بعضی از ویژگیهای زبان C عبارت‌انداز:

- زبان C یک زبان میانی است. زبانهای برنامه‌سازی را می‌توان به سه دسته تقسیم کرد: **زبانهای سطح بالا**، **زبانهای میانی**، **زبانهای سطح پایین** (جدول ۱-۱). علت میانی بودن زبان C این‌است‌که، از طرفی همانند زبان سطح پایینی مثل اسمبلی قادر است مستقیماً به حافظه دستیابی داشته باشد و با مفاهیم بیت، بایت و آدرس کار کند و از طرف دیگر، برنامه‌های این زبان، همچون زبانهای سطح بالایی مثل پاسکال، از قابلیت خوانایی بالایی برخوردارند. به عبارت دیگر، دستورالعملهای این زبان، به زبان محاوره‌ای انسان نزدیک است، که این ویژگی، مربوط به زبانهای سطح بالا است.
- زبان C، یک زبان ساخت یافته است. در این زبان با استفاده از حلقه‌های تکراری مثل while، for و do while
- زبان C، قابل انعطاف و بسیار قدرتمند است. در این زبان، هیچ محدودیتی برای برنامه‌نویس وجود ندارد. هر آنچه را که فکر می‌کنید، می‌توانید در این زبان پیاده‌سازی کنید.
- C، زبان برنامه‌نویسی سیستم است. برنامه‌های سیستم، برنامه‌هایی هستند که امکان بهره‌برداری از سخت‌افزار و سایر نرم‌افزارها را فراهم می‌کنند. بعضی از برنامه‌های سیستم عبارت‌انداز: سیستم عامل، مفسر[1]، کامپایلر، ویراستارها، واژه‌پردازها، مدیریت بانکهای اطلاعاتی و اسمبلر.
- ارتباط تنگاتنگی بین زبان C و اسمبلی وجود دارد و به‌این‌ترتیب می‌توان از تمام قابلیتهای اسمبلی در زبان C استفاده کرد. چگونگی برقراری ارتباط بین این دو زبان، در فصل ۲۱ به طور مفصل مورد بحث قرار می‌گیرد.
- C، زبان قابل حمل است. معنای قابلیت حمل این‌است‌که برنامه‌هایی که به زبان C، در یک نوع کامپیوتر (مثل آی.بی.ام) نوشته شدند، بدون انجام تغییرات یا انجام تغییراتِ اندک، در کامپیوترهای دیگر (مثل VAX و DEC) قابل استفاده‌اند.
- C، زبان کوچکی است. تعداد **کلمات کلیدی**[2] این زبان انگشت‌شمار است (۳۰ کلمه کلیدی - جدول ۳-۱). تصور نشود که هرچه تعداد کلمات کلیدی زبان بیشتر باشد، آن زبان قدرتمند است. به عنوان مثال، زبان بیسیک در حدود ۱۵۰ کلمه کلیدی دارد ولی قدرت زبان C به مراتب بیشتر از زبان بیسیک است. توجه داشته باشید که بعضی از کامپایلرهای C، علاوه براین ۳۲ کلمه کلیدی، کلمات دیگری را به زبان اضافه کرده‌اند (جدول ۴-۱).
- C نسبت به حروف حساس است[3]. یعنی در این زبان، بین حروف کوچک و بزرگ تفاوت است و تمام کلمات کلیدی این زبان با حروف کوچک نوشته می‌شوند. به عنوان مثال، while یک کلمه کلیدی است ولی WHILE اینطور نیست. توصیه می‌شود که تمام برنامه‌های C با حروف کوچک نوشته شوند.

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

انواع داده‌ها

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

تعریف متغیرها

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

نام متغیر نوع داده ;

مثال ۱-۱

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

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

تعریف ثوابت

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

#define <نام ثابت> <مقدار>

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

مثال ۵-۱

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

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

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

const <نوع داده> <نام ثابت> = <مقدار>

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

مثال ۶-۱

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

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

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

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

طراحی الگوریتم

در طراحی الگوریتم برای حل مسئله، لازم است قدم‌به‌قدم **رویه‌هایی**[۲] نوشته شوند-**الگوریتم**- و سپس بررسی شود که آیا الگوریتم، مسئله را به درستی حل می‌کند یا خیر. نوشتن الگوریتم، مشکل‌ترین بخش حل مسئله است. سعی نکنید تمام جزئیات مسئله را حل کنید، بلکه سعی کنید شیوه طراحی **بالا به پایین**[۳] را به کار ببرید. در روش طراحی بالا به پایین، ابتدا مراحل اصلی مسئله که باید حل شوند، مشخص می‌گردند و سپس با حل هر مرحله اصلی، کل مسئله حل می‌شود. اغلب الگوریتمها معمولاً مراحل زیر را دارا هستند:

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

پاک کردن صفحه خروجی

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

خواندن کاراکتر با توابع getch() و getche()

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

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

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

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

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

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

جدول ۱-۲ کاراکترهای‌فرمت در دستور ()printf .

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

جدول ۲-۲ کاراکترهای‌کنترلی در دستور ()printf .

کاراکتر	عملی که باید انجام شود
\f	موجب انتقال کنترل به صفحه جدید می‌شود
\n	موجب انتقال کنترل به خط جدید می‌شود
\t	انتقال به ۸ محل بعدی صفحه‌نمایش
\"	چاپ کوتیشن (")
\'	چاپ کوتیشن (')
\0	NULL (رشته تهی)
\\	back slash
\v	انتقال کنترل به ۸ سطر بعدی
\N	ثابت‌های مبنای ۸ (N عدد مبنای ۸ است)
\xN	ثابت‌های مبنای ۱۶ (N عدد مبنای ۱۶ است)

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

توضیح

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

عرض × طول = مساحت مستطیل

۲ × (عرض + طول) = محیط مستطیل

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

خروجی

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

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

توضیح

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

```
#include <conio.h>
#include <stdio.h>
int main()
{
   int x, y, n;
   float ave;
   clrscr();
   printf("\nEnter three integers:");
   scanf("%d%d%d", &x, &y, &n);
   ave = (float)(x + y + n) / 3;
   printf("\nave=%6.2f", ave);
   getch();
   return 0;
}
```

خروجی

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

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

توضیح

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

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

خروجی

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

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

توضیح

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

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

خروجی

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

ساختارهای تکرار

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

ساختار تکرار for

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

روش اول :

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

روش دوم :

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

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

روش اول :

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

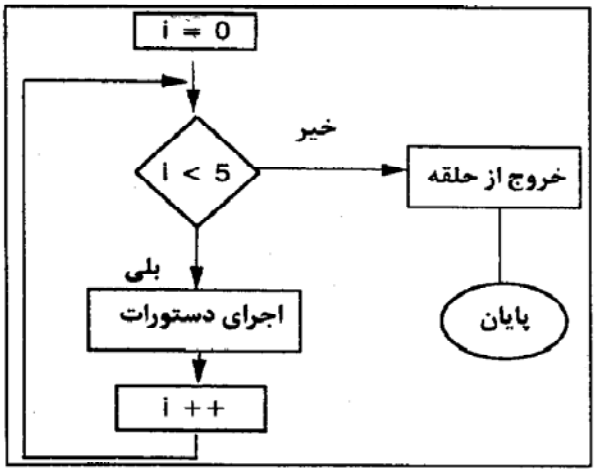
روش دوم :

```
for  (;;)
        ; دستور
```

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

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

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



شکل ۱-۳ شیوه اجرای حلقه for.

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

توضیح

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

متغیرهای برنامه

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

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

خروجی

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

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

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

خروجی

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

حلقه‌های تکرار تودرتو

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

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

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

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

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

توضیح

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

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

خروجی

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

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

توضیح

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

0! = 1

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

متغیرهای برنامه

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

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

نمونه‌هایی از خروجی برنامه

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

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

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

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

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

ساختار تکرار while

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

روش اول :

```
while (شرط)
        دستور ;
```

روش دوم :

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

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

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

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

خروجی

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

ساختار تکرار do ... while

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

روش اول :

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

روش دوم :

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

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

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

توضیح

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

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

خروجی

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

ساختارهای تصمیم

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

ساختار تصمیم if

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

روش اول:

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

روش دوم:

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

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

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

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

خروجی

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

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

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

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

خروجی

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

ساختار تصمیم else if

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

مثال ۱۶-۳

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

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

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

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

خروجی

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

دستور break

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

```
break ;
```

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

مثال ۱۸-۳

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

توضیح

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

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

دستور goto

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

goto <برچسب>

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

مثال ۲۰-۳

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

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

خروجی

The maximum value of x is : 100

ساختار تصمیم switch

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

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

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

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

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

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

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

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

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

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

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

توضیح

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

متغیرهای برنامه

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

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

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

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

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

سری فیبوناچی = ۱ ۱ ۲ ۳ ۵ ۸ ۱۳ ...

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

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

توابع و برنامه‌سازی ساخت‌یافته

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

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

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

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

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

نوشتن توابع

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

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

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

```
<نوع تابع>    نام تابع    (لیست پارامترها);
```

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

شکل ۱-۴ ساختار تابع.

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

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

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

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

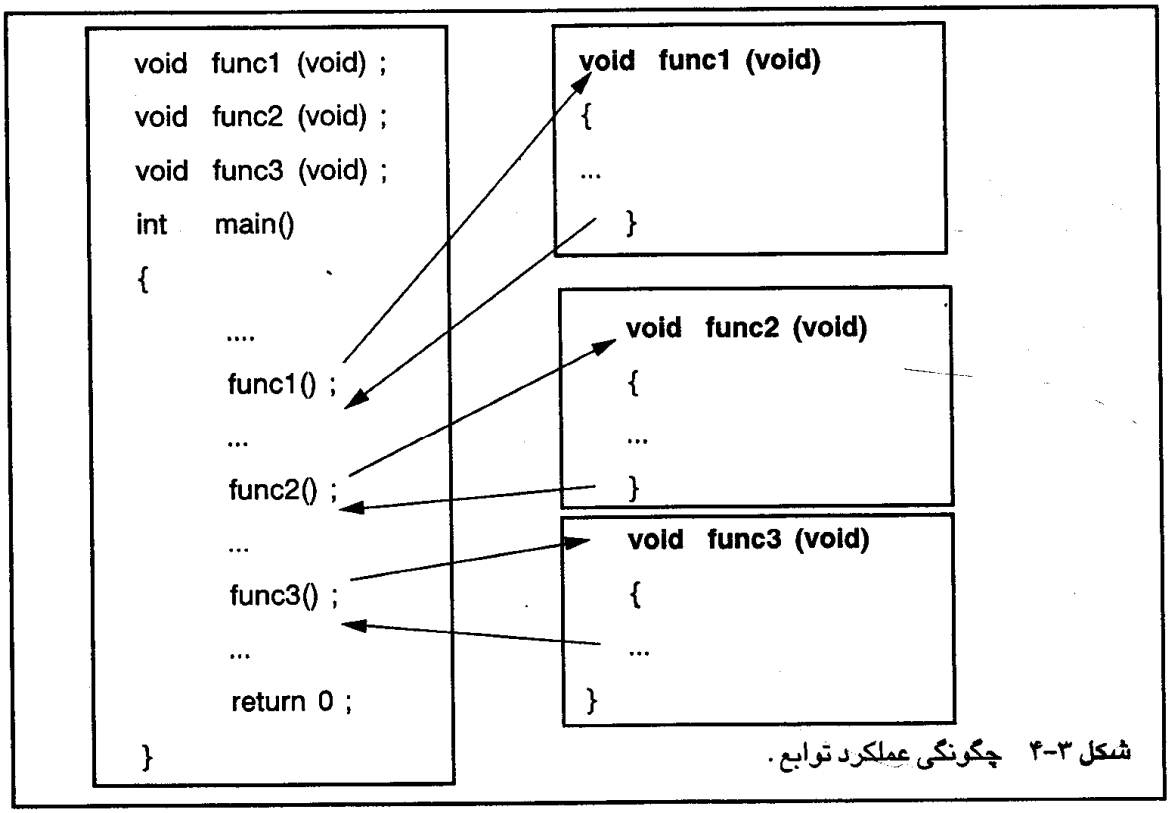
```
#include    <stdio.h>
void    sample (int   x , int   y) ;        الگوی تابع
int     main()
{
        int   a , b ;
        ...                          آرگومان‌های تابع
        sample (a , b) ;                    فراخوانی تابع
        ...
        return  0 ;
}                                   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

شکل ۲-۴
شیوه به کارگیری تابع در برنامه.

تابع چگونه کار می‌کند

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

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



شکل ۳-۴ چگونگی عملکرد توابع.

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

توضیح

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

```
#include <stdio.h>
#include <conio.h>
void findmax(int, int, int);
int main()
{
  int x, y, m;
  clrscr();
  printf("\nEnter three integer numbers:");
  scanf("%d%d%d", &x, &y, &m);
  findmax(x, y, m);
  return 0;
}
//**********************
void findmax(int p1, int p2, int p3)
{
  int maxp;
  maxp = (p1 > p2) ? p1 : p2;
  maxp = (p3 > maxp) ? p3 : maxp;
  printf("\maximum=%d", maxp);
  getch();
}
```

الگوی تابع

عنوان تابع

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

توضیح

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

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

خروجی

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

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

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

مساحت دایره = π * r * r

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

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

خروجی

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

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

توضیح

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

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

خروجی

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

بازگشتی

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

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

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

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

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

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

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

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

خروجي

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

آرایه‌های یک بعدی

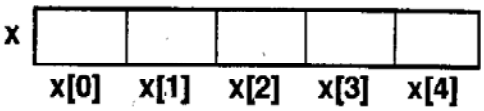
در آرایه یک بعدی که نام دیگر آن لیست است، با یک اندیس می‌توان به عناصر آرایه دست یافت. آرایه‌های یک بعدی در C به صورت زیر تعریف می‌شوند:

[طول آرایه] نام آرایه نوع آرایه

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

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

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



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

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

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

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

مثال ۱-۵

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

توضیح

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

متغیرهای برنامه

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

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

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

توضیح

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

متغیرهای برنامه

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

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

خروجی

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

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

توضیح

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

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

خروجی

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

مرتب‌سازی حبابی -

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

4 9 3 1

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

x[0] با x[1]	**یعنی**	**(۴ با ۹)**	**جابجایی انجام نمی‌شود.**	
x[1] با x[2]	**یعنی**	**(۹ با ۳)**	**جابجایی انجام می‌شود:**	**4 3 9 1**
x[2] با x[3]	**یعنی**	**(۹ با ۱)**	**جابجایی انجام می‌شود:**	**4 3 1 9**

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

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

x[0] با x[1]	**یعنی**	**۴ با ۳**	**جابجایی صورت می‌گیرد:**	**3 4 1 9**
x[1] با x[2]	**یعنی**	**۴ با ۱**	**جابجایی صورت می‌گیرد:**	**3 1 4 9**

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

3 1 4 9

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

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

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

1 3 4 9

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

مثال ۵-۵

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

توضیح

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

متغیرهای برنامه

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

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

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

خروجی

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

جستجو در آرایه

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

جستجوی ترتیبی

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

مثال ۶-۵

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

توضیح

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

متغیرهای برنامه

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

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

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

خروجی

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

جستجوی دودویی

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

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

مثال ۷-۵

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

توضیح

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

متغیرهای برنامه

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

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

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

خروجی

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

آرایه‌های چند بعدی

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

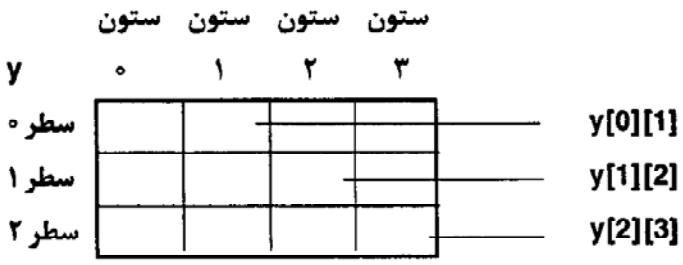
در C، آرایه‌هایی با بیش از دو بُعد (آرایه‌های n بعدی) قابل استفاده‌اند. ولی اغلب برنامه‌نویسان حداکثر از آرایه‌های دوبعدی استفاده می‌نمایند. برای تعریف آرایه دوبعدی در C به صورت زیر عمل می‌شود:

[بُعد ۲] [بُعد ۱] نام آرایه نوع آرایه

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

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

این آرایه را می‌توان به صورت زیر نمایش داد:



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

توضیح

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

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

خروجی

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

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

توضیح

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

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

$z[0][0] = x[0][0] \times y[0][0] + x[0][1] \times y[1][0] = ۱\times۲ + ۳\times۹ = ۲۹$

$z[0][1] = x[0][0] \times y[0][1] + x[0][1] \times y[1][1] = ۱\times۷ + ۳\times۶ = ۲۵$

$z[1][0] = x[1][0] \times y[0][0] + x[1][1] \times y[1][0] = ۴\times۲ + ۵\times۹ = ۵۳$

$z[1][1] = x[1][0] \times y[0][1] + x[1][1] \times y[1][1] = ۴\times۷ + ۵\times۶ = ۵۸$

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

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

برنامه‌ای که تعدادی مقادیر صحیح را از ورودی خوانده آنها را در آرایه‌ای قرار می‌دهد. سپس توسط تابعی، کوچکترین عنصر آرایه را تعیین می‌کند (حداکثر تعداد اعداد، ۲۰ است). آخرین عدد ورودی، صفر است.

توضیح

چون حداکثر تعداد اعداد ۲۰ است، طول آرایه را ۲۰ در نظر می‌گیریم. در برنامه اصلی، عناصر آرایه را از ورودی می‌خوانیم و آرایه را همراه با تعداد واقعی عناصر آن، به تابعی ارسال می‌کنیم. تابع کوچکترین عنصر و محل وجود آن را پیدا می‌کند و به خروجی می‌برد. تابع **()findmin** دو پارامتر دارد. پارامتر **list** مربوط به آرایه و پارامتر **size** مربوط به تعداد عناصر آرایه است.

```
#include <stdio.h>
#include <conio.h>
int findmin(int [], int);
int main()
{
   int list[20], num, size=0 ;
   clrscr();
   do{
     printf("type list[%d] : ", size) ;
     scanf("%d", &list[size]) ;
   } while(list[size ++] != 0) ;
   size --;
   num = findmin(list, size) ;

   printf("\n minimum is:%d ",num) ;
   getch();
   return 0;
}
 //************************
int findmin(int arr[], int size)
{
     int i, min1 ;
     min1 = arr[0] ;
     for(i = 0 ; i < size; i++)
         if(arr[i] < min1)
            min1 = arr[i] ;
     return(min1) ;
}
```

خروجی

```
type list [0] : 10
type list [1] : 5
type list [2] : 18
type list [3] : 0
minimum is : 5
```

رشته‌ها

در زبان C، رشته نوع جدیدی نیست، بلکه به صورت آرایه‌ای از کاراکترها تعریف می‌شود. رشته‌ها برای ذخیره، بازیابی و دستکاری متنها (مثل اسامی افراد)، مورد استفاده قرار می‌گیرند. در C برای تعیین انتهای رشته از کاراکتر خاصی به نام (تهی =NULL) استفاده می‌شود که با '\0' مشخص می‌گردد. بنابراین، آخرین رشته برابر با '\0' می‌باشد. لذا اگر رشته‌ای به نام s را به صورت زیر تعریف کنید، فقط از ۹ کاراکتر می‌توانید استفاده کنید، زیرا کاراکتر آخر '\0' است.

```
char    s[10] ;
```

اگر محتویات این رشته را به طریقی که بعداً گفته می‌شود برابر با "ali" قرار دهید، این رشته به صورت زیر نمایش داده می‌شود:

s[0]	s[1]	s[2]	s[3]	[4]	s[5]	s[6]	s[7]	s[8]	s[9]
a	l	l	\0	?	?	?	?	?	?

در این نمایش، سه کاراکتر اول برای ذخیره رشته به کار رفت و کاراکتر چهارم برابر '\0' است و بقیه کاراکترها مورد استفاده قرار نگرفته‌اند. به‌این‌ترتیب، دریافتید که طول رشته‌ها را باید یک واحد بیش از آنچه که نیاز دارید تعیین کنید.

خواندن رشته با تابع ()gets

علاوه بر امکانات ذکر شده، با تابع ()gets نیز که الگوی آن در فایل stdio.h قرار دارد، می‌توان رشته‌ها را از ورودی خواند. این تابع پس از خواندن رشته، سطر جاری را رد می‌کند. نحوه کاربرد آن به صورت زیر است :

```
gets  (متغیر رشته‌ای) ;
```

در دستورات زیر، رشته str1 تعریف و سپس از ورودی خوانده شد. پس از واردکردن رشته باید کلید Enter را فشار داد.

```
char   str1[21] ;
gets   (str1) ;
```

تفاوت ()gets و ()scanf در خواندن رشته‌ها

در تابع ()gets فقط کلید Enter انتهای رشته را مشخص می‌کند. لذا رشته می‌تواند حاوی فاصله (space) و یا Tab باشد. در حالی که در تابع ()scanf، فاصله و Tab نیز به عنوان جداکننده تلقی شده، انتهای رشته را مشخص می‌کند. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
char   s[51] ;
gets   (s) ;
scanf  ("%s" , s) ;
```

اگر در پاسخ به این دستورات، رشتهٔ "computer science" را وارد کنیم، تابع ()gets کل رشته را می‌خواند، در حالی که تابع ()scanf فقط رشتهٔ "computer" را می‌پذیرد. زیرا فضای خالیِ بعد از آن، به عنوان انتهای رشته تلقی می‌شود.

چاپ رشته با تابع ()puts

این تابع که در فایل stdio.h قرار دارد، برای نوشتن رشته‌ها در خروجی مورد استفاده قرار می‌گیرد. این تابع پس از نوشتن رشته، سطر جاری را رد می‌کند. کاربرد آن به صورت زیر است :

```
puts  ("رشته") ;
puts  (متغیر رشته‌ای) ;
```

به عنوان مثال، دستورات زیر را در نظر بگیرید :

```
char   name = "ali" ;
puts   ("your name is :") ;
puts   (name) ;
```

با دستور اول، رشته "ali" در name قرار می‌گیرد. دستور دوم پیامی را در خروجی چاپ می‌کند و دستور سوم متغیر name را در خروجی چاپ می‌کند.

برنامه‌ای که رشته‌ای را خوانده تمام حروف کوچک آن رشته را به حروف بزرگ تبدیل کرده چاپ می‌کند.

توضیح

در این برنامه، تابعی به نام supper() نوشته شد که حروف کوچک را به حروف بزرگ تبدیل می‌کند. تفاوت کد اسکی حروف کوچک و بزرگ ۳۲است. یعنی اگر از کد اسکی حروف کوچک ۳۲ واحد کم شود به حروف بزرگ تبدیل می‌شوند. لذا برای تبدیل 'a' به 'A' کافی است از 'a' به اندازه ۳۲ واحد کم کنیم.

```
#include <stdio.h>
#include <conio.h>
void upper(char []);
int main()
{
  char s[21];
  clrscr();
  printf("enter a string:");
  gets(s);
  upper(s);
  puts("result is:");
  puts(s);
  getch();
  return 0;
}
 //***********************
void upper(char s[])
{
  int i;
  for(i = 0; s[i]; i++)
      if(s[i] >= 'a' && s[i] <= 'z')
  s[i] -= 32;
}
```

خروجی

```
enter a string : computer science
result is : COMPUTER SCIENCE
```

برنامه‌ای که دو رشته را از ورودی خوانده محتویات آن دو رشته را با یکدیگر عوض می‌کند.

توضیح

رشته‌های s1 و s2 کاراکتر به کاراکتر به کمک متغیر کمکی temp جابجا می‌شوند. اگر طول رشته‌ها متفاوت باشد، کاراکترهای باقیمانده از رشته طولانی‌تر، به رشته دیگر منتقل می‌شود. سپس انتهای هر رشته برابر با '\0' قرار می‌گیرد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
      char s1[81], s2[81], temp;
      int i, j;
      clrscr();
      printf("enter string <s1> : ") ;
      gets(s1) ;
      printf("enter string <s2> : ") ;
      gets(s2) ;
      for(i = 0; s1[i] && s2[i]; i++) {
          temp = s1[i] ;
          s1[i] = s2[i] ;
          s2[i] = temp ;
      } //end of for
      if(s1[i]) {     //s1 has many char          طول s1 بیشتر است
         j = i ;
         while(s1[i])
         s2[i] = s1[i++] ;
         s2[i]='\0' ;
         s1[j]='\0' ;
      }//end of if
      else if (s2[i]) {    //s2 has many char       طول s2 بیشتر است
           j = i ;
           while(s2[i])
           s1[i] = s2[i++] ;
           s2[j]='\0' ;
           s1[i]='\0' ;
      } //end of else if
      printf("new content of s1  is:") ;
      puts(s1) ;
      printf("new content of s2  is:") ;
      puts(s2) ;
      getch();
      return 0;
}
```

خروجی

```
enter string <s1> : computer
enter string <s2> : science
new content of s1 is : science
new content of size : computer
```

انتساب رشته‌ها (کپی کردن رشته در رشته دیگر)

اگر متغیری به نام x و با مقدار ۵ داشته باشید، دستور ; x = y موجب می‌شود تا هر آنچه در x وجود دارد در y قرار گیرد (کپی شود). اگر رشته‌ای به نام s1 با محتویات "computer" داشته باشیم و بخواهیم آن را در رشته دیگری به نام s2 کپی کنیم، با دستور s2 = s1 امکان‌پذیر نیست. به عبارت دیگر با دستور انتساب نمی‌توان رشته‌ای را در رشتهٔ دیگر کپی کرد. لذا در C دستور زیر نادرست است:

s = "all" ; **(نادرست)**

برای کپی کردن رشته‌ای در رشته دیگر و یا انتساب رشته‌ای به رشته دیگر، از تابع ()strcpy استفاده می‌شود. این تابع در فایل string.h قرار دارد و به صورت زیر به کار می‌رود:

strcpy (str1 , str2) ;

با اجرای این دستور، آنچه که در str2 است در str1 کپی می‌شود. دستورات زیر را در نظر بگیرید:

```
strcpy (st, "computer") ;
strcpy (s, st) ;
```

دستور اول رشته "computer" را در st قرار می‌دهد و دستور دوم رشته st را در s کپی می‌کند. لذا، رشته s نیز برابر با "computer" خواهد شد.

در کاربرد تابع ()strcpy چنانچه طول str2 بیش از str1 باشد، تا آنجایی که عمل کپی امکان‌پذیر باشد، عمل کپی در آن رشته صورت می‌گیرد و کاراکترهای اضافی بلافاصله پس از str1 قرار می‌گیرند. در نتیجه، اگر متغیرهایی بعد از str1 قرار داشته باشند، محتویات آنها از بین خواهند رفت. به عنوان مثال، دستورات زیر را در نظر بگیرید. با این دستورات سه کاراکتر از رشته موردنظر در s قرار می‌گیرد و بقیه کاراکترها نیز در حافظه بعد از s واقع می‌شوند.

```
char  s[3] ;
strcpy (s, " computer science") ;
```

مقایسه رشته‌ها

اگر بخواهید دو متغیر x و y را با هم مقایسه کنید، از دستور if (x == y) استفاده می‌نمایید. اما اگر s1 و s2 دو رشته باشند، برای مقایسه آنها نمی‌توانید از دستور if (s1 == s2) استفاده کنید. برای مقایسه رشته‌ها از تابع strcmp() که در فایل string.h قرار دارد به صورت زیر استفاده می‌شود:

```
strcmp (s1,s2)
```

حاصل کار این تابع یک عدد است که مقدار آن بیانگر وضعیت دو رشته نسبت به هم است. اگر عدد برگردانده شده توسط تابع برابر با صفر باشد، این دو رشته با هم مساویند. اگر این عدد منفی باشد s1 < s2 و اگر این عدد مثبت باشد s1 > s2 است.

منظور از مقایسه رشته‌ها، مقایسه کاراکتری آنهاست. اولین کاراکتر از رشته اول با اولین کاراکتر از رشته دوم مقایسه می‌شود. اگر مساوی باشند، کاراکترهای بعدی با هم مقایسه می‌شوند. اگر تمام کاراکترهای دو رشته با هم مساوی باشند، آن دو رشته با هم مساویند. با رسیدن به اولین مورد اختلاف، کاراکتری که بزرگتر است، رشتهٔ حاوی آن کاراکتر، بزرگتر خواهد بود. پس از مطالعه آرایه‌ای از رشته‌ها، مثالی را در این خصوص خواهید دید.

الحاق دو رشته

با استفاده از تابع strcat() می‌توان دو رشته را با هم الحاق کرد. الحاق کردن دو رشته به معنی قراردادن یک رشته در انتهای رشته دیگر است. به عنوان مثال، اگر s1 برابر با "computer" و s2 برابر با "science " باشد، الحاق s2 به انتهای s1 موجب می‌شود تا s1 حاوی رشته "computer science" شود. تابع strcat() در فایل string.h قرار دارد و به صورت زیر به کار می‌رود.

```
strcat (s1, s2) ;
```

با این دستور، s2 در انتهای s1 قرار می‌گیرد. چنانچه طول رشته s1 طوری باشد که گنجایش s2 را نداشته باشد، بقیه رشته s2 در ادامه رشته s1 قرار می‌گیرد و در نتیجه چنانچه متغیرهایی بعد از s1 وجود داشته باشند، محتویات آنها از بین می‌رود.

اشاره‌گرها

آدرس هر متغیر را در حافظه، **اشاره‌گر** گویند. منظور از آدرس حافظهٔ متغیر چیست؟ می‌دانید که بایت، یکی از تقسیمات حافظه است. به عبارت دیگر، حافظه کامپیوتر، مجموعه‌ای از چندین بایت است. هر بایت دارای یک شماره ردیف است. شماره ردیف هر بایت از حافظه را آدرس آن محل از حافظه گویند. ضمناً می‌دانید که متغیرها نامی برای محلهای حافظه‌اند و لذا بایت‌هایی از حافظه را اشغال می‌کنند. آدرس اولین بایتی از حافظه که به متغیر اختصاص می‌یابد، **آدرس** آن متغیر نام دارد.

اشاره‌گرها در C کاربرد فراوانی دارند، به طوری که اغلب قابلیتهای C به نقش اشاره‌گر در این زبان برمی‌گردد. استفاده از اشاره‌گر ها در C، قابلیتهای زیر را فراهم می‌کند:

۱. تخصیص حافظه پویا. در این نوع تخصیص حافظه، برنامه در زمان اجرا از سیستم حافظه می‌گیرد و در صورت عدم نیاز، آن حافظه را به سیستم برمی‌گرداند.

۲. موجب بهبود کارایی بسیاری از توابع می‌شود (توابع را با استفاده از آدرس آنها می‌توان فراخوانی کرد).

۳. کار با رشته‌ها و آرایه‌ها را آسان می‌کند.

۴. ارسال آرگومان‌ها از طریق فراخوانی با ارجاع را امکان‌پذیر می‌سازد.

متغیرهای اشاره‌گر

اشاره‌گر می‌تواند در متغیری ذخیره شود. اما، گرچه اشاره‌گر یک آدرس حافظه است و آدرس حافظه نیز یک عدد است، ولی نمی‌توان آن را در متغیرهایی از نوع int، double و یا غیره ذخیره کرد. بلکه متغیری که می‌خواهد اشاره‌گر را ذخیره کند باید از نوع اشاره‌گر باشد. این متغیرها را **متغیرهای اشاره‌گر** گویند. برای تعریف متغیرهای اشاره‌گر در C، به صورت زیر عمل می‌شود:

نوع * متغیر :

به‌این‌ترتیب، برای تعریف متغیر اشاره‌گری که بخواهد آدرس متغیرهایی را نگهداری کند، باید نوع متغیر اشاره‌گر را همنوع با آن متغیرها در نظر گرفت و کنار متغیر اشاره‌گر، علامت * را قرار داد. به عنوان مثال دستور زیر را در نظر بگیرید:

```
int    * p ;
```

این دستور را می‌توان به صورتهای زیر تفسیر کرد:

۱. p متغیر اشاره‌گری از نوع int است.

۲. p آدرس محلهایی از حافظه را که محتویات آنها مقادیری از نوع صحیح‌اند نگهداری می‌کند.

۳. p می‌تواند به محلهایی اشاره کند که محتویات آنها مقادیری از نوع صحیح می‌باشند.

اکنون دستورات زیر را در نظر بگیرید:

```
int     *p1, *p2, v1, v2 ;
double  *f1, f2 ;
char    *ch ;
```

متغیرهای پویا

چون اشاره‌گر می‌تواند آدرس محلی از حافظه را نگهداری کند، از طریق آدرس آن محل می‌تواند محتویات آن محل را دستکاری کند. بنابراین لزومی ندارد آدرس محلی که در اشاره‌گر قرار می‌گیرد، دارای نام باشد. برای این منظور باید آدرس محلی از حافظه در اشاره‌گر قرار گیرد. امتیاز این روش این‌است‌که پس از اینکه کار با آن محل حافظه به اتمام رسید، می‌توان آن حافظه را آزاد کرد و به سیستم تحویل داد. این روش تخصیص حافظه را **تخصیص حافظه پویا** گویند. می‌توان اینطور تصور کرد که محلهایی از حافظه، که بدین طریق، آدرس آنها در اشاره‌گر قرار می‌گیرند، **متغیرهای بی‌نام** هستند. این متغیرها را **متغیرهای پویا** نیز می‌گویند، زیرا در زمان اجرای برنامه توسط برنامه‌نویس ایجاد شده و از بین می‌روند.

تخصیص حافظه پویا

برای اخذ حافظه از سیستم، از تابع malloc() استفاده می‌شود. این تابع، حافظه‌ای را از سیستم گرفته، آدرس آن را در یک اشاره‌گر قرار می‌دهد. تابع malloc() که در فایل stdlib.h قرار دارد، به صورت زیر به کار می‌رود:

اشاره‌گر = (نوع) malloc (size) ;

در این روش کاربرد، **(نوع)**، نوع اشاره‌گری است که آدرس حافظه باید در آن قرار گیرد و size میزان حافظه به بایت است و مشخص می‌کند که این تابع چند بایت از حافظه را باید از سیستم اخذ کند. اگر تابع malloc() به هر دلیلی نتواند حافظه‌ای را از سیستم بگیرد و در اختیار برنامه‌نویس قرار دهد، مقدار تهی (NULL) را در اشاره‌گر قرار می‌دهد. اشاره‌گر تهی، اشاره‌گری است که به جایی اشاره نمی‌کند. دستورات زیر را در نظر بگیرید:

```
int  * p ;
p = (int *) malloc (sizeof(int)) ;
```

دستور اول، p را اشاره‌گر صحیح تعریف می‌کند و دستور دوم حافظه‌ای به اندازه sizeof(int) بایت را از سیستم گرفته آدرس آن را در p قرار می‌دهد. عبارت (* int) که قبل از malloc() آمده است، تبدیل نوع (type casting) است.

برگرداندن حافظه به سیستم

حافظه‌ای که به صورت پویا تخصیص یافت، پس از استفاده باید به سیستم برگردانده شود. حافظه‌ای که به وسیله malloc() اختصاص یافت با free() به سیستم برمی‌گردد. تابع free() در فایل stdlib.h قرار دارد و به صورت زیر به کار می‌رود:

free (اشاره‌گر) ;

به عنوان مثال، حافظه‌ای را که قبلاً با تابع malloc() تخصیص یافت و آدرسش در p قرار گرفت با دستور زیر به سیستم برمی‌گردد:

```
free (p) ;
```

برنامه‌ای که از طریق تخصیص حافظه پویا، دو مقدار را از ورودی خوانده مجموع مربعات آنها را محاسبه می‌کند و به خروجی می‌برد.

توضیح

پس از اجرای تابع malloc() برای تخصیص حافظه پویا، باید تست کرد که آیا حافظه اختصاص یافت یا خیر. اگر حافظه اختصاص نیافته باشد، مقدار NULL در اشاره‌گر قرار می‌گیرد. به همین منظور، در این برنامه، چنانچه NULL برگردانده شود، برنامه با تابع exit(1) خاتمه می‌یابد. وظیفه این تابع خاتمه دادن به برنامه است و در فایل stdlib.h قرار دارد.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main()
{
  int *x, *y, s;
  clrscr();
  x = (int *) malloc(sizeof(int));
  if(!x){
     printf("\n allocation failure.");
     exit(1);
  }
  y = (int *) malloc(sizeof(int));
  if(!y){
     printf("\n allocation failure.");
     exit(1);
  }
  printf("\n enter two integers:");
  scanf("%d%d", x, y);
  s = *x * *x + *y * *y;
  printf("\n sum of square is:%d", s);
  free(x);
  free(y);
  getch();
  return 0;
}
```

خروجی

```
enter two integers : 5  10
sum of square is : 125
```

اشاره گرها و توابع

در فصل ۴ دو روش برای ارسال آرگومان‌ها به توابع مطرح شد ۱. از طریق فراخوانی با مقدار ۲. از طریق فراخوانی با ارجاع. روش اول در آن فصل مورد بحث قرار گرفت و در مثال‌هایی به کار برده شد. در ارسال آرگومان‌ها با روش فراخوانی با ارجاع، به جای مقدار آرگومان‌ها، آدرس آنها در پارامتر قرار می‌گیرد. بدین‌ترتیب، در تابعی که فراخوانی می‌شود، به آدرس متغیرهای موجود در برنامه فراخوان دسترسی داریم. از طریق آدرس این متغیرها می‌توانیم محتویات آنها را دستکاری کنیم.

مثال ۲-۶

برنامه‌ای که دو عدد را از ورودی خوانده، با استفاده از تابعی محتویات آنها را با هم عوض می‌کند.

توضیح

چون در این برنامه، تابع باید دو متغیر از تابع اصلی را دستکاری کند، هر دو مقدار از طریق فراخوانی باارجاع ارسال می‌شوند (شکل ۵-۶ پس از اجرای برنامه). تابع ()swap برای جابجایی محتویات دو متغیر از طریق آدرس آنها، از متغیر کمکی temp استفاده می‌کند. ابتدا با دستور temp=*a آنچه که در x است در temp قرار می‌گیرد. سپس با دستور *a=*b آنچه که در y است در x قرار می‌گیرد و در آخر، با دستور *b=temp آنچه که در temp است در y قرار می‌گیرد(زیرا a به x و b به y اشاره می‌کند).

```
#include <stdio.h>
#include <conio.h>
void swap(int *, int *);
int main()
{
    int x = 10, y = 20 ;
    clrscr();
    printf("\n first value of x,y are:%d, %d", x, y) ;
    swap(&x, &y);
    printf("\n final value of x, y are:%d, %d", x, y) ;
    getch();
    return 0;
}
void swap(int *a, int *b)
{
    int temp ;
    temp=*a ;
    *a=*b ;
    *b=temp ;
}
```

خروجی

```
first value of x, y are : 10 , 20
final value of x, y are : 20 , 10
```

برنامه‌ای که با استفاده از اشاره‌گر تابع، تابعی را فراخوانی کرده، دو رشته را از ورودی خوانده، تشخیص می‌دهد که آیا این دو رشته با هم مساویند یا خیر (به تحلیل مثال توجه کنید).

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void check(char *a, char *b, int (*cmp)(const char *, const char *));
//int cmp(char *, char *);
int main() {
    char s1[80], s2[80] ;
    int (*p)(const char *, const char *);
    clrscr();
    p = strcmp;
    printf(" enter first string :") ;
    gets(s1) ;
    printf(" enter second string :") ;
    gets(s2) ;
    check(s1, s2, p);
    getch();
    return 0;
}
//**********************
void check(char *a, char *b, int (*cmp)(const char *, const char *)) {
   if(!(cmp)(a, b))
       printf(" the strings are equal.");
   else
       printf(" strings are not equal.");
}
```

آرگومان‌های تابع ()check عبارت‌اند از رشته‌های s1 و s2 و اشاره‌گر p و پارامترهای آن عبارت‌اند از دو اشاره‌گر رشته‌ای و یک اشاره‌گر از نوع تابع. پرانتزهایی که در اطراف cmp* آمده‌اند، ضروری‌اند، زیرا در غیر اینصورت، این اشاره‌گر به عنوان اشاره‌گر تابعی محسوب نخواهد شد. در تابع ()check دستور (*cmp)(a, b) موجب اجرای تابع ()strcmp می‌شود، زیرا آدرس آن تابع در اشاره‌گر cmp قرار دارد. ضمناً این دستور، روش کلی فراخوانی توابع از طریق آدرس آنها را نشان می‌دهد. توجه داشته باشید که تابع ()strcmp، از طریق فراخوانی ()check به صورت زیر نیز اجرا می‌شود:

```
check (s1, s2, strcmp) ;
```

ممکن‌است فراخوانی توابع از طریق آدرس، کمی دشوار به نظر برسد، اما این روش، در نوشتن کامپایلرها و برنامه تجزیه‌کننده کاربرد فراوانی دارد.

اشاره گرها و آرایه‌ها

در زبان C، بین آرایه‌ها و اشاره گرها ارتباط نزدیکی وجود دارد. اشاره گرها حاوی آدرس‌اند و اسم آرایه نیز یک آدرس است. اسم آرایه اولین عنصر آرایه را مشخص می‌کند؛ به عبارت دیگر، اسم آرایه، آدرس اولین محلی را که عناصر آرایه از آنجا به بعد در حافظه ذخیره می‌شوند، نگهداری می‌کند. پس **اسم آرایه**، یک اشاره گر است. به عنوان مثال، دستورات زیر را در نظر بگیرید:

```
int  table [5] ;
int  * ptr ;
```

اگر اولین عنصر آرایه در محل ۱۰۰۰ حافظه وجود داشته باشد، table به محل ۱۰۰۰ حافظه اشاره خواهد کرد (شکل ۷-۶). به موارد زیر توجه کنید:

ptr = table ;			**ptr و table به ابتدای آرایه اشاره می‌کنند**
***(ptr + 1)**	**معادل است با**	**table [1]**	**عنصر دوم آرایه**
ptr [2]	**معادل است با**	***(table + 2)**	**عنصر سوم آرایه**
***ptr**	**معادل است با**	***table**	**عنصر اول آرایه**

برنامه‌ای که ۵ عدد را از ورودی خوانده، در آرایه‌ای قرار می‌دهد و سپس عناصر آرایه را از آخرین عنصر به اولین عنصر به خروجی می‌برد.

```
#include <stdio.h>
#include <conio.h>
int main() {
    int arr[5] ,i ;
    clrscr();
    printf("\n enter five value :") ;
    for(i=0 ; i<5 ; i++)
      scanf("%d",&arr[i]) ;
    printf("\n the reverse output is :");
    for(i=4 ; i>=0 ; i -- )
        printf("%4d",*(arr+i)) ;
    getch();
    return 0;
}
```

آرایه پویا

در ابتدای این فصل، با مفهوم متغیرهای پویا آشنا شدید. آرایه‌ها نیز می‌توانند پویا باشند. یعنی حافظه مورد نیاز آرایه را نیز می‌توان در حین اجرای برنامه از سیستم اخذ کرد و پس از استفاده، به سیستم برگرداند. برای تخصیص حافظه به آرایه نیز از تابع ()malloc استفاده می‌شود. دستورات زیر را در نظر بگیرید:

```
int  *x, n ;
scanf ("%d",&n) ;
x = (int *) malloc (sizeof (int) * n) ;
```

دستور سوم حافظه‌ای را که اندازه آن با sizeof(int) * n مشخص می‌شود اختصاص داده آدرس آن را در x قرار می‌دهد. لذا x یک آرایه صحیح با n عنصر است. اکنون با مثال ساده‌ای، کاربرد آرایه‌های پویا را تشریح می‌کنیم.

مثال ۶-۶

برنامه‌ای که تعداد n عدد را از ورودی خوانده در آرایه‌ای پویا قرار می‌دهد و سپس آرایه را به تابع ارسال می‌کند. تابع، میانه اعداد موجود در آرایه را پیدا کرده، به برنامه برمی‌گرداند.

توضیح

برای محاسبه میانه اعداد، باید آنها را مرتب کرد. اگر تعداد اعداد زوج باشد، میانگین دو عدد وسط برابر با میانه آن اعداد است و اگر تعداد اعداد فرد باشد، میانه عددی است که نیمی از اعداد از آن بزرگتر و نیمی دیگر از آن کوچکتر باشند. در این برنامه از چهار تابع استفاده شده است که شرح وظایف آنها در ادامه آمده است. در این برنامه، برای

ذخیره آرایه از تخصیص حافظه پویا استفاده شد. به‌این‌ترتیب که، تعداد عناصر آرایه (n) از ورودی خوانده شد و سپس با استفاده از تابع malloc() حافظه به آرایه اختصاص یافت. پس از استفاده از آرایه، حافظه آن به سیستم پس داده شده است. دستور زیر، حافظه‌ای را برای ذخیره n عنصر صحیح ایجاد می‌کند.

```
int   p ;
p = (int *) malloc (n * sizeof (int)) ;
```

شرح وظایف برنامه‌ها

برنامه main(): تخصیص حافظه به آرایه و فراخوانی زیربرنامه‌ها. در این برنامه، n تعداد عناصر آرایه، i شمارنده حلقه تکرار و mead میانه اعداد است و p به آرایه اشاره می‌کند.

تابع pinput(): خواندن عناصر آرایه از ورودی، آرگومان اول آن اشاره‌گر به آرایه و آرگومان دوم، طول آرایه است.

تابع bubble(): آرایه را مرتب می‌کند. آرگومان اول اشاره‌گر به آرایه و آرگومان دوم، طول آرایه است.

تابع median(): این تابع، اشاره‌گر به آرایه، طول آرایه و متغیر مربوط به میانه عدد را گرفته، میانه را محاسبه می‌کند و در آن متغیر قرار می‌دهد (فراخوانی باارجاع).

تابع pout(): عناصر آرایه را پس از مرتب شدن به خروجی می‌برد.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void pinput(int *, int);
void bubble(int *, int);
void median(int *, int, float *);
void pout(int *, int);
int main() {
  int n, i;
  int  *p;
  float mead;
  clrscr();
  printf(" enter number of items(n):");
  scanf("%d", &n);
  p = (int *)malloc(n * sizeof(int));
  if(!p){
       printf(" aloocation failure!");
       getch();
       exit(1);
  }
  pinput(p, n);
  bubble(p, n);
  printf("\n sorted data are:");
  pout(p, n);
```

```
  median(p, n, &mead);
  printf("\n median = %5.2f", mead);
  getch();
  return 0;
}
 //*****************************
void pinput(int  *p, int n)
{
  int i;
  for(i = 0; i < n; i ++){
     printf("enter number %d:", i + 1);
       scanf("%d", p + i);
  }
}
 //*****************************
void bubble(int  *temp, int len)
{
   int i, j, item;
   for(i = len - 1 ; i > 0; i --)
       for(j = 0; j < i ; j++)
           if(*(temp + j) > *(temp + j + 1)) {
              item = *(temp + j);
              *(temp + j) = *(temp + j + 1);
              *(temp + j + 1) = item ;
           }//end of if
}
 //**************************
void median(int *x, int n, float *mead)
{
  if(n % 2 == 0)
       *mead =(float) (*(x + ((n - 1) / 2)) + *(x + (n / 2))) / 2;
  else
       *mead = *(x + (n - 1) / 2);
}
 //*****************
void pout(int *p, int n)
{
  int i;
  for(i = 0; i < n; i ++){
     printf("%3d",*(p+i));
  }
  printf("\n");
}
```

اشاره‌گرها و رشته‌ها

در بخش قبلی، ارتباط بین آرایه‌ها و اشاره‌گرها مطرح شد و دیدید که نام آرایه، یک اشاره‌گر است. چون رشته‌ها در C به صورت آرایه‌ای از کاراکترها تعریف می‌شوند، ارتباط بین رشته‌ها و اشاره‌گرها همانند ارتباط بین آرایه‌ها و اشاره‌گرهاست.

مثال ۷-۶

برنامه‌ای که یک رشته و کاراکتری را از ورودی خوانده، تعداد دفعات تکرار کاراکتر را در رشته پیدا می‌کند و در خروجی چاپ می‌نماید.

توضیح

در این برنامه از تابع char_count() برای شمارش تعداد تکرار کاراکتر استفاده شده است. رشته و کاراکتر خوانده شده، به تابع تحویل داده می‌شوند. متغیر string حاوی رشته، ch حاوی کاراکتر و count تعداد تکرار ch است.

```
#include <stdio.h>
#include <conio.h>
int char_count(char *, char);
int main()
{
    char string[40] , ch ;
    int count ;
    clrscr();
    printf(" enter string for search:");
    gets(string) ;
    printf(" enter a character:") ;
    ch = getche() ;
    count = char_count(string, ch) ;
    printf("\n number of occurs of char ") ;
    printf(" <%c> is:%d", ch, count);
    getch();
    return 0;
}
```

```
//*******************
int char_count(char *s, char letter)
{
    int count=0 ;
    while(*s)
      if(*s++ == letter)
         count ++ ;
     return(count) ;
}
```

```
enter string for search : a book about computer.
enter a character : a
number of occurs of char <a> is : 2
```

برنامه‌ای که دو رشته را از ورودی خوانده، سپس آن دو رشته را طوری با هم مقایسه می‌کند که اولین کاراکتر مورد اختلاف دو رشته را می‌یابد.

توضیح

تابع main() رشته‌های st1 و st2 را از ورودی خوانده تابع first_dif() را فراخوانی می‌کند. تابع first_dif() رشته‌ها را کاراکتر به کاراکتر مقایسه می‌کند. اگر دو کاراکتر نامساوی باشند، چک می‌کند که آیا متغیر ignore_case برابر با یک است یا خیر، اگر برابر با یک باشد، کاراکترهای دو رشته را به حروف بزرگ تبدیل می‌کند و سپس آنها را با هم مقایسه می‌کند. اگر پس از تبدیل نیز با هم مساوی نبودند، اولین مورد اختلاف پیدا شده، مقایسه خاتمه می‌یابد. هرگاه دو کاراکترِ مورد مقایسه با هم برابر باشند، کاراکترهای بعدی مقایسه می‌شوند.

متغیرهای برنامه

تابع	متغیر	هدف
main()	st1,st2	رشته‌هایی که با هم مقایسه می‌شوند
	compcase	نوع مقایسه را مشخص می‌کند. اگر برابر با یک باشد، تفاوتی بین حروف کوچک و بزرگ نیست و اگر صفر باشد، بین حروف کوچک و بزرگ تفاوت است.
first_dif()	s1,s2	رشته‌هایی که باید با هم مقایسه شوند (پارامتر تابع)
	ignore_case	مانند compcase در برنامه اصلی عمل می‌کند
	i	شمارنده حلقه
	a,b	کاراکترهایی از رشته‌های s1 و s2 که باید با هم مقایسه شوند.

```
#include <stdio.h>
#include <conio.h>
first_dif(char *, char *, int);
int main()
{
      char st1[40], st2[40] ;
      int compcase = 1 ;
      clrscr();
      printf(" enter first string:") ;
      gets(st1) ;
      printf(" enter second string:");
      gets(st2) ;
      first_dif(st1, st2, compcase) ;
      getch();
      return 0;
}
//*********************
first_dif(char *s1, char *s2, int ignore_case)
    /* if ignore_case is 1 ,ignore case
          of letter */
{
    int i ;
    char a , b ;
    for(i = 0; *s1 && *s2; s1++, s2++, i++)
      if(*s1 != *s2) {
         if(ignore_case) {
            a = (*s1 >= 'a' && *s1 <= 'z') ? *s1 -= 32 : *s1;
            b = (*s2 >= 'a' && *s2 <= 'z') ? *s2 -= 32 : *s2;
            if(a != b)
               break ;
         }//end of if
      else
         break;
      }//end of if
      if(*s1 || *s2) {
         printf("\n strings are not equal,");
         printf(" the first difference occurs in :%d", i + 1);
      }
      else
         printf("\n strings are equal .") ;
}
```

تمرین:

اشکالات هر یک از بخشهای زیر را بیابید :

a)
```
int   *num ;
printf("%d", *num) ;
```

b)
```
short    *numptr, result ;
void     *genericptr = numptr ;
result = *genericptr +7 ;
```

برنامه‌ای بنویسید که نام و شماره تلفن تعدادی از مشتریان مخابرات را از ورودی خوانده، در آرایه‌هایی ذخیره نماید. شماره تلفنها در آرایه عددی و نام مشتریان در آرایه‌ای از اشاره گرهای رشته‌ای ذخیره شوند. سپس نامی از ورودی خوانده شده، شماره تلفن وی را مشخص کرده، در خروجی چاپ کند. برنامه باید برای ادامه کار، از کاربر سئوال کند. اگر کاربر جواب منفی داد، برنامه خاتمه پیدا می‌کند. توابعی برای خواندن اطلاعات، جستجو و چاپ نتایج جستجو بنویسید.

برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده، به تابعی ارسال کند و تابع آن را به‌طور معکوس به خروجی ببرد.

برنامه‌ای بنویسید که سه مقدار عددی را به عنوان آرگومان پذیرفته، به تابعی ارسال کند و تابع بزرگترین مقدار آنها را پیدا کند. برنامه باید تعداد آرگومان‌ها را کنترل کند.

برنامه‌ای بنویسید که رشته عددی را که حاوی نقطه اعشار است از ورودی خوانده، آن را به عدد اعشاری تبدیل کند. به عنوان مثال، رشتهٔ "123.42" را به عدد 123.42 تبدیل نماید. تابعی برای خواندن رشته، تابعی برای تبدیل و تابعی برای نوشتن عدد در خروجی بنویسید. پارامترها از طریق فراخوانی باارجاع به توابع ارسال شوند.

برنامه‌ای بنویسید که رشته‌هایی را از ورودی خوانده، فقط آن رشته‌هایی را که با حرف 'b' شروع می‌شوند در خروجی چاپ کند و به جای آخرین رشته، فقط کلید Enter را فشار دهید.

برنامه‌ای بنویسید که دو رشته را از ورودی خوانده، یکی را در دیگری کپی کند.

تابعی بنویسید که یک رشته و یک مقدار عددی را به عنوان آرگومان پذیرفته، تعدادی از کاراکترهای این رشته را که با این عدد مشخص می‌شود در رشته دیگری قرار داده، برگرداند. سپس برنامه‌ای بنویسید که از آن استفاده کند.