[count])) ;
            ++count ;
       } while (count < tag) ;
  }
```

۶.می‌دانیم که تاس شش‌وجهی همگن است و در هر رویهٔ آن یکی از اعداد ۱ تا ۶ نقش بسته است. پس در هر پرتاب یکی از این ۶ رقم را مشاهده خواهیم کرد. حال می‌خواهیم تجربه کنیم که اگر این تاس را ۶۰۰۰ بار پرتاب کنیم، هریک از ۶ رویهٔ تاس چند بار ظاهر خواهد شد. می‌خواهیم به جای اینکه در عمل تاس را ۶۰۰۰ بار پرتاب کنیم، این کار را با استفاده از تابع rand شبیه‌سازی کنیم.

```
# include <stdio.h>
# include < stdlib.h>
# include < time.h>
# define SIZE 7
main ()
   {
     int face , roll , frequency[SIZE] = {0} ;
     srand (clock()) ;
```

```
    for (roll = 1 ; roll< = 6000 ; roll ++)
      {
        face = rand() % 6 + 1 ;
        + + frequency[face] ;
      }
    printf("%s %17 s \n" , "face" , "frequency") ;
    for (face=1 ; face<=SIZE-1 ; face+ +)
      printf("%4d%17\n" , face , frequency[face]) ;
  }
```

خروجی برنامه

Face	Frequency
1	1037
2	987
3	1013
4	1028
5	952
6	983

.۷

```
# include<stdio.h>
# define size 5
main ()
  {
    int a[size] = (0 , 1 , 2 , 3 , 4) ;
    int i ;
    void modifyarray (int [ ]) ;
    void modifyelement (int) ;
    printf ("%s\n" , "Effects of passing entire array call by reference: \n") ;
    printf ("The values of the original array are:  \n") ;
    for (i=0 ; i<size ; i + +)
      printf("%3d" , a[i]) ;
    printf("\n") ;
    modifyarray(a) ;
    printf("the values of the modified array are: \n") ;
    for (i=0 ; i<=size-1 ; i++)
      printf("%3d" , a[i]) ;
    printf("\n%s\n" ,"Effects of passing array element call by value") ;
    printf("The value of a [3] is %d\n" , a[3]) ;
    modifyelement (a[3]) ;
    prnitf("The value of a[3] is %d\n" , a[3]) ;
  }
```

```
void modifyarray (int b[ ])
   {
     int j ;
     for (j = 0 ; j< = size-1 ; j+ +)
       b[j] * = 2 ;
   }
void modifyelement (int e)
   {
     printf("Value in modifyelement is %d\n" , e* = 2) ;
   }
```

خروجی برنامه

```
Effects of passing entire array call by reference:
The values of the oreiginal array are:
0       1       2       3       4
the values of the modified array are:
0       1       2       3       4
Effects of passing array element call by value:
The value of a[3] is 6
Value in modifyelement is 12
The value of a[3] is 6
```

۸. برنامهٔ قسمت اول به شکل زیر است.

```
# include<stdio.h>
main ()
  {
    int response [99] = { 6 , 7 , 8 , 9 , 8 , 7 , 8 , 9 , 8 , 9 ,
                          7 , 8 , 9 , 5 , 9 , 8 , 7 , 8 , 7 , 8 ,
                          6 , 7 , 8 , 9 , 3 , 9 , 8 , 7 , 8 , 7 ,
                          7 , 8 , 9 , 8 , 9 , 8 , 7 , 7 , 8 , 9 ,
                          6 , 7 , 8 , 7 , 8 , 7 , 9 , 8 , 9 , 2 ,
                          7 , 8 , 9 , 8 , 9 , 8 , 9 , 7 , 5 , 3 ,
                          5 , 6 , 7 , 2 , 5 , 3 , 9 , 4 , 6 , 4 ,
                          7 , 8 , 9 , 6 , 8 , 7 , 8 , 9 , 7 , 8 ,
                          7 , 4 , 4 , 2 , 5 , 3 , 8 , 7 , 5 , 6 ,
                          4 , 5 , 6 , 1 , 6 , 5 , 7 , 8 , 8 , 7 } ;
    int frequency [10] = {0} ;
    int n ;
    void mean (int[ ]) ;
    void median (int [ ]) ;
```

```
    void mode (int [ ] , int [ ]) ;
    mean (response) ;
    median (response) ;
    mode (frequency , response) ;
  }
void mean(int answer[ ])
  {
    int j , total = 0 ;
    printf("%s\n%s\n %s\n" , "* * * * * *" , "Mean" , "* * * * * *") ;
    for (j=0 ; j<=98 ; j + +)
      total + answer [j] ;
    printf ("The mean value for this run is: ")
    printf(" %d/99 = %.4f \n\n" , total , (float) total/99) ;
  }
```

برنامهٔ قسمت دوم به شکل زیر است.

```
void median(int answer[ ])
  {
    int j , pass , hold ;
    printf("\n%s\n%s\n%s\n") , "* * * * * *" "median" , "* * * * * *") ;
    printf("The unsorted array of responses is\n") ;
    for (j%20 = = 0)
      printf("\n") ;
    printf("%2d ", answer[j]) ;
  }
printf("\n\n") ;
  for (pass = 0 ; pass <= 97 ; pass+ +)
    for (j=0 ; j<=97 ; j+ +)
      if (answer[j]>answer[j+1])
        {
          hold = answer[j] ;
          answer[j] = answer[j+1] ;
          answer[j+1] = hold ;
        }
printf("The sorted array is\n") ;
  for (j=0 ; j<=98 ; j + +)
    {
      if (j%20 = = 0)
        printf("\n") ;
      printf("%2d" , answer[j]) ;
    }
printf("\n\n") ;
printf("The median is the 50the element of the sorted 99 element array\n") ;
printf("For this run the median is %d\n\n" , answer[49]") ;
```

برنامهٔ قسمت سوم به شکل زیر است.

```
void mode(int freq[ ] , int answer[ ]) ;
  {
    int rating , j , h , largest = 0 , modevalue = 0 ;
    printf("\n%s\n %s\n%s\n" , "* * * * * *" , "mode" , "* * * * * *") ;
    for (rating = 1 ; rating<=9 ; rating + +)
      freq [rating] = 0 ;
    for (j=0 ; j<=98 ; j+ +)
      + + freq[answer[j] ] ;
    printf("\n%s %11s%19s\n\n" , "Response" , "Frequency" , "Histogram") ;
    for (rating = 1 ; rating<=9 ; rating + +)
      {
        printf("%d %11d" , rating , freq[rating]) ;
        if (freq[rating] > largest)
          {
            largest = freq[rating] ;
            modevalue = rating ;
          }
        for (h=1 ; h<=freq[rating] ; h + +)
        printf("*") ;
        printf ("\n") ;
      }
    printf ("The mode is the most frequent value. \n") ;
    printf ("For this tun the mode is %d" , modevalue) ;
    printf ("which occurred %d times. \n" , largest) ;
  }
```

خروجی برنامه

```
* * * * * *
Mean
* * * * * *
The mean value for this run is:  681 / 99 = 6. 8788
* * * * * *
Median
* * * * * *
The unsorted array of responses is
6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7
The sorted array is
```

```
1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5
5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
The median is the 50th element of the sorted 99 element array
For this run the median is 7
* * * * *
Mode
* * * * * *
Response        Frequency       Histogram
1       1       *
2       3       * * *
3       4       * * * *
4       5       * * * * *
5       8       * * * * * * * *
6       9       * * * * * * * * *
7       23      * * * * * * * * * * * * * * * * * * * * * * *
8       27      * * * * * * * * * * * * * * * * * * * * * * * * * * *
9       19      * * * * * * * * * * * * * * * * * * *
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

توضیح. الف) در برنامهٔ مورد نظر، پاسخهای داده شده از ۹۹ نفر، به‌صورت مقداردهی اولیه، به آرایه response نسبت داده شده است، سپس با فراخوانی تابع mean، میانگین این مقادیر محاسبه و چاپ می‌گردد (قسمت اول).

ب) سپس تابع median فراخوانی شده است. در این تابع، مقادیر مزبور به همان صورت در هر سطر ۲۰ عنصر چاپ می‌گردد که این کار با حلقهٔ for در تابع مزبور انجام می‌گیرد. سپس برای اینکه بتوان میانه را تعیین کرد، آرایهٔ مزبور به صورت صعودی مرتب می‌گردد که منطق این کار نیز واضح است. دوباره عناصر آرایه در هر سطر ۲۰ عنصر چاپ می‌گردد و در پایان کار نیز میانهٔ به دست آمده که مقدار آن ۷ است چاپ می‌شود. (قسمت دوم).

ج) سپس تابع mode فراخوانی شده است. در این تابع فرکانس یا تواتر تکرار هر مقدار از پاسخها (بین ۱ تا ۱۰) به دست می‌آید و نمودار میله‌ای آن نیز رسم می‌گردد که منطق این

کار را در مثالهای قبلی ملاحظه کردید. در ضمن، مد مقادیر، یعنی عنصری که بیشتر از بقیه عناصر تکرار شده است، تعیین می‌گردد که برای مقادیر داده شده در این تمرین، عنصر مورد نظر، عنصر ۸ با تعداد ۲۷ بار تکرار است.

.۹

```
# include < stdio.h>
# define SIZE 100
main ()
   {
    int a [SIZE] , x , searchkey , element ;
    int linearsearch (int [ ] , int , int) ;
    for (x = 0 ; x<SIZE ; x + +)
       scanf ("%d" , &a [x]) ;
    printf ("Enter integer search key: \n") ;
    scanf ("%d" , &searchkey) ;
    element = lineanrsearch (a , searchkey , SIZE) ;
    if (element ! = -1)
       printf ("Found value in element %d\n" , element) ;
    else
       printf ("Value not found\n") ;
   }
int linearsearch(int array[ ] , int key , int size)
   {
    int n ;
    for (n = 0 ; n < size ; n + +)
      if (array [n] = = key)
         return n ;
    return -1 ;
   }
```

.۱۰

```
# include<stdio.h>
main ()
   {
    int a[8][8] , b[8][8] , c[8][8] , m , n , i , j ;
    scanf("%d %d" , &m , &n) ;
    for (i = 0 ; i<m ; + + I)
      for (j = 0 ; j<n ; + + j)
        scanf("%d" , &a[i][j]) ;
    for (i = 0 ; i<m ; + + i)
```

```
        for (j = 0 ; j<n ; + + j)
            scnaf("%d" , &b[i][j]) ;
      for (i = 0 ; i<m ; + + i)
        for (j = 0 ; j<n ; + + j)
            c[i][j] = a[i][j] + b[i][j] ;
      for (i = 0 ; i<m ; + +i)
        {
            printf ("\n") ;
            for (j = 0 ; j<n ; + + j)
               printf ("%5d" , c[i][j]) ;
        }
  }
```

.۱۱

```
# include <stdio.h>
maint ()
   {
      int a[7][7] , b[7[[7] , c[7][7] ;
      int i , j , k , m , n , h ;
      scanf ("%d %d %d" , &m , &n , &h) ;
      for (i=0 ; j<m ; + + i)
        for (j=0 ; j<n ; + + j)
            scanf ("%d" , &a[i][j]) ;
      for (i=0 ; i<m ; + + i)
        for (j=0 ; j<h ; + + j)
            scanf ("%d" , &b [i][j]) ;
      for (i=0 ; i<m ; + + i)
        for (j=0 ; j<h ; + + j)
            {
              c[i][j] = 0 ;
              for (k=0 ; k<n ; + + k)
                 c[i][j] = c[i][j] + a[i][k] * b[k][j] ;
            }
      for (i=0 ; i<m ; + + i)
        {
            printf ("\n") ;
            for (j=i ; j<h ; + + j)
               printf ("%5d" , c[i][j]) ;
        }
   }
```

۱۲. در این برنامه چنانچه خروجی آن برای حالتی که کاراکترهای ورودی آن رشتهٔ Payam Noor University باشد، به صورت زیر است. ملاحظه می‌کنید که پس از خواندن کاراکترهای مورد نظر به آرایهٔ sentence، علامت پایان رشته، یعنی 0\، نیز به دنبال آن افزوده شده است.

```
# include <stdio.h>
main ()
   {
     char c , sentence[80] ;
     int i = 0 ;
     printf (Enter a line of text:  \n") ;
     while ((c = getchar()) !='\n') ;
        sentence[i + +] = c ;
     sentence[i] = '\0' ;
     printf ("\n The line entered was:  \n") ;
     puts (sentence) ;
   }
```

خروجی برنامه

```
Enter a line of fext:
This is a test
The line entered was:
 This is a test:
```

۱۳.

```
int len(s)
char s [ ] ;
   {
     int count = 0 ;
     while (s[count] !=`\0')
        count + + ;
     return (count) ;
   }
```

۱٤.

```
void strcat(S1, S2)
char S2 [ ] , S2 [ ] ;
   {
     int i , j ;
     for (i = 0 ; S1 [i] ! = '\0' ; + + i);
```

```
    for (j = 0 ; S2 [j] ! = '\0' ; S1 [i + +] S2 [j]);
}
```

در زیر راه دیگری برای الحاق یا مجاورسازی دو رشته نشان داده شده است که در اینجا با استفاده از تابع کتابخانه‌ای strlen طول رشتهٔ اول به دست می‌آید و سپس رشتهٔ دوم به دنبال رشتهٔ اول درج می‌شود.

```
void strcat (S1 , S2)
char S1 [ ] , S2 [ ] ;
  {
    int i ;
    i = strln (S1) ;
    for (j=0 ; S2 [j]! = '\0' ; S1 [+ +i] = S2 = [j]) ;
  }
```

.۱۵

```
int strops (S1 , S2)
char S1[ ] , S2[ ] ;
  {
    int len1 , len2 , i , j1 , j2 ;
    len1= strlen(S1) ;
    len2 = strlen(S2) ;
    for (i=0 ; i + len2<= len1 ; i + +)
      for (j1= i , j2 = 0 ; j2<len && S1[j1] = = S2[j2] ; j1 + + , j2 + +)
        if (j2 = = len2)
          return(i) ;
    return (-1) ;
  }
```

.۱۶

```
void substr(S1 , S2 , i , j)
char S1[ ] , S2[ ] ;
int i , j ;
  {
    int k , m ;
    for (k = i , m = 0 ; m<j ; S2 [m+ +] = S1[k+ +]) ;
      S2[m] = '\0' ;
  }
```

.۱۷

```
# include <stdio.h>>
# include < stdlib.h>
main ()
   {
     int i , n = 0 ;
     char x[10][12] ;
     void reorder (int n , char x[ ] [12]) ; /* function prototype */
     printf ("Enter each string on a separate line below\n\n") ;
     printf ("type 'END' when finished\n\n") ;
                                          /* read in the list of strings */
     do {
           printf ("string %d: " , n+1) ;
           scanf ("%s" , &x [n]) ;
         } while (strcmpi (x[n+ +] , "END")) ;
                                          /* reorder the list of strings */
     reorder (--n , x) ;
                                          /* display the reordered list of strings */
     printf("\n\n reordered list of strings:  \n") ;
     for (i = 0 ; i<n ; + +i)
       printf ("string %d:  %s" , i +1 , x[i]) ;
   }
void reorder(int n , char x[ ][12]) /* rearrange the list of strings */
   {
     char temp[12] ;
     int i , item ;
     for (item = 0 ; item<n-1 ; + + item)
                                     /* find the lowest of all remaining strings */
       for (i = item+1 ; i<n ; + + i)
         if (strcmpi (x[item] , x [i]) > 0)
           { /* interchange the two strings */
             strcpy (temp , x[item]) ;
             strcpy (x[item] , x[i]) ;
             strcpy (x[i] , temp) ;
           }
     return ;
   }
```

خودآزمایی ۸

.۱

```
int len(s)
char *s ;
   {
     char *t ;
     t = s ;
     while (*t! = `\0')
        t + + ;
     return (t-s) ;
   }
```

در اینجا از دو اشاره‌گر استفاده شده است. اشاره‌گر اول به ابتدای رشته اشاره می‌کند. اشاره‌گر دوم از ابتدا تا انتهای رشته را پیمایش می‌کند. در انتها تفاضل دو اشاره‌گر طول دو رشته را مشخص می‌کند.

.۲

```
include<stdio.h>
main ()
  {
     char line[80] ;
     int vowel = 0 ;
     int constant = 0 ;
     int digit = 0 ;
     int whitespace = 0 ;
     void scanline (char line[ ] , int *pv , int *pc , int *pd , int *pw);
     printf("Enter a line of text \n") ;
     scanf("%[^\n]", line) ;
     scanline (line , &vowel , &constant , &digit , &whitespace) ;
     printf("vowel = \n" , vowel) ;
     printf("constant = \n" , constant) ;
     printf("digit = \n" , digit) ;
     printf("whitespace = \n" , whitespace) ;
  }
void scanline (char line[ ] , int *pv , int *pc , int *pd , int *pw)
  {
     char c ;
     int count = 0 ;
     while ((c = toupper(line[count])) != '\0 ')
```

```
    {
      if (c = ='A' || c = = 'E' || c = = 'I' || c = = '0' || c = = 'u')
        ++ *pv ;
      else if (c >= 'A' && c<= 'Z')
        ++ *pc ;
      else if (c>= '0' && c <= '9')
        ++ *pd ;
      else if (c = = ' ' || c = = '\t')
        ++ *pw ;
      ++count ;
    }
  return ;
}
```

.۳

```
Strcmp (char *s1 , char *s2)
{
  while (*s1)
    if (*s1 - *s2)
      return *s1 - *s2 ;
    else
      {
        s1 + + ;
        s2 + + ;
      }
  return `\0´ ;
}
```

در اینجا دو اشاره‌گر s1 و s2 به ترتیب آدرس دو رشته را دریافت می‌کنند؛ یعنی در آغاز، آدرس اولین خانهٔ دو آرایهٔ مربوط به دو رشتهٔ مورد نظر را دارند. در دستور if محتوای این دو خانه با یکدیگر مقایسه می‌شوند. چنانچه یکسان نباشند، تفاضل آن دو (یعنی تفاضل کد اسکی کاراکترهای محتوای این دو خانه) مساوی صفر نخواهد بود. پس دو رشته یکسان نیستند. درنتیجه، تابع مزبور عددی غیرصفر (تفاضل s1 و s2) را که معرف true است برمی‌گرداند. در غیر این صورت دستورهای مربوط به else اجرا می‌شود و مقدار هر دو اشاره‌گر یک واحد افزایش می‌یابد. در نتیجه آدرس کاراکتر بعدی از دو رشته را خواهند داشت. حال اگر حلقهٔ while کامل گردد، یعنی عمل مقایسه کاراکترهای دو رشته تا آخرین کاراکتر آنها که '\0` است ادامه یابد، پس دو رشته یکسان‌اند و تابع مزبور '\0` را که کد اسکی

آن صفر است به عنوان معرف false برمی‌گرداند (زیرا به طوری که در فصلهای پیش گفتیم در زبان C، مقدار صفر، معرف false و غیرصفر معرف true است.)

٤.

```
void Concat (s1 , s2 , s3)
char *s1 , *s2 , *s3 ;
   {
     while (* s1! = `\0')
       *s3++ = *s1++ ;
     while (* s2! = `\0')
       *s3++ = *s2++ ;
     *s3 = `\0'
   }
```

توضیح. s1، s2 و s3 سه اشاره‌گرند که به ترتیب آدرس اولین بایت و رشتهٔ s1 و s2 و همچنین رشتهٔ سوم را که از الحاق آن دو به دست خواهد آمد دارند. در داخل حلقهٔ while اول، محتوای رشتهٔ اول تا موقعی که انتهای رشته (یعنی کاراکتر'\0`) ظاهر نشده است، در رشتهٔ سوم که با s3 به آن اشاره شده است قرار می‌گیرد. سپس در حلقهٔ while دوم، محتوای رشتهٔ دوم به دنبال آن قرار می‌گیرد. در پایان، علامت پایان رشته، یعنی '\0`، در انتهای آن قرار می‌گیرد.

٥.

```
# include <stdio.h>
main ()
   {
    int i , n , *x ;
    void reorder (int n , int *x) ;
    printf ("\n How many numbers will be entered?") ;
    scanf ("\%d" , &n) ; /* read in a value for n */
    printf ("\n") ;
    x = (int *) malloc (n * sizeof (int)) ; /* allocate memory */
    for (i=0 ; i<n ; + +i) /* read in the list of numbers */
       {
        printf ("i=%d x=" , i+1) ;
        scanf ("%d" , x + i) ;
       }
    reorder (n , x) ;
```

```
    print ("n\n\ reordered list of numbers:  \n\n") ;
      for(i=0 ; i<n ; + + i)
        printf("i=%d x=%d\n" , i+1 , *(x+i)) ;
  }
void reorder(int n , int *x) /* rearrange the list of numbers */
  {
    int i , item , temp ;
    for (item = 0 ; item<n-1 ; + + item)
      for (i=item+1 ; i<n ; + + i)
        if (*(x+i) < *(x + item))
          {
            temp = *(x + item) ;
            *(x+item) = *(x+i) ;
            *(x+i) = temp ;
          }
    return ;
  }
```

توضیح. در این برنامه، آرایه‌ای که عناصر آن از نوع int است، با استفاده از اشاره‌گر به مقادیر صحیح تعریف شده است. حافظهٔ لازم برای عناصر آن از طریق تابع کتابخانه‌ای malloc به صورت پویا از سیستم اخذ شده و آدرس آغاز آن در متغیر اشاره‌گر (متغیر x) قرار داده شده است. در تمامی طول تابع اصلی و تابع فرعی برای مراجعه به عناصر آرایه (جهت خواندن، نوشتن، مقایسه، و جابه‌جایی) از اشاره‌گر استفاده شده است؛ مثلاً ملاحظه می‌کنید که تابع scanf به آدرس i اُمین عنصر به جای &x[i] با x + I مراجعه می‌کند. همچنین تابع printf برای چاپ مقدار i اُمین عنصر، به جای x[i] از *(x + i) استفاده کرده است.

در تابع فرعی reorder نیز آرگومان دوم به جای یک آرایه از نوع مقادیر صحیح، یک متغیر اشاره‌گر به مقدار صحیح معرفی شده است و در درون آن برای مراجعه به مقدار هر عنصر i اُم آرایه به جای x[i] از *(x + i) استفاده شده است.

٦. این برنامه را در آرایه‌ها دیدیم. در اینجا هدف آن است که هر کدام از آرایه‌های دوبعدی به‌صورت اشاره‌گر به مجموعه آرایهٔ یک بعدی تعریف شود. در مراجعه به عناصر آرایه‌ها نیز به همین روش از اشاره‌گرها استفاده می‌شود. در ضمن هر قسمت از عملیات برنامه به کمک تابع فرعی انجام می‌گیرد.

```
# include <stdio.h>
```

```
# define maxcols 30
main ()
  {
    int nrows , ncols ;
    int (*a)[maxclos] , (*b)[maxcols] , (*c)[maxcols] ; /* pointer defination */
        /* function prototypes */
    void readinput (int (*a)[maxcols] , int nrows , int ncols) ;
    void computesums (int (*a)[maxcols] , int (*b)[maxcols]
   int (*c)[maxcols] , int nrows , int ncols) ;
    void writeoutput (int (*c)[maxcols] , int nrows , int ncols) ;
    printf ("how many rows?") ;
    scanf ("%d" , & nrows) ;
    printf ("How many columns?") ;
    scanf ("%d" , & ncols) ;
        /* allocate initial memory */
    *a = (int *) malloc (nrows * ncols * sizeof (int)) ;
    *b = (int *) malloc (nrows * ncols * sizeof (int)) ;
    *c = (int *) malloc (nrows * ncols * sizeof (int)) ;
    printf ("\n\nfirst table: \n") ;
    readinput (a , nrows , ncols) ;
    printf ("\n\nsecond table: \n") ;
    readinput (b , nrows , ncols) ;
    computesums (a , b , c , nrows , ncols) ;
    printf ("\n\nsums-of the elements:  \n\n") ;
    writeoutput (c , nrows , ncols) ;
  }
void readinput (int (*a)[maxcols], int m, int n)
  { /* read in a table of integers */
    int row , col ;
    for (row = 0 ; row<m ; + + row)
      {
        printf ("\nenter data for row no. %2d\n" , row + 1) ;
        for (col = 0 ; col<n ; + + col)
          scanf ("%d" , (*(a+row) + col)) ;
      }
    return ;
  }
void computesums (int(*a)[maxcols] , int (*b)[maxcols] , int (*c)[maxcols] , int m ,
                                                                          int n)
   {
    int row , col ;
    for (row = 0 ; row<m ; + + row)
      for (col = 0 , col<n ; + + col)
```

```
            *(*(c+row) + col) = *(*(a+row) + col) + *(*(b+row) + col) ;
    return;
  }

void writeoutput (int(*a) [maxcols] , int m , int n)
  { /* write out a table of integers */
    int row , col ;
    for (row = 0 ; row<m ; + +row)
      {
        for (col = 0 ; col<n ; + + col)
          printf("%4d" , *(*(a+row) + col)) ;
        printf ("\n") ;
      }
    return ;
  }
```

توضیح. در این برنامه، a، b و c اشاره‌گرهایی به گروهی از آرایه‌های یک بعدی پیوسته (مجاور هم) تعریف شده‌اند که بزرگی همهٔ آنها maxcols است. در توصیف توابع در تابع اصلی و همچنین توصیف آرگومانها در درون توابع، آرایه‌ها به همین طریق معرفی شده‌اند. چون a، b و c به جای آرایه اشاره‌گر معرفی شده‌اند، باید برای هر آرایه با استفاده از تابع کتابخانه‌ای malloc حافظه اختصاص یابد که این کار در تابع اصلی انجام شده است. برای مثال دستور زیر را درنظر بگیرید.

```
*a = (int *) malloc (nrows * ncols * sizeof (int)) ;
```

در این دستور a* به اولین عنصر در آرایهٔ اول اشاره می‌کند. به طریق مشابه، (a+1)* به اولین عنصر در آرایهٔ دوم و همین طور (a+2)* به اولین عنصر در آرایهٔ سوم اشاره می‌کند. بنابراین یک بلوک حافظه که بزرگی آن nrows * ncols برای مقادیر صحیح است، با شروع از اولین عنصر آرایه اختصاص می‌یابد. نحوهٔ عمل دو آرایه دیگر نیز به همین طریق است.

هر یک از عناصر آرایه با استفاده از عملگر غیرمستقیم به صورت تکراری پردازش می‌شود. برای مثال در تابع readinput به هر عنصر تابع به صورت زیر مراجعه می‌شود.

```
scanf("%d" , (*(a+row)+col)) ;
```

به طریق مشابه عمل جمع کردن عناصر متناظر دو آرایه در تابع computesums به صورت زیر نوشته می‌شود.

```
*(*(c+row) + col) = *(*(a+row) + col) + *(*(b+row) + col) ;
```

همچنین اولین دستور printf در تابع writeoutput به صورت زیر نوشته شده است.

printf ("%4d" , *(*(a+row) +col)) ;

به هرحال می‌توانیم در درون توابع، از روش متعارف در مورد برخورد با عناصر آرایه استفاده کنیم. بنابراین در تابع readinput می‌توانیم برای خواندن مقادیر عناصر آرایه، به جای دستور scanf("%d" , (*(a+row) + col)) ; دستور زیر را به کار بریم.

scanf("%d" , &a[row][col]) ;

به طریق مشابه در تابع computesums می‌توانیم به جای دستور

((c+row)+col) = *(*(a+row)+col) + *(*(b+row)+col) ;

دستور زیر را به کار بریم.

c[row][col] = a[row][col] + b[row][col] ;

همین طور در تابع writeoutput می‌توانیم به جای دستور

printf("%4d" , *(*(a+row) + col)) ;

دستور زیر را به کار بریم.

printf ("%4d" , a[row][col]) ;

۷. برنامهٔ زیر فهرستی از رشته‌ها را براساس این روش به ترتیب الفبا مرتب می‌کند. تعداد رشته‌ها مشخص نیست. ولی حداکثر ۱۰ رشته درنظر گرفته شده و پایان کار با رشتهٔ "end" مشخص شده است. یعنی هر موقع عمل ورود یا خوانده شدن رشته‌ها تمام شد، کلمهٔ "end" را در پایان ورود داده‌ها وارد می‌کنیم. آرایهٔ رشته‌ای x[] برای نگهداری آدرس رشته‌ها منظور شده است. رشته‌ها در تابع اصلی خوانده می‌شوند که برای این کار با استفاده از تابع malloc، حافظهٔ مورد نیاز به صورت پویا اختصاص داده می‌شود. سپس با فراخوانده شدن تابع فرعی reorder (که آرگومانهای آن تعداد رشته‌ها و آرایهٔ آدرس رشته‌هاست)، رشته‌های مورد نظر به ترتیب الفبا مرتب می‌گردد. پس از آن نتیجه در تابع اصلی چاپ می‌شود.

```
# include <stdio.h>
# include <stdio.h>
main ()
  {
    int i , n = 0 ;
    char *x[10] ;
    int reorder (int n , char *x[ ]) ;
    printf ("enter each string on a separate line below\n\n") ;
    printf ("type "end" when finished\n\n") ;
    do { /* read in the list of string */
```

```
            x[n] = malloc(12 * sizeof (char)) ; /* allocate memory */
            printf("%s" , x [n]) ;
            scanf("%s" , x [n]) ;
          }
      while (strcmpi (x[n+ +] , "end")) ;
        reorder(--n , x) ;
      printf("n\nreordered list of strings:  \n") ;
                    /* display the reordered list of strings */
      for (i = 0 ; i<n ; + + i)
        printf("\nstring %d%s" , i+1 , x[i]) ;
   }
reorder (int n , char *x[ ])          /* rearrange the list of strings */
   {
     char *temp ;
     int i , item ;
     for (item = 0 ; item<n-1 + + item)
       for (i = item+1 ; i<n ; + +i)
         if (strcmpi (x[item] , x[i] >0)
            {
              temp = x[item] ;
              x[item] = x[i] ;
              x[i] = temp ;
            }
     return ;
  }
```

۸.

```
void swap (int *a , int *b)
   {
     int temp ;
     temp = *a ;
     *a = *b ;
     *b = temp ;
  }
```

۹.

```
# include<stdio.h>
main ()
   {
     char str[80] , ch ;
     int count = 0 ;
     printf ("enter string for search:  \n") ;
```

```
    gets(str) ;
    printf ("enter a character") ;
    ch = getchar() ;
    counting (str , ch , &count) ;
    printf ("%s %c %d" , str , ch , count) ;
  }
void counting (char *s , char ch , int *c)
  {
    while (*str)
      if (*s++ = = ch)
        *c++ ;
    return() ;
  }
```

.۱۰

۱. P اشاره‌گری به مقداری صحیح است.
۲. P آرایه‌ای ۱۰ عنصری از اشاره‌گرها به مقادیر صحیح است.
۳. P اشاره‌گری به آرایه‌ای ۱۰ عنصری با مقادیر صحیح است.
٤. P تابعی است که اشاره‌گر به مقدار صحیح برمی‌گرداند.
۵. P تابعی است که آرگومانی را که اشاره‌گری به کاراکتر است می‌پذیرد و مقدار صحیح برمی‌گرداند.
٦. P تابعی است که آرگومانی را که اشاره‌گری به کاراکتر است می‌پذیرد و اشاره‌گر به مقدار صحیح برمی‌گرداند.
۷. P اشاره‌گری به تابع است که آرگومانی را که آن هم اشاره‌گری به کاراکتر است می‌پذیرد و مقدار صحیح برمی‌گرداند.
۸. P تابعی است که آرگومانی را که آن هم اشاره‌گری به کاراکتر است می‌پذیرد و اشاره‌گری به آرایه‌ای ۱۰ عنصری از نوع مقادیر صحیح برمی‌گرداند.
۹. P تابعی است که آرگومانی را که آن هم اشاره‌گری به آرایه‌ای کاراکتری است می‌پذیرد و مقدار صحیح برمی‌گرداند.
۱۰. P تابعی است که آرگومانی را که آن هم آرایه‌ای از اشاره‌گرها به کاراکترهاست می‌پذیرد و مقدار صحیح برمی‌گرداند.

۱۱. P تابعی است که آرگومانی را که آرایه‌ای کاراکتری است می‌پذیرد و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۱۲. P تابعی است که آرگومانی را که آن هم اشاره‌گری به آرایه‌ای کاراکتری است می‌پذیرد و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۱۳. P تابعی است که آرگومانی را که آن هم آرایه‌ای از اشاره‌گرها به کاراکترهاست می‌پذیرد و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۱٤. P اشاره‌گری به تابعی است که آن هم آرگومانی می‌پذیرد که اشاره‌گری به آرایه‌ای کاراکتری است و مقدار صحیح برمی‌گرداند.
۱۵. P اشاره‌گری به تابعی است که آن هم آرگومانی می‌پذیرد که اشاره‌گری به آرایه‌ای کاراکتری است و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۱٦. P اشاره‌گری به تابعی است که آن هم آرگومانی را می‌پذیرد که آرایه‌ای از اشاره‌گرها به کاراکترهاست و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۱۷. P آرایه‌ای ۱۰ عنصری از اشاره‌گرهایی به توابع است که هرکدام از آن توابع مقداری صحیح برمی‌گرداند.
۱۸. P آرایه‌ای ۱۰ عنصری از اشاره‌گرهایی به توابع است که هر کدام از آن توابع آرگومانی که کاراکتر است می‌پذیرند و مقدار صحیح برمی‌گرداند.
۱۹. P آرایه‌ای ۱۰ عنصری از اشاره‌گرهایی به توابع است که هرکدام از آن توابع آرگومانی که کاراکتری است می‌پذیرد و اشاره‌گری به مقدار صحیح برمی‌گرداند.
۲۰. P آرایه‌ای ۱۰ عنصری از اشاره‌گرهایی به توابع است که هرکدام از آن توابع آرگومانی که اشاره‌گری به کاراکتر است می‌پذیرد و اشاره‌گری به مقدار صحیح برمی‌گرداند.

خودآزمایی ۹

.۱

```
struct date {
        int month ;
        int day ;
        int year ;
```

```
            } ;
struct student {
                    long int no ;
                    char name [20] ;
                    struct date d ;
                  } ;
struct student a ;
main ()
  {
    a. no = 123456 ;
    strcpy (a. name , "Amiri") ;
    a. d. month = 11 ;
    a. d. day = 5 ;
    a. d. year = 1980 ;
  }
```

همانطور که مشاهده می‌کنید برای دستیابی به فیلدهای متداخل از دو علامت نقطه استفاده می‌شود.

نکتهٔ ۱. برای مقداردهی اولیه به ساختار متداخل از دستور زیر استفاده می‌کنیم.

```
struct student b = {359672 , "Ahmadi" , 8 , 23 , 1979} ;
```

که در آن 8 در فیلد ماه، 23 در فیلد روز و 1979 در فیلد سال ذخیر می‌شوند.

نکتهٔ ۲. متغیر d که از نوع struct date است ٦ بایت فضا می‌گیرد. پس کل فضایی که متغیر a در ساختار فوق اشغال می کند برابر است با ۳۰ = ٦ + ۲۰ + ٤ بایت.

.۲

```
# include < stdio. h>
# include < stdlib. h>
struct phone {
                  long int no ;
                  char name[20] ;
                } ;
main ()
  {
    int n , i , k ;
    struct phone x[100] ;
    char str [20] ;
    printf ("How many name ? ") ;
    scanf ("%d" , &n) ;
    for (i = 0 ; i < n ; i + +)
```

```
        {
          printf ("Enter name and phone number:  ") ;
          scanf ("%s %ld" , x[i]. name , &x[i]. no) ;
        }
    while (1) {
      printf ("\n 1-Enter name \n") ;
    printf ("2-Exit \n") ;
    scanf ("%d" , &k) ;
    if (k= = 1)
        {
         printf ("Enter name": ) ;
         scanf ("%s" , str) ;
         for (i=0 ; i<n ; i+ +)
           if (!strcmpi (x[i]. name , str))
               {
                 printf ("phone number:  %d \n" ; x[i]. no) ;
                 goto m1 ;
               }
          printf ("\n not exist") ;
         m1:
        }
    if (k= = 2)
       exit(0) ;
  } /* end while */
```

.۳

خروجی برنامه

```
65 A 65 A
```

نکته. دستور p-> c معادل (*p). c است.

.٤

```
#include<stdio.h>
#include<math.h>
typedef union {
                    float fexp ; /* floating point exponent */
                    int nexp ; /* integer point exponent */
                  } nvals ;
```

```
typedef struct {
                    float x ;
                    char flag ;
          /* 'f' if exponent is floating point and 'i' if exponent is integer */
                    nvals EXP ;
                  }
main()
  {
     values A ;
     float power(values A) ;
     int i ;
     float n , y ;
     printf("y = x^n \n Enter a value for x:  ") ;
     scanf("%f" , &A.x) ;
     printf("\n Enter a value for n:  ") ;
     scanf("%f" , &n) ;
     i = (int) n ;
     A.flag = (i = = n) ? 'i':  'f' ;
     if (A.flag = = 'i')
       A.EXP.nexp = i ;
     else
       A.exp.fexp = n ;
     if (A.flag = = 'f' && A.x <= 0.0)
       printf(" \n ERROR ") ;
     else {
            y = power(A) ;
            printf(" \n y = %.4f" , y) ;
          }
    float power(values A)
      {
        int i ;
        float y = A.x ;
        if (A.flag = = 'i') /* integer exponent */
          { if (A.exp.nexp = = 0)
              y = 1.0 ; /* zero exponent */
            else {
              for (i = 1 ; i<abs(A.exp.nexp) ; ++i)
                y *=A.x ;
              if (A.exp.nexp < 0)
                y = 1/y ; /* negative integer exponent */
            }
      }
     else /* floating point exponent */
       y = exp(A.exp.fexp * log(A.x)) ;
```

```
    return(y) ;
  }
```

توجه کنید که تعریف ساختارها و یونیونها نسبت به تمام توابع برنامه به صورت خارجی است اما متغیر ساختار A درون هر دو تابع متغیر محلی تعریف شده است.

خودآزمایی ۱۰

.۱

```
#include<stdio.h>
#include<conio.h>
main(void)
  {
    FILE *fp ;
    int fact , n ;
    clrscr() ;
    scanf("%d",&n) ;
    fact = 1 ;
    while (n>1)
      fact *= n--;
    if ((fp = fopen("fact.txt", "w"))== NULL)
      {
       fprintf(stderr, "Cannot open input file.\n") ;
       return 1 ;
      }
    fprintf(fp,"%d",fact) ;
    fclose(fp) ;
    return 0 ;
  }
```

توضیح. ابتدا عددی صحیح از ورودی خوانده می‌شود و در متغیر n جای می‌گیرد. سپس با استفاده از حلقهٔ while فاکتوریل آن محاسبه و در متغیر fact جایگزین می‌شود. حال فایلی جهت نوشتن با نام fact. txt ایجاد و با دستور fprintf مقدار فاکتوریل در آن نوشته می‌شود.

.۲

```
#include <stdio.h>
main ()
```

```
{
   FILE *fp ;
   char *msg ;
   int i = 0 ;
   fp = fopen ("str. txt" , "w") ;
   gets (msg) ; // read string
   while (msg[i])
     putc (msg[i++] , fp) ; // write string to file
   while (i >= 0)
     putc(msg[i--] , fp); // write invert of string to file
   fclose(fp) ;
   return 0;
}
```

توضیح. رشتهٔ ورودی، در آرایه‌ای کاراکتری به شکل اشاره‌گر و به نام msg* قرار می‌گیرد. سپس با استفاده از حلقهٔ while و تابع putc یکبار از ابتدا تا انتها و بار دیگر از انتها به ابتدا در فایلی متنی به نام str نوشته می‌شود.

.۳

```
#include <string.h>
#include <stdio.h>
main()
{
   FILE *fp ;
   char str[ ] = "0123456789" ; // an array with 10 character
   if ((fp = fopen("test.dat", "w")) = = NULL)
     {
       fprintf (stderr, "Cannot open input file. \n") ;
       return 1 ;
     }
   fwrite(&str , strlen(str) , 1 , fp) ; // create a file containing 10 bytes
   fclose(fp) ; // close the file
   return 0 ;
}
```

توضیح. از آنجایی که هرکاراکتر یک بایت حافظه اشغال می‌کند، پس کافی است فایلی متنی ایجاد کنیم و ۱۰ کاراکتر در آن بنویسیم. در این برنامه آرایه‌ای به نام str با ۱۰ کارکتر شامل اعداد ۰ تا ۹ مقداردهی می‌شود. سپس فایل متنی به نام test. dat همراه با پیغام مناسب جهت حصول اطمینان از باز شدن آن ایجاد می‌کنیم و با دستور fwrite محتوای str را در آن

می نویسیم. فایل ایجاد شده ۱۰ بایت حجم دارد.

٤.

```
# include <stdio.h>
# include <stdlib.h>
void main (int argc , char *argv[ ])
  {
    FILE *in ;
    char ch ;
    int openbraket = 0 , closebraket = 0 ;
    int openparant = 0 , closeparant = 0 ;
    if (argc! = 2)
      {
        printf ("\n The number of argument is incorrect. ") ;
        exti (0) ;
      }
    in = fopen (argv [1] , "r") ;
    if (in = = NULL)
      {
        printf ("\n file cannot open. ") ;
        exit(0) ;
      }
    ch = getchar() ;
    while (ch ! = EOF)
      {
        if (ch = = '{')
           openbraket + + ;
        else if (ch = = ' }')
           closebraket + + ;
        else if (ch = = '(')
           openparant + + ;
        else if (ch = = ') ')
           closparant + + ;
        ch = getc (in) ;
      }
    printf ("\n the number of open braket is:  %d" , openbraket) ;
    printf ("\n the number of close braket is:  %d ", closebraket) ;
    printf ("\n the number of open parantheses is:  %d" , openparant) ;
    printf ("\n the number of close parantheses is:  %d" , closeparant ") ;
    fclose (in) ;
  }
```

۵.

```
# include <stdio. h>
# include <stdlib. h>
main ()
  {
    FILE *p ;
    int i ;
    char str[10] ;
    struct employee
      {
        char name [10] ; // name of employee
        int hp ; // hour pay
        int h ; // total hours
      } emp ;
    clrscr() ; // clear the screen
    p = fopen ("employ.dat" , "w+b") ; // open a binary file for input and output
    if (p = = NULL)
      {
        printf ("cannot open file") ;
        printf ("\n press any key …") ;
        getch() ;
        exit (1) ;
      }
    gotoxy (14 , 2) ; // cursor position
    puts(" INPUT ") ;
    gotoxy (3 , 3) ;
    printf (" name          hour pay total hours ") ;
    gotoxy (3 , 4) ;
    printf ("-----------     ----------- -------------") ;
    i = 5 ;
    while (1)
      {
        gotoxy (3 , i) ;
        gets (emp.name) ;
        if (!emp.name [0])
          break ;
        gotoxy(21 , i) ;
        gets(str) ; // input name
        emp.hp = atoi (str) ; //convert string type to integer
        gotoxy(34 , i) ;
        gets(str) ; // input total hours
        emp.h = atoi(str) ; //convert string type to integer
        i + + ;
```

```
        fwrite (&emp , sizeof (struct employee) , 1 , p) ;
      }
   rewind (p) ; // return to beginning of the file
   clrscr() ;
   gotoxy(7 , 2) ;
   puts (" OUTPUT ") ;
   gotoxy(3 , 3) ;
   puts (" name            salary") ;
   gotoxy(3 , 4) ;
   puts ("------------      -----------") ;
   i = 5 ;
   fread (&emp , sizeof (struct employee) , 1 , p) ;
   while (!feof (p))
      {
        gotoxy (3 , i) ;
        puts (emp.name) ;
        gotoxy (21 , i) ;
        printf ("%d" , emp.hp * emp.h) ; // calculate and print salary
        i + + ;
        fread (&emp , sizeof (struct employee) , 1, p) ;
      }
   getch() ;
}
```

خروجی برنامه

```
                                    INPUT
name                hour pay        total hour
-------------------------------------------------
amir                200             180
hamid               300             150
majid               250             100
```

```
                        OUTPUT
name                    Salary
-----------------------------------
amir                    36000
hamid                   45000
majid                   25000
```

.٦

```
#include <stdio.h>
main()
  {
    FILE *in , *out ;
    char ch ;
    if ((in = fopen("input", "rb")) = = NULL)
      {
        printf ("Cannot open input file. \n") ;
        exit(1) ;
      }
    if ((out = fopen("input", "wb")) = = NULL)
      {
        printf ("Cannot open output file. \n") ;
        exit(1) ;
      }
    while (!feof(in))
      {
        ch = getc(in) ;
        if (!feof (in)
          putc(ch , out) ;
      }
    fclose(in) ; // close the input file
    fclose(out) ; // close the output file
  }
```

در این برنامه ملاحظه می‌کنید که فایل مبدأ فایل ورودی و فایل خروجی نسخهٔ کپی شده در نظر گرفته شده است. فایلها در حالت باینری باز شده‌اند و تابع feof نیز برای کنترل و تشخیص پایان فایل به کار رفته است. در این برنامه داده‌ها کاراکتر به کاراکتر از فایل ورودی خوانده و در فایل خروجی نوشته می‌شود. همین برنامه را می‌توان به گونه‌ای نوشت که داده‌ها خط به خط خوانده و سپس نوشته شوند.

```
#include <stdio.h>
#include <stddef.h>
#define size 100
int COPY(input , output)
char *input , *output ;
  {
    FILE *in , *out ;
    char line[size] ;
    if ((in = fopen("input", "r")) = = NULL)
```

```
    {
      printf ("Cannot open input file. \n") ;
      exit(1) ;
    }
  if ((out = fopen("input", "w")) = = NULL)
    {
      printf ("Cannot open output file. \n") ;
      exit(1) ;
    }
  while (fgets(line , size-1 , in) !=NULL)
    fputs(line , out) ;
  fclose(in) ; // close the input file
  fclose(out) ; // close the output file
  return 1 ;
}
```

در این برنامه ملاحظه می‌کنید که فایلها در حالت متن باز شده‌اند و از توابع fgets و fputs استفاده شده است.

آزمونهای کلی

آزمــون ۱

۱. کدام یک از تعریف متغیرهای زیر در زبان C غیرمجاز است؟

الف) int table2 ;　　ب) float table_2 ;

ج) int table2_ ;　　د) int 2table_ ;

۲. کدام یک از جمله‌های زیر غلط است؟

الف) حافظه‌ای که متغیر short int اشغال می‌کند کوچک‌تر یا مساوی حافظه‌ای است که متغیر int اشغال می‌کند.

ب) حافظه‌ای که متغیر long int اشغال می‌کند بزرگ‌تر یا مساوی حافظه‌ای است که متغیر short int اشغال می‌کند.

ج) بسته به نوع کامپایلر ممکن است مقدار حافظه‌ای که متغیر long int اشغال می‌کند کمتر از حافظه‌ای باشد که متغیر int اشغال می‌کند.

د) در متغیری از نوع unsigned int فقط اعداد صحیح مثبت را می‌توان نگهداری کرد.

۳. با اجرای دستور ; (scanf ("%3" , &a

الف) یک رقم صحیح به درونa انتقال می‌یابد.

ب) دو رقم به درون a انتقال می‌یابد.

ج) سه رقم اعشاری به درون a انتقال می‌یابد.

د) حداکثر سه رقم صحیح به درون a انتقال می‌یابد.

۴. کدام یک از جمله‌های زیر غلط است؟

الف) در زبان C برای نگهداری طول رشته از عدد صفر در انتهای آن استفاده می‌شود.

ب) برای انتقال آرایه به‌عنوان آرگومان به تابع باید از اشاره‌گر استفاده کرد.

ج) نام آرایه در واقع اشاره‌گر به اولین خانهٔ آن آرایه است.

د) از عملگر * برای به دست آوردن آدرس متغیر استفاده می‌شود.

۵. خروجی تابع زیر به ازای ورودی x = 1 چه مقداری است؟

```
int F(int x)
{
  switch (x)
    {
      case 0:
      case 2:
      case 4:
      case 6: x = x \ 2 ;
      case 1:
      case 3: x + + ;
       break ;
      default: x = x * 2 ;
    }
  return(x) ;
}
```

الف) 0

ب) 4

ج) 2

د) 1

۶. خروجی برنامهٔ زیر کدام یک از گزینه‌های زیر است؟

```
int F()
{
        static int i = 10 ;
        i = i * 2 ;
        return (i) ;
}
void main()
{
        int i = 1 , j = 2 ;
        i = F() ;
        j = F() ;
        printf ("i = %d , j = %d" , i , j) ;
}
```

الف) i = 2 و j = 20

ب) i = 1 و j = 2

ج) i = 2 و j = 2

د) i = 20 و j = 40

۷. کدام یک از جمله‌های زیر غلط است؟

الف) محدودهٔ شناسایی (Scope) متغیری که در داخل تابع تعریف می‌شود محدود به آن تابع است.

ب) اگر متغیری از نوع static در تابع داشته باشیم مقدار آن متغیر پس از اتمام تابع از بین نمی‌رود.

ج) مقداردهی اولیه به متغیر static تنها یک مرتبه انجام می‌شود.

د) در دو تابع مختلف نمی‌توان دو متغیر هم‌نام تعریف کرد.

۸. تابع زیر چه عملی انجام می‌دهد؟

```
void F(char *s)
{
        int i , j , temp ;
        for (j = 0 ; * (S+j) ! = 0 ; j + +) ;
        for (i = 0 ; i < j ; i + + , j --)
 {
        temp = * (S + i) ;
        *(S + i) = * (S + j) ;
        *(S + j) = temp ;
 }
}
```

الف) این تابع هیچ تأثیری بر رشتهٔ ورودی خود ندارد.

ب) این تابع طول رشتهٔ ورودی خود را باز می‌گرداند.

ج) عملکرد این تابع وابسته به رشتهٔ ورودی آن است و به‌طور کلی نمی‌توان گفت که چه عمل خاصی انجام می‌دهد.

د) این تابع رشتهٔ ورودی خود را معکوس می‌کند.

۹. خروجی برنامهٔ زیر کدام یک از گزینه‌های زیر است؟

```
void main()
{
  int i = 5 , j = 10 , K ;
  K = (i + +) + i*2 + (--j) ;
  printf ("%d" , K) ;
}
```

الف) ۲۴

ب) ۲۵

ج) ۲۶

د) ۲۸

۱۰. در مورد اعلان زیر در زبان C کدام یک از جمله‌های زیر صحیح است؟

```
struct student {
        char name[20] ;
        int no ;
} S , *pS ;
```

الف) این اعلان رکوردی با نام student تعریف می‌کند.

ب) متغیر S رکوردی از نوع Student و متغیر pS متغیری از نوع اشاره‌گر به student است.

ج) اگر تعریف متغیرهای S و pS را از اعلان حذف کنیم باز اعلان باقیمانده اعلان مجاز در زبان C محسوب می‌شود.

د) هر سه مورد

۱۱. تابع زیر چه عملی انجام می‌دهد؟

```
void F(char *S)
{
   int i ;
   for (i = 0 ; S[i]! = 0 ; i + +)
 if ((S[i]>'a') && (S[i]<='z'))
    S[i] = S[i] + 'A' - 'a' ;
}
```

الف) این تابع هیچ عملی انجام نمی‌دهد چون دارای خطای زمان کامپایل (ترجمه) است.

ب) این تابع حلقه‌ای بی‌نهایت دارد بنابراین برنامه هیچ گاه از آن خارج نمی‌شود.

ج) این تابع حروف کوچک رشتهٔ S را به حرف بزرگ تبدیل می‌کند.

د) این تابع حروف بزرگ رشتهٔ S را به حروف کوچک تبدیل می‌کند.

۱۲. خروجی برنامهٔ زیر کدام یک از گزینه‌های زیر است؟

```
int i = 0 ;
void F(int j)
{
        i + + ;
        j = j + 3 ;
}
void main()
{
      for (i=1 ; i<=2 ;)
      F(i) ;
      printf ("%d" , i) ;
}
```

الف) این تابع خروجی ندارد چون دارای حلقه‌ای بی‌نهایت است.

ب) 3

ج) 4

د) 2

۱۳. در مورد تابع F که به صورت زیر تعریف شده است کدام یک از گزینه‌های زیر صحیح است؟

```
void F(int *i , int j)
{
.
.
.
}
```

الف) تغییرات متغیر j در داخل F بر متغیر معادل آن در قسمت فراخوانندهٔ F تأثیر دارد.

ب) درصورتی که a و b دو متغیر از نوع int باشند می‌توان F را با دستور F(a , b) فراخوانی کرد.

ج) آرگومان i در تابع F حتماً باید نام آرایه‌ای در قسمت فراخوانندهٔ F باشد.

د) اشاره‌گر i در تابع F در واقع آدرس متغیری در قسمت فراخوانندهٔ F است که با استفاده از آن می‌توان محتوای آن متغیر را تغییر داد.

۱۴. در زبان C کدام یک از توابع زیر برای به دست آوردن طول رشته به کار می‌رود؟

الف) strcpy　　ب) strlen

ج) strcmp　　د) strcat

۱۵. قطعه برنامهٔ زیر چه عملی انجام می‌دهد؟

```
int *p , n ;
scanf ("%d" , &n) ;
p = (int *(malloc (n*sizeof(int)))) ;
```

الف) عمل خاصی انجام نمی‌دهد.

ب) دارای اشکال است چون به اشاره‌گری به غیر آدرس نمی‌توان مقداری نسبت داد.

ج) آرایه‌ای با نام p ایجاد می‌کند که نوع عناصر آن int و تعداد عناصر آن 2n است.

د) آرایه‌ای با نام p ایجاد می‌کند که نوع عناصر آن int و تعداد عناصر آن n است.

۱۶. اگر fp اشاره‌گر فایل باشد، دستور لازم برای باز کردن فایل program. pas به منظور اضافه کردن ".End" به آخر آن کدام یک از گزینه‌های زیر است؟

الف) fp = fopen ("program. pas" , "w") ;

ب) fp = fopen ("program. pas" , "wb") ;

ج) fp = fopen ("program. pas" , "a") ;

د) fp = fopen ("program. pas" , "ab") ;

۱۷. اگر fp اشاره‌گر فایل باشد، دستوری که رشتهٔ".End" را به انتهای فایل اضافه کند کدام یک از گزینه‌های زیر است؟

الف) fprintf (fp , "%s" , "End.") ;　　ب) fscanf (fp , "%s" , "End.") ;

ج) fwrite ("End. " , 4 , 4 , fp) ;　　د) هیچ کدام

۱۸. تعریف آرایهٔ مقابل را درنظر بگیرید.

```
int A[6] = {10 , 20 , 30} ;
```

مقدار اولیهٔ عنصر A[3] چقدر است؟

الف) 30 ب) 0

ج) 10 د) 20

۱۹. کدام یک از گزینه‌های زیر ثابت کاراکتری مجاز نیست؟

الف) ' a ' ب) ' \a '

ج) "/ ' د) ' "\ '

۲۰. با توجه به برنامهٔ زیر کدام گزینه صحیح است؟

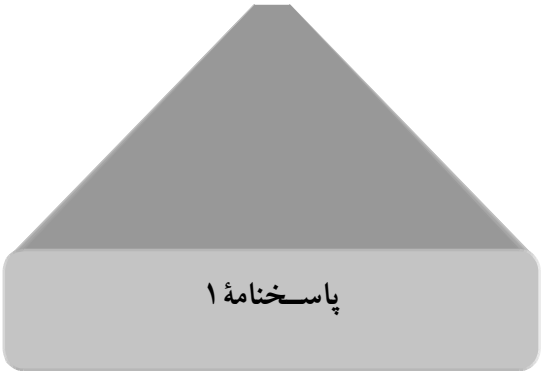
```
float a , b ;
main ()
{
        static float a ;
        void noor (void) ;
        …
}
void noor (void)
{
        static float a ;
        int b ;
        …
}
```

الف) در برنامه a و b متغیرهای خودکارند.

ب) a در main متغیر ایستا و b خودکار است.

ج) b در noor متغیر خودکار و a در noor متغیر ایستاست.

د) b در main خارجی ولی b در noor سرتاسری است.



پاســخنامهٔ ۱

	الف	ب	ج	د		الف	ب	ج	د
۱				●	۱۱				●
۲			●		۱۲	●			
۳				●	۱۳				●
۴				●	۱۴		●		
۵			●		۱۵				●
۶				●	۱۶				●
۷				●	۱۷	●			
۸				●	۱۸		●		
۹			●		۱۹			●	
۱۰			●		۲۰			●	

آزمــون ۲

۱. در دستورهای زیر مقداری که نمایش داده می‌شود چیست؟

```
char x = 256 ;
putchar (x) ;
```

الف) 256 ب) 0

ج) 255 د) 1

۲. اگر c = ´w´ و i = 7 و g = 5.5، مقدار عبارت زیر چیست؟

```
c ! = `p` || i + g < = 10
```

الف) 1 ب) 0

ج) 12.5 د) هیچ کدام

۳. در دستورالعمل زیر، دستور printf ("wait") چند بار اجرا می‌شود؟

```
for (i = 0 ; ; + + i) printf ("wait") ;
```

الف) یکبار ب) صفر بار

ج) بی‌نهایت بار د) هیچ کدام

۴. با توجه به دستور static int y [8] = {5 , 5 , 5} ; کدام گزینه صحیح است؟

الف) سه مقدار سه عنصر اول آرایه 5 و بقیهٔ عناصر آرایه صفر است.

ب) مقدار کلیهٔ عناصر آرایه صفر است.

ج) مقدار کلیهٔ عناصر آرایه 5 است.

د) سه عنصر آخر آرایه 5 و بقیهٔ عناصر 0 است.

۵. اگر a = 1 و b = 2 و c = 3، پس از اجرای دستورالعمل مقدار c چیست؟

```
c + = (a > 0 & & a < = 10) ? ++a: a / b ;
```

الف) 4

ب) 5

ج) 6

د) 3

۶. خروجی قطعه برنامهٔ زیر چیست؟

```
char x = `R` ;
switch (x)
```

```
{
   case `r`:
   case `R`: printf ("RED") ;
   case `b`:
   case `B`: printf ("BLUE") ;
}
```

الف) RED

ب) REDBLUE

ج) BLUE

د) BLUERED

۷. مقدار عبارت (10 = < (S = 5 + 4)) + 2 چیست؟

الف) 3 ب) 4

ج) 2 د) هیچ کدام

۸. اگر داشته باشیم ; char c = ´ b'، مقدار عبارت زیر چیست؟

(c > = `a' && c < = `z') ? (`A' + c - `a'): c

الف) b

ب) B

ج) ` a'

د) `A'

۹. کدام گزینه درست است؟

الف) مقادیر پیش‌فرض برای متغیرهای از نوع خارجی (external) صفر است.

ب) مقادیر پیش‌فرض برای متغیرهای از نوع static صفر است.

ج) الف و ب

د) هیچ کدام

۱۰. در مورد متغیرهای از نوع شمارشی کدام گزینه صحیح است؟

الف) می‌توان آنها را از ورودی با دستور scanf گرفت.

ب) می‌توان آنها را با دستور printf نمایش داد.

ج) الف و ب

د) هیچ کدام

۱۱. باتوجه به ; float x = 5.75 مقدار عبارت ((int)x)%2 برابر است با:

الف) 2 ج) 5

ب) 0 د) 1

۱۲. اگر داشته باشیم int x = 5، استفاده از &(x+1)

الف) صحیح است.
ب) مقدار 6 را برمی‌گرداند.
ج) غلط است.
د) اظهار نظری نمی‌توان کرد.

۱۳. قطعه برنامهٔ زیر چه کاری انجام می‌دهد؟

```
float *px , *py , t ;
t = *px ;
*px = *py ;
*py = t ;
```

الف) آدرسهای px و py را جابه‌جا می‌کند.
ب) محتوی مقادیری که با px و py اشاره می‌شود را جابه‌جا می‌کند.
ج) الف و ب
د) هیچ کدام

۱۴. دستورالعملهای زیر چه کاری انجام می‌دهد؟

```
int *x , i ;
for (i = 0 ; i < 50 ; + + i)
 *(x + i) = i ;
```

الف) مقادیر 0 تا 50 را به ترتیب در 50 عنصر اول آرایه x قرار می‌دهد.
ب) مقادیر 1 تا 50 را به ترتیب در 49 عنصر اول آرایه x قرار می‌دهد.
ج) مقادیر 0 تا 49 را در 50 عنصر اول آرایه x قرار می‌دهد.
د) هیچ کدام

۱۵. با توجه به دستورالعملهای زیر کدام گزینه صحیح است؟

```
int x = 12 ;
x = !x ;
```

الف) دستور ; x=!x غلط است.
ب) مقدار x پس از اجرای دستورالعملهای فوق 12 است.
ج) مقدار x پس از اجرای دستورالعملهای فوق 1 است.
د) مقدار x پس از اجرای دستورالعملهای فوق صفر است.

۱۶. باتوجه به دستورالعملهای زیر کدام گزینه صحیح است؟

```
char i ;
for (i = 0 ; i < 256 ; + + i)
 printf ("\n payam_noor ") ;
```

الف) دستورالعمل printf اصلاً اجرا نمی‌شود.
ب) دستورالعمل printf 256 بار اجرا می‌شود.

ج) دستورالعمل printf 255 بار اجرا می‌شود.

د) دستورالعمل printf بی‌نهایت اجرا می‌شود.

۱۷. خروجی قطعه برنامهٔ زیر چیست؟

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

الف) 7 ب) 5

ج) 4 د) 9

۱۸. این تعریف چند بایت حافظه مصرف می‌کند؟

```
union test {
   long int i ;
   char s[80] ;
   float f ;
} ;
```

الف) 88

ب) 86

ج) 80

د) حافظه‌ای مصرف نمی‌شود.

۱۹. کدام گزینه صحیح است؟

الف) gets رشته‌ای را از ورودی می‌گیرد.

ب) getchar کاراکتری را از ورودی می‌گیرد و روی صفحهٔ مانیتور نمایش می‌دهد.

ج) getch کاراکتری را از ورودی می‌گیرد و روی صفحهٔ مانیتور نمایش می‌دهد.

د) کلیهٔ موارد بالا

۲۰. پس از اجرای دستورالعملهای زیر مقادیر a , b , d چیست؟

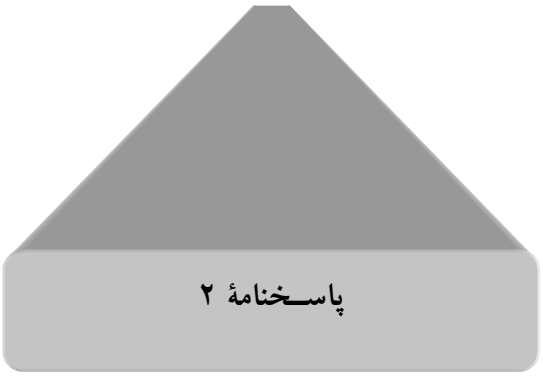
```
int a = 3 , b = 2 , d ;
d = a + + * b + + + a ;
```

الف) b = 3 و a = 4 و d = 9

ب) b = 3 و a = 5 و d = 12

ج) b = 2 و a = 5 و d = 15

د) هیچ کدام



پاسـخنامهٔ ۲

	الف	ب	ج	د		الف	ب	ج	د
۱		●			۱۱				●
۲	●				۱۲			●	
۳			●		۱۳		●		
۴	●				۱۴			●	
۵		●			۱۵				●
۶		●			۱۶				●
۷			●		۱۷		●		
۸		●			۱۸				●
۹			●		۱۹				●
۱۰				●	۲۰	●			

آزمــون ۳

۱. کدام یک از شناسه‌های زیر در زبان C مجاز نیست؟

الف) Define
ب) While
ج) If
د) int

۲. نوع داده‌های مقدماتی عبارت‌اند از:

الف) double , float , int , char
ب) float , int
ج) char , float , int
د) double , float , int

۳. کدام یک از گزینه‌های زیر درست است؟

الف) رشتهٔ n کاراکتری n+۱ عنصر آرایه نیاز خواهد داشت.

ب) رشتهٔ n کاراکتری n عنصر آرایه نیاز خواهد داشت.

ج) رشتهٔ n کاراکتری n–۱ عنصر آرایه نیاز خواهد داشت.

د) هیچ کدام

۴. کدام یک از اعلانهای زیر صحیح است؟

الف) char arr[ ] = "Tehran" ;
ب) char arr[5] = "Tehran" ;
ج) char arr[6] = "Tehran" ;
د) char arr[7] = ' Tehran ' ;

۵. فرض کنید i متغیر صحیح و f متغیر ممیز شناور باشد. در این صورت کدام یک از عبارات زیر مجاز است؟

الف) (i+f)%2
ب) ((int)(i+f))%2
ج) (int)((i+f)%2)
د) هیچ کدام

۶. در انتهای برنامه مقدار j چقدر است؟

```
# include < stdio. h >
main ()
{
  int i = 1 , j = 0 , k = 14 ;
  while (i <= k)
  {
     i *= 2 ;
     j += i ;
```

الف) 2
ب) 30
ج) 13
د) 60

```
  }
}
```

۷. اگر داشته باشیم ; int a , b [10]، کدام یک از دستورهای زیر درست است؟

الف) scanf ("%d %d" , *a , b)　　ب) scanf ("%d %d" , &a , b)

ج) scanf ("%d %d" , a , b)　　د) scanf ("%d %d" , a , &b)

۸. دستور break در میان کدام یک از دستورهای زیر استفاده می‌شود؟

الف) while　　ب) switch

ج) do-while　　د) همهٔ موارد فوق

۹. متغیرهای A[4][8] چندخانه از حافظه را اشغال می‌کنند؟

الف) ۳۲ خانه　　ب) ۱۲ خانه

ج) ۴ خانه　　د) ۸ خانه

۱۰. در برنامهٔ زیر مقدار x درون main پس از آنکه کنترل از func به main انتقال یافت چقدر است؟

```
# include < stdio. h>
void func (int x) ;
main ()
func(x) ;
  { int x = 4 ;
    printf ("\n x=%d" , x) ;
     printf ("\n x = %d" , x) ;
  }
void func (int x)
  { x * = 4 ;
    printf ("\n %d" , x) ;
    return ;
  }
```

الف) x = 4

ب) x = 16

ج) x = 8

د) هیچ کدام

۱۱. خروجی برنامهٔ زیر چیست؟

```
# include < stdio. h >
int funct (int a)
main ()
{
   int a = 10 ;
```

الف) 10

ب) 45

```
  printf ("%d" , funct (a)) ;
}
int funct (int a)
{
   if (a>0)
     return (a+funct (a-1)) ;
 }
```

ج) 55

د) 57

۱۲. تابع مقابل چه عملی انجام می‌دهد؟

```
funct (char *s1 , char *s2)
{
  while (*s1)
     if (*s1-*s2)
        return *s1-*s2 ;
    else {s1 + + ; s2 + + ; }
  return '\0' ;
}
```

الف) رشتهٔ s1 را به آخر s2 اضافه می‌کند.

ب) رشتهٔ s2 را به آخر s1 اضافه می‌کند.

ج) اشکال کامپایلری دارد.

د) رشتهٔ s1 را با رشتهٔ s2 مقایسه می‌کند.

۱۳. کدام یک از نوعهای تعریف شدهٔ کاربر است؟

الف) struct , union , pointer

ب) typedef , pointer , define

ج) union , struct , typedef

د) define , pointer , struct

۱۴. با توجه به تعریف زیر مقدار (*X+2) چیست؟

```
static int X[8] = {10 , 20 , 30 , 40 , 50 , 60 , 70 , 80} ;
```

الف) 30

ب) 12

ج) 20

د) 40

۱۵. تعریف ساختارهای مقابل را درنظر بگیرید. جهت استفاده از فیلد day از ساختار customer چه عبارتی می‌نویسیم؟

```
struct date{
   int month ;
   int day ;
   int year ;
} ;
struct account {
   int acc-no ;
   char acct-type ;
   char name [80] ;
  float balance ;
  struct date lastpayment
  ;
} customer ;
```

الف) cusomter. day

ب) customer -> lastpayment. day

ج) customer. lastpayment. day

د) customer -> day

۱۶. با توجه به تعریف آرایهٔ زیر مقدار اولیهٔ عنصر digits[3] چقدر است؟

```
int digits [5] = {2 , 3 , 3} ;
```

الف) 0 ب) 3

ج) 2 د) 6

۱۷. کدام یک از تعاریف زیر صحیح است؟

الف) char C[3] = "CAT" ; ب) char C[4] = "CAT" ;

ج) char C[ ] = "CAT" ; د) موارد ب و ج

۱۸. کدام یک از گزاره‌های زیر درست‌تر است؟

الف) برچسب default در انتهای دستور switch ظاهر می‌گردد.

ب) برچسب default نمی‌تواند در دستور switch ظاهر گردد.

ج) برچسب default می‌تواند در هر نقطه از دستور switch ظاهر گردد.

د) برچسب default حتماً باید در دستور switch ظاهر گردد.

۱۹. با توجه به دو گروه اعلانهای زیر، کدام یک از گزاره‌های زیر صحیح است؟

```
1) typedef int age ;
   age male , female ;
2) int male , femal ;
```

الف) اعلان گروه ۱ صحیح نیست. ب) دو گروه از اعلانها معادل‌اند.

ج) دو گروه از اعلانها معادل نیستند. د) هیچ کدام

۲۰. با توجه به قطعه برنامهٔ زیر کدام گزینه صحیح است؟

```
int I = 8 , J = 5 ;
I + = (J-2) ;
```

الف) I = 11 ب) I = 3

ج) I = 5 د) هیچ کدام

پاســخنامهٔ ۳

	الف	ب	ج	د		الف	ب	ج	د
۱				●	۱۱			●	
۲	●				۱۲				●
۳	●				۱۳			●	
۴	●				۱۴		●		
۵		●			۱۵			●	
۶		●			۱۶	●			
۷		●			۱۷				●
۸				●	۱۸			●	
۹	●				۱۹		●		
۱۰	●				۲۰	●			

آزمــون ۴

۱. این دستور scanf درست است یا خیر؟

```
int ch[20] ;
scanf ("%d" , ch) ;
```

الف) درست است.
ب) درست نیست.
ج) در بعضی از گونه‌ها درست است.
د) به سیستم عامل بستگی دارد.

۲. دقت ارقام داده‌های نوع float کدام گزینه است؟

الف) ۵ رقم
ب) ۱۰ رقم
ج) ۶ رقم
د) ۱۵ رقم

۳. مقدار sum چقدر است؟

```
# include < iostream. h >
# include < stdio. h >
main ()
{
  int count = 1 , sum = 0 , n = 10 ;
  while (count < = n)
    {
      count * = 2 ;
      sum + = count ;
    }
}
```

الف) 2
ب) 14
ج) 30
د) 60

۴. خروجی این دستور چیست؟

```
printf ("\n%x" , 2097151) ;
```

الف) 2097151
ب) 1fffffff
ج) 2fffff
د) 1fffff

۵. در برنامهٔ زیرمقدار x , u چقدر است؟

```
# include < stido. h >
float fun (float x) ;
main ()
{
```

الف) x = 27 ،u = 27

```
    float x = 3 , u ;
    u = fun (x) ;
    printf ("%d %d , x , u) ;
    return 0 ;
 }
float fun(float x)
 {
    float y = 0 ;
    y = x*x*x ;
    x = x*x*x ;
    return(y) ;
}
```

ب) x = 3، u = 0

ج) x = 27، u = 0

د) x = 3، u = 27

۶. مقدار n چقدر است؟

```
# include < stdio. h >
main ()
{
    int n = 0 ;
    for (; ;)
       {
         n += 3 ;
         if (n < = 10) continue ;
            n + = 4 ;
         if (n > = 25) break ;
       }
    printf ("%d" , n) ;
    return0 ;
}
```

الف) n = 11

ب) n = 26

ج) n = 29

د) n = 30

۷. در برنامهٔ زیر مقدار a را حساب کنید:

```
# include < stdio. h >
main ()
{
 int a = 13 , b = 3 ;
 a = ((a/b) *b) + (a%b) ;
 printf ("%d" , a) ;
 return 0 ; }
```

الف) a = 14

ب) a = 13

ج) a = 12

د) a = 11

۸. longfloat چند بیتی است؟

الف) ۳۲ بیت

ب) ۸ بیت

ج) ۶۴ بیت

د) ۱۶ بیت

۹. معادل عبارت ریاضی 5 > a > 3 در زبان C چیست؟

الف) (a>3) && (a<5)　　ب) (a<5) & (a>3)

ج) (a>3) and (a<5)　　د) a>3 and a<5

۱۰. اگر متغیر i منفی باشد مقدار flag چقدر است؟

```
flag = (-x<0) ? 0: 100 ;
```

الف) صفر　　ب) 100

ج) 1-　　د) 1

۱۱. این تابع چه عملی انجام می‌دهد؟

```
int even (int n)
{
   if (n%2 = = 0)
   return (1) ;
   else
   return (0) ;
}
```

الف) عدد صفر را برمی‌گرداند

ب) اشکال کامپایلری دارد.

ج) عدد صحیح را دریافت می‌کند. اگر زوج بود عدد ۱ وگرنه عدد صفر را برمی‌گرداند.

د) عدد صحیح را دریافت می‌کند و آن را به زبان اصلی برمی‌گرداند.

۱۲. کدام یک از علامتهای زیر برای دستورهای تفسیری (comment) به کار برده می‌شود؟

الف) { }　　ب) (* *)

ج) [ ]　　د) /* */

۱۳. در اعلان زیر متغیر shirt از چه نوعی است؟

```
union id{
 char color [12] ;
 int SIZE ;
 }shirt ;
```

الف) int

ب) id

ج) char

د) int , char

۱۴. اگر p اشاره گر باشد، کدام گزینه هم ارز دستور زیر است؟

```
p = &str[0] ;
```

الف) p = str
ب) p = *str
ج) p = &str
د) p = str[ ]

۱۵. برای استفاده از فیلد year از ساختار customer چه عبارتی می‌نویسیم.

```
struct date
{
   int month ;
   int day ;
   int year ;
} ;
struct account
 {
char family [20] ;
int accno ;
float payment ;
struct date lastpay ;
}customer , customer2 ;
```

الف) customer. lastpay. year
ب) customer. year
ج) customer (year)
د) customer→lastpay→year

۱۶. کدام گزینه صحیح است؟

الف) #define name text ;
ب) #define PI = 3.1415 ;
ج) #define TRUE 1
د) #define name ´ALI´

۱۷. کدام گزینه توابع ورودی را نشان می دهد؟

الف) cin , cout , getch , putch
ب) getchar , scanf , cin
ج) getchar , putchar , gets
د) gets , puts , scanf

۱۸. خروجی این برنامه چیست؟

```
#define ABS(a) (a)<0 ? -(a): (a)
main ()
{
   printf ("%d" , ABS (10 - 20)) ;
}
```

الف) -10
ب) +10
ج) 30
د) -30

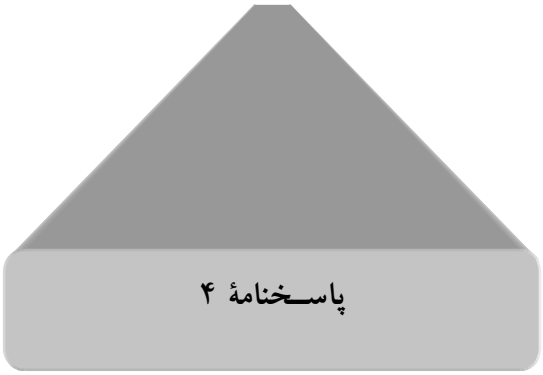
۱۹. تابع fseek این امکان را می‌دهد که

الف) به فایل به‌صورت ترتیبی دسترسی پیدا کنیم.
ب) فایل را در مد سیستم باز کنیم.
ج) به فایل به‌صورت تصادفی دسترسی پیدا کنیم.
د) فایل را در مد استاندارد باز کنیم.

۲۰. کدام تابع برای الحاق دو رشته استفاده می‌شود؟

الف) strcat　　ب) strcmp

ج) strcpy　　د) strset



	الف	ب	ج	د		الف	ب	ج	د
۱	●				۱۱			●	
۲			●		۱۲				●
۳			●		۱۳		●		
۴				●	۱۴	●			
۵				●	۱۵	●			
۶				●	۱۶			●	
۷		●			۱۷		●		
۸			●		۱۸		●		
۹	●				۱۹			●	
۱۰	●				۲۰	●			

آزمــون ۵

۱. کدام گزینه مربوط به انواع داده‌ها در زبان C نیست؟

الف) float　　ب) real

ج) integer　　د) char

۲. کدام گزینه مربوط به انواع ثابتها (Constants) در زبان C نیست؟

الف) integer　　ب) string

ج) char　　د) pointers

۳. کدام گزینه مربوط به عملگرهای (operators) مجاز در زبان C نیست؟

الف) جمع　　ب) ضرب

ج) تقسیم　　د) توان

۴. کدام عملگر یا علامت تقدم بیشتری دارد؟

الف) پرانتز　　ب) عملگرهای منطقی

ج) عملگرهای ریاضی　　د) =

۵. اگر P آدرس متغیری از نوع اعشاری را در خود داشته باشد، حاصل عبارت زیر چیست؟

&p[3] - &p[0]

الف) 3　　ب) 6

ج) 12　　د) هیچ کدام

۶. مقدار (&A) * برابر است با

الف) A　　ب) آدرس A

ج) نوع A　　د) (A+1)

۷. کدام یک از توابع زیر در برنامه C لازم است؟

الف) printf　　ب) scanf

ج) getchar　　د) main

۸. کدام گزینه مربوط به دستورهای حلقه نیست؟

الف) while　　ب) do while

ج) switch　　د) for

۹. در کدام جمله، داخل پرانتز سه عبارت (جمله) جداگانه دارد؟

الف) while ب) do while

ج) switch د) for

۱۰. در کدام تابع از عملگر & استفاده می‌شود؟

الف) printf ب) scanf

ج) putchar د) getchar

۱۱. حداقل چند مرتبه جمله‌های do while محاسبه خواهد شد؟

الف) صفر ب) یک

ج) دو د) سه

۱۲. جملهٔ if – else با کدام یک از جمله‌های زیر رابطهٔ نزدیکی دارد؟

الف) for ب) while

ج) switch د) break

۱۳. کدام یک از جمله‌های زیر صحیح است؟

الف) جملهٔ continue باعث خروج از حلقه می‌شود.

ب) جملهٔ break باعث ادامهٔ حلقه می‌شود.

ج) جملهٔ break باعث خروج از حلقه می‌شود.

د) جملهٔ continue باعث ادامهٔ محاسبات در حلقه می‌شود.

۱۴. متغیرهای نوع automatic در کدام تابع یا توابع شناخته می‌شوند.

الف) فقط در تابعی که معرفی می‌شوند.

ب) در خارج از تابعی که معرفی می‌شوند.

ج) در تمام توابع شناخته می‌شوند.

د) در تابع main

۱۵. متغیر نوع عمومی یا global در کدام توابع شناخته می‌شود؟

الف) در تابع main ب) در تابعی که معرفی می‌شود.

ج) در تمام توابع د) در هیچ تابعی شناخته نمی‌شود.

۱۶. آرایه با رشته چه فرقی دارد؟

الف) رشته همان آرایه است و فرقی باهم ندارند.

ب) در آخرین مکان یا خانهٔ آرایه اگر علامت 0\ بگذاریم تبدیل به رشته می‌شود.

ج) رشته متغیر ساده است.

د) آرایه‌های کاراکتری را رشته می‌نامند.

۱۷. اگر قبل از تعریف تابع نوع آن مشخص نباشد، چه نوع مقداری را برمی‌گرداند؟

الف) هیچ مقداری　　ب) integer

ج) float　　د) char

۱۸. اشاره‌گر کدام یک از گزینه‌های زیر است؟

الف) ثابت　　ب) تابع

ج) آرایه　　د) متغیر

۱۹. علامت * در اشاره‌گرها به چه معنایی است؟

الف) برای آدرس اشاره‌گر

ب) برای مقداری که در اشاره‌گر موجود است.

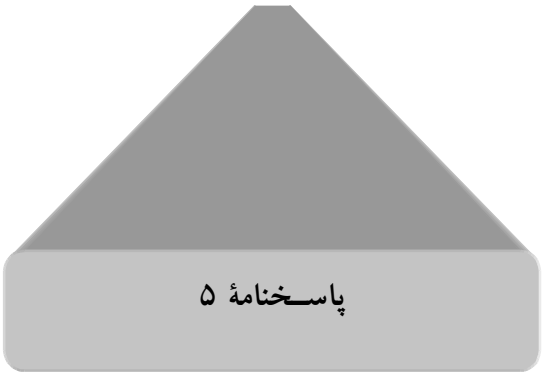
ج) برای مقدار متغیری که اشاره‌گر به آن متغیر اشاره می‌کند.

د) برای مقدار خانه یا مکان حافظهٔ مجاور به شناور

۲۰. برای نوشتن اعداد در فایل معمولاً با کدام مد فایل را باز می‌کنیم؟

الف) a + t　　ب) wt

ج) a + b　　د) wb



پاســخنامهٔ ۵

	الف	ب	ج	د		الف	ب	ج	د
۱		●			۱۱		●		
۲		●			۱۲			●	
۳				●	۱۳			●	
۴	●				۱۴	●			
۵	●				۱۵			●	
۶	●				۱۶		●		
۷				●	۱۷		●		
۸			●		۱۸	●			
۹				●	۱۹			●	
۱۰		●			۲۰				●

آزمــون ۶

۱. جواب اجرای این قسمت از برنامه چیست؟

```
int i , fact = 1 ;
scanf ( " %d " , &n) ;
for(i = 0 ; i <= n , i + +)
 fact = fact * i ;
```

الف) صفر

ب) n!

ج) ۱!

د) هیچ کدام

۲. این قطعه از برنامه چند دفعه اجرا می‌شود؟

```
i = 0 ;
do
{
 printf ( " computer " ) ;
 printf ( "\n " ) ;
{
while (i>0) ;
```

الف) یک دفعه

ب) هیچ دفعه

ج) بی‌نهایت

د) نامشخص

۳. کدام گزینه اشتباه است؟

الف) unsigned short int a ;

ب) signed char b ;

ج) long float ;

د) signed long double ;

۴. کدام یک از گزینه‌های زیر جزء شناسه‌های زبان C محسوب نمی‌شود؟

الف) four

ب) sum

ج) auto

د) str

۵. خروجی برنامهٔ زیر چیست؟

```
# include<stdio. h>
void recurse(int i) ;
main ()
{
   recurse (0) ;
   return 0 ;
}
void recurse(int i)
{
   if (i<10)
      {
        recurse(i +1) ;
```

الف) 0 1 2 3 4 5 6 7 8 9

ب) 2 3 4 5 6 7 8 9 0 1

ج) 9 8 7 6 5 4 3 2 1

د) 1 2 3 7 8 9

```
        printf ("%d" , i) ;
      }
  return ;
}
```

۶. خروجی برنامهٔ زیر چیست؟

```
# include<stdio. h>
main ()
{
   int a = 13 , b = 3 ;
   a = ((a/b) *b) + (a%b) ;
   printf ("a=%d" , a) ;
   return 0 ;
 }
```

الف) a = 14

ب) a = 13

ج) a = 12

د) a = 11

۷. جواب برنامهٔ زیر چیست؟

```
# include< iostream. h>
# include< stdio. h>
main ()
{
   int a = 1 , b = 2 , c = 3 ;
   c + = (a>0 && a< = 10) ? + + a: a/b ;
   cout <<"c = "<< c ;
   return 0 ;
}
```

الف) c = 3

ب) c = 4

ج) c = 5

د) c = 6

۸. خروجی برنامهٔ زیر چیست؟

```
# include < iostream. h>
# include < stdio. h>
# include < math. h>
main ()
 {
   double a = 4 , b = 2.0 , c ;
   int d ;
   c = pow (a , b) + sqrt (a) ;
   d = ((int)c) % 5 ;
   cout <<"d="<<d ;
   return 0 ;
}
```

الف) d = 2

ب) d = 3

ج) d = 4

د) هیچ کدام

۹. این دستور چه مقادیری را چاپ می‌کند؟

```
printf ("\n %d %x %o" , 19 , 19 , 19) ;
```

الف) 23 و 13 و 19 ب) 19 و 19 و 19

ج) 13 و 13 و 13 د) 25 و 19 و 19

۱۰. تابع فرعی زیر چه عملی را انجام می‌دهد؟

```
Str (s , t)
char s[ ] , t[ ] ;
{
 int i , j ;
 i = j = 0 ;
 while (s[i] = '\0')
 i + + ;
while ((s[i + +] = t[j + +1] = '\0');
}
```

الف)آرایهٔ کاراکتری t را به آخر آرایهٔ کاراکتری s متصل می‌کند.

ب) آرایهٔ کاراکتری S را به آخر آرایهٔ کاراکتری t متصل می‌کند.

ج) آرایهٔ کاراکتری S را روی آرایهٔ کاراکتری t کپی می‌کند.

د) آرایهٔ کاراکتری t را روی آرایهٔ کاراکتری S کپی می‌کند.

۱۱. نتیجهٔ این دستور چیست؟

```
putchar (tolower ('M')) ;
```

الف) صفر ب) m

ج) اسکی‌کد m د) هیچ کدام

۱۲. با توجه به برنامهٔ زیر مقدار PI را حساب کنید.

```
# include <stdio. h>
# define PI 3.1415
main ()
{
   float x = 7. 0 ;
   PI + = x ;
   return (0) ;
}
```

الف) PI = 10.1415

ب) PI = 10

ج) PI تغییر نمی‌کند و برنامه صحیح است.

د) PI تغییر نمی‌کند و برنامه غلط است.

۱۳. مقدار Sqr (t) , t چقدر است؟

```
main ()
{
   int t = 10;
```

```
  printf ("%d" , Sqr (t)) ;
}
Sqr (int x)
{
  int y ;
  y = x * x ;
  return (y) ;
}
```

الف) sqr(t) = 6 , t = 5

ب) sqr(t) = 100 , t = 10

ج) sqr(t) = 225 , t = 15

د) sqr(t) = 49 , t = 7

۱۴. تابع زیر تعداد فضای خالی را در رشتهٔ ورودی می‌شمارد. کدام گزینه صحیح است؟

```
CountSpace(char s[ ])
{
  int i , count = 1 ;
  for (i = 0 ; i <strlen (s) ; + + i)
  if (s[i] = = ′ ′)
  count + + ;
  ……
}
```

الف) return(i) ;

ب) return (i +1) ;

ج) return (count -1) ;

د) return(count +1) ;

۱۵. مقدار W را حساب کنید.

```
# include < iostream. h>
# include < stdio. h>
main ()
{
  int i ;
  float x = 10 , y = 20 , u = 5 , w = i ;
  for (i = 1 ; i < = 10 ; + + i)
  switch(i)
  {
    case 1:
    case 9:
    x + = 10 ;
    break ;
    case 2 ;
    case 10 ;
    y + = 20 ;
    break ;
    default:
    u + = 5 ;
  }
  w = x + y + u ;
}
```

الف) 35

ب) 65

ج) 95

د) 125

۱۶. مقدار n را حساب کنید.

```
# include<iostream. h>
# include<stdio. h>
main ()
 {
   int n = 0 ;
   for (; ;)
   {
     n + = 3 ;
     if (n< = 10) continue ;
        n + = 4 ;
     if (n> = 25) break ;
   }
 }
```

الف) 11
ب) 26
ج) 29
د) 30

۱۷. در بانک اطلاعاتی کدام توابع استفادهٔ بیشتری دارند؟

الف) fread , fwrite
ب) getc , putc
ج) fscant , fprintf
د) fgets , fputs

۱۸. مقدار u و x کدام گزینه است؟

```
# include< stdio. h>
float f (float *x) ;
main ()
 {
   float x = 5 , u ;
   u = f (&x) ;
 }
float f (float *x)
 {
   float y = 0 ;
   y = (*x) *(*x) *(*x) ;
   *x = (*x) *(*x) *(*x) ;
   return (y) ;
 }
```

الف) u = 125 , x = 125
ب) u = 125 , x = 5
ج) u = 0 , x = 125
د) u = 5 , x = 125

۱۹. با توجه به تعریف ; int *p[10] کدام گزینه درست است؟

الف) p اشاره‌گر به ۱۰ مقدار صحیح است.

ب) p اشاره‌گر به عنصر دهم آرایه‌ای از مقادیر صحیح است.

ج) p اشاره‌گر به آرایهٔ ۱۰ عنصری از مقادیر صحیح است.

د) P آرایه‌ای ۱۰ عنصری از اشاره‌گرها به مقادیر صحیح است.

۲۰. مقدار x پس از اجرای این برنامه چیست؟

```
main ()
{
  int x = 1 ;
 loop: x + + ;
  if (x<100)
  goto loop ;
  printf ("%d" , x) ;
}
```

الف) صفر

ب) 1

ج) 100

د) 99

پاسـخنامهٔ ۶

	الف	ب	ج	د		الف	ب	ج	د
۱	●				۱۱				●
۲	●				۱۲		●		
۳				●	۱۳		●		
۴			●		۱۴			●	
۵	●				۱۵			●	
۶		●			۱۶			●	
۷			●		۱۷	●			
۸		●			۱۸	●			
۹	●				۱۹				●
۱۰	●				۲۰			●	

آزمــون ۷

۱. مهم‌ترین خصوصیت زبان C کدام است؟

الف) قابل حمل است.
ب) همه منظوره است.
ج) خلاصه و کامل است.
د) کد اجرایی سریع دارد.

۲. متغیر در زبان C

الف) باید با یکی از حروف الفبا شروع شود.
ب) ممکن است با رقم شروع شود.
ج) ممکن است با ٪ شروع شود.
د) محدودیتی ندارد.

۳. تعداد حروف متغیر در زبان C

الف) باید حداکثر ۸ کاراکتر باشد.
ب) باید حداکثر ۶ کاراکتر باشد.
ج) باید حداکثر ۱۲ باشد.
د) محدودیتی ندارد.

۴. جمله‌های مقابل قسمتی از برنامه‌اند. پس از اجرا

```
m = 14 ;
n = + + m ;
```

الف) m =14 و n =15
ب) m =15 و n =15
ج) m =14 و n =14
د) m =15 و n =14

۵. جمله‌های زیر قسمتی از برنامه‌اند. پس از اجرا

```
m = 14 ;
n = m - - ;
```

الف) m =13 و n =14
ب) m =13 و n =13
ج) m =14 و n =14
د) m =14 و n =13

۶. روش غیرمستقیم دسترسی به داده‌ها با کدام گزینه امکان‌پذیر است؟

الف) آرایه
ب) اشاره‌گر
ج) ساختار
د) یونیون

۷. اگر داشته باشیم unsigned u ; short s ، نوع مقدار نهایی عبارت 3*u + s کدام است؟

الف) int
ب) unsigned
ج) short
د) long

۸. اگر داشته باشیم ; int i ; float f ، نوع مقدار نهایی عبارت 3.0*f + i کدام است؟

الف) int
ب) float
ج) double
د) unsigned

۹. اگر داشته باشیم ; int i ; float f ، پس از اجرای دستور ; f = i = 12.34 کدام گزینه صحیح است؟

الف) f =12.34
ب) f =12.3
ج) f =12
د) f =12.0

۱۰. اگر ; sum = 105. 0 ، مقدار sum پس از اجرای ; sum / = 3 + 7 کدام است؟

الف) sum =10.5
ب) sum = 42.0
ج) sum = 18.0
د) sum = 60.0

۱۱. پس از اجرای ; s = (t = 2 , t + 3) مقدار s کدام است؟

الف) s =2
ب) s = 3
ج) s = 4
د) s =5

۱۲. پس از اجرای برنامهٔ زیر مقدار متغیر two چیست؟

```
void main ()
 {
   int one ;
   void func () ;
   one = 1 ;
   func () ;
 }
 void func ()
 {
   int two ;
   two = one + 1;
 }
```

الف) 1
ب) 2
ج) نامشخص است
د) 5

۱۳. پس از اجرای برنامهٔ زیر خروجی چیست؟

```
void main ()
 {
```

الف) 1

```
   int one = 1 ;
   float one = 24.6 ;
   printf ("%d\n" , one) ;
}
```

ب) 24

ج) 24.0

د) 24.6

۱۴. در برنامهٔ زیر متغیرهای a , b

```
# include<std io. h>
int a , b ;
void main ()
{
   a = 14 ;
   b = 10 ;
   printf ("%d\n" , a + b) ;
}
```

الف) local (محلی) اند.

ب) global (عمومی) اند.

ج) automatic اند.

د) static اند.

۱۵. در تابع زیر برای خواندن متغیری که آرگومان قرار گرفته است در جملهٔ scanf به جای خطچین باید نوشت

```
void func (int *arg)
{
   scanf ("%d" , …) ;
   ……..
}
```

الف) * arg

ب) &arg

ج) arg

د) ** arg

۱۶. خروجی برنامهٔ زیر چیست؟

```
void main ()
{
   int x ;
   void change () ;
   x = 10 ;
   change (&x) ;
   printf ("%d\n" , x) ;
}
void change (int *y)
{
   *y = 1 ;
}
```

الف) صفر

ب) 1

ج) معلوم نیست.

د) 10

۱۷. خروجی برنامهٔ زیر چیست؟

```
void main ()
{
   int x ;
```

```
   void change () ;
   x = 10 ;
   change (x) ;
   printf ("%d\n" , x) ;
}
void change (int y)
{
   y = 1 ;
}
```

الف) صفر

ب) I

ج) 10

د) معلوم نیست

۱۸. کدام گزینه اشتباه است؟

الف) int color [3] = "RED" ;

ب) int digits [ ] = {1 , 2 , 3} ;

ج) static float x[6] = {0.2 , 0 , 1.5} ;

د) char color[3] = {'R' , 'E' , 'E'} ;

۱۹. در برنامهٔ زیر به جای نقطه‌چین استفاده از کدام گزینه الزامی است؟

```
.....
main ()
{
  double x = 10.0 , y = 0.0 ;
  do
   {
     printf ("%f" , pow (x , y)) ;
     y + + ;
  } while (y<11) ;
}
```

الف) #include <conio. h>

ب) #include <math. h>

ج) #include <iostream. h>

د) #include <stdlib. h>

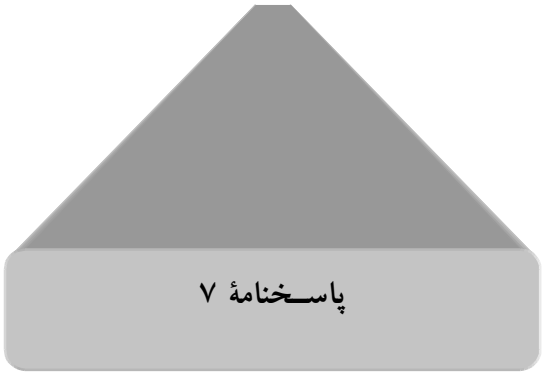
۲۰. اگر داشته باشیم int a = 1 , b = 3 ; ، حاصل عبارت a > 2 && (! b) برابر است با

الف) صفر

ب) 1

ج) -1

د) معلوم نیست.



پاســخنامهٔ ۷

	الف	ب	ج	د		الف	ب	ج	د
۱	●				۱۱	●			
۲	●				۱۲			●	
۳				●	۱۳		●		
۴		●			۱۴		●		
۵	●				۱۵			●	
۶		●			۱۶		●		
۷		●			۱۷			●	
۸		●			۱۸	●			
۹				●	۱۹		●		
۱۰	●				۲۰	●			

آزمون ۸

۱. کدام یک از مجموعه شناسه‌های زیردر زبان C معتبر است؟

الف) double , define , Union , Main
ب) for , while , -if , +main
ج) begin , break , else , int
د) Printf , Scanf ,Real , Char

۲. با توجه به تکه برنامهٔ زیر کدام گزینه صحیح است؟

```
int a , x = 2 , y = 7;
a = y %3 ;
p = (x>y || x = = y/3) ? (a= = 1) + 2: (X > y) + 5
```

الف) P = 3
ب) P = 2
ج) P = 5
د) P = 6

۳. خروجی دستور زیر چیست؟

```
printf ("%d" , (a>b) ? ((b>c)? c: b): ((a>c) ? c: a)) ;
```

الف) ماکزیمم مقدار بین a , b , c
ب) مینیمم مقدار بین a , b , c
ج) میانگین سه مقدار a , b , c
د) در زمان کامپایل error می‌دهد

۴. خروجی این برنامه چیست؟

```
main ()
{ int i = 1 ;
  switch (i)
  {
    case 1: printf ("one") ;
    case 2: printf ("two") ;
    default: printf ("else") ;
  }
}
```

الف) one
ب) else
ج) onetwo
د) onetwoelse

۵. با توجه به دستورهای زیر مقدار ذخیره شده در متغیرهای b و a کدام‌اند؟

```
int a , b ;
a = -17/3 ;
b = -15/8 ;
```

الف) a = -5 و b = -1
ب) a = -5 و b = 1
ج) a = 5 و b = -1
د) a = 1 و b = 5

۶. با اعلان متغیرهای int a و float b کدام یک از گزینه‌ها صحیح است؟

الف) a = 13/3 ب) b = 13%5

ج) a = 7.5 × 6 د) همهٔ موارد

۷. باتوجه به تکه برنامهٔ زیر مقدار چاپ شده کدام است؟

```
# define P(a) (a) ? 20: 30
int x = 10 , y = -5 ;
printf ("%d" , P (x%3-y) -5) ;
```

الف) 30 ب) 25

ج) 20 د) 15

۸. در برنامهٔ زیر اگر متغیرهای A = 10 ، B = 15، و C = 4 باشند، چه مقداری چاپ می‌شود؟

```
if (!(A>B) ‖ (C>A) && (A<B))
   if (!(B<A) && (B<C))
      printf("1") ;
   else
      printf ("2") ;
else if (B>C && C>A) ‖!(A<B)
   printf ("3") ;
else
   printf ("4") ;
```

الف) 1

ب) 2

ج) 3

د) 4

۹. اگر P، آدرس متغیری از نوع اعشاری را در خود داشته باشد، عبارت P + 5 به

الف) ۲۰ بایت بعد از P اشاره می‌کند. ب) ۱۰ بایت بعد از P اشاره می‌کند.

ج) ۵ بایت بعد از P اشاره می‌کند. د) ۱ بایت بعد از P اشاره می‌کند.

۱۰. خروجی قطعه برنامهٔ زیر چیست؟

```
int x , *p , **q ;
x = 10 ;
p = &x ;
q = &p ;
printf ("%d" , **q) ;
```

الف) مقدار متغیر x

ب) آدرس متغیر x

ج) 1

د) 0

۱۱. خروجی این برنامه چیست؟

الف) 5

```
main ()
{
  struct easy {
```

```
  int num ;
  char ch ;
  } e = {5 , 'A'} ;
printf ("%03d" , easy. num) ;
}
```

ب) 035

ج) 005

د) هیچ کدام

۱۲. کدام گزینه صحیح است؟

الف) آرایه‌ای از اشاره‌گرها، نوعی از اشاره‌گر به اشاره‌گر است.

ب) اشاره‌گرها را می‌توان با یکدیگر مقایسه کرد.

ج) اشاره‌گرها فراخوانی با آدرس را در مورد توابع امکان‌پذیر می‌سازند.

د) هر سه گزینه

۱۳. بازگرداندن بیش از یک مقدار، از یک تابع، با کدام روش میسر می‌شود؟

الف) فراخوانی با مقدار ب) فراخوانی با آدرس

ج) هر دو د) هیچ کدام

۱۴. اگر تابع P در دستور printf ("%d" , P(10 , P(10 , 7))) ; فراخوانی شود، چه مقداری چاپ می‌شود؟

```
P (a , b)
int a , b ;
{
  return (a + b) ;
  else
   return (b - a) ;
}
```

الف) 27

ب) 0

ج) 7

د) -7

۱۵. اگر تابع f در دستور printf ("%d" , f(10)) ; فراخوانی گردد، چه مقداری چاپ می‌شود؟

```
f (int n)
{
  if (n>0) return (n+f(n-2)) ;
}
```

الف) 30

ب) 20

ج) 18

د) 2

۱۶. با درنظر گرفتن تعریف آرایهٔ x کدام گزینه صحیح است؟

```
static x = {10 , 20 , 30 , 40 , 50 , 60 , 70 , 80} ;
```

الف) x* یعنی آدرس شروع بردار x ب) (x+2*) یعنی 12 و (x+2)* یعنی 30

ج) (x+2*) یعنی 30 و (x+2)* یعنی 12 د) (x+2) یعنی 30

۱۷. با توجه به برنامهٔ اصلی و دو تابع f1 و f2 خروجی این مجموعه چه خواهد بود؟

```
int a = 100 , b = 200 ;
main ()
 {
   int i ;
   int f1(int i) ;
   for (i =1 ; i<= 5 ; + +i)
   printf ("%d" , f1(i));
 }
f1 (int x)
 {
   int c , d ;
   int f2(int c) ;
   c = f2 (x) ;
   d = (c<100) ? (a+c): b ;
   return(d) ;
}
f2 (int x)
 {
   static int p = 1 ;
   p *= x ;
   return (p) ;
 }
```

الف) 101 , 102 , 106 , 124 , 200

ب) 101 , 102 , 103 , 104 , 105

ج) 1 , 2 , 6 , 24 , 100

د) هیچ کدام

۱۸. خروجی دستورهای زیر چه خواهد بود؟

```
for (i = 0 ; ++i<= 10 ; i += 2) ;
printf ("%d%d" , i-- , ! i) ;
```

الف) 13 , 0

ب) 12 , 0

ج) 11 , 0

د) 10 , 0

۱۹. رشتهٔ "abc\ndef\nghi" به آرایهٔ چند عنصری نیاز دارد؟

الف) ۱۳ ب) ۱۴

ج) ۱۱ د) ۱۲

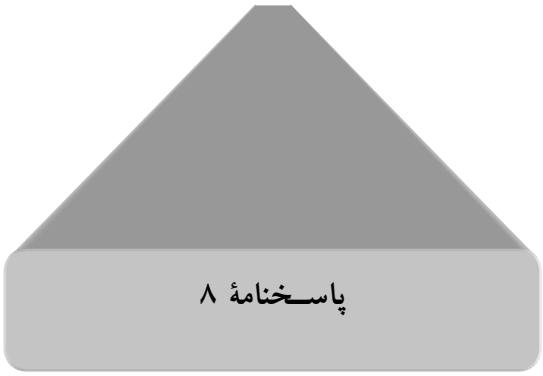
۲۰. کدام گزینه صحیح نیست؟

الف) هر رکورد مجموعه‌ای از چند فیلد است.

ب) فایل مجموعه‌ای از داده‌های مرتبط به هم است.

ج) داده‌های مربوط به هریک از اجزای فیلد رکورد نام دارد.

د) هر فایل مجموعه‌ای از چند رکورد است.



	الف	ب	ج	د		الف	ب	ج	د
۱				●	۱۱				●
۲	●				۱۲				●
۳		●			۱۳		●		
۴				●	۱۴			●	
۵	●				۱۵	●			
۶				●	۱۶		●		
۷				●	۱۷	●			
۸		●			۱۸	●			
۹	●				۱۹				●
۱۰	●				۲۰			●	

آزمــون ۹

۱. کدام یک از گزاره‌های زیر درست است؟

الف) C زبان برنامه‌نویسی ساخت‌یافته و همه‌منظوره است.

ب) برنامه‌های زبان C قابلیت انتقال بالایی دارند حتی بیشتر از سایر زبانهای سطح بالا.

ج) زبان C باتوجه به توانایی نوشتن خیلی مختصر برنامه‌های منبع معروف شده است.

د) همهٔ موارد فوق

۲. اگر int معمولی بتواند از ۳۲۷۶۸- تا ۳۲۷۶۷+ تغییر نماید، در این صورت unsigned int در کدام دامنه‌های زیر تغییر می‌کند؟

الف) صفر تا ۶۵۵۳۵ ب) صفر تا ۳۲۷۶۷

ج) صفر تا ۱۶۳۸۴ د) ۱۶۳۸۴- تا ۱۶۳۸۳

۳. کدام یک از اعلانهای زیر راه مناسب برای جایگزینی رشته در آرایهٔ کاراکتری است؟

الف) char text[ ] = "Esfahan" ب) char text[7] = "Esfahan"

ج) هر دو د) هیچ کدام

۴. نوع دستور زیر چیست؟

```
if (x>a)
   y = 5.0 ;
else
   y = 10.0 ;
```

الف) دستور مرکب

ب) دستور کنترل

ج) دستور عبارتی

د) دستور انتقال

۵. برنامه‌ای در C شامل اعلانها و جایگزینی مقادیر اولیهٔ زیر است.

```
float x = 0.005 , y = -0.01 ;
```

مقدار عبارت 2 * x + y = = 0 برابر است با

الف) 0 ب) 0.001

ج) 1 د) 0.015

۶. مقدار a چقدر است؟

```
#include<stdio. h>
main ()
```

الف) a = ′c′

```
{
  char a , c = ´c´ , d = ´d´ ;
  a = (c<d) ? c: d ;
}
```

ب) a = ´d´

ج) a = 0

د) a = 1

۷. برنامهٔ زیر را در نظر بگیرید.

```
#include<stdio. h>
main ()
{
  char line[80] ;
  ……
  scanf ("%s" , line) ;
  ……
}
```

اگر رشتهٔ Isfahan is a city از دستگاه ورودی استاندارد و در هنگام اجرای برنامه وارد گردد، کدام قسمت از رشته در آرایهٔ line جایگزین می‌گردد؟

الف) کل رشته

ب) فقط کلمهٔ Isfahan

ج) فقط حرف تکی I

د) هیچ کلمه‌ای در آرایه جایگزین نمی‌شود.

۸. با توجه به اعلان زیر کدام گزینه خروجی صحیح را چاپ می‌کند؟

```
int i , J , k ;
```

الف) printf ("%f%f%d" , sqrt(i+j) , abs (i-k) , i) ;

ب) printf ("%c%d%d" , sqrt (i+j) , abs (i-k) , i) ;

ج) printf ("%f%d%d" , sqrt (i+j) , abs (i-k) , i) ;

د) printf ("%d%d%d" , sqrt (i+j) , abs (i-k) , i) ;

۹. برنامهٔ زیر چه عملی انجام می‌دهد؟

```
# include<stdio. h>
main ()
{
   int i = 0 ;
  for (i =0 ; i < = 9 ;)
  printf ("%d" , i + +) ;
}
```

الف) اعداد صحیح ۰ تا ۹ را چاپ می‌کند.

ب) اعداد صحیح ۱ تا ۹ را چاپ می‌کند.

ج) اعداد صحیح ۰ تا ۸ را چاپ می‌کند.

د) اعداد صحیح ۱ تا ۸ را چاپ می‌کند.

۱۰. تابع f چه می‌کند؟

```
f (char *p[ ] , char *q[ ])
{
  register int i ;
   for (i = 0 ; p[i] ; + + i)
   if (!strcmp (p[i] , q))
   return i ;
   return –1 ;
}
```

الف) به دنبال کاراکتر q در رشتهٔ p می‌گردد. اگر آن را یافت موقعیت آن و در غیر این‌صورت 1- را برمی‌گرداند.

ب) در رشتهٔ q می‌گردد. اگر q در p بود موقعیت شروع آن و در غیر این‌صورت 1- را برمی‌گرداند.

ج) رشتهٔ q را در رشتهٔ p کپی می‌کند و آدرس شروع آن را برمی‌گرداند. اگر عمل کپی صورت نگرفت 1- را برمی‌گرداند.

د) رشتهٔ q را به دنبال رشتهٔ p می‌چسباند و موقعیت شروع آن را برمی‌گرداند. اگر p خالی باشد 1- را برمی‌گرداند.

۱۱. دستور continue در میان کدام یک از دستورهای زیر استفاده نمی‌شود؟

الف) while　　ب) for

ج) switch　　د) do while

۱۲. کدام گزینه درست نیست؟

الف) در زبان C هر ارزش غیرصفر به معنی true و ارزش صفر به معنی false است.

ب) در زبان C ارزشی که برای ثابت ´x´ و "x" درنظر گرفته می‌شود یکسان نیست.

ج) خطای منطقی از کامپایل و اجرا شدن برنامه جلوگیری می‌کند.

د) تقسیم بر صفر خطای کمپایل محسوب نمی‌شود.

۱۳. اگر متغیر x مثبت باشد، مقدار a چقدر است؟

```
a = (x < = 0) ? 5: 200
```

الف) نامفهوم　　ب) 200

ج) 5　　د) 0

۱۴. خروجی حاصل از برنامهٔ مقابل چقدر است؟

```
# include < stdio. h>
# define Rows 3
# define columns 3
int x[Rows][columns] = {11 , 12 , 13 , 14 , 15 , 16 , 17 , 18 ,
                                                    19} ;
main ()
{
   int a , b , c = 99 ;
  for (a = 0 ; a < Rows ; + + a)
  for (b = 0 ; b < columns ; + + b)
  if (x[a][b]< c) c = x[a][b] ;
   printf ("%d" , c) ;
}
```

الف) 11

ب) 19

ج) 99

د) 0

۱۵. کدام دسته از عملگرهای زیر جزء عملگرهای زبان C اند؟

الف) (+ , - , ^ , *)

ب) (+ , - , % , / , *)

ج) (! , = ! , = :)

د) (= = , != , * *)

۱۶. خروجی حاصل از برنامهٔ مقابل چیست؟

```
# include<stdio. h>
main ()
{
  int a , b = 0 ;
   int c[5] = {1 , 2 , 3 , 4 , 5} ;
   for (a = 0 ; a<5 ; ++a)
   b += c[a] ;
   printf ("%d" , b) ;
}
```

الف) 15

ب) 10

ج) اجرا نخواهد شد.

د) 14

۱۷. خروجی دستورهای زیر چیست؟

```
i =1
printf ("%d , %d , %d" , i , i + + , i) ;
```

الف) ۱، ۱، ۱

ب) ۱، ۲، ۱

ج) ۲، ۲، ۱

د) هیچ کدام

۱۸. کدام یک از عبارات زیر درست است؟

الف) تمام عضوهایی که یک یونیون را تشکیل می‌دهند از فضای حافظه به‌طور مشترک استفاده می‌کنند.

ب) یونیونها برای کاربردهایی مناسب‌اند که چندین عضو دارند و نیازی نباشد همزمان مقادیر در تمام عضوها جایگزین شوند.

ج) یونیون ممکن است عضوی از ساختار و ساختار عضوی از یونیون باشد.

د) همهٔ موارد فوق

۱۹. اگر تابع زیر را به‌صورت swap(5 , -5) صدا بزنیم کدام گزینه صحیح است؟

```
void swap (int A , int B)
{
  A = A + B ;
  B = A - B ;
  A = A - B ;
  printf ("%d , %d" , A , B) ;
}
```

الف) 0 , 0 ب) -5 , 5

ج) -5 , 5 د) 0 , 0/

۲۰. فرض کنید a متغیر صحیح بدون علامت با مقدار 0x6db7 باشد. عبارت زیر چه عملی انجام می‌دهد؟

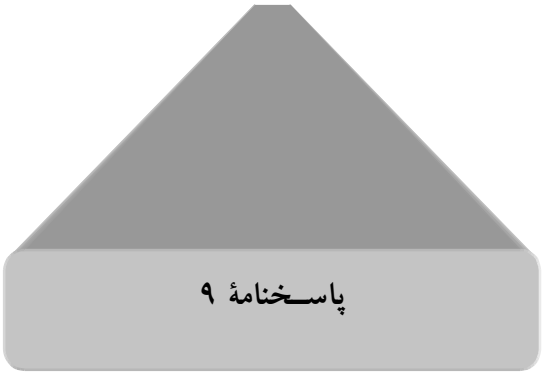
```
b = a << 6 ;
```

الف) بیت ششم از سمت چپ را به راست منتقل می‌کند.

ب) تمام بیتهای a را شش خانه به راست انتقال می‌دهد و الگوی بیتهای حاصل را در متغیر صحیح بدون علامت b قرار می‌دهد.

ج) تمام بیتهای a را شش خانه به چپ انتقال می‌دهد و الگوی بیتهای حاصل را در متغیر صحیح بدون علامت b قرار می‌دهد.

د) هیچ کدام



پاســخنامهٔ ۹

	الف	ب	ج	د		الف	ب	ج	د
۱				●	۱۱			●	
۲	●				۱۲			●	
۳	●				۱۳		●		
۴		●			۱۴	●			
۵			●		۱۵		●		
۶	●				۱۶	●			
۷		●			۱۷				●
۸			●		۱۸				●
۹	●				۱۹		●		
۱۰		●			۲۰			●	

آزمـون ۱۰

۱. "3" در زبان C معرف چیست؟

الف) عدد صحیح
ب) ثابت کراکتری
ج) ثابت رشته‌ای
د) هیچ کدام

۲. کدام یک عملگر یکانی (unary) اند؟

الف) (- , + + , sizeof , *)
ب) (+ , - , % , !)
ج) (- , + + , -- , !)
د) (< , != , &)

۳. کدام دستور رشتهٔ ; {'t' , 'e' , 's' , 't' , '\0'} = [5]char str را خالی می‌کند؟

الف) str = 0
ب) str[0] = 0
ج) str = NULL
د) str[0] = NULL

۴. در صورتی که بخواهیم نام جدیدی برای نوع موجود ایجاد کنیم از کدام گزینه باید استفاده کرد؟

الف) structure
ب) union
ج) typedef
د) enum

۵. کدام گزینه صحیح است؟

```
main ()
{
  int *p ;
  static int a[ ] = {2 , 4 , 6 , 8 , 10} ;
  ……….
  for (i = 0 ; i<5 ; ++i)
    printf ("%d" , ……) ;
}
```

الف) P = a;
* (p + i);

ب) P = &a [0];
(*p + i);

ج) *P = a;
(*p + i);

د) P = &a [0];
* (p + i);

۶. حداقل تعداد دفعاتی که حلقI while اجرا می‌شود چند بار است؟

الف) هیچ بار
ب) یکبار
ج) دوبار
د) به تعداد دفعات مورد نیاز

۷. باتوجه به دستورهای زیر مقادیر چاپ شده چقدر است؟

```
i = 5 ;
printf ("%d", i++) ;
printf ("%d", --i) ;
printf ("%d", i) ;
```

الف) 4 6 5
ب) 5 6 5
ج) 4 5 5
د) 5 5 5

۸. حلقهٔ for چه موقع مناسب‌تر از حلقهٔ while است؟

الف) وضعیت پایانی به‌طور ناگهانی روی دهد.
ب) برنامهٔ حلقه حداقل یک بار اجرا شود.
ج) برنامه حداقل یک بار اجرا شود.
د) تعداد دفعات گردش حلقه معلوم باشد.

۹. خروجی این برنامه چیست؟

```
main ()
{
  int i ;
  for(i = 0 ; + + i <5 ; i + +) ;
   printf ("%d" , i + +) ;
}
```

الف) 0 , 2 , 4 , 6 , 8
ب) 5
ج) 1 , 3 , 5 , 7 , 9
د) 6

۱۰. کدام گزینه از کلاس حافظه موجودیت متغیر محلی را سراسری می‌کند ولی وضوح را تغییر نمی‌دهد.

الف) register
ب) extern
ج) auto
د) static

۱۱. مقادیر x[2][0] و x[0][3] در تعریف زیر چند است؟

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

الف) x[0][3]=0 x[2][0]=0
ب) x[0][3]=7 x[2][0]=7

ج) x[0][3]=0 x[2][0]=7

د) x[0][3]=3 x[2][0]=7

۱۲. مقادیر a، b و c بعد از اجرای دستور زیر و ورودی 9 5678 1234 چیست؟

```
int a , b , c ;
scanf ("%3d %3d %3d" , &a , &b , &c) ;
```

الف) c = 789 , b = 456 , a =123 ب) c = 9 , b = 5678 , a =1234

ج) c = 567 , b = 4 , a =123 د) هیچ کدام

۱۳. خروجی دستورهای زیر چیست؟

```
int i = 0 ;
for (; i < = a ;)
printf (%d" , i++) ;
```

الف) اعداد صحیح 0 الی a

ب) اعداد صحیح 1 الی 10

ج) اعداد صحیح 1 الی a

د) اعداد صحیح 0 الی 10

۱۴. عملکرد تابع زیر چیست؟

```
void func(void)
{
  char c ;
  if ((c = getchar()) != EOLN (
     func() ;
  putchar(c) ;
}
```

الف) بدون خروجی

ب) خروجی رشته معادل ورودی

ج) خروجی رشته وارونهٔ ورودی

د) هیچ کدام

۱۵. عملکرد تابع زیر چیست؟

```
int a , b , c ;
return ((a<b)?((b<c)?c: b): ((a<c) ? c: a)) ;
```

الف) تابع مینیمم ب) تابع ماکزیمم

ج) تابع میانگین د) هیچ کدام

۱۶. فراخوانی با آدرس در توابع، با استفاده از کدام گزینه امکان پذیر است؟

الف) اشاره‌گر ب) ساختار

ج) یونیون د) نوع شمارشی

۱۷. تابع f چه انجام می‌دهد؟

```
f (unsigned x)
{
```

```
   int b ;
   for (b = 0 ; x ; x >> = 1)
     if (x f 1)
        b++ ;
   return (b) ;
}
```

الف) خطای منطقی دارد.

ب) تعداد صفرهای متغیر x را می‌شمارد.

ج) متغیر x را به تعداد b به راست شیفت می‌دهد و برمی‌گرداند.

د) تعداد یکهای متغیر x را می‌شمارد.

۱۸. خروجی دستور زیر کدام گزینه است؟

```
printf("%d" , toupper(tolower('H'))) ;
```

الف) h ب) H

ج) کد اسکی h د) کد اسکی H

۱۹. دستور زیر چه عملی انجام می‌دهد؟

```
putc(x , stdout) ;
```

الف) کاراکتر x در صفحه نمایش نوشته می‌شود.

ب) متغیر x در صفحه نمایش نوشته می‌شود.

ج) کاراکتر x از صفحه کلید خوانده می‌شود.

د) کاراکتری از ورودی خوانده و در متغیر x قرار داده می‌شود.

۲۰. کدام گزینه صحیح نیست؟

الف) خطای املایی (syntax error) موجب می‌شود برنامه اجرا نگردد.

ب) خطای منطقی موجب عدم کامپایل و اجرای برنامه می‌گردد.

ج) تقسیم به صفر خطای از نوع اجرایی (run time error) است.

د) پیغام warning خطا نیست.

پاســخنامهٔ ۱۰

	الف	ب	ج	د		الف	ب	ج	د
۱			●		۱۱			●	
۲			●		۱۲			●	
۳				●	۱۳	●			
۴			●		۱۴			●	
۵	●				۱۵		●		
۶	●				۱۶	●			
۷				●	۱۷				●
۸				●	۱۸				●
۹		●			۱۹		●		
۱۰				●	۲۰		●		

آزمــون ۱۱

۱. کدام گزینه درست است؟

الف) در زبان C کلمات کلیدی وجود دارد که همگی با حروف کوچک نوشته می‌شوند.

ب) تقسیم بر صفر خطای کامپایل محسوب می‌شود.

ج) خطای منطقی از کامپایل و اجرا شدن برنامه جلوگیری می‌کند.

د) موارد الف و ج

۲. کدام گزینه درست است؟

الف) در زبان C هر ارزش غیرصفر به معنی true و ارزش صفر به معنی false است.

ب) در زبان C ارزشی که برای ثابت ´x´ و "x" درنظر گرفته می‌شود یکسان است.

ج) دو تعریف ; "Alt" =[3]char t و ; "Alt" = [ ]char t در زبان C یکسان نیستند.

د) موارد الف و ج

۳. قرار گرفتن توابع در زبان C ممکن است

الف) قبل از main باشد و یکی ترجمه شوند.

ب) بعد از main باشد و یکجا ترجمه شوند.

ج) جدای از هم ترجمه و در دو فایل مجزا قرار گیرند.

د) موارد الف، ب و ج

۴. مقدار ذخیره شده در متغیر A چقدر است؟

```
A = 5 * 3 %4/2 * 3 + 5/2 ;
```

الف) 6 ب) 5

ج) 5.5 د) 8

۵. در دستور fgets (buf , 60 , fp) شرط اتمام خواندن رشته از اشاره‌گر فایل fp کدام است؟

الف) رسیدن به انتهای فایل ب) رسیدن به انتهای خط و \n

ج) خواندن شصت کاراکتر از فایل د هر سه مورد

۶. درصورتی که ; c = -15 و ; d = 6 ، خروجی دستور زیر چه خواهد بود؟

```
printf ("\n x=%d , y=%d" , c/d , c%d) ;
```

الف) x = -2 , y = -3 ب) x = -3 , y = -2

ج) x = 3 , y = -2 د) x=-3 , y = 2

۷. خروجی تکه برنامهٔ زیر چه خواهد بود؟

```
int i = 10 ;
for (; i > 0 ;)
 printf ("%d" , --i) ;
```

الف) چاپ اعداد صحیح ۹ تا صفر ب) چاپ اعداد صحیح ۱۰ تا ۱

ج) چاپ اعداد صحیح ۹ تا ۱ د) چاپ اعداد صحیح ۱۰ تا صفر

۸. با اجرای تکه برنامهٔ زیر محتوای P چه خواهد بود؟

```
P = 10 ;
n = 256 ;
a = n%10 ;
P += (a= = 6 ‖ n>=a) ? (a= =5) + 5: + + a ;
```

الف) 16

ب) 15

ج) 10

د) 17

۹. خروجی دستورهای مقابل چه خواهد بود؟

```
#define PI 3.14
printf ("PI=%f" , PI) ;
```

الف) PI = 3.14 ب) 3.14 = 3.14

ج) PI = PI د) در زمان کامپایل error می‌دهد.

۱۰. با فرض ; &y = [2]x* کدام گزینه صحیح است؟

الف) y = **x[2] ; ب) y = &x[2] ;

ج) y = x[2] ; د) y = *x[ ] ;

۱۱. کدام گزینه معادل قطعه برنامهٔ زیر است؟

```
S = 0 ;
for (i = 0 ; i < = n ; s + = + + i) ;
```

الف)

```
S = 0 ; i = 0;
while (i<=n)
s + = + + i ;
```

ب)

```
S = 0; i = n ;
while (i>=0)
 s + = -- i ;
```

ج) د) موارد الف و ج

```
S = 0 ; i = n ;
while (i>=0)
  s + = i-- ;
```

۱۲. کدام جمله در مورد متغیرهای ایستا صحیح است؟

الف) قابل دستیابی در بیرون از تابع تعریف شده

ب) حفظ مقادیر قبلی هنگام اجرای مجدد تابع

ج) مورد الف و ب

د) هیچ کدام

۱۳. خروجی قطعه برنامهٔ زیر چه خواهد بود؟

```
int i ;
char ch ;
for (i = 0 , ch = ´A´ ; i<4 ; i + + , ch + = 2 * i)
printf ("%c" , ch) ;
```

الف) ABCD ب) ACEG

ج) ACGM د) ACGMP

۱۴. با اجرای تکه برنامهٔ زیر خروجی چه خواهد بود؟

```
for (i = 0 ; i<50 ; i+ +)
printf ("%d %c" , a[i] , (i%10 = = 9) ? ´\n´: ´b´) ;
```

الف) هر ۹ عنصر بردار روی یک سطر چاپ می‌گردد.

ب) هر ۱۰ عنصر بردار روی یک سطر چاپ می‌گردد.

ج) هر ۱۱ عنصر بردار روی یک سطر چاپ می‌گردد.

د) هر عنصر بردار روی یک سطر چاپ می‌گردد.

۱۵. خروجی برنامهٔ مقابل چه خواهد بود؟

```
int i , j , k = 5 ;
for (i = 0 , j = 10 ; i<3 ; + + i , j --)
        switch (i+j)
{
  case 15: k + = i ;
  case 10: k * = --j ;
```

الف) 4

ب) 0

```
  default: k -- ;
}
printf ("%d" , k) ;
```

ج) 42

د) 6

۱۶. اگر تابع f به فرم ; printf ("%d" , f(10 , f(7/10))) احضار گردد چه چاپ می‌شود؟

```
f (a , b)
int a , b
{
  if (a>b)
     return (a+b) ;
  else
    return (b-a) ;
}
```

الف) 13

ب) 3

ج) 0

د) 27

۱۷. اگر ; a = 0xabda و بخواهیم ۵ بیت سمت چپ و ۷ بیت راست بدون تغییر و بیتهای وسط معکوس شوند کدام گزینه درست است؟

الف) a &= 0x01E0

ب) a ^ 0x0lE0

ج) a ^= 0xFlEF

د) a ^= 0x0lE0

۱۸. با درنظر گرفتن برنامهٔ زیر کدام گزینه درست است؟

```
int i = 0 ;
main ()
{
  void f1(void) ;
  f1() ;
  printf ("%d" , i) ;
}
void f1(void)
{
  if (getchar()! = ´\n´)
    {
      + + i ;
      f1() ;
    }
   return ;
}
```

الف) احضار تابع f1 به این صورت درست نیست.

ب) تابع f1 چیزی برنمی‌گرداند.

ج) تعداد کاراکتر خوانده شده از صفحه کلید چاپ می‌گردد.

د) موارد ب و ج

۱۹. با توجه به تعریف مقابل کدام گزینه صحیح است؟.

```
int P[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ,
               10} ;
int Q[3][4] = {
```

الف) Q[2][3] = 9 , P[1][2] =7

```
{1 , 2 , 3}
{4 , 5 , 6}
{7 , 8 , 9 , 10}
} ;
```

ب) Q[2][3] =10 , P[1][2] =7

ج) Q[2][3] = 0 , P[1][2] = 0

د) Q[2][3] = 9 , P[1][2] = 0

۲۰. (&A)* برابر است با

الف) A

ب) آدرس A

ج) نوع A

د) A+1

پاســخنامهٔ ۱۱

	الف	ب	ج	د		الف	ب	ج	د
۱	●				۱۱				●
۲				●	۱۲		●		
۳				●	۱۳			●	
۴		●			۱۴		●		
۵				●	۱۵				●
۶	●				۱۶	●			
۷			●		۱۷				●
۸		●			۱۸				●
۹	●				۱۹	●			
۱۰	●				۲۰	●			

آزمــون ۱۲

۱. محل قرار گرفتن توابع در زبان C کدام است؟

الف) قبل از main
ب) بعد از main
ج) هر دو
د) هیچ کدام

۲. "0x3" در زبان C معرف چیست؟

الف) عدد ثابت صحیح
ب) ثابت کاراکتری
ج) ثابت رشته‌ای
د) ثابت هگزادسیمال

۳. خروجی دستورهای زیر کدام گزینه است؟

```
char d ;
d = 'a' ;
printf ("%d" , d) ;
```

الف) a
ب) 97
ج) d
د) هیچ کدام

۴. آخرین دستور بعد از حلقهٔ زیر چه دستوری باشد تا میانگین اعداد خوانده شده چاپ شود؟

```
sum = 0 ;
j = 1 ;
for (I = 0 ; I<10 ; + + I)
{
   scanf (&a) ;
   sum + = a ;
   j + + ;
}
```

الف) printf ("\n%f" , sum / I) ;
ب) printf ("\n%f , sum / j) ;
ج) printf ("\n%f , sum / (j+1)) ;
د) printf ("\n%f" , sum / (I+1)) ;

۵. خروجی دستور زیر با مقدار اولیهٔ ۱ برای متغیر صحیح i چیست؟

```
printf ("%d , %d , %d" , i , i + + , i) ;
```

الف) ۱, ۱, ۱
ب) ۱, ۲, ۱
ج) ۲, ۲, ۱
د) هیچ کدام

۶. با اجرای تکه برنامهٔ زیر خروجی چه خواهد بود؟

```
unsigned x = 0xabcd ;
int y = x ;
printf ("x = %x , y = %x" , x>>3 , y>>5) ;
```

الف) x = 1579 , y = F579
ب) x = F579 , y = 1579
ج) x = 5E68 , y = 79A0
د) x = 5E68 , y = 79AF

۷. در برنامهٔ زیر جمعاً چند عدد خوانده می‌شود؟

```
k = 2 ;
for (I = k+ + ; I<--k ; I--)
scanf ("%d" , &a) ;
```

الف) 7
ب) 3
ج) 6
د) 5

۸. خروجی دستورهای زیر چیست؟

```
int i = 0 ;
for (; i < = a ;)
printf (%d" , i + +) ;
```

الف) اعداد صحیح 0 الی a
ب) اعداد صحیح 1 الی 10
ج) اعداد صحیح 1 الی a
د) اعداد صحیح 0 الی 10

۹. خروجی دستورهای زیر در صورت فشار دادن کلید "b" چیست؟

```
switch (getchar())
{
case ´a´:
printf ("aaaa") ;
break ;
case ´b´:
printf ("bbbb") ;
default:
printf ("OTHER") ;
break ;
}
```

الف) bbbb
ب) OTHER
ج) aaaa
د) هیچ کدام

۱۰. با توجه به تعریف زیر، چاپ سومین عنصر آرایهٔ p با کدام دستور امکان پذیر است؟

```
static int p[ ] = {1 , 2 , 3 , 4 , 0} ;
```

الف) prrntf("%d", p[3]);
ب) prrntf("%d", *(p+3));
ج) prrntf("%d", *p[2]);
د) prrntf("%d", *(p+2));

۱۱. خروجی دستورهای زیر کدام است؟

```
int i , j , k = 0 ;
for (i = 0 ; i<5 ; + + i)
```

الف) 10

```
  for (j = 0 ; j<5 ; + + j)
    {
      k + = (i – j + 1) ;
      break ;
    }
printf ("%d" , k) ;
```

ب) 15
ج) 5
د) هیچ کدام

۱۲. عملکرد دستورهای زیر کدام است؟

```
char c1 ;
c1 = (c1>=65 && c1<=90) ? ('a' + cl -'A'): c1 ;
```

الف) تبدیل حرف کوچک به بزرگ
ب) تبدیل حرف بزرگ به کوچک
ج) تبدیل حرف به کد اسکی کد
د) تبدیل کد اسکی به حرف

۱۳. حداکثر مقدار i چند است؟

```
trans(3) ;
void trans(int n) ;
{
   static int i = 0 ;
  if (n>0)
    {
      trans (n-1) ;
      printf("%d" , + + i) ;
      trans(n-1) ;
    }
}
```

الف) ۳
ب) ۶
ج) ۷
د) هیچ کدام

۱۴. تابع زیر را در نظر بگیرید. کدام گزینه صحیح است؟

```
funct ()
{
   int x ;
   x = 100 ;
   x = x / 10 ;
}
```

الف) تعریف تابع غلط است زیرا دستور return ندارد.
ب) تابع مقدار ۱ را برمی‌گرداند.
ج) تابع مقدار صفر را برمی‌گرداند.
د) تابع مقداری برنمی‌گرداند.

۱۵. کدام جمله در مورد متغیرهای ایستا صحیح است؟

الف) قابل دستیابی در بیرون از تابع تعریف شده
ب) حفظ مقادیر قبلی هنگام اجرای مجدد تابع
ج) هر دو
د) هیچ کدام

۱۶. کدام جمله صحیح نیست؟

الف) آرایه‌های خودکار مقداردهی اولیه می‌شوند.

ب) آرایه‌های ایستا مقداردهی اولیه می‌شوند.

ج) وقتی که آرایه‌ای به تابعی به عنوان آرگومان فرستاده می‌شود Call by reference صورت می‌گیرد.

د) رشته را می‌توان آرایه‌ای از کاراکترها درنظر گرفت.

۱۷. کدام یک از جمله‌های زیر صحیح است؟

الف) متغیر short با short int برابر است.

ب) متغیر long با long int برابر است.

ج) قدرمطلق کمیتهای صحیح بدون علامت بزرگ‌تر از کمیتهای صحیح معمولی است.

د) هر سه مورد

۱۸. کدام گزینه صحیح است؟

الف) # define PI = 3.141593 ب) # define PI = 3.141593 ;

ج) # define PI 3.141593 ; د) هیچ کدام

۱۹. کدام گزینه صحیح است؟

الف) با استفاده از کاراکتر تبدیل (S) نمی‌توان رشته‌ای را که فاصلهٔ خالی دارد وارد کرد.

ب) بااستفاده‌از کاراکترهای‌محاط‌شدهٔ [\n^]% می‌توان رشتهٔ شامل‌فاصله را نیز وارد کرد.

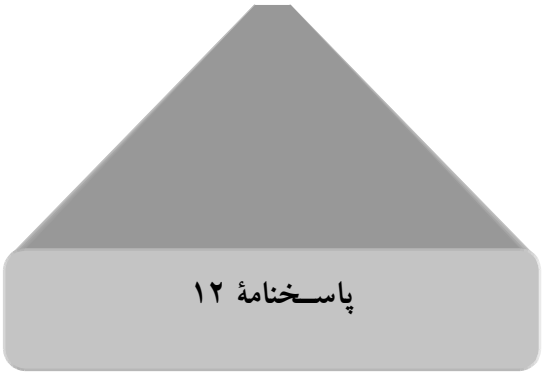
ج) تابع scanf برای آرایه‌ها و رشته‌ها نیازی به علامت & ندارد.

د) هر سه مورد

۲۰. روش غیرمستقیم دسترسی به داده‌ها با کدام گزینه امکان‌پذیر است؟

الف) آرایه ب) اشاره‌گر

ج) ساختار د) یونیون



پاســخنامهٔ ۱۲

	الف	ب	ج	د		الف	ب	ج	د
۱			●		۱۱		●		
۲			●		۱۲		●		
۳		●			۱۳			●	
۴	●				۱۴			●	
۵			●		۱۵		●		
۶	●				۱۶	●			
۷	●				۱۷				●
۸	●				۱۸				●
۹				●	۱۹				●
۱۰				●	۲۰		●		

تمرینهای اضافی

۱. برنامه‌ای بنویسید که رشته‌ای را از ورودی دریافت ، کاراکتر اول آن را بزرگ و کل رشته را مجدد چاپ کند.

۲. برنامه‌ای بنویسید که رشته‌ای را از ورودی دریافت کند، در صورتی که حرف a در آن وجود داشته باشد، آن را به x تبدیل کند.

۳. برنامه‌ای بنویسید که کاراکتری از ورودی دریافت و کد اسکی آن را به دسیمال و هگزادسیمال و اکتال چاپ کند.

۴. برنامه‌ای بنویسید که معادل هر کاراکتر را به رمز مورس تبدیل کند.

۵. برنامه‌ای بنویسید که عددی صحیح کوچک‌تر از ۱۰۰ را از ورودی بخواند و آن را با حروف تایپ کند. (مثلاً ورودی ۲۶؛ خروجی twenty six)

۶. برنامه‌ای بنویسید که جمله‌ای را از ورودی بخواند، و تعداد کلمات و تعداد کل کاراکترهای آن را در خروجی چاپ کند.

۷. برنامه‌ای بنویسید که عددی کوچک‌تر از ۱۰۰ از ورودی بخواند. در صورتی که عدد اول باشد، لگاریتم آن را محاسبه و چاپ کند.

۸. برنامه‌ای بنویسید که عددی از ورودی بخواند و مشخص کند این عدد مثبت است یا منفی و صحیح است یا اعشاری. همچنین تعداد ارقام آن را مشخص کند.

۹. برنامه‌ای بنویسید که عددی از ورودی بخواند و مشخص کند که این عدد جزء سری فیبوناچی است یا نه.

۱۰. برنامه‌ایبنویسیدکه کلیهٔ کاراکترها را به همراه کد اسکی آنها در دو ستون چاپ کند.

۱۱. برنامه‌ای بنویسید که تمام حالتهایی را که می‌توان یک سکه ۱۰ ریالی را به سکه‌های ۵ و ۲

و ۱ ریالی خرد کرد محاسبه و چاپ کند.

۱۲. برنامه‌ای بنویسید که n عدد از ورودی بگیرد، و میانگین، میانه و انحراف از معیار آنها را چاپ کند.

۱۳. برنامه ای بنویسید که خروجی آن به شکل زیر باشد.

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

۱۴. برنامه‌ای بنویسید که خروجی آن به شکل زیر باشد.

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

۱۵. برنامه‌ای بنویسید که خروجی آن به شکل زیر باشد.

```
 1
 1 2
 1 2 3
 1 2 3 4
 1 2 3 4 5
```

۱۶. برنامه‌ای بنویسید که خروجی آن به شکل زیر باشد.

```
1
232
34543
4567654
56787654
```

۱۷. برنامه‌ای بنویسید که مقدار درآمد(income) را بخواند و میزان مالیات (tax) را برای هر مقدار بر حسب جدول زیر محاسبه و دو مقدار درآمد و مالیات را چاپ کند.

نرخ مالیات	مقدار درآمد
۵ درصد	کمتر از ۱۰۰۰۰۰ ریال
۱۰ درصد	بین ۱۰۰۰۰۰ تا ۲۰۰۰۰۰ ریال
۱۵ درصد	بیش از ۲۰۰۰۰۰ زیال

۱۸. برنامه‌ای بنویسید که مقادیر f(x) را با توجه به تابع $f(x)= 3x^2 + 5x + 1$ محاسبه و چاپ کند. حد متغیر x بین 0.0 تا 3.0 و واحد تغییرات بین دو حد مذکور 0.1 است.

۱۹. برنامه‌ای بنویسید که مقدار π =3.141592... را برای ۱۰۰ عنصر از رشتهٔ زیر با استفاده از این فرمول محاسبه کند:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots$$

۲۰. برنامه‌ای بنویسید که تمام اعداد دو رقمی را که در آنها رقم سمت راست کوچک‌تر از رقم سمت چپ است چاپ کند.

۲۱. برنامه‌ای بنویسید که تمام اعداد دو رقمی را که دو رقم یکسان دارند چاپ کند (مثل ۲۲ یا ۳۳).

۲۲. برنامه‌ای‌بنویسیدکه تمام‌اعداد سه‌رقمی را که در آنها رقم وسط صفر است چاپ‌کند.

۲۳. دو تاس را پرتاب می‌کنیم. برنامه‌ای بنویسید که نتایج پرتاب را مشخص کند.

۲۴. برنامه‌ای بنویسید که تمام اعداد چهار رقمی را که در آنها رقمهای اول و چهارم با هم و رقمهای دوم و سوم با هم قرینه‌اند، محاسبه و چاپ کند (مثلاً ۲۵۵۲).

۲۵. برنامه‌ای بنویسید که تمام اعداد کوچک‌تر از ۱۰۰۰ را که در آنها هر عدد برابر حاصل جمع دو عدد اول است مشخص و چاپ کند.

۲۶. برنامه‌ای بنویسید که از اعداد صحیح کوچک‌تر از ۱۰۰۰ آنهایی را که رابطهٔ $a^2 + b^2 = c^2$ در موردشان صادق است مشخص و چاپ کند.

۲۷. برنامه‌ای بنویسید که آرایه‌ای ۱۰ عنصری تعریف کند، به طوری که هر عنصر آن اندیس متناظر آرایه باشد.

۲۸. برنامه‌ای بنویسید که دو آرایهٔ m و n عنصری را merge و در خروجی چاپ کند.

۲۹. برنامه‌ای بنویسید که ماتریس ترانهادهٔ ماتریس $m \times n$ را مشخص و چاپ کند.

۳۰. برنامه‌ای بنویسید که عناصر قطر اصلی ماتریس مربعی را صفر کند.

۳۱. برنامه‌ای بنویسیدکه‌عناصر آرایهٔ n عنصری کاراکتری را به صورت نزولی مرتب کند.

۳۲. برنامه‌ای بنویسیدکه کوچک‌ترین‌عنصر آرایه‌ای $m \times n$ رابه‌همراه اندیس‌آن‌چاپ کند.

۳۳. برنامه‌ای بنویسید که در آن هر عنصر آرایه‌ای n عنصری را برابر معکوس اندیس آن عنصر

قرار دهد و چاپ کند.

۳۴. برنامه‌ای‌بنویسیدکه عددی از ورودی بخواند و ریشهٔ سوم آن‌را محاسبه و چاپ کند.

۳۵. برنامه‌ای بنویسید که تک‌تک عناصر آرایه‌ای n عنصری را معکوس و چاپ کند. (مثلاً x بشود $(\frac{1}{x})$

۳۶. برنامه‌ای‌بنویسیدکه عددی یک‌رقمی‌از ورودی‌بخواند و معادل رومی آن‌را چاپ کند.

۳۷. برنامه‌ای بنویسید که مقدار y را از رابطهٔ $y = X^{\frac{3}{4}}$ محاسبه و چاپ کند.

۳۸. برنامه‌ای بنویسید که مقدار y را از رابطهٔ $y = x^n$ محاسبه و چاپ کند (n عدد اعشاری است).

۳۹. برنامه‌ای بنویسید که با روش آزمایش و خطا ریشهٔ معادلهٔ $\acute{x} + 3\acute{x} - 1 = 0$ را محاسبه و چاپ کند.

۴۰. برنامه‌ای بنویسید که بتواند مقدار فاکتوریلهای بزرگ را محاسبه کند (مثلاً !50).

۴۱. برنامه‌ای بنویسید که تاریخ میلادی را دریافت و به تاریخ شمسی تبدیل کند.

۴۲. برنامه‌ای‌بنویسیدکه تاریخ‌میلادی را دریافت و مشخص کند چندمین روز سال است.

۴۳. برنامه‌ای‌بنویسیدکه تاریخ‌شمسی را دریافت و مشخص کند چندمین روز سال است.

۴۴. برنامه‌ای بنویسید که توابع مثلثاتی برای زوایای صفر تا نود درجه را محاسبه و به صورت جدولی نمایش دهد.

۴۵. فرمول بسط سینوس به شکل زیر است. برنامه‌ای بنویسید که مقادیر x را بخواند و sin آن را محاسبه و چاپ کند. جمله‌های رشته تا حدی ادامه می‌یابد که قدرمطلق هر جمله کمتر از 10^{-5} باشد (x بر حسب رادیان است).

$$Sin(x) = x - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \dots$$

۴۶. برنامه‌ای بنویسید که مشابه ماشین‌حساب عمل‌کند. برنامه باید چهارعمل‌اصلی، توان‌رسانی، ریشهٔدوم‌وسوم، لگاریتم، فاکتوریل‌و نیزحافظه‌برای ذخیره‌سازی عدد را شامل باشد.

۴۷. فرض کنید به مدت n سال مقدار a را در ابتدای هر سال به حساب پس انداز واریز

می‌کنید. اگر بهرهٔ حساب به میزان سالانه i درصد باشد، مجموع پول اندوخته شده پس از n سال با فرمول زیر به دست می‌آید. برنامه ای بنویسید که این فرمول را محاسبه کند.

$$Y = a[(1+ i/100) + (1+ i/100)^2 + (1+ i/100)^3 + \ldots + (1+ i/100)^n]$$

۴۸. برنامه‌ای بنویسید که جفت اعدادی را پیدا کند که حاصل جمعشان برابر حاصل ضربشان باشد.

۴۹. تابعی بنویسید که دو عدد صحیح از ورودی بخواند و کوچک‌ترین مضرب مشترک آنها را محاسبه کند.

۵۰. تابعی بنویسید که کاراکتری را در محل دلخواهی از رشته درج کند.

۵۱. تابعی بازگشتی بنویسید که فرمول جبری زیر را محاسبه کند.

$$Y = x_1 + x_2 + \ldots + x_n$$

۵۲. تابعی بازگشتی بنویسید که فرمول جبری زیر را محاسبه کند.

$$Y = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \frac{x^4}{24} + \ldots + (-1)^n x^n/n!$$

۵۳. تابعی بنویسید که سه بار خودش را فراخوانی کند.

۵۴. زمان را به صورت ساختاری شامل ساعت، دقیقه و ثانیه در نظر بگیرید. برنامه‌ای بنویسید که وقت فعلی را از ورودی بگیرد و مشخص کند که 5 ساعت و 42 دقیقه و 58 ثانیه بعد، وقت چیست؟

۵۵. با استفاده از داده‌های نوع شمارشی برنامه‌ای بنویسید که یکی از ارقام ۱ تا ۷ را بگیرد و روز متناظر با آن را نمایش دهد (برای مثال اگر عدد ۱ ورودی باشد، sunday نمایش داده شود).

۵۶. برنامه‌ای بنویسید که به انتهای فایل متنی، یک خط متن اضافه کند.

۵۷. برنامه‌ای بنویسید که فایلی متنی را بخواند. سپس هر کاراکتر آن را با مکمل آن جایگزین کند و در فایل دیگری بنویسد.

۵۸. برنامه‌ای بنویسید که فایلی را بخواند، محتوای آن را کدگذاری و در فایل دیگری ذخیره کند. سپس محتوای فایل جدید را از حالت کدگذاری خارج در خروجی استاندارد چاپ کند (برای کدگذاری فایل می‌توانید مثلاً کد اسکی هرکاراکتر را یک‌واحد افزایش دهید).

۵۹. برنامه‌ای بنویسید که داده‌های فایل متنی را از انتها به ابتدا بازنویسی کند.
۶۰. برنامه‌ای بنویسید که تعداد حروف صدادار موجود در فایل متنی را مشخص کند.
۶۱. برنامه‌ای بنویسید که تعداد خطهای فایل را مشخص کند.
۶۲. برنامه‌ای بنویسید که تعداد تکرار کلمه‌ای خاص را در فایل متنی مشخص کند.
۶۳. برنامه‌ای بنویسید که محتوای دو فایل متنی را در فایل سومی merge کند.
۶۴. برنامه‌ای بنویسید که داده‌های دو فایل متنی را با یکدیگر تعویض کند.
۶۵. برنامه‌ای بنویسید که در فایل متنی، کلمه‌ای خاص را جستجو کند و با پیغام مناسب گزارش دهد.
۶۶. برنامه‌ای بنویسید که حجم محتویات فایل متنی را مشخص کند(برحسب بایت).
۶۷. برنامه‌ای بنویسید که جدول شطرنجی (۸×۸) در فایل ایجاد و ذخیره کند.
۶۸. برنامه‌ای بنویسید که در فایل متنی، کلمه‌ای خاص را حذف کند.
۶۹. برنامه‌ای بنویسید که فهرستی از نام و شماره تلفنهای افراد را ذخیره کند. برنامه باید قابلیت اضافه کردن، حذف کردن و جستجوی اطلاعات را داشته باشد.
۷۰. برنامه‌ای بنویسید که فهرستی از مشخصات دانشجویی شامل نام، شمارهٔ دانشجویی و معدل را از ورودی دریافت و در فایلی ذخیره کند. برنامه باید قابلیت مرتب‌سازی رکوردها (sort) بر حسب نام و نیز شمارهٔ دانشجویی را داشته باشد و بتواند نفرات ممتاز اول تا سوم را مشخص کند.
۷۱. برنامه‌ای بنویسید که با استفاده از ماکرو، توان و فاکتوریل را محاسبه کند. برنامه باید با فشار کلید F1 توان و با فشار کلید F2 فاکتوریل را چاپ کند.

ضمیـمهٔ ۱

مجموعه کاراکترهای اسکی

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
000	NUL	032	**blank**	064	@	096	`
001	SOH	033	!	065	A	097	a
002	STX	034	“	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	"	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	'	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	y
021	ANK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	120	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	¦
029	GS	061	=	093	]	125	}
030	RS	062	>	094	↑	126	~
031	US	063	?	095	_	127	DEL

توجه. ۳۲ کاراکتر اول و همین طور آخرین کاراکتر، از کاراکترهای کنترلی است و قابل چاپ نیستند.

ضمیـمهٔ ۲

کاراکترهای فرمان (Escape Sequences)

Character	Escape sequence	ASCII value
bell	\a	007
backspace	\b	008
horizontal tab	\t	009
newline(line feed)	\n	010
vertical tab	\v	011
form feed	\f	012
carriage return	\r	013
quotation mark(″)	\″	034
apostrophe(′)	\'	039
question mark(?)	\?	063
backslash(\)	\\	092
null	\0	000
octal number	\000	(0 represents an octal digit)

ضمیـمهٔ ۳

جدول انواع داده‌ها در زبان C

نوع داده	اندازه به بیت	محدودهٔ قابل قبول
char	۸	۱۲۷- تا ۱۲۷
unsigned char	۸	۰ تا ۲۵۵
signed char	۸	۱۲۸- تا ۱۲۷
int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
Short	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
signed int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
signed short int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
unsigned int	۱۶	۰ تا ۶۵۵۳۵
unsigned short in	۱۶	۰ تا ۶۵۵۳۵
long int	۳۲	۶۴۸، ۴۸۳، ۱۴۷، ۲- تا ۶۴۷، ۴۸۳، ۱۴۷، ۲
signed long int	۳۲	۶۴۷، ۴۸۳، ۱۴۷، ۲- تا ۶۴۷، ۴۸۳، ۱۴۷، ۲
unsigned long int	۳۲	۰ تا ۲۹۵، ۹۶۷، ۲۹۴، ۲
float	۳۲	با ۶ یا ۷ رقم دقت (در فاصله $۱۰^{-۳۷}$ تا $۱۰^{+۳۷}$)
double	۶۴	با ۱۰ رقم دقت (در فاصله $۱۰^{-۳۷}$ تا $۱۰^{+۳۷}$)
long float	۶۴	با ۱۰ رقم دقت (در فاصله $۱۰^{-۳۷}$ تا $۱۰^{+۳۷}$)

ضمیـمهٔ ۴

جدول تقدم عملگرها

Precedence Group	Operators	Aassocitivity
function , array structure member , pointer to structure member	() [ ] →	L→R
unary operators	- ++ -- ! ~ * & sizeof (type)	R→L
arithmetic multiply, divide and remainder	* / %	L→R
arithmetic add and subtract	+ -	L→R
bitwise shift operators	<< >>	L→R
relational operators	< <= > >=	L→R
equality operators	= = !=	L→R
bitwise and	&	L→R
bitwise exclusive or	^	L→R
bitwise or	|	L→R
logical and	&&	L→R
logical or	||	L→R
conditional operator	?:	R→L
assignment operators	= += -= *= /= %= &= ^ = |= <<= >>=	R→L
comma operator	,	L→R

واژه‌نامهٔ انگلیسی ـ فارسی

account	حساب‌کردن
ampersand	علامت &
area	ناحیه
arithmetic	محاسباتی
array	آرایه
ASCII	کد اسکی
assignment	انتساب
auto	خودکار
average	میانگین
beep	صدای بوق کامپیوتر
binary	دودویی
blank	خالی
bubble sort	مرتب‌سازی حبابی
buffer	حافظهٔ واسط
built-in	درون‌ساخت
call	فراخوانی
case	مورد
circumflex	علامت ^
class	کلاس
comment	توضیح
compound statement	دستور مرکب
condition	وضعیت
conditional statement	دستور شرطی
constant	ثابت
control string	رشتهٔ کنترلی
count	شمارش
data	داده
decimal	دهدهی
default	پیش‌فرض
define	تعریف کردن
digit	رقم
display	نمایش
editor	ویرایشگر
enumerate	شمردن
equation	معادله
error	خطا
exclusive	انحصاری
exponent	توان
expression	عبارت
extern	خارجی
float point	ممیز شناور
formatting character	کاراکتر فرمت‌بندی
function	تابع

global variable	متغیرهای عمومی
header	عنوان
hexadecimal	سیستم عددی شانزده
identifier	شناسه
index	شاخص
information	اطلاعات
initialize	مقداردهی اولیه
input	ورودی
integer	عدد صحیح
interchange	مبادله‌کردن
label	برچسب
library	کتابخانه
liner search	جستجوبه‌روش‌خطی
linker	پیونددهنده
leader	بارکننده
local	محلی
logical	منطقی
loop	حلقه
main	اصلی
mark	نشانه
member	عضو
memory	حافظه
modifier	اصلاح‌کننده
module	ماژول، پیمانه
null	تهی
number	عدد
numeric	عددی
nested	آشیانه‌ای
octal	سیستم‌عددی‌هشت
operator	عملگر
output	خروجی
pointer	اشاره‌گر
precision	دقت
preprocessor	پیش‌پردازنده
prime	اولین
procedure	رویه
process	پردازش
program	برنامه
prototype	نمونهٔ اولیه
qualifier	توصیف‌کننده
random	تصادفی
recursive	بازگشتی
register	ثبت کردن
record	مرتب‌سازی
repetitive statement	دستور حلقهٔ تکرار
return	بازگشت
root	ریشه
runtime	زمان اجرا
save	ذخیره کردن

scope	دامنه
search	جستجو
selection sorting	مرتب‌سازی انتخابی
sequential	ترتیبی
signed	علامت‌دار
source	منبع
sort	مرتب‌سازی
space	فضا
statement	دستور
static	ایستا
storage	مخزن
string	رشته
structure	ساختار
subroutine	زیرروال
switch	تعویض
symbolic	نمادی
tag	برچسب
text	متن
type	نوع
unary	یکانی
underscore	زیرخط
union	اجتماع
unsigned	بی‌علامت
update	به روز آوری
user	کاربر
value	مقدار
variable	متغیر
vector	بردار
version	نسخه
void	بی‌اعتبار
warning	پیام هشدار