فصل اول – کلیات و مفاهیم

- **مقدمه**

زبان C در اوایل دهه ۱۹۷۰ میلادی توسط دنیس ریچی در لابراتوار کمپانی BELL و بعنوان زبان برنامه‌نویسی سیستمها طراحی گردید. این زبان از دو زبان پیشین بنامهای BCPL و B منتج شده است که این دو نیز در همین لابراتوار ساخته شده بودند ..زبان C تا سال ۱۹۷۸ منحصر به استفاده در همین لابراتوار بود تا اینکه توسط دو تن بنامهای ریچی و کرنیه نسخه نهایی این زبان منتشر شد . به سرعت کامپایلرها و مفسرهای متعددی از C توسعه یافت لیکن برای جلوگیری از ناسازگاری های ایجاد شده و نیز حفظ قابلیت حمل زبان ، تعاریف متحدالشکلی توسط استاندارد ANSI ارایه گردید . آنچه در این درس ارایه شده بر اساس همین استاندارد میباشد .

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

زبان C ، بطور گسترده ای در دسترس میباشد . کامپایلرهای تجارتی آن برای بیشتر کامپیوترهای شخصی ، مینی کامپیوترها و نیز در mainframe قابل استفاده اند .

C ، زبانی است همه منظوره ، ساختیافته سطح بالا (مانند زبان پاسکال و فرترن) و انعطاف‌پذیر که برخی از خصوصیات زبانهای سطح پایین را که معمولاً در اسمبلی یا زبان ماشین موجود است ، نیز دارا می‌باشد . در عین حال C برای کاربردهای ویژه طراحی نشده است و می‌توان از آن در همهٔ زمینه‌ها ، بخصوص به لحاظ نزدیکی آن به زبان ماشین در برنامه نویسی سیستم استفاده کرد . بنابراین C بین زبانهای سطح بالا و سطح پایین قرار دارد و در نتیجه اجازه می‌دهد که برنامه نویس خصوصیات هر دو گروه زبان را بکار برد . از اینرو در بسیاری از کاربردهای مهندسی بطور انحصاری زبان C را بکار 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 بطور کلی مستقل از ماشین یا نوع کامپیوتر است و تقریباً تحت کنترل هر سیستم عاملی ، اجرا میگردد .

کامپایلرهای C معمولاً فشرده و کم حجم می‌باشد و برنامه‌های هدف ایجاد شده بوسیلهٔ آنها در مقایسه با سایر زبانهای برنامه‌سازی سطح بالا ، خیلی کوچک و کارآمد می‌باشد . (کامپایلر یا مفسر ، خود برنامه ای کامپیوتری است که برنامه سطح بالا را بعنوان یک داده ورودی می پذیرد و برنامه ایجاد شده به زبان ماشین را بعنوان خروجی ایجاد می کند .)

برنامه‌های C در مقایسه با سایر زبانهای برنامه‌سازی سطح بالا ، به‌راحتی قابل انتقال می‌باشند . دلیل این کار آن 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 خصوصیات و ویژگیهای کامپیوتر میزبان مربوط نوشته شده است . این توابع کتابخانه‌ای تا حدودی استاندارد می‌باشد و معمولاً هر تابع کتابخانه‌ای در نسخه‌های متعدد C بشکل یکسان قابل دسترسی می‌باشد .

C ، روش برنامه‌سازی ماژولار را پشتیبانی می‌کند . همچنین از نظر عملگرها نیز یک زبان قوی بوده و شامل اپراتورهای گوناگونی برای دستکاری روی داده‌ها در سطح bit می‌باشد .

بطور کلی جامعیت ، عمومیت ، خوانایی ، سادگی ، کارآیی و پیمانه ای بودن که همگی از مشخصات یک برنامه ایده آل است توسط زبان C قابل پیاده سازی میباشد .

ویژگیهای فوق موجب شده زبان C بعنوان یکی از قویترین و محبوبترین زبانهای برنامه سازی دنیا مطرح شود .

- **کاراکترها (Characters)**

زبان برنامه‌نویسی C مجموعه‌ای خاص از کاراکترها را شناسایی می‌کند. این مجموعه که در حکم مصالح اولیه جهت شکل دادن به اجزا اصلی برنامه هستند عبارتند از :

- **حروف بزرگ و کوچک :** زبان C برخلاف بعضی زبانها مثلاً پاسکال بین حروف بزرگ و کوچک فرق می‌گذارد . مثلاً FOR با for یکسان نیست .
- **ارقام دهدهی :** شامل ۰ تا ۹
- **کاراکترهای مخصوص :** شامل ~ [ ] { } < > ? * / = - + ! @ # $ % ^
- **جای خالی یا Blank**
- **کاراکترهای فرمت‌دادن یا Formating Characters** ، که عبارتند از :

کاراکتر خط جدید (New Line) یا \n

کاراکتر برگشت به عقب (Back Space) یا \b

کاراکتر Horizontal Tab یا \t

کاراکتر Vertical Tab یا \v

کاراکتر تغذیه فرم (Form Feed) یا \f

کاراکتر ابتدای سطر یا \r

کاراکتر تهی یا \0

و غیره که با کاربرد آنها آشنا خواهید شد.

- **شناسه‌ها (Identifiers)**

شناسه‌ها علایم سمبولیکی هستند که برای مراجعه به انواع داده‌ها مانند مقادیر ثابت ، متغیرها ، نوعها و توابع بکار برده می‌شوند. به عبارتی دیگر شناسه‌ها اسامی هستند که به عناصر مختلف برنامه مانند متغیرها ، توابع و آرایه‌ها اختصاص داده می‌شود. یک شناسه C دنباله‌ای است از حروف ، ارقام یا

علامت زیر خط که با حروف یا علامت زیر خطدار شروع می‌شود. برحسب قرارداد شناسه‌هایی که با Underscore شروع می‌گردند فقط در برنامه‌های سیستم کاربرد دارند و در برنامه‌های کاربردی غیرقابل استفاده‌اند. در انتخاب طول اسامی یا تعداد کاراکترهای آن از نظر C محدودیتی وجود ندارد گرچه هر زبان قواعد و محدودیت خاص خودش را بکار می‌برد . طول اسامی در زبان C استاندارد تا ۳۱ کاراکتر مجاز است .

مثال – اسامی زیر شناسه های معتبر هستند :

x1 , sum , payam_noor , maximum

مثال – اسامی زیر شناسه های معتبر نیستند :

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

- **متغیرها (Variables)**

متغیرها در زبان C شناسه‌هایی هستند که محلهایی از حافظه را به خود اختصاص می‌دهند. یک متغیر ترکیبی است از ارقام ، حروف و علامت زیرخط (_) . لازم به ذکر است متغیرهایی که با علامت زیر خط شروع می شوند برای متغیرهای داخلی سیستم رزرو شده‌اند. طول هر متغیر در ANSI استاندارد تا ۳۱ کاراکتر است ، ولی در بعضی از کامپایلرهای قدیمی زبان محدود به ۸ کاراکتر می‌باشد.

بعضی از شناسه‌های زبان C کلمات رزرو شده یا کلیدی هستند. یعنی معنی و مفهوم آن از قبل در زبان تعریف و پیش‌بینی شده است. بنابراین نمی‌توانند در برنامه به عنوان شناسه‌های تعریف شده به وسیله برنامه‌نویس بکار برده شوند . متداولترین کلمات کلیدی C به شرح زیر است :

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

البته در بعضی از کامپایلرهای زبان C ممکن است کلمات کلیدی دیگری نیز وجود داشته باشد که باید به کتاب راهنمای مربوطه مراجعه شود . توجه داشته باشید که همه کلمات کلیدی با حروف کوچک نوشته می‌شود پس main کلمه کلیدی است درحالی که Main کلمه کلیدی نمی‌باشد زیرا حرف اول آن بزرگ است . همینطور void کلمه کلیدی است اما VOID کلمه کلیدی نیست .

- **علامت توضیح (Comment)**

در زبان C هر عبارتی که بین دو علامت /* و */ قرار گیرد صرفاً بعنوان توضیحات محسوب می‌گردند . مثلاً اگر بخواهیم در مورد یک دستور توضیح دهیم که چه کاری را انجام می‌دهد در هر

جای برنامه که فضای خالی مجاز باشد می‌توان برای توضیح از علامت فوق استفاده کرد. (در اغلب نسخه ها علامت // هم مجاز است .)

مثال – در برنامه 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 برنامه C**

همه برنامه‌های C شامل یک یا چندین تابع هستند که یکی از آنها تابع اصلی یا main نامیده می‌شود . هر برنامه فقط یک تابع اصلی دارد و برنامه همیشه با اجرای تابع اصلی آغاز می‌گردد. تعریف توابع دیگر ممکن است قبل یا بعد از تابع اصلی قرار گیرد . بطور کلی می‌توان گفت که هر برنامه به زبان C حداقل دارای اجزای مقدماتی بترتیب زیر است :

```
main( )                          تابع اصلی
{                                شروع تابع اصلی
 variables declaration ;         تعریف متغیرها
 program statements ;            دستورات برنامه
}                                پایان تابع اصلی
```

- **دستورالعملهای اجرایی**

در هر برنامه دستورالعملهای اجرایی باید بعد از تعریف متغیرها درج شوند . دستوری قابلیت اجرا دارد که در پایان آن دستور، یک علامت سمی کولون (;) نوشته شود .

برای فهم بهتر این موضوع اولین برنامه را ارائه و سپس به تشریح آن می‌پردازیم .

مثال – برنامه‌ای بنویسید که مساحت مستطیلی به طول ۶ و عرض ۳ را محاسبه کرده و چاپ کند.

حل : برنامه مورد نظر بشکل زیر میباشد :

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

پس از اجرای برنامه ، خروجی برنامه بصورت زیر نمایش داده می‌شود :

```
area  = 18
```

حال به تشریح برنامه می پردازیم .

خط اول برنامه اعلان می‌کند که کتابخانه مربوط به توابع ورودی و خروجی برای دستیابی به توابع آن آماده شود. C یکی از زبانهایی است که به لحاظ داشتن توابع توکار (از پیش فرض شده) بسیار غنی می‌باشد. هر مجموعه توابعی که عملیات ویژه‌ای را انجام می‌دهد در یک مجموعه تحت 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) بوده و h نیز معرف header یا عنوان است. تابع printf نیز یکی از توابع خروجی می‌باشد .

حال به توضیح #include می‌پردازیم . برنامه‌های نوشته شده به زبان C قبل از اینکه به‌وسیله کامپایلر ترجمه گردد در اختیار یک برنامه دیگری تحت عنوان پیش‌پردازنده یا preprocessor قرار می‌گیرد . یکی از کاربردهای اصلی این برنامه آن است که کتابخانه‌های مورد نیاز برنامه منبع را یعنی کتابخانه‌هایی را که توابع بکار رفته در برنامه منبع را شامل است برای استفاده آماده می‌نماید . این کار بوسیله دستور include که در ابتدای آن علامت # و به دنبال آن نام کتابخانه در داخل علامت :

" " یا <>

می‌آید انجام می‌گیرد که اولین عبارت در برنامه بالا همین کار را برای ما انجام می‌دهد .

در خط بعد تابع اصلی تعریف شده است . پنج خط بعدی متن برنامه اصلی را تشکیل می‌دهند که از پنج دستور ساده تشکیل شده است. پایان هر دستور را سمی کولون (;) مشخص می‌نماید چون متن برنامه بیش از یک دستور است مجموع آنها بعنوان دستور مرکب یا یک بلاک در داخل یک زوج آکولاد قرارداده می‌شود. درواقع هر آکولاد چپ برای کامپایلر C به معنی شروع بلاک و هر آکولاد

راست معرف پایان آن است . البته در بعضی مواقع برحسب مورد آکولادها را می‌توان بصورت تودرتو نیز بکار برد .

اولین دستور در متن برنامه یا همان شروع آکولاد ، توصیف متغیرها است. سه دستور بعدی دستورات محاسباتی و جایگزینی می‌باشند و در آخر نیز دستور خروجی است که در آن تابع printf برای چاپ فرمت دار می‌باشد . اولین آرگومان تابع مذکور متن داخل گیومه می‌باشد که تابع آن را به همان صورت در خروجی چاپ می‌کند البته در بعضی از قسمتهای متن که شامل علامت % باشد مانند %d به کامپایلر 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 بطور کامل به بحث فرمت متغیرها خواهیم پرداخت .

مثال – برنامه‌ای بنویسید که طول و عرض مستطیلی را که به ترتیب a و b نامیده می‌شود از طریق دستگاه ورودی استاندارد خوانده و با فراخواندن تابعی به نام rectangle مساحت آن محاسبه گردیده ، سپس طول ، عرض و مساحت در برنامه یا تابع اصلی روی دستگاه استاندارد خروجی ، نمایش داده شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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 = ٥ و b = ٤ باشد ، خروجی برنامه مذکور بصورت زیر خواهد بود :

length = 5 width = 4 area = 20

توضیح – خط اول تا چهارم مانند برنامه قبلی است . در خط پنجم تابع فرعی Rectangle اعلان شده است که مقدار صحیح برمی‌گرداند و آرگومانهای آن نیز a و b می‌باشند که مقادیر صحیح هستند . در خط ششم ، تابع ورودی استاندارد scanf بکار رفته است . این تابع که جزءِ کتابخانه stdio.h است ، اطلاعات را از طریق ورودی استاندارد که صفحه کلید می‌باشد ، دریافت می‌کند . فرمت و مکانیسم

کار این گونه توابع بعد تشریح خواهد شد . در اینجا یادآور می‌شویم که فرم کلی تابع مزبور بصورت:

scanf (control string , argl , arg2 , … , argn) ;

است که در آن رشتهٔ کنترلی (control string) که در داخل گیومه (" ")بکار برده می‌شود ، اطلاعات مورد نیاز دربارهٔ فرمت اقلام داده‌های ورودی را شامل است و عناصر :

arg1 , arg2 , ... argn

نیز آرگومانهایی هستند که اقلام داده های ورودی را معرفی می‌نمایند . در این دستور حرف Ampersand ، یعنی "&" اپراتور یا عملگر آدرس است . پس &a و &b اشاره‌گر می‌باشند و معرف آنند که دو مقدار به آدرسهایی از حافظه به نامهای a و b خوانده شود . در رشته کنترلی نیز که در داخل گیومه قرار دارد ، از چپ به راست %d اول معرف فرمت اولین آرگومان یا داده ورودی بعنوان عدد صحیح است و بدنبال آن %d دوم نیز معرف فرمت دومین آرگومان بعنوان عدد صحیح است . درواقع آرگومانها ، معرف اشاره‌گرهایی هستند که آدرس اقلام داده‌ها را در داخل حافظه کامپیوتر مشخص می‌سازند .علامت \n در رشته کنترلی تابع printf ، موجب انتقال به سطر جدید می‌گردد . بنابراین اطلاعات بعدی در سطر جدید نشان داده خواهد شد .

- **تمرین و پاسخ**

تمرین ۱ – کدامیک از اسامی زیر مجاز است بعنوان نام متغیردر برنامه بکار برده شود؟

حل : پاسخ مورد نظر در مقابل هر اسم در جدول زیر نمایش داده شده است :

Integer	مجاز است
-19	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز (-)
Lesson four	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز فاصله
Unit_25	مجاز است
define	مجاز نیست ، بدلیل کلمه کلیدی بودن
Loop2	مجاز است
Star565void	مجاز است
Please?	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز (؟)
computer_ p_ q	مجاز است
C+ +	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز (+)
S#	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز (#)
Five$	مجاز نیست ، بدلیل وجود کاراکتر غیر مجاز ($)

تمرین ۲ – برنامه زیر مساحت مربعی به ضلع ۵ را محاسبه کرده و چاپ می کند . قسمتهای

مختلف آنرا شرح دهید .

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

حل : 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 شده اند . سه خط بعد دستورات اصلی برنامه است که شامل مقدار دهی اولیه متغیر x ، محاسبه مساحت درمتغیر S وچاپ آن در خروجی میباشد . آکولاد بسته پایان تابع اصلی است .

تمرین ۳ – ویژگیهای یک برنامه ایده آل را شرح دهید .

حل : پاسخ مورد نظر به شرح زیر است :

تمامیت یا جامعیت : به معنای درستی محاسبه است . باید مشخص باشد که هر چیز دیگری که به برنامه افزوده می شود اگر در درستی اجرای برنامه نقشی نداشته باشد بی معنی است .

وضوح : به معنای خوانا بودن برنامه با تاکید بر اهمیت منطقی آن است . در اینصورت تعقیب منطق برنامه بدون نیاز به تشریح برای هر برنامه نویس دیگر امکان پذیر خواهد بود .

سادگی : وضوح و درستی برنامه که معمولاً با آسان نوشتن آن حفظ میشود از جمله موضوعات مرتبط با شکل برنامه هستند .

کارآیی : مربوط به سرعت اجرا و بهره وری کار حافظه میباشد .

ماژولار یا پیمانه ای بودن : بسیاری از برنامه ها میتوانند به مجموعه ای از زیر برنامه ها تجزیه شوند . این کار امکان تغییرات آتی برنامه را افزایش میدهد .

عمومیت : در برنامه از پارامترهایی استفاده شود که بتواند با مقادیر مختلف داده ها به درستی کار کند .

تمرین ٤ – چند نمونه از زبانهای سطح بالا ، سطح پایین و سطح میانی را نام ببرید .

حل : پاسخ مورد نظر به شرح زیر است :

زبان سطح بالا : پاسکال ، فرترن ، کوبول ، بیسیک

زبان سطح پایین : زبان ماشین و اسمبلی

زبان سطح میانی : زبان C و forth

فصل دوم- انواع داده‌ها

- **مقدمه**

دسته‌بندی داده‌ها به انواع مختلف ، یکی از تواناییهای مدرن زبانهای برنامه‌نویسی است. زبان C مجموعه کاملی از انواع داده‌ها را پشتبانی می‌کند که آنها را می‌توان به روشهای زیر دسته‌بندی کرد :

- داده‌های از نوع صحیح (integer)
- داده‌های از نوع اعشاری (floating point)
- داده‌های از نوع کاراکتر(character)

همچنین داده‌ها در زبانهای برنامه نویسی بصورت مقادیر ثابت و مقادیر متغیر بکار برده می‌شوند. مقادیر ثابت ، مقادیری هستند که در طول برنامه تغییر نمی‌کنند، اما متغیرها می‌توانند در طول اجرای برنامه، مقادیر مختلفی از داده‌ها را بپذیرند .

مقادیر صحیح و 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 خود هستند. علاوه بر داده‌هایی از نوع اسکالر، داده‌هایی از نوع مجموعه‌ای وجود دارند که شامل: آرایه ، رکورد ، ساختار و اجتماع می‌باشند ، که در سازماندهی متغیرهایی که بطور منطقی به یکدیگر مرتبط هستند، مفید می‌باشند. این نوع داده‌ها نیز در فصول بعدی مورد بررسی قرار خواهند گرفت.

- **اعلان متغیرها**

در زبان C هر متغیر، پیش از آنکه در دستوری از برنامه بکار برده شود، باید تعریف گردد . دستورات مربوط به تعریف کردن متغیرها، اطلاعات لازم در مورد نوع داده‌هایی که متغیرهای مورد نظر می‌پذیرند و اینکه چند بایت حافظه اشغال می‌کنند و چگونگی تفسیر آنها را در اختیار کامپایلر قرار می‌دهد. برای اعلان یا تعریف متغیرهایی از نوع integer کلمه کلیدی یا کلمه رزرو شده int را نوشته و به دنبال آن اسامی متغیرهای مورد نظر را که با کاما از یکدیگر تفکیک می‌گردند می‌نویسیم، مانند:

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

البته می‌توان هریک از متغیرها را در دستوری جداگانه و همین‌طور در سطری جداگانه معرفی کرد، مانند:

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

که در سطر اول دو متغیر با دو دستور جداگانه اعلان شده و متغیر سوم نیز در سطر جدید با دستور جداگانه اعلان شده است. واضح است روش اول که در آن هر سه متغیر در یک سطر و با یک دستور اعلان شده است، ساده‌تر می‌باشد. بنابراین کلمه رزرو شده int داده‌هایی از نوع صحیح را مشخص می‌کند. نه کلمه کلیدی برای اعلان داده‌هایی از نوع اسکالر وجود دارد که در جدول زیر نشان داده شده است .

کلمات کلیدی برای توصیف متغیرها

اصلی	اصلاح کننده
int float char double enum	short long signed unsigned

پنج کلمه ستون اول نوع اصلی یا پایه‌ای هستند. چهار کلمه ستون دوم را اصلاح‌کننده یا modifier و نیز توصیف‌کننده یا qualifier نامند که به طریقی پنج نوع اصلی را توصیف می‌کنند. بعبارت دیگر

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

هرگونه اعلان متغیرها در داخل یک بلاک باید قبل از اولین دستور اجرایی ظاهر شود. به هرحال ترتیب اعلان آنها فرق نمی‌کند. بعنوان مثال دو روش اعلان زیر از نظر نتیجه عملکرد یکسان است :

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

- **انواع مقادیر صحیح**

زبان C از لحاظ بزرگی عناصر و همچنین از نظر نمایش داخلی آنها استاندارد ویژه‌ای بکار نمی‌برد. بطور کلی اعداد صحیح مثبت ، منفی و صفر و نیز متغیرهایی که مقادیر صحیح را می‌پذیرند، ۱۶ یا ۳۲ بیت حافظه اشغال می‌کنند و در کامپیوتر های IBM و سازگار با آن، منفی آنها به فرم متمم ۲ یا :

2'complement

می‌باشد. فرم اولیه داده‌هایی از نوع صحیح، همان int بعنوان 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

یا ترکیبی از آنها نیز ممکن است بکار برده شود .

داده‌هایی که با این کلمات توصیف می‌گردند، ممکن است از کامپایلری به کامپایلر دیگر تفسیر متفاوت داشته باشند. ولی اساس آنها یکسان است. اگر مقادیر صحیح در یک کامپایلر در حالت عادی ۲ بایت باشد، بین short int وint فرقی نخواهد بود و هر دو ۱۶ بیت یا ۲ بایت حافظه بکار خواهند برد. در ضمن short int را می‌توان فقط بصورت short نیز بکار برد (یعنی پیش فرض آن است که short همان short int می‌باشد). در چنین حالتی long int نیز ٤ بایت حافظه اشغال خواهد کرد که آن را هم می‌توان فقط بصورت long بکار برد (یعنی در اینجا نیز پیش فرض آن است که long همان long int می‌باشد) ولی چنانچه مقادیر صحیح در حالت عادی ٤ بایت حافظه اشغال نماید، short int یا فقط short ۲ بایت حافظه بکار خواهد برد. اما بین long int , int (یا فقط long) تفاوتی وجود نخواهد داشت و هر دو ٤ بایت حافظه اشغال خواهند نمود.

در مواردی متغیرها ، فقط دارای مقادیر غیرمنفی خواهند بود ، مثلاً متغیری که برای شمارش بکار برده می‌شود، یکی از این موارد است. زبان C اجازه می‌دهد که این گونه متغیرها را با بکار بردن توصیف‌کننده unsigned ، بدون علامت اعلان کنیم. یک مقدار صحیح بدون علامت از نظر میزان

حافظه 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 بکار می‌رود و در نتیجه مقادیر صحیح بدون علامت همیشه غیرمنفی بوده و بزرگی آن می‌تواند 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 توصیف کننده signed پیش‌بینی نشده است. ولی استاندارد ANSI آن را پشتیبانی می‌کند. در اغلب حالات به‌صورت پیش‌فرض، متغیرها signed هستند. لذا نیازی به بکاربردن توصیف کننده signed در آنها نخواهد بود. یک استثنا بر این حالت داده‌هایی از نوع کاراکتر char type است که بصورت پیش‌فرض می‌تواند signed یا unsigned باشد که بستگی به کامپایلر دارد. در اغلب کامپایلرها پیش‌فرض signed char می‌باشد .

متغیرهایی که معرف اعداد صحیح هستند، می‌توانند بصورتهای زیر توصیف گردند:

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

مثال – در زیر نمونه هایی از نحوه معرفی متغیرهایی از نوع 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  tik , tak ;
```

تعریف متغیرها از نوع long int و long هم‌ارز است ، بنابراین مثال ۱ را می‌توان بصورت :

```
long  temp , Pnoor ;
```

نوشت. همینچنین short , short int نیز معادل هم می‌باشند، پس مثال ۲ را می‌توان بصورت :

```
short  y1 , y2 , y3 ;
```

نوشت. مثال ۳ را نیز می‌توان به این شکل نوشت :

```
unsigned  m , n  ;
```

- **داده های کاراکتری**

یکی از انواع داده هایی که در برنامه نویسی استفاده میشود داده های کاراکتری است . در بسیاری از زبانهای برنامه سازی داده‌های عددی و داده‌های کاراکتری با یکدیگر تفاوت دارند . مثلاً عدد ۲ یک داده عددی و حرف A داده کاراکتری درنظر گرفته می‌شوند. در عمل هم ، کاراکترها بعنوان اعداد در حافظه کامپیوتر ذخیره می‌گردند، و هر کاراکتر دارای یک کد عددی است. کدهای مختلفی وجود دارد که دو نوع آن به نام ascii به مفهوم کد استاندارد آمریکایی برای تبادل اطلاعات یا :

American standard code for information interchange

و دیگری ebcdic به مفهوم کد توسعه یافته bcd یا :

Extended binary-coded decimal interchange

که IBM روی سیستم بزرگ خود بکار می‌برد، بیشتر، متداول است . ه البته در زبان C ، کد نوع ascii متداول است. در همه کدگذاریها برای هر کارکتر یک نماد عددی وابسته است که در مورد کدگذاری آسکی به آن ascii code گویند، که مقدار آن در فاصله صفر تا ۲۵۵ واقع 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 یک مقدار صحیح یک بایتی است که می‌تواند هم برای نگهداری اعداد و هم برای نگهداری کاراکترها بکار برده شود.

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

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

مقدار a برابر ۵۳ ، یعنی برابر آسکی کد کاراکتر ‘5‘ و مقدار b برابر ۵ خواهد بود .

مثال – برنامه زیر یک کاراکتر را از ورودی خوانده 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 بعنوان فرمت متغیرهای کاراکتری بکار برده می‌شود مانند %d که برای متغیرهای مقادیر صحیح و %f که برای متغیرهای مقادیر اعشار بکار برده میشوند (توابع ورودی و خروجی و فرمت متغیرها در فصل چهار مورد بررسی قرار خواهند گرفت .)

در مجموعه کاراکتر اسکی ، کد کاراکترها دارای ترتیبی براساس همان ترتیب کاراکترها هستند برای مثال کد حروف بزرگ :

‘A‘ برابر ۶۵

‘B‘ برابر ۶۶

... ...

‘Z‘ برابر ۹۰

می‌باشد. کد حروف کوچک از ۹۱ تا ۱۲۲ است. با توجه به ارزش کد حروف، تابع زیر حروف بزرگ را به حروف کوچک تبدیل می‌کند. (توابع در فصل شش بررسی خواهد شد .) :

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

تابع مذکور به اسکی کد هر کاراکتر دریافتی ۳۲ واحد اضافه می‌کند که درنتیجه حروف بزرگ به حروف کوچک تبدیل می‌شود. مثلاً اسکی کد حرف ‘A‘ که ۶۵ می‌باشد، ۳۲ واحد از اسکی کد حرف ‘a‘ که ۹۷ می‌باشد، کوچکتر است. مشابه آن می‌توان تابعی برای تبدیل حروف کوچک به بزرگ تعریف کرد، که در این حالت باید از اسکی کد حرف مورد نظر ۳۲ واحد کسر گردد تا به حرف بزرگ مشابه خود تبدیل شود .

در سیستم کدگذاری غیر ascii مانند ebcdic، تفاوت عددی کد حروف بزرگ و کوچک ۳۲ نمی‌باشد. بنابراین در چنین حالتی تابع تعریف شده بالا نتیجه مطلوب را نمی‌دهد. برای جلوگیری از چنین اشتباهی زبان C دارای توابع کتابخانه‌ای به اسامی:

toupper , tolower

است که به ترتیب عمل تبدیل کاراکترها از بزرگ به کوچک و از کوچک به بزرگ را انجام می‌دهند.

- **مقادیر ثابت صحیح**

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

0 , 12 , 35 , +4356 , 76592

در C بطور پیش فرض اعداد صحیح در مبنای ۱۰ تعریف شده اند. اما مبناهای ۸ و ۱۶ نیز کاربرد زیادی دارند. زیرا ۸ و ۱۶ توانهایی از مبنای ۲ بوده و این گونه سیستم های عددنویسی برای کامپیوترها مناسب تر است. برای مثال عدد ۶۵۵۳۶ در یک ماشین ۱۶ بیت همان عدد ۱۰۰۰۰ در مبنای ۱۶ می‌باشد .

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 بوده و پیشوند ندارد .بنابراین در مورد اعداد :

0531 +04163 , -0326 ,

صفر سمت چپ به معنای آن است که اعداد مزبور در مبنای ۸ می‌باشند. لذا اگر عدد در مبنای ۱۰ باشد، اولین رقم سمت چپ آن نمی‌تواند صفر باشد. بدیهی است در مبنای ۸ فقط هشت نشانه صفر تا ۷ بعنوان ارقام بکار برده می‌شوند. همچنین در مبنای ۱۶ نیز، شانزده نشانه مختلف بکار برده خواهد شد که ده نشانه آن همان نشانه‌های متداول در مبنای ۱۰ یعنی صفر تا ۹ میبا شد و شش نشانه دیگر حروف :

A , B , C , D , E , F

می‌باشد که به ترتیب معادل :

10 , 11 , 12 , 13 , 14 , 15

در مبنای ۱۰ هستند. مثالهای زیر نمونه‌ای از اعداد مبنای ۱۶ هستند :

0x15 , 0x327 , 0x5AB , 0xFE6

اگر طول هر کلمه ماشین مورد نظر ۱۶ بیت باشد ، طول آنها از 32k- تا 32k+ یعنی از 32768- تا 32767+ که معادل با 2^{15}-1 و یا معادل 077777 مبنای ۸ و یا 7FF مبنای ۱۶ است، تغییر خواهد کرد. ولی اگر طول هر کلمه ۳۲ بیت باشد طول آنها از 2G- تا 2G+ یعنی از :

-2, 147,483, 648 تا 2, 147, 483,647

که معادل 2^{31}-1 است ، خواهد بود .

مقادیر ثابت صحیح بدون علامت یا unsigned integer constants با قرار دادن u حرف اول کلمه 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 می‌تواند به هر دو صورت حروف بزرگ یا کوچک نوشته شوند. 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 می‌گردد. جدول زیر مثالهایی از مقادیر ثابت صحیح را با درنظر گرفتن موارد بالا نشان می‌دهد.

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 بعنوان مشخص‌کنندهٔ فرمت یا format specifier در رشته کنترل فرمت بکار می‌برد.

مثالهایی از مقادیر ثابت صحیح بدون علامت و طولانی

مقدار ثابت	سیستم عددنویسی
56780u	مبنای 10 (بدون علامت)
123456789L	مبنای 10 (طولانی)
123456789uL	مبنای 10 (بدون علامت و طولانی)
0123456uL	مبنای 8 (بدون علامت)
0123456uL	مبنای 8 بدون علامت و طولانی)
0563214L	مبنای 8 (طولانی)
0777777u	مبنای 8 (بدون علامت)
0x12545678L	مبنای 16 (طولانی)
0xABCDEFu	مبنای 16 (بدون علامت)
0xEEF123AbuL	مبنای 16 (بدون علامت و طولانی)

مثال – برنامه زیر عددی در مبنای ۱۶ (با پیشوند 0x یا بدون آن) را از طریق ترمینال می‌خواند و معادل آن را در مبناهای ده و هشت چاپ می‌کند :

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

- **داده‌های اعشاری (floating-point type)**

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

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

0.356 , 5.0 , 3.14 , 7. , .75

روش دوم که نمایش علمی یا scientitic notation نیز نامیده می‌شود، یک روش کوتاه‌نویسی مفید می‌باشد. 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 بکار برده می‌شود. برای مثال :

3E2 به مفهوم $3*10^{2}$ و 125.7E-3- به مفهوم $-125.7*10^{-3}$ می‌باشد. 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 می‌باشد . مانند مثالهای زیر:

0. , 0.0 , 1. , 0.2 0.125E-3 , 2E-5 , -3.14 , 825.25 ,

البته قسمت نما نمی‌تواند یک عدد کسری باشد ، پس 34.5E3.5 درست نیست. در بعضی نسخه های C برای اینکه مشخص کنند که مقادیر مورد نظر یک کلمه اشغال کرده است حرف F را به آخر آن اضافه می‌کنند ، مانند نمونه زیر :

3.25E5F

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

به هرحال دقت مقادیر ثابت با ممیز شناور ممکن است برحسب نسخه‌های مختلف تغییر کند، ولی همه آنها حداقل ۶ رقم با معنی را می‌پذیرد؛ در برخی تا ۱۸ رقم با معنی نیز امکان‌پذیر است.

برای اعلان متغیرهایی از نوع floating point از دو کلمه 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 روی 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 ٤ 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 نمی‌باشد. 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 بصورت long float تعریف کرد . بنابراین دو روش اعلان زیر معادل می‌باشند:

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

به هرحال اگر کلمه توصیف‌کننده long به تنهایی جلوی متغیری بکار برده شود، آن متغیر بصورت پیش‌فرض از نوع مقادیر صحیح خواهد بود . بنابراین با دستور زیر :

```
long  a , b , c ;
```

سه متغیر c , b , a از نوع صحیح خواهند بود .

باتوجه به کلمات کلیدی مندرج در جدول زیر برای اعلان متغیرها و با درنظرگرفتن توضیحات و مثالهای مذکور در این فصل، می‌توان نوع داده‌های اصلی را که تا اینجا مورد بحث قرار گرفت، با درنظرگرفتن ترکیب آنها با توصیف‌کننده‌های دیگر به اختصار بصورت جدول زیر نشان داد.

خلاصه جدول انواع داده‌ها

نوع	اندازه به بیت	محدوده یا بازه 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 $(۱۰^{+۳۷}$

- **مقداردهی اولیه متغیرها**

در صورتی که از پیش مقدار شروع متغیر را بدانیم می‌توانیم به هنگام تعریف، مقدار اولیه 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 بکار می‌بریم. بعنوان مثال هر یک از روشهای زیر متغیرهای :

```
a , b , c
```

را توصیف کرده و به ترتیب مقادیر 25- , 13 , 12 را به آنها اختصاص می‌دهد.

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 که مقدار اولیه می‌پذیرند، جدا از سایر متغیرها توصیف گردند، مانند مثال زیر:

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

مثال - چند نمونه دیگر از مقداردهی اولیه متغیرها در زیر نشان داده شده است :

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

- **ثابتهای رشته‌ای**

یک ثابت رشته‌ای ، شامل دنباله‌ای از کاراکترها می‌باشد که در داخل گیومه دوبل قرار داده می‌شوند. مانند نمونه های زیر:

five$" , "p4"" , "1380-02-06" , hello world" " , "256" , "university"

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

مثال – ثابت رشته‌ای زیر شامل سه نشانه مخصوص (special character) است که با escape sequence متناظرشان نشان داده شده‌اند:

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

که در آن نشانه‌ها یا کاراکترهای مخصوص عبارتند از :

\t ، horizontal tab

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

\n خط جدید یا new line .

کامپایلر بطور اتوماتیک 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 ، قابل رؤیت نمی‌باشد . به هرحال می‌توان هریک از کاراکترها را در داخل رشته امتحان کرد که آیا کاراکتر null می‌باشد یا نه ؟ در خیلی موارد مشخص ساختن پایان یک رشته بوسیلهٔ یک کاراکتر مخصوص مانند کاراکتر null نیاز به تعیین حداکثر طول برای رشته را از بین می‌برد. بعنوان

مثال رشته فوق دارای ۳۸ کاراکتر است که شامل پنج فضای خالی و چهار کاراکتر مخصوص است که با escape sequence معرفی شده‌اند و در پایان کاراکتر null می‌باشد که انتهای رشته را مشخص می‌سازد .

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

باز هم توجه داشته باشید که یک رشته n کاراکتری نیاز به آرایه n+۱ عنصری خواهد داشت. زیرا یک کاراکتر null نیز بطور اتوماتیک بعنوان کاراکتر پایانی در آن قرار داده خواهد شد. برای مثال اگر رشته:

"COMPUTER"

در یک آرایه یک بعدی کاراکتری به نام book ذخیره گردد، خانه اول آن یعنی :

book[0]

شامل کاراکتر C و خانه آخر یعنی :

book[8]

شامل کاراکتر null خواهد بود که معرف پایان رشته است .

مبحث رشته‌ها و آرایه‌ها و کاربرد آنها در فصل جداگانه‌ای بطور مشروح مورد بحث قرار خواهد گرفت.

- **اپراتور cast**

می‌توان تبدیل یک نوع به نوع دیگر را به‌صورت صریح نیز انجام داد. این کار به کمک اپراتور 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 یک اپراتور unary می‌باشد؛ 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 می‌شود و به h نسبت داده می‌شود. حال اگر بخواهیم مقدار 1.5 که نتیجه واقعی عبارت ریاضی 3/2 است به k و i یا هر دوی آنها را به‌وسیلهٔ 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 بصورت :

(float) k / (float) i ; یا k / (float) i ;

نیز نوشت که نتیجه باز هم 1.5 می‌گردد . یعنی نتیجه سه روش مزبور هم‌ارز می‌باشد.

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 به این طریق است که type 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 نتیجه d1+d2 به int تبدیل می‌شود.

بنابراین فرمت اپراتور cast بصورت زیر است :

(data type) expression

حال برای آنکه نقش اپراتور cast را بهتر متوجه شوید، به نتیجه و عملکرد دو مجموعه دستورات 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، 4 بایت حافظه پیش‌بینی می‌شود. ولی درنتیجه 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 :

0.25 E+7

باشد، مقدار 2500000 نمایش داده خواهد شد.

همین‌طور در مثال دوم برای متغیر x که از نوع short int اعلان شده است، دو بایت حافظه پیش‌بینی می‌شود، ولی درنتیجه اجرای دستور printf سطر دوم، به دلیل (char)x مقدار آن به نوع کاراکتر تبدیل می‌گردد و محتوای دو بایت حافظه مزبور به‌صورت یک رشته دوبایتی نمایش داده می‌شود و به همین دلیل است که در فرمت چاپ مقدار متغیر مزبور، از فرمت "%s" که برای رشته می‌باشد، استفاده شده است. حال اگر برای مثال محتوای حافظه مربوط به متغیر x به‌صورت :

1 2 3 4

باشد، موقع نوشتن به‌صورت رشته :

"1 2 3 4"

چاپ می‌شود .

در اینجا به اختصار یادآوری می‌شود که فرمتهای :

"%s", "%c", "%f", "%d"

بترتیب برای متغیرهای از نوع مقادیر صحیح ، اعشار، کاراکتر و رشته بکار برده می‌شود.

- **داده ها از نوع void**

داده از نوع void (تهی) در استاندارد ریچی وجود نداشت و بعد به استاندارد ANSI افزوده شده است. 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.

مثال – تابعی به نام FF1 که دارای دو آرگون x , y است بصورت زیر تعریف شده است :

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

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

- **پیش پردازنده**

پیش‌پردازنده را می‌توان برنامه جداگانه‌ای در نظر گرفت که قبل از کامپایلر واقعی اجرا می‌گردد. هنگامی که شما یک برنامه را کامپایل می‌کنید، پیش‌پردازنده بطور اتوماتیک 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 ما را آماده ساخته و وارد برنامه ما کند. برخلاف دستورات C که به سمی‌کولون ختم می‌شوند، پایان جملات آن با خط جدید مشخص می‌گردد .

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 بکار ببرند، مفید است. با این کار شما به

جای اینکه اطلاعات مشترک دو فایل را بصورت دوبله در هر دو منظور نمایید، آن را فقط در یک فایل قرار می‌دهید ، سپس هر موقع آن اطلاعات بوسیلهٔ فایل دوم مورد نیاز باشد ، به طریق مذکور آن را برای استفاده فایل دوم نیز آماده می‌کنید.

تعریف فرمان #include دارای دو فرم زیر است :

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

در فرم اول ، پیش‌پردازنده فقط محل خاصی را که به‌وسیلهٔ عامل مشخص شده است، نگاه می‌کند. این محل جایی است که include file های سیستم، مانند header files برای کتابخانه زمان اجرا یا Runtime Library نگهداری می‌شوند. اگر فرم دوم بکار برده شود، پیش‌پردازنده فهرست یا دایرکتوری را که شامل فایل مبنا است، نگاه می‌کند. اگر فایل include file را در آنجا پیدا نکند، پس از آن مشابه فرم اول، محل خاص مورد نظر را جستجو می‌کند. اسامی include file ها بر حسب قرارداد، به پسوند "h" ختم می‌شوند .

حال ببینیم وقتی که پیش‌پردازنده با فرمان #include<stdio.h> مواجه می‌شود، چه پیش می‌آید؟ پیش‌پردازنده دایرکتوری تعریف شده به‌وسیلهٔ 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 شما 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 بصورت زیر بکار می‌برید :

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

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

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

فرمان #define

همانطور که می‌توان با توصیف یا اعلان کردن متغیر، اسمی را به یک محل از حافظه وابسته کرد و به آن محل با آن نام (که همان متغیر مورد نظر است) مراجعه نمود، به همان طریق می‌توان اسمی را به یک مقدار ثابت وابسته کرد و آن را با همان اسم که ثابت سمبولیکی نامیده می‌شود، مشخص نمود و هنگام نیاز، به آن مراجعه کرد. این گونه متغیرها را که معمولاً با حروف برزگ 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 بجای 15 از book استفاده نمود . بنابراین دو دستور زیر هم‌ارز می‌باشد :

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

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

براساس دستور define مقدار book در حافظه بصورت مقدار ثابت 15 است که در طول برنامه تغییر نمی‌کند. انتخاب نام برای مقادیر ثابت دارای چند فایده مهم است :

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 بصورت :

```
#define  Pi  3.1415
```

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

دوم آنکه اگر مجبور باشیم در یک برنامه یک مقدار ثابت طولانی نامانوس را چندین بار بکار بریم، ساده‌تر آن خواهد بود که با دستور #define یک نام مناسب برای آن انتخاب کنیم و بجای ثابت مزبور از آن نام استفاده کنیم .

مثلاً اگر مجبور باشیم همان عدد پی را با 6 رقم اعشار در قسمتهای متعددی از برنامه بکار بریم، این عمل هم پردردسر و هم اشتباه‌زا خواهد بود. لذا بهتر است مثل حالت قبل، یک اسم برای آن انتخاب کنیم .

سوم آنکه اگر طبیعت مقدار ثابت طوری باشد که در زمانهای مختلف تغییر کند. مثل نرخ مالیات، یا اجرت ساعت کار و یا درصدی که بعنوان 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 توسط 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 دو درصد باشد ، کافی است به آن قبلاً نام TAX اختصاص داده باشیم و با دستور :

```
#define  TAX  0.02
```

مقدار جدید را جایگزین قبلی نماییم 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، دنباله کاراکترهای متناظر آن جایگزین می‌گردد. ثابتهای سمبولیکی معمولاً در آغاز برنامه تعریف می‌گردند و فرم آن بصورت زیر است :

```
#define  name  text
```

که در آن، name معرف نام سمبولیک است که معمولاً با حروف بزرگ نوشته می‌شوند؛ 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، یک دستور واقعی C نمی‌باشد.

مثال – در زیر، نمونه‌های دیگری از ثابتهای سمبولیکی نشان داده شده است :

```
#define     Temp        524
#define     Pi          3.1415
#define     true        1
#define     false       0
#define     Name        " payam noor "
#define     Byte_Size   8
```

خصیصه #define که برای تعریف ثابتهای سمبولیک بکار برده می‌شود، یکی از چندین خصیصه‌ای است که در پیش‌پردازنده وجود دارد .

- **تمرین و پاسخ**

تمرین ۱ – برنامه‌ای بنویسید که مساحت دایره ای به شعاع R را محاسبه کرده نمایش دهد .

```
#include <stdio.h>
#define  p  3.1415
main( )
{
  float  R , s  ;
  scanf("%f ", &R );
  s = p * R * R ;
  printf("%f ", s) ;
}
```

توضیح – در برنامه بالا p از ثابتهای سمبولیک بوده و مقدار آن با دستور define معرفی شده اسـت. شعاع دایره و نیز مساحت از نوع اعشاری تعریف شده . سپس شعاع دایره از ورودی خوانـده شـده و دو بار در عدد p ضرب میشود . در نهایت مساحت s بشکل اعشاری چاپ میگردد .

تمرین ۲ ـ برنامه‌ای بنویسید که مجموع ۱۰۰ جمله اول سری زیر را محاسبه و چاپ کند .

y = 1 + 1/2 + 1/3 + 1/4 + ...

```
#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 برنامه بالا متغیر k شمارنده بوده و مقادیر صحیح مثبت کوچکتر از ۱۰۰ را شامل میشود . لذا از نوع بدون علامت یا unsigned تعریف شده است . (استفاده از نوع int ونیز short هم مجاز است .)

تمرین ۳ ـ برنامه‌ای بنویسید که ضرایب معادلات درجه دوم را بخواند جوابهای آن را به‌دست آورده و چاپ کند . اگر معادله ریشه حقیقی نداشته باشد پیغام مناسب نمایش دهد .

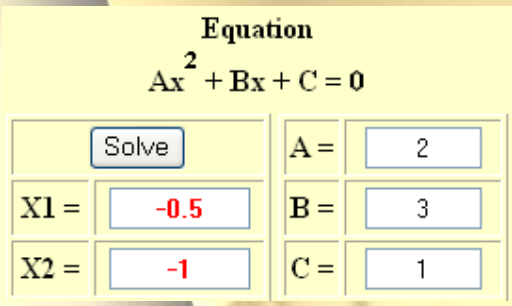
حل : برنامه مورد نظر در زیر نمایش داده شده است .

```
#include<math.h>
#include<stdio.h>
main( )
  {
    float  a , b , c , x1 , x2 , delta , d ;
    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 برنامه بالا sqrt از توابع از پیش تعریف شده ریاضی در زبان C است که جذر آرگومان خود را (که در اینجا متغیر delta است) برمی‌گرداند. این تابع جزءِ کتابخانه "math.h" می‌باشد. لذا در خط اول با دستور #include قرار گرفته است .

نمونه کاربردی این برنامه را در زیر ملاحظه می کنید :



تمرین٤ ـ خروجی برنامهٔ زیرچیست ؟

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

حل : برنامه صحیح بوده و پیغام خطایی صادر نمی‌گردد . خروجی برنامه عدد 244 می‌شود . ابتدا باینری معادل 12- در متغیر یک بایتی ch ریخته میشود:

$(12)\ 10 = (00001100)_2$

برای بدست آوردن 12- از باینری 12+ متمم ۲ می‌گیریم :

$(-12) = (11110100)_2$

چون ch یک متغیر بدون علامت است پس رشته 1110100 به‌عنوان یک عدد بدون علامت فرض می‌شود :

$(11110100) = (128 + 64 + 32 + 16 + 4) = 244$

در دستور printf نیز محتوای متغیر ch به‌صورت یک عدد صحیح(به دلیل %d) چاپ می‌شود

تمرین۵ ـ خروجی برنامهٔ زیرچیست ؟

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

حل : خروجی برنامه عدد 144 می‌شود .

مقدار 100*4 برابر 400 می‌شود که حاصل در یک بایت متغیر C جا نمی‌گیرد و حداقل دو بایت جا نیاز دارد. معادل عدد 400 در دو بایت به شکل زیر است :

1001 0000	0000 0001

چون متغیر C تنها یک بایت است فقط بایت سمت راست 400 یعنی 1001 0000 ذخیره می‌گردد و داریم :

1001 0000 بدون علامت 144 = 128 + 16 ←

فصل سوم – دستورات کنترلی

- **مقدمه**

یکی از امکانات زبانهای برنامه‌سازی مدرن، استفاده از دستورات و ساختارهای کنترلی است و درنتیجه این امکان را فراهم می سازند که یک قطعه از برنامه چندین بار تا موقعی که شرط ویژه‌ای برقرار است، اجرا گردد.

زبان برنامه نویسی C ، دارای قلمرو گسترده ای از این نوع ساختارها است. در حالت عادی،

دستورهای هر برنامه بطور متوالی اجرا می‌گردد . اما اگر منطق برنامه ایجاب کند که یک دستور یا مجموعه‌ای از دستورها در صورت وجود، یا عدم وجود شرط یا شرایط خاصی اجرا گردند، باید شیوه دیگری اتخاذ کرد. دستورها یا ساختارهای کنترلی دستورهایی هستند که چنین زمینه‌ای را در برنامه‌نویسی فراهم می‌کنند.

مهمترین دستورهای کنترلی شامل دستورهای :

do-while , while , for

بعنوان ساختارهای حلقه‌های تکرار و دستورهای :

switch , if

بعنوان دستورهای شرطی یا ساختارهای تصمیم‌گیری و بالاخره دستورهای :

goto , continue , break

می‌باشد . تابع exit را که آن هم در کنترل جریان یک برنامه نقش دارد، در اینجا مورد بحث قرار می‌دهیم .

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

در این فصل، مهمترین دستورها و ساختارهای کنترلی زبان C مورد بررسی قرار می‌گیرد.

- **دستور while**

این دستور یکی از دستورهای کنترلی زبان C است که بوسیلهٔ آن یک حلقه تا موقعی که شرط معینی برقرار باشد، اجرا می‌گردد. فرم کلی این دستور بصورت زیر است :

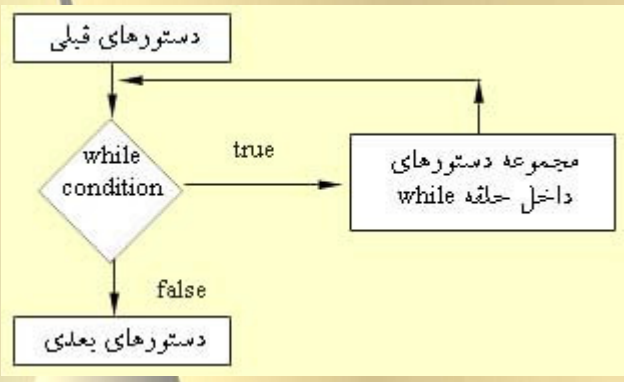
```
while (condition)
    statement ;
```

در فرم بالا پس از عبارت while فقط یک دستور بکار رفته است . اما می‌توان مجموعه‌ای از دستورها نیز بکار برد. از طرفی بطوری که در فصول قبل نیز بیان شده است، در زبان C ، هر دستور به یک علامت سمی‌کولون ختم می‌شود و مجموعه‌ای از دستورها (یعنی بیش از یک دستور) را نیز دستورهای مرکب و یا block نامند که اصطلاح بلاک بیشتر متداول است و در زبان C برای مشخص ساختن آن، مجموعه دستورهای مورد نظر در داخل یک زوج آکولاد قرار می‌گیرد. بنابراین، در چنین حالتی فرم دستور while بصورت زیر خواهد بود:

```
while (condition)
  {
```

```
    statements ;
}
```

همچنین نمودار کلی دستور while را می‌توان بصورت شکل زیر نمایش داد:



مکانیسم و نحوه عملکرد دستور در while به این طریق است که تا موقعی که شرط مورد نظر که پس از کلمه کلیدی while در داخل پرانتزها نوشته شده، برقرار باشد، مجموعه دستورهای داخل حلقه while بصورت تکراری اجرا خواهد شد. شرط مورد نظر می‌تواند با استفاده از اپراتورهای رابطه‌ای بصورت عبارات رابطه‌ای بیان گردد و یا می‌تواند بصورت یک عبارت منطقی باشد که در این صورت تا موقعی که عبارت مزبور ارزش درست یا true داشته باشد حلقه مورد نظر اجرا خواهد شد.

درضمن بطوری که در گذشته نیز بیان شد، در زبان C ارزش نادرست یا false متناظر با صفر و ارزش درست یا true متناظر با مقادیر غیرصفر (مثبت یا منفی) می‌باشد.

مثالهای زیر، کاربرد و نحوهٔ استفاده از ساختار کنترلی while را مشخص می‌سازد.

مثال – برنامه‌ای بنویسید که اعداد صحیح صفر تا ۱۰۰ را در روی خطوط متوالی چاپ کند .

حل : این برنامه می‌تواند به دو روش زیر باشد :

روش دوم

```
#include<stdio.h>
main( )
  {
    int  number=0;
    while (number<=100)
      printf ("%d\n", number
  ++);
   }
```

روش اول

```
#include<stdio.h>
main( )
 {
  int  number=0;
  while (number<=100)
   {  printf("%d\n",
   number);
     ++ number;
   }
 }
```

مثال – برنامه‌ای بنویسید که عدد صحیح غیرمنفی n را بخواند، فاکتوریل آن را حساب کند و با خود عدد چاپ نماید.

حل : برنامه مورد نیاز می‌تواند بصورت زیر باشد :

```
#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 ) ;
 }
```

مثال – برنامه زیر یک خط از متنی با حروف کوچک را کاراکتر به کاراکتر به آرایه letter می‌خواند . سپس با استفاده از تابع کتابخانه‌ای toupper متن مزبور را به حروف بزرگ تبدیل نموده و چاپ می‌کند :

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

در برنامه بالا دو عبارت while جداگانه بکار برده شده که یکی برای خواندن متن به حافظه و دیگری برای تبدیل حرف کوچک به بزرگ و چاپ متن تبدیل شده می‌باشد. برای مثال اگر متن زیر:

the book is on the table

بعنوان ورودی وارد گردد، عبارت :

THE BOOK IS ON THE TABLE

نمایش داده خواهد شد. یادآوری می‌شود که در برنامه مزبور حلقه را می‌توان بصورت مجموعه عبارات زیر نیز نوشت :

```
letter [count] = getchar( ) ;
while ( letter[count]!eol )
 {
   count = count + 1 ;
   letter[count]=getchar( ) ;
 }
```

بدیهی است خوانایی و درک منطق بکار رفته در حالت اخیر ساده‌تر است.

مثال – برنامه‌ای بنویسید که نمرات امتحانی n نفر دانشجو را خوانده، معدل کلاس، نمره امتحانی شاگرد اول و شاگرد آخر کلاس را تعیین کرده، چاپ کند. n از روی اولین دستور ورودی خوانده شود.

حل : برنامه مورد نیاز در زیر نشان داده شده است :

```
#include<stdio.h>
main ( )
 {
   int   n , count =1 ;
   float  x , av , 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 ;
     }
                  /*calculate the average*/
    av = sum / n ;
    printf ("\n average= %f , max= %f , min= %f ", av , max , min) ;
  }
```

- **دستور do-while**

بطوری که ملاحظه شد، در دستور while آزمایش شرط برای ادامه حلقه، در آغاز هر تکرار حلقه انجام می‌گیرد. گاهی مطلوب است که این آزمایش در پایان حلقه صورت پذیرد. این کار با دستور کنترلی do-while امکان‌پذیر است. فرم کلی دستور do-while بصورت زیر است :

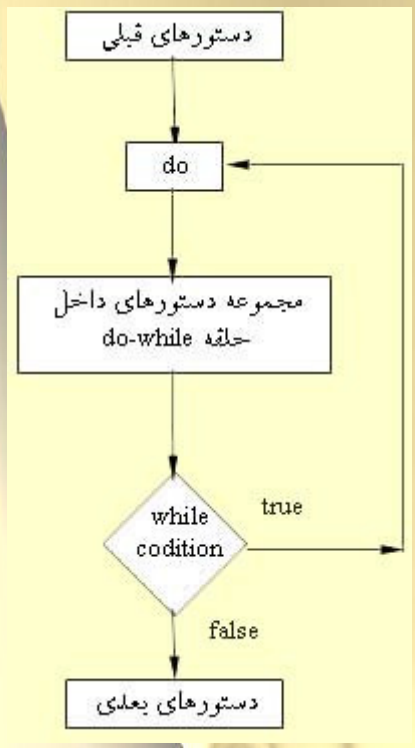
```
do
 {
    statements
 }while (condition) ;
```

بدیهی است اگر حلقه 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 مورد آزمایش قرار می‌گیرد. 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 نتیجه آن مشابه while می‌باشد.

برای اغلب کاربردها، آزمایش شرط ادامه برای اجرای حلقه، بطور طبیعی در آغاز حلقه صورت

می‌گیرد. بدین لحاظ دستور do-while در مقایسه با دستور while کاربرد کمتری دارد. چارت یا نحوه عملکرد این دستور در شکل زیر نشان داده شده است:



حال برای نشان دادن مکانیسم استفاده از این ساختار، مثالهایی در مبحث do-while حل می‌گردد.

مثال – برنامه‌ای بنویسید که اعداد صحیح صفر تا ۱۰۰ را در روی خطوط متوالی چاپ کند .

حل : این برنامه با استفاده از do-while می‌تواند بصورت زیر باشد :

```
#include<stdio.h>
main ( )
  {
    int  number = 0 ;
    do
      printf ("%d\n", number + +) ;
    while ( number<=100 ) ;
  }
```

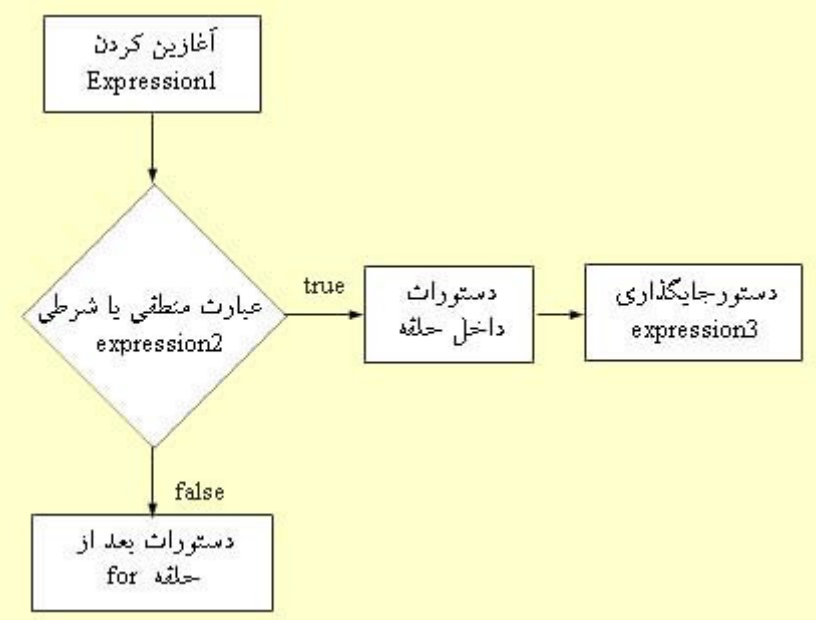
مثال – برنامه‌ای بنویسید که عدد صحیح غیرمنفی n را خوانده ، فاکتوریل آن را حساب کند و با خود عدد چاپ نماید.

حل : برنامه فوق با استفاده از do-while می‌تواند بصورت زیر باشد :

```
#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 که خیلی شبیه به دستور while می‌باشد، یکی دیگر از دستورهای کنترلی است که زیاد بکار برده می‌شود و چارت آن در زیر نشان داده شده است :



فرم کلی این دستور به‌صورت زیر می‌باشد :

for (expression1; expression2; expression3) statement ;

که در آن expression1 برای آغازین کردن پارامتری که حلقه را کنترل می‌نماید (و شاخص یا index نامیده می‌شود) بکار برده می‌شود؛ expression2 یک شرط را معرفی می‌نماید که باید برای ادامه اجرای حلقه صادق باشد و expression3 نیز برای تغییر مقدار پارامتری که در آغاز به expression1 اختصاص داده شده بکار برده می‌شود.

معمولاً expression1 یک عبارت جایگذاری، expression2 یک عبارت منطقی یا رابطه‌ای و expression3 یک unary expression و یا یک عبارت جایگذاری می‌باشد .

وقتی که دستور for اجرا می‌گردد، قبل از هر گذر در داخل حلقه ، expression2 ارزیابی و

آزمایش می‌گردد. اما expression3 در پایان هر گذر ارزیابی می‌گردد. بنابراین می‌توان گفت که دستور for هم‌ارز دستورهای زیر است :

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

اجرای حلقه به‌صورت تکراری تا هنگامی که expression2 غیرصفر باشد، یعنی تا زمانی که شرط معرفی شده با expression2 برقرار و یا true باشد، اجرا خواهد گردید .

مثال – برنامه‌ای بنویسید که اعداد صحیح صفر تا ۱۰۰ را در روی خطوط متوالی چاپ کند .

حل : این برنامه با استفاده از حلقه for می‌تواند به‌صورت زیر باشد :

```
#include<stdio.h>
main ( )
 {
   int  number ;
   for ( number=0 ; number<=100 ; + + number )
     printf ("%d\n", number) ;
  }
```

ملاحظه می‌کنید که اولین خط به 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 ۱۰۰ تجاوز نکرده باشد، ادامه خواهد یافت. سومین expression پس از هر بار گذر حلقه، مقدار number را یک واحد افزایش می‌دهد.

از دیدگاه نحوی نیاز نیست هر سه expression مربوط به دستور for بکار برده شود، اگرچه بکار بردن سمی‌کولون مربوط به هر expression الزامی می‌باشد.

به هرحال نتیجه حذف هر expression باید بطور کامل فهمیده شود. اگر آغازین کردن و یا تغییر شاخص و 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 بصورت نامتناهی ادامه خواهد یافت مگر اینکه خروج از حلقه به طریق دیگری مثلاً بوسیلهٔ یک دستور break یا return پیش‌بینی گردد. از دیدگاه عملی در اغلب کاربردهای دستور for، هر سه expression منظور می‌گردد .

مثال – برنامه‌ای بنویسید که عدد صحیح غیرمنفی n را بخواند، فاکتوریل آن را حساب کند و با خود عدد چاپ نماید.

حل : برنامه فوق با استفاده از حلقه for می‌تواند بصورت زیر باشد :

```
#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) ;
 }
```

مثال – برنامه‌ای بنویسید که نمرات امتحانی دانشجویان کلاسهای مختلف را خوانده، معدل هر کلاس را حساب کرده، چاپ نماید. تعداد کلاسها و همچنین تعداد دانشجویان هر کلاس مشخص نیست و درنتیجه باید از طریق دستگاه ورودی داده شود.

حل : برنامه مطلوب برای انجام این کار در زیر نشان داده شده است :

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

حلقه‌ها می‌توانند به‌صورت تودرتو(nested loops) بکار برده شوند که در این حالت حلقه داخلی باید کاملاً توسط حلقه بیرونی احاطه گردد. بعبارت دیگر دو حلقه نباید یکدیگر را قطع نمایند. همچنین هر حلقه باید بوسیلهٔ شاخص جداگانه‌ای کنترل گردد . برنامه فوق از دو حلقه تشکیل شده است. در آغاز تعداد کلاسها (loops) خوانده شده و بر اساس آن حلقه بیرونی تکرار می‌گردد یعنی به ازای هر کلاس یک بار اجرا می‌شود. هر بار در داخل حلقه بیرونی، متغیر sum مساوی صفر قرار داده شده،

سپس تعداد دانشجویان یا تعداد نمره‌های امتحانی n خوانده می‌شود. پس از آن حلقه درونی، به تعداد دانشجویان کلاس مورد نظر n اجرا می‌شود که در هر تکرار نمره امتحانی یک نفر دانشجو خوانده شده و به sum افزوده می‌گردد. هر بار پس از کامل شدن حلقه درونی، جمع نمره‌های امتحانی یک کلاس بدست می‌آید که سپس معدل کلاس با نام average حساب گردیده، چاپ می‌شود.

- **دستور if و if-else**

دستورهای شرطی (conditional statements) (که می‌توان آنها را ساختارهای تصمیم نیز نامید) دستوراتی هستند که موجب می‌گردند تا در صورت وجود شرط یا شرایطی، مجموعه‌ای از دستورها (و یا یک دستور) اجرا گردند و یا در صورت وجود شرط یا شرایطی، یک مجموعه از دستورها و در صورت عدم وجود آن، مجموعه دیگری از دستورها اجرا شوند.

دستور if به صورتهای if و if-else بکار برده می‌شود که هر دو فرم و کاربرد آنها با ارائه مثالهای متعدد، بررسی می‌شود.

فرم کلی دستور if به‌صورت زیر است :

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

و درصورتی که مجموعه‌ای از دستورها مورد نظر باشد، فرم دستور if بصورت زیر خواهد بود:

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

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

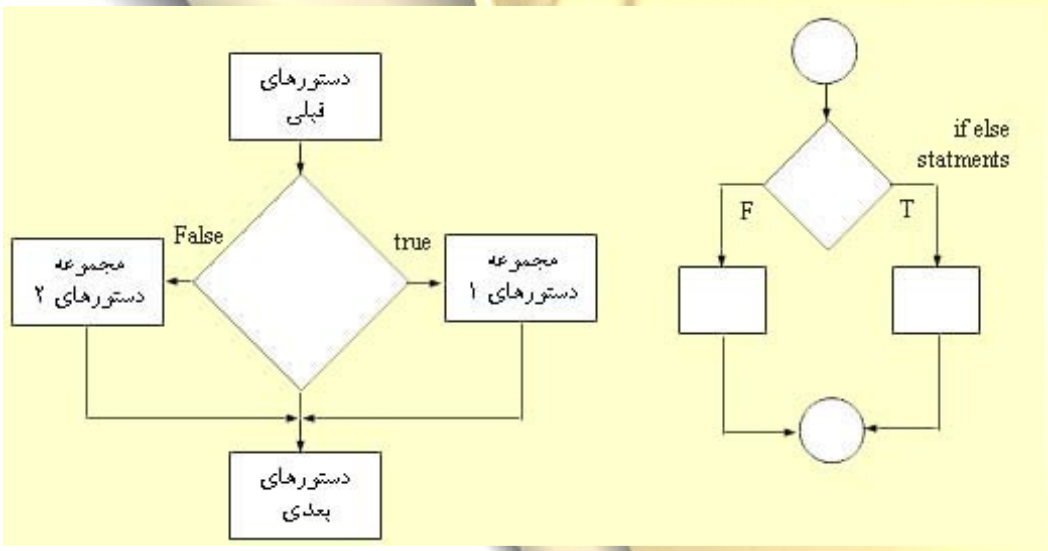
ساختار دستور if-else نیز به‌صورت زیر می‌باشد:

```
if (condition)
 {
  statements1;
 }
 else{
       statements2 ;
```

```
}
next statement ;
```

در این حالت نیز condition مشابه قبل، یک عبارت شرطی و یا یک عبارت منطقی است که اگر شرط مورد نظر برقرار باشد و یا نتیجه 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.

چارت هر دو ساختار، در زیر نشان داده شده است :



اگر دستورهای بعد از if یا else بیش از یک دستور نباشد، نیازی به بکار بردن زوج آکولاد نخواهد بود.

ساختار if را می‌توان بصورت تودرتو (nested if-else) نیز بکار برد که در این صورت، هم if و هم else می‌تواند بصورت تودرتو تا هرچند سطح که منطق برنامه ایجاب نماید، تکرار گردند. مثالهای زیر این حالت را نشان می‌دهند.

مثال – برنامه‌ای بنویسید که متنی را خوانده ، تعداد محلهای خالی ، ارقام ، حروف ، پایان خط و جمع سایر نشانه‌های موجود در آن متن را شمرده ، چاپ نماید.

حل : برنامه مورد نظر به شکل زیر است :

```
#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) ;
  }
```

مثال – برنامه‌ای بنویسید که اسکی کد یک کاراکتر 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 اسکی کد آنها کوچکتر از ۳۲ است و قابل چاپ نیستند و ۳۲ که آسکی کد کاراکتر فضای خالی یا ‘ ’ است، پس آسکی کد کنترل کاراکترها از آسکی کد ‘ ’ کوچکتر است)، حروف بزرگ، حروف کوچک و یا سایر کاراکترها هستند .

حل : برنامه مورد نظر به شکل زیر است :

```
#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) ;
  }
```

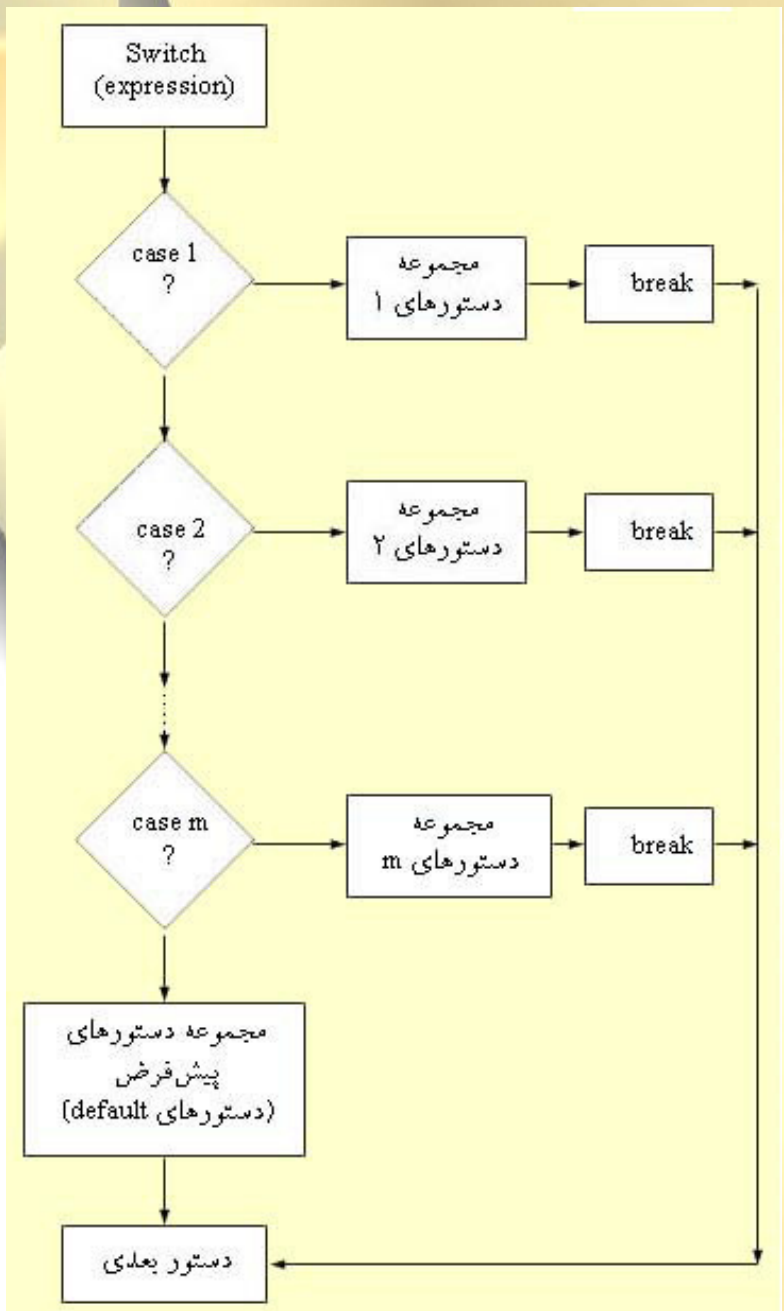
- **دستور switch**

این ساختار یا دستور موجب می‌گردد که یک گروه 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. بطوری که قبلاً ملاحظه کردید، در دستور if-else بر اثر وجود یا عدم وجود شرط (یا شرایطی)، یکی از دو مجموعه دستورها انتخاب و اجرا می‌گردد. بعبارت دیگر در اینجا حداکثر دو انشعاب و یا دو شاخه وجود دارد که باید فقط بر اثر نتیجه تست شرط مورد نظر، یکی از آن دو شاخه یا مسیر انتخاب گردد. دستور سویچ این امکان را فراهم می‌سازد که از بین شاخه‌های متعدد (بیش از دو شاخه) فقط یکی انتخاب گردد که قانون انتخاب را نتیجه ارزیابی عبارت ذکر شده در switch تعیین می‌کند. بنابراین، این دستور، اجازه چندین انشعاب را می‌دهد.

فرم کلی ساختار switch و همچنین چارت آن در زیر نشان داده شده است :

```
switch (عبارت)
{
case 1 مقدار :
          مجموعه دستورهای 1
       break ;
case 2 مقدار :
          مجموعه دستورهای 2
       break ;
   ....
   ....
   ....
```

```
case m مقدار :
            مجموعه دستورهایm
            break ;
default :
        مجموعه دستورهای m+1
}
```



در دستور switch نتیجه عبارت یا expression که پس از کلمه کلیدی switch در داخل یک زوج پرانتز قرار دارد، یک عدد صحیح (مقدار صحیح) خواهد بود. expression می‌تواند یک کاراکتر نیز

باشد زیرا هر کاراکتر نیز مقدار صحیح هم‌ارزی دارد که آسکی‌کد آن کاراکتر است.

نحوه عمل ساختار switch بدین طریق است که :

عبارت ذکر شده در switch محاسبه و ارزیابی می‌شود؛ سپس نتیجه آن از بالا به پایین به ترتیب با مقدار ذکر شده در case (مقدار ۱ ، مقدار ۲ ، .. و مقدار m) مقایسه می‌گردد. نتیـجه 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 بعنوان پیش‌فرض، اجرا می‌گردد.

در دستور switch استفاده از حالت default دلخواه می‌باشد. بنابراین چنانچه حالت default بکار 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 : بطوری که در بالا بیان شد، عبارت ذکر شده در switch فقط مقادیر صحیح و یا کاراکتر را خواهد داشت. بنابراین برخلاف دستور if نمی‌توان در آن از عبارت رابطه‌ای و منطقی استفاده کرد؛ بعبارت 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 تعمیم‌یافته دستور if-else برای m انشعاب تصور کرد.

دستور switch اغلب برای پردازش فرامین صفحه کلید بکار می‌رود. مانند فرمان انتخاب گزینه‌ای از یک منو (فهرست).

مثال – برنامه‌ای بنویسید که مقداری برای متغیر کاراکتری c که معرف رنگ است، از طریق صفحه کلید دریافت کند. اگر مقدار دریافتی یکی از کاراکترهای w , g , b , r باشد به ترتیب کلمات :

white , green , blue , red

را چاپ می‌کند.

حل : برنامه مورد نظر به شکل زیر است :

```
#include<stdio.h>
main ( )
{
 char  c ;
 c = getchar( ) ;
 switch (c)
  {
    case 'r':    printf ("red") ;
                 break ;
    case 'b':    printf ("blue") ;
                 break ;
```

```
    case 'g':    printf ("green") ;
                 break ;
    case 'w':    printf ("white") ;
                 break ;
  }
}
```

مثال – فرض کنید که نوع (type) یک متغیر، در یک متغیر دیگر از نوع عدد صحیح بصورت کد ذخیره شده است، که مقادیر درست آن عبارتند از:

صفر بعنوان تعریف نشده یا نامشخص (undefined)، 1 بعنوان int ، 2 بعنوان float و 3 بعنوان double.

برنامه‌ای بنویسید که مقدار متغیر معرف نوع را از طریق ورودی دریافت نماید و برحسب مقدار دریافتی که بصورت کد می‌باشد، پیغام مناسب چاپ کند.

حل : برنامه مورد نظر به شکل زیر است.

```
#include<stdio.h>
main ( )
{
 int  type ;
 scanf("%d", &type) ;
 switch (type)
  {
    case 0 : printf("undefined variable \n") ;
             break ;
    case 1 : printf("an integer variable \n") ;
             break ;
    case 2 : printf("a floatin point variable \n") ;
             break ;
    case 3 : printf("a double precision variable \n") ;
             break ;
    default :  printf("illegal variable \n") ;
  }   \*end of switch *\
 }
```

بعضی مواقع برای چندین حالت یک نوع عمل انجام می‌گیرد. در این موارد می‌توانیم همان دستورها را با چندین ثابت case مشخص کنیم. برای مثال فرض کنید که می‌خواهیم تعداد حروف صدادار و بی‌صدا را در یک ورودی از نوع رشته، بشماریم. اگر کاراکتر جاری که خوانده می‌شود در متغیر ch ذخیره شده باشد، می‌توانیم دستورهای زیر را بکار ببریم:

```
switch (ch)
{
   case  'a':
   case  'e':
   case   'i':
   case  'o':
   case  'u':  vowels ++ ;
               break ;
```

```
    default :  nonvowels ++ ;
}
```

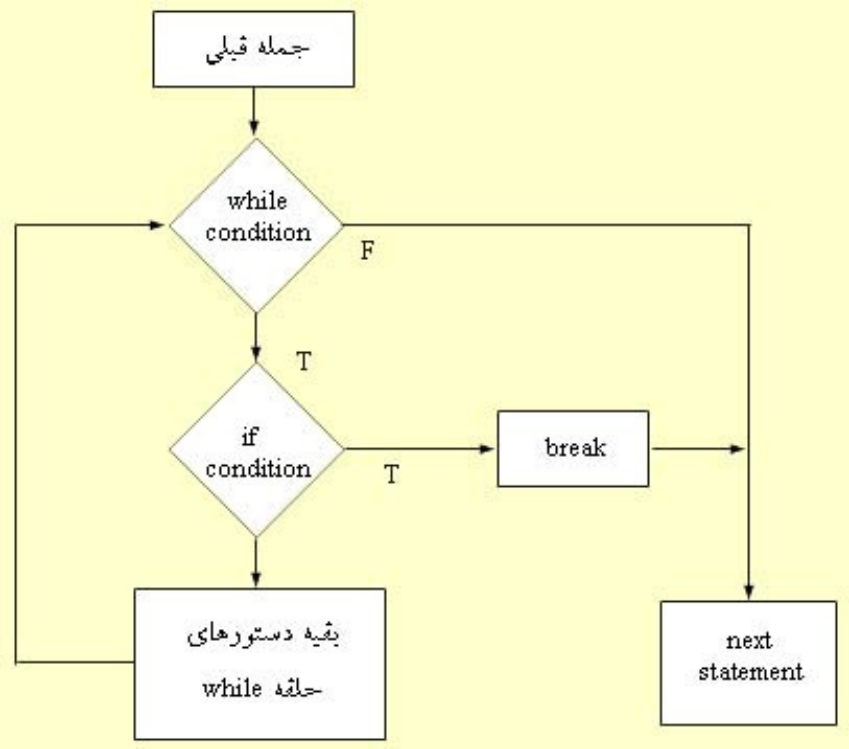
- **break و continue**

دو دستور خاص break و continue جریان طبیعی کنترل را قطع می‌نمایند. دستور break کنترل را به پایان ساختاری که break در داخل آن بکار برده شده است، انتقال می‌دهد. درواقع موجب خروج از حلقه و یا ساختار می‌گردد. در مثال زیر تستی برای منفی بودن آرگومان انجام می‌گیرد. اگر نتیجه true بود، یک دستور break بکار برده شده تا کنترل را به اولین جمله بلافاصله پس از حلقه انتقال دهد :

```
while(1)
    {    scanf ("%f", &x) ;
         if (x<0.0)
           break ;     /* exit loop if the value is negative*/
         printf ("\n %f ", sqrt(x)) ;
    }
```

فرم کلی و نیز نمودار دستور break بصورت زیر است :

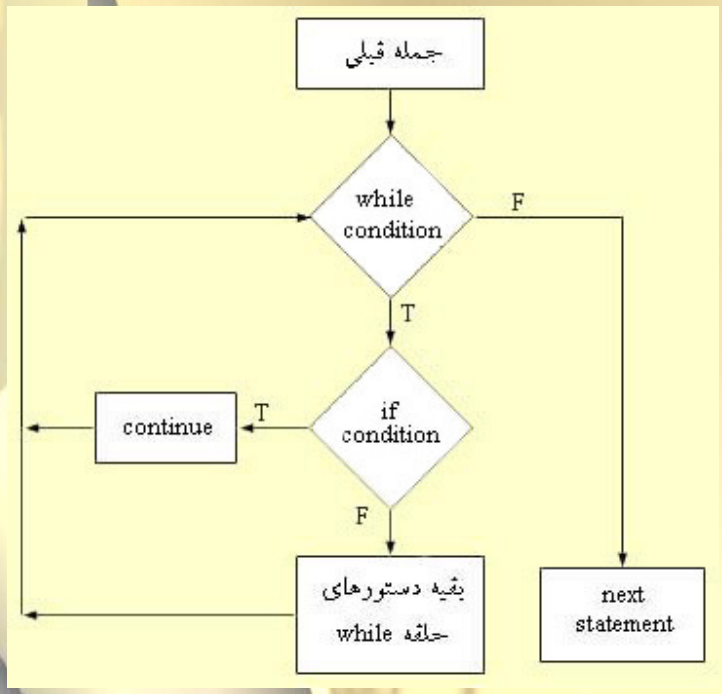
```
break ;
```



همچنین می‌توان دستور مزبور را به طریق مشابه برای خروج از حلقه‌های for و do-while نیز بکار برد. کاربرد دیگر break در جمله switch است که قبلاً ملاحظه کردید .

دستور continue موجب می‌گردد که باقیمانده تکرار جاری یک حلقه، نادیده گرفته شود و تکرار بعدی حلقه بلافاصله آغاز گردد . این دستور را می‌توان با ساختارهای for , do-while , while بکار برد و فرم کلی و نمودار آن بصورت ساده زیر است :

```
continue ;
```



مثال – برنامه‌ای بنویسید که تعداد n عدد را خوانده و میانگین اعداد غیرمنفی را حساب نماید (n از اولین دستور ورودی خوانده شود).

حل : برنامه مطلوب به شکل زیر نشان داده شده است :

```
#include<stdio.h>
main ( )
 {
   float  x , average , sum = 0 ;
   int  n , count , navg = 0 ;
        /* initizlize and read in a value for n*/
   printf("how many numbers? ") ;
   scanf("%d", &n) ;
        /* read in the number*/
   for (count=1 ; count<n ; + + count)
     {
       scanf ("%f", &x) ;
       if (x<0)
         continue ;
       sum+ = x ;
```

```
      ++ navg ;
    }
  average = sum / navg ;
  printf ("\n the average is %f \n", average) ;
}
```

- **اپراتور کاما**

اپراتور کاما در C پایین ترین تقدم را دارد. این اپراتور یک اپراتور باینری می‌باشد. اپراندهای آن عبارت یا expression هستند و فرم کلی آن بصورت زیر می‌باشد :

expression1, expression2

که در آن اول expression1 سپس expression2 محاسبه می گردد .

کاما بعضی مواقع در جمله for بکار برده می‌شود و اجازه آغازین کردن (مقدار اولیه دادن) چندگانه و پردازش چندین اندیس یکجا را به ما می‌دهد . برای مثال دستورهای :

```
for (sum=0 , i=1 ; i<=n ; ++i )
   sum += i ;
printf("\n %d", sum) ;
```

موجب خواهد شد که مجموع اعداد صحیح از ۱ تا n محاسبه و بدست آمده و چاپ شود. می‌توان این دید را بیشتر گسترش داد و عبارت فشرده‌تری بدست آورد . برای مثال دستورهای زیر همان نتایج بالا را بدست خواهد داد :

```
for (sum=0 , i=1 ; i<n ; sum+=i , ++i ) ;
    /* empty statement*/
printf ("\n %d", sum) ;
```

همچنین برای محاسبه مجموع اعداد زوج و فرد در فاصله ۱ تا n می‌توان نوشت :

```
even-sum = odd-sum = 0 ;
for ( cnt=0 , j=2 , k=1 ; cnt<n ; cnt+ =2 , j+=2 , k+ =2 )
   {
     even-sum += j ;
     odd-sum += k ;
    }
printf("%d %d", even-sum , odd-sum) ;
```

بنابراین اپراتور کاما اجازه می‌دهد در جایی که بطور متعارف یک expression ظاهر خواهد شد دو یا چند expression ظاهر گردد و بیشترین کاربرد آن در جمله for می‌باشد. برای مثال می‌توان جمله for زیر را که نمونه آن در مثال بالا مشاهده شد، نوشت :

```
for (expression1a , expression1b ; expression2 ; expression3)
  statement
```

که در آن expression1a و expression1b دو عبارتی هستند که به‌وسیلهٔ کاما از یکدیگر جدا شده‌اند. در این نوع کاربرد دو عبارت بطور جداگانه دو اندیس را آغازین خواهند کرد که بطور همزمان در داخل حلقه for بکار برده خواهند شد. ممکن است یک جمله for بطریق مشابه، اپراتور کاما را به روش زیر بکار برد :

for (expression1 ; expression2 ; expression3a , expression3b)

اپراتور کاما از چپ به راست عمل می‌کند (یعنی تقدم عملیات از چپ به راست می‌باشد). همچنِین یادآوری میشود اغلب کاماها در برنامه‌ها، معرف اپراتورهای کاما نمی‌باشد. مثلاً در لیست آرگومانهای تابع، کاماها آرگومانها را از یکدیگر تفکیک می‌نمایند و نیز وقتی که چندین متغیر با هم توصیف یا آغازین می‌گردند، کاماهای بکار برده شده برای تفکیک متغیرها می‌باشد و درنتیجه اپراتور کاما نیستند.

- **دستور goto**

دستور goto یک انتقال کنترل بدون شرط است که اصول برنامه‌سازی ساخت‌یافته را لغو می‌نماید و درنتیجه مطلوب نمی‌باشد. این دستور، توالی عادی عملیات را به هم می‌زند. فرم کلی آن به‌صورت زیر است :

```
goto  label ;
```

که در آن label شناسه‌ای است که محل انتقال و یا محل دستور بعدی را که باید اجرا گردد، نشان می‌دهد. به هرحال محل انتقال کنترل و یا جمله هدف مورد نظر باید دارای label باشد و فرم آن می‌تواند بصورت زیر باشد :

```
label : statment
```

بطوری که ملاحظه می‌گردد ، باید به دنبال label ، علامت کولون (:) بیاید مانند قطعه برنامه زیر :

```
{
---------
---------
goto  error ;
---------
---------
}
error :  printf("\n an error has occurred") ;
```

به هرحال گاهی پیش می‌آید که دستور goto ساده و مفید می‌باشد. برای مثال موقعیتی را درنظر بگیرید که ضرورت پیدا کند کنترل، تحت شرایطی از داخل دو حلقه تودرتو، به خارج انتقال داده شود. در این حالت اگر از goto استفاده نشود، یک راه آن خواهد بود که دوباره جمله if-break بکار برده شود که هرکدام برای خروج از یکی از حلقه‌ها خواهد بود. بدیهی است در چنین حالتی

استفاده از goto ساده‌تر خواهد بود .

مثال – برنامه‌ای بنویسید که عدد صحیح num را از طریق ورودی بخواند. سپس جذر آن را با استفاده از تابع sqrt بدست آورده، چاپ کند. چنانچه مقدار ورودی، عدد منفی باشد، پیغام مناسبی چاپ کند .

حل : برنامه مورد نظر که در آن از دستور goto استفاده شده در زیر نشان داده شده است .

```
#include<stdio.h>
#include<math.h>   /*for sqrt ( ) function*/
main ( )
  {
    int  num ;
    scanf("%d", &num) ;
    if (num<0)
      goto  bad-val ;
    else
      {
        printf("the square root of num is %f", sqrt (num)) ;
        goto  end ;
      }
    bad-val :  printf("error: negative value.\n") ;
    exit(1) ;
    end :  exit (0);
}
```

- **تابع exit**

این تابع که در کتابخانه استاندارد وجود دارد، موجب خاتمه پذیرفتن کل برنامه می‌گردد. به لحاظ اینکه تابع exit اجرای برنامه را بطور کامل قطع می‌کند و موجب برگشت به سیستم عامل می‌گردد، گاهی این تابع بعنوان یک دستور کنترل برنامه بکار برده می‌شود.

این تابع معمولاً با آرگومان صفر فراخوانی می‌شود که دلالت بر خاتمه پذیرفتن طبیعی و نرمال می‌کند. اما آرگومانهای دیگری نیز بعنوان بیانگر خطا در آن بکار برده می‌شود.

یک کاربرد متعارف exit موقعی است که یک شرط الزامی خاص برای اجرای برنامه، برقرار نباشد، برای مثال یک بازی کامپیوتری را فرض کنید که در آن وجود کارت گرافیکی رنگی در سیستم ضروری است. تابع اصلی این بازی ممکن است مشابه زیر باشد :

```
main ( )
  {
    if  (!color-card( ))
      exit(1) ;
    play( ) ;
  }
```

که در آن color-card تابعی است که کاربر تعریف کرده که اگر کارت رنگی در سیستم موجود باشد true برمی‌گرداند و در غیر اینصورت false برمی‌گرداند و اجرای برنامه قطع می‌شود .

- **تمرین و پاسخ**

تمرین ۱ – برنامه‌ای بنویسید که یک سطر متن را بخواند و تعداد محلهای خالی (blank) موجود در آن را تعیین کرده و چاپ کند.

```
#include<stdio.h>
main ( )
{
  int  ch , count = 0 ;
  printf("enter a sentence:\n");
  ch = getchar( ) ;
  while (ch!='\n)
    {
      if (ch = =' ')
        count + + ;
      ch = getchar( )
    }
  printf("the number of  spaces is %d\n", count) ;
}
```

توضیح – به کمک 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 حلقه while انجام می‌گیرد. هر بار در درون حلقه while کاراکتر خوانده شده با کاراکتر معرف فضای خالی مقایسه می‌گردد تا درصورت تطابق ، یک واحد به شمارنده count که تعداد محل خالی در متن را می‌شمارد، افزوده شود. به‌طوری که قبلاً بیان شد، در حلقه while شرط مورد نظر برای اجرای بدنه 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 بعنوان تعداد محلهای خالی موجود در متن مورد نظر چاپ می‌شود.

تمرین ۲ – مبلغ ۵۰۰۰۰۰ ریال در آغاز سال، در بانکی به‌عنوان تشکیل سرمایه به حساب گذاشته شده است که هر سال ۱۶ درصد سود به آن تعلق می‌گیرد. سود، در پایان هر سال محاسبه گردیده و به موجودی افزوده می‌شود و سرمایه برای سال جدید منظور می‌گردد . برنامه‌ای بنویسید که موجودی را برای ۵ سال متوالی، در پایان هر سال محاسبه و چاپ کند .

حل: برنامه مورد نظر به همراه خروجی آن در زیر نمایش داده شده است :

```
#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

تمرین ۳ – 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 و یا از کاراکترهای دیگر است .

حل: این برنامه که در زیر نمایش داده شده است ، نحوه استفاده از دستور if-else تودرتو را نشان می‌دهد :

```
#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) ;
}
```

تمرین 4 – عددی که مساوی مجموع مقسوم‌علیه‌های خود باشد را عدد کامل نامند مانند عدد ۶ یعنی :

$$6 = 3 + 2 + 1$$

برنامه‌ای بنویسید که عددی صحیح و مثبت خوانده و تعیین کند که آیا این عدد کامل است یا نه .

حل : برنامه مورد نظر در زیر نشان داده شده است .

```
#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 یک حلقه for هر بار باقیمانده تقسیم n بر اعداد ۲ تا k به‌دست می‌آید. چنانچه باقیمانده برابر صفر باشد؛ یعنی n بر i بخش‌پذیر باشد مقدار i به‌عنوان 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 مقسوم‌علیه‌های n را خواهد داشت. سپس بررسی می‌شود که آیا این مقدار برابر خود n است یا نه ، که برحسب مورد پیغام مناسب چاپ می‌شود.

تمرین ۵ – برنامه‌ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را تهیه و چاپ کند .

حل : برنامه مورد نظر به همراه خروجی آن در زیر نمایش داده 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") ;
     }
```

```
}
```

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

تمرین ۶ – در ریاضی سری فیبوناچی fibonacci **به‌صورت زیر تعریف شده است :**

1 1 2 3 5 8 13 ...

که در آن دو جمله اول و دوم برابر یک است؛ در مورد بقیه جملات هر جمله برابر است با مجموع دو جمله قبلی آن . برنامه‌ای بنویسید تا جملات اول تا 10 **سری مزبور و مجموع آنها را محاسبه و چاپ کند .**

حل : برنامه مورد نظر به همراه خروجی آن ، در زیر نشان داده شده است :

```
#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 <=1۰ ; + +i)
   {
    a2 = a0 + a1 ;
    a0 = a1 ;
    a1 = a2 ;
    printf("%d ", a2) ;
   }
}
```

خروجی برنامه

```
1  2  3  5  8  13  21  34  55۱
```

تمرین ۷ – برنامه بنویسید که چهار عمل اصلی یک ماشین حساب (جمع، تفریق، ضرب و تقسیم)

را انجام دهد .

حل: 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'+', '-', '*' و یا'/' باشد عمل ریاضی مربوط روی مقادیر عملوندها انجام گیرد. برنامه مورد نظر و خروجی آن برای بعضی مقادیر نمونه، در زیر نشان داده شده است .

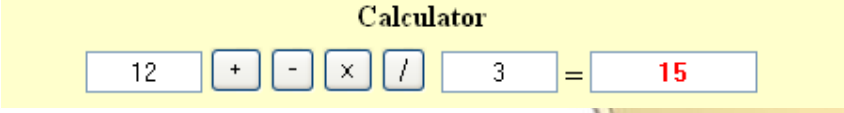
```
main ( )
{
  float  num1, num2 ;
  char  op ;
  while (1)
    {
    printf("Type number, operator, number : \n");
    scanf("%f %c %f , &num1, &op, &num2);
    switch (op)
      {
      case '+':
              printf("=%f", num1 + num2 ) ;
              break ;
      case '-':
              printf("=%f", num1 - num2 ) ;
              break ;
      case '*':
              printf("=%f", num1 * num2 ) ;
              break ;
      case '/':
              printf("=%f", num1 / num2 ) ;
              break ;
      default :
              printf ("Error") ;
      }
  printf("\n") ;
  }
}
```

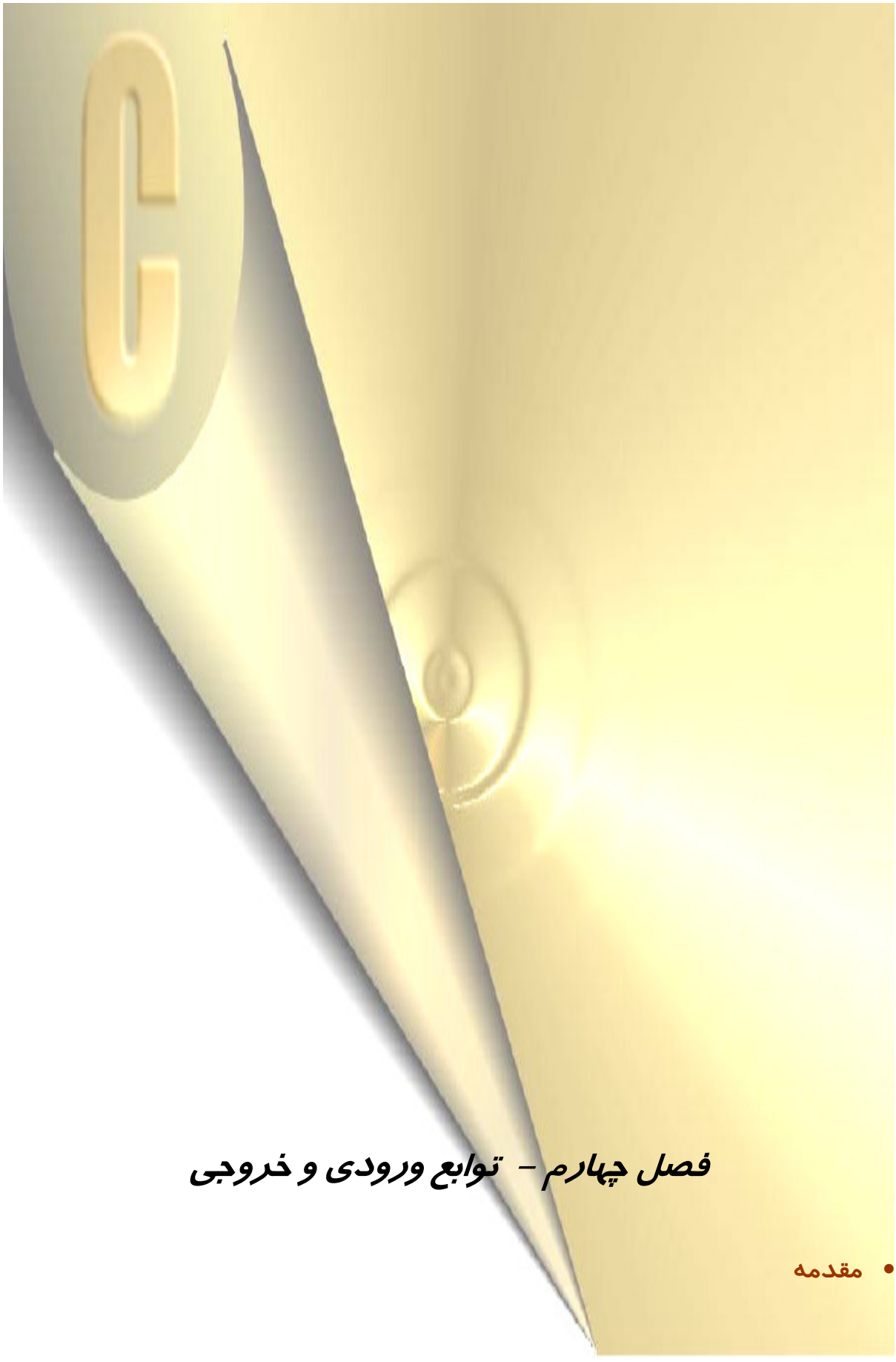
خروجی برنامه

```
Type number , operator , number :
5 + 4.2 = 9.2

Type number , operator , number :
50 = 600.000000×12
```

نمونه کاربردی از این برنامه به این شکل است :





فصل چهارم - توابع ورودی و خروجی

- مقدمه

زبان C با مجموعه‌ای از توابع کتابخانه‌ای همراه است که تعدادی از آنها تابع ورودی و خروجی می‌باشند. برخی از این توابع متداول که اغلب در کتابخانه یا فایل "stdio.h" قرار دارند در این فصل مورد بررسی قرار می‌گیرند .

توابع فرمت دار scanf و printf اجازه انتقال اطلاعات میان کامپیوتر و دستگاههای ورودی و خروجی استاندارد را می‌دهند. دو تابع getchar و putchar موجب انتقال یک کاراکتر به حافظه و بالعکس می‌گردد. دو تابع gets و puts نیز ورود و خروج رشته‌ها را ساده‌تر می‌سازند.

برخی از توابع ورودی و خروجی، نیاز به آرگومان ندارند . بعضی از این توابع مقداری را برنمی‌گردانند. برخی بصورت مستقل بکار برده شوند و برخی ممکن است در داخل عبارات و دستورها بکار برده شوند.

- **توابع ورودی و خروجی فرمت‌دار**

دو تابع scanf و printf (که با آنها آشنا هستید)، توابع فرمت‌دار هستند که می‌توانند روی هر نوع داده‌ای از نوع توکار یا built-in عمل نمایند . این داده ها ، داده‌هایی هستند که ساختار آنها به‌صورت پیش‌فرض برای کامپایلر شناخته شده است، مانند داده‌های از نوع اعداد، کاراکتر و رشته . اصطلاح فرمت‌دار یا formantted به این مفهوم است که این توابع می‌توانند داده‌ها را در فرمتهای مختلف که تحت کنترل شماست، بخوانند و بنویسند. تابع printf برای نوشتن داده‌هایی روی کنسول که به‌طور متعارف، صفحه نمایش تصویر یا مونیتور است، بکار برده می‌شود. تابع scanf که مکمل تابع printf است، داده‌ها را از صفحه کلید می‌خواند.

- **تابع printf( )**

این تابع می‌تواند به تعداد دلخواه آرگومان بپذیرد که آرگومان اول دارای مفهوم خاص است و رشته فرمت format string و یا رشته کنترل control string نامیده می‌شود. رشته کنترل در داخل زوج گیومه یا double quotation قرار می‌گیرد و مشخص می‌کند که باید چند قلم آرگومان داده چاپ شود و فرمت آنها چگونه است. فرم کلی این تابع بصورت زیر است :

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

و یا :

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

رشته کنترل دربردارنده دو نوع اقلام داده است. نوع اول شامل رشته یا کاراکترهایی است که به همان صورت روی صفحه تصویر، نمایش داده خواهد شد و نوع دوم شامل فرامین فرمت است که با علامت "%" آغاز می‌گردد که به دنبال آن کد فرمت و یا مشخص‌کننده 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، بکار می‌روند .

برخی مؤلفین کد فرمت را ، فرامین فرمت نیز می‌گویند .

لیست کد یا فرامین فرمت که در فرمان printf بکار برده می‌شود، در جدول زیر نشان داده شده است .

لیست فرامین فرمت برای خروجی به‌وسیلهٔ تابع printf ( )

کد	معنی و مفهوم فرمان فرمت
%c	داده، بصورت تک‌کاراکتر نمایش داده می‌شود.
%d	داده، بصورت عدد صحیح علامت‌دار نمایش داده می‌شود.
%i	داده، بصورت عدد صحیح علامت‌دار نمایش داده می‌شود.
%f	داده، بصورت عدد اعشاری، ممیز شناور و بدون نما یا توان نمایش داده می‌شود.
%e	داده، بصورت floating-point ولی به فرم نمایی یا علمی نمایش داده می‌شود.
%g	داده، بصورت floating-point به فرم %f یا %e (هرکدام کوتاهتر باشد) نمایش داده می‌شود.
%u	داده، بصورت عدد صحیح بدون علامت نمایش داده می‌شود.
%s	داده، بصورت رشته نمایش داده می‌شود.
%o	داده، بصورت عدد صحیح در مبنای ۸ نمایش داده می‌شود.
%x	داده، بصورت 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 بکار برده می‌شود. (مثل ld)
%p	در مورد داده از نوع اشاره‌گر بکار برده می‌شود .
%%	یک علامت % چاپ می‌کند که در توربو c وجود دارد .

مثال – خروجی برنامه زیر :

```
#include<stdio.h>
main ( )
  {
   float  A=2 , B=3.0 ;
   printf("%f %f %f %f ", A , B , A+B , B/A) ;
  }
```

بصورت زیر خواهد بود :

1.500000 5.000000 3.000000 2.000000

توجه داشته باشید که آرگومانهای اول و دوم تک‌متغیر می‌باشند. ولی آرگومانهای سوم و چهارم عبارات محاسباتی هستند که اول مقدار آنها محاسبه می‌گردد و سپس نتیجه، براساس کد فرمت مربوط چاپ می‌شود.

مثال – در قطعه برنامه زیر مقادیر متغیرهایی از نوعهای مختلف چاپ می‌گردد.

```
#include<stdio.h>
main ( )
  {
    int  st_no =21385 ;
    char  st_name[16] = {"book"};
    float   grade =14.25 ;
    printf(" %d %s %f " , st_no , st_name , grade) ;
  }
```

اولی عدد صحیح، دومی آرایه‌های از نوع کاراکتر یا به عبارت دیگر رشته (رشته‌ها بعداً مورد بررسی قرار خواهد گرفت) و سومی از نوع floating point است که به ترتیب با فرمان فرمت %f, %s, %d مشخص شده‌اند. خروجی دستور مزبور بصورت زیر خواهد بود.

21385 book 14.25

در برنامه مذکور به‌علت وجود یک محل خالی بین فرامین فرمت در رشته کنترل، بین مقادیر چاپ شده نیز یک فضای خالی ایجاد گردید.

مثال – در برنامه زیر، هر دو دستور printf دارای آرگومانهای یکسان هستند؛ ولی در اولی با فرمان فرمت %f و در دومی با فرمان فرمت %e معرفی شده‌اند (وجود "\n" به تعداد دو بار در اولی، موجب ایجاد یک سطر خالی، بین خروجی دو دستور می‌گردد).

```
#include<stdio.h>
main ( )
  {
    double  A=5000.0 , B=0.0025 ;
    printf(" %f %f %f %f\n\n ", A , B , A*B , A/B) ;
    printf(" %e %e %e %e ", A , B , A*B , A/B) ;

  }
```

و خروجی برنامه مزبور بصورت زیر خواهد بود:

5000.000000 0.002500 12.500000 2000000.000000

5.000000e+03 2.500000e-03 1.250000e+01 2.000000e+06

در سطر اول مقادیر A , B , A*B , A/B به فرم استاندارد floating point نشان داده شده،

درحالی‌که در سطر دوم همان مقادیر، به علت استفاده از فرمان فرمت "%e"، به فرم نمایش علمی، یعنی با نما ظاهر شده است.

ملاحظه کنید که هر مقدار تا ۶ رقم دقت، پس از نقطه اعشار نمایش داده شده است. به هرحال این تعداد ارقام را می‌توان، با قرار دادن میزان دقت در رشته کنترلی (که بعد بیان خواهد شد) 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 تعیین شده 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 مانند m (به‌عنوان مشخص‌کننده 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 یا blank باقی بماند. ولی اگر طول رشته یا عدد بزرگتر از طول پیش‌بینی شده برای آن باشد، طول پیش‌بینی شده نادیده گرفته شود و اطلاعات بطور کامل نمایش داده شود. اگر بخواهید در مورد اطلاعات عددی فضای اضافی با صفر پر شود، سمت چپ عدد m رقم "0" را قرار دهید. برای مثال %05d موجب می‌گردد که اگر مقدار چاپ شده از پنج رقم کمتر باشد، سمت چپ آن به تعداد لازم صفر قرار گیرد بطوری که مقدار چاپ شده پنج‌رقمی باشد.

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

مثال – به برنامه زیر دقت کنید :

```
#include<stdio.h>
main ( )
 {
   int  i = 12345 ;
   float  x =345.125 ;
   printf("3d %5d %8d\n\n", i , i , i) ;
   printf("%3f %10f %13f\n\n", x , x , x) ;
   printf("%3e %10e %13e\n\n", x , x , x) ;
 }
```

و خروجی برنامه فوق بصورت زیر خواهد بود :

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

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 بطور کامل نمایش داده شده است، اگرچه طول میدان پیش‌بینی شده کمتر از عدد مورد نظر می‌باشد (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 رشته 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 رشته کنترلی است؛ سه محل خالی دیگر نیز برای پُرکردن حداقل فضای پیش‌بینی شده برای میدان مورد نظر (که در اینجا هشت کاراکتر برای یک عدد پنج رقمی است)، می‌باشد.

شرایط مشابهی برای دو سطر دیگر خروجی وجود دارد. فقط باید توجه کرد که کد فرمت برای سطر دوم از نوع f و برای سطر سوم از نوع e می‌باشد.

مثال – در برنامه زیر همان مثال قبلی با استفاده از کد فرمت نوع g نوشته شده است.

```
#include<stdio.h>
main ( )
 {
   int  i = 12345 ;
   float  x =345.125 ;
   printf("%3d %5d %8d \n", i , i , i) ;
   printf("%g %10g %13g \n", x , x , x) ;
   printf("%3g %13g %16g \n", x , x , x) ;
 }
```

و خروجی برنامه مذکور بصورت زیر خواهد بود:

```
12345 12345   12345
345.125   345.125     345.125
345.125     345.125       345.125
```

در اینجا مقادیر floating point با کد فرمت f چاپ شده است زیرا این فرم، نتیجه را کوتاهتر نشان می‌دهد. فواصل بین مقادیر چاپ شده نیز براساس درنظر گرفتن حداقل پهنای پیش‌بینی شده در رشته کنترلی برای آرگومانهای مورد نظر است. چگونگی پیش‌بینی حداقل طول میدان در یک تابع printf در بالا بیان شد. می‌توان ماکزیمم تعداد ارقام اعشار در مورد مقادیر floating point یا ماکزیمم تعداد کاراکتر برای یک رشته را نیز مشخص کرد. مشخصه 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 همینطور است)، طول میدن 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، جزءِ اعشار گرد می‌شود، بطوری که تعداد ارقام آن با دقت یا precision پیش‌بینی شده مطابقت نماید. مثال زیر این موضوع را روشن می‌سازد.

مثال – به برنامه زیر دقت 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) ;
  }
```

و خروجی برنامه بالا بصورت زیر خواهد بود :

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

سطر اول به وسیلهٔ کد فرمت f ایجاد شده است. ملاحظه می‌کنید که عدد سوم به لحاظ مشخصه دقت مربوط (که یک رقم اعشار پیش‌بینی گردیده) گرد شده است. همچنین مشاهده می‌کنید که با درنظر گرفتن حداقل طول میدان (هفت کاراکتر) در سمت چپ عدد سوم، دو محل فضای خالی به لحاظ حداقل طول میدان و یک محل فضای خالی نیز به لحاظ وجود یک محل خالی بین فرمان فرمت فیلدهای دوم و سوم، ایجاد شده است.

سطر دوم که با کد فرمت نوع e ایجاد شده، دارای مشخصه‌های مشابه سطر اول است. بازهم ملاحظه می‌شود که عدد سوم گرد شده است تا با دقت پیش‌بینی شده (۳ رقم اعشار) تطابق داشته باشد. همچنین چهار فضای خالی نیز با درنظر گرفتن حداقل طول میدان (دوازده کاراکتر) و وجود یک محل خالی بین فرمان فرمت فیلدهای دو و سوم، در سمت چپ آن ایجاد شده است.

ضرورتی ندارد که مشخصه دقت با حداقل طول میدان توأم بکار برده شود. بلکه می‌توان مشخصه دقت را بدون ذکر حداقل طول میدان نیز بکار برد. ولی به هرحال نقطه معرف اعشار باید در جلوی آن بکار برده شود.

علاوه بر پهنای میدان، شاخص دقت و کاراکترهای تبدیل، هر گروه از رشته کنترل می‌تواند

شامل یک flag نیز باشد که در شکل ظاهری خروجی اثر می‌گذارد. flag، بلافاصله بعد از علامت (%) قرار می‌گیرد. بعضی کامپایلرها اجازه می‌دهند که بیش از یک flag بکار برده شود، که در این صورت باید بطور پیاپی قرار گیرند. flagهای متداول در جدول زیر درج شده‌اند.

Flag های متداول و نقش آنها

Flag	معنی و نقش آنهـــا
–	اقلام داده‌ها در داخل میدان، از سمت چپ تراز می‌شوند (فضای خالی مورد نیاز برای پرکردن حداقل پهنای میدان نیز از طرف راست انباشته می‌شود.)
+	در مورد اقلام عددی، علامت آن (+ یـا -) نیـز در جلـوی آن ظـاهر مـی‌گـردد. بدون استفاده از این flag فقط در مورد مقادیر منفـی علامـت آنهـا در خروجـی ظاهر می‌شود.
0	در مورد اقلام عددی موجب می‌شود که فضای خالی سمت چپ به جای blank با صفر پرشود. البته فقط در مورد اقلامی کـه از سـمت راسـت تـراز مـی‌گردنـد، مصداق دارد.
▪▪ blankspace	در مورد هر قلم داده عددی علامت‌دار و مثبت یک blank در جلـوی آن ظـاهر می‌شود. فلاگ (+) آن را لغو می‌کند (اگر هر دو باهم بکار برده شوند).
# (با نوع تبدیل -0 و x-)	موجب می‌شود که در جلوی اقلام داده اکتـال یـا هگزادسـیمال متنـاظراً 01 و 0x ظاهر شود.
# (با نوع تبدیل g-, f- e- ,)	موجب می‌شود که در تمام اعداد floating-point، علامـت ممیـز "." ظـاهر شـود، اگرچه عدد مورد نظر جزءِ اعشار نداشته باشد؛ همچنین در مورد نوع تبـدیل g از برش دادن (نادیده گرفته شدن) صفرهای بـدون معنـی راسـت trailing zeros جلوگیری می‌کند.

مثال – برنامه زیر کاربرد فلاگها را در مورد مقادیر صحیح و اعشاری (با ممیز شناور) نمایش می‌دهد:

```
#include<stdio.h>
main ( )
 {
  int  a =123 ;
  float  b=15.0 , c = -3.5 ;
```

```
  printf(": %5d %7.0f %10.1e : \n", a , b , c) ;
  printf(": %-5d %-7.0f %-10.1e : \n", a , b , c) ;
  printf(": %+5d %+7.0f %+10.1e : \n", a , b , c) ;
  printf(": %7.0f %#7.0f %7g %#g :" , b , b , c , c) ;
 }
```

خروجی برنامه بالا بصورت زیر خواهد بود (که در آن علامت ":" آغاز اولین فیلد و پایان آخرین فیلد را نمایش می‌دهد.)

```
: 123     15    -3.5+00 :
: 123  15     -3.5e+00 :
: +123     +15   -3.5e+00 :
:+123  +15    -3.5e+00 :
:    15     15.    -3.5 -3.500000 :
```

اولین خط، خروجی را بدون استفاده از فلاگ نمایش می‌دهد که هر عدد در فضای مشخص شده 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 که این بار بعلت وجود فلاگ "-" همه اعداد از سمت چپ تراز شده‌اند. خط سوم، نقش استفاده از فلاگ "+" را نمایش می‌دهد. در این حالت اعداد مانند خط اول از سمت راست تراز می‌شوند. اما علامت مربوط نیز (در هر دو حالت مثبت و منفی‌بودن) در جلوی آنها ظاهر شده است.

در خط چهارم، نقش ترکیب دو فلاگ "-" و "+" را مشاهده می‌کنید و ملاحظه می‌کنید که ضمن اینکه اعداد (بعلت وجود فلاگ "-") از سمت چپ تراز شده‌اند، علامت مربوط نیز (به‌علت وجود فلاگ "+") در جلوی آنها ظاهر شده است. بالاخره خط پنجم دو مقدار floating point را یکبار بدون وجود فلاگ و بار دیگر با استفاده از فلاگ "#" نمایش می‌دهد، و ملاحظه می‌کنید که نقش این فلاگ آن است که در مورد عدد 15 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 – برنامه زیر را درنظر بگیرید که اعداد را در مبنای ۱۰ ، ۸ و ۱۶ نمایش می‌دهد :

```
#include<stdio.h>
main ( )
 {
   int  i = 1234 , j = 0155 , k = 0xa06b ;
   printf(": %6u %6o %6x : \n", i , j , k) ;
   printf(": %-6u %-6o %-6x : \n", i  ,j , k) ;
   printf(": %#6u %#6o %#6X : \n" ,  i , j , k) ;
   printf(": %06u %06o %06X : \n", i , j , k) ;
 }
```

خروجی این برنامه بصورت زیر خواهد بود (در اینجا نیز ":" آغاز اولین فیلد و پایان آخرین فیلد را در هر خط نمایش می‌دهد.)

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

اولین خط، بدون استفاده از فلاگ، اعداد را بدون علامت و بترتیب 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 پیش‌بینی شده برای آنها از سمت چپ تراز شده‌اند.

در خط سوم نتیجه استفاده از فلاگ "#" در خروجی مشاهده می‌گردد. این فلاگ موجب می‌گردد که در جلوی اعداد در مبنای ۸ و ۱۶ به‌ترتیب "0" و "0x" ظاهر شود. همچنین ملاحظه می‌کنید که به سبب استفاده از حروف بزرگ "X" در کاراکتر تبدیل، حروف موجود در اعداد مبنای ۱۶، در خروجی بصورت 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 (تا پرشدن فضای پیش‌بینی شده با صفر پر شود. در اینجا نیز بعلت استفاده از حروف بزرگ "X" در کاراکتر تبدیل، حروف موجود در اعداد مبنای ۱۶ ، در خروجی، بصورت حروف بزرگ ظاهر شده‌اند.

- **تابع scanf( )**

داده‌های ورودی می‌توانند از طریق یک دستگاه ورودی استاندارد به کمک تابع کتابخانه‌ای scanf وارد کامپیوتر شوند. بطوری که در گذشته نیز بیان شد، تابع scanf نیز تابع فرمت‌دار می‌باشد و مشابه تابع printf ولی در جهت عکس عمل می‌کند. به کمک این تابع می‌توان داده‌های عددی، کاراکتر، رشته‌ها و یا ترکیبی از آنها را وارد کامپیوتر کرد. فرمت این تابع مشابه فرمت تابع printf است. فرم کلی این تابع بصورت زیر است :

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

و یا :

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

نقش رشته کنترل مشابه همان است که در مورد تابع printf بیان شد، لذا از توضیح بیشتر خودداری می‌شود.

مشابه printf این تابع نیز می‌تواند هر تعداد آرگومان را دارا باشد، که در آن اولین آرگومان رشته فرمت یا رشته کنترل است. همچنین این تابع، اغلب همان کد فرمت تابع printf را بکار می‌برد؛ برای مثال بطوری که در مثالهای قبلی نیز مشاهده شد، کدهای فرمت :

%s, %c , %f , %d

به ترتیب برای خواندن داده‌هایی از نوع مقادیر: صحیح، اعشاری (کسری)، کاراکتر و رشته بکار برده می‌شود. تفاوت مهم بین این دو تابع آن است که در جلوی آرگومانها ، اپراتور آدرس یعنی "&" نیز قرار می‌گیرد. برای مثال دستور:

```
scanf("%d" , &num) ;
```

سیستم را هدایت می‌کند که داده ورودی را به‌صورت عدد صحیح از طریق ترمینال دریافت کند و این مقدار را در متغیر num ذخیره نماید. البته اگر بخواهید مقداری را برای متغیر رشته‌ای بخوانید، نیازی به اپراتور "&" نخواهد بود؛ زیرا به‌طوری که بعد ملاحظه خواهید کرد، رشته‌ها در زبان c بصورت آرایه‌ای از نوع کاراکتر معرفی می‌گردد و نام آرایه نیز معرف آدرس آرایه (یعنی آدرس اولین عنصر آن) می‌باشد، مانند مثال زیر:

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

و اگر در این برنامه برای متغیر name رشته "book" را وارد کرده باشیم، خروجی برنامه بالا کلمه book خواهد بود.

فرامین یا کدهای فرمت ، برای داده‌های ورودی (که مشخص‌کننده تبدیل نیز نامیده می‌شوند) در جدول زیر نشان داده شده است.

کد یا کاراکترهای فرمت در تابع scanf ( )

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

در استفاده از تابع scanf ، آرگومانها بعنوان متغیرها یا آرایه‌هایی که نوع آن باید با کد فرت مربوط تطابق داشته باشد، نوشته می‌شود و همانطوری که بیان شد، در جلوی آنها اپراتور آدرس یعنی "&" نیز بکار برده می‌شود (درواقع آرگومانها اشاره‌گرهایی هستند که مشخص می‌کنند اقلام داده‌ها در حافظه کامپیوتر، در کجا ذخیره می‌شود). به هرحال به‌طوری که بعد در مورد آرایه‌ها توضیح داده خواهد شد، در مورد استفاده از نام آرایه نباید اپرتور "&" را جلوی آن قرار داد و این موضوع را در مورد رشته‌ای که آرایه‌ای از کاراکترها می‌باشد، ضمن مثال ملاحظه نمودید. مثال زیر این موضوع را روشنتر می‌سازد.

مثال – برنامه زیر را درنظر بگیرید:

```
#include<stdio.h>
main ( )
 {
   int  id ;
   char  name[5] ;
   float  grade ;
   .....
   .....
   scanf("%d %s %f " , &id , name , &grade) ;
   .....
   .....
  }
```

در تابع scanf استفاده شده در برنامه بالا، رشته کنترلی شامل ۳ گروه است :

" %d , %s , %f "

اولین گروه "%d" دلالت بر این دارد که آرگومان متناظر با آن (یعنی &id) معرف مقدار صحیح در مبنای دهدهی است.

دومین گروه "%s" دلالت بر این دارد که دومین آرگومان (یعنی name) معرف یک رشته است.

و بالاخره سومین گروه "%f" مشخص می‌کند که آرگومان سوم (یعنی "&grade")از نوع اعشار با ممیز شناور است .

ملاحظه می‌کنید که در جلوی متغیرهای عددی id و grade اپراتور "&" آورده شده است، ولی در مورد متغیر name به لحاظ اینکه نام آرایه می‌باشد، اپراتور مزبور وجود ندارد.

اقلام واقعی داده‌ها عبارتند از مقادیر عددی، کاراکتر تنها یا رشته‌ها و یا ترکیبی از آنها که از طریق دستگاه ورودی استاندارد (معمولاً یک صفحه کلید) وارد می‌شوند. به هرحال اقلام داده‌ها باید ازنظر نوع به‌ترتیب ورود، متناظر با آرگومانهای مربوط باشند. داده‌های عددی به‌صورت مقادیر ثابت عددی نوشته می‌شوند که اگر در مبنای 8 یا 16 باشند، نیاز به اینکه با "0" (در مورد مبنای 8) و "0x" یا "x" (در مورد مبنای 16) آغاز گردند، نمی‌باشد.

مقادیر اعشاری با ممیز شناور ممکن است شامل علامت ممیز یا علامت نما و یا هر دو باشد. اگر

دو یا چندین اقلام داده وارد شوند، باید به‌وسیله کاراکترهای تفکیک‌کننده که whitespace نامیده می‌شوند، از یکدیگر جدا شوند. کاراکترهای فضای خالی یا space یا blank ، tab و خط جدید یا newline متداول‌ترین کاراکترهای whitespace می‌باشند.

اقلام داده‌ها را می‌توان روی دو یا چندین خط ادامه داد؛ زیرا بطوری که بیان شد، خط جدید نیز کاراکتر whitespace درنظر گرفته می‌شود.

برای مثال اگر داده‌های ورودی برنامه بالا به ترتیب 12345 , sum , 97.5 باشد، می‌توان آنها را به صورتهای زیر وارد کامپیوتر نمود:

7.5912345 sum

12345

یا

sum

.597

یا

12345 sum

97.5

و یا

12345

sum 97.5

رشته‌هایی که با کد فرمت "%s" بکار برده می‌شوند، با یک کاراکتر whitespace خاتمه می‌یابند. بنابراین اگر رشته‌ای شامل کاراکتر whitespace باشد، نمی‌تواند به طریق مذکور در مثال قبل وارد کامپیوتر گردد. به هرحال راههایی برای کار با این‌گونه رشته‌ها و ورود آن به حافظه کامپیوتر وجود دارد. یک راه استفاده از تابع getchar داخل یک حلقه تکرار است مانند مثال زیر:

مثال – برنامه زیر یک خط متن حداکثر به طول 79 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 حلقه for ، شمارنده k از صفر شروع می‌شود و مقدار آن در هر تکرار یک واحد افزایش می‌یابد و در هر تکرار یک کاراکتر به‌وسیلهٔ 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 حلقه بعدی محتوای آرایه line[] که دربردارنده رشته دریافت شده است، چاپ می‌گردد. (دو تابع getchar و putchar دوباره بررسی خواهد شد). راه دیگر برای ورود رشته‌ها به حافظه کامپیوتر، استفاده از تابع gets است که در مبحث رشته‌ها مورد بحث قرار خواهد گرفت.

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

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 می‌تواند در داخل رشته‌ها منظور گردد.) به محض اینکه کاراکتری خوانده شود که در داخل کروشه‌ها وجود نداشته باشد، عمل خواندن خاتمه می‌پذیرد. درضمن یک کاراکتر null بطور اتوماتیک به پایان رشته افزوده خواهد شد.

مثال – برنامه زیر کاربرد تابع scanf را برای خواندن رشته‌هایی که شامل حروف بزرگ و فضای خالی است، نشان می‌دهد. طول این رشته نامشخص خواهد بود. اما حداکثر برابر ۷۹ کاراکتر (یعنی 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 ، رشته COMPUTER SCIENCE وارد شود، وقتی که برنامه اجرا می‌گردد، تمامی رشته مزبور به آرایه line نسبت داده می‌شود. به هرحال اگر یکی از حروف رشته 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 آرایه 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 رشته مورد نظر را در داخل کروشه ذکر کنیم (که این روش در اغلب موارد مانند مثال بالا طولانی خواهد بود)، فقط کاراکترهایی را که مجاز نیستیم در رشته‌ها بکار ببریم، مشخص نماییم. 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 بقیه 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، می‌تواند شامل هر کاراکتر آسکی به جز کاراکتر خط جدید باشد؛ 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- فرض کنید که یک برنامه C شامل دستورهای زیر است :

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

وقتی که تابع scanf در برنامه بالا اجرا می‌گردد، رشته‌ای به طول نامشخص (ولی حداکثر 79 کاراکتر) از طریق دستگاه ورودی استاندارد وارد می‌گردد و به line نسبت داده می‌شود. هیچ گونه محدودیتی در مورد کاراکترهای تشکیل‌دهنده رشته وجود نخواهد داشت، فقط باید در یک خط بگنجد. برای مثال رشته زیر می‌تواند از طریق صفحه کلید وارد گردد و به line نسبت داده شود.

WE LEARN MATHEMATICS .

- **تابع getchar( )**

برای خواندن یک کاراکتر از طریق دستگاه ورودی، می‌توان علاوه بر تابع scanf از تابع getchar استفاده نمود. تابع مزبور که جزءِ کتابخانه I/O زبان استاندارد C است ، یک کاراکتر از دستگاه ورودی استاندارد (معمولاً صفحه کلید) می‌خواند. این تابع دارای آرگومان نیست و بطور متعارف در یک دستور انتساب یا جایگذاری بکار برده می‌شود و کاراکتر دریافتی از ورودی را به متغیری که در سمت چپ دستور جایگذاری مورد نظر است، برمی‌گرداند (اختصاص می‌دهد) و فرم کلی آن بصورت 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 .

مثال – دستورهای زیر را در یک برنامه، درنظر بگیرید :

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

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

اگر هنگام خواندن کاراکتر توسط تابع getchar ، شرایط پایان فایل پیش آید مقدار سمبولیکی EOF بطور خودکار برگشت داده خواهد شد (این مقدار در داخل فایل stdio.h اختصاص داده خواهد شد. بطور متعارف مقدار ۱- به EOF اختصاص داده می‌شود، اگرچه ممکن است این مقدار از کامپایلری به کامپایلر دیگر فرق کند.) ظاهر شدن EOF به این طریق ، راه ساده‌ای برای تشخیص پایان فایل در هنگام اجرای آن است . (این مورد ، در مبحث فایلها بیشتر مورد بحث قرار خواهد گرفت، لذا به هیچ وجه نگران آن نباشید). می‌توان تابع getchar را نیز برای خواندن رشته چند کاراکتری بصورت حلقه تکرار بکار برد که در هر تکرار یا کاراکتر را بخواند .

- **تابع putchar( )**

تابع putchar را می‌توان برای شمارش یک کاراکتر روی خروجی استاندارد که معمولاً صفحه نمایش است بکار برد. درواقع نقش آن مشابه تابع getchar در جهت عکس می‌باشد. طبیعی است کاراکتری که انتقال می‌یابد بصورت ثابت کاراکتری و یا بصورت یک متغیر از نوع کاراکتر به‌عنوان آرگومان تابع مزبور بکار برده می‌شود. فرم کلی این تابع بصورت زیر می‌باشد:

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

البته می‌توان ثابت کاراکتری را نیز بعنوان آرگومان تابع مزبور بکار برد.

مثال – برنامه زیر بتدریج در هر بار که یک کاراکتر می‌خواند، آن را روی خروجی چاپ می‌کند :

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

در مثال بالا، C کاراکتر توصیف شده است. هر بار که یک کاراکتر از طریق دستگاه ورودی استاندارد به C 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" می‌باشد، و همیشه غیرصفر است، براساس قوانین زبان c، این عبارت همیشه درست یا true می‌باشد. بنابراین ساختار:

```
while (1)
  {
    ----------
    ----------
  }
```

یک حلقه نامتناهی است، لذا تنها راه متوقف ساختن برنامه به‌وسیلهٔ یک وقفه یا interrupt می‌باشد که این کار با control-c عملی خواهد شد.

راه دیگر برای نوشتن برنامه بالا بصورت زیر است :

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

همان برنامه می‌تواند بصورت زیر نوشته شود :

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

بطوری که در گذشته نیز بیان شد EOF در برنامه بالا یک علامت سمبولیک پایان فایل است. آنچه که در واقعیت به‌عنوان نشانه پایان فایل بکار برده می‌شود، تابع سیستم است. برای این کار اغلب عدد ۱- بکار برده می‌شود، ولی سیستمهای مختلف می‌تواند مقادیر متفاوتی داشته باشد. با گنجانیدن فایل stdio.h و بکار بردن ثابت سمبولیکی EOF ، برنامه را قابل حمل یا قابل اجرا ساخته‌ایم. یعنی فایل مبنا می‌تواند روی سیستمهای مختلف بدون تغییرات اجرا شود .

ملاحظه می‌کنید که در دو روش اخیر، متغیر C به جای char به‌صورت int معرفی گردیده است. هرچه که برای نشان دادن پایان فایل بکار برده شده باشد، نمی‌تواند مقداری باشد که یک کاراکتر را معرفی نماید. حال چون C به‌صورت int معرفی شده است، می‌تواند مقادیر تمام کاراکترهای ممکن و همینطور مقدار ویژه EOF را نگهداری نماید.

همانطوری که بیان شد، 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.

حال بعنوان نمونه و برای آشنایی کافی با نحوه استفاده از این دو تابع، به چند مثال بعدی توجه نمایید.

مثال – برنامه زیر یک خط متن را از ورودی با حروف کوچک دریافت داشته، 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 برنامه بالا از تابع توکار toupper استفاده شده است. نقش این تابع آن است که حروف کوچک را به بزرگ تبدیل می‌کند. بنابراین اگر حروف ورودی بطور اولیه (هنگام تایپ) حروف بزرگ، ارقام و مشابه آن باشند، به فرم اولیه خود نمایش داده خواهند شد. برای مثال اگر ورودی بصورت :

my book is on the table .

باشد، خروجی بصورت زیر خواهد بود.

MY BOOK IS ON THE TABLE .

مثال – برنامه 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 بصورت رمز در می‌آورد و نمایش می‌دهد .)

```
#include<stdio.h>
#define  space  ' '
main ( )
  {
    char  ch ;
    ch = getchar ( ) ;       /* read a character */
    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 */
    }
}
```

بعنوان مثال اگر ورودی بصورت: computer science باشد، خروجی بصورت زیر خواهد بود :

dpnqvufs tdjfodf

با ترکیب دو عبارت خواندن و تست کردن پایان متن در یک عبارت، برنامه مزبور را می‌توان بصورت ساده‌تر 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 یک روش 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 می‌گردد. وجود پرانتز دور عبارت :

ch = getchar( )

آن را اپراند چپ اپراتور =! می‌سازد. اگر آن را حذف کنیم نتیجه مطلوب بدست نمی‌آید زیرا اپراتور =! نسبت به اپراتور = تقدم بالاتری دارد. بنابراین اگر دستور مزبور را بصورت :

while (ch = getchar( )!= '\n')

بنویسیم، اول عبارت getchar( )!= '\n' ارزیابی می‌گردد که یک عبارت 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.

مثال – برنامه زیر برنامه ساده‌ای است که کاراکترها را از طریق ورودی صفحه کلید دریافت می‌کند و آنها را به صفحه نمایش می‌فرستد. این نوع عمل را *منعکس کردن ورودی* نامند. برنامه مورد نظر هنگام دریافت کاراکتر # از ورودی، خاتمه می‌پذیرد.

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

```
main ( )
  {
    char ch ;
    while ((ch=getchar( )) != '#' )
       putchar(ch) ;
  }
```

- **توابع دیگر**

علاوه بر تابع getchar( ) دو تابع getch( ) و getche( ) نیز برای خواندن یک کاراکتر از ورودی بکار می‌رود .

تابع getche( )

اگر بخواهیم کاراکتری به کمک تابع scanf و یا تابع getchar خوانده شود، باید پس از تحریر (فشردن کلید) کاراکتر مورد نظر، کلید Enter نیز تحریر شود؛ یعنی درواقع دو تابع مزبور تا موقعی که کلید برگشت (که به آن carriage return یا به اختصار CR نیز گفته می‌شود) فشرده نشود، ورودی را در بافر نگه می‌دارد. پس از تحریر کلید برگشت یا CR ، دادهٔ تایپ شده در اختیار برنامه قرار می‌گیرد. حسن این روش آن است که اگر ما 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 تحریر کنیم. عیب این کار آن است که این عمل در محیط محاوره‌ای امروز وقت‌گیر و دردسرزا است؛ لذا 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. ایراد دیگر آن است که کاراکتر تحریر شده ، روی صفحه تصویر، نمایش داده می‌شود که این عمل را echoing نامند.

تابع getch( )

این تابع مانند تابع getche است با این تفاوت که کاراکتر تحریر شده دیگر در صفحه تصویر، ظاهر نمی‌گردد. درواقع حرف e در آخر تابع getche به مفهوم echo (عکس‌العمل) است .

در مورد هریک از این سه تابع وقتی که کنترل اجرای برنامه به این توابع می‌رسد، برنامه منتظر فشردن کلیدی (تحریر کاراکتری) در صفحه کلید می‌شود. اگر متغیر مورد نظر از نوع کاراکتری باشد (در برنامه با فرمت %c توصیف شده باشد) مقدار کاراکتری کلید فشرده شده به این متغیر، منتقل می‌شود (نسبت داده می‌شود) و درصورتی که این متغیر از نوع عددی باشد (در برنامه با فرمت %d توصیف شده باشد)، اسکی‌کد کاراکتر مربوط به کلید فشرده شده، در این متغیر قرار می‌گیرد.

توابع puts( ) , gets( )

دو تابع puts( ) , gets( ) این امکان را فراهم می‌سازند که بتوانید رشته‌هایی از کاراکترها را از طریق کنسول بخوانید و بنویسید (دستگاههای ورودی و خروجی استاندارد را کنسول نامند که در مورد

ریزکامپیوترها معمولاً صفحه کلید ورودی استاندارد و صفحه تصویر، خروجی استاندارد یا به‌عبارت دیگر کنسول را تشکیل می‌دهند).

تابع gets( ) یک رشته از کاراکترها را که از طریق صفحه کلید وارد می‌شود، می‌خواند و آنها را در آدرسی قرار می‌دهد که با آرگومانهای آن تعیین شده است که یک اشاره‌گر کاراکتر است. شما کاراکترهای رشته مورد نظر را تایپ می‌کنید و در پایان، کلید برگشت یا CR را تحریر می‌کنید. با این عمل به‌طور خودکار کاراکتر null یا '\0' نیز در پایان رشته قرار داده می‌شود. در اینجا اگر کاراکترهایی اشتباه تایپ شود، می‌توان آن را قبل از فشردن کلید Enter با استفاده از کلید backspace تصحیح کرد. درواقع 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 .

مثال – برنامه زیر رشته‌ای را (حداکثر به طول ۷۹ کاراکتر) از طریق صفحه کلید می‌خواند و در آرایه str قرار می‌دهد سپس آن را روی صفحه تصویر، نمایش می‌دهد :

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

فراخوانی تابع puts در مقایسه با همان فراخوانی printf دارای هزینه بالا سری (overhead) کمتری است و درنتیجه سریعتر از آن عمل می‌کند؛ زیرا تابع puts فقط می‌تواند یک رشته از کاراکتر را به خروجی بفرستد و نمی‌تواند مشابه printf تبدیل فرمت انجام دهد. همچنین نمی‌تواند مقادیر عددی را به‌عنوان 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. کاربرد اساسی این تابع در مبحث رشته‌ها دوباره به‌طور مفصل مورد بحث قرار خواهد گرفت .

جدول زیر خلاصه توابع ورودی خروجی مربوط به کنسول را نشان می‌دهد.

توابع i/o مربوط به کنسول

توابـــع	عمـــل (Operation)
getchar( )	یک کاراکتر از صفحه کلید می‌خواند و منتظر CR می‌ماند.
getche( )	یک کاراکتر از صفحه کلید می‌خواند، ولی منتظر CR نمی‌ماند.
getch( )	یک کاراکتر بدون انعکاس روی مونیتور، از صفحه کلید می‌خواند و منتظر CR نمی‌ماند.
putchar( )	یک کاراکتر روی صفحه مونیتور می‌نویسد.
gets( )	یک رشته را از صفحه کلید می‌خواند.
puts( )	یک رشته را روی صفحه تصویر می‌نویسید.
scanf( )	داده‌های ورودی (ترکیبی از مقادیر عددی، کاراکتر و رشته) را از ورودی استاندارد (صفحه کلید) به کامپیوتر وارد می‌کند.
printf( )	داده‌های خروجی (ترکیبی از مقادیر عددی، کاراکتر و رشته) را از کامپیوتر روی دستگاه خروجی استاندارد (صفحه مونیتور) می‌نویسد.

- **تمرین و پاسخ**

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

"Payam noor university"

```
#include < stdio . h>
main ( )
{
  printf ("Payam noor university \n") ;
  puts ("Payam noor university ") ;
}
```

تمرین ۲ - برنامه‌ای بنویسید که کاراکتری از ورودی خوانده و کاراکتر بعدی آنرا در خروجی چاپ کند .

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

تمرین ۳ - برنامه‌ای بنویسید که عدد صحیح m1 **وعدد اعشاری** m2 **، اطلاعات کاراکتری و آدرس متغیر** m3 **را در خروجی چاپ می‌کند .**

حل : برنامه مورد نظر در زیر نمایش داده شده است .

```
# 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 .

تمرین ٤ - خروجی برنامه 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) ;
}
```

خروجی برنامه

```
x1 = 2.000000e + 04
x2 = 2.000000E + 04
x3 = 2e + 04
```

توضیح – این برنامه شکلهای مختلف نمایی یا علمی را نمایش میدهد .

تمرین ٥ - خروجی برنامه زیر چیست ؟

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

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 بدلیل طول میدان کوچکتر یعنی ۲ گرد شده است .

تمرین ۶ – برنامه‌ای بنویسید که سه عدد صحیح را از ورودی خوانده ، و میانگین آنها را چاپ کند.

حل : چون می‌خواهیم حاصل تقسیم به‌صورت اعشاری باشد باید تبدیل نوع انجام شود .بنابراین قبل از تقسیم ، نوع اعشاری یعنی float را در داخل پرانتز قرار می‌دهیم (استفاده از cast). در این برنامه ، a , b , c سه متغیر صحیح هستند که 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) ;
}
```

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

حل : برنامه مورد نظر در زیر نمایش داده شده است .

```
# 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 جابجا کردن محتویات دو متغیر بدون استفاده از متغیر سوم ، آن دو متغیر را جمع کرده ، در یکی از آنها قرار می‌دهیم . سپس با عمل تفریق ، این جابجایی 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 .

تمرین ۸ – برنامه‌ای بنویسید که سن شما را برحسب روز از ورودی خوانده ، مشخص کند که چند سال ، چند ماه ، چند هفته و چند روز است .

حل : برنامه مورد نظر در زیر نمایش داده شده است .

```
# include < stdio . h>
main ( )
{
  int  days , years , months , weeks ;
  printf (" Enter your age in days :  ") ;
  scanf ("%d" , &days) ;
  years = days / ۳۶۵ ;
  days % = ۳۶۵ ;
  months = days /۳۰ ;
  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 باقیمانده تقسیم استفاده شده است . برای به‌دست آوردن سال ، باید تعداد روزها را بر ۳۶۵ تقسیم کرد . برای به‌دست آوردن روزهای باقیمانده (پس از تبدیل به سال)، باید باقیمانده تعداد کل روزها را نسبت به ۳۶۵ به‌دست آورد . این روند برای محاسبه تعداد ماه ، هفته و روز نیز بکار 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 .

فصل پنجم- عبارت ، دستور، عملگر

- **عبارات (Expressions)**

عبارت در یک زبان برنامه‌نویسی ، مجموعه معنی‌دار از مقادیر عددی ، متغیرها و یا بطور کلی اقلام داده‌ها است که با استفاده از علائم یا عملگرهای محاسباتی ، قیاسی و منطقی با یکدیگر ترکیب شده‌اند و می‌توان آنها را به صورتهای زیر دسته‌بندی کرد :

عبارات محاسباتی : ترکیبی از مقادیر ثابت ، متغیرهای صحیح و اعشاری با استفاده از مجموعه عملگرهای محاسباتی است که تحت قاعده خاصی تشکیل می‌گردد ، مانند مثالهای زیر :

(1) a*x + b

(2) a / (b + c)

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

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

در زیر مثالهایی از عبارات ریاضی و معادل آنها در زبان C ارائه شده است :

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

عبارات قیاسی : عبارات قیاسی ترکیبی از عبارات محاسباتی با استفاده از عملگرهای قیاسی با رعایت قوانین مربوط به نحوه بکار بردن عملگرها است (بعد در همین فصل همه عملگرها به‌طور مشروح بیان خواهد شد .)

نتیجه حاصل از اجرای این عبارت قیاسی همیشه درست و نادرست است . یعنی اگر شرط یا شرطهای بکار رفته در عبارت قیاسی برقرار باشد ، نتیجه عبارت مزبور درست است وگرنه نادرست

می‌باشد ، که در اغلب زبانها (مانند پاسکال) آنها را به‌ترتیب true یا false نامند ؛ ولی در زبان C مقدار true برابر یک و مقدار false برابر صفر می‌باشد (بعد در این زمینه توضیح بیشتر داده خواهد شد) .

بطوری که بعد ملاحظه خواهید کرد در عبارات قیاسی حق تقدم اجرا ، اول با عبارات محاسباتی می‌باشد و سپس عمل مقایسه مورد نظر انجام می‌گیرد . مثالهای زیر این مطلب را روشنتر می‌کند :

عبارت ریاضی	معادل آن در زبان C
$a > 5$	a > 5
$a + b - c \leq 3.14$	a+ b - c <= 3.14
$3a + b \neq 25$	3*a + b != 25

عبارات منطقی : عبارت منطقی مجموعه‌ای از عبارات محاسباتی و قیاسی است که در آن حداقل یک عملگر منطقی نیز بکار رفته است . معمولاً این گونه عبارات از دو گروه قبلی پیچیده‌تر می‌باشند .

در اغلب زبانها ، عبارات قیاسی که در طرفین یک عملگر منطقی قرار می‌گیرند ، باید در داخل پرانتز محصور شوند .

مثالهای زیر نحوه استفاده از عبارات منطقی را مشخص‌تر می‌کند :

عبارت ریاضی	معادل آن 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)

- **دستورها (Statements)**

دستور ، حکمی است که سبب می‌شود کامپیوتر ، عملی انجام دهد و بطور متعارف بر دو نوع تقسیم می‌گردد :

دستورهای ساده و دستورهای ساختیافته .

دستورهای ساده : دستورهای ساده ، دستورهای غیرشرطی هستند که متداول‌ترین آنها دستورهای زیر می‌باشند :

- جایگزین کردن مقداری معین به یک متغیر که به آن دستور انتساب یا assignment statement نیز می‌گویند .
- خواندن و نوشتن
- فراخوانی یک تابع
- انتقال کنترل به نقطه‌ای از برنامه (go to statement)

به نمونه‌هایی از این گونه دستورها که با آنها آشنا شدید ، توجه نمایید :

```
area = length * width ;
scanf("%d%d ", &a , &b) ;
printf("%d%d",c , d) ;
goto  a ;
fact(m) ;
```

دستورهای ساختیافته : دستورهای ساختیافته ، دستورهایی هستند که از انواع ساختارهای الگوریتمی ساخته شده‌اند که متداول‌ترین آنها عبارتند از :

- دستور مرکب (compound statement) که شامل دو یا چند دستور متوالی است که در داخل یک زوج آکولاد محصور شده‌اند . توجه دارید که در زبان C هر دستور ساده به یک سمی‌کولون ختم می‌شود .
- دستور حلقه تکرار (repetitive statement)
- دستور شرطی (conditional statement)

نمونه‌ای از دستور مرکب عبارت است از :

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

نمونه‌ای از دستور حلقه عبارت است از :

```
for ( i =1 ; i<=100 ; + + i )
  sum = sum + i ;
```

نمونه‌ای از دستور شرطی عبارت است از :

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

- **عملگرهای محاسباتی (Arithmetic Operators)**

عملگرها یا اپراتورها (operators) ، نشانه‌ها یا علائمی هستند که در عبارات بکار می‌روند و به کمک آنها می‌توان اعمالی را روی انواع داده انجام داد .فهرست عملگرهای محاسباتی در جدول زیر آمده است :

عملگرهای محاسباتی

نام عملگر	نشانه (symbol)	فرم	نوع عمل
جمع	+	a + b	a به‌علاوه b
تفریق	-	a - b	a منهای b

منهای یکانی	-	-a	منهای a (مقدار a با علامت نفی)
به‌علاوه یکانی	+	+a	مقدار عملوند a
ضرب	*	a * b	a ضرب در b
تقسیم	/	a / b	a تقسیم بر b
باقیمانده (remainder)	%	a % b	باقیمانده تقسیم a بر b
یک واحد افزایش (increment)	++	a++ , ++a	افزایش یک واحد به مقدار a
یک واحد کاهش (decrement)	--	a-- , --a	کاهش یک واحد از مقدار a

چهار عملگر + , - , * , / مشابه همان است که در سایر زبانهای برنامه‌نویسی نیز وجود دارد و می‌تواند تقریباً روی همه نوع داده‌های توکار یا built in یا نوع داده‌های استاندارد موجود در زبان C بکار برده شود . اگر عملگر "/" روی مقادیر صحیح یا کاراکتر بکار برده شود ، جزءِ اعشار برش داده می‌شود . مثلاً مقدار 10/3 برابر 3 خواهد بود یعنی فقط جزءِ صحیح آن درنظر گرفته خواهد شد .

عملگر % مشابه سایر زبانها عمل می‌کند و هر دو عملوند آن باید مقدار صحیح باشد و این عملگر ، باقیمانده تقسیم را بدست می‌دهد. مثلاً مقدار 10%3 برابر یک خواهد بود . یعنی باقیمانده تقسیم 10 بر 3 مساوی یک می‌باشد .

دو عملگر ++ و -- به‌طور متعارف در سایر زبانهای برنامه‌نویسی وجود ندارند . عملگر ++ یک واحد به عملوند خود اضافه می‌کند و عملگر -- یک واحد از آن کم می‌کند و هر دو ، عملگر تک‌عملوندی می‌باشند . درواقع دو دستور :

a++ ; و ++a ;

معادل :

a = a + 1 ;

و همینطور دو دستور :

a– – ; و – –a ;

معادل :

a = a – 1 ;

می‌باشد .

ولی به هرحال سرعت عمل دو عملگر ++ و -- از سرعت عمل عملگر انتساب (یعنی عملگر =) بالاتر است .

در بعضی کاربردها تفاوتی بین ++a ; و a++ ; وجود دارد که با مثال زیر این تفاوت مشخص می‌گردد :

مثال - به خروجی قطعه برنامه زیر توجه کنید :

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

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

خروجی برنامه

```
5    ۵
     ۶
```

مثال - به خروجی قطعه برنامه زیر توجه کنید :

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

خروجی

برنامه

```
6  5
   6
```

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 می‌باشد ، چاپ می‌گردد . در مثال دوم با اجرای دستور printf اولی مقدار a (که مساوی 5 است) چاپ می‌شود و سپس دستور ++a اجرا می‌گردد؛ یعنی به a یک واحد افزوده می‌شود بعد مقدار آن که برابر 6 شده است ، چاپ می‌گردد .

دستور printf دوم در اینجا نیز مشابه مثال اول عمل می‌کند .

روش عملکرد در مورد a-- و --a نیز به طریق مشابه می‌باشد .

ترتیب تقدم این گروه از عملگرها به‌صورت زیر است :

ترتیب تقدم عملگرها
-- ++ بالاترین تقدم
- منهای تک‌عملوندی (unary minus)
* / %
- + پایینترین تقدم

که در مورد عملگرهای هم‌تقدم نیز ترتیب تقدم از چپ به راست می‌باشد . البته درصورت وجود پرانتز ، تقدم آن از تقدم همه عملگرها بالاتر است .

مثالهای زیر روش عملکرد عملگرهای محاسباتی را نشان می‌دهد .

مثال – فرض کنید که a و b متغیرهایی از نوع صحیح بوده ، مقدار آنها به ترتیب برابر 10 و 3 باشد . همچنین فرض کنید c و d متغیرهایی از نوع اعشاری بوده و مقدار آنها به ترتیب برابر 12.5 و 2.0 باشد . حال چندین عبارت محاسباتی با مقادیر آنها در زیر نشان داده شده است :

عبارت	مقدار	عبارت	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		

مثال – فرض کنید که c1 و c2 متغیرهایی از نوع کاراکتر می‌باشند که به‌ترتیب معرف کاراکترهای A و E هستند .

حال چندین عبارت محاسباتی با مقادیر آنها در زیر نشان داده شده است . لازم به یادآوری است که در عبارات مورد نظر هر کجا c1 و c2 بکار رفته ، به جای آنها اسکی‌کد معرف کاراکتر مربوط به آنها بکار برده شده است ؛ یعنی در مورد متغیر c1 که معرف کاراکتر A می‌باشد عدد 65 (آسکی‌کد حرف A) و در مورد متغیر c2 نیز که معرف کاراکتر E می‌باشد عدد 69 (آسکی‌کد حرف E) بکار برده شده است . همچنین توجه داشته باشید که عدد 5 با نماد 5 که معرف یک کاراکتر محسوب می‌گردد و آسکی‌کد آن 53 می‌باشد ، متفاوت است .

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

یادآوری – تفسیرنحوه عمل عملگر باقیمانده ، وقتی که یکی از عملوندهای آن منفی باشد کاملاً مشخص نیست . ولی به هرحال اغلب گونه‌های c ، علامت عملوند اول را برای علامت باقیمانده بکار می‌برد . بنابراین دستورهایی مانند :

```
a = (a/b * b) + (a % b) ;
```

بدون توجه به علامت a و b همیشه درست عمل خواهد کرد .

مثال – فرض کنید که a و b متغیرهایی از نوع صحیح باشند و مقدار آنها به‌ترتیب برابر 11 و -3

باشد . حال چند عبارت محاسباتی با مقادیر آنها در زیر نشان داده شده است :

عبارت	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 بود ، مقدار a / b باز هم برابر 3- می‌شد ، 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 یکانی (Unary Operators)**

زبان C ، دارای یک مجموعه از عملگرها است که فقط روی یک عملوند عمل می‌کنند که آنها را عملگرهای یکانی یا تک‌عملوند می‌نامند و اغلب آنها 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 که جزءِ عملگرهای محاسباتی دسته‌بندی شد ؛ مانند مثالهای زیر :

```
–125        – 3.14        – 0.25        –5e–4
–a          – (x+y)       –3 * (a+b)
```

اگر جلوی مقادیر بالا علامتی قرار نگیرد ، علامت آنها مثبت درنظر گرفته می‌شود ، ولی به هرحال می‌توان جلوی آنها یک علامت "+" قرار داد که در این صورت "+" را که فقط دارای یک عملوند است ، به‌علاوه یکانی یا unary plus نامند . مانند مثالهای زیر :

```
+125        +3.14         +0.25         +5e – 4
+a          +(x+y)        +3 * (a+b)
```

بدیهی است اگر عملگر مزبور بکار برده نمی‌شد ، مقدار اقلام بالا باز هم تغییر پیدا نمی‌کرد و به همین دلیل کاربرد این عملگر متداول نیست .

از عملگرهای تک‌عملوندی دیگر دو عملگر ++ و -- است که جزءِ عملگرهای محاسباتی بیان شد .

از عملگرهای تک‌عملوندی دیگر عملگر "cast" است که آن را عملگر نوع یا type نیز گویند که در فصل مربوط به معرفی انواع داده‌ها در زبان C مورد بحث قرار گرفت .

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

unsignded int , long , int , float , unsigned

و مشابه آنها ، که در این صورت عملگر مزبور بزرگی این نوع ساختمان داده‌های استاندارد را برمی‌گرداند ، و یا اینکه عملوند آن یک عبارت یا expression است که در این صورت بزرگی آن عبارت را برحسب بایت برمی‌گرداند . جدول زیر نقش و فرم کاربرد این عملگر را نشان می‌دهد .

جدول عملگر sizeof

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

مثال – به دستورات زیر توجه فرمائید :

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

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 . مثال آخری 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 .

مثال – درنتیجه اجرای دو دستور زیر :

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

مقادیر ٤٤ نمایش داده خواهد شد ، که 4 اول معرف طول متغیر a است (که از نوع float می‌باشد) و 4 دوم معرف طول نوع داده float می‌باشد . 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 می‌باشد که جزء عملگرهای منطقی توضیح داده خواهد شد .

یادآوری – در مقابل عملگرهای تک‌عملوندی ، عملگرهای باینری یا دوعملوندی وجود دارد که دارای دو عملوند هستند . مانند عملگرهای محاسباتی (+ - * / %) .

- **عملگرهای رابطه‌ای (مقایسه‌ای)**

عملگرهای رابطه‌ای همانطور که از نامشان پیداست ، رابطه بین دو مقدار را تعیین می‌کنند . این عملگرها در جدول زیر نشان داده شده است .

عملگرهای رابطه‌ای

نام عملگر	نشانه (symbol)	فرم	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 و کلیدی در مورد عملگرهای رابطه‌ای (و همینطور عملگرهای منطقی که بعد بررسی خواهند شد) در مفهوم مقدار true و false نهفته است . در زبان C ، true هر مقدار غیر از صفر و false مقدار صفر است .

عباراتی که عملگرهای رابطه‌ای یا منطقی را بکار 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 . همینطور عبارت :

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

بصورت زیر ارزیابی خواهد شد :

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

ترتیب تقدم بین خود عملگرهای رابطه‌ای بصورت جدول زیر است :

ترتیب تقدم عملگرهای رابطه‌ای

بالاترین تقدم	> >= < <=
پایین ترین	= = !=

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

بطوری که بیان شد ، عبارات رابطه‌ای و منطقی ، نتیجه 0 یا 1 ایجاد می‌کنند . بنابراین قطعه برنامه زیر درست است و مقدار ۱ را روی صفحه نمایش ، نشان خواهد داد :

```
int  a ;
a = 100 ;
printf (" %d " , a>15 ) ;
```

مثال – فرض کنید که I ، J و K متغیرهایی از نوع صحیح هستند و مقدار آنها به‌ترتیب 1 و 2 و 3 می‌باشد . با فرض بالا چندین عبارت مقایسه‌ای (که عبارت منطقی نیز گفته می‌شود) همراه با تفسیر و نتیجه نهایی یا مقدار آنها در زیر نشان داده شده است :

عبارت	تفسیر	مقدار
I < J	true	1
(I+J) >= k	true	1
(J+k) > (k+5)	false	0
K != 3	false	0
J = = 2	true	1

یادآوری : اگر در مثالهای بالا پرانتز نیز بکار نمی‌رفت ، درنتیجهٔ بدست آمده تغییری حاصل نمی‌شد . پرانتز فقط گویایی برنامه را برای کاربر واضح‌تر نشان می‌دهد .

مثال – فرض کنید که i متغیری از نوع صحیح با مقدار 7 و f متغیری از نوع اعشار با مقدار 5.5 و C متغیری از نوع کاراکتر‘w’ باشد که آسکی‌کد آن برابر 119 است . با این مفروضات ، چندین عبارت مقایسه‌ای همراه با تفسیر و نتیجه نهایی یا مقدار آنها در زیر نشان داده شده است :

عبارت	تفسیر	مقدار
f > 5	true	1
(i + f) <= 10	false	0
c = =119	true	1
c != ‘p‘	true	1
c >= 10 * (i = f)	false	0

جدول زیر مثالهایی را از نحوه عملکرد و برخورد کامپایلر با عبارات مقایسه‌ای پیچیده نشان می‌دهد .(به فرض اینکه متغیرهایی بصورت زیر تعریف شده‌اند .)

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

مثالهایی از عبارات مقایسه‌ای

عبارت	عبارات معادل آن	نتیجه
j > m	j > m	0
m / n	(m/n) < x	1
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 انتساب (Assignment Operators)**

در زبان C ، سمبول یا علامت ‘=’ به مفهوم مساوی نیست ، بلکه یک عملگر جایگذاری یا عملگر انتساب است ، که از قبل با آن آشنا هستید . 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 که در سمت چپ علامت قرار دارد ، نام یک شناسه یا متغیر و آنچه که در سمت راست آن قرار دارد ، مقدار یا value متغیر مزبور است . پس دستور بالا به مفهوم :

"k مساوی 123 است"

نمی‌باشد ، بلکه به مفهوم :

"مقدار 123 را به k اختصاص بده"

می‌باشد . پس باید به تمایز بین نام متغیر و مقدار متغیر توجه کرد . حال دستور متعارف زیر را درنظر بگیرید :

```
K = K+1 ;
```

از نظر ریاضی این دستور دارای مفهوم نیست . اما از دیدگاه برنامه‌سازی ، دستور مزبور یک دستور جایگذاری می‌باشد و بدین مفهوم است که : « متغیری را که نام آن K است پیدا کنید ، سپس به مقدار آن یک واحد اضافه کنید و پس از آن مقدار جدید را به متغیری که نام آن K است ، (درواقع به همان متغیر) اختصاص یا نسبت دهید .

بطور کلی در زبان C ، چندین عملگر مختلف انتساب یا جایگذاری وجود دارد که همه آنها برای تشکیل یک عبارت انتساب یا عبارت جایگذاری بکار برده می‌شود که مقدار یک عبارت را به یک شناسه یا متغیر ، اختصاص یا نسبت می‌دهد . متداول‌ترین عملگر انتساب ، عملگر ‘=‘ است . فرم کلی دستور انتساب به‌صورت زیر است :

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

به هرحال باید توجه داشته باشید که عملگر انتساب ، یعنی ‘=‘ کاملاً با عملگر مساوی که علامت ‘= =‘ می‌باشد ، متفاوت است .

بطوری که در بالا بیان شد ، عملگر انتساب برای اختصاص و نسبت دادن یک مقدار به یک شناسه یا متغیر بکار برده می‌شود ، درحالی که عملگر تساوی یا برابری ، برای تعیین اینکه آیا دو عبارت دارای مقدار یکسان هستند یا نه ، بکار می‌رود . پس این دو عملگر نمی‌توانند به جای یکدیگر بکار برده شوند .

اگر دو عملوند عملگر انتساب از نظر نوع ، یکسان نباشند ، مقدار عبارت یا عملوند طرف راست بطور خودکار به نوع شناسه یا متغیر طرف چپ عملگر تغییر می‌یابد . بنابراین اگر نتیجه عبارت سمت راست عملگر مزبور از نوع float و عملوند سمت چپ آن از نوع int باشد ، جزءِ اعشاری آن حذف خواهد شد .

مثال – فرض کنید که a و b متغیرهایی از نوع int هستند و مقدار b برابر 5 باشد . با این مفروضات ، به نتیجه بعضی عباراتی که در جدول آمده است ، توجه نمایید :

مثالهایی از دستورهای انتساب

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 می‌گردد که در عبارات بالا در مورد کاراکترها ، آسکی‌کد آنها جایگزین می‌شود . مثلاً در آخرین عبارت ، به جای E و B به ترتیب مقادیر 69 و 66 قرار می‌گیرد و نتیجهٔ آن برابر :

69 - 66/3 = 69-22 = 47

خواهد بود. همچنین توجه کنید که در عبارت :

a = 2 * (b/2) ;

به لحاظ اینکه پرانتز دارای تقدم بالاتری است ، اول b/2 محاسبه می‌گردد که نتیجهٔ آن برابر 2 خواهد شد و سپس نتیجهٔ حاصل در 2 ضرب می‌شود که بنابراین نتیجهٔ نهایی برابر 4 خواهد شد .

در زبان C ، می‌توان دستورهای انتساب چندگانه بکار برد مانند :

a = b = 5 ;

فرم کلی این گونه دستورهای انتساب بصورت زیر است :

V1 = V2 = ... = V*n* = expression ;

و در چنین حالتی ، تقدم عمل انتساب از راست به چپ می‌باشد ؛ بنابراین دستور :

a = b = c = 5 ;

معادل : a = (b = (c = 5)) ; می‌باشد .

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

+ = , -= , *= , /= , %=

را بدست آورد . می‌توان آنها را عملگرهای محاسباتی جایگذاری یا arithmetic assignment operators نامید که چکیدهٔ آنها در جدول نشان داده شده است .

عملگرهای محاسباتی جایگذاری

عملگر	نام	مثال	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 باقیمانده تقسیم	a %b ;	a = a%b ;

مثال – اگر متغیرهای a و b از نوع int باشند و مقدار آنها به ترتیب برابر 10 و 15 باشد ، دستور :

a += b ;

معادل دستور :

a = a + b ;

می‌باشد که درنتیجه مقدار a برابر 25 خواهد شد .

- **عملگرهای منطقی (Logical Operators)**

عملگرهای منطقی بطور متعارف برروی عملوندها یا عبارات منطقی که دارای دو ارزش درست یا true و نادرست یا false هستند ، عمل می‌کنند . جدول زیر عملگرهای منطقی را نشان می‌دهد .

عملگرهای منطقی

عملگر	علامت مفعول	فرم	نتیجه
و (AND) منطقی	&&	a&&b	اگر a و b هر دو برابر یک یا غیرصفر (true) باشند ، نتیجه برابر یک ، در غیراینصورت برابر صفر خواهد بود .
یا (OR) منطقی	||	a || b	اگر b یا a یا هر دو غیرصفر باشند نتیجه برابر با یک ، در غیراینصورت 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 .

بطوری که در گذشته بیان شد ، در زبان C ، ارزش نادرستی یا false با مقدار صفر و ارزش درستی یا true با مقادیر غیرصفر مشخص می‌گردد .

درضمن یادآوری می‌شود که در بین عملگرهای منطقی ، عملگر"!" بالاترین تقدم و عملگرد"||" پایینترین تقدم را دارد .

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 به‌صورت ترکیبی با عملگرهای رابطه‌ای بکار برده می‌شوند . درواقع 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 بکار می‌روند .

مثال – جدول زیر مثالهایی از عبارات منطقی را همراه با معادل هر عبارت ، به اختصار نشان می‌دهد :

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

عبارت	عبارت معادل آن	نتیجه
j && m	(j) && (m)	0
j<m && n<m	(j<m) && (n<m)	1
m+n || !j	(m+n) || (!j)	1

1	((x*5)&&5) || (m/n)	X*5 && 5 || m/n
1	((j<=10)&&(x>=1))&&m	J<=10 && x>=1 &&m
0	((!x) || (!n)) || (m+n)	!x || !n || m+n
1	((x*y)<(j+m)) || n	x *y<j+m || n
2	((x>y)+(!j)) || (n+ +)	(x>y)+!j || n++
2	(j || m) + (x || (+ +n))	(j || m) + (x || ++n)

معمولاً یک عبارت رابطه‌ای پیچیده ، بعنوان قسمت شرطی یک دستور حلقه یا در یک دستور if بکار برده می‌شود . 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 مکانیسم 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 بدست آید ، باید آن را بصورت زیر نوشت :

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

- **عملگر شرطی (:?)**

عملگر شرطی (condtional operator) تنها عملگری است که دارای سه عملوند می‌باشد که فرم آن در جدول زیر نشان داده شده است .

عملگر	علامت	فرم	عمل
شرطی (conditional)	?:	a ? b : c	اگر a غیرصفر باشد ، نتیجه b و گرنه نتیجه c است .

درواقع عملگر شرطی ، فرم کوتاهی برای دستور متداول if...else است ؛ برای مثال عبارت :

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

را می‌توان بصورت زیر نوشت :

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

اولین عملوند ، تست شرط است که باید از نوع اسکالر باشد . عملوندهای دوم و سوم نتیجه نهایی عبارت را نشان می‌دهند که برحسب مقدار عملوند اول ، فقط یکی از آن دو انتخاب می‌گردد . عملوندهای دوم و سوم می‌توانند 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 .)

در واقع فرم کلی عبارت شرطی بصورت زیر است :

```
exp1 ? exp2 : exp3
```

که در آن ابتدا عبارت اول (exp1) ارزیابی می‌شود ، اگر نتیجه آن درست یا true (یعنی مقدار آن غیرصفر) باشد ، عبارت دوم (exp2) ارزیابی می‌شود و مقدار عبارت شرطی ، برابر نتیجه 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 نتیجه ارزیابی عبارت سوم خواهد بود .

مثال – 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 خواهد یافت . مانند مثال زیر .

مثال – به عبارت شرطی زیر توجه کنید :

```
flag = (i<0) ? 0 : 100 ;
```

در اینجا اگر متغیر i منفی باشد ، به متغیر flag مقدار صفر نسبت داده خواهد شد . وگرنه به متغیر flag مقدار 100 اختصاص خواهد یافت .

به هرحال این عملگر ، خوانایی و گویایی برنامه را اغلب کاهش می‌دهد ، لذا باید در کاربرد آن دقت شود .

- **عملگر کاما (Comma Operator)**

عملگر کاما "," (که در گذشته نیز در مبحث دستورهای کنترلی مورد بحث قرار گرفت) اجازه می‌دهد که چندین عمل در یک دستور انجام گیرد . نحوهٔ عملکرد و فرم این عملگر در جدول زیر نشان داده شده است .

عملگر کاما

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

یک روش برای استفاده از عملگر کاما به‌صورت زیر است :

```
متغیر = (عبارت۱ , عبارت 2) ;
```

که در آن عملگر کاما موجب می‌گردد که اول "عبارت 1" ارزیابی شده و سپس "عبارت 2" ارزیابی شود و نتیجه ارزیابی "عبارت 2" به متغیر مورد نظر نسبت داده شود . در این گونه موارد معمولاً "عبارت 1" و "عبارت 2" با یکدیگر مرتبط هستند .

مثال – عبارت مقابل 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 نتیجه آن برابر 20 خواهد بود ؛ که در پایان ، این مقدار به متغیر a نسبت داده می‌شود ؛ بنابراین پس از اجرای دستور مزبور مقدار a برابر 20 خواهد شد .

کاربرد دیگر از عملگر کاما ، در رابطه با دستور for است که این نوع کاربرد آن بیشتر متداول است و در گذشته نیز در مبحث دستورهای کنترلی ، نمونه آن را ملاحظه کردید .

- **عملگرهای حافظه (Memory Operators)**

چندین عملگر وجود دارد که به شما اجازه می‌دهند که به محلهای حافظه و محتوای آنها دستیابی داشته باشید . با بعضی از این عملگرها ، مانند عملگر آدرس (یعنی &) آشنا هستید . این گونه عملگرها در جدول زیر آورده شده‌اند .

عملگرهای حافظه

نام عملگر	علامت	فرم	عمل
آدرس	&	&a	آدرس متغیر a را در اختیار قرار می‌دهد .
محتوا	*	*a	مقدار شیء یا داده‌ای را که آدرس آن در a ذخیره شده است ، در اختیار قرار می‌دهد .
عناصر آرایه	[ ]	a[5]	مقدار عنصر پنجم آرایه را در اختیار قرار می‌دهد .
نقطه (dot)	.	a . b	مقدار عنصر b از عضو ساختار a را در اختیار قرار می‌دهد.
پیکان راست (Right Arrow)	->	p-> a	متغیر اشاره‌گر p قرار دارد (یعنی یا p به آن اشاره شده است) در اختیار قرار می‌دهد .

- **تقدم عملگرها**

جدول زیر تقدم عملگرها را به اختصار نشان می‌دهد و در داخل هر دسته از عملگرها که تقدم یکسان دارند ، برحسب اینکه تقدم آنها از چپ به راست یا به اختصار L→R و یا از راست به چپ یا به اختصار R→L باشد نمایش می‌دهد .

تقدم عملگرها به اختصار

Operators	Precedence Group	Aassocitivity
( ) [] →	function , array structure member , pointer to structure member	L→R
- ++ -- ! ~ * & sizeof (type)	unary operators	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

L→R	logical or	| |
R→L	conditional operator	? :
R→L	assignment operators	= += -= *= /= %= &= ^ = |= <<= >>=
L→R	comma operator	,

- **تمرین و پاسخ**

تمرین ۱ـ برنامه‌ای بنویسید که عددی را خوانده و قدرمطلق آن را چاپ کند .

```
# include < stdio . h>
main ( )
{
float  x ;
scanf ("%f" , &x) ;
if (x<0)  x = -x ;
  printf ("%f" , x) ;
}
```

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

$$C = 5/9 * (F - 32)$$

حل : برنامه مورد نظر در زیر نمایش داده شده است . اگر قبل از 5 عبارت (float) نوشته نشود آنگاه تقسیم به‌صورت صحیح انجام شده 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

```
۶۵ Enter farenheit  :
Celsius = 18 . 333334
```

تمرین ۳ـ اگر $b = 5$ **و** $a = 2$ **باشد حاصل عبارت** ; $y = a*b + ++a + 10$ **چیست ؟**

حل : y برابر 28 می‌شود .

درواقع دستور فوق معادل دو دستور زیر است :

```
+ + a ;                  → a = 3
y = a * b + a + 10 ; → y = 3 * 5 + 3 + 10 = 28
```

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

حل : برنامه مورد نظر در زیر نمایش داده شده است :

```
# 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 برنامه بالا با استفاده از دستور if این شرط بررسی میشود .

تمرین ٥ ـ دستورات زیر را بترتیب اجرا کرده و در هر قسمت معادل بیتی و محتوای متغیر x را مشخص سازید .

دستورات	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

نکته – 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 .

نکته – 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 .

نکته – xor هر عدد با خودش (توسط عملگر ^) برابرصفر می‌شود .

نکته – اپراتورهای 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 .

نکته – 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 غلط غیرصفر یعنی درست) .

تمرین ۶ ـ خروجی قطعه برنامهٔ زیر چیست ؟

```
int  x = 7 , y = 8 ;
printf ("%d" , x && y) ;   → 1
printf ("%d" , x & y) ;    → 0
```

حل : در دستور چاپ اول 7 && 8 عدد 8 درست و 7 درست تعبیر می‌شود و با توجه به عملگر منطقی && ترکیب دو مقدار درست True یا ۱ 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        → 0 عدد
```

فصل ششم - توابع (به انضمام کلاسهای حافظه)

- **مقدمه**

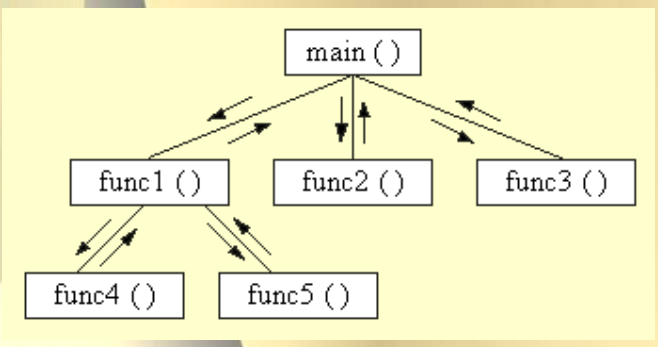
بطور کلی روش مناسب برای حل یک مسئله بزرگ آن است که مسئله را به عناصر یا واحدهای کوچک قابل کنترل تجزیه کرده ، برای هر واحد ، یک برنامه بنویسیم . سپس آنها را با یکدیگر ترکیب کرده ، بصورت یکپارچه بکار بریم . این روش ، برنامه‌سازی از بالا به پایین یا top down programming نامیده می‌شود . در اغلب برنامه‌های بزرگ واقعی برای سادگی کار ، از این روش استفاده میشود .

در حالت کلی و بدون توجه به زبان برنامه‌سازی خاص ، برنامهٔ مورد نظر برای پیاده‌سازی هر واحد یا ماژول را ، زیربرنامه و یا زیرروال گویند . در زبان پاسکال به آنها تابع یا رویه گفته میشود. پیمانه یا ماژول ، در زبان C ، تابع نامیده میشود . یعنی برای پیاده‌سازی هر ماژول می‌توان یک تابع نوشت . حال اگر یک برنامه بزرگ را به واحدهای کوچکتری به نام پیمانه یا ماژول تجزیه کنیم و برای هر ماژول یک تابع بنویسیم و سپس آنها را با یکدیگر ترکیب کرده ، در یک برنامه یا تابع اصلی بکار ببریم ، چنین شیوه‌ای را برنامه‌سازی پیمانه‌ای یا modular programing نامند.

بطوری که بیان شد ، هر برنامه در زبان C ، مجموعه‌ای از یک یا چندین تابع است که یکی و فقط یکی از آنها ، به نام تابع اصلی است که نام آن main می‌باشد و بقیه تابع فرعی هستند . توابع ، به دو گروه دسته‌بندی می‌شوند ؛ یک سری توابعی هستند که از پیش تعریف شده‌اند و آنها را تابع آماده یا توکار یا built in functions و یا توابع کتابخانه‌ای library functions نامند . کتابخانه C استاندارد ، مجموعه‌ای غنی از این گونه توابع دارد که برای محاسبات ریاضی ، انجام عملیات روی نوشته‌ها و کاراکترها ، انجام عملیات ورودی و خروجی ، انجام عملیات در زمینه‌های گرافیکی و غیره می‌باشد ، که متداول‌ترین آنها مورد بررسی قرار خواهند گرفت . توربو C و همینطور گونه‌های دیگر C ، از این لحاظ دارای کتابخانه‌های گسترده‌تری می‌باشند که آنها نیز به اختصار مورد بررسی قرار خواهند گرفت . وجود این گونه توابع از پیش تعریف شده ، کار برنامه‌نویسان را ساده‌تر کرده ، توانمندی آنها را برای نوشتن برنامه‌های کارآمد ، بالا می‌برد .

گروه دیگر از توابع فرعی آنهایی هستند که توسط برنامه‌نویس تعریف می‌گردند . C ، مشابه سایر زبانهای برنامه‌سازی ، اجازه می‌دهد که برنامه‌نویس ، برحسب سلیقهٔ خود ، زیربرنامه‌هایی را

به‌عنوان تابع فرعی طراحی کند و آنها را برحسب نیاز در برنامه یا تابع اصلی و یا سایر توابع فراخواند . این شیوه ، به برنامه‌ساز امکان می‌دهد که یک برنامه بزرگ را بصورت پیمانه‌ای یا ماژولار طراحی نماید ؛ یعنی کار یا پروژهٔ مورد نظر را به قسمتهای کوچکتری تجزیه کرده ، برای هر قسمت یک تابع بنویسید و آنها را به کمک تابع اصلی به یکدیگر پیوند دهد و برحسب لزوم آنها را فراخواند . شکل زیرنمونه‌ای از این روش طراحی برنامه را نشان می‌دهد .



طراحی برنامه بصورت پیمانه‌ای یا ماژولار ، یعنی بصورت مجموعه‌ای از توابع ، مزایایی دارد که مهمترین آنها عبارتند از :

- ساده شدن کنترل و خطایابی
- انجام تغییرات در برنامه و اصلاح آن
- قابلیت استفاده مجدد از تابع
- امکان همکاری برنامه‌نویسان متعدد در نوشتن یک برنامه

در این فصل ایجاد و کاربرد توابع مورد بررسی قرار میگیرد .

• نحوهٔ تعریف تابع

بطوری که بیان شد ، هر برنامه C ، از یک یا چندین تابع تشکیل می‌گردد که یکی (و فقط یکی) از آنها به‌عنوان تابع اصلی می‌باشد . بطور کلی هر تابع شامل موارد زیر است :

- **عنوان تابع (Function Heading) :** که شامل نام تابع است و به دنبال آن در داخل یک زوج پرانتز ، نشانوندها یا آرگومانهای arguments تابع (در صورت وجود داشتن) با اعلان نوع آرگومان می‌آید . نام تابع نیز باید از لحاظ نوع یا type توصیف گردد . یعنی مشخص شود که تابع چه نوع مقداری را برمی‌گرداند .

- **یک بلوک یا جمله مرکب** : که آن را بدنه می‌نامند و شامل موارد زیر است :

الف) توصیف متغیرهای محلی (local variables declaration) ـ هر متغیری که جزءِ آرگومانهای تابع

نباشد ، یعنی مخصوص خود تابع باشد ، متغیر محلی نامیده می‌شود ، که باید در آغاز تابع ، یعنی پس از عنوان تابع ، اعلان گردد .

ب) سایر دستورهای تابع ـ این قسمت درواقع ، دستورهای اجرایی و متن تابع را تشکیل می‌دهد.

شکل زیر ، عناصر یک تابع را نمایش می‌دهد :

در مورد آرگومانهای تابع یا در همان داخل زوج پرانتز ، هر آرگومان جداگانه توصیف می‌گردد و یا اینکه در سطر جداگانه‌ای ، یکجا توصیف می‌گردد که هر دو روش با مثالهایی نشان داده خواهند شد .

وقتی که تابعی در داخل تابع اصلی فراخوانی می‌شود ، اصطلاح آرگومان بکار برده می‌شود . ولی در خود تابع فرعی ، یعنی در تعریف تابع فرعی ، اصطلاح پارامتر بکار برده می‌شود .

دستور return : خروج از تابع یا با دستور return انجام می‌گیرد و یا با رسیدن به انتهای تابع . در حالت دوم نیز متعارف است که دستور return را بکار برند ، گرچه الزامی نیست .

اگر نیاز باشد که تابع ، مقدار 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 فراخواننده آن) برمی‌گردد .

چنانچه 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 فراخواننده آن خواهد بود .

شکل زیرتابعی را نشان می‌دهد که مقداری را به برنامه فراخواننده آن برمی‌گرداند . در اینجا مقدار x که برابر ۳ می‌باشد به تابع main برگردانده می‌شود و در آنجا به متغیر a اختصاص می‌یابد:

تابع فرعی

```
func( )
....    {
x = 3 ;
rerum(x) ;
        }
```

تابع اصلی

```
main( )
....    {
a       =
func( ) ;
        }
```

محدودیت دستور return آن است که می‌تواند فقط یک مقدار را به تابع فراخواننده آن برگرداند . چنانچه نیاز باشد که بیش از یک مقدار به تابع فراخواننده بازگردانده شود ، باید شیوهٔ دیگری بکار برد که در فصل مربوط به اشاره‌گرها بررسی خواهد شد .

مثال – برنامه‌ای بنویسید که عدد صحیح n را بخواند ، سپس با فراخوانده‌شدن تابعی به نام fact ، فاکتوریل عدد مزبور محاسبه گردد و همراه با خود عدد در همان تابع روی صفحه تصویر نمایش داده شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

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

مثال – همان برنامه را به طریقی بنویسید که فاکتوریل عدد به توسط تابع فرعی برگردانده شده ، در تابع اصلی چاپ شود .

حل : برنامه مورد نظر در زیر نمایش داده شده است :

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

در مثال اول که تابع مقداری را برنمی‌گرداند ، اولاً تابع به‌عنوان void توصیف شده ؛ ثانیاً در فراخوانی تابع ، فقط نام تابع در یک دستور بکار رفته است . در مثال دوم که تابع باید مقداری را برگرداند ، اولاً نوع مقداری را که باید تابع برگرداند ، مشخص شده است که در این مثال از نوع int می‌باشد ؛ ثانیاً نام تابع در تابع اصلی ، در طرف راست یک دستور جایگذاری یا انتساب بکار رفته است ؛ ثالثاً در تابع فرعی نیز دستور return دارای آرگومان است و نقش دستور مزبور به این طریق است که اول مقدار آرگومان دستور return به نام تابع نسبت داده می‌شود ، سپس کنترل به تابع اصلی برمی‌گردد . در مثال اول نیازی به دستور return نیست . یعنی با رسیدن به انتهای تابع ، کنترل بطور خودکار به تابع فراخواننده (در اینجا تابع اصلی) 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 ، یعنی به فرم :

return ;

بکار برده می‌شود .

دستور return برخلاف مثال دوم بالا فقط در پایان تابع بکار برده نمی‌شود . بلکه هر کجا منطق برنامه ایجاب کند ، بکار می‌رود. بطور کلی ، هر کجا در درون یک تابع اجرای برنامه بطور منطقی به اجرای دستور return منجر شود ، کنترل عملیات به برنامه فراخواننده برمی‌گردد . مثال بعد این موضوع را نشان می‌دهد .

مثال – تابعی بنویسید که کاراکتری را از دستگاه ورودی استاندارد (صفحه کلید) بخواند و آن را اگر جزءِ حروف بزرگ باشد به حرف کوچک همنام آن برگرداند (مثلاً حرف A را به a تبدیل کند) ؛ در غیر این‌صورت خود کاراکتر دریافت شده را برگرداند .

حل : برنامه مورد نظر در زیرنشان داده شده است :

```
char  convert (void)
  {
    char  ch ;
    ch = getche( ) ;    /* read a character */
    if (ch > 64  &&  ch < 91)    /* if uppercase */
         return (ch + 32) ;      /* return converted value */
    else
         return (ch) ;
  }
```

می‌دانیم که آسکی‌کد حروف بزرگ بین ۶۵ تا ۹۰ است (۶۵ برای A و ۹۰ برای Z) و آسکی‌کد حروف کوچک نیز از حروف همنام خود ، ۳۲ واحد بیشتر است . پس در تابع بالا تست می‌شود که آیا آسکی‌کد حرف خوانده شده در فاصله باز ۶۵ تا ۹۱ است یا نه ؟ درصورت مثبت بودن پاسخ (یعنی از

حروف A تا Z بودن حرف خوانده شده) مقدار ۳۲ واحد به آسکی‌کد آن حرف افـزوده مـی‌گـردد تـا حرف مزبور به حرف کوچک همنام خود تبدیل شود و نتیجه به تابع فراخواننده آن برگردانده شود . در غیر این‌صورت همان حرف بدون تبدیل به تابع فراخواننده ، برگردانده می‌شود .

توابعی که مقادیر بازگشتی آنها عدد صحیح نیست

اگر نوع مقداری که تابع برمی‌گرداند ، بطور صریح مشخص نشده باشد ، پیش‌فرض آن است که تابع مزبور مقدار صحیح برمی‌گرداند . در مورد اغلب توابع C ، این پیش‌فرض قابل قبول است . به هرحال اگر نوع مقداری را که تابع برمی‌گرداند غیر از مقدار صحیح باشد ، باید دو عمل زیر انجام گیرد :

اول نوع تابع (یعنی نوع مقداری را که تابع برمی‌گرداند) بطور 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 به‌صورت سراسری یا عمومی قبل از همه توابعی که آن را فرا خواهند خواند (بطور متعارف قبل از تابع main) مشخص و توصیف گردد .

مثال زیر این موضوع را روشنتر می‌سازد :

مثال -

```
main ( )
 {
    float  sum (float , float) ;
    float  first , second ;
    first = 123.45 ;
    second = 25.15 ;
    printf ("%f , sum (first , sencond)) ;
  }
 float sum (a , b)
     {
       float  a , b ;
       return  a + b ;
      }
```

راه دیگر آن است که تابع مزبور ، قبل از تابع فراخواننده (بطور متعارف قبل از تابع main) توصیف گردد مشابه برنامه زیر :

```
float  sum ( )
main ( )
 {
    float  first , second ;
    first = 193.45 ;
    second = 25.15 ;
    printf ("%f , sum(first , second)) ;
  }
```

```
float  sum(a , b)
  {
    float  a , b ;
    return  a+b ;
  }
```

یادآوری : در بیشتر نسخه‌های زبان C ، نیازی نیست که نوع آرگومانها را در تعریف تابع ، تعیین کرد . مانند مثال بالا که در تعریف تابع sum قبل از تابع main ، آرگومانهای آن مشخص نشده است .

- **نحوهٔ فراخوانی تابع**

بطور کلی فراخوانی یک تابع با نوشتن نام و پارامترهای آن (در صورت داشتن پارامتر) بصورت یک تک دستور و یا در یک دستور جایگذاری ، انجام می‌شود . راه دیگر برای فراخوانی تابع آن است که نام تابع در یک دستور خروجی و یا یک عبارت محاسباتی بکار برده شود . بطور کلی اگر تابعی بصورت void توصیف شود ، می‌تواند بعنوان یک اپراند یا عملوند در هر عبارت قابل قبول در زبان C بکار برده شود . در مثال زیر نام تابع در یک دستور خروجی بکار رفته است .

مثال – محاسبه فاکتوریل عدد صحیح n :

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

- فراخوانی با مقدار یا فراخوانی توسط ارزش (call by value)
- فراخوانی توسط آدرس (call by address) یا فراخوانی توسط ارجاع (call by reference)

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 نسخه دریافت شده از آرگومانها انجام دهد ، در تابع فراخواننده آن منعکس نمی‌گردد . یعنی مقدار آن آرگومانها ، در تابع فراخواننده تغییر نخواهد کرد .

در مثال دوم ، تابع fact را درنظر بگیرید . تابع مزبور عدد صحیح n را بعنوان 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 کپی یا نسخه‌ای از همان مقدار را دارد . وقتی که تابع فرعی عملیاتی روی n انجام می‌دهد درواقع این عملیات در روی متغیر n در تابع فرعی و یا در روی محتوای حافظه‌ای که با نام n در تابع فرعی پیش‌بینی شده است ، انجام می‌گیرد و هیچ‌گونه تأثیری در مقدار n در تابع اصلی ندارد . به همین دلیل این گونه فراخوانی را فراخوانی با مقدار گویند . برای روشنتر شدن مطلب ، به مثال زیر توجه کنید و خروجی تابع اصلی و فرعی را در نتیجه اجرای دستورهای printf ملاحظه نمایید :

```
#include<stdio.h>
main ( )
{  int  a = 5 , b = 12 ;
   printf ("\n %d %d" , a , b) ;
   func (a , b) ;
   printf ("\n %d %d" , a , b) ;
}
void func (int a , int b)
{   printf ("\n%d %d" , a , b);
   a = a + 3 ;
   b = b - 3 ;
   printf ("\n %d %d" , a , b) ;
}
```

و خروجی برنامه مزبور بصورت زیر خواهد بود :

```
5       12
5       12
8       9
5       12
```

نحوه عملکرد برنامه مزبور به این طریق است که با اجرای اولین دستور printf در تابع اصلی ، مقادیر a و b در تابع اصلی که به ترتیب 5 و 12 می‌باشد ، چاپ می‌گردد (سطر اول خروجی). سپس با فراخوانی تابع فرعی یک نسخه از مقادیر a و b بعنوان آرگومان به تابع فرعی گذر داده می‌شود و با اجرای اولین دستور printf در تابع مزبور که آنها هم به ترتیب 5 و 12 می‌باشد ، بصورت سطر دوم خروجی بالا ، چاپ می‌گردد . حال با اجرای دو دستور :

```
a = a + 3 ;
```

```
b = b - 3 ;
```

در تابع مزبور ، به مقدار a سه واحد افزوده و از مقدار b ، سه واحد کاسته می‌شود . در واقع اکنون مقادیر جدید a و b در تابع فرعی به ترتیب 8 و 9 می‌گردد که با اجرای دستور printf دوم در آن تابع ، مقادیر جدید a و b در آن تابع بصورت 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 .

درواقع دو حافظه با نام a و دو حافظه با نام b وجود دارند که یکی در تابع اصلی و دیگری در تابع فرعی است . هر موقع در تابع اصلی به a یا b مراجعه شود ، مقادیر a و b (به عبارت دیگر محتوای خانه‌هایی از حافظه که به نام a و b هستند) در تابع اصلی چاپ می‌شود و هر موقع در تابع فرعی به a و b مراجعه شود ، محتوای حافظه‌هایی که در تابع فرعی برای مقادیر a و b درنظر گرفته شده است ، چاپ می‌گردد .

مطلب مهم دیگر آن است که آرگومانهایی که به تابع فرعی فرستاده می‌شوند ، متغیرهای مجازی یا متغیرهای مستعار نامیده می‌شوند که اسامی آنها در تابع فرعی می‌تواند غیر از آنچه باشد که در تابع اصلی نامگذاری شده است . بنابراین برنامه مزبور را می‌توان بصورت زیر نیز نوشت :

```
#include<stdio.h>
main ( )
{   int  a = 5 , b = 12 ;
    printf ("\n %d %d" , a , b) ;
    func (a , b) ;
    printf ("\n %d %d" , a , b) ;
}
void func (int c , int d)
{  printf ("\n%d %d" , c , d) ;
  c = c + 3 ;
  d = d - 3 ;
  printf ("\n %d %d" , c , d) ;
}
```

ملاحظه می‌گردد که در این روش فراخوانی تابع ، مقدار آرگومان ، به پارامتر تابع فراخوانده شده ، که آن را پارامتر رسمی یا فرمال پارامتر نیز گویند) کپی می‌شود . یادآور می‌شویم که در فراخوانی تابع ، مقادیر یا متغیرهای انتقالی ، در تابع فراخواننده ، آرگومان یا argument و در تابع فراخوانده شده ، پارامتر (parameter) نامیده می‌شود .

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 بکار 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 به لحاظ اینکه 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 بعنوان آرگومان به یک تابع ، استثنا بر فرم 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 یک اشاره‌گر به اولین عنصر در آرایه (یعنی آدرس اولین عنصر آرایه) به تابع گذر می‌کند (به خاطر داشته باشید که در C نام یک آرایه بدون زیرنویس یا index آن ، یک اشاره‌گر به اولین عنصر در آرایه است) .

سه روش برای توصیف یا تعریف پارامتری که اشاره‌گر آرایه را دریافت می‌کند ، وجود دارد که به شرح زیر بیان می‌گردد :

روش اول – پارامتر مورد نظر بصورت 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 display(num)
int  num[10] ;
{
  int  i ;
  for ( i=0 ; i<10 ; + +i )
     printf ("\n %d" , num[i] ) ;
}
```

در اینجا با اینکه پارامتر num بصورت یک آرایه ۱۰ عنصری از نوع int توصیف شده است ، C ، بطور اتوماتیک آن را به یک اشاره‌گر 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 – راه دوم برای توصیف یک پارامتر آرایه آن است که آن را بعنوان آرایه‌ای معرفی کنیم که اندازه (تعداد خانه‌های آن) مشخص نشده است به شکل زیر :

```
void display (num)
int  num[ ] ;
```

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

که در آن num بصورت یک آرایه از نوع int و با اندازه نامعلوم معرفی شده است . این روش نیز در واقع num را بعنوان یک اشاره‌گر int تعریف می‌کند .

روش سوم – راه سوم آن است که num را بعنوان یک اشاره‌گر int تعریف کنیم که اغلب برنامه‌نویسان حرفه‌ای این روش را بکار می‌برند . این روش در زیر نشان داده شده است :

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

که در آن "*" ، عملگر اشاره‌گر می‌باشد که این قسمت در فصل مربوط به اشاره‌گرها ، مورد بحث قرار خواهد گرفت .

یک نکته مهم که لازم به یادآوری است ، آن است که وقتی که یک آرایه بعنوان آرگومان یک تابع بکار برده می‌شود ، آدرس آن به تابع گذر داده می‌شود . این حالت در زبان C ، یک استثنا بر فراخوانی با مقدار ، در رابطه با قرارداد گذر دادن 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 نتیجه عملیات تابع روی آرایه‌ها ، در تابع اصلی (تابع فراخواننده) نیز منعکس خواهد شد .

- **توابع بازگشتی (Recursive Function)**

از نظر ریاضی ، یک تابع بازگشتی تابعی است که برحسب خودش تعریف می‌شود . بعنوان مثال فاکتوریل عدد صحیح و مثبت m که آن را با m! نمایش می‌دهند ، بصورت زیر تعریف می‌گردد :

$$m! = m(m-1)(m-2) \ldots 3 . 2 . 1$$

در ضمن فاکتوریل صفر برابر ۱ می‌باشد .

راه دیگر برای تعریف فاکتوریل که روش بازگشتی می‌باشد ، بصورت زیر است :

اگر m = ۱ باشد خروج . در غیر این‌صورت : $m(m-1)!$

ملاحظه می‌گردد که در روش دوم ، در صورتی که m بزرگتر از یک باشد ، فاکتوریل آن به‌طور مستقیم محاسبه نمی‌گردد . بلکه برحسب فاکتوریل عدد (m-1) تعریف می‌گردد . مثلاً فاکتوریل عدد ۵ بصورت :

$$5! = 5 * 4!$$

تعریف می‌گردد ؛ یعنی تابع دوباره برحسب خودش تعریف می‌شود . بعبارت دیگر تعریف تابع بر روی خود تابع برمی‌گردد ؛ بنابراین فاکتوریل ٤ دوباره بصورت :

$$4! = 4 * 3!$$

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 اینجا معمولاً باید به عقب برگشت و مقادیر ایجاد شده را در یکدیگر ضرب کرد تا در پایان ، فاکتوریل عدد ۵ بدست آید .

از نظر برنامه‌نویسی ، یک تابع را بازگشتی نامند ، اگر بطور مستقیم و یا غیرمستقیم ، تابع خود را فراخواند (صدا کند) . در بعضی زبانهای برنامه‌سازی مانند بیسیک و فرترن ۷۷ ، روش تعریف تابع بصورت بازگشتی پیش‌بینی نشده ؛ یعنی امکان خودفراخوانی تابع وجود ندارد . در بعضی از زبانها این امکان پیش‌بینی شده ، ولی هنگام تعریف تابع ، باید بطور صریح مشخص گردد که تابع بصورت بازگشتی تعریف می‌گردد تا کامپایلر ، تابع مزبور را بعنوان یک تابع بازگشتی بشناسد . زبان برنامه‌سازی PL/I از این گروه است . در C ، همهٔ توابع می‌توانند بصورت بازگشتی بکار برده شوند و نیاز نیست که این عمل بطور صریح بیان گردد .

ایده تابع بازگشتی در فرم اولیه‌اش خیلی ساده است . برای مثال برنامه زیر را ملاحظه و بررسی نمایید .

مثال –

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

طبیعی است که برنامه بالا پایان‌ناپذیر و یا نامتناهی است ؛ یعنی تابع مرتب خودش را فرامی‌خواند و در هر فراخوانی عبارت : " HELLO " چاپ می‌گردد . واضح است که این نوع خودفراخوانی توابع ، مطلوب نمی‌باشد ؛ یعنی باید تعداد دفعات خودفراخوانی ، متناهی باشد تا مانند مثال بالا با حلقه بدون پایان مواجه نشویم .

مثال – تابع زیر مجموع اعداد طبیعی ۱ تا n را محاسبه می‌کند که نحوهٔ عملکرد آن برای بعضی مقادیر n در جدول زیر تجزیه ، تحلیل و نمایش داده شده است .

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

مقدار برگردانده شده	فراخوانی تابع

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

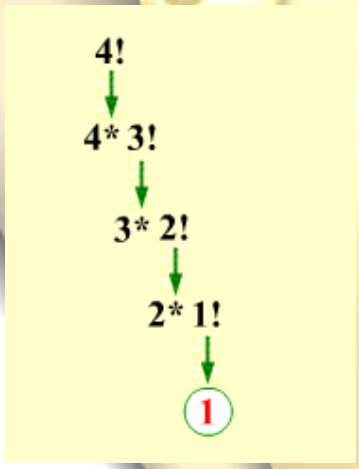
مثال – در برنامه زیر تابع مربوط به محاسبه فاکتوریل n به‌صورت بازگشتی همراه با خروجی آن نشان داده شده است . همچنین شیوهٔ کار و نحوهٔ عملکرد تابع بازگشتی برای محاسبهٔ فاکتوریل عدد 4 نمایش داده شده است .

تابع مزبور در داخل تابع اصلی ، به کمک حلقه for ، ۱۱ بار فراخوانی شده است .

```
#include <stdio.h>
long  factorial (long) ;
main( )
 {
    int  i ;
    for (i=0 ; i<=10; i + +)
      printf(" %2d! = %ld \n" , i , factorial
    (i)) ;
 }
long  factorial (long number)
 {
  if (number < =1)
    return 1 ;
  else
     return (number * factorial (number - 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**

بعضی مواقع مفید است (یا ضرورت دارد) که هنگام اجرای برنامه ، اطلاعاتی را به آن انتقال دهیم. روش کلی آن است که اطلاعات مورد نظر را با بکار 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 را با استفاده از خط فرمان توربو C ترجمه می‌کنید، چیزی

مشابه زیر تایپ می‌نمایید .

program-name >tcc

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

اگر بخواهید برنامه خود را با بکار بردن کامپایلر خط فرمان بورلند C ترجمه کنید ، دستور بالا را بصورت زیر تایپ می‌نمایید :

program-name >bcc

برای دریافت آرگومان خط فرمان ، دو آرگومان مخصوص توکار به اسامی argc و argv بکار برده می‌شود . تا به حال در تمام توابع main که بکار برده شد ، چیزی در داخل زوج پرانتز که پس از نام تابع می‌آید ، بکار برده نشد . یعنی پرانتزها توخالی بودند بصورت :

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 بصورت سنتی argc و argv نامیده می‌شوند ، بکار برده شوند . اولین آرگومان، یعنی argc باید یک متغیر از نوع صحیح باشد . درحالی که دومی ، یعنی argv یک آرایه از اشاره‌گرهایی به کاراکترها می‌باشد. درواقع آرایه‌ای از رشته‌ها است . 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 .

مثال – برنامه زیر ، نحوه تعریف پارامترهای argc و argv را در تابع main نشان می‌دهد :

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

که اغلب در گونه‌های جدید C ، می‌توان آن را به فرم فشرده‌تر زیر نوشت :

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

اجرای یک برنامه معمولاً با مشخص ساختن نام برنامه (درواقع نام فایلی که دربردارنده برنامه هدف ، ترجمه شده است) در سطح سیستم عامل آغاز می‌گردد . نام برنامه بعنوان یک فرمان سیستم‌عامل تفسیر می‌گردد . بنابراین سطری که نام برنامه در آن ظاهر می‌گردد ، معمولاً بعنوان خط فرمان تلقی می‌گردد .

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 . برای مثال :

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 بکار برده شود که در چنین حالتی تمامی پارامترها درون گیومه قرار می‌گیرد .

نام برنامه بعنوان اولین پارامتر در argv ذخیره می‌گردد که به دنبال آن سایر پارامترها می‌آیند . بنابراین اگر بدنبال 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 بطور اتوماتیک مقدار (m+1) نسبت داده خواهد شد . توجه داشته باشید که مقدار argc بطور صریح در خط فرمان تعیین نمی‌گردد .

مثال - برنامه ساده 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 (یعنی بدون نیاز به تعیین تعداد) پارامتر ، 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 . برای مثال فرض کنید که نام برنامه ، sample باشد و خط فرمان که اجرای برنامه را شروع می‌کند ، به‌صورت زیر است :

sample red white blue

اجرای برنامه مزبور خروجی زیر را بعنوان نتیجه ، ایجاد خواهد کرد :

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

خروجی بالا به ما می‌گوید که 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 ترجمه برنامه مبنای #sample.c بدست آمده است .)

به طریق مشابه ، اگر خط فرمان به‌صورت sample red "white blue" باشد ، خروجی حاصل ، بصورت زیر خواهد بود :

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

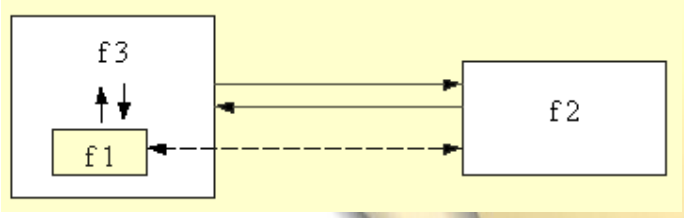
در این حالت ، رشته "white blue" بعلت وجود گیومه ، بعنوان یک پارامتر ساده تعبیر خواهد شد (یعنی هرکدام از دو رشته white و blue ، پارامتر مستقل درنظر گرفته نخواهند شد .)

همین که پارامترها به طریق بالا وارد شدند ، می‌توانند برحسب نیاز ، در برنامه بکار برده شوند . یکی از کاربردهای متداول آن است که اسامی فایلهای داده‌ها را بعنوان پارامترهای خط فرمان مشخص کنید .

- **استفاده از چند تابع (در یک برنامه)**

در زبان C ، می‌توان در هر برنامه به هر تعداد که نیاز باشد تابع تعریف کرد و بکار برد و هر تابع می‌تواند تابع دیگری را فراخوانی کند . در اغلب زبانهای برنامه‌سازی مانند زبان پاسکال می‌توان توابع را بصورت تودرتو تعریف کرد که در این صورت تابعی که در درون یک تابع تعریف شده ، نمی‌تواند توسط تابع دیگر فراخوانی شود . درواقع تابعی که در درون یک تابع تعریف شده ، نسبت به توابع دیگر قابل رؤیت نیست اما در زبان C اینطور نیست یعنی توابع بصورت تودرتو تعریف نمی‌گردند و همه توابع به‌صورت یکسان دستیابی می‌شوند و هر تابع می‌تواند تابع دیگر را فراخوانی کند . برای اینکه مفهوم بالا روشن شود ، به شکل زیر توجه کنید که ارتباط توابع در C و پاسکال را نمایش می‌دهد و مفهوم توابع تودرتو را روشن می‌سازد.

ارتباط توابع در پاسکال

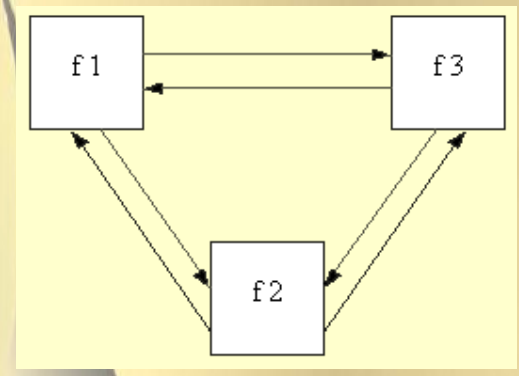


دو تابع f1 و f2 نمی‌توانند با یکدیگر ارتباط برقرار کنند .

دو تابع f1 و f3 می‌توانند با یکدیگر ارتباط برقرار کنند .

دو تابع f2 و f3 می‌توانند با یکدیگر ارتباط برقرار کنند .

ارتباط توابع در C



همه توابع می‌توانند با یکدیگر ارتباط برقرار کنند .

از شکل قبل استنباط می‌گردد که در زبان C ، توابع تودرتو (یعنی امکان تعریف دو تابع به‌صورت تودرتو) وجود ندارد .

- **قلمرو متغیرها**

حال می‌خواهیم قلمرو متغیرها را بررسی کنیم . یعنی تعیین نماییم که هر متغیر در چه قسمتی از برنامه و یا در کدامیک از توابع شناخته شده می‌باشد .

متغیرهایی که در داخل یک تابع توصیف گردند ، متغیرهای محلی یا local variables و یا متغیرهای خصوصی یا private variables نامیده می‌شوند . این گونه متغیرها فقط در درون همان تابع شناخته می‌شوند و سایر توابع یا تابع اصلی main ، آنها را نمی‌شناسد و به همین دلیل به آنها متغیرهای محلی یا خصوصی می‌گویند . یعنی خاص آن تابع مورد نظر است . در C ، به این گونه متغیرها بعنوان متغیرهای اتوماتیک یا automatic نیز ارجاع می‌شود به هرحال این گونه متغیرها فقط باید در داخل بلوکی که توصیف شده‌اند ، بکار برده شوند و در خارج بلوک خاص خود شناخته شده نیستند . یادآوری می‌شود که هر بلوک در زبان C با آکولاد باز شروع شده ، به آکولاد بسته خاتمه می‌پذیرد .

نکته مهم دیگر درباره 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 نمی‌باشند .

متداول‌ترین بلوکی که متغیرهای محلی توصیف می‌گردند ، بلوک مربوط به یک تابع است . برای مثال دو تابع زیر را درنظر بگیرید :

```
func1( )
    {
     int  x ;
     x = 10 ;
    }
func2( )
    {
      int  x ;
      x = -25 ;
    }
```

ملاحظه می‌شود که متغیر x دوبار توصیف شده است ؛ یکبار در تابع اول ، بار دیگر در تابع دوم .

با اینکه نام متغیر در هر دو تابع یکسان است ، ولی هیچ ارتباطی بین این دو وجود ندارد . x اولی فقط در درون func1 و x دومی نیز فقط در درون func2 شناخته می‌شود . درست مثل آن است که دو نفر دانشجو با یک نام (همنام) مثلاً با نام علی در دو کلاس مختلف ، وجود دارند . بنابراین ، این دو دانشجو ، دو شخصیت مختلف هستند . برحسب اینکه در کدامیک از این دو کلاس نام علی برده شود ، فقط دانشجوی مربوط به آن کلاس خاص مورد نظر خواهد بود . در این مثال ممکن است علی اولی مربوط به کلاس اول در کلاس حضور داشته باشد ، درحالی که علی دومی مربوط به کلاس دوم ، غایب بوده باشد . پس هیچ‌گونه رابطه‌ای بین این دو نفر که نام آنها یکسان است ، وجود ندارد .

متداول است که نام متغیرهای یک تابع را در آغاز تابع توصیف کنیم . ولی به هرحال این کار الزامی نیست ؛ بلکه می‌توان متغیرهای محلی را در درون هر بلوک که یک مجموعه کد (دستورها) می‌باشند ، توصیف کرد .

برای روشن شدن این مطلب ، به تابع زیر توجه کنید :

```
fun1( )
{
   int  t ;
   scanf ("%d" , &t) ;
   if ( t= =1)
      {
        char s[80] ;    /*  رشته s فقط هنگام ورود به این بلوک ایجاد خواهد شد .  */
        printf ("Enter  Name : ") ;
        gets(s) ;
        process(s) ;
      }
}
```

در اینجا متغیر محلی s هنگام ورود به بلوک مربوط به if ایجاد می‌گردد و هنگام خروج از آن تخریب می‌گردد . بعلاوه ، s 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 می‌گردد که در اینجا در هر بار ورود به بلوک ، متغیرهای محلی ، ایجاد شده و هنگام

خروج ، تخریب می‌گردند . در فراخوانی تابع باید به این خاصیت توجه کرد . یعنی وقتی که یک تابع فراخوانی می‌شود ، متغیرهای محلی آن ایجاد می‌گردند و هنگام خروج یا برگشت از آن تابع ، تخریب می‌گردند . 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 یا global variables متغیرهایی هستند که در طول تمام برنامه شناخته شده می‌باشند و می‌توانند به‌وسیلهٔ هر قسمت از برنامه بکار برده شوند . همچنین این گونه متغیرها مقادیر خود را در تمامی وقت اجرای برنامه نگه می‌دارند . متغیرهای عمومی با توصیف آنها در خارج هر تابع ، ایجاد می‌گردند . بنابراین به‌طور متعارف قبل از تابع main توصیف می‌گردند و به‌وسیلهٔ هر عبارتی در درون هریک از توابع ، قابل دسترسی می‌باشند . در قطعه برنامه زیر متغیر count خارج از توابع توصیف شده است که در اینجا این کار قبل از تابع main انجام گرفته است . به هرحال می‌توانست قبل از بکار برده شدن ، در هر جای دیگر نیز توصیف شده باشد . ولی باید خارج از تابعی که آن را بکار می‌برد توصیف گردد . روش متعارف آن است که متغیرهای عمومی در بالای برنامه توصیف گردند . مانند مثال زیر :

```
int  count ;        /* count is global */
main( )
 {
  count = 100 ;
  func1( ) ;
 }
func1( )
  {
   int  temp ;
   temp = count ;
   func2( ) ;
   printf("count is %d" , count) ;     /* print 100 */
  }
func2( )
  {
   int  count ;
   for (count = 1 ; count < 10 ; count + +)
      putchar (" ; ") ;
  }
```

با توجه به قطعه برنامه فوق ملاحظه می‌گردد که با اینکه متغیر count در main و همین‌طور در func1 توصیف نشده است ، هر دوی آنها می‌توانند آن را بکار برند . اما به هرحال در func2 یک متغیر محلی به نام count توصیف شده است . بنابراین هرکجا در این تابع ، متغیر مزبور بکار 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 بصورت محلی توصیف شده مراجعه شود ، هیچ تأثیری در مورد متغیر عمومی حاصل نخواهد شد .

حافظه مربوط به متغیرهای عمومی در ناحیه 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 بکار برده می‌شوند ، کاربرد زیادی دارند . به هرحال باید به دلایل زیر از بکار بردن غیرضروری متغیرهای عمومی خودداری نمود :

- آنها در تمام مدت اجرای برنامه ، حافظه را اشغال می‌کنند .
- استفاده از متغیر عمومی درحالی که متغیر محلی بتواند همان نقش را به نحو احسن 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 متغیرها در سطح گسترده و وسیع ، ممکن است به لحاظ اثرات جنبی ناخواسته و ناشناخته ، منجر به بروز اشتباهات در برنامه گردد .

یک مشکل عمده در تکوین برنامه‌های بزرگ ، تغییر ناگهانی مقدار یک متغیر است . زیرا متغیر در جایی دیگر از برنامه بکار برده می‌شود

- **کلاسهای حافظه**

کلاس حافظه ، زمان ابقاءِ یک متغیر و قلمرو (scope) آن را در یک برنامه مشخص می‌کند .

بطور کلی در زبان C ، چهار کلاس حافظه وجود دارد که عبارتند از :

ـ کلاس حافظهٔ اتوماتیک (automatic)

ـ کلاس حافظهٔ استاتیک (static)

ـ کلاس حافظهٔ ثبات (register)

ـ کلاس حافظهٔ خارجی (external)

این چهار کلاس حافظه به ترتیب با چهار کلمه کلیدی :

auto , static , register , extern

شناخته می‌شوند .

مثال – نحوه توصیف چندین متغیر با مشخص ساختن کلاس حافظه‌ها ، در زیر بیان شده است :

```
auto  int  a , b , c ;
extern  float  Root1 , Root2 ;
static  int  count = 6 ;
register  int  k ;
```

مثال سوم علاوه بر اینکه متغیر count را توصیف می‌کند و نوع کلاس آن را ازنظر حافظه ، بیان

می‌کند ، به آن مقدار اولیه نیز اختصاص می‌دهد .

کلاس حافظه اتوماتیک (auto)

متغیرهای با کلاس حافظه اتوماتیک همیشه در درون یک تابع توصیف می‌گردند و نسبت به همان تابع ، متغیرهای محلی می‌باشند . یعنی قلمرو کاربرد آنها و یا حیطه عملکرد آنها به همان تابع محدود است . بنابراین متغیرهایی که در توابع مختلف از این نوع حافظه تعریف می‌گردند ، از یکدیگر مستقل می‌باشند . اگرچه همنام باشند .

متغیرهایی که در درون یک تابع توصیف می‌گردد ، از لحاظ کلاس حافظه ، به‌طور پیش‌فرض اتوماتیک می‌باشند . یعنی در اینجا قرار دادن کلمه کلیدی auto در جلوی آنها ضروری نیست . پس اگر کلاس حافظه متغیری غیر از اتوماتیک باشد ، باید حتماً توسط کلمه کلیدی مربوط به آن کلاس حافظه ، به‌طور صریح مشخص گردد . می‌توان هنگام توصیف اینگونه متغیرها به آنها مقدار اولیه نیز اختصاص داد.

در زیر ، متغیرهایی با استفاده از توصیف auto و هم بدون آن آورده شده‌اند که نتیجه آنها یکسان است :

```
int  a , b , c ;
float d , e ;
char ch ;
int  a[50] ;
float  k = 3.14 ;
char ch ='A' ;
char s[15] = "noor" ;
```

```
auto  int  a , b , c ;
auto float d , e ;
auto char ch ;
auto int a[50] ;
auto float k = 3.14 ;
auto char ch ='A' ;
auto char s[15] = "noor" ;
```

به هرحال همان‌طور که بیان شد ، متغیرهای اتوماتیک 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 منطق برنامه‌ای ایجاب می‌کند که به یک متغیر اتوماتیک 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 مقدار آن متغیر مورد نظر نیز دوباره بازنشانده شود .

متغیرهای خارجی (extern)

به منظور کمک به مدیریت پروژه‌های بزرگ و سرعت بخشیدن به عمل ترجمه ، زبان برنامه‌سازی C ، اجازه می‌دهد که ماجول های جداگانه ترجمه شدهٔ یک برنامه 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 ، همه متغیرهای عمومی در یک فایل توصیف می‌گردند و در

سایر فایلها ، آن متغیرها با استفاده از کلمه کلیدی extern ، توصیف می‌گردند .

بطور کلی اگر حجم برنامه‌ای بزرگ باشد ، می‌توان آن را به قسمتهای منطقی کوچکتر به نام ماجول یا واحد تجزیه کرد و هر واحد را در یک فایل جداگانه قرار داد و ترجمه یا کامپایل نمود و سپس آنها را با یکدیگر اجرا کرد . 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 کلمه کلیدی extern معرفی کنیم . با این عمل به کامپایلر گفته می‌شود که این متغیرها در جای دیگری (درواقع در واحد اصلی) تعریف شده‌اند . یعنی اینها متغیرهای خارجی هستند .

مثال - در زیر ، دو ماجول 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 به‌طور دقیق از فایل اول نسخه‌برداری شده‌اند . ولی کلمه کلیدی extern در توصیف آنها افزوده شده است . توصیف‌کننده extern به کامپایلر می‌گویند 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 دوباره حافظه واقعی برای آنها ، بشناسد .

وقتی که یک متغیر عمومی را در درون یک تابع در همان فایلی که توصیف متغیرهای عمومی نیز در آن صورت گرفته است ، بکار می‌برید ، نیاز به بکار بردن extern نیست . مانند متغیر x در func1 و File1 و متغیرهای x , y در func2 و y در func3 از File2 مثال بالا .

به هرحال ممکن است extern را نیز انتخاب کنید . گرچه این روش کمتر متداول است . برای مثال قطعه برنامه 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 به‌صورت عمومی (global) تعریف شده‌ند ، لذا قلمرو آنها از همان محل یا نقطه تعریفشان تا انتهای برنامه می‌باشد و در بقیه برنامه و همه توابعی که بعد از آن می‌آیند شناخته شده‌اند . بنابراین توسط همه آن توابع قابل دسترسی هستند . حال در هرکدام از این توابع ، مقادیری به این‌گونه 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 نیاز به اقلام داده ورودی متعددی دارد ، یک را ساده در اختیار ما قرار می‌دهد . درضمن یادآور می‌شویم که طریق دیگر برای انتقال اطلاعات بین توابع را در فصل مربوط به اشاره‌گرها خواهیم دید .

به هرحال اگر تعریف تابع قبل از تعریف متغیرهای خارجی بیاید ، باید آن متغیرها در تابع مزبور نیز به‌عنوان متغیرهای خارجی توصیف گردند . همچنین باید توجه داشت که در توصیف متغیرها بعنوان 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 به‌عنوان 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 نمی‌باشد . ولی بعد می‌توان در همان تابع با دستور انتساب یا خواندن ، مقادیر دلخواه دیگری به آنها اختصاص داد .

متغیرهای استاتیک (static)

ما ، در این بخش و بخش دیگر ، بین برنامه تک‌فایلی یا single-file program که در آن تمامی برنامه در درون یک فایل مبنا قرار دارد ، با یک برنامه چندفایلی یا multi-file program که در آن توابع تشکیل‌دهنده یک برنامه در فایلهای مبنای جداگانه قرار دارند ، تمایزی قائل هستیم . قوانین حاکم بر کلاس حافظه استاتیک برای این دو حالت متفاوت است .

در یک برنامه تک‌فایل ، هرکدام از متغیرهای استاتیک در همان فایلی که مربوط به آن هستند تعریف می‌گردند . بنابراین قلمرو آنها مشابه متغیرهای اتوماتیک است . یعنی نسبت به تابعی که در درون آن تعریف شده‌اند ، محلی می‌باشند . اما به هرحال این گونه متغیرها برخلاف متغیرهای

اتوماتیک ، در تمامی طول حیات 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 دارا هستند که در موقع خروج از تابع داشتند . این ویژگی ، این اجازه را به تابع می‌دهد که متغیرهای مورد نظر در تمامی مدت اجرای یک برنامه مقدار خود را از دست ندهند .

متغیرهای استاتیک در درون یک تابع به همان شیوه‌ای که در مورد متغیرهای اتوماتیک بیان شد ، تعریف می‌گردند . با این تفاوت که توصیف متغیر باید با مشخص ساختن کلاس حافظه به کمک کلمه کلیدی static مشخص گردد . این گونه متغیرها می‌توانند مشابه همان متغیرهای اتوماتیک ، در درون تابع بکار برده شوند. به هرحال آنها نمی‌توانند خارج از تابعی که در درون آن تعریف شده‌اند ، مورد دستیابی قرار گیرند .

می‌توان متغیرهای اتوماتیک و ایستا یا استاتیک را همنام با متغیرهای خارجی تغریف کرد . بنابراین در چنین مواردی هر گونه تغییر در روی متغیرهای اتوماتیک و استاتیک ، هیچ‌گونه نقشی در مورد متغیرهای عمومی همنام با آنها نخواهد داشت .

مثال – برنامه 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 به‌صورت اعشاری و خارجی معرفی شده‌اند . اما a ، مجدداً در درون تابع main بصورت اعشاری و استاتیک تعریف شده است . بنابراین در این تابع فقط دو متغیر c و b به‌عنوان متغیرهای خارجی شناخته می‌شوند و متغیر محلی a مستقل از متغر 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 نوع حافظه استاتیک می‌باشد ، ولی b ازنظر نوع حافظه ، اتوماتیک می‌باشد . بنابراین هنگام خروج از تابع ، متغیر a مقدار قبلی خود را حفظ خواهد کرد ، اما b از دست خواهد داد . متغیر c نیز در این تابع بعنوان متغیر خارجی شناخته خواهد شد . ولی a و b که محلی هستند از متغیرهای a و b خارجی ، مستقل خواهند بود .

هنگام توصیف متغیرهای محلی استاتیک ، می‌توان آنها را آغازین کرد . یعنی به آنها مقدار اولیه نیز اختصاص داد که اگر به این‌گونه متغیرها مقدار اولیه اختصاص داده نشود ، مقدار آنها توسط

اغلب کامپایلرها ، صفر درنظر گرفته خواهد شد .

متغیرهای ثبات یا رجیستر (register)

نوع دیگری از مشخصه حافظه که فقط در مورد متغیرهایی از نوع int و char قابل اعمال است ، مشخصه ثبات یا register می‌باشد . این مشخصه موجب می‌گردد که متغیرهای با این نوع ، حافظه را به جای حافظه اصلی کامپیوتر (که به‌طور متعارف همه متغیرها در آن ذخیره می‌گردند) ثبات CPU درنظر گرفته شود که در این صورت هر نوع عملیات روی این‌گونه متغیرها ، خیلی سریع انجام می‌گیرد و این شیوه معمولاً در مورد متغیرهای کنترل‌کننده حلقه بکار برده می‌شود 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 بکار روند محدود است .

مثال – در تابع زیر متغیر i از این نوع معرفی شده است :

```
int  noor (m , i)
int  m ;
register int  i ;
 {
    register int  k ;
    k = 1 ;
    for ( ;  i>0 ; i--)
       k = k+m ;
    return  k ;
 }
```

در این مثال هر دو متغیر i و k بصورت متغیری با نوع حافظه register توصیف شده‌اند . زیرا هر دو در درون حلقه بکار برده می‌شوند .

تمرین و پاسخ

تمرین ۱ – برنامه‌ای بنویسید که دو عدد صحیح m و n را از طریق ورودی بخواند و سپس بزرگترین مقسوم علیه مشترک آن دو را بدست آورده ، چاپ کند .

حل : براساس تعریف 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 و 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 این‌صورت بجای عدد اولی ، عدد دومی را قرار می‌دهیم و بجای عدد دومی نیز

باقیمانده را قرار می‌دهیم و مجدد باقیمانده تقسیم عدد اولی بر دومی را بدست می‌آوریم . این عمل را آنقدر ادامه می‌دهیم تا باقیمانده برابر صفر گردد .

براساس منطق بالا ، برنامه مورد نظر در زیر نشان داده شده است :

```
#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) ;
 }
```

تمرین ۲ – برنامه تمرین یک را به کمک یک تابع فرعی بنویسید .

حل : برنامهٔ مورد نظر در زیر نشان داده شده است :

```
int  BMM(int a , int b)
 {
   int  r ;
   r = a % b ;
   while (r!=0)
     {  a = b ;
        b = r ;
        r = a % b ;
     }
   return(b) ;
 }
```

تمرین ۳ – برنامه تمرین یک را با استفاده از تابع بازگشتی بنویسید .

حل : برنامهٔ مورد نظر در زیر نشان داده شده است . در این برنامه تابع فرعی به‌صورت 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  I  y=%d  I  BMM=%d  I\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 ‌

تمرین ٤ – برنامه‌ای بنویسید که عددی را در مبنای ۱۰ بخواند ، معادل آن را در مبنای ۲ بدست آورده ، چاپ کند .

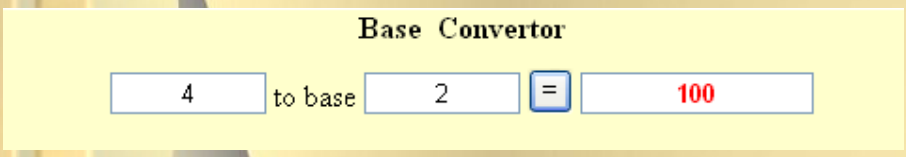
حل : ساده‌ترین راه برای بدست آوردن معادل یک عدد در مبنای دیگری مانند d آن است که عدد مورد نظر را بر مبنای مزبور تقسیم کنیم . باقیمانده حاصل ، مرتبه اول عدد حاصل در مبنای جدید را تشکیل خواهد داد ؛ سپس خارج قسمت را بر مبنای جدید تقسیم کنیم که این بار باقیمانده حاصل ، مرتبه دوم عدد حاصل در مبنای جدید را بدست می‌دهد . این عمل را ادامه می‌دهیم تا خارج قسمت برابر صفر گردد . برنامه زیر براساس این روش نوشته شده است که در آن باقیمانده‌ها در آرایه‌ای قرار داده می‌شوند و در پایان کار ، محتوای آرایه 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 خانه آخر آرایه قرار دارد ، لذا محتوای آرایه باید با شروع از خانه آخر به طرف خانه 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

```
decimal no : 23 The
binary value : 10111 Its
```

نمونه کاربردی این برنامه به این شکل است :



تمرین ۵ – برنامه تمرین قبل را بصورت یک تابع فرعی بنویسید . درضمن مبنای جدید را نیز d درنظر بگیرید که به فرض d کوچکتر از ۱۰ می‌باشد .

حل : برنامه مورد نظر در زیر نشان داده شده است (که در آن m ، عدد مورد نظر در مبنای ۱۰ و d نیز مبنای جدید می‌باشد .)

```
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] ) ;
}
```

تمرین ۶ – برنامه تمرین ٤ را بدون استفاده از آرایه بنویسید .

حل : در اینجا در هر تقسیم متوالی عدد مورد نظر بر مبنای جدید ، باقیمانده 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 نتیجه حاصل درست درآید .

برنامهٔ مورد نظر در زیر نشان داده شده است :

```
#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)
```

تمرین ۷ – برنامه تمرین ۶ را با فراخوانی یک تابع فرعی به‌صورت بازگشتی ، بنویسید .

حل : برنامه مورد نظر در زیر نشان داده شده است .

```
#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
10011  :  is 19 the binary equivalent of
```

تمرین ۸ – تابعی بنویسید که عدد صحیح m در مبنای ۲ را دریافت نماید و معادل آن را در مبنای ۱۰ بدست آورد و برگرداند .

حل : برای این کار باید عدد مزبور را مرتب بر ۱۰ تقسیم کنیم و باقیمانده حاصل را در توانهای متوالی از ۲ یعنی در اعداد :

, 2 , 4 , 8 , ... ۱

ضرب کنیم و این نتایج را با یکدیگر جمع نماییم . براساس این منطق ، تابع مورد نظر درزیر 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) ;
 }
```

تمرین ۹ – برنامه‌ای بنویسید که عدد صحیح n را دریافت کند و تعیین نماید که آیا عدد اول است یا نه ؟ و برحسب مورد پیغام مناسبی چاپ کند .

حل : می‌دانیم که اعداد اول ، اعدادی هستند که جز بر یک و خودشان ، بخش‌پذیر نمی‌باشند . باز هم می‌دانیم کلیه اعداد اول به غیر از عدد ۲ ، اعداد فرد می‌باشند . بنابراین در برنامه مزبور اول تست می‌شود که آیا عدد دریافتی مساوی یا کوچکتر از ۳ می‌باشد یا نه؟ که در صورت مثبت بودن پاسخ ، برنامه خاتمه می‌پذیرد . در غیر این‌صورت تست می‌شود که آیا عدد مورد نظر عدد زوج است یا نه؟ در صورت مثبت بودن پاسخ ، پیغامی مبنی بر اینکه عدد دریافتی عدد اول نمی‌باشد ، چاپ می‌گردد و برنامه خاتمه می‌پذیرد . درصورتی که هیچیک از دو حالت بالا مصداق پیدا نکند با استفاده از تابع کتابخانه ای sqrt ، جذر عدد مزبور محاسبه می‌گردد . سپس در یک حلقه 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 نمی‌باشد چاپ می‌گردد و برنامه پایان می‌یابد . چنانچه حلقه for کامل گردد ، عدد دریافتی عدد اول می‌باشد . بنابراین پس از چاپ پیغام مناسبی مبنی بر اینکه عدد دریافتی عدد اول می‌باشد ، برنامه به پایان می‌رسد .

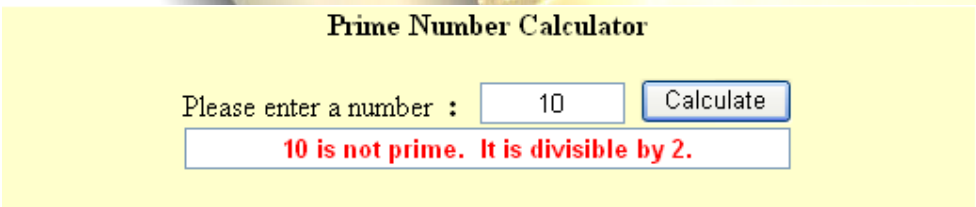
برنامه مورد نظر در زیر نشان داده شده است :

```
#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
```



تمرین ۱۰ – ازنظر ریاضی، عددی را که برابر مجموع مقسوم‌علیه‌های خودش (بغیر از خودش) باشد ، عدد کامل یا complete number نامند . مانند عدد ۶ که مقسوم‌علیه‌های آن ۱ ، ۲ ، ۳ می‌باشد و داریم :

۶ = ۱ + ۲ + ۳

برنامه‌ای بنویسید که عدد صحیح m را دریافت کند و تعیین نماید که آیا عدد کامل است یا نه و برحسب مورد ، پیغام مناسبی چاپ کند .

حل : برنامه موردنظر با نمونه‌ای از خروجی آن در زیر نشان داده شده است . در برنامه مزبور کلیه مقسوم‌علیه‌های آن عدد که مساوی یا کوچکتر از نصف عدد مزبور است بدست آمده و با یکدیگر جمع می‌گردند . سپس نتیجه حاصل با خود عدد مقایسه می‌گردد . اگر با یکدیگر مساوی بودند ، عدد مورد نظر یک عدد کامل است ؛ وگرنه ، 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

```
complete number  a is   6
complete number  a not  is   40
```

تمرین ۱۱ – 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 می‌باشد ، تعیین نماید و چاپ کند.

حل : برنامه 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
```

تمرین ۱۲ – اعداد دوقلو (twin numbers) ، دو عدد اول را گویند که تفاوت یا تفاضل آنها برابر ۲ باشد . مانند دو عدد ۳ و ۵ یا دو عدد ۱۱ و ۱۳ .

برنامه‌ای بنویسید که عدد صحیح m را دریافت کند و کلیه اعداد دوقلوی مساوی یا کوچکتر از آن را تعیین و چاپ کند .

حل : برنامه مورد نظر با خروجی آن برابر m = 100 در زیر نشان داده شده است . در این برنامه ، دو حلقه تودرتو بکار برده شده که حلقه بیرونی ، اعداد ۳ تا m را درنظر می‌گیرد و حلقه 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 بعبارت دیگر باقیمانده 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
```

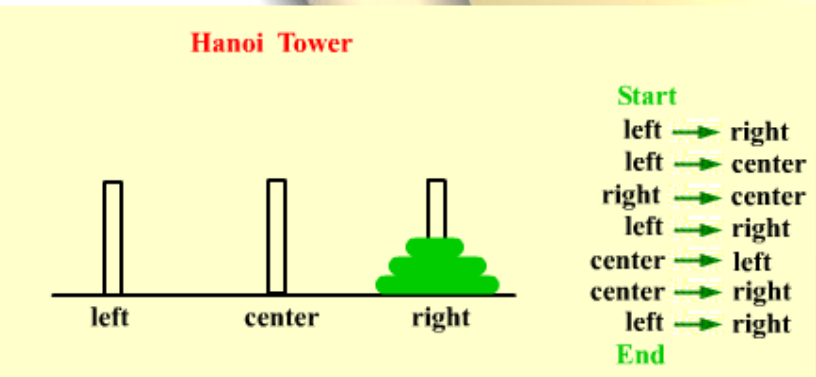
تمرین ۱۳ – تابعی بنویسید که عدد صحیح m را دریافت کند . اگر m ، عدد زوج بود مقدار یک وگرنه مقدار صفر برگرداند .

حل : تابع مورد نظر در زیر نشان داده شده است :

```
int  even ( int  m )
 {
   if (m%2 = = 0)
      return(1) ;
   else
       return(0) ;
  }
```

تمرین ۱٤ – برج هانوی (tower of Hanoi)

برج هانوی یکی از بازیهای معروف برای بچه‌ها است . در اینجا سه میله و تعدادی صفحات با قطرهای مختلف وجود دارد که در وسط آنها سوراخ وجود دارد و درنتیجه می‌توانند روی این میله‌ها سوار شوند . بازی به این طریق انجام می‌گیرد که در آغاز کار همهٔ صفحه‌ها در روی میله سمت چپ به‌صورت نزولی (یعنی صفحه بزرگتر در زیر و صفحه کوچکتر در بالا) قرار دارند . نمایش این حالت برای سه صفحه ، در شکل زیر نشان داده شده است .



شیوه یا استراتژی بازی به این طریق است که باید این صفحات با رعایت دو شرط زیر از میله سمت چپ (به‌عنوان میله مبدأ) به میله سمت راست (به‌عنوان میله مقصد) انتقال یابند :

ـ هر بار فقط مجاز به انتقال یا جابجایی یک صفحه از روی میله‌ای به روی میلهٔ دیگر هستیم (می‌توان از میله وسط نیز به‌عنوان واسطه و کمکی استفاده کرد) .

ـ هیچ‌وقت صفحه‌ای با قطر بزرگتر روی صفحه‌ای با قطر کوچکتر قرار نگیرد . برنامه‌سازی برای حل این مسئله به روش عادی یا روش تکراری یا Iteration method بسیار طولانی و مشکل است . ولی می‌توان آن را به سادگی با روش بازگشتی ، برنامه‌سازی کرد . قبل از آنکه روش برنامه‌سازی این مسئله را بیان کنیم ، یک روش استدلال ریاضی را در مورد اثبات قضیه که آن را استقراءِ 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 ارائه می‌دهند . در اینجا نیز از همین روش استفاده می‌کنیم . یعنی فرض می‌کنیم که برای انتقال n-1 صفحه از روی میله دلخواهی به روی میله دلخواه دیگر ، با رعایت دو شرط مذکور در بالا ، راهی وجود داشته باشد . با قبول این فرض می‌توان الگوریتم زیر را برای انتقال n صفحه از روی میله مبدأ به روی میله مقصد ارائه داد :

الف) n-1 صفحات رویی را (با استفاده از روشی که فرض کردیم وجود دارد) از روی میله چپ (میله مبدأ) به روی میلهٔ وسطی (میله واسطه) انتقال دهید .

ب) صفحه n اُم را (که بزرگترین صفحه است و اکنون آزاد شده است) از میلهٔ مبدأ (میله چپ) به میلهٔ مقصد (میله راست) انتقال دهید .

ج) n-1 صفحات را با استفاده از روشی که فرض کردیم وجود دارد از میله وسطی (میلهٔ واسطه) به روی میلهٔ راست (میله مقصد) انتقال دهید .

نکته : واضح است که اگر n = ۰ باشد نیازی به هیچ‌گونه نقل و انتقال نخواهد بود و درواقع این حالت قانون توقف الگوریتم را نشان می‌دهد .

نکته : مراحل اولی و آخری (یعنی دو مرحله ۱ و ۳) الگوریتم مزبور حالت بازگشتی دارد ؛ یعنی نقل و انتقال n-1 صفحه نیز برحسب n-2 بیان خواهد شد تا اینکه n = 0 گردد .

حال باتوجه به موارد مشروح در بالا و الگوریتم بیان شده ، برنامه و تابع مربوط به مسئله برج هانوی که به‌صورت بازگشتی تعریف گردیده ، در زیر نشان داده شده است :

```
#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 برنامه مزبور برای n = 3 به‌صورت 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
```

تمرین ۱۵ – برنامه زیر یک مستطیل توپر دور عبارت "COMPUTER SCIENCE" رسم می‌کند که در اینجا 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") ;
    }
```

تمرین ۱۶ – مثال صوتی :

می‌دانیم که چاپ کردن کاراکتر مخصوص '\X7' که در آسکی‌کد استاندارد ، bell نام دارد ، یک صدای کوتاه به نام beep در بلندگوی دستگاه ایجاد می‌کند . در برنامه زیر با فراخوانی تابع twobeep دو بار صدای کوتاه (منقطع) در بلندگوی دستگاه ایجاد می‌گردد .

```
main( )
  {
```

```
    twobeep( ) ;
    printf("type any character : ") ;
    getche( ) ;
    twobeep( ) ;
  }
twobeep( )
  {
    int  k ;
    printf("\x7") ;                    /* first beep */
    for (k=1 ; k<5000 ; k+ +) ;        /* dealy */
                                       /* (null statement) */
    printf("\x7") ;                    /* second beep */
  }
```

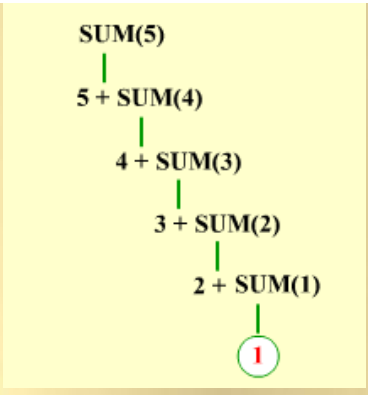
برنامه در آغاز ، تابع twobeep را فرا می‌خواند که درنتیجهٔ اجرای آن ، دوبار صدای کوتاه یا beep شنیده می‌شود ؛ سپس برنامه از شما می‌خواهد که کلیدی را فشار دهید . هنگامی که این کار را انجام دادید ، دوباره مانند قبل ، دوبار صدای مقطع یا بوق متوالی با فاصله کوتاه شنیده می‌شود . حلقه for در بین دوبار دستور printf موجب تأخیر زمانی یا delay بین دو صدای متوالی می‌گردد . ملاحظه می‌کنید که در بدنه این حلقه ، هیچ دستوری وجود ندارد . بلکه فقط در انتها شامل علامت سمی کولون (;) است که نمایانگر یک دستور تهی یا null statement می‌باشد .

تمرین ۱۷ – نحوهٔ عملکرد تابع بازگشتی زیر را که در آن n ، عدد صحیح غیرمنفی می‌باشد ، بیان کنید .

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

حل : تابع مزبور مجموع اعداد 1 تا n را محاسبه می‌کند و برمی‌گرداند . ردیابی نحوهٔ عملکرد تابع مزبور برای

n = 5 به‌صورت زیر خواهد بود .



فصل هفتم – آرایه‌ها

• مقدمه

آرایه ، مجموعه عناصری است که دارای ویژگیها و صفات یکسان می‌باشند . بعبارت دیگر آرایه یک فضای پیوسته از حافظه اصلی کامپیوتر است که می‌تواند چندین مقدار را در خود جای دهد. همه عناصر یک آرایه از یک نوع بوده و بوسیله اندیس مشخص می‌شوند.

از نظر ریاضی معمولاً آرایه‌های یک‌بعدی را بردار (vector) و آرایه‌های دوبعدی را ماتریس نامند. همچنین بطریق مشابه می‌توان آرایه‌های چندبعدی را تعریف کرد . در این فصل قابلیت های متعدد آرایه ها مورد بررسی قرار خواهند گرفت .

• تعریف آرایه‌ها

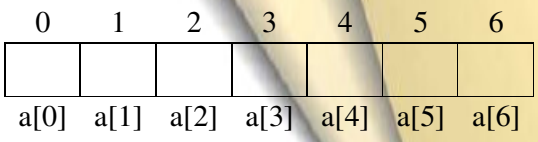
در زبان C ، آرایهٔ یک بعدی بصورت زیر تعریف می‌گردد :

type array-name [array-size] ;

که در آن array-name نام آرایه (که از قانون نامگذاری متغیرها تبعیت می‌کند) ، array-size بزرگی و یا تعداد عناصر آن و type نیز نوع عناصر آن را مشخص می‌کند . برای مثال اگر آرایه a دارای ۷ عنصر از نوع int باشد ، بصورت :

```
int  a[7] ;
```

معرفی می‌گردد و خانه‌های اختصاص داده شده به آن بصورت متوالی خواهد بود ، مانند شکل زیر :



ملاحظه می‌شود که شماره خانه‌ها از صفر تا شش می‌باشد . یعنی 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 برنامه‌سازی خوب آن است که اندازهٔ آرایه به‌صورت یک ثابت سمبولیک تعریف گردد ؛ مانند مثال زیر :

```
#define size 50
int a[size] ;    /* spaces for a[0] , a[1] , … , a[49]  i s  allocated */
```

آرایه‌ها ازنظر نوع اختصاص حافظه (storage class) می‌توانند بصورت اتوماتیک ، استاتیک و خارجی یا external باشند . اما نمی‌توانند بصورت register تعریف گردند . بنابراین در حالت کلی می‌توان یک آرایه یک‌بعدی را بصورت زیر تعریف کرد :

```
storage-class   data-type  array-name [expression] ;
```

که در آن expression ، می‌تواند یک عدد صحیح یا یک متغیر از نوع int و یا یک عبارت ساده 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 بعنوان معرف بزرگی آرایه خواهد بود . متعارف آن است که بزرگی آرایه بصورت یک عدد صحیح و یا متغیری از نوع عدد صحیح تعیین گردد . واضح است که اگر بزرگی آرایه به‌صورت یک متغیر از نوع int بیان گردد ، باید مقدار آن هنگام تعریف عناصر آن ، تشخیص داده شود .

مثالهای زیر نمونه‌هایی از تعریف چند آرایه یک بعدی را نشان می‌دهد :

```
int  a[50] ;
float  b[25] ;
static  float  c[15] ;
char  text[80] ;
```

آرایه a از نوع int با ۵۰ عنصر و آرایه b از نوع float با ۲۵ عنصر و آرایه C از نوع float با ۱۵ عنصر و از لحاظ کلاس حافظه ، از نوع استاتیک تعریف شده است و آرایه text از نوع char با ۸۰ عنصر تعریف شده است .

- **مراجعه به عناصر آرایه**

وقتی که آرایه‌ای را تعریف می‌کنیم ، کامپایلر یک مجموعه حافظه به‌صورت پیوسته (متوالی) برای آن پیش‌بینی می‌کند . وقتی که آرایه‌ای ایجاد شد ، برای مراجعه به هر عنصر دلخواه آن کافی است پس از نام آرایه ، شماره عنصر مورد نظر را در درون یک زوج کروشه قرار دهیم . مثلاً اگر a ، نام آرایه‌ای باشد که از پیش تعریف شده و مقادیری نیز به آن اختصاص داده شده باشد . دستور :

```
k = a [5] ;
```

یک کپی از محتوای خانهٔ شمارهٔ ۵ آرایه را به متغیر k نسبت می‌دهد .

یادآور می‌شویم که نامگذاری آرایه نیز از قانون نامگذاری متغیرها تبعیت می‌کند و نوع عناصر آن نیز مانند متغیرهای معمولی می‌تواند int ، float ، char و غیره باشد .

- **کلاسهای حافظه درآرایه (و نحوه مقداردهی اولیه آنها)**

بطوری که بیان شد ، آرایه‌ها از نظر نوع حافظه می‌توانند اتوماتیک ، استاتیک و یا خارجی تعریف گردند ، ولی نمی‌توانند به‌صورت register تعریف شوند . آرایه‌های از نوع کلاس اتوماتیک برخلاف متغیرهای اتوماتیک ، نمی‌توانند هنگام تعریف آنها ، مقدار اولیه بپذیرند .

حال باتوجه به توضیحات بالا می‌توان فرم کلی مقداردهی اولیه به آرایه‌ها ، یا آغازین کردن آرایه‌ها را به‌صورت زیر بیان کرد :

storage-class data-type array-name [size] = {value1 , value2 , ... value*m*} ;

که در آن value1 ، value2 ...value*m* به ترتیب مقدار اولیه اولین ، دومین ، ... و m اُمین عنصر آرایه را مشخص می‌کند .

فرض کنید که آرایه‌های a ، b و c بصورت 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 آرایه b ازنظر کلاس حافظه به‌صورت استاتیک معرفی شده ، ولی کلاس حافظه دو آرایه a و c به‌طور صریح بیان نشده است . بنابراین باید فرض کرد که هر دوی آنها خارجی می‌باشند . پس مقادیر عناصر سه آرایه مزبور بصورت زیر خواهد بود :

a [0] = 1	b [0] = 2.5	c (0) = a
a [1] = 2	b [1] = 3.5	c (1) = b
a [2] = 3	b [2] = 1.25	c (2) = c
a [3] = 4	b [3] = 12.5	
a [4] = 5	b [4] = 3.14	
a [5] = 6		
a [6] = 7		

اگر آرایه‌ای بصورت مقداردهی اولیه تعریف گردد ، نیاز نیست بزرگی یا اندازهٔ آن را مشخص کنیم ؛ مانند مثال زیر :

```
int  a[ ] = {3 , 2 , -2 , 4} ;
```

که عناصر آن بصورت زیر خواهد بود :

a [0] = 3

a [1] = 5

a [2] = -2

a [3] = 4

در همهٔ مثالهای بالا ، آرایه‌هایی که مقادیر اولیه گرفته‌اند و کلاس حافظهٔ آنها به‌طور صریح مشخص نشده ، از نوع خارجی فرض شده‌اند (یعنی از نوع اتوماتیک نیستند) . حال به چند مثال ساده

زیر ، در رابطه با نحوه خواندن اطلاعات به درون آرایه ، انجام عملیات روی آن به نمایش نتایج حاصل ، توجه نمایید .

مثال – برنامه‌ای بنویسید که نمرهٔ امتحانی ۲۵ نفر دانشجویان کلاسی را در درس برنامه‌نویسی بخواند و تعداد دانشجویان مردود و همچنین تعداد دانشجویانی را که نمرهٔ امتحانی آنان کمتر از معدل کلاس است ، تعیین و چاپ نماید .

حل : برنامه مورد نظر در زیرنشان داده شده است :

```
# include<stdio.h>
main ( )
  {
    float  a[25] , av , sum = 0 ;
    int  i , f = 0 , p = 0 ;
    for ( i = 0 ; i<25 ; ++ i )
      {
        scanf ("%f " , &a[i] ) ;
        sum = sum + a[i] ;
        if  (a[i]<10)
          f = f + 1 ;
       }
     av = sum / 25 ;
     for (i = 0 ; i<25 ; ++i)
        if (a[i] <av)
           p = p + 1 ;
     printf ("%f %d %d" , av , f , p) ;
  }
```

در برنامه بالا ، f و p به‌ترتیب معرف تعداد دانشجویان مردود و دانشجویانی است که نمرهٔ آنان کمتر از معدل کلاسی است که در آغاز مقدار اولیه صفر داده شده است. به لحاظ ساده بودن منطق برنامه از توضیح بیشتر خودداری می‌گردد . در ضمن یادآور می‌شویم که این برنامه را به‌طور 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 نمرات دانشجویان را با یک متغیر 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 نمره دانشجویان به حافظه کامپیوتر ، فقط نمره نفر آخر را در حافظه خواهیم داشت و درنتیجه در مورد دانشجویان اول تا بیست و چهارم نمی‌توانیم تعیین کنیم که نمره کدامیک از آنها کمتر از معدل کلاس است . پس در این مثال استفاده از آرایه الزامی است که درنتیجهٔ آن ۲۵ نمره به ۲۵ آدرس مختلف خوانده می‌شود . همچنین باید توجه کنیم که چون اندیس از صفر شروع می‌شود ، شماره 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 .

بنابراین فرم کلی تعریف آرایهٔ n بعدی به‌صورت زیر خواهد بود :

storage-class data-type array-name[size1][size2]... [size*n*] ;

که در آن size1 , size2 , ..., sizen به‌ترتیب بزرگی بعد یکم تا n اُم آرایه می‌باشد .

برای مثال یک آرایهٔ دوبعدی ۳ در ٤ از نوع int را می‌توان به‌صورت زیر معرفی کرد :

```
int  a[3][4] ;
```

درواقع آرایه دوبعدی m × n (شامل m سطرو n ستون) را می‌توان به‌صورت شکل زیر نشان داد :

	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]

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] ;
```

همچنین می‌توان هنگام تعریف آرایه‌های دوبعدی یا بالاتر به آنها مقدار اولیه نسبت داد که البته باید آرایه مورد نظر ، کلاس حافظه استاتیک یا خارجی داشته باشد .

مثال – 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 دارای کلاس حافظه خارجی می‌باشد . آرایهٔ مزبور را می‌توان جدولی تصور کرد که دارای ۳ سطر و ٤ ستون (٤ 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 به‌صورت 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 می‌باشد .

نحوه اختصاص مقادیر اولیه به عناصر آرایه را می‌توان با دسته‌بندی کردن آنها در داخل زوج آکولاد تغییر داد ؛ به‌عنوان مثال در آرایه 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 :

```
int  array[3][4] = {
                    {1 , 2 , 3 , 4} ,
                    {5 , 6 , 7 , 8} ,
                    {9 , 10 , 11 , 12}
                   } ;
```

با مثال قبلی 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 به‌صورت 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 به‌صورت :

```
int  array[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9) ;
```

تعریف کنیم ، مقادیر نسبت داده شده به عناصر آرایه مورد نظر به‌صورت 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 یک تابع (بعنوان آرگومان)**

در زبان C ، وقتی که نام آرایه بعنوان آرگومان یک تابع ظاهر می‌شود ، بعنوان آدرس اولین عنصر آرایه تعبیر می‌گردد . بنابراین هنگامی که آرایه‌ای را بعنوان آرگومان به یک تابع گذر یا انتقال می‌دهیم ، تمامی آرایه به آن تابع انتقال می‌یابد . این روش انتقال آرگومان به تابع ، با آنچه که در مورد انتقال متغیرهای معمولی به یک تابع در فصل مربوط به توابع بیان شد و فراخوانی با مقدار یا call by value نامیده شد ، متفاوت است . روش فراخوانی جدید را فراخوانی با آدرس یا call by address و یا فراخوانی با ارجاع یا call by refernce نامند . در این روش به جای کپی داده‌ها ، آدرس آنها به تابع انتقال می‌یابد . تشریح کامل این روش در فصل اشاره‌گرها یا pointers بیان خواهد شد .

برای گذر دادن یک آرایه به یک تابع باید فقط نام آن بدون کروشه و بدون اندیس ، بعنوان آرگومان واقعی در فراخوانی تابع ظاهر گردد . در تعریف تابع نیز باید آرگومان فرمال متناظر آن به همان طریق نوشته شود و بعنوان آرایه ، تعریف گردد . بدین طریق که نام آرایه همراه با یک زوج کروشه بدون اندیس نوشته شود و نوع داده آن نیز مشخص گردد .

مثال – قطعهٔ برنامه زیر نحوه فرستادن یا گذردادن یک آرایه را به یک 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 است که معرف تعداد عناصر آرایه است . دیگری آرایهٔ یک‌بعدی list که نوع عناصر آن float است . همچنین می‌بینید که در فراخوانی تابع ، آرایهٔ list به‌صورت یک متغیر ساده ظاهر گردیده 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 یک تناظر بین آرگومان واقعی n و آرگومان فرمال a وجود دارد . به طریق مشابه تناظری بین آرگومان واقعی list و آرگومان فرمال x وجود دارد . درواقع لازم نیست که آرگومانهای واقعی با آرگومانهای فرمال همنام باشند و به همین دلیل آرگومانهای فرمال را ، متغیرهای مجازی یا ساختگی یا dummy variables نیز می‌نامند . همچنین ملاحظه می‌کنید که بزرگی آرایه x ، در توصیف آرگومان فرمال ، مشخص نشده است . همینطور مشاهده می‌کنید که در تعریف تابع بصورت یک پیش‌نمونه یا prototype در تابع اصلی ، پس از نام تابع فقط یک زوج پرانتز تهی بکار 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 بکار برده شد .) قطعه برنامه 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 یک تابع ، آدرس اولین عنصر آرایه به تابع منتقل می‌شود . یعنی نام آرایه ، بعنوان آدرس اولین عنصر آرایه تفسیر می‌گردد . و وقتی که تابع فراخوانی می‌شود ، این آدرس به آرگومان فرمال متناظر آن اختصاص می‌یابد . پس آرگومان مزبور بعنوان اشاره‌گر به اولین عنصر آرایه برمی‌گردد . بنابراین هر عملی که در تابع فرعی روی آرایه انجام گیرد ، نتیجه آن در تابع اصلی (یا تابع فراخواننده) نیز منعکس می‌گردد و بطوری که بیان شد ، این شیوه 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 می‌تواند به سادگی در درون تابع مورد دسترسی قرار گیرد ؛ و بطوری که بیان شد

هر عنصر آرایه که در درون تابع تغییر یابد ، اثر این تغییر در تابع فراخواننده نیز منعکس خواهد شد .

برنامه زیر یک برنامه ساده را نشان می‌دهد که یک آرایه ۳ عنصری را به یک 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 می‌دهد .

```
# 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 ظاهر می‌گردد ، مقادیر :

a [0] = 1 , a [1] = 2 , a [2] = 3

به عناصر آرایه نسبت داده می‌شود و همین مقادیر با دستور printf نمایش داده می‌شود . 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 داده می‌شود .

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 بوسیلهٔ تابع modify روی آرایه ، در تابع اصلی نیز منعکس شده است .

- **آرایه‌ها و رشته‌ها**

رشته‌ها در زبان C ، بعنوان آرایه‌ای از کاراکترها تعریف می‌گردد و هر رشته به کاراکتر null که پایان رشته را دلالت می‌کند ، خاتمه می‌یابد . ثابت رشته‌ای در داخل دوبل گیومه قرار داده می‌شود و وقتی که ذخیره می‌گردد ، کاراکتر null بطور اتوماتیک به انتهای آن افزوده می‌شود . در داخل یک برنامه ، کاراکتر null در فرم escape sequence با '\0' نشان داده می‌شود . یک 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 . برای مثال رشتهٔ :

" hello there "

یک آرایهٔ ۱۲ کاراکتری است (فضای خالی و \0 نیز نیز بعنوان یک کاراکتر شمرده می‌شود .) متغیرهای رشته‌ای متغیرهایی هستند که مقادیر آنها می‌تواند یک رشته باشد . درواقع متغیرهای رشته‌ای ، آرایه‌هایی از نوع char می‌باشند . برای مثال قطعه 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 بعنوان پایان رشته ، حداکثر ۱۲ کاراکتر می‌باشد .

```
# 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 به‌صورت یک 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 بکار برده شده است . زیرا بطوری که بیان شد نام آرایه ، آدرس اولین خانه

یا اولین عنصر آن است . همچنین هم در موقع خواندن مقدار برای name و همینطور در چاپ مقدار آن از کد فرمت %s استفاده شده است .

می‌توان یک مجموعه از رشته‌ها را بصورت آرایه‌ای از رشته‌ها تعریف کرد . در اینجا چون خود رشته به تنهایی یک آرایه می‌باشد ، هر مجموعه از رشته‌ها می‌توانند یک آرایهٔ دوبعدی را تشکیل دهند که اگر آنها را به‌صورت جدول مستطیل شکل تصور کنیم ، هر سطر این جدول معرف یکی از رشته‌ها خواهد بود .

مثال – قطعه برنامه زیر اسامی ۱۵ نفر را به آرایه رشته‌ای name می‌خواند و آنها را در سطرهای متوالی چاپ می‌کند .

```
# include<stdio.h>
main ( )
  {
    char  name[15][12] ;
    int  i ;
    for ( i= ; 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 ارایه بکار برده شده است . یعنی درواقع اگر آرایه دوبعدی در این مجموعه از رشته‌ها را همانطور که در گذشته بیان شد ، یک جدول مستطیل شکل فرض کنیم که در هر سطر آن یک رشته قرار دارد . در خواندن اطلاعات هر بار یک سطر خوانده می‌شود که آدرس آن سطر بکار برده می‌شود و در نتیجه اپراتور آدرس نیز بکار 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 (Sorting)**

یکی از کاربردهای متداول آرایه ها ، استفاده از آنها در روشهای مرتب سازی است .

چنانچه مجموعه ای از داده‌ها یا اطلاعات ، براساس یک ویژگی یا نظم خاص سازمان‌دهی شوند ، این عمل را مرتب سازی گویند . در اینصورت پیدا کردن یک عنصر دلخواه در درون آن مجموعه ، ساده‌تر و سریعتر انجام می‌گیرد . بنابراین عمل مرتب کردن ، بمنظور سرعت بخشیدن به عمل جستجو می‌باشد .

تعداد مقایسه ها و نیز تعداد جابجایی عناصر، از عوامل اساسی در مورد سرعت مرتب سازیها میباشند. طبیعی است که هر روش مرتب سازی که دارای سرعت بالا و منطق ساده باشد و همچنین حافظه کمتری اشغال کند ، مطلوبتر است . بر این اساس روشهای متعددی ارائه شده که در ادامه

چند نمونه بررسی میشود .

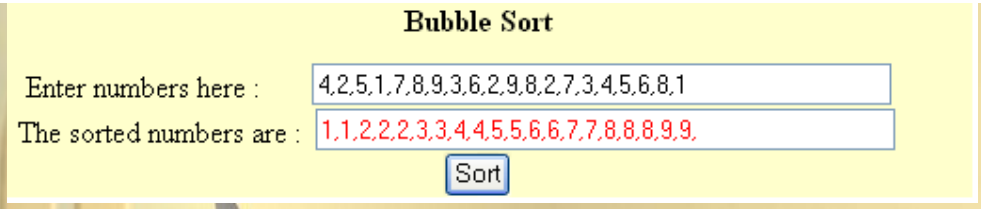
روش مرتب‌سازی حبابی (Bubble Sort)

این روش از نظر منطق ، ساده‌ترین و از نظر کارآیی پایین ترین روش مرتب‌سازی اطلاعات بشمار می رود . 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 بقیه عناصر به‌صورت 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 شده ، 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 خانه n-1 آرایه منتقل می‌شود . اگر این عمل را بصورت تکراری برای n-1 بار انجام دهیم و برای سرعت عمل بیشتر ، در هر بار تکرار نیز خانه آخر محدوده جاری آرایه را نادیده بگیریم ، یعنی هر بار از طول ناحیه مورد مقایسه یک واحد کاسته شود ، در پایان ، عناصر آرایه بصورت صعودی مرتب می‌گردد . اگر بخواهیم که عناصر آرایه مورد نظر به‌صورت نزولی مرتب شود ، باید در مقایسه 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 .

مثال – تابع زیر آرایه n عنصری A را به روش حبابی ، مرتب می نماید .

```
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 (Selection Sort)

در این روش مرتب سازی ، شیوه کار بدین صورت است که محل بزرگترین عنصر را در درون آرایه n عنصری مورد نظر پیدا می‌کنیم 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 درواقع دومین بزرگترین عنصر آرایه 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 اینصورت در تکرار آخر فقط دو عنصر a_1 و a_2 را خواهیم داشت که باید بزرگترین آن دو در خانه a_2 قرار گیرد .

برتری این روش نسبت به روش مرتب سازی حبابی آن است که تعداد تعویض عناصر کمتر می‌شود . زیرا در این روش ، در هر تکرار حداکثر یکبار جابجایی انجام می‌گیرد .

مثال – تابع زیر آرایه n عنصری A را به روش انتخابی مرتب می نماید .

```
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 (Searching)**

یکی دیگراز کاربردهای متداول آرایه ها ، استفاده از آنها در روشهای جسجو است .

هر گاه در داخل مجموعه‌ای از عناصر ، دنبال عنصر خاصی بگردیم ، این عمل را جستجو کردن یا searching نامند . جستجو ، در اغلب زمینه‌ها ، به‌ویژه در مورد بانکهای اطلاعاتی و سیستمهای تجاری ،

کاربرد زیادی دارند .

شیوه های متعددی برای جستجو وجود دارد که دو روش متداول آن ، در ادامه بررسی می‌گردد .

جستجو به روش خطی (Linear Search)

این روش ساده‌ترین راه برای جستجو در یک آرایه یا جدول نامرتب است . برای اینکار عنصر مورد جستجو را بطور متوالی با عناصر اول تا nاُم آن جدول مقایسه می کنیم . چنانچه در حین مقایسه ، عنصر مزبور پیدا شد ، شمارهٔ آن یادداشت شده و عمل جستجو خاتمه می یابد . در غیر این صورت پیغام مناسب صادر می شود . در این روش رابطه بین تعداد عناصر جدول و تعداد مقایسه ها ، بصورت یک معادله درجه اول خواهد بود .

مثال – تابع زیر عنصر x را در آرایه n عنصری A به روش جستجوی خطی جستجو میکند. اگر پیدا شد ، اندیس آن ، و در غیر اینصورت 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 (Binary Search)

روش دیگر برای جستجوی عنصری در داخل یک جدول یا آرایه ، روش دودویی است . این روش ، درصورتی 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 اینصورت 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 ، عمل جستجو خاتمه می‌یابد و پیغامی مبنی بر عدم وجود عنصر مورد جستجو در داخل جدول مورد نظر چاپ می‌گردد .

مثال – 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 ۵۶۷				جستجوی عدد ۲۰۵			
X	l	h	m	X	l	h	m
۲۱۸	1	10	5	218	1	10	5
500	6	10	۸	141	1	4	2
538	9	10	9	192	3	4	3
567	10	10	10	205	4	4	4

مثال – تابع زیر در یک آرایه مرتب شده n عنصری ، به روش دودویی عنصر x را جستجو میکند. اگر پیدا شد اندیس آن ، و در غیر اینصورت 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 نسخه های زبان C ، مجموعهٔ وسیعی از توابع کتابخانه‌ای در مورد عملیات روی رشته‌ها را پشتیبانی می‌کنند که در مبحث توابع کتابخانه‌ای مورد بحث قرار خواهند گرفت . در اینجا چند تابع

بسیار متداول که در نوشتن برنامه‌ها کاربرد زیادی دارند ، به شرح زیر بیان می‌گردد:

نام تابع	عمل تابع
strcpy (s1 , s2)	رشته s2 را روی رشته s1 کپی می‌کند .
strcat (s1 , s2)	رشته s2 را به دنبال رشته s1 ملحق (ضمیمه) می‌کند.
strlen (s)	طول رشته s را برمی‌گرداند .
strcmp (s1, s2)	رشته s1 را با رشته s2 مقایسه می‌کند .

اگر بخواهیم درمورد مقایسه دو رشته ، تمایز بین حروف بزرگ و کوچک نادیده گرفته شود ، تابع strcmp بصورت strcmpi بکار برده می‌شود .

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 بکار بریم .

حال به چند مثال در مورد رشته‌ها توجه نمایید .

مثال – برنامه زیر نحوه 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) ;
}
```

اگر این برنامه اجرا کنیم و برای دو رشته s1 و s2 کلمه "computer science" را وارد گردد، خروجی برنامه مزبور بصورت زیر خواهد بود :

```
Lengths : 16   16
The strings are equal
```

```
computer science
computer sciencecomputer science
```

مثال – برنامه‌ای بنویسید که اسامی n نفر دانشجو را بخواند ، سپس آنها را به ترتیب الفبا مرتب و چاپ کند . n ، حداکثر ۱۵ می‌باشد که با اولین دستور ورودی خوانده می‌شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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] ) ;
   }
```

- **تمرین و پاسخ**

تمرین ۱ – با استفاده از آرایه برنامه‌ای بنویسید که جملات اول تا دهم سری فیبوناچی را محاسبه و چاپ کند .

حل : برنامه مورد نظر با خروجی آن در زیر نشان داده شده است :

```
#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  551
```

تمرین ۲ – برنامه ای بنویسید که اعداد زوج ۲ تا ۲۰ را به عناصر یک آرایه ۱۰ عنصری نسبت دهد و سپس شماره هر عنصر و مقدار متناظر آن را در دو ستون ، نشان داده و چاپ کند .

حل : برنامه مورد نظر با خروجی آن در زیر نشان داده شده است :

```
# 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

تمرین ۳ – برنامهٔ زیریک آرایه ۱۲ عنصری را هنگام تعریف آن ، مقداردهی اولیه می‌کند . سپس مجموع مقادیر آن را محاسبه و چاپ می‌کند که خروجی آن نیز برای مقادیر مورد نظر ، نشان داده شده است :

```
# include<stdio.h>
# define  size  12
main( )
```

```
{
   int  a[size] = {1 , 3 , 5 , 4 , 7 , 2 , 99 , 16 , 45 , 67 , 89 , 45} ;
   int  i , total = 0 ;
   for (i=0 ; i<size  ; i++)
      total + = a[i] ;
   printf ("total of array element values is %d" , total) ;
 }
```

خروجی برنامه

```
total of array element values is 383
```

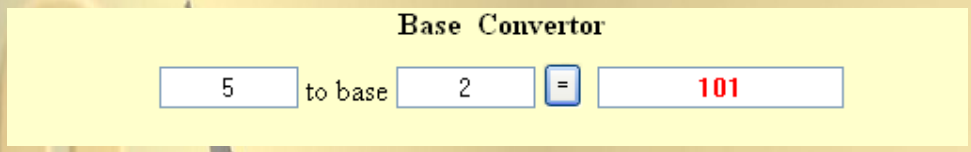
تمرین ٤ – برنامه‌ای بنویسید که عددی صحیح را دریافت کرده ، معادل آن را به شکل باینری (در مبنای ۲) چاپ کند .

حل : برنامه مورد نظر در زیر نشان داده شده است .

```
#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 کرده ، خارج‌قسمت و باقیمانده را بدست آورد که باقیمانده، مقدار مرتبه اول عدد در مبنای جدید خواهد بود. سپس باید دوباره خارج‌قسمت را بر مبنای d تقسیم کرد که باقیمانده آن مقدار مرتبه دوم عدد در مبنای جدید خواهد بود. این عمل را باید به ترتیب ادامه داد تا خارج‌قسمت مساوی صفر گردد . در برنامه بالا d = 2 است. بنابراین اولین باقیمانده تقسیم عدد دریافتی در اولین عضو آرایه یعنی a[0] قرار داده می‌شود سپس خارج‌قسمت عدد مزبور بر ۲ بدست می‌آید. باقیمانده 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 آرایه a قرار دارد که در پایان محتوای آرایه مزبور به‌عنوان معادل عدد m در مبنای ۲ چاپ می‌شود.

نمونه کاربردی از این برنامه به این شکل است:



تمرین ۵ – برنامه‌ای بنویسید که ۱۰ عدد صحیح را به‌صورت مقداردهی اولیه ، به عناصر آرایه‌ای نسبت دهد . سپس گراف آنها را بصورت نمودار میله‌ای افقی یا هیستوگرام (histogram) **رسم کند . به طریقی که به ازای هر مقدار عددی ، به همان تعداد ستاره در کنار هم چیده شوند . مثلاً عدد** 5 **را به‌صورت :**

*****

نمایش دهد .

حل : برنامه مورد نظر برای مقادیر عددی : 19 , 3 , 15 , 7 , 11 , 9 , 13 , 5 , 17 , 1 همراه با خروجی برنامه در زیر نشان داده شده است :

```
# include<stdio.h>
# define  size 10
main ( )
  {
      int  n[size] = {19 , 3 , 15 , 7 , 11 , 9 , 13 , 5 , 17 , 1} ;
      int  i , j ;
      printf("%s%13s%17s\n" , "element" , "value" , "histogram") ;
      for ( i=0 ; i<size ; i++ )
        {
          printf("%7d%13d" , i , n[i] ) ;
          for (j=1 ; j < = n[i] ; j++)
              printf("%c% , `*') ;
          printf("\n") ;
      }
  }
```

خروجی برنامه

Element	value	histogram
0	19	*******************
1	3	***
2	15	***************
3	7	*******
4	11	***********
5	9	*********
6	13	*************
7	5	*****

در برنامه مزبور ، حلقه بیرونی ۱۰ بار (یعنی به تعداد عناصر آرایه) تکرار می‌گردد ، که در هر تکرار حلقه درونی نیز به تعداد مقدار عددی موجود در خانه iاُم آرایه مزبور (که در آن i ، شماره تکرار حلقه بیرونی است) تکرار می‌گردد و هر بار یک ستاره چاپ می‌کند ، در ضمن هر بار که حلقه درونی کامل می‌گردد ، قلم نوشتار به سطر جدید انتقال می‌یابد تا ستاره‌های متناظر با مقدار بعدی ، در سطر جدید چاپ گردد .

تمرین ۶ - برنامه‌ای بنویسید که با استفاده از تابع کتابخانه‌ای rand **که اعداد تصادفی ایجاد می‌کند ، ۶۰۰۰ بار پرتاب یک تاس تخته نرد را شبیه‌سازی کند .**

می‌دانیم که تاس ، یک شش‌وجهی همگن است که 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 رویه 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 :

حل : برنامه مورد نظر با خروجی آن برای یک آزمایش نمونه ، در زیر نشان داده شده است :

```
# 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

تمرین ۷- تفاوت بین فراخوانی با مقدار و فراخوانی با آدرس (یا فراخوانی با ارجاع) در این

فصل به اختصار بیان شد . برنامه زیر 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 را نشان می‌دهد . به‌طوری که ملاحظه می‌کنید ، در این برنامه ، در حلقه for اول ، مقادیر آرایه چاپ می‌گردد . سپس تابع modifyarray(a) فراخوانی می‌شود که در این فراخوانی ، نام آرایه (یعنی آدرس آغاز آرایه) به تابع انتقال می‌یابد . تابع مزبور مقدار هریک از عناصر آرایه را دو برابر می‌کند . پس از برگشت کنترل به تابع اصلی ، مقادیر آرایه چاپ می‌گردد و ملاحظه می‌کنید که نتیجه عملکرد تابع فرعی روی آرایه ، در تابع اصلی نیز منعکس شده است . سپس با فراخوانی تابع modifyelement یک عنصر آرایه ، به آن تابع انتقال می‌یابد (که درواقع فراخوانی با مقدار است) . تابع مزبور مقدار عنصر دریافتی را دوبرابر می‌کند و نتیجه چاپ می‌گردد . پس از برگشت مجدد کنترل به تابع اصلی ، مقدار عنصر مزبور دوباره چاپ می‌شود و مشاهده می‌کنید که نتیجه عملکرد تابع modifyelement در تابع اصلی منعکس نشده است .

```
# 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 :
4     3     2     1    0
the values of the modified array are :
4     3     2     1    0
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
```

تمرین ۸ - محاسبه میانگین ، میانه و مد

در علم آمار ، سه پارامتر میانگین (mean) ، میانه (median) و مد (mode) یک مجموعه مقادیر ، بصورت زیر تعریف می‌گردند :

الف) میانگین n مقدار $a_1 , a_2 , ... , a_n$ که آن را میانگین حسابی نیز نامند ، برابر است با :

$$mean = \frac{a_1 + a_2 + ... + a_n}{n} = \frac{\sum_{i=1}^{n} a_i}{n}$$

ب) اگر عناصر را بصورت صعودی مرتب کنیم ، چنانچه تعداد عناصر ، فرد باشد ، مقدار عنصر وسطی را میانه نامند و چنانچه تعداد عناصر زوج باشد ، نصف مجموع دو مقدار وسطی را میانه نامند .

ج) مد ، عبارت از عنصری است که بیشتر از بقیه تکرار شده باشد .

نتیجه یک پرسش از ۹۹ نفر که پاسخهای داده شده ، مقادیر عددی صحیح بین ۱ تا ۱۰ می‌باشد ، مفروض است . برنامه‌ای بنویسید که ۳ پارامتر میانگین ، میانه و مد را محاسبه و چاپ نماید و همچنین چارت میله‌ای یا هیستوگرام (histogram) پاسخها را بصورت ستاره رسم کند .

حل : برنامه مورد نظر و نمونه‌ای از نتیجه اجرای آن در ادامه نشان داده شده است .

برنامه 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) ;
}
```

برنامه 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]") ;
}
```

برنامه 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
  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
  9 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 برنامه 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 حلقه for در تابع مزبور انجام می‌گیرد ؛ سپس برای اینکه بتوان میانه را تعیین کرد ، آرایه مزبور به‌صورت 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 میانه بدست 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 (که بین ۱ تا ۱۰ می‌باشد) بدست می‌آید و چارت 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 می‌باشد .

تمرین 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 شماره آن را برگرداند . در غیر اینصورت مقدار ۱- برگرداند . سپس در تابع اصلی پیغام مناسبی مبنی بر اینکه عدد مورد نظر پیدا شده است یا نه ، چاپ شود.

حل : برنامه مورد نظر در زیر نشان داده شده است .

```
# 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 ;
  }
```

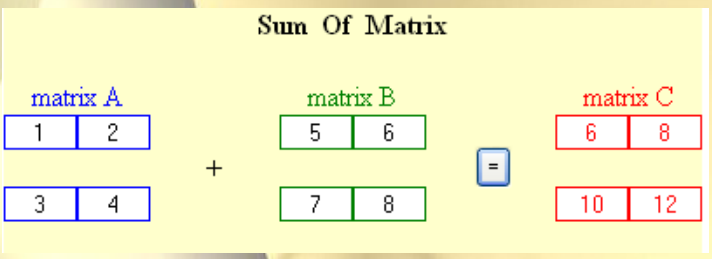
تمرین ۱۰ – برنامه‌ای بنویسید که عناصر دو ماتریس a و b را که دارای m سطر و n ستون باشند ، به حافظه بخواند و سپس مجموع آن دو را براساس قانون جمع ماتریسها بدست آورد و چاپ کند . m و n حداکثر ۸ می‌باشند که با اولین دستور ورودی خوانده می‌شوند .

حل : برنامه مورد نظردر زیر نشان داده شده است :

```
# 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] ) ;
     }
}
```

نمونه کاربردی از این برنامه به این شکل است :



تمرین 11 – برنامه‌ای بنویسید که عناصر دو ماتریس a و b را به حافظه بخواند و سپس حاصل ضرب آن دو را براساس قانون ضرب ماتریسها بدست آورد و نتیجه را به فرم ماتریس (آرایه دوبعدی) چاپ کند . ماتریس a دارای m سطر و n ستون و ماتریس b ، دارای n سطر و h ستون می‌باشد . m و n و h حداکثر ۷ می‌باشند که از اولین دستور ورودی خوانده می‌شوند .

حل : برنامه مورد نظر درزیر نشان داده شده است :

```
# 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] ) ;
        }
       }
```

تمرین 12 – برنامه‌ای بنویسید که یک سطر متن را با استفاده از تابع getchar هر بار یک کاراکتر به یک آرایه کاراکتری بخواند. سپس آرایه کاراکتری مزبور را به‌صورت یک رشته چاپ کند .

حل : برنامه مورد نظر با نمونه‌ای از خروجی آن برای حالتی که کاراکترهای ورودی آن ، رشته :

This is a test

باشد ، در زیر نشان داده شده است . ملاحظه می‌کنید که پس از خواندن کاراکترهای مورد نظر به آرایه 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 :
```

تمرین 13 – تابعی بنویسید که طول یک رشته را بدست آورده و برگرداند .

حل : تابع مورد نظر در زیر نشان داده شده است :

```
int  len( s )
char s [ ] ;
 {
   int  count = 0 ;
   while (s[count] !=`\0')
      count + + ;
   return (count) ;
 }
```

تمرین 14 – تابعی بنویسید که بدون استفاده از تابع کتابخانه‌ای Strcat **، دو رشته را به هم ملحق کند (یعنی رشته دوم را به آخر رشته اول ضمیمه نماید) .**

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
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 رشته اول بدست آمده و سپس رشته دوم به دنبال رشته اول درج گردیده است .

```
void  strcat (S1 , S2)
char S1 [ ] , S2 [ ] ;
 {
   int  i ;
   i = strlen (S1) ;
   for ( j=0 ; S2[j] != '\0' ; S1[+ +i] = S2 = [j] ) ;
 }
```

تمرین 15 – تابعی بنویسید که در درون یک رشته ، زیر رشته دیگری را جستجو کند . اگر وجود داشت ، محل آن (یعنی از چندمین کاراکتر رشته اول شروع شده است) را برگرداند ، در غیر اینصورت مقدار ۱- برگرداند .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
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) ;
   }
```

تمرین 16 – تابعی بنویسید که در درون رشته S1 ، از کاراکتر i اُم آن به طول j کاراکتر در رشته S2 کپی کند .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
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' ;
  }
```

تمرین 17 – برنامه‌ای بنویسید که یک مجموعه از رشته‌ها را به حافظه بخواند و سپس با فراخوانده شدن تابعی ، رشته‌های مزبور به ترتیب الفبا مرتب گردد . تعداد رشته‌ها حداکثر ۱۰ رشته می‌باشد که پایان آن با کلمهٔ "END" مشخص شده است .

حل : برنامه و تابع مورد نظر در زیر نشان داده شده است :

```
# 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 ;
}
```

فصل هشتم – اشاره‌گر (Pointer)

- **مقدمه**

در اغلب زبانهای برنامه نویسی قدیمی مانند فرترن و کوبول مفهومی بنام اشاره‌گر وجود ندارد . اما یکی از ویژگیهای بارز زبان C ، کاربرد متعدد اشاره‌گرها و انجام عملیات محاسباتی روی آنها می‌باشد. اشاره‌گر ، متغیری است که آدرس متغیر دیگری را در خود نگه می‌دارد . یعنی به آدرس متغیر دیگر اشاره می‌کند. بعبارت دیگر مقدار آن ، آدرس یک خانه از حافظه است . اشاره گر روش غیر مستقیم دسترسی به داده هاست 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 : لیستهای پیوندی ، درختها و گرافها
- عمل تخصیص حافظه بصورت پویا

در این فصل کاربردهای متعدد اشاره گرها در زبان C مورد بررسی قرار میگیرد .

• تعریف اشاره‌گر

برای استفاده از اشاره گر در برنامه ابتدا باید تعریف شود . روش کلی تعریف متغیری از نوع اشاره‌گر بصورت زیر است :

```
data-type * ptvar ;
```

که در آن ptvar نام متغیر مورد نظر است و data-type نوع متغیری است که آدرس آن می‌تواند در متغیر اشاره‌گر ptvar قرار گیرد . نماد `*` نیز اپراتور اشاره‌گر است .

بطور کلی هر داده ذخیره شده در حافظه کامپیوتر یک یا چندین بایت متوالی از خانه‌های حافظه را اشغال می‌کند . در صورتی می‌توان به یک داده دسترسی داشت که آدرس اولین خانه یا اولین بایت آن را در حافظه بدانیم . آدرس محل متغیر a در حافظه بوسیلهٔ عبارت &a تعیین می‌گردد که در آن & اپراتور یکانی یا تک‌اپراندی است و اپراتور آدرس نامیده می‌شود که آدرس اپراند یا عملوند خود را بدست می‌دهد . حال فرض کنید که متغیر a از نوع int باشد و متغیر pa را نیز بعنوان یک متغیر اشاره‌گر ، بصورت زیر توصیف کرده باشیم :

```
int *pa ;
```

حال می‌توان بوسیلهٔ این دستور جایگذاری :

```
pa = &a ;
```

آدرس متغیر a را به اشاره‌گر pa نسبت داد . pa یک اشاره‌گر a نامیده می‌شود . 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 می‌نماید و به همین لحاظ آن را متغیر اشاره‌گر نامند . شکل زیر رابطه بین pa و a را نشان می‌دهد :

رابطه بین اشاره گر و متغیر

داده‌ای که بوسیلهٔ a معرفی می‌گردد (یعنی داده‌ای که در خانه a از حافظه ذخیره شده است)

بوسیلهٔ عبارت *pa مورد دسترسی قرار می‌گیرد که در آن * یک 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 می‌نمایند . پس با اجرای دو دستور :

```
pa = &a ;
k = *pa ;
```

k و a هر دو یک مقدار را معرفی خواهند کرد . یعنی مقدار a بطور غیرمستقیم به k نسبت داده خواهد شد . بعبارت دیگر نتیجهٔ دو دستور مزبور مشابه نتیجه دستور زیر است :

```
k = a
```

بنابراین ملاحظه می‌گردد که اپراتور * در مورد *pa محتوای محلی را برمی‌گرداند که آدرس آن در pa قرار دارد و به همین لحاظ به آن ، عملگر غیرمستقیم نیز گویند .

- **پیدا کردن آدرس داده**

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 یک متغیر به جای نام آن متغیر ، از طریق آدرس آن دستیابی شود . برای بدست آوردن آدرس یک متغیر ، اپراتور ampersand یا & بکار برده می‌شود . برای مثال فرض کنید که متغیر k از نوع long int و آدرس آن ۱۲۳٤ باشد ، دستور :

```
Ptr = &k ;
```

مقدار ۱۲۳٤ (آدرس متغیر k) را در متغیر Ptr ذخیر می‌کند که البته باید Ptr از نوع اشاره‌گری که به متغیری از نوع long int اشاره می‌کند ، توصیف شده باشد .

مثال – قطعه برنامه زیر مقدار و آدرس متغیر k را چاپ می‌کند .

```
# include <stdio.h>
main ( )
 {
   int  K = 5 ;
   printf ("the value of K is : %d\n" , k) ;
   printf ("the address of K is : %p\n" , &k) ;
  }
```

خروجی برنامه

```
the value of k is : 5
the address of k is : 1234
```

یادآوری می‌شود که در تابع printf برای چاپ آدرس متغیر از کد فرمت %p استفاده شده است . این کد فرمت ممکن است روی کامپایلرهای قدیمی وجود نداشته باشد .

همچنین می‌توان قطعه برنامه بالا را بصورت زیر نیز نوشت :

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

```
main ( )
 {
  int  k = 5 ;
  int  *pk ;
  pk = &k ;      /* assigns the address of  k  to pk */
  printf ("the address of  k is : %p\n" , pk) ;
 }
```

خروجی این برنامه نیز مشابه قبلی خواهد بود .

از مطالب بحث شده در این فصل نتیجه می‌گیریم که اپراتور ستاره یعنی * به دو مفهوم جداگانه بکار برده می‌شود :

الف) در معرفی متغیرها به‌عنوان اپراتور 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 یک متغیر اشاره‌گر قرار دارد . برنامه زیر این مفاهیم را روشنتر می‌کند .

```
# 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 برنامه مزبور متغیر pch بعنوان اشاره‌گری به متغیرهایی از نوع کاراکتر اشاره می‌کند ، توصیف شده است . متغیرهای ch1 و ch2 نیز از نوع کاراکتر اعلان شده‌اند که به متغیر ch1 ، کاراکتر `a´ به‌عنوان مقدار اولیه نسبت داده شده است . در دستور printf اول آدرس متغیر pch چاپ

می‌گردد که به فرض ۱۰۰٤ می‌باشد . سپس آدرس متغیر ch1 ، به pch نسبت داده می‌شود و در دستور printf دوم ، مقدار pch (آدرس متغیر ch1) که به فرض ۲۰۰۱ می‌باشد ، چاپ می‌گردد . در دستور printf سوم ، محتوای خانه‌ای از حافظه که آدرس آن در متغیر pch قرار دارد (یعنی مقدار متغیر ch1) که 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´ می‌باشد) چاپ می‌گردد .

نکته – عبارت *pch می‌تواند در سمت چپ یک دستور جایگذاری ظاهر شود . مثلاً در همان برنامه بالا پس از اجرای دستور :

```
pch = & ch1 ;
```

با دستور :

```
*pch = `b´ ;
```

کاراکتر b به متغیر ch1 نسبت داده می‌شود .

- **مقداردهی اولیه اشاره‌گر**

به متغیرهایی از نوع اشاره‌گر نیز می‌توان هنگام اعلان آنها ، مشابه سایر متغیرها مقدار اولیه نیز نسبت داد و به عبارت دیگر آن را آغازین کرد . به هر حال مقدار اولیه مورد نظر باید یک آدرس باشد . پس یک اشاره‌گر می‌تواند ، NULL یا یک آدرس را به‌عنوان مقدار اولیه بپذیرد . برای مثال می‌توان دستورهایی بصورت زیر نوشت :

```
int  x ;
int *px = &x ;
```

به هرحال نمی‌توان یک متغیر را قبل از اینکه توصیف یا اعلان گردد ، در دستوری بکار برد . بنابراین مجموعه دستورهای زیر غیرقابل قبول است :

```
int *px = &x ;
int  x ;
```

همچنین می‌توان یک اشاره‌گر را بصورت :

```
int *ptr = 0 ;
```

مقداردهی اولیه کرد که برای مشخص ساختن بعضی شرایط خاص بکار برده می‌شود .

در حالت کلی ، نسبت دادن یک مقدار صحیح به یک متغیر اشاره‌گر ، دارای مفهوم نیست . به هرحال ، مثال اخیر یک حالت استثناءِ در این مورد است که همانطور که در بالا بیان شد ، برای مشخص ساختن بعضی شرایط خاص بکار برده می‌شود . در چنین مواردی توصیه می‌گردد که یک ثابت سمبولیک مانند NULL را که معرف صفر باشد تعریف نمود و آن را به اشاره‌گر اختصاص داد . این روش ، تأکید می‌کند که اختصاص دادن صفر ، معرف یک شرط ویژه می‌باشد .

مثال- یک برنامه به زبان C می‌تواند تعاریف و عبارات زیر را شامل باشد :

```
# define   NULL   0
float  x , y ;
float  *pr = NULL ;
```

در این مثال متغیرهای x و y بصورت متغیرهایی از نوع ممیز شناور و pr بصورت یک متغیر اشاره‌گر اعلان شده است 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 اولیه صفر یا NULL به یک اشاره‌گر ، هم‌ارز است ، ولی NULL ترجیح داده می‌شود .

- **اشاره‌گر تهی (NULL pointer)**

زبان C مفهوم اشاره‌گر NULL را پشتیبانی می‌کند و آن اشاره‌گری است که به هیچ شئی قابل قبول (معتبر) اشاره نمی‌کند . یک اشاره گر NULL ، هر اشاره‌گری است که مقدار صحیح صفر به آن نسبت داده باشد ؛ مانند مثال زیر.

مثال – در مثال زیر اشاره گر p مقدار صفر دارد :

```
char  *p ;
p = 0 ;
```

اشاره‌گرهای NULL به ویژه در دستورهای مربوط به کنترل جریان مفید هستند . زیرا اشاره‌گرهایی با مقدار صفر بعنوان false درنظر گرفته می‌شوند ، درحالی که متغیرهای اشاره‌گر با سایر مقادیر ، true منظور می‌گردند .

مثال – در برنامه زیر حلقه while تا موقعی که p یک اشاره‌گر NULL نباشد ، عمل تکرار را ادامه می‌دهد .

```
char  *p ;
   ....
   ....
while (p)
    {
      ....
      ....
    }
```

این گونه کاربرد اشاره‌گرها به ویژه در کاربردهایی که آرایه‌هایی از اشاره‌گرها را بکار می‌برد ،

آشکار می‌گردد که دوباره در همین فصل بررسی خواهد شد .

- **عملیات روی اشاره‌گرها**

عملیات متداولی که روی اشاره‌گرها انجام می‌شوند ، عبارتند از :

الف) عمل انتساب

در مورد اشاره‌گرها نیز مشابه سایر انواع متغیرها ، می‌توان مقداری را به یک متغیر اشاره‌گر نسبت داد . به هر حال بطوری که مشاهده شده ، این عمل مانند سایر متغیرها گسترده نمی‌باشد . به یک متغیر اشاره‌گر می‌توان آدرس یک متغیر یا مقدار صفر را نسبت داد .

ب) اعمال محاسباتی

زبان C ، اجازه می‌دهد که یک مقدار صحیح را به یک اشاره‌گر اضافه کنید و یا از آن کسر نمایید ؛ برای مثال اگر p یک متغیر اشاره‌گر باشد ، عباراتی مشابه :

p+5 , 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 بایت (۵×۲ = ۱۰)

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 بایت (۵×٤ = ۲۰)

بعد اشاره خواهد نمود . بنابراین p+5 همیشه به مفهوم به اندازه 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 بدست آید . بنابراین عملیات محاسباتی روی اشاره‌گرها ، چهار عمل افزودن ، کاستن ، ++ و -- می‌باشد .

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 . درواقع این تفاضل تعداد اشیاءِ بین این دو اشاره‌گر را معرفی می‌نماید .

- **انتقال مقادیر به تابع**

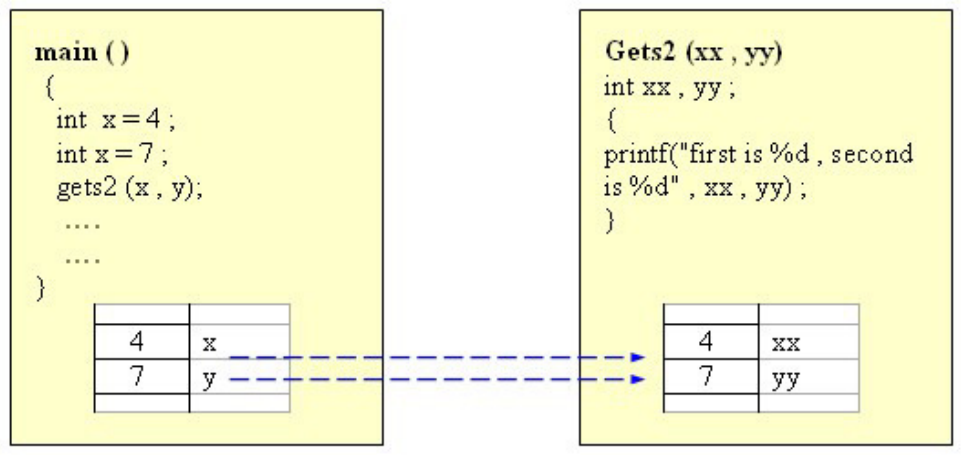
حال ببینیم وقتی مقادیری را به تابع گذر می‌دهیم ، چه اتفاقی رخ می‌دهد . در اینجا برنامه ساده‌ای را ملاحظه می‌کنید که دو مقدار صحیح ٤ و ٧ را به تابعی به نام gets2 می‌فرستد :

```
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 نکته مهمی را نشان می‌دهد :

نکته مهم این است که تابع دو مقدار از برنامه فراخواننده آن دریافت می‌کند و آنها را بطور جداگانه ، یعنی به‌صورت دوبله در فضای حافظه 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 که به جای x و y بکار رفته است (البته می‌توانست همان x و y نیز بکار برده شود) .

شکل زیر این مکانیسم 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) .



- **انتقال اشاره‌گر به تابع**

اشاره‌گرها ، اغلب بعنوان آرگومان ، به یک تابع گذر داده می‌شود . این عمل اجازه می‌دهد که عناصر داده‌های برنامه فراخواننده (که معمولا تابع main است) به‌وسیلهٔ تابع فراخوانده شده ، قابل

دستیابی باشند و در داخل تابع فراخوانده شده تغییر یابند و نتیجه در تابع یا برنامه فراخواننده نیز اعمال گردد . این گونه کاربرد اشاره‌گر را گذر دادن آرگومانها به وسیلهٔ آدرس و یا محل نامند .

وقتی که آرگومانها به وسیلهٔ مقدار ، گذر داده می‌شوند . عناصر داده به تابع کپی می‌شوند . بنابراین هر گونه تغییرات اعمال شده در روی آنها در درون تابع یا روتین فراخواننده ، اثر نمی‌گذارد . وقتی که آرگومان بصورت آدرس انتقال می‌یابد (یعنی وقتی که یک اشاره‌گر به یک 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 ۱ و ۲ دو گونه از تابعی به نام add1 را نشان می‌دهد که آرگومان خود را یک واحد افزایش می‌دهد . مثال ۱ متغیر count را با روش فراخوانی با مقدار ، به تابع گذر می‌دهد . تابع add1 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 ۲ ، برنامه مورد نظر ، متغیر count را با فراخوانی آدرس ، گذر می‌دهد . یعنی آدرس count (نه مقدار آن) به تابع add1 گذر داده می‌شود . تابع add1 یک اشاره‌گر به مقدار صحیح را که در اینجا countptr نامیده شده است بعنوان آرگومان دریافت می‌کند . تابع add1 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.

مثال ۱.- افزایش مقدار یک متغیر با بکار بردن فراخوانی با مقدار :

```
#include <stdio.h>
main ( )
 {
   int  count = 7 ;
   int  add1(int) ;
   printf ("the original value of count is %d\n" , count) ;
   count = add1 (count) ;
   printf ("the new value of count is %d\n" , count) ;
}
int  add1(int c)
 {
    return ++c ;     \* incrementsl variable c */
 }
```

خروجی برنامه

```
the original value of count is
7
the new value count is 8
```

مثال ۲ - افزایش مقدار یک متغیر به وسیلهٔ فراخوانی با آدرس :

```
# include <stdio.h>
main ( )
 {
  int  count = 7 ;
  int  add1 (int *) ;
  printf("the original value of count is %d\n" , count) ;
  add1 (& count) ;
  printf ("the new value of count is %d/n" , count) ;
 }
void  add1 (int  *countptr)
 {
   + + (*countptr) ;       /* increments count in main */
  }
```

خروجی برنامه

```
the original value of count is 7
the new value count is 8
```

پارامتر متناظر تابعی که یک آدرس را بعنوان آرگومان دریافت می‌کند ، باید یک اشاره‌گر باشد . مثلاً عنوان تابع add1 در مثال قبلی بصورت زیر است :

```
void  add1 (int  *countptr)
```

و بیان می‌کند که add1 آدرس یک متغیر از نوع int را دریافت خواهد کرد و آن را در countptr ذخیره خواهد نمود .

- **انتقال دوطرفه اطلاعات (بین تابع اصلی و فرعی به کمک اشاره‌گرها)**

وقتی که تابعی ، آدرس متغیری را در برنامه فراخواننده آن بداند ، می‌تواند هم مقادیری را در این متغیرها قرار دهد (یعنی مقادیر آنها را تغییر دهد) و هم مقادیر آن متغیرها را بکار برد . بنابراین به کمک اشاره‌گرها می‌توان مقادیر را هم از برنامه فراخواننده به تابع فراخوانده شده و هم از تابع فراخوانده شده به برنامه فراخواننده آن (درواقع در هر دو جهت) گذر داد . البته قبلاً در مبحث توابع ، انتقال مقادیر به روش فراخوانی با مقدار را دیده‌ایم . ولی روش مذکور در بالا ، 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 . مثال زیر اهمیت و کاربرد آن این موضوع را نشان می‌دهد .

مثال – برنامه‌ای بنویسید که ضرایب معادله درجه دومی را بخواند و سپس با فراخوانده شدن تابع فرعی به نام root ریشه‌های معادله مزبور محاسبه گردد و به تابع اصلی برگردانده شود . اگر معادله ریشه حقیقی نداشته باشد ، تابع فرعی هر دو ریشه را بعنوان صفر برگرداند . در ضمن اگر معادله ریشه داشته باشد ، ضرایب معادله همراه با ریشه‌های آن در تابع اصلی چاپ شود در غیر اینصورت ضرایب آن همراه با پیغام مناسب چاپ شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
#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 x1 , x2 (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 x1 , x2 نسبت داده می‌شود (یعنی در محلهایی که آدرس آنها در متغیر اشاره‌گر px1 و px2 می‌باشد قرار می‌گیرد) در غیر اینصورت 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 رابطه نزدیکی وجود دارد . قطعه برنامه 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 ;
```

هم‌ارز می‌باشند . پس در قطعه برنامه 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 با دو طریق زیر انجام دهیم :

str[4] یا *(p+4)

هر دو دستور ، پنجمین عنصر را برمی‌گردانند .

همینطور اگر داشته باشیم :

```
int  a[15] , *p ;
p = &a[0] ;
```

دو عبارت a[3] و *(p+3) هم‌ارز بوده و هر دو چهارمین عنصر از ۱۵ عنصر را برمی‌گردانند .

یعنی بطور کلی عنصر :

a[k]

هم‌ارز با :

*(p + k)

خواهد بود و هر دو محتوای خانه k اُم آرایه a و یا (k+1) اُمین عنصر از مجموعه عناصر مزبور را برمی‌گردانند و همینطور عبارت مزبور هم‌ارز :

*(a+k)

می‌باشد . زیرا a نام آرایه ، معرف آدرس آغاز آن می‌باشد و (a+k) آدرس خانه k اُم آرایه خواهد بود و در نتیجه عنصر :

*(a + k)

محتوای خانه k اُم از آرایه 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 همینطور :

*(a + k)

سریعتر از a[k] عمل می‌کند . بدین لحاظ استفاده از اشاره‌گرها برای دستیابی به عناصر آرایه ، یک روش بسیار متداول در زبان C می‌باشد .

حال برای تفسیر k در عبارتهای *(a+k) و *(p+k) به برنامه زیر توجه نمایید .

```
/* prints 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]) ;
 }
```

خروجی برنامه

```
92
81
70
69
58
```

برنامه مزبور یک برنامه ساده است که در آن برای دستیابی به عناصر آرایه ، از روش متعارف علامتگذاری آرایه استفاده شده است .

حال همان برنامه با روش بکارگیری از اشاره‌گر بصورت زیر خواهد بود :

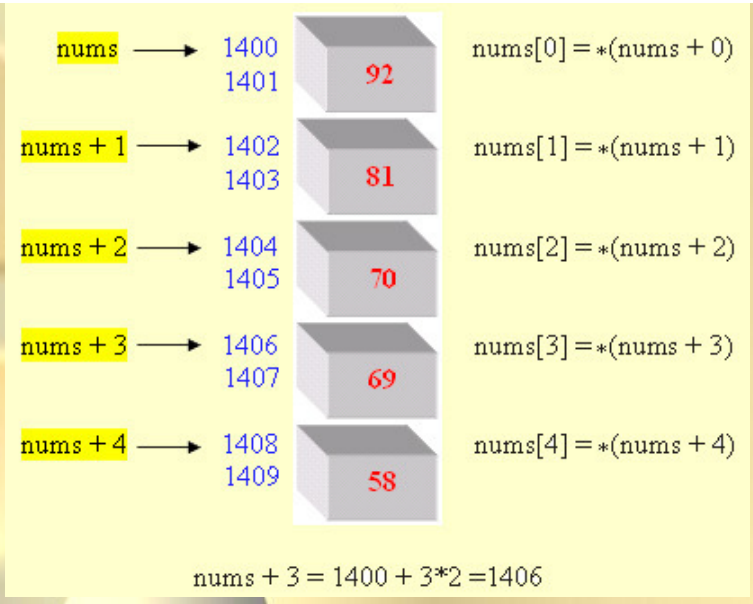
```
/* uses pointers to print out values from array */
main ( )
 {
  static int nums[ ]= {92 , 81 , 70 , 69 , 58}
  int  k ;
  for (k=0 ; k<5 ; k++)
     printf (" %d\n ", *(nums + k)) ;
  }
```

شکل زیر 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 برابر ۲ بایت می‌باشد . 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.

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 یک عنصر از آرایه وجود دارد . یکی بصورت nums+k در فرم اشاره‌گر و دیگری بصورت nums[k] در فرم آرایه .

حال اجازه دهید با ارائه یک برنامه ساده ، رابطه بین عناصر آرایه و آدرس آنها را بررسی و ملاحظه نماییم .

مثال – برنامه 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]
```

و یا :

```
*( x + i )
```

معرفی نمود . 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 بکار برد . اصولاً همه جا می‌توانیم به جای x[i] هم‌ارز آن *(x+i) را بکار ببریم . به هرحال عباراتی مشابه :

```
x  و  x + i  و  &x[i]
```

که معرف آدرس هستند ، نمی‌توانند در سمت چپ دستور جایگذاری بکار برده شوند . همچنین آدرس یک آرایه نمی‌تواند بطور دلخواه تغییر یابد . بنابراین عبارتی مشابه x++ مجاز نمی‌باشد .

قبلاً بیان شد که نام آرایه ، بطور واقعی یک اشاره‌گر به اولین عنصر آرایه است . در نتیجه باید امکان داشته باشد که یک آرایه را بجای روش قراردادی متداول ، بعنوان یک متغیر اشاره‌گر تعریف کرد .

به هرحال تعریف آرایه به روش قراردادی ، موجب می‌گردد که یک بلوک ثابت از حافظه ، در آغاز اجرای برنامه رزرو شود . ولی اگر آرایه برحسب یک متغیر اشاره‌گر توصیف شود ، با این عمل رزرو کردن جا ، اتفاق نمی‌افتد . درنتیجه موقع استفاده از اشاره‌گر برای معرفی یک 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 اولیه 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 . بعضی از این گونه کاربردها ، طی مثالهایی در ادامهٔ این فصل ارائه می‌گردد .

اگر آرایه بصورت یک متغیر اشاره‌گر تعریف گردد ، نمی‌توان به عناصر آرایه‌های عددی ، مقدار اولیه نسبت داد . در نتیجه این گونه موارد نیاز به تعریف آرایه بصورت روش عادی و قراردادی دارد .

مثال – اگر بخواهیم a را بصورت یک آرایه ۱۰ عنصری از مقادیر صحیح تعریف کنیم ، می‌توان آن را به جای :

```
int  a[10] ;
```

بصورت :

```
int *a ;
```

نوشت . یعنی a را بعنوان متغیر اشاره‌گر تعریف کرد .

به هرحال در روش دوم ، به a بعنوان آرایه ۱۰ عنصری یک بلوک حافظه اختصاص داده نمی‌شود ، بلکه a بصورت یک متغیر اشاره‌گر تعریف شده است .

برای اختصاص حافظه مورد نیاز به a جهت معرفی یک آرایهٔ ۱۰ عنصری ، می‌توان از تابع malloc بصورت زیر استفاده نمود :

```
a = malloc (10 * sizeof(int)) ;
```

این تابع یک بلوک حافظه 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 ، بصورت 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 ، معادل هستند . لذا دستور بالا قابل قبول است . اگرچه از لحاظ اطمینان کامل می‌توان از تبدیل نوع cast استفاده کرد و آن را بصورت زیر بکار برد :

```
a = (int *) malloc (10 * sizeof (int)) ;
```

این گونه اختصاص حافظه به روش اختصاص حافظه بصورت پویا موسوم است .

به هرحال اگر قرار باشد به عناصر آرایه ، مقدار اولیه نیز اختصاص یابد ، باید a بجای متغیر اشاره‌گر بصورت آرایه توصیف گردد مشابه زیر :

```
int  a[10] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10} ;
```

یا :

```
int  [ ] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10}
```

در برنامه‌نویسی با C ، ممکن است برای مراجعه به عناصر آرایه بجای روش معمول ، عباراتی برحسب اشاره‌گرها بکار ببریم . این روش در آغاز کار کمی غیرطبیعی به نظر می‌آید ، ولی می‌توان با کمی تمرین ، به سادگی تجربه لازم را در این مورد بدست آورد . در مثال زیر ضمن اینکه روش استفاده از تابع malloc نشان داده می‌شود ، به عناصر آرایه نیز با استفاده از این تکنیک دستیابی می‌گردد .

مثال – مرتب کردن یک مجموعه اعداد (Sort) .

برنامه مورد نظر در زیر نشان داده شده است :

```
# 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 ، بصورت یک اشاره‌گر به یک مقدار صحیح تعریف شده است . در آغاز به کمک تابع کتابخانه‌ای malloc به متغیر اشاره‌گر ، حافظه ، اختصاص داده شده است . (یعنی حافظه 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 بجای :

&a[k]

از :

a + k

استفاده شده است . به طریق مشابه ، در تابع printf برای معرفی مقدار k اُمین عنصر ، بجای :

a[k]

از :

*(a + k)

استفاده شده است . ملاحظه می‌شود که در تابع فرعی نیز آرگومان آن بجای آرایه ، متغیر اشاره تعریف شده است .

اشاره‌گرها و آرایه‌های چند بعدی

دیدیم که عناصر یک آرایهٔ یک بعدی می‌تواند برحسب یک اشاره‌گر (نام آرایه) و یک مقدار بعنوان 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 بصورت :

*(a + 5)

مراجعه کرد که در آن مقدار آفست همان 5 است که به نام آرایه افزوده شده است و به کمک اپراتور * به مقدار آن دسترسی پیدا می‌کنیم .

حال می‌توان گفت که یک آرایه نیز مجموعه‌ای از آرایه‌های یک‌بعدی است . بنابراین می‌توان یک آرایه دو بعدی را بصورت یک اشاره‌گر به یک گروه پیوسته و مجاور هم از آرایه‌های یک بعدی تعریف کرد . درنتیجه می‌توان توصیف یک آرایه دو بعدی را بجای :

```
data-type   array[d1][d2] ;
```

(که در آن منظور از d1 ، d2 به‌ترتیب بزرگی ابعاد اول و دوم آرایه است) بصورت 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] ;
```

بصورت زیر نوشت :

```
data-type   (*ptvar)[d2][d3]…[dn] ;
```

که در آنها ، data-type ، نوع عناصر آرایه و array نیز نام آرایه است . و عناصر :

d1 , d2 , ..., dn

نیز به ترتیب ماکزیمم عناصر هر اندیس یا هر بعد آرایه می‌باشند . درضمن توجه کنید که ptvar نیز نام متغیر اشاره‌گر است .

- **انتقال آرایه به تابع (بعنوان آرگومان)**

بطوری که در فصل آرایه‌ها بیان شد ، در زبان C ، نام هر آرایه‌ای که بعنوان آرگومان یک تابع بکار برده شود ، بعنوان آدرس اولین عنصر آرایه تفسیر می‌گردد . برای مثال برنامه 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 ما آرگومان را بعنوان اشاره‌گر به اولین عنصر آرایه توصیف کنیم . برای این کار ، دو راه بصورت زیر وجود دارد :

راه دوم	راه اول
func(ar) float ar[ ] ; { .... }	**func(ar)** float *ar ; { .... }

راه دوم ، ar را به‌عنوان آرایه‌ای با اندازه (یا بزرگی) نامشخص ، توصیف می‌کند . آرایه هم‌اکنون در تابع اصلی ایجاد شده است ، آنچه گذر داده می‌شود ، یک اشاره‌گر به اولین عنصر از آرایه است . چون کامپایلر می‌داند که عبارت آرایه منتج به اشاره‌گر به اولین عنصر آرایه می‌گردد ، پس ar را مشابه توصیف ar در روش اول ، به یک اشاره‌گر از نوع float تبدیل می‌کند . بنابراین هر دو گونه ازنظر نحوهٔ عملکرد ، معادل و هم‌ارز یکدیگر می‌باشند .

به هرحال ازنظر واضح‌تر بودن ، ممکن است روش دوم ترجیح داده شود . زیرا این روش تأکید می‌کند که آنچه که باید گذر داده شود آدرس پایه یا آدرس اولین عنصر یک آرایه است . در روش اول ، راهی برای تشخیص اینکه آیا ar به آغاز یک آرایه از نوع float و یا تنها به یک عنصر از نوع float اشاره می‌کند یا نه ، وجود ندارد .

- **مقایسه اشاره‌گرها**

دو اشاره‌گر را می‌توان در یک عبارت رابطه‌ای با یکدیگر مقایسه کرد . برای مثال اگر q و p دو اشاره‌گر باشند ، دستور زیر یک دستور درست می‌باشد :

```
if  (p<q)
  printf ("p  points to lower memory than  q") ;
else
    printf ("q  points to lower memory than  p") ;
```

- **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 اشاره‌گرهایی هستند که می‌توانند آدرس متغیرهایی از نوع مقادیر صحیح را در خود داشته باشند. بعنوان مثال برای اختصاص دادن آدرس متغیری به نام z به عنصر سوم آرایه مزبور ، می‌نویسیم :

```
*x[2] = &z ;
```

همینطور برای بدست آوردن مقدار z از دستور **x[2] 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 بعنوان آرگومان تابع قرار می‌دهیم . برای مثال تابع display می‌تواند آرایه x را بصورت زیر دریافت نماید :

```
void  display (int  *a[ ] )
{
   int  k ;
   for ( k=0 ; k<10 ; k+ +)
      printf (" %p" , *a[k] ) ;
}
```

توجه داشته باشید که در مثال بالا ، a یک اشاره‌گر به مقادیر صحیح نیست بلکه یک اشاره‌گر به آرایه‌ای از اشاره‌گرهایی به مقادیر صحیح است . بنابراین نیاز است که پارامتر a بعنوان آرایه‌ای از اشاره‌گرهایی به مقادیر صحیح به همان طریق که نشان داده شد ، توصیف شود .

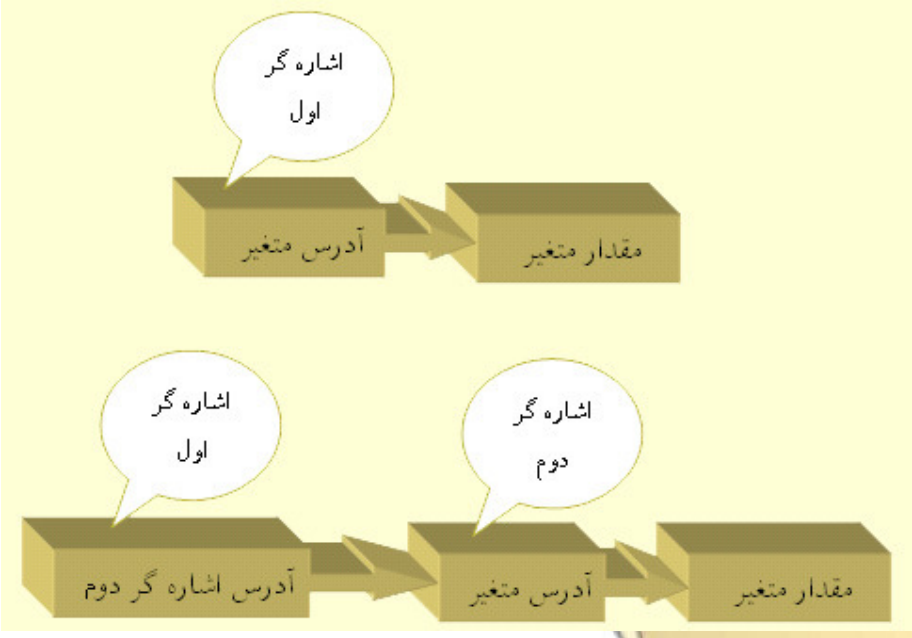
آرایه‌های اشاره‌گر اغلب برای نگهداری اشاره‌گرهایی به رشته‌ها بکار برده می‌شوند .

- **اشاره‌گر به اشاره‌گر**

بطوری که در گذشته بیان شد ، اگر متغیری آدرس متغیر دیگری را در خود نگه دارد آن را اشاره‌گر نامند . حال اگر متغیر دوم نیز از نوع اشاره‌گر باشد در این صورت متغیر اول یک اشاره‌گر به اشاره‌گر است .چنین موقعیتی را اشاره‌گر به اشاره‌گر نامند . گاهی ممکن است تصور و فهم اشاره‌گر به اشاره‌گر برای دانشجویان ایجاد اشکال کند .

شکل زیر مفهوم اشاره‌گر به متغیر عادی و اشاره‌گر به اشاره‌گر را روشن می‌کند :

نمایش اشاره‌گر به متغیر عادی و اشاره‌گر به اشاره‌گر



همانطور که ملاحظه می‌کنید ، مقدار یک اشاره‌گر معمولی ، آدرس یک 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 یک اشاره‌گر به اشاره‌گر از نوع float است :

```
float  **z ;
```

به هرحال باید توجه داشته باشید که z یک اشاره‌گر به یک مقدار اعشار نیست ؛ بلکه یک اشاره‌گر به اشاره‌گری 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 بکار 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 به‌عنوان اشاره‌گر به یک متغیر int و q نیز به‌عنوان اشاره‌گر به یک اشاره‌گری که می‌تواند آدرس متغیری از نوع int را داشته باشد ، توصیف شده است . حال نتیجه 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 .

باتوجه به مباحث این فصل می‌توان نتایج زیر را در مورد اشاره‌گرها بیان کرد :

۱ ـ دو عملگر یا اپراتور در مورد اشاره‌گرها مورد استفاده قرار می‌گیرند که عبارتند از : * و & . عملگر & یک عملگر یکانی است که آدرس عملوند یا اپراند خود را مشخص می‌کند . عملگر * نیز یک عملگر یکانی است که محتویات یک آدرس حافظه را مشخص می‌کند . بنابراین عملگر * مکمل عملگر & می‌باشد .

۲ ـ اعمال متداول روی اشاره‌گرها عبارتند از :

الف) عمل انتساب یا جایگذاری

ب) اعمال محاسباتی که شامل : عمل جمع ، عمل تفریق ، عمل + + و عمل - - است .

پ) مقایسه اشاره‌گرها

۳ ـ اشاره‌گرها دارای ویژگیهای زیر می‌باشند :

الف) عمل تخصیص حافظه به‌صورت پویا را امکان‌پذیر می‌کنند .

ب) کار با آرایه‌ها و رشته‌ها را آسان می‌کنند .

پ) موجب بهبود کارآیی بسیاری از توابع می‌گردند .

ت) فراخوانی با آدرس را در مورد توابع امکان‌پذیر می‌سازند ؛ درنتیجه برگردان بیش از یک مقدار ، از یک تابع ، میسر می‌گردد .

ث) ایجاد ساختارهای پیچیده‌تر مثل لیستهای پیوندی ، درختهای دودویی ، گرافها و ... را به راحتی امکان‌پذیر می‌سازند .

- **تمرین و پاسخ**

تمرین۱ ـ 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 شکل زیر دو رشته را از لحاظ یکسان بودن با یکدیگر مقایسه می‌کند که اگر یکسان نبودند true **و در غیر اینصورت** false **برمی‌گرداند . در واقع نقش تابع کتابخانه‌ای** strcmp **را بازی می‌کند.**

```
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 خانه دو آرایه مربوط به دو رشته مورد نظر را دارند . در دستور if محتوای این دو خانه با یکدیگر مقایسه می‌شوند . چنانچه یکسان نباشند ، تفاضل آن دو (یعنی تفاضل آسکی‌کد کاراکترهای محتوای این دو خانه) مساوی صفر نخواهد بود . پس دو رشته یکسان نیستند . درنتیجه ، تابع مزبور یک عدد غیرصفر (تفاضل s1 و s2) را که معرف true می‌باشد برمی‌گرداند ، و در غیر

اینصورت دستورهای مربوط به else اجرا می‌شود و مقدار هر دو اشاره‌گر یک واحد افزایش می‌یابد و در نتیجه آدرس کاراکتر بعدی از دو رشته را خواهند داشت . حال اگر حلقه while کامل گردد ، یعنی عمل مقایسه کاراکترهای دو رشته تا آخرین کاراکتر آنها که '\0` می‌باشد ادامه یابد ، پس دو رشته یکسان هستند و تابع مزبور '\0` را که آسکی‌کد آن صفر است بعنوان معرف false برمی‌گرداند (زیرا بطوری که در فصول پیش بیان شد در زبان C ، مقدار صفر ، معرف false و غیرصفر معرف true می‌باشد .)

تمرین۲ ـ تابعی بنویسید که با استفاده از اشاره‌گر ، طول یک رشته را بدست آورد .

حل : تابع مورد نظر در زیر نشان داده شده است :

```
Strcmp (char  *s1 , char  *s2)
 {
   while (*s1)
      if (*s1 - *s2)
         return  *s1 - *s2 ;
      else
         {  s1 + + ;
           s2 + + ;
         }
   return `\0´ ;
 }
```

تمرین ۳ ـ تابعی بنویسید که با استفاده از اشاره‌گر ، دو رشته را باهم مجاورسازی کند ؛ یعنی دو رشته را به هم متصل کرده ، رشته سومی تشکیل دهد .

حل : تابع مورد نظر در زیر نشان داده شده است :

```
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 اشاره‌گر هستند که به ترتیب آدرس اولین بایت و رشته s1 و s2 و همچنین رشته سوم که از الحاق آن دو بدست خواهد آمد ، دارند . در داخل حلقه while اول ، محتوای رشته اول تا موقعی که انتهای رشته (یعنی کاراکتر'\0`) ظاهر نشده است ، در رشته سوم که با s3 به آن اشاره شده است ، قرار می‌گیرد . سپس در حلقه while دوم ، محتوای رشته دوم بدنبال آن قرار می‌گیرد . در پایان ، علامت پایان رشته ، یعنی '\0` در انتهای آن قرار می‌گیرد .

تمرین ٤ ـ برنامه‌ای بنویسید که با استفاده از اشاره‌گر و روش اختصاص حافظه به‌صورت پویا ، حافظه‌ای برای یک آرایه n عنصری اختصاص دهد و عناصر آرایه را به حافظه بخواند .

سپس با فراخوانده شدن یک تابع ، عناصر آرایه مزبور با بکارگیری اشاره‌گر ، به‌صورت صعودی مرتب گردد و نتیجه در تابع اصلی چاپ شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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 یک اشاره‌گر به مقادیر صحیح ، تعریف شده است . حافظه لازم برای عناصر آن از طریق تابع کتابخانه‌ای malloc بصورت پویا از سیستم اخذ شده و آدرس آغاز آن در یک 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 ، جابجایی) 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 ، بجای :

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 .

تمرین ۵ ـ برنامه‌ای بنویسید که عناصر دو ماتریس (آرایه دوبعدی) را به حافظه بخواند و مجموع آن دو را براساس قانون جمع ماتریسها ، در آرایه C قرار داده و نتیجه را به‌صورت ماتریس چاپ کند .

حل : این برنامه در گذشته در فصل آرایه‌ها نوشته شده است . در اینجا هدف آن است که هر کدام از آرایه‌های دوبعدی به‌صورت یک اشاره‌گر به یک مجموعه آرایه 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 یک تابع فرعی انجام می‌گیرد .

برنامه مورد نظر در زیر نشان داده شده است :

```
# include <stdio.h>
# define  maxcols  30
main ( )
{
  int  nrows , ncols ;
  int (*a)[maxclos] , (*b)[maxcols] , (*c)[maxcols] ;     /* pointer definations */
  /* 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 به‌عنوان اشاره‌گرهایی به گروهی از آرایه‌های یک بعدی پیوسته (مجاور هم) که بزرگی همه آنها maxcols می‌باشد ، تعریف شده‌اند . 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 به‌عنوان 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 آرایه اول اشاره می‌کند . بطریق مشابه ، *(a+1) به اولین عنصر در آرایه دوم و همینطور *(a+2) به اولین عنصر در آرایه سوم اشاره می‌کند. بنابراین یک بلوک حافظه که بزرگی آن :

```
nrows * ncols
```

برای مقادیر صحیح است. با شروع از اولین عنصر آرایه ، اختصاص می‌یابد . نحوه عمل ، برای دو آرایه دیگر نیز به همین طریق است .

هر یک از عناصر آرایه با استفاده از عملگر غیرمستقیم بصورت تکراری مورد پردازش قرار می‌گیرد . برای مثال در تابع readinput به هر عنصر تابع بصورت زیر مراجعه می‌شود :

```
scanf("%d" , (*(a+row)+col )) ;
```

بطریق مشابه عمل جمع کردن عناصر متناظر دو آرایه در تابع computesums بصورت زیر نوشته می‌شود :

```
*(*(c+row) + col ) = *(*(a+row) + col ) + *(*(b+row) + col ) ;
```

همچنین اولین دستور printf در تابع writeoutput بصورت 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 بکار بریم :

```
scanf("%d" , &a[row][col] ) ;
```

بطریق مشابه در تابع computesums می‌توانیم به جای دستور :

```
*(*(c+row)+col ) = *(*(a+row)+col) + *(*(b+row)+col ) ;
```

دستور زیر را بکار بریم :

```
c[row][col] = a[row][col] + b[row][col] ;
```

همینطور در تابع writeoutput می‌توانیم به جای دستور :

```
printf("%4d" , *(*(a+row) + col )) ;
```

دستور زیر را بکار بریم :

```
printf ("%4d" , a[row][col] ) ;
```

تمرین ۶ ـ یک راه برای مرتب کردن رشته‌ها با استفاده از اشاره‌گرها ، آن است که

آدرس رشته‌ها را در آرایه‌ای از اشاره‌گرها قرار دهیم . سپس در مقایسه دو رشته ، اگر نیاز به جابجایی آن دو با یکدیگر باشد ، به جای این کار آدرس دو رشته را در درون آرایه اشاره‌گرها که آدرس رشته‌ها را دارد ، با یکدیگر عوض نماییم .

حل : برنامه زیر لیستی 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 رشته "end" مشخص شده است . یعنی هر موقع عمل ورود یا خوانده شدن رشته‌ها تمام شد ، کلمهٔ "end" را بعنوان پایان ورود داده‌ها ، وارد می‌کنیم . آرایه رشته‌ای x[] برای نگهداری آدرس رشته‌ها منظور شده است . رشته‌ها در تابع اصلی خوانده می‌شود که برای این کار با استفاده از تابع malloc حافظه مورد نیاز بصورت پویا اختصاص داده می‌شود. سپس با فراخوانده شدن تابع فرعی reorder (که آرگومانهای آن تعداد رشته‌ها و آرایه آدرس رشته ها می‌باشد) ، رشته‌های مورد نظر به ترتیب الفبا مرتب می‌گردد و 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 ;
}
```

تمرین ۷ ـ تابعی بنویسید که مقادیر دو متغیر را با استفاده از اشاره گر تعویض کند .

حل : تابع مورد نظر در زیر نشان داده شده است :

```
void swap (int  *a , int  *b)
  {
    int temp ;
    temp =  *a ;
    *a = *b ;
    *b = temp ;
}
```

تمرین ۸ ـ برنامه‌ای بنویسید که یک رشته و کاراکتری را از ورودی بخواند . سپس با فراخواندن یک تابع فرعی ، تعداد دفعاتی را که کاراکتر مورد نظر در رشته مزبور وجود داشته باشد ، بشمارد و چاپ کند .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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( ) ;
 }
```

تمرین ۹ ـ برای آشنایی بیشتر با نحوه اعلان یا توصیف متغیرها در اشاره گرها ، در جدول زیر مثالهایی با توضیح آنها ارائه شده که به ترتیب از ساده به حالت پیچیده‌تر تنظیم شده‌اند .

int *p ;	P ، یک اشاره‌گر به یک مقدار صحیح است .

int *p[10] ;	P ، یک آرایه ۱۰ عنصری از اشاره‌گرها به مقادیر صحیح است .
int (*p)[10] ;	P ، یک اشاره‌گر به یک آرایه ۱۰ عنصری با مقادیر صحیح است .
int *p(void) ;	P ، یک تابع است که یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند
int p(char *a) ;	P ، یک تابع است که آرگومانی را که یک اشاره‌گر به یک کاراکتر است ، می‌پذیرد و یک مقدار صحیح برمی‌گرداند .
int *p(char *a) ;	P ، یک تابع است که یک آرگومان که یک اشاره‌گر به یک کاراکتر است ، می‌پذیرد و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند
int (*p) (char *a) ;	P ، یک اشاره‌گر به یک تابع است که یک آرگومان را که آن هم یک اشاره‌گر به یک کاراکتر است ، می‌پذیرد و یک مقدار صحیح برمی‌گرداند .
int (*p(char *a))[10] ;	P ، یک تابع است که یک آرگومان را که آن هم یک اشاره‌گر به یک کاراکتراست ، می‌پذیرد و یک اشاره‌گر به یک آرایه ۱۰ عنصری از نوع مقادیر صحیح ، برمی‌گرداند .
int p(char(*a)[ ]) ;	P ، یک تابع است که یک آرگومان را که آن هم یک اشاره‌گر به یک آرایه کاراکتری است ، می‌پذیرد و یک مقدار صحیح برمی‌گرداند .
int p(char *a[ ]) ;	P ، یک تابع است که یک آرگومان را که آن هم آرایه‌ای از اشاره‌گرها ه کاراکترها می‌باشد ، می‌پذیرد و یک مقدار صحیح برمی‌گرداند .
int *p(char a[ ]) ;	P ، یک تابع است که یک آرگومان را که یک آرایه کاراکتری است ، می‌پذیرد و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .
int *p(char (*a)[ ]) ;	P ، یک تابع است که یک آرگومان را که آن هم یک اشاره‌گر به یک آرایه کاراکتری است ، می‌پذیرد و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .
int *p(char *a[ ]) ;	P ، یک تابع است که یک آرگومان که آن هم یک آرایه از اشاره‌گرها به کاراکترها می‌باشد ، می‌پذیرد و یک اشاره‌گر به یک مقدار صحیح

برمی‌گرداند .	
P ، یک اشاره‌گر به یک تابع است که آن هم یک آرگومان می‌پذیرد که یک اشاره‌گر به یک آرایه کاراکتری است و یک مقدار صحیح برمی‌گرداند .	int (*p)(char (*a)[ ]) ;
P ، یک اشاره‌گر به یک تابع است که آن هم یک آرگومان می‌پذیرد که یک اشاره‌گر به یک آرایه کاراکتری است و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .	int *(*p)(char (*a)[ ]) ;
P ، یک اشاره‌گر به یک تابع است که آن هم یک آرگومان را می‌پذیرد که آرایه‌ای از اشاره‌گرها به کاراکترها می‌باشد و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .	int *(*p)(char *a[ ]) ;
P ، یک آرایه ۱۰ عنصری از اشاره‌گرهایی به توابع می‌باشد که هرکدام از آن توابع یک مقدار صحیح برمی‌گرداند .	int (*p[10])(void) ;
P ، یک آرایه ۱۰ عنصری از اشاره‌گرهایی به توابع است که هر کدام از آن توابع یک آرگومان که یک کاراکتر می‌باشد ، می‌پذیرند و یک مقدار صحیح برمی‌گرداند .	int (*p[10])(char a) ;
P ، یک آرایه ۱۰ عنصری از اشاره‌گرهایی به توابع است که هرکدام از آن توابع یک آرگومان که یک کاراکتر است ، می‌پذیرد ، و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .	int *(*p[10])(char a) ;
P ، یک آرایه ۱۰ عنصری از اشاره‌گرهایی به توابع است که هرکدام از آن توابع یک آرگومان که اشاره‌گری به یک کاراکتر می‌باشد ، می‌پذیرد ، و یک اشاره‌گر به یک مقدار صحیح برمی‌گرداند .	int *(*p[10])(char *a) ;

فصل نهم – نوعهای تعریف شده

- مقدمه

اغلب ، در کاربردهای مختلف با مجموعه‌ای از داده‌ها که صفات یکسان ندارند ، مواجه هستیم که در این حالت ، آرایه‌ها قابل استفاده نیستند. برای رفع مشکل در این گونه موارد زبان C ، اجازه می‌دهد که کاربر به پنج طریق نوع داده‌هایی با سلیقه خود ایجاد نماید که عبارتند از :

۱ ـ **ساختار** (**structure**) : عبارت است از دسته‌بندی متغیرهایی با صفات مختلف تحت یک نام.

۲ ـ **bit field :** گونهٔ خاصی از ساختار است و این امکان را فراهم می‌سازد که به سادگی بتوانیم به بیتهای درون یک کلمه نیز دستیابی داشته باشیم .

۳ ـ **اجتماع** (**union**) **:** بوسیله آن میتوان قسمتی از حافظه را برای استقرار دو یا چندین نوع متفاوت از داده‌ها تعریف کرد .

٤ ـ **شمارشی** (**enumeration**) **:** لیستی از نشانه‌ها یا سمبولها است که می‌توان با مقادیر صحیح شمارشی ۱ ، ۲ و ... به آنها مراجعه کرد .

۵ ـ **typedef :** نام جدیدی برای یک نوع (type) موجود ایجاد می‌کند .

- **ساختار (Structure)**

یک ساختار ، مجموعه‌ای از متغیرها می‌باشد که تحت یک نام به آنها مراجعه می‌گردد . این شیوه ، وسیلهٔ ساده‌ای برای یکپارچه ساختن مجموعه‌ای از داده‌ها یا اطلاعات مربوط به هم ، در اختیار ما قرار می‌دهد . بعنوان مثال فرض کنید که از هر دانشجوی کلاسی شماره دانشجویی ، نام و نمره امتحان وی با فرمت زیر مورد نیاز است :

st-no	name	grade

در اینجا نمی‌توان این ۳ فیلد را در یک آرایه ۳ عنصری قرار داد . زیرا فیلد اول از نوع int ، فیلد دوم از نوع رشته (آرایه کاراکتری) و فیلد سوم از نوع float می‌باشد . یعنی صفات آنها یکسان نیستند . حال خواسته آن است که بتوان به این مجموعه ۳ عنصری ، تحت یک نام مراجعه کرد و اگر نیاز باشد بتوان به عناصر یا فیلدهای آن نیز مراجعه نمود . این گونه مجموعه را در زبان پاسکال و کوبول بعنوان رکورد یا record و در زبان C به‌عنوان ساختار یا structure تعریف می‌کنند . اگر این مجموع را rec بنامیم ، نحوه تعریف آن در زبان C ، به‌صورت زیر خواهد بود :

```
struct   rec
   {
      int  st-no ;
      char  name[15] ;
      float  grade ;
   } ;
```

در اینجا کلمه کلیدی struct برای تعریف داده از نوع ساختار بکار رفته که پس از آن نام انتخابی برای مجموعهٔ مورد نظر بکار برده شده و سپس در داخل یک زوج آکولاد ، عناصر یا اعضای

مجموعهٔ مورد نظر تعریف شده‌اند و در پایان نیز سمی‌کولون `:´ به‌عنوان پایان تعریف ساختار بکار 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 ;
```

درواقع با این تعریف ، به کامپایلر می‌گوییم که متغیرهای x و z از نوع struct می‌باشد که تحت نام rec تعریف شده است .

وقتی که ما یک ساختار را تعریف می‌کنیم ، درواقع یک نوع پیچیده ، مرکب از عناصر ساختار را تعریف می‌کنیم ؛ بنابراین در مثال بالا ، کلمهٔ rec ، یک متغیر نیست بلکه معرف نوع داده است ، ولی کلمات x و z متغیرهایی هستند که نوع آنها از نوع rec می‌باشد .

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 می‌باشد ، آمده است . لذا دیگر نیازی به معرفی آنها با دستور :

```
struct   rec
```

نمی‌باشد .

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 نمود . 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 :

member1 , member2 , ...member*m*

نشان دهیم ، فرم کلی ترکیب یک ساختار را می‌توان بصورت زیر تعریف کرد :

```
struct  structure-name
        {
type  variable1-name ;
type  variable2-name ;
        ....
        ....
type  variablem-name ;
        } ;
```

```
struct  tag
        {
member1 ;
member2 ;
        ....
        ....
member m ;
        } ;
```

که در آن struct ، کلمهٔ کلیدی برای تعریف داده از نوع ساختار (structure) و tag ، نام این ساختار داده‌ها و همچنین :

member1 , member2 , … , member***m***

نیز معرف توصیف اعضای ساختار مزبور می‌باشند .

هریک از اعضای ساختار می‌تواند متغیر معمولی ، اشاره‌گر ، آرایه ، یا حتی ساختار دیگری باشد .

با اینکه نام اعضای یک ساختار می‌تواند با متغیرهای دیگری که در جایی دیگر از برنامه تعریف شده‌اند ، همنام باشد ، ولی گویاتر آن است که در اینگونه موارد ، اسامی همنام بکار نبریم .

به اعضای یک ساختار نمی‌توان کلاس حافظه اختصاص داد و همچنین نمی‌توان هنگام تعریف یک ساختار ، به اعضای آن مقدار اولیه نسبت داد .

وقتی ترکیب ساختار یک بار تعریف گردید ، می‌توان متغیرهایی از نوع ساختار مزبور بصورت زیر توصیف کرد :

storage-class struct tag variable1 , variable 2 , … , variable *m* ;

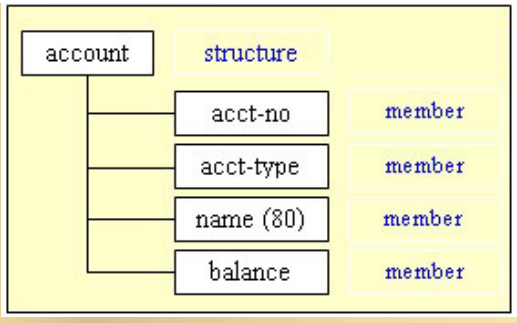
که در آن storage-class یک مشخص‌کننده کلاس حافظه است که بکار بردن آن اختیاری می‌باشد ؛ ولی کلمه کلیدی struct الزامی است و tag نیز نام ساختار است و متغیرهای :

variable1 , variable 2 , … variable *m*

نیز متغیرهایی از نوع ساختار tag می‌باشند .

مثال – در شکل زیر یک نمونه از تعریف ساختار ، همراه با شکل یا ترکیب شمای 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 که شامل ٤ فیلد یا عنصر می‌باشد که عبارتند از :

یک مقدار صحیح acct-no بعنوان شماره حساب .

یک تک‌کاراکتر acct-type بعنوان نوع حساب .

یک آرایه کاراکتری یا رشته name[80] بعنوان نام صاحب حساب .

یک کمیت یا مقدار balance بعنوان 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 ساختار هستند که ترکیب عناصر آنها با ساختار account مشخص شده است .

بطوری که بیان شد می‌توان توصیف ساختار را با متغیرهایی از نوع ساختار مزبور ترکیب کرد و آنها را یکجا ، بصورت زیر تعریف نمود :

```
storage-class   struct   tag {
                        member1 ;
                        member2 ;
                        . . .
                        member m ;
                       } variable1 , variable2 , . . . , variable m ;
```

در چنین موقعیتی ، انتخاب نام برای ساختار یعنی بکار بردن tag اختیاری است .

مثال – توصیف زیر ، معادل دو توصیف مثال قبل 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 بکار نبرد . (ولی بطوری که در گذشته بیان شد ، این شیوه توصیه نمی‌گردد) .

یک ساختار ممکن است به‌عنوان عضوی از ساختار دیگر تعریف گردد . در چنین حالتی باید تعریف ساختار مُحاط (یعنی ساختار بکار رفته در درون ساختار دیگر) قبل از تعریف ساختار بیرونی ظاهر گردد .

مثال – یک برنامه C ممکن است شامل تعریف های زیر باشد که به همراه شمای آن در زیر 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 می‌گردد که ساختار account شامل ساختار دیگری ، یعنی date بعنوان یکی از اعضا یا فیلدهای خود می‌باشد . در چنین موقعیتی باید تعریف date قبل از تعریف account بیاید

- **اختصاص مقادیر اولیه**

می‌توان به اعضا یا فیلدهای متغیر ساختار ، مشابه روشی که در مورد عناصر آرایه ملاحظه گردید ، 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 . فرم کلی آن بصورت زیر است :

storage-class struct tag variable = {value1 , value2 , … , value***m***} ;

که در آن value1 , value2 , ...valuem معرف مقادیری است که باید به ترتیب به عناصر اول ، دوم ، … و m اُم متغیر ساختار اختصاص یابد .

مثال – این مثال نحوه اختصاص اولیه به اعضای یک 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 = {12345 , `R´ , "payam noor" , 5586.50 , 5 , 24 , 75} ;
```

بنابراین ، customer یک متغیر ساختاری استاتیک از نوع account است که به عناصر آن مقادیر اولیه اختصاص داده شده است . به اولین عنصر عدد صحیح 12345 ، به دومین عنصر کاراکتر `R´ ، به سومین عنصر رشته :

" payam noor "

و به چهارمین عنصر یا balance مقدار اعشاری 5586.50 اختصاص داده شده است . آخرین عنصر آن ، یک ساختار است که شامل سه مقدار صحیح (از چپ به راست ، معرف : ماه ، روز و سال) است . بنابراین به آخرین عضو customer مقادیر صحیح :

5 , 24 , 75

اختصاص داده شده است .

- **آرایه‌ای از ساختارها**

می‌توان آرایه‌ای از ساختارها تعریف کرد . یعنی آرایه‌ای تعریف کرد که هر عنصر آن یک ساختار باشد که شاید این شیوه ، متداول‌ترین کاربرد ساختارها باشد . برای توصیف آرایه‌ای از ساختارها ، باید اول ساختار مورد نظر را توصیف کرد . سپس متغیری از نوع آرایه توصیف نمود که عناصر آن از نوع ساختار مزبور باشد .

مثال – آرایه‌ای ۱۰۰ عنصری به نام customer تعریف کنید که عناصر آن ساختاری از نوع مثال قبل یعنی از نوع accont باشد .

حل : تعریف آرایه مورد نظر به دو صورت زیر خواهد بود :

روش اول

```
struct  date {
        int month
        int day ;
        int year ;
} ;
struct  account {
        int  acct-no ;
        int  acct-type ;
        char name[80] ;
```

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] ;
```

```
    float balance ;
    struct date lastpayment ;
} ;
struct  account  customer[100] ;
```

در این توصیف ، customer یک آرایه ۱۰۰ عنصری است که هر عنصر آن ساختاری از نوع account است . درضمن هر دو روش توصیف بالا هم‌ارز یکدیگر می‌باشد .

به یک آرایه از ساختارها نیز می‌توان مقادیر اولیه اختصاص داد . به‌خاطر داشته باشید که در اینجا عنصر آرایه ، یک ساختار است که باید مقادیر اولیه متناظر با هر عنصر ، به اعضای آن اختصاص داده شود . مثال زیر نحوه عمل را در این مورد نشان می‌دهد .

مثال – یک برنامه ، 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 ساختارها است که اندازه آن مشخص نشده است . مقادیر اولیه ، اندازه آرایه و حافظه لازم را تعریف می‌کند ؛ لذا اندازهٔ آن برابر ۷ خواهد بود ، زیرا ۷ سطر از مقادیر اولیه منظور شده است که از صفر تا ۶ 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 آرایه مزبور با این شیوه ، بصورت 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 یک ساختار**

بطور معمول هر عضو یک ساختار بعنوان یک هویت مستقل ، بصورت جداگانه مورد پردازش قرار می‌گیرد . به هر عضو یا عنصر ساختار با استفاده از عملگر `.` یا عملگر عضویت (که گاهی آن را عملگر یا اپراتور period یا "dot" نیز نامند) دستیابی یا رجوع می‌شود . با استفاده از این عملگر ، به هر عنصر یک ساختار بصورت زیر دستیابی می‌شود :

structure-name . element-name

یا به عبارت دیگر :

variable . member

اپراتور نقطه نسبت به سایر اپراتورها ، حتی اپراتورهای unary ، تقدم بالایی دارد . برای مثال دو عبارت :

++ (variable . member) و ++ variable . member

معادل هم می‌باشند . یعنی اپراتور ++ به عضو ساختار (structure member) عمل خواهد کرد نه به تمام متغیر ساختار . به طریق مشابه ، دو عبارت :

&variable . member و &(variable . member)

معادل یکدیگر می‌باشند . بنابراین هر دو عبارت مزبور به آدرس عضو ساختار دستیابی می‌کند ، نه به آدرس متغیر ساختار .

حال برای اینکه با مفهوم پردازش ساختارها بیشتر آشنا شوید ، مثالی به شرح زیر ارائه می‌گردد .

مثال – برنامه‌ای بنویسید که عملیات زیر را انجام دهد :

۱ ـ مشخصات n نفر دانشجویان کلاس را که فرمت آن به‌صورت زیر است ، به آرایه‌ای از ساختار بخواند و سپس سابقه هر دانشجو را در یک سطر جداگانه چاپ کند . n ، حداکثر ۲۵ می‌باشد که از طریق دستگاه ورودی استاندارد دریافت می‌شود .

۲ ـ شماره دانشجویی را از طریق ورودی بخواند و در لیست دانشجویان جستجو کند . اگر وجود داشت بقیه مشخصات وی چاپ شود ، وگرنه ، پیغام "not found" چاپ گردد .

۳ ـ سوابق دانشجویان برحسب شماره دانشجویی بصورت صعودی مرتب شود و سپس سابقه هر دانشجو در یک سطر جداگانه چاپ گردد .

st-nd	name		grade
	l-name	f-name	

متغیرهای st-no ، l-name ، f-name و grade به‌ترتیب معرف شماره دانشجویی ، نام خانوادگی ، اسم کوچک و نمره امتحانی دانشجو می‌باشد . درضمن ملاحظه می‌گردد که سابقه هر دانشجو بعنوان یک ساختار شامل ۳ فیلد :

grade ، name ، st-no

می‌باشد که در آن فیلد یا عضو name خودش یک ساختار دوعضوی می‌باشد ؛ پس در اینجا با ساختارهای تودرتو مواجه هستیم . اگر اسم ساختار اصلی را با strec نمایش دهیم به هرکدام از اعضای آن می‌توان به‌ترتیب با :

strec.grade , strec.name , strec.st-no

رجوع کرد . همچنین به عناصر یا اعضای name می‌توان با :

strec.name.f-name , strec.name.l-name

رجوع کرد . ملاحظه می‌گردد که برای دستیابی به اعضای درونی ساختار ، اپراتور نقطه به‌صورت تکراری یا تودرتو مواجه هستیم . اگر اسم ساختار اصلی را با strec نمایش دهیم به هرکدام از اعضای آن می‌توان به‌ترتیب با :

strec.grade , strec.name , strec.st-no

رجوع کرد .

همچنین به عناصر یا اعضای name می‌توان با :

strec.name.f-name , strec.name.l-name

رجوع کرد . ملاحظه می‌گردد که برای دستیابی به اعضای درونی ساختار ، اپراتور نقطه به‌صورت تکراری یا تودرتو بکار رفته است .

حل : برنامه مورد نظر در شکل زیر نشان داده شده 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 ; l<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**

راههای مختلفی برای انتقال یا گذر دادن اطلاعاتی از نوع ساختار به یک تابع و بالعکس ، وجود دارد . می‌توان اعضای ساختار و یا تمامی ساختار را به یک تابع گذر داد . مکانیسم 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 یک تابع ، به‌عنوان آرگومان به آن انتقال داد ؛ و یا اینکه عضو خاصی از یک ساختار را با دستور return به‌وسیلهٔ تابعی به تابع فراخواننده آن برگرداند . برای انجام این کار ، با هریک از اعضای ساختار مشابه یک متغیر معمولی تک‌مقدار ، برخورد می‌شود .

مثال – اسکلت کلی یک برنامه 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 شیوه انتقال اعضای یک ساختار به یک تابع را نمایش می‌دهد . در اینجا مقادیر:

customer.balance , customer.acct-no , customer.name

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 ، بدست 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 پیش‌نمونه adjust در تابع main( ) را به‌صورت زیر نیز توصیف کرد :

```
float  adjust (char[ ] , int , float) ;
```

به کمک یک اشاره‌گر به نوع ساختار ، می‌توان تمامی یک ساختار را به‌عنوان آرگومان به یک تابع انتقال داد . این شیوه ، مشابه انتقال آرایه به یک تابع می‌باشد . بدیهی است که این روش ، انتقال به‌وسیلهٔ آدرس می‌باشد . 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 را که در مورد انتقال آرایه به تابع گفته شده ، مشاهده می‌کنید .

مثال – برنامه ساده 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 یک ساختار به یک تابع را با گذر دادن آدرس آن (یعنی یک اشاره‌گر به ساختار) به تابع ، نمایش می‌دهد .

customer ، از نوع ساختار است و از لحاظ کلاس حافظه ، نیز استاتیک می‌باشد ، که به اعضای آن مقادیر اولیه اختصاص داده شده است . ابتدا در تابع اصلی ، این مقادیر اولیه 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 ، یک اشاره‌گر به یک 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 برنامه مزبور اجرا شود ، خروجی زیر را مشاهده خواهید کرد :

Nader 3333 c 33.33
Payam 9999 r 99.99

- **بازگشت اشاره‌گر به ساختار (توسط یک تابع)**

یک تابع می‌تواند اشاره‌گری به یک ساختار را به توابع فراخواننده آن برگرداند . مثال زیر نحوه عمل را نمایش می‌دهد :

مثال – برنامه‌ای بنویسید که با فراخواندن تابعی به نام search ، آرایه‌ای از ساختارها که هر ساختار مشابه مثال قسمت قبل شامل سابقه یک نفر مشتری بانک است و همچنین شماره حساب یک نفر مشتری را ، در اختیار تابع قرار دهد . تابع مزبور شماره حساب مورد نظر را در مجموعه لیست مشتریان جستجو کند . اگر چنین رکوردی وجود داشت آدرس آن وگرنه، NULL (صفر) برگرداند. سپس در تابع اصلی ، درصورت وجود داشتن چنین رکوردی ، بقیه مشخصات صاحب شماره حساب مذکور نمایش داده شود در غیر اینصورت پیغام مناسبی چاپ شود . عمل فراخوانی تابع برای پیدا کردن رکورد خاص تا موقعی که شماره حساب ارسالی به تابع ، مخالف صفر باشد ، ادامه یابد .

حل : برنامه مورد نظر که در آن سوابق دارندگان حساب بصورت مقدار اولیه به آرایه‌ای از ساختارها نسبت داده شده است و همچنین تابع مزبور در زیر نشان 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 ، که دارای مقدار ۳ می‌باشد . یعنی برای سادگی کار ، فقط ۳ رکورد ، بعنوان نمونه ذخیره شده است . بدیهی است که در عمل ، مقدار n خیلی بزرگتر خواهد بود .

می‌توان تمامی یک ساختار را بطور مستقیم بعنوان یک آرگومان به یک تابع انتقال داد و یا با دستور return ، یک ساختار بطور مستقیم از یک تابع برگردانده شود (برخلاف آرایه که نمی‌تواند به‌وسیلهٔ تابع برگردانده شود) . چنین انتقال ساختاری ، به‌صورت مقدار خواهد بود . بنابراین اگر وسیله تابع ، تغییراتی روی اعضای ساختار داده شود ، نتیجهٔ آن در تابع فراخواننده و یا در خارج از تابع مذکور ، اعمال نخواهد شد . به هرحال اگر ساختار تغییریافته ، به نقطه فراخوانی از برنامه برگردانده شود ، تغییرات حاصل ، در اینجا شناخته خواهد شد . مثال بعدی این موارد را روشنتر می‌سازد .

مثال – برنامه زیر را ملاحظه نمایید :

```
# 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 ;
}
```

برنامه مزبور به جای اشاره‌گر به نوع ساختار ، تمامی ساختار (درواقع کپی ساختار را به تابع می‌فرستند) .

حال تابع adjust یک ساختار را بعنوان آرگومان دریافت می‌کند (برخلاف مثال قبل که یک اشاره‌گر به یک ساختار را دریافت می‌کرد) . در اینجا چیزی از تابع به تابع main برگردانده نشده است .

اگر برنامه مزبور اجرا شود ، خروجی زیر حاصل خواهد شد :

```
Nader 3333 c 33.33
Nader 3333 c 33.33
```

در اینجا ، نتیجه اجرای دستورهای جایگذاری که در تابع adjust بکار 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 . برنامه اصلاح شده با این منظور ، در زیر نشان داده شده است :

```
#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) ;
 }
```

یک ساختار را که همان ساختار تغییریافته در تابع adjustدر این برنامه ، برخلاف مثال قبل ، تابع برمی‌گرداند . mainمزبور است ، به تابع

اجرای این برنامه ، خروجی زیر را ایجاد می‌کند :

```
Nader 3333 c 33.33
Payam 9999 r 99.99
```

بنابراین ملاحظه می‌شود که تغییرات اعمال شده در تابع adjust ، در تابع main نیز منعکس شده است . زیرا این بار ، ساختار تغییریافته بطور مستقیم به قسمت فراخوانی برنامه برگردانده شده است . (این خروجی را با خروجی دومثال قبلی مقایسه کنید) .

- **ساختار داده‌ها و اشاره‌گرها**

مشابه دستیابی به سایر آدرسها ، می‌توان با بکار بردن اپراتور آدرس ، یعنی `&´ به آدرس آغاز یک ساختار دسترسی داشت . بعنوان مثال اگر variable معرف یک متغیر از نوع ساختار باشد ، &variable آدرس آغاز آن متغیر را نشان خواهد داد . همچنین می‌توان یک متغیر اشاره‌گر به یک ساختار بصورت زیر تعریف کرد :

```
type *ptvar ;
```

که در آن type ، یک نوع داده است که دلالت بر ترکیب ساختار می‌کند و ptvar نیز معرف نام متغیر

اشاره‌گر است . حال می‌توان آدرس آغاز یک متغیر ساختار را بصورت زیر به این اشاره‌گر نسبت داد :

```
ptvar = &variable ;
```

مثال – توصیف زیر را در مورد یک 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 یک متغیر ساختار از نوع acount و pc نیز یک متغیر اشاره‌گر است که می‌تواند آدرس یک متغیر ساختار از نوع acount را در خود داشته باشد . حال با دستور زیر ، آدرس آغاز customer به pc نسبت داده می‌شود :

```
pc = &customer ;
```

توصیف متغیر اشاره‌گر را می‌توان به‌صورت زیر با توصیف ساختار ، ترکیب کرد :

```
struct {
    member 1 ;
    member 2 ;
    . . .
    member m ;
} variable , *ptvar ;
```

که در آن ، variable ، یک متغیر از نوع ساختار و ptvar نیز نام متغیر اشاره‌گر را معرفی می‌کند .

مثال – توصیف زیر ، معادل دو توصیف معرفی شده در مثال قبل 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 یک عضو ساختار می‌توان به‌صورت زیر دسترسی داشت :

```
ptvar -> member
```

که در آن ptvar متغیر اشاره‌گر به نوع ساختار مورد نظر است و عملگر -> که در فصل اشاره‌گر نیز

مورد بحث قرار گرفت ، قابل مقایسه با عملگر نقطه ، یعنی `.´ است ؛ بنابراین عبارت :

ptvar -> member

هم‌ارز عبارت :

variable . member

می‌باشد که در آن variable ، یک متغیر از نوع ساختار است که در گذشته مورد بحث قرار گرفت .

عملگر <- نیز مانند عملگر `.´ دارای بالاترین تقدم است .عملگر <- می‌تواند با عملگر نقطه ترکیب شود و بدین طریق عمل دستیابی به یک زیرعضو یا submember را در درون یک ساختار (یعنی دستیابی به عضوی از ساختار که خودش عضو ساختار دیگر است) فراهم سازد . بنابراین می‌توان به یک زیرعضو به‌صورت زیر دسترسی داشت :

ptvar ->member . submember

به چند طریق مشابه عملگر <- می‌تواند برای دستیابی به یک عنصر از آرایه که خودش عضوی از یک ساختار است ، بکار برود . این عمل با دستوری مشابه زیر فراهم می‌گردد :

ptvar -> member [expression]

که در آن expression ، یک مقدار صحیح غیرمنفی است که دلالت بر عنصر آرایه می‌کند (یعنی یک عنصر آرایه را مشخص می‌سازد .)

مثال – تعریف ساختار و متغیرهایی را بصورت زیر درنظر بگیرید (که مشابه آن را در صفحات پیش نیز ملاحظه کردید) :

```
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) بصورت 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 می‌تواند به یکی از سه روش زیر انجام گیرد :

customer . acct_no
pc-> acct_no
(*pc) . acct_no

وجود پرانتز در مورد عبارت آخری ضروری است ، زیرا عملگر نقطه نسبت به عملگر *` دارای تقدم بالاتری است و چنانچه پرانتز بکار برده نشود ، کامپایلر نتیجهٔ اشتباه ایجاد می‌کند ؛ زیرا pc که یک اشاره‌گر است ، بصورت مستقیم با عملگر نقطه سازگار نیست .

به طریق مشابه می‌توان به موجودی مشتری به یکی از سه روش زیر:

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 دسترسی داشت :

(*pc) . lastpayment . month
pc->name
(*pc) . name

همینطور می‌توان به سومین کاراکتر نام مشتری به یکی از روشهای زیر دسترسی داشت :

customer . name[2]
*(customer . name + 2)
pc->name[2]
pc->(name+2)
(*pc) . name[2]
*((*pc) . name + 2)

یادآوری – اعضای ساختار می‌تواند اشاره‌گر نیز باشد ؛ این روش در ساختارهایی مانند لیستهای پیوندی ، درختها ، گرافها و مشابه آن کاربرد زیادی دارد که بعضی از آنها در اینجا مورد بررسی قرارمی‌گیرد .

- **عضو ساختار**

یک متغیر اشاره‌گر می‌تواند عضو یک ساختار باشد . برای مثال مشخصات یک نفر با فرمت :

نام	نام خانوادگی

را می‌توان بصورت ساختار زیر تعریف کرد :

```
struct  names {
            char  *lastname ;
            char  *firstname ;
};
```

که در اینجا ، lastname و firstname اشاره‌گرهایی هستند که درواقع معرف دو رشته (یا دو آرایه کاراکتری) می‌باشند.

همچنین عضو یک ساختار می‌تواند اشاره‌گری باشد که آدرس عناصر دیگر و یا آدرس ساختار دیگری را در خود داشته باشد و 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 بصورت زیر تعریف کرد :

```
struct  tag {
            member 1 ;
            member 2 ;
            ....
            ....
            ....
            struct  tag &name ;
            } ;
```

که در آن name ، متغیری از نوع اشاره‌گر است که می‌توان آدرس متغیر دیگری از نوع ساختار به فرم tag را در آن قرار دارد . یکی از مهمترین کاربردهای روش بالا ، در ایجاد ساختارهایی به نام لیست پیوندی و انجام عملیات یا پردازش روی آن می‌باشد .

- **اجتماع (union)**

union ، محلی از حافظه است که توسط چندین متغیر که ممکن است نوع یا type آنها نیز یکسان نباشد ، بکار برده می‌شود . درواقع union ، متغیری است که امکان ذخیره کردن انواع مختلف داده ، در مکان مشترکی از حافظه را فراهم می‌کند .

تعریف union مشابه تعریف ساختار (structure) است با این تفاوت که در مورد union تمام اعضای تشکیل‌دهندهٔ آن یک فضا را به‌صورت اشتراکی اشغال می‌کنند . (برخلاف ساختار که هر عضو آن دارای محل حافظه یا مکان خاص خودش است .) این گونه ساختمان داده‌ها در کاربردهایی مفید می‌باشند که دارای اعضای چندگانه از نوع مختلف هستند ، ولی در هر زمان باید فقط به یکی از این اعضا مقداری نسبت داده شود . به هرحال کاربر باید بداند که هر لحظه ، چه نوع داده ، در حافظه

مشترک مورد نظر ذخیره شده است .

در حالت کلی ترکیب یک union می‌تواند بصورت زیر تعریف گردد :

```
union  tag {
        member 1 ;
        member 2 ;
        …
        …
        …
        member m ;
} ;
```

که در آن union ، یک کلمه کلیدی است .

مثال – به تعریف زیر توجه کنید :

```
union  union_type {
                    int  i ;
                    char  ch ;
                    } ;
```

این تعریف نیز مشابه تعریف یک ساختار ، موجب توصیف یا اعلان متغیر نمی‌گردد . بلکه فقط تعریف یک نوع داده است . برای اعلان هر متغیری از نوع یک union مورد نظر ، باید نام آن را به دنبال تعریف union بکار برد و یا بطور جداگانه آن را اعلان نمود . پس روش اعلان متغیرهایی از نوع union مشابه اعلان از نوع ساختار است .

با توجه به مطالب بالا ، متغیر x به دو روش اعلان شده است که نتیجه نهایی هر دو ، هم‌ارز می‌باشد :

روش اول

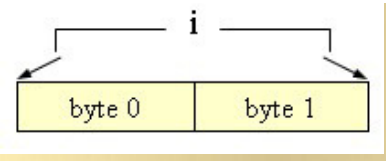
```
union  tag {
   int  i ;
   char ch ;
} ;
union tag x ;
```

روش دوم

```
union tag {
   int  i ;
   char  ch ;
} x ;
```

در اینجا ، متغیرهای i و ch که به ترتیب از نوع int و char می‌باشند ، دارای حافظه مشترک هستند که البته متغیر i ، دو بایت و متغیر ch ، یک بایت حافظه را اشغال می‌کند ، ولی آدرس شروع آنها ، یکسان می‌باشد ؛ مشابه شکل زیر :

نحوه اشغال یک حافظهٔ مشترک به‌وسیلهٔ دو عضو یک union



وقتی که یک union اعلام می‌گردد ، کامپایلر به‌طور اتوماتیک یک متغیر که بتواند بزرگترین عضو union را در خود جای دهد ، ایجاد می‌کند که در مثال بالا ، بزرگی آن دو بایت می‌باشد .

باتوجه به مطالب بالا می‌توان فرم کلی اعلان متغیرهایی از نوع union را به‌صورت زیر نشان داد :

storage _ class union tag variable1 ,variable 2 , … , variable *n* ;

که در آن storage_class یک گزینه اختیاری می‌باشد که اگر بکار نبریم حافظه مورد نظر از کلاس اتوماتیک خواهد بود .

بطوری که بیان شد می‌توان اعلان متغیرهایی از نوع union را به‌دنبال تعریف union بکار برد. پس می‌توان فرم بالا را بصورت زیر تعریف کرد :

```
storage_class union tag {
            member 1 ;
            member 2 ;
            ...
            ...
            ...
            member m ;
} variable 1 , variable 2 , … , variable n ;
```

مثال – یک برنامه C ممکن است شامل اعلان زیر باشد :

```
union id {
    char color [12] ;
    int size ;
} shirt , blouse ;
```

در این مثال ، دو متغیر shirt و blouse از نوع union که با نام id تعریف شده است ، می‌باشند . هرکدام از این دو متغیر در هر لحظه یا معرف یک رشته ۱۲ کاراکتری color و یا معرف مقدار صحیح size خواهد بود . بدیهی است چون رشته مزبور ۱۲ کاراکتری می‌باشد و نیاز به حافظه بیشتری در مقایسه با size دارد ، درنتیجه کامپایلر ، بطور اتوماتیک ۱۲ بایت حافظه برابر هر متغیر union اختصاص خواهد داد.

یک union می‌تواند عضو یک ساختار باشد ؛ همچنین یک ساختار می‌تواند عضو یک union باشد .

مثال – تعریف زیر را درنظر بگیرید :

```
union id {
      char color[12] ;
      int s□□e□;; ٥      struct c□othes {
```

```
۵      struct  c□othes {
                    char  manufacturer[20] ;
                    float  cost ;
                    union  id  description ;
       } shirt , blouse ;
}
```

در این مثال ، دو متغیر shirt و blouse متغیرهایی از نوع ساختار هستند که با نام clothes تعریف شده است و هرکدام از آنها شامل اعضای زیر است :

یک رشته به نام manufacturer (نام تولیدکننده لباس) ، یک مقدار از نوع اعشار به نام cost (که معرف بهای لباس است) و یک union به نام description (که نوع لباس را شرح می‌دهد) . در اینجا ممکن است که union ، معرف یک رشته یا color و یا معرف یک 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 ;
```

نحوه دستیابی به عناصر یا اعضای یک union ، مشابه همان است که در مورد ساختارها بیان شد . یعنی این کار با استفاده از دو عملگر :

`.` و `->`

انجام می‌گیرد . بنابراین اگر variable یک متغیر union باشد دستور :

variable . member

یک عضو آن را معرفی و مشخص می‌کند . به طریق مشابه ، اگر ptvar ، یک متغیر اشاره‌گر باشد که به یک union اشاره می‌کند (یعنی آدرس یک متغیر از نوع union را در خود دارد) در اینصورت :

ptvar->member

به یک عضو آن union اشاره خواهد کرد .

مثال – برنامه ساده 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 union تک‌کاراکتری می‌باشد برخلاف مثال قبلی که یک آرایه ۱۲ کاراکتری بود . انجام این تغییرات بدین لحاظ صورت گرفته که نسبت دادن مقادیر به اعضای union آسانتر باشد .

ملاحظه می‌کنید که به عضو :

shirt . description . color

مقدار 'w`نسبت داده شده است . پس باید توجه داشته باشید که عضو دیگر union ، یعنی :

shirt . description

مقدار با معنی نخواهد داشت . سپس با دستور printf ، مقدار هر دو عضو union نمایش داده شده است . سپس به عضو :

shirt . description . size

مقدار ۱۲ نسبت داده شده است . بدیهی است که این مقدار ، روی مقدار حافظه مشترک که برای دو عضو مزبور پیش‌بینی شده است ، خواهد نشست . سپس بار دیگر با دستور printf مقدار هر عضو ، نمایش داده شده است .

اجرای برنامه مزبور ممکن است نتایج زیر را بعنوان خروجی ایجاد کند :

```
2
w-24713
@ 12
```

سطر اول نشان می‌دهد که ۲ بایت حافظه به union ، اختصاص داده شده است ، تا بتواند بزرگترین عضو خود را که یک مقدار صحیح است ، در خود جای دهد . در سطر دوم ، داده اولی 'w` است که دارای معنی و مفهوم می‌باشد . اما داده دوم یعنی ۲۴۷۱۳- دارای معنی و مفهوم نیست . در سطر سوم ، داده اولی @ می‌باشد که بدون معنی است . اما داده دوم (که ۱۲ می‌باشد) دارای معنی و مفهوم است .

- **نوع شمارشی**

یکی دیگر از داده‌های از نوع اسکالر ، نوع شمارشی می‌باشد . بعضی زبانهای دیگر مانند زبان پاسکال نیز این نوع داده‌ها را بعنوان یکی از انواع داده‌های استاندارد پشتیبانی می‌کنند . یک داده از

نوع شمارشی ، مشابه یک ساختار و یا یک union می‌باشد و اعضای آن ، ثابتهایی هستند که بعنوان شناسه نوشته می‌شوند . اگرچه مقدار یا ارزش آنها از نوع مقدار صحیح علامت‌دار می‌باشد . این ثابتها ، مقادیری را معرفی می‌کنند که می‌توان به متغیرهای شمارشی متناظر با آنها نسبت داد .

در حالت کلی یک نوع شمارشی بصورت زیر تعریف می‌شود :

enum tag {member 1 , member 2 , **...** , member *m*} ;

که در آن enum یک کلمه کلیدی است و tag نیز اسمی است که داده شمارشی را مشخص می‌کند که دارای این ترکیب است و عناصر :

member 1 , member 2 , **...** , member *m*

نیز شناسه‌هایی را مشخص می‌کند که ممکن است (می‌تواند) به متغیرهایی از نوع tag نسبت داده شود . اسامی اعضا باید متفاوت (متمایز از یکدیگر) باشند .

وقتی که نوع شمارشی تعریف شد ، می‌توان متغیرهای شمارشی متناظر با آن را بصورت زیر اعلان کرد :

storage_class enum tag variable1 , variable 2 , … , variable *n* ;

که در آن storage_class گزینه اختیاری می‌باشد که کلاس حافظه را مشخص می‌کند . enum نیز کلمه کلیدی است که باید بکار برده شود ؛ tag اسمی است که در تعریف نوع شمارشی ظاهر می‌گردد و بالاخره :

variable1 , variable 2 , … , variable *n*

متغیرهای شمارشی از نوع tag می‌باشند .

تعریف شمارشی را می‌توان با اعلان متغیرها ، ترکیب کرد و بصورت زیر بکار 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 می‌باشد و می‌توان آن را بکار نبرد .

فرم بالا را می‌توان به اختصار بصورت زیر نوشت :

enum enum_type_name {enumeration list} variable_list ;

که در فرم مزبور بکار بردن نام نوع شمارشی یعنی enum_type_name اختیاری است . نام نوع شمارشی ، برای اعلان متغیرهایی از آن نوع می‌باشد . قطعه برنامه زیر ، یک نوع شمارشی به نام coin تعریف می‌کند و نوع متغیر money را از این نوع اعلان می‌کند :

```
enum coin {penny , nickel , dime , quarter , half_dollar , dollar} ;
enum coin money ;
```

با داشتن تعریف و اعلان بالا ، دستورهای زیر کاملاً درست و معتبر می‌باشد :

```
money = dime ;
if (money = = quarter)  prinf ("is a quarter\n") ;
```

نکتهٔ مهمی که در اینجا باید در مورد نوع شمارشی فهمید آن است که هر سمبول معرف یک

مقدار صحیح است و می‌تواند در هر عبارت از نوع مقادیر صحیح بکار برده شود . برای مثال عبارت :

```
printf ("the number of nickels in a quarter is %d" , quarter + 2) ;
```

کاملاً معتبر می‌باشد .

مقدار اولین سمبول شمارشی برابر ۰ ، مقدار دومین سمبول شمارشی برابر ۱ ، و بالاخره مقدار n اُمین سمبول شمارشی برابر:

n - 1

می‌باشد . مگر اینکه به طریق دیگری مقداردهی اولیه شده باشند ، بنابراین عبارت :

```
printf ("%d %d" , penny , dime) ;
```

مقادیر 0 2 را روی صفحه تصویر نمایش خواهد داد .

می‌توان به هریک از سمبولها ، مقدار اولیه نسبت داد . برای این کار باید پس از سمبول موردنظر علامت '=` و سپس مقدار صحیح مطلوب را بکار ببریم . وقتی که به یکی از سمبول 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 مقادر سمبولها بصورت زیر خواهند بود :

penny	0
nickel	1
dime	2
quarter	100
half_dollar	101
dollar	102

بطور متعارف اینطور فرض می‌شود که سمبولهای یک نوع شمارشی می‌توانند بطور مستقیم بعنوان ورودی یا خروجی باشند . ولی اینطور نیست . برای مثال قطعه برنامه زیر آنچه را که انتظار داریم ، انجام نمی‌دهد :

```
money = dollar ;
printf ("%d" , money) ;
```

به خاطر داشته باشید سمبول dollar به‌طور ساده مقدار یک رشته نیست . بلکه یک اسم برای یک مقدار صحیح است . بنابراین تابع printf نمی‌تواند رشته "dollar" را نمایش دهد . به طریق مشابه نمی‌توانید با بکار بردن یک رشته هم‌ارز ، به یک متغیر شمارشی یک مقدار نسبت بدهید . بعنوان مثال دستور زیر درست کار نمی‌کند :

```
money = "penny" ;
```

به هرحال به طریق دیگری می‌توان مقدار رشته‌ای سمبولهای مورد نظر را ایجاد کرد . مثلاً قطعه برنامه 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 بعنوان شاخص یا index آن آرایه بکار برد تا یک مقدار شمارشی را به رشته متناظر آن ترجمه کند . برای مثال قطعه برنامه زیر رشته مورد نظر را بعنوان خروجی ، تولید خواهد کرد :

```
char name[ ] = {
                "penny" ,
                "nickel" ,
                "dime" ,
                "quarter" ,
                "half_dollar" ,
                "dollar"
               } ;
  printf ("%c" , name [(int) money] ) ;
```

البته این روش ، درصورتی درست کار می‌کند که به هیچ یک از سمبولها ، مقدار اولیه نسبت داده نشده باشد . زیرا شاخص آرایهٔ رشته‌ها ، همیشه از ۰ شروع می‌شود .

مثال – فرض کنید که یک برنامه C شامل دستور زیر است :

```
enum  colors {black , blue , cyan , green , magenta , red , white , yellow} ;
colors  foreground , background ;
```

در سطر اول ، یک نوع شمارشی به نام colors تعریف شده که شامل هشت ثابت است که اسامی آنها عبارتند از :

black , blue , cyan , green , magenta , red , white , yellow

در سطر دوم متغیرهای foreground و background به‌عنوان شمارشی ، از نوع colors اعلان شده‌اند ؛ بنابراین به هریک از این متغیرها می‌توان هریک از هشت ثابت زیر را نسبت داد :

black , blue , cyan , ... , yellow

می‌توان دو دستور بالا را با یکدیگر ترکیب و بصورت زیر نوشت:.

```
enum colors {black , blue , cyan , green , magenta , red , white , yellow}foreground , background ;
```

در این مثال مقادیر صحیح هشت ثابت بالا ، بصورت زیر خواهد بود :

```
black     0
blue      1
cyan      2
green     3
magenta   4
```

```
red       5
white     6
yellow    7
```

به هرحال بطوری که بیان شد ، اگر به برخی از ثابتها ، مقدار اولیه نسبت داده شود ، مقادیر مزبور تغییر خواهد یافت . بعنوان مثال اگر نوع شمارشی مزبور را بصورت :

```
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
```

به هرحال متغیرهای شمارشی نمی‌توانند بطور کامل ، مشابه متغیرهای از نوع مقدار صحیح ، مورد پردازش قرار گیرند . مثلاً نمی‌توان به آنها مقدار جدید نسبت داد و یا آنها را مقایسه کرد . همینطور نمی‌توان آنها را به‌عنوان ورودی به درون کامپیوتر خواند و به متغیر شمارشی دیگر نسبت داد (می‌توان یک مقدار صحیح را از ورودی دریافت کرد و آن را به یک متغیر شمارشی نسبت داد . گرچه این شیوه ، متداول نمی‌باشد .) همچنین فقط می‌توان مقدار صحیح متغیر شمارشی را به‌عنوان 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 بعنوان نمونه‌هایی از عملیات روی متغیرهای شمارشی بکار 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 \n") ;
  }
```

- **تمرین و پاسخ**

تمرین ۱ – ساختاری برای مشخصات دانشجویی زیر بنویسید .

شمارهٔ دانشجو	نام	تاریخ تولد		
		ماه	روز	سال
123456	Amiri	11	5	1980

حل :

```
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 می‌شود برای دستیابی به فیلدهای متداخل از دو علامت نقطه استفاده می‌شود.

نکته : برای مقداردهی اولیه به ساختار متداخل از فرم 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 .

نکته : متغیر d که از نوع struct date است ۶ بایت فضا می‌گیرد . پس کل فضایی که متغیر a در ساختار فوق اشغال می کند برابر است با :

۴ + ۲۰ + ۶ = ۳۰ بایت

تمرین ۲ – با استفاده از استراکچر برنامه‌ای بنویسید که نام n نفر همراه با شماره تلفن شان را ذخیره کند . سپس نام فردی را گرفته شماره تلفن آن را نمایش دهید .

حل :

```
# 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 */
}
```

تمرین 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) ;
}
```

خروجی برنامه

```
65  A  65  A
```

نکته – دستور p-> c معادل (*p) . c می‌باشد .

فصل دهم- فایلها

• مقدمه

متغیرهای معمولی ، آرایه‌ها و ساختمانها ، همگی در حافظه RAM قرار دارند . لذا پس از خاموش شدن کامپیوتر و یا خروج از برنامه ، داده‌هایی که در آنها ذخیره شده‌اند از بین می‌روند و برای استفاده مجدد از آنها ، باید مجدداً وارد گردند که قطعاً این کار مقرون به صرفه نیست . زیرا نه‌تنها مستلزم صرف وقت زیادی است ، بلکه حوصله انجام کار را نیز از برنامه‌نویس سلب می‌نماید . برای رفع این مشکل نوعی ساختمان دادهٔ دیگر به نام فایل مورد استفاده قرار می‌گیرد . این نوع ساختمان داده بر روی حافظه جانبی مثل دیسک ، نوار و غیره تشکیل می‌گردند . چون اطلاعات موجود در روی حافظه جانبی با قطع جریان برق ، قطع اجرای برنامه و یا دلایلی از این قبیل از بین نمی‌روند ، به دفعات زیادی مورد استفاده قرار می‌گیرند .

هر فایل شامل مجموعه‌ای از داده‌های مرتبط به هم است . مانند داده‌های مربوط به کلیه دانشجویان یک دانشگاه ، داده‌های مربوط به هریک از اجزای فایل ، یک رکورد نام دارد . به‌عنوان مثال ، در یک دانشگاه داده‌های مربوط به هر دانشجو تشکیل یک رکورد را می‌دهند . لذا می‌توان گفت که هر فایل ، مجموعه‌ای از چند رکورد است . اگر باز هم دقیق‌تر به فایل دانشجویان دانشگاه پرداخته شود ، مشاهده می‌گردد که هر دانشجو ممکن است چند قلم داده داشته باشد . مثل نام دانشجو ، تعداد واحدهایی که گذرانده ، نمره هر درس و غیره به هریک از اجزای یک رکورد ، فیلد گفته می‌شود . لذا می‌توان گفت که هر کورد مجموعه‌ای از چند فیلد است .

در زبان C فایل داده ، می‌تواند هر دستگاهی مثل : صفحه نمایش ، صفحه کلید ، چاپگر ، ترمینال، دیسک ، نوار و غیره باشد .

داده‌ها ممکن است به چهار روش در فایل ذخیره شده و سپس مورد بازیابی قرار گیرند :

- داده‌ها ، کاراکتر به کاراکتر در فایل نوشته شده و سپس کاراکتر به کاراکتر از فایل خوانده شوند .
- داده‌ها به‌صورت رشته‌ای از کاراکترها ، در فایل نوشته شده و سپس به‌صورت رشته‌ای از کاراکترها مورد دسترسی قرار گیرند .
- داده‌ها در حین نوشتن بر روی فایل ، با فرمت خاصی نوشته شده و سپس با همان فرمت خوانده شوند .
- داده‌ها به شکل ساختمان (رکورد) بر روی فایل نوشته شده و سپس به‌صورت ساختمان از فایل خوانده شوند .

برای هریک از موارد فوق توابع خاصی در زبان C منظور شده‌اند که در این فصل مورد بررسی قرار خواهند گرفت .

- **انواع فایل**

داده‌ها ممکن است در فایل به دو صورت وجود داشته باشند که عبارتند از :

- text
- binary

این دو روش ذخیره شدن داده‌ها ، در موارد زیر با یکدیگر تفاوت دارند :

- تعیین انتهای خط .
- تعیین انتهای فایل .
- نحوهٔ ذخیره شدن اعداد بر روی دیسک .

در فایل text اعداد بصورت رشته‌ای از کاراکترها ذخیره می‌شوند ، ولی در فایل باینری اعداد با همان صورتی که در حافظه قرار می‌گیرند بر روی دیسک ذخیره می‌گردند . بعنوان مثال ، در فایل text عدد ۵۲۶ سه بایت را اشغال می‌کند . زیرا هر رقم آن ، بصورت یک کاراکتر در نظر گرفته می‌شود ، ولی در فایل باینری این عدد در ۲ بایت ذخیره می‌گردد (چون عدد ۵۲۶ یک عدد صحیح است و اعداد صحیح در دو بایت ذخیره می‌شوند) .

در فایل text ، کاراکتری که پایان خط را مشخص می‌کند ، در حین ذخیره شدن بر روی دیسک باید به کاراکترهای :

CR/LF ، line feed ، carriage

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 تبدیلات مستلزم صرف وقت است ، لذا دسترسی به اطلاعات موجود در فایلهای text کندتر از فایلهای باینتری است .

اختلاف دیگر فایلهای text و باینری در تشخیص انتهای فایل است . در هر دو روش ذخیره فایلها ، طول فایل توسط سیستم نگهداری می‌شود و انتهای فایل باتوجه به این طول مشخص می‌گردد . در حالت text کاراکتر 1A (در مبنای ۱۶) و یا ۲۶ (در مبنای ۱۰) مشخص‌کنندهٔ انتهای فایل است . (این کاراکتر با فشار دادن کلید Ctrl به همراه کلید Z تولید می‌شود) . در حین خواندن داده‌ها از روی فایل text ، وقتی کنترل به این کاراکتر رسید بیانگر این است که داده‌های موجود در فایل تمام شده‌اند . در فایل باینری ممکن است عدد 1A (در مبنای ۱۶) و یا ۲۶ (در مبنای ۱۰) جزئی از اطلاعات بوده ، بیانگر انتهای فایل نباشد . لذا نحوهٔ تشخیص انتهای فایل در فایل باینری با فایل text متفاوت است .

از نظر نحوهٔ ذخیره و بازیابی داده‌ها در فایل دو روش وجود دارد :

- سازمان فایل ترتیبی (sequential) .
- سازمان فایل تصادفی (random) .

در سازمان فایل ترتیبی ، رکوردها به همان ترتیبی که از ورودی خوانده می‌شوند در فایل قرار

می‌گیرند و در هنگام بازیابی ، به همان ترتیبی که در فایل ذخیره شده‌اند مورد دسترسی قرار می‌گیرند .

در سازمان فایل تصادفی ، به هر رکورد یک شماره اختصاص می‌یابد . لذا اگر فایل دارای n رکورد باشد ، رکوردها از ۱ تا n شماره‌گذاری خواهند شد . وقتی که رکورد در یک فایل با سازمان تصادفی قرار گرفت ، محل آن توسط یک الگوریتم پیداکنندهٔ آدرس که با فیلد کلید ارتباط دارد مشخص می‌شود . در این صورت دو رکورد با فیلد کلید مساوی ، نمی‌توانند در فایل تصادفی وجود داشته باشند . در سازمان فایل تصادفی مستقیماً می‌توان به هر رکورد دلخواه دسترسی پیدا کرد بدون اینکه رکوردهای قبل از آن خوانده شوند .

- **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 عبارتند از :

- نام فایل .
- نوع فایل ازنظر ذخیرهٔ اطلاعات (text یا binary) .
- نوع فایل ازنظر ورودی ـ خروجی (آیا فایل فقط به‌عنوان ورودی است ، آیا فقط به‌عنوان خروجی است یا هم به‌عنوان ورودی است و هم به‌عنوان خروجی) .

یک فایل ممکن است طوری باز شود که فقط عمل نوشتن اطلاعات بر روی آن مجاز باشد . به چنین فایلی ، فایل خروجی گفته می‌شود . اگر فایل طوری باز گردد که فقط عمل خواند اطلاعات از آن امکان‌پذیر باشد به چنین فایلی ، فایل ورودی گفت می‌شود . اگر فایل طوری باز شود که هم عمل نوشتن اطلاعات بر روی آن مجاز باشد و هم عمل خواندن اطلاعات از آن ، به چنین فایلی ، فایل ورودی و خروجی گفته می‌شود . اگر فایلی قبلاً وجود نداشته باشد ، در حین بازشدن باید به‌عنوان 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 گرفته و دارای الگوی زیر است :

```
FILE   *fopen (char *filename , *mode)
```

در الگوی فوق ، 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 می‌تواند شامل نام درایو و یا هر مسیر موجود روی دیسک باشد . mode مشخص می‌کند که فایل چگونه باید باز شود (ورودی ، خروجی و یا ...) . مقادیری که می‌توانند بجای mode در تابع fopen قرار گیرند ، همراه با مفاهیم آنها در جدول زیر آمده‌اند .

مقادیر معتبر mode در تابع fopen( )

mode	مفهوم
r (rt)	فایلی از نوع text را به‌عنوان ورودی باز می‌کند .
w (wt)	فایلی از نوع text را به‌عنوان خروجی باز می‌کند .
a (at)	فایل را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه نمود .
rb	فایلی از نوع باینری را به‌عنوان ورودی باز می‌کند .
wb	فایلی از نوع باینری را به‌عنوان خروجی باز می‌کند .
ab	فایل موجود از نوع باینری را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه نمود .
r + (r+t)	فایل موجود از نوع text را به‌عنوان ورودی و خروجی باز می‌کند .
w + (w+t)	فایلی از نوع text را به‌عنوان ورودی و خروجی باز می‌کند .
a + (a+t)	فایل موجود از نوع text را به‌عنوان ورودی و خروجی باز می‌کند .
r + b	فایل موجود از نوع باینری را به‌عنوان 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 باز کردن فایل باید یک اشاره‌گر از نوع فایل تعریف گردد تا به فایلی که توسط تابع fopen باز می‌شود اشاره نماید . اگر فایل به دلایلی باز نشود این اشاره‌گر برابر با NULL خواهد بود . به‌عنوان مثال ، دستورات زیرا را درنظر بگیرید :

```
FILE  *fp ;                    (1)
fp = fopen ("A : test" , "w") ;   (۲)
```

دستور (۱) ، متغیر fp را از نوع اشاره‌گر فایل تعریف می‌کند و دستور (۲) فایلی به نام text را بر روی درایو A ایجاد می‌نماید (چون حالت "w" ، فایل را به‌صورت خروجی باز می‌کند) .

برای تشخیص این که 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 بکار 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 توسط تابع fclose انجام می‌شود که دارای الگوی زیر است :

```
int  fclose (FILE  *fp)
```

در الگوی فوق ، fp به فایلی اشاره می‌کند که باید توسط تابع fclose بسته شود . به‌عنوان مثال ، دستور :

```
fclose (p) ;
```

موجب بستن فایلی می‌شود که p به آن اشاره می‌کند

- **putc , getc**

برای نوشتن یک کاراکتر در فایل ، از توابع putc و fputc استفاده می‌شود . طریقه استفاده از این دو تابع یکسان است . تابع putc در نسخه‌های جدید C و نیز fputc در نسخه‌های قدیمی C وجود داشته است . چون تابع putc بصورت ماکرو تعریف شده است سرعت عمل آن بالا است . الگوی تابع putc بصورت زیر است :

```
int  putc (int  ch , FILE *fp)
```

که در الگوی فوق ، 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 وfgetc استفاده نمود . نحوه بکارگیری این دو تابع یکسان است . تابع fgetc در گونه‌های قدیمی C و نیز getc در گونه‌های جدید C وجود دارد . چون تابع getc بصورت ماکرو پیاده‌سازی شده است از سرعت بیشتری برخوردار است . الگوی این تابع بصورت زیر است :

```
int  getc (FILE  *fp)
```

که در الگوی فوق ، fp اشاره‌گری است که مشخص‌می‌کند کاراکتر مورد نظر از کدام فایل باید خوانده شود . در مورد خواندن و نوشتن داده‌ها بر روی فایل باید به نکاتی توجه داشت :

وقتی کاراکترهایی بر روی فایل نوشته می‌شوند باید مکان بعدی 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 یک متغیر به نام موقعیت سنج فایل استفاده می‌کند که با هر دستور خواندن و یا نوشتن بر روی فایل ، مقدار این متغیر به‌طور اتوماتیک تغییر می‌کند تا موقعیت فعلی فایل را مشخص نماید . لذا عمل نوشتن بر 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 نمود . 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 اینصورت ، سیستم پیام خطایی را مبنی بر نبودن اطلاعات در فایل صادر می‌کند .

در حین خواندن داده‌ها از فایل text پس از رسیدن به انتهای فایل تابع getc یا fgetc علامت EOF را برمی‌گرداند . لذا در هنگام خواندن داده‌ها از فایل text می‌توان به عمل خواندن ادامه داد ، تا اینکه

کاراکتر خوانده شده برابر با EOF گردد . در فایل باینتری برای تست کردن انتهای فایل از تابع feof استفاده می‌گردد . الگوی این تابع به‌صورت زیر است :

```
int  feof (FILE  *fp)
```

که در الگوی فوق ، fp اشاره‌گری است که مشخص می‌کند این تابع باید بر روی چه فایلی عمل کند . تابع fopen علاوه بر تشخیص انتهای فایلهای باینری برای تشخیص انتهای فایلهای text نیز استفاده می‌شود . اکنون به بررسی مثالهایی در مورد فایلهای text و باینری و چگونگی تست انتهای این دو نوع فایل می‌پردازیم .

مثال – برنامه‌ای بنویسید که کاراکترهایی را از ورودی خوانده و در یک فایل متنی قرار دهد و سپس داده‌های موجود در این فایل را خوانده و به فایل دیگری منتقل کند . آخرین کاراکتر ورودی را نقطه در نظر بگیرید .

حل :

```
# 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) ;
  }
```

مثال – برنامه‌ای بنویسید که کاراکترهایی را از صفحه کلید گرفته و در یک فایل باینری قرار دهد و سپس کاراکترهای موجود در این فایل را خوانده و به یک فایل باینری دیگر منتقل کند . اسامی فایلهای ورودی و خروجی به‌عنوان آرگومان تابع اصلی به برنامه وارد می‌شوند .

حل :

```
# 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) ;
}
```

- **putc , getc**

برای نوشتن یک کاراکتر در فایل ، از توابع putc و fputc استفاده می‌شود . طریقه استفاده از این دو تابع یکسان است . تابع putc در نسخه‌های جدید C و نیز fputc در نسخه‌های قدیمی C وجود داشته است . چون تابع putc بصورت ماکرو تعریف شده است سرعت عمل آن بالا است . الگوی تابع putc بصورت زیر است :

```
int  putc (int  ch , FILE *fp)
```

که در الگوی فوق ، 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 وfgetc استفاده نمود . نحوه بکارگیری این دو تابع یکسان است . تابع fgetc در گونه‌های قدیمی C و نیز getc در گونه‌های جدید C وجود دارد . چون تابع getc بصورت ماکرو پیاده‌سازی شده است از سرعت بیشتری برخوردار است . الگوی این تابع بصورت زیر است :

```
int  getc (FILE  *fp)
```

که در الگوی فوق ، fp اشاره‌گری است که مشخص‌می‌کند کاراکتر مورد نظر از کدام فایل باید خوانده شود . در مورد خواندن و نوشتن داده‌ها بر روی فایل باید به نکاتی توجه داشت :

وقتی کاراکترهایی بر روی فایل نوشته می‌شوند باید مکان بعدی 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 یک متغیر به نام موقعیت سنج فایل استفاده می‌کند که با هر دستور خواندن و یا نوشتن بر روی فایل ، مقدار این متغیر به‌طور اتوماتیک تغییر می‌کند تا موقعیت فعلی فایل را مشخص نماید . لذا عمل نوشتن بر 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 نمود . 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 اینصورت ، سیستم پیام خطایی را مبنی بر نبودن اطلاعات در فایل صادر می‌کند .

در حین خواندن داده‌ها از فایل text پس از رسیدن به انتهای فایل تابع getc یا fgetc علامت EOF را برمی‌گرداند . لذا در هنگام خواندن داده‌ها از فایل text می‌توان به عمل خواندن ادامه داد ، تا اینکه

کاراکتر خوانده شده برابر با EOF گردد . در فایل باینتری برای تست کردن انتهای فایل از تابع feof استفاده می‌گردد . الگوی این تابع به‌صورت زیر است :

```
int  feof (FILE  *fp)
```

که در الگوی فوق ، fp اشاره‌گری است که مشخص می‌کند این تابع باید بر روی چه فایلی عمل کند . تابع fopen علاوه بر تشخیص انتهای فایلهای باینری برای تشخیص انتهای فایلهای text نیز استفاده می‌شود . اکنون به بررسی مثالهایی در مورد فایلهای text و باینری و چگونگی تست انتهای این دو نوع فایل می‌پردازیم .

مثال – برنامه‌ای بنویسید که کاراکترهایی را از ورودی خوانده و در یک فایل متنی قرار دهد و سپس داده‌های موجود در این فایل را خوانده و به فایل دیگری منتقل کند . آخرین کاراکتر ورودی را نقطه در نظر بگیرید .

حل :

```
# 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) ;
  }
```

مثال – برنامه‌ای بنویسید که کاراکترهایی را از صفحه کلید گرفته و در یک فایل باینری قرار دهد و سپس کاراکترهای موجود در این فایل را خوانده و به یک فایل باینری دیگر منتقل کند . اسامی فایلهای ورودی و خروجی به‌عنوان آرگومان تابع اصلی به برنامه وارد می‌شوند .

حل :

```
# 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) ;
}
```

- **فایل وسیلهٔ ورودی ـ خروجی**

می‌توان یک فایل را هم بعنوان وسیلهٔ ورودی و هم بعنوان وسیلهٔ خروجی مورد استفاده قرار داد . برای این منظور کافی است در تابع fopen بجای mode از یکی از عبارات :

r + t یا r+

برای باز کردن فایل text موجود بعنوان ورودی و خروجی و یا یکی از عبارات :

w + t یا w+

برای ایجاد یک فایل text بعنوان ورودی و خروجی ، و یا یکی از عبارات :

a + t یا a+

برای ایجاد فایل text و یا باز کردن فایل text موجود ، بعنوان ورودی و خروجی ، و یا عبارت :

r + b

برای باز کردن فایل باینری موجود ، بعنوان ورودی و خروجی ، و یا عبارت :

w + b

برای ایجاد یک فایل باینری بعنوان ورودی و خروجی و یا عبارت :

a + b

برای ایجاد و یا بازکردن فایل موجود باینری بعنوان ورودی و خروجی استفاده نمود . بعنوان مثال، دستورات زیر را درنظر بگیرید :

```
fp1 = fopen ("test . dat" , "w+b") ;        (۱)
fp2 = fopen ("sample . dat" , "r+b") ;      (۲)
fp3 = fopen ("test2 . dat" , "a+t") ;       (۳)
```

دستور (۱) ، فایلی به نام test . dat را از نوع باینری و به‌صورت ورودی و خروجی باز می‌کند که اشاره‌گر fp1 به آن اشاره می‌کند . اگر این فایل قبلاً وجود داشته باشد محتویات قبلی آن ازبین خواهند رفت .

دستور (۲) ، فایلی به نام sample . dat را که اکنون بر روی 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 (۳) ، فایلی به نام test 2 . dat را از نوع text و به‌صورت ورودی و خروجی باز می‌کند . اگر فایل test2 . dat قبلاً وجود نداشته باشد ، ایجاد خواهد شد و اگر وجود داشته باشد اطلاعات قبلی آن محفوظ بوده و اطلاعات جدید به انتهای آن اضافه خواهد شد .

باتوجه به مطالبی که تاکنون در مورد فایلها گفته شد ، 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 بسته و مجدداً آن را باز نمود . اصولاً شاید در فایلهایی که فقط بعنوان خروجی و یا فقط به‌عنوان 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 بعنوان یک نیاز مطرح است . برای این منظور از تابعی به نام rewind استفاده می‌گردد . الگوی این تابع در فایل stdio.h قرار داشته و بصورت زیر است :

```
void  rewind (FILE  *fp)
```

در الگوی فوق ، fp به فایلی اشاره می‌کند که موقعیت‌سنج آن باید به ابتدای فایل اشاره نماید .

مثال – برنامه زیر رشته‌هایی را از ورودی خوانده و در یک فایل text قرار می‌دهد و سپس محتویات این فایل را خوانده و به صفحه 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) ;
}
```

- **ferror**

در حین انجام کار با فایلها ممکن است خطایی رخ دهد . بعنوان مثال ، عدم وجود فضای کافی برای ایجاد فایل ، آماده نبودن دستگاهی که فایل باید در آنجا تشکیل گردد و یا مواردی از این قبیل

منجر به بروز خطا می‌شوند . با استفاده از تابع ferror می‌توان از بروز چنین خطایی مطلع گردید . الگوی تابع ferror در فایل stdio.h قرار داشته و به‌صورت زیر است :

```
int  ferror (FILE *fp)
```

در الگوی فوق ، fp اشاره‌گری است که مشخص می‌کند این تابع باید بر روی چه فایلی عمل کند . این تابع یک تابع منطقی است . بدین معنی که اگر خطایی در رابطه با فایلها رخ داده باشد این تابع ارزش درست و در غیر اینصورت 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 نمود .

مثال – برنامه زیر کاراکترهای tab را از فایل حذف کرده و بجای 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) ;
  }
```

- **remove**

برای حذف فایلهای غیرضروری می‌توان از تابع remove استفاده کرد . الگوی این تابع در فایل stdio . h قرار داشته و بصورت زیر است :

```
int  remove (char *filename)
```

که در الگوی فوق filename به نام فایلی که باید حذف شود ، اشاره می‌کند . اگر عمل تابع با موفقیت انجام شود مقدار صفر و در غیر اینصورت مقداری غیر از صفر برگردانده خواهد شد .

مثال - برنامه زیر نام فایلی را بعنوان آرگومان پذیرفته 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 ("erase %s (y/n) : " , argv[1] );
  gets (str) ;
  if  (toupper (*str) = = 'y')
    if  (remove (argv[1]))
      {
        printf ("cannot erase file\n") ;
        exit (1) ;
      }
}
```

- **fprintf , fscanf**

اگر لازم باشد که داده‌ها با فرمت خاصی در فایل نوشته و یا از آن خوانده شوند می‌توان از دو تابع fprintf و fscanf استفاده نمود . این دو تابع دقیقاً کار توابع printf و scanf را در ورودی خروجی معمولی (غیر از فایل) انجام می‌دهند . الگوی این توابع در فایل stdio . h قرار داشته و بصورت زیر می‌باشند :

```
int  fprintf (FILE *fp , "*control_string , …" , char arg , …)

int  fscanf (FILE *fp , "*control_string , …" , char arg , …)
```

در الگوهای فوق ، fp اشاره‌گری است که مشخص می‌کند اعمال این توابع باید بر روی چه فایلی انجام شود . control_string مشخص می‌کند که داده‌ها یا args باید با چه فرمتی نوشته و یا خوانده شوند .

مثال – برنامه زیر یک رشته و یک عدد صحیح را از ورودی خوانده و آن را در یک فایل می‌نویسد و سپس از این فایل خوانده و در صفحه نمایش چاپ می‌کند .

```
# include "stdio . h"
# include "stdlib . h"
# include "io . h"
void main (void)
  {
    FILE *fp ;
    char  s[80] , number [10] ;
    int  t ;
    if  (( fp = fopen ("test" , "w")) = = NULL)
    {
      printf ("cannot open file\n") ;
      exit (1) ;
    }
    printf ("\n enter string : ") ;
    gets (s) ;
```

```
    strcat (s , "\n") ;
    printf ("\n enter anumber : ") ;
    gets (number) ;
    t = atoi (number) ;
    fprintf (fp , "%s%d , s , t) ;
    fclose (fp) ;
    if  ((fp = fopen ("test" , "r")) = = NULL)
      {
           printf ("cannot open file \n") ;
           exit (1) ;
      }
    fscanf (fp , "%s%d" , &s , &t) ;
    printf ("\nstring = %s , digit = %d" , s , t) ;
  }
```

در مورد توابع fprintf و fscanf باید توجه داشت که علیرغم اینکه ورودی خروجی با این دو تابع آسان است ، اما اطلاعات به همان صورتی که در صفحه نمایش ظاهر می‌شوند در فایل ذخیره می‌گردند . بعنوان مثال ، عدد ۲۶۷ که در صفحهٔ نمایش ، ۳ بایت را اشغال می‌کند اگر توسط تابع fprintf بر روی فایل نوشته شود نیز ۳ بایت را اشغال خواهد کرد (توجه داریم که عدد ۲۶۷ یک عدد صحیح است و می‌تواند در دو بایت ذخیره شود) . این بدین معنی است که هر رقم به‌صورت کاراکتر تلقی می‌گردد . اگر این عدد توسط تابع fscanf از روی فایل خوانده شود ، باید عمل تبدیل کاراکتر به عدد صورت گیرد که مستلزم صرف وقت است . برای جلوگیری از این مشکل پیشنهاد می‌شود که از دو تابع fread و fwrite که در ادامه بررسی خواهند شد استفاده گردد .

- **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 بر روی فایل بکار می‌روند . البته این دو تابع از سرعت کمی برخوردارند که توصیه می‌شود از آنها استفاده نگردد . برای ورودی خروجی رکورد و همچنین سایر ورودی خروجیها می‌توان از دو تابع fread و fwrite استفاده نمود که از سرعت بالایی برخوردارند . الگوی این تابع در فایل stdio.h قرار داشته و بصورتهای زیر می‌باشند :

```
int  fread (void *buffer , int num_byte , int count , FILE *fp)

int  fwrite (void *buffer , int num_byte , int count , FILE *fp)
```

در الگوهای فوق ، پارامتر buffer 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 توسط num_byte مشخص گردید و باید در فایل نوشته و یا از فایل خوانده شوند . اشاره‌گر fp به فایلی اشاره می‌کند که توابع fread و fwrite باید بر روی آنها عمل کنند .

مثال – مجموعه دستورات زیر را درنظر بگیرید :

```
char  student [20] ;
char  str [10] ;
fwrite (student , sizeof (char) , 20 , fp) ;
fread (str , sizeof (char) , 10 , fp) ;
```

دستورات اول و دوم رشته‌هایی به طولهای ۲۰ و ۱۰ را تعریف می‌کنند . دستور سوم ، تعداد ۲۰ بایت از اطلاعات موجود در آرایه 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 خوانده و در متغیر str قرار می‌دهد . توابع fread و fwrite بیشتر در ورودی خروجی رکورد مورد استفاده قرار می‌گیرند .

- **دستگاههای ورودی ـ خروجی استاندارد**

وقتی اجرای یک برنامه به زبان C آغاز می‌شود ، پنج فایل بطور اتوماتیک باز می‌شوند که اشاره‌گرهای آنها در جدول زیر مشاهده می‌گردند .

دستگاههای ورودی خروجی استاندارد

اشاره‌گر دستگاه (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)

حال بعنوان مثال ، مجموعه دستورات زیر را در نظر بگیرید :

```
putc (ch , stdout) ;                    (۱)
printf (stdout , "%d , %d" , a , b) ;   (۲)
fscanf (stdin , "%d , %d" , &x , &y) ;  (۳)
```

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 همانطور که بطور اتوماتیک باز می‌شوند ، بطور اتوماتیک نیز بسته خواهند شد و نیاز به بستن آنها توسط برنامه‌نویس نیست .

```
#include<stdio.h>
main ( )
{
  FILE  *fp;
  int  fact , n ;
  scanf("%d",&n) ;
  fact = 1 ;
  while ( n>1 )
     fact * = n - - ;    // calculate factorial
  fp = fopen("fact . txt", "w") ;
  fprintf (fp," %d", fact ) ;
  fclose(fp) ;
  return 0 ;
}
```

توضیح – ابتدا عددی صحیح از ورودی خوانده شده و در متغیر n جای میگیرد . سپس با استفاده از حلقه while فاکتوریل آن محاسبه شده و در متغیر fact جایگزین میشود . حال فایلی جهت نوشتن با نام fact . txt ایجاد شده و با دستور fprintf مقدار فاکتوریل در آن نوشته میشود .

تمرین ۲ – برنامه‌ای بنویسید که رشته ای از ورودی دریافت کند . سپس رشته ورودی را بهمراه معکوس آن رشته در فایلی درج کند .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
#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 – رشته ورودی ، در آرایه ای کارکتری بشکل اشاره گر و بنام msg* قرار میگیرد . سپس با استفاده از حلقه while و تابع putc یکبار از ابتدا تا انتها و بار دیگر از انتها به ابتدا در فایلی متنی بنام str نوشته میشود .

تمرین ۳ – برنامه‌ای بنویسید که فایلی متنی به حجم ۱۰ بایت ایجاد کند .

حل : برنامه مورد نظر در زیر نمایش داده شده است :

```
#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 آنجاییکه هر کارکتر یک بایت حافظه اشغال میکند ، پس کافیست فایلی متنی ایجاد کرده و ۱۰ کارکتر در آن بنویسیم. در این برنامه آرایه ای بنام str با ۱۰ کارکتر شامل اعداد ۰ تا ۹ مقداردهی شده ، سپس فایل متنی بنام 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 .

تمرین ٤ – برنامه‌ای بنویسید که برنامهٔ موجود درفایل دیگری را به‌عنوان فایل ورودی ، پذیرفته و تعداد پرانتزهای باز و بسته و همچنین تعداد آکولادهای باز و بسته آنرا شمارش کند . نام فایل ورودی به‌عنوان آرگومان تابع اصلی ، به برنامه وارد شود .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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) ;
  }
```

تمرین ۵ – برنامه‌ای بنویسید که مشخصات شغلی کارمندان یک سازمان را که شامل نام ، تعداد ساعت کار و کارمزد ساعتی میباشد ، دریافت کرده و در یک فایل قرار دهد . سپس با استفاده از این اطلاعات ، حقوق دریافتی آنها را محاسبه کرده و چاپ کند .

حل : برنامه مورد نظر در زیر نشان داده شده است :

```
# 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

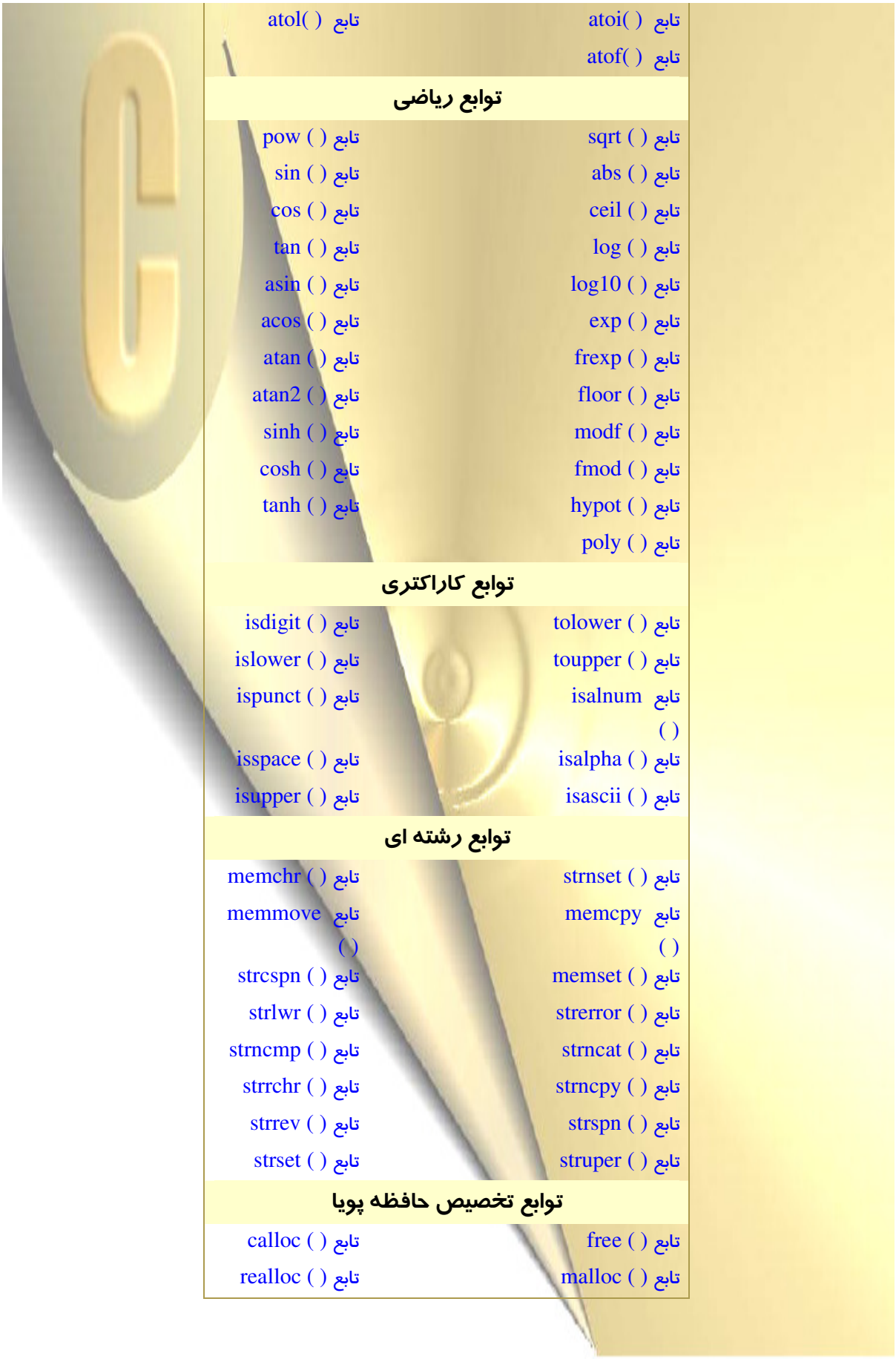
name	salary
amir	36000
hamid	45000
majid	25000

توابع کتابخانه‌ای

زبان C توسط یکسری توابع کتابخانه ای که عملیات و محاسبات پر کاربرد را انجام میدهند کامل میشود . این توابع کتابخانه ای به خودی خود قسمتی از زبان نیستند هر چند که همه مکمل های زبان آنها را شامل میشوند . بعضی توابع مقداری را باز میگردانند ، برخی دیگر با بازگرداندن ۱ یا ۰ نشان میدهند که شرط درست است یا نه ، برخی دیگر نیز عملیات خاصی را بر روی داده ها انجام میدهند . معمولاً عملیاتی که وابسته به نوع کامپیوتر میباشند بصورت توابع کتابخانه ای نوشته میشوند .

بطور کلی می‌توان گفت که توابع در زبان C از تنوع زیادی برخوردار بوده ، جهت انجام محاسبات ریاضی ، اعمال کاراکتری ، مقایسه و تغییرات رشته‌ای ، تبدیل نوعهای مختلف به یکدیگر ، ترسیمات گرافیکی ، کار بر 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 یک فایل header جای دارد . یک تابع کتابخانه ای بسادگی با نوشتن اسم تابع بهمراه لیست آرگومانهایی که بیان کننده اطلاعات منتقل شونده به تابع هستند مورد دسترسی قرار میگیرد . آرگومانها باید در داخل پرانتز ها قرار گرفته و بوسیله کاما از هم جدا شوند . در ادامه ضمن بررسی برخی توابع ، الگو و نیز فایل header آن معرفی می‌گردد . با کلیک روی نام هر تابع توضیحات مربوطه نمایش داده میشود .

توابع تبدیل نوع	



تابع atoi( )	تابع atol( )
تابع atof( )	

توابع ریاضی

تابع sqrt ( )	تابع pow ( )
تابع abs ( )	تابع sin ( )
تابع ceil ( )	تابع cos ( )
تابع log ( )	تابع tan ( )
تابع log10 ( )	تابع asin ( )
تابع exp ( )	تابع acos ( )
تابع frexp ( )	تابع atan ( )
تابع floor ( )	تابع atan2 ( )
تابع modf ( )	تابع sinh ( )
تابع fmod ( )	تابع cosh ( )
تابع hypot ( )	تابع tanh ( )
تابع poly ( )	

توابع کاراکتری

تابع tolower ( )	تابع isdigit ( )
تابع toupper ( )	تابع islower ( )
تابع isalnum ( )	تابع ispunct ( )
تابع isalpha ( )	تابع isspace ( )
تابع isascii ( )	تابع isupper ( )

توابع رشته ای

تابع strnset ( )	تابع memchr ( )
تابع memcpy ( )	تابع memmove ( )
تابع memset ( )	تابع strcspn ( )
تابع strerror ( )	تابع strlwr ( )
تابع strncat ( )	تابع strncmp ( )
تابع strncpy ( )	تابع strrchr ( )
تابع strspn ( )	تابع strrev ( )
تابع struper ( )	تابع strset ( )

توابع تخصیص حافظه پویا

تابع free ( )	تابع calloc ( )
تابع malloc ( )	تابع realloc ( )

تابع clrscr( )

این تابع یکی از پر کاربرد ترین توابع زبان C بوده و برای پاک کردن صفحه نمایش در خروجی در مد متنی بکار می رود و آرگومان ندارد. الگوی آن بشکل زیر است و در فایل conio.h قرار دارد :

```
void  clrscr (void )
```

توابع تبدیل نوع

این توابع در زبان C برای انجام تبدیل نوع داده ها به یکدیگر بکار میروند و در فایل stdlib.h قرار دارند .

تابع atoi( )

این تابع برای تبدیل نوع رشته به نوع integer بکار میرود . الگوی آن بشکل زیر است :

```
int  atoi ( const  char  *s)
```

تابع atof( )

این تابع برای تبدیل نوع رشته به نوع ممیز شناور بکار میرود . الگوی آن بشکل زیر است :

```
double  atof ( const  char  *s)
```

تابع atol( )

این تابع برای تبدیل نوع رشته به نوع long بکار میرود . الگوی آن بشکل زیر است :

```
long  atol ( const  char  *s)
```

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 یک عدد مثبت را محاسبه می‌کند و الگوی آن بصورت زیر است :

```
double  sqrt (double  x)
```

تابع pow ( )

این تابع توانهای یک مبنا را محاسبه می‌کند و الگوی آن بصورت زیر می‌باشد :

```
double  pow (double  x , double  y)
```

نتیجهٔ تابع ، عبارت x^y است . اگر مبنا صفر باشد و یا توان منفی یا صفر باشد این تابع عمل نخواهد کرد .

تابع abs ( )

این تابع برای محاسبهٔ قدر مطلق اعداد صحیح بکار می‌رود . اگر آرگومان این تابع یک عدد منفی باشد نتیجهٔ حاصل از تابع یک عدد مثبت است و اگر آرگومان تابع ، مثبت و یا صفر باشد نتیجه ، یک عدد مثبت یا صفر خواهد بود . الگوی این تابع که در فایل stdlib. h و نیز فایل math.h وجود دارد و بصورت زیر است :

```
int  abs (int  x)
```

تابع fabs ( )

این تابع برای محاسبهٔ قدرمطلق اعداد اعشاری مورد استفاده قرار می‌گیرد و الگوی آن بصورت زیر است :

```
double  fabs (double  x)
```

تابع cabs ( )

این تابع برای محاسبهٔ قدرمطلق اعداد موهومی بکار می‌رود . الگوی این تابع در فایل math. h قرار دارد . ساختمان اعداد موهومی بصورت زیر تعریف شده است :

```
struct  complex {
double  x ;
double  y ;
} ;
```

الگوی این تابع بصورت زیر است :

```
double  cabs (struct  complex  num)
```

تابع sin ( )

این تابع برای محاسبه سینوس یک زاویه برحسب رادیان بکار می‌رود و دارای الگوی زیر است :

double sinh (double arg)

تابع cos ( )

این تابع برای محاسبهٔ کسینوس یک زاویه برحسب رادیان بکار می‌رود و الگوی آن بصورت زیر است :

double cos (double arg)

تابع tan ( )

این تابع برای محاسبه تانژانت یک زاویه که برحسب رادیان می‌باشد بکار می‌رود و الگوی آن بصورت زیر است :

double tan (double arg)

تابع asin ( )

این تابع برای محاسبه آرک سینوس یک عدد بکار رفته و الگوی آن بصورت زیر است :

double asin (double arg)

مقادیری که آرگومان این تابع قبول می‌کند در بازه ۱- تا ۱ است و نتیجه حاصل بصورت رادیان و در بازه $\pi/2$ و $\pi/2$ است .

تابع acos ( )

این تابع برای محاسبهٔ آرک کسینوس یک عدد مورد استفاده قرار می‌گیرد و الگوی آن بصورت زیر است :

double acos (double arg)

مقادیری را که آرگومان این تابع می‌پذیرد در بازه ۱- تا ۱ است و نتیجهٔ حاصل بصورت رادیان و در بازه صفر و π است .

تابع atan ( )

این تابع برای محاسبه آرکتانژانت یک عدد بکار می‌رود و الگوی آن بصورت

زیر تعریف شده است :

double atan (double arg)

مقادیری که توسط این تابع محاسبه می‌شوند بصورت رادیان و در بازه $-\pi/2$ و $\pi/2$ است .

تابع atan2 ( )

این تابع دو آرگومان را دریافت کرده و اولین آرگومان را بر دومین آرگومان تقسیم کرده و آرک‌تانژانت نتیجه حاصل را محاسبه می‌کند . الگوی تابع بصورت 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 یک زاویه برحسب رادیکان بکار رفته و الگوی آن بصورت زیر است :

double sinh (double arg)

تابع cosh ( )

این تابع برای محاسبهٔ کسینوس هیپربولیک یک عدد بکار می‌رود و الگوی آن بصورت زیر است :

double cosh (double arg)

تابع tanh ( )

این تابع برای محاسبه تانژانت هیپربولیک یک زاویه برحسب رادیان بکار می‌رود و الگوی آن بصورت زیر است :

double tanh (double arg)

تابع ceil ( )

این تابع کوچکترین عدد صحیح بزرگتر یا مساوی با یک عدد را که بعنوان آرگومان آن می‌باشد محاسبه می‌کند و الگوی آن بصورت زیر است :

double ceil (double x)

تابع ( ) log

این تابع لگاریتم طبیعی یک عدد مثبت را محاسبه می‌کند.

(پایه لگاریتم طبیعی عدد e = ۲/۷۱۸۲۰۰۰ **است) الگوی این تابع بصورت زیر است :**

double log (double x)

تابع ( ) log10

این تابع لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می‌کند . الگوی این تابع بصورت زیر است :

double log10 (double x)

تابع ( ) exp

این تابع برای محاسبهٔ توانی از e **(پایه لگاریتم طبیعی) مورد استفاده قرار گرفته و الگوی آن بصورت 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 بصورت کسری و در بازه ۰.۵ تا کمتر از یک (۱) ، و قسمت دوم بصورت توان است . الگوی تابع بصورت زیر است :

double frexp (double num , int *exp)

تابع frexp **عدد** num **را را بصورت** $num = کسر * 2^{exp}$ **درمی‌آورد که قسمت کسر بعنوان نتیجه عمل توسط تابع برگردانده می‌شود و قسمت توان در متغیر** exp **قرار می‌گیرد .**

تابع ( ) ldexp

این تابع دارای دو آرگومان است و الگوی آن بصورت زیر است :

double ldexp (double num , int exp)

نتیجهٔ حاصل از 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 بصورت 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 بصورت زیر است :

double fmod (double x , double y)

تابع modf ( )

این تابع یک عدد را به دو قسمت صحیح و اعشاری تجزیه می‌کند و الگوی آن بصورت زیر است

double modf (double x , int *y)

قسمت اعشاری عدد x **به‌عنوان نتیجه عمل تابع برگردانده می‌شود 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 بصورت زیر است :

double hypot (double x , double y)

در الگوی فوق ، x **و** y **دو ضلع قائم مثلث قائم‌الزاویه می‌باشند .**

تابع poly ( )

این تابع برای ارزیابی یک چندجمله‌ای بکار می‌رود و دارای الگوی زیر است :

double poly (double x , int n , double C[ ])

در این الگو ، n **تعداد جمله ،** C **آرایه‌ای است که ضرایب چندجمله‌ای را نگهداری می‌کند و** x **درجه چندجمله‌ای را مشخص می‌نماید . بعنوان مثال ، اگر** n = 3 **باشد . چندجمله‌ایی که ارزیابی می‌شود بصورت زیر است :**

```
C[3]x + C[2]x + C[1]x + C[0]
```

توابع کاراکتری

توابع کاراکتری برای ورودی ـ خروجی کاراکترها و مقایسهٔ آنها با یکدیگر ، تبدیل حروف کوچک و بزرگ به یکدیگر مورد استفاده قرار می‌گیرند .

الگوی این توابع در فایل ctype. h **قرار دارد .**

تابع ( ) tolower

این تابع کاراکتری را بعنوان آرگومان پذیرفته و آن را به حرف کوچک انگلیسی تبدیل می‌کند . الگوی تابع بصورت زیر است :

```
int  tolower (int  ch)
```

تابع ( ) toupper

این تابع کاراکتری را بعنوان آرگومان پذیرفته 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 بصورت زیر است :

```
int  toupper (int  ch)
```

تابع ( ) isalnum

این تابع کاراکتری را بعنوان آرگومان می‌گیرد . اگر این کاراکتر هیچکدام از حروف a **تا** z (A **تا** Z) **و یا ارقام** ۰ **تا** ۹ **نباشد صفر را بعنوان عمل برمی‌گرداند و در غیر اینصورت نتیجهٔ برگردانده شده توسط این تابع غیر از صفر خواهد بود . الگوی این تابع بصورت زیر است :**

```
isalnum (int  ch) int
```

تابع ( ) isalpha

این تابع کاراکتری را بعنوان آرگومان پذیرفته و تشخیص می‌دهد که این کاراکتر از حروف a **تا** z **و یا** A **تا** Z **هست یا نه . اگر این کاراکتر یکی از این حروف نباشد نتیجه حاصل از تابع ، عدد صفر و در غیر اینصورت صفر نیست . الگوی این تابع بصورت زیر است :**

```
int  isalpha (int ch)
```

تابع ( ) isascii

این تابع کاراکتری را بعنوان آرگومان پذیرفته و تشخیص می‌دهد که آیا این کاراکتر در بازهٔ صفر تا Ox7f هست یا خیر . اگر چنین نباشد نتیجه حاصل از تابع برابر با صفر وگرنه غیر از صفر خواهد بود . الگوی این تابع بصورت زیر است :

int isascii (int ch)

تابع ( ) isdigit

این تابع کاراکتری را به‌عنوان آرگومان پذیرفته 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 بصورت زیر است :

int isdigit (int ch)

تابع ( ) islower

این تابع کاراکتری را از ورودی خوانده 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 بصورت زیر است :

int islower (int ch)

تابع ( ) ispunct

این تابع کاراکتری را بعنوان آرگومان پذیرفته و تشخیص می‌دهد که آیا این کاراکتر یکی از کاراکترهای ویرایش مثل کاما ، نقطه و غیره هست یا خیر و اگر نباشد حاصل کار تابع ، عدد صفر وگرنه غیر از صفر خواهد بود . الگوی تابع بصورت زیر است :

int ispunct (int ch)

تابع ( ) isspace

این تابع کاراکتری را بعنوان آرگومان پذیرفته و مشخص می‌کند که آیا این کاراکتر یکی از کاراکترهای فضای خالی و یا:

new line ، carriage return ، form feed ، vertical tab ، horizontal tab

هست یا خیر و اگر نباشد ، حاصل کار تابع صفر است و در غیر اینصورت مخالف صفر خواهد بود . الگوی این تابع بصورت زیر است :

```
int  isspace (int  ch)
```

تابع ( ) isupper

این تابع کاراکتری را بعنوان آرگومان پذیرفته و تشخیص می‌دهد که آیا این کاراکتر از حروف بزرگ انگلیسی (A تا Z) هست یا خیر. و اگر نباشد حاصل کار تابع صفر است و در غیر اینصورت مخالف صفر خواهد بود . الگوی تابع بصورت زیر است :

```
int  isupper (int  ch)
```

توابع رشته‌ای

این توابع معمولاً برای مقایسه و جستجوی کاراکترها و یا رشته‌ای در رشتهٔ دیگر بکار می‌روند . الگوی توابع رشته‌ای در فایل "string. h" قرار دارد .

تابع ( ) strset

این تابع محتویات یک رشته را با یک کاراکتر پر می‌کند . الگوی این تابع بصورت زیر است:

```
strset (str , 'x') ;
```

این تابع رشته str را با x پر می‌کند .

تابع ( ) strnset

این تابع می‌تواند یک کاراکتر را به تعداد دفعات مشخصی در یک رشته کپی کند . الگوی این تابع بصورت 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 رشته مورد نظر کپی شود .

تابع ( ) memchr

این تابع کاراکتری را در یک آرایه مورد جستجو قرار می‌دهد و محل اولین وقوع

آن را مشخص می‌کند . اگر این کاراکتر پیدا شد شمارهٔ آن محل را در یک اشاره‌گر کاراکتری قرار می‌دهد وگرنه کاراکتر تهی را در اشاره‌گر منظور می‌کند . الگوی تابع بصورت زیر است :

void *memchr (const void *buffer , int ch , unsigned count)

در الگوی فوق buffer **به آرایه‌ای اشاره می‌کند که عمل جستجو باید در آن انجام شود و** ch **کاراکتری را مشخص می‌کند که باید در آرایه مورد جستجو قرار گیرد .** count **محلی را در آرایه مشخص می‌کند که عمل جستجو باید از ابتدای آرایه تا آن محل انجام شود .**

تابع ( ) memcpy

این تابع قسمتی از یک آرایه را در آرایه دیگر کپی می‌کند . الگوی این تابع بصورت زیر است :

void *memcpy (void *to , const void *from , unsigned count)

در الگوی فوق ، from **اشاره‌گری است که به آرایه منبع (آرایه‌ای که کاراکترهای مورد نظر در آنجا قرار دارند) اشاره می‌کند و** to **اشاره‌گری است که به آرایه مقصد (آرایه‌ای که کاراکترها باید به آنجا کپی شوند) اشاره می‌کند . تعداد کاراکترهایی که باید کپی شوند توسط آرگومان** count **مشخص می‌شود . بعنوان مثال ، اگر** count **برابر با ۱۰ باشد یعنی اولین ۱۰ کاراکتر آرایه** from **باید به ده محل اول آرایه** to **کپی شوند. همانطوری که از الگوی این تابع پیداست این تابع یک اشاره‌گر است که پس از مقایسه ، به آرایه‌ای اشاره می‌کند که** to **به آن اشاره می‌کرده است.**

تابع ( ) memmove

این تابع تعدادی از کاراکترها را از یک آرایه به آرایه دیگر کپی می‌کند . تفاوت آن با تابع memcpy **در این است که اگر در تابع** memcpy **دو آرایه روی هم قرار گرفته باشند عمل کپی انجام نمی‌شود ، ولی در تابع** memmove **عمل کپی به درستی انجام می‌شود . الگوی این تابع بصورت 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 آرایه منبع به آرایه مقصد کپی شوند .**

تابع ( ) memset

این تابع کاراکتری را در چند عنصر آرایه کپی می‌کند . الگوی این تابع بصورت زیر است :

void *memset (void *buf , int ch , unsigned count)

در الگوی فوق اشاره‌گر buf **به آرایه‌ای اشاره می‌کند که عمل کپی باید در آن انجام شود و** ch **به کاراکتری اشاره می‌کند که باید در آرایه کپی شود و** count **مشخص می‌کند که کاراکتر** ch **باید در چند عنصر آرایه (با شروع از اولین محل) کپی گردد. این تابع معمولاً برای ارزش‌دهی اولیه آرایه‌ها استفاده می‌شود .**

تابع ( ) strcspn

این تابع رشته‌ای را در یک رشته دیگر جستجو می‌کند و اولین محلی از این رشته را که حتی یکی از کاراکترهای رشتهٔ مورد جستجو در آن محل باشند ، بعنوان نتیجهٔ عمل برمی‌گرداند . الگوی این تابع بصورت 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 ، بعنوان نتیجهٔ عمل تابع برگردانده می‌شود .**

تابع ( ) strerror

این تابع موجب می‌شود تا بتوان در اثر بروز خطایی در برنامه ، پیام خطایی را صادر کرد . نوع پیام خطا در اختیار برنامه‌نویس است . الگوی این تابع بصورت زیر است :

char *strerror (char *str)

تابع ( ) strlwr

این تابع رشته‌ای را بعنوان آرگومان پذیرفته و کلیهٔ حروف بزرگ آن را به کوچک تبدیل می‌کند و دارای الگوی زیر است :

char *strlwr (char *str)

در الگوی فوق ، str **، به رشته‌ای اشاره می‌کند که کلیهٔ حروف بزرگ موجود در آن**

باید به حروف کوچک تبدیل گردند .

تابع strncat ( )

این تابع قسمتی از یک رشته را به انتهای رشتهٔ دیگر الحاق می‌کند و سپس رشتهٔ نتیجه را به کاراکتر تهی ختم می‌نماید . الگوی تابع بصورت 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 . (زیرا رشته‌ای از یک رشته را با زیررشته‌ای از رشتهٔ دیگر مقایسه می‌نماید) . الگوی این تابع بصورت 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 شده و یک عدد صحیح بعنوان نتیجه عمل برگردانده می‌شود .

تابع strncpy ( )

این تابع تعداد مشخصی از کاراکترهای یک رشته را در یک رشته دیگر کپی می‌کند . الگوی این تابع بصورت 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 یک رشته جستجو می‌کند و محل آخرین وقوع این کاراکتر را در رشته پیدا کرده و به یک اشاره‌گر نسبت می‌دهد . اگر این کاراکتر پیدا نشود کاراکتر تهی در اشاره‌گر قرار خواهد گرفت . الگوی این تابع بصورت زیر

است :

```
char *strrchr (const  char *str , int ch)
```

تابع ( ) strspn

این تابع رشته‌ای را در یک رشتهٔ دیگر جستجو می‌کند و باعث برگشت طول زیررشتهٔ حاوی همهٔ کاراکترهای رشتهٔ مورد جستجو می‌شود . الگوی این تابع بصورت 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 یک 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 یک رشتهٔ کاراکتری ، کلیهٔ حروف کوچک را به حروف بزرگ تبدیل می‌کند . الگوی تابع بصورت 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 .

توابع تخصیص حافظه پویا

متغیرهای معمولی و آرایه‌ها در حین ورود به بلوک ایجاد شده و تا خاتمه اجرای بلاک 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 گفته می‌شود . 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 بشکل 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 بکار می رود 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 یک آرایه (شامل** num **عنصر که هر عنصر آن دارای طولی برابر با** size **باشد) استفاده می‌گردد . آدرس اولین بایت حافظه‌ای که توسط این تابع در اختیار استفاده‌کننده قرار می‌گیرد در یک اشاره‌گر قرار داده می‌شود . اگر این اشاره‌گر تهی باشد به معنی تخصیص نیافتن این حافظه است (عدم تخصیص حافظه ممکن است بدلیل 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 بکار رفته و دارای الگوی زیر است :

void *malloc (unsigned size)

الگوی این تابع در فایل "stdio . h" **قرار دارد . میزان حافظه‌ای که توسط این تابع از سیستم اخذ می‌شود توسط آرگومان** size **مشخص می‌گردد . آدرس حافظه گرفته شده از سیستم توسط تابع** malloc **در یک 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 توسط توابع تخصیص حافظه مثل malloc **، بکار می‌رود . الگوی این تابع بصورت زیر است و در فایل** "stdlib . h" **قرار دارد :**

*realloc (void *ptr , unsigned size)

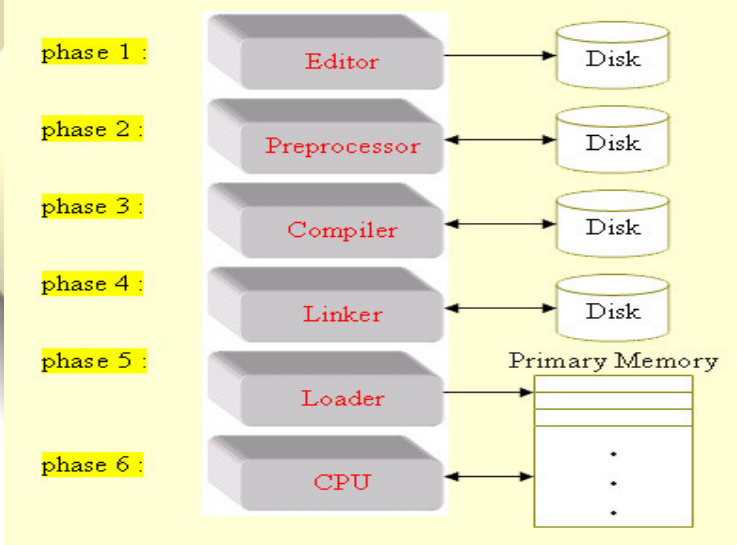
برای تغییر میزان حافظهٔ تخصیص یافته ، تنها ذکر اشاره‌گر به آن حافظه و همچنین اندازهٔ جدید این حافظه بعنوان آرگومان این تابع کافی است . میزان جدید حافظه ممکن است کمتر و یا بیشتر از حافظهٔ تخصیص یافته باشد . درصورت وجود اطلاعات در حافظهٔ تخصیص یافتهٔ قبلی ، آنها از بین نخواهند رفت . تابع realloc **پس از اخذ حافظه از سیستم ، آدرس آن را در یک اشاره‌گر قرار می‌دهد . اگر این اشاره‌گر تهی باشد ، بدین معنی است که حافظه ، تخصیص نیافته است .**

محیط توربو C (برای پیاده‌سازی و اجرای برنامه‌ها)

- **مقدمه**

یک برنامه C ، شش فاز یا مرحله را طی می‌کند تا بطور کامل اجرا شود و خروجی خود را در اختیار کاربر یا برنامه ساز قرار دهد . نقش هر مرحله یا فاز ، مطابق شکل زیر به اختصار بیان می‌گردد .

مراحل ایجاد یک برنامه



در اولین مرحله که ویرایش است ، به کمک یک ویرایشگر یا ویراستار ، برنامهٔ مورد نظر که آن را برنامه منبع یا source program **نامند تایپ می‌گردد و غلطهای املایی یا دستوری آن به کمک ویراستار تصحیح می‌شود و نتیجه نهایی بصورت یک فایل روی حافظهٔ جانبی که معمولاً دیسک است ، ذخیره می‌گردد . پسوند این فایل برحسب محیط های مختلف برنامه‌سازی ، ممکن است** c **یا** cpp **و مشابه آن باشد .**

در مرحلهٔ دوم ، برنامه‌ساز فرمانی برای ترجمهٔ برنامه صادر می‌کند و کامپایلر ، برنامهٔ مورد نظر را به زبان ماشین ترجمه می‌کند که نتیجهٔ حاصل را برنامهٔ هدف یا object code **نامند که پسوند فایل مورد نظر در این مرحله نیز معمولاً** obj **می‌باشد .**

در روند برنامه‌سازی C ، قبل از اینکه فاز ترجمه شروع شود یک برنامه به نام پیش‌پردازنده یا preprocessor **بطور اتوماتیک اجرا می‌شود . پیش‌پردازندهٔ C ، فرامین خاصی**

را که معمولاً preprocessor directives نامیده می‌شوند ، 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 . متداول‌ترین این فرامین در مبحث پیش‌پردازنده مورد بحث قرار می‌گیرد . قبل از تبدیل برنامه به زبان ماشین ، پیش‌پردازنده بطور اتوماتیک توسط کامپایلر احضار می‌گردد .

مرحلهٔ بعدی مرحلهٔ الحاق کردن یا پیوند دادن یا Linking می‌باشد . بطور متعارف برنامه‌های C ، شامل مراجعه به توابعی است که در جای دیگری مانند کتابخانه‌های استاندارد ، یا کتابخانه‌های برنامه‌سازان دیگر که روی پروژه‌های خاص کار می‌کنند، تعریف شده‌اند . برنامهٔ هدف یا object code ایجاد شده با کامپایلر ، به‌طور نمونه دارای قسمتهای خالی به‌علت نبودن یا عدم وجود این گونه توابع است که در برنامهٔ مورد نظر به آنها مراجعه شده است . نرم افزار linker یا پیونددهنده ، کد توابع مورد مراجعه را به object code برنامهٔ مورد نظر ما در محلهای مربوط الحاق و ضمیمه می‌کند تا یک برنامهٔ قابل اجرا ایجاد شود . برنامه یا فایل حاصل ، پسوند exe ، خواهد داشت.

مرحلهٔ پنجم ، بار کردن (load) برنامهٔ قابل اجرا 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 حافظه اصلی قرار می‌دهد .

بالاخره کامپیوتر تحت کنترل cpu ، برنامه را اجرا می‌کند که طبیعتاً در این مرحله ، باید داده‌های مورد نیاز نیز در اختیار آن قرار گیرد .

- **نصب توربو C**

نسخه های متعددی از محیط برنامه نویسی زبان C وجود دارد مانند Turbo C و Borland C. در اینجا محیط برنامه نویسی :

Turbo C (Version 3.0)

مورد بررسی قرار میگیرد .برای نصب این نرم افزار می‌توانید از فایل install.exe که همراه توربو C ارائه میشود ، استفاده کنید . در توضیحات موجود در راهنمای کاربران توربو C فایلی با نام Readme نحوهٔ استفاده از این برنامه را شرح داده است . شیوهٔ کار بسیار ساده بوده و به این ترتیب است که 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 راه اندازی سیستم توسط این نرم‌افزار ، تکمیل شود . پس از نصب چند زیر فهرست به نامهای :

Bgi , Bin , Lib , Classlib , Include

در فهرست اصلی ساخته میشود که فایل اجرایی نرم افزار در زیرفهرست Bin و بنـام tc.exe وجود دارد .

- **نحوهٔ استفاده از نرم افزار**

برای ایجاد برنامه‌ها از Integrated Development Environment ویا به اختصار IDE موجود در محیط توربو C استفاده می کنیم که بسیار شبیه محیط توربو پاسکال است . در این محیط ، تمام عملیات لازم برای ایجاد و تکمیل یک برنامه به‌صورت یکنواخت و روی یک فرم در روی صفحهٔ نمایش ، از طریق منوها و پنجره‌ها در دسترس است . IDE ، تقریباً یک محیط ایده‌آل 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 کرده و فایل اجرایی ساخت و آن را اجرا نمود . برای فعال کردن IDE به زیرفهرستی که توربو C در آن قرار دارد تغییر مسیر دهید ؛ سپس فایل tc. exe را اجرا کنید . بعنوان مثال اگر فهرست نصب شده TC باشد جلوی خط فرمان Dos بشکل زیر تایپ کنید :

D : \ TC \ BIN \ tc.exe

و سپس کلید Enter را فشار دهید . پس از نصب وقتی که توربو C برای اولین بار اجرا می‌شود ، صفحهٔ نمایش را به صورت شکل زیر مشاهده خواهید کرد :

نمایش IDE

وقتی که برای اولین بار IDE را احضار می‌کنید منوبار فعال می‌شود اگر اینطور نبود ، 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 به 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 و یا برای اینکه آن منو را فعال کنید کافی است کلید Enter را فشار دهید که در این صورت منوی مورد نظر باز شده و محتوای آن را ملاحظه خواهید کرد . درواقع هریک از منوهای اصلی دارای یک یا چند منوی فرعی می‌باشد . برای خروج از یک منو (درواقع بستن یک منو) و برگشت به منوبار ، کلید [F10] را فشار دهید .

می‌توانید با استفاده از کلیدها ، در روی منوها حرکت کرده و آنها را بررسی کنید و تا موقعی که کلید Enter را فشار نداده‌اید ، هیچگونه اتفاقی نمی‌افتد . برای فعال کردن یک منوی پررنگ شده باید کلید Enter را فشار دهید به‌عنوان مثال برای انتقال کنترل از منوبار به پنجرهٔ ویرایش ، با کلیدهای جهت دار ، 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. بطور کلی برای انتخاب منوی اصلی دو راه وجود دارد :

روش اول ـ با کلیدهای جهت دار حرکت کرده و منوی مورد نظر را انتخاب و یا به عبارت دیگر پررنگ کنید و سپس کلید Enter را فشار دهید.

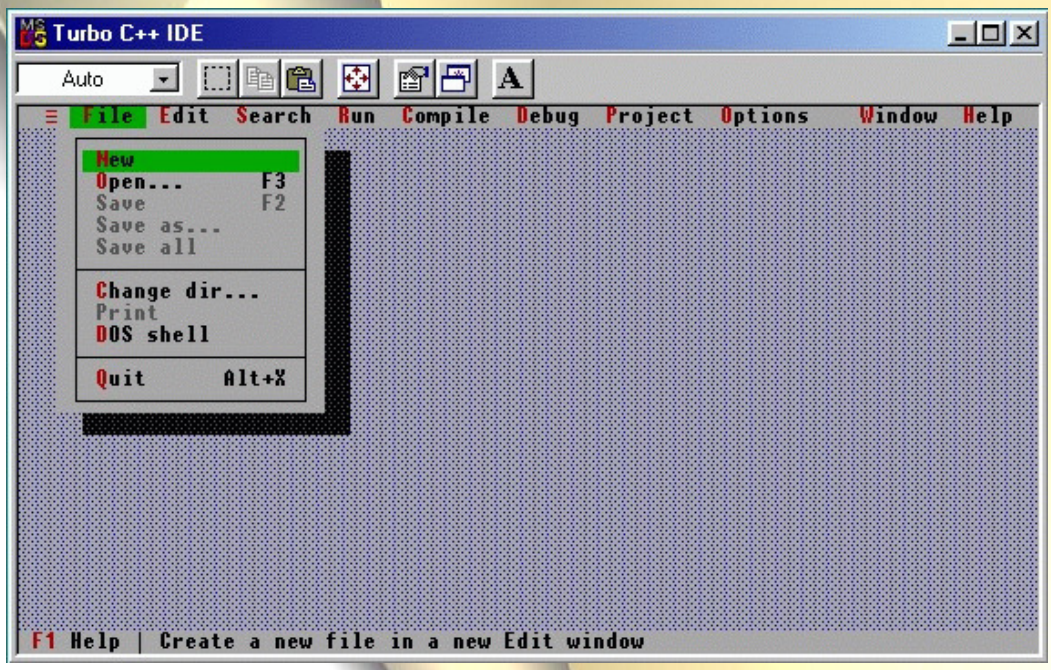
روش دوم ـ اولین حرف نام منو را (که معمولاً به رنگ قرمز می‌باشد) تایپ کنید . به‌عنوان مثال برای انتخاب Edit ، حرف E را تایپ کنید .

در ادامه ، بعضی از منوها که کاربرد بیشتری دارند به اختصار شرح داده میشوند .

- **منوی File**

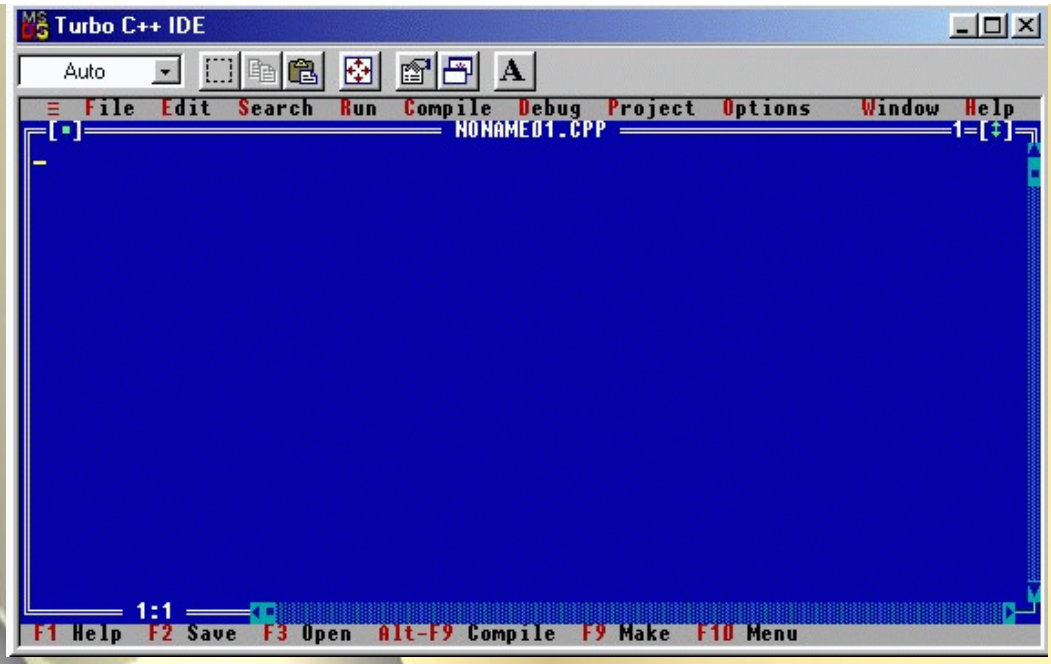
در این منو می‌توان اعمالی مانند : ایجاد فایل جدید ، باز کردن فایل ، ذخیره‌سازی فایل ، چاپ فایل و خروج از نرم افزار را انجام داد . این منو شامل چند گزینه یا فرمان است که در شکل زیر نشان داده شده است :

گزینه‌های منوی File



ـ فرمان New : این فرمان محتوای ویراستار را پاک می‌کند و آن را برای ایجاد و ویرایش فایل جدید آماده کرده فایلی با عنوان noname01.cpp باز می‌کند . 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 صفحه ای مطابق شکل زیر نمایش داده میشود :

نمایش فایل جدید



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 داده 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 کرده و یا از لیست ارائه شده فایل را انتخاب سپس کلید Enter را فشار می‌دهیم .کلید میانبر این فرمان [F3] میباشد .

ـ فرمان Save : این فرمان فایل موجود در محیط توربو C را روی دیسک ذخیره می‌کند . اگر فایل نامی نداشته باشد (یعنی نام آن noname.cpp باشد) نام جدید فایل را از شما می‌پرسد . کلید میانبر این فرمان [F2] میباشد .

ـ فرمان Save as : با اجرای این فرمان میتوان فایل موجود را با اسم جدید ذخیره کرد .

ـ فرمان Print : برای چاپ فایل جاری توسط 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 ، بکار 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 مربوطه 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 اشکال زدایی خطاهای موجود در منطق برنامه مناسب است .کلید میانبر آن [F7] میباشد .

- **منوی Window**

ـ فرمان Close : برای بستن پنجره فایل جاری اسفاده میشود . کلید میانبر آن Alt + F3 میباشد .

ـ فرمان User screen : این فرمان برای نمایش خروجی برنامه اجرا شده مناسب است . کلید میانبر آن Alt + F5 میباشد .

- **منوی Help**

ـ فرمان Index : لیست دستورات 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 : در حالتی که مکان نما ، زیر دستوری قرار داشته باشد با انتخاب این گزینه راهنمای مربوط به آن دستور مستقیماً نمایش داده میشود . کلید میانبر آن Ctrl + F1 میباشد.

- **مراحل ایجاد برنامه در IDE**

برای ایجاد یک برنامه ساده در محیط توربو C به ترتیب زیر عمل میکنیم :

- ایجاد فایل جدید با فرمان New
- نوشتن کد برنامه 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 صورتیکه فایل مورد نظر را با نام test1 **ذخیره کرده باشید در پایان مراحل فوق سه فایل به اسامی :**

test1.cpp **و** test1.obj **و** test1.exe

روی دیسک خواهید داشت که اولی فایل محتوی کد برنامه ، دومی فایل کامپایل شده و سومی فایل اجرایی برنامه خواهد بود .

فهرست منابع

Byron S. Gottfried , Programming With C
Herbert Schildt , Turbo C the complete reference
Herbert schildt , Advanced Turbo C
Kris Jamsa , Turbo C programmer's Library
Kent A. Barclay , C problem solving and programming
N. Kalicharan , C by Example
راهنمای نرم افزار توربو سی نسخه ۳.۰
راهنمای نرم افزار بورلند سی نسخه ۵.۰