فصل پنجم

« توابع»

5-1 مقدمه

برنامه‌های واقعی و تجاری بسیار بزرگ‌تر از برنامه‌هایی هستند که تاکنون بررسی کردیم. برای این که برنامه‌های بزرگ قابل مدیریت باشند، برنامه‌نویسان این برنامه‌ها را به زیربرنامه‌هایی بخش‌بندی می‌کنند. این زیربرنامه‌ها «تابع» نامیده می‌شوند. توابع را می‌توان به طور جداگانه کامپایل و آزمایش نمود و در برنامه‌های مختلف دوباره از آن‌ها استفاده کرد. این بخش‌بندی در موفقیت یک نرم‌افزار شی‌گرا بسیار موثر است.

5-2 توابع کتابخانه‌ای C++ استاندارد

«کتابخانهٔ C++ استاندارد» مجموعه‌ای است که شامل توابع از پیش تعریف شده و سایر عناصر برنامه است. این توابع و عناصر از طریق «سرفایل‌ها» قابل دستیابی‌اند. قبلا برخی از آن‌ها را استفاده کرده‌ایم: ثابت INT_MAX که در <climits> تعریف شده (مثال 1-2)، تابع sqrt() که در <cmath> تعریف شده (مثال 15-2)، تابع rand() که در <cstdlib> تعریف شده (مثال 22-4) و تابع time() که در

<ctime> تعریف شده (مثال 24-4). اولین مثال این بخش، استفاده از یک تابع ریاضی را نشان می‌دهد.

× مثال 5-1 تابع جذر ()sqrt

ریشهٔ دوم یک عدد مثبت، جذر آن عدد است. ریشهٔ دوم 9، عدد 3 است. می‌توانیم تابع جذر را به شکل یک جعبهٔ سیاه تصور کنیم که وقتی عدد 9 درون آن قرار گیرد، عدد 3 از آن خارج می‌شود و وقتی عدد 2 در آن قرار گیرد، عدد 1/41421 از آن خارج می‌شود. تابع مانند یک برنامهٔ کامل، دارای روند ورودی – پردازش – خروجی است هرچند که پردازش، مرحله‌ای پنهان است. یعنی نمی‌دانیم که تابع روی عدد 2 چه اعمالی انجام می‌دهد که 1/41421 حاصل می‌شود. تنها چیزی که لازم است بدانیم این است که عدد 1/41421 جذر است و مجذور آن، عدد ورودی 2 بوده است.

برنامهٔ سادهٔ زیر، تابع از پیش تعریف شدهٔ جذر را به کار می‌گیرد:

```
#include <cmath>      // defines the sqrt() function
#include <iostream>   // defines the cout object
using namespace std;
int main()
{ //tests the sqrt() function:
   for (int x=0; x < 6; x++)
      cout << "\t" << x << "\t" << sqrt(x) << endl;
}
```

```
    0       0
    1       1
    2       1.41421
    3       1.73205
    4       2
    5       2.23607
```

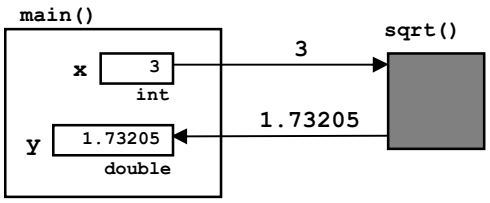
این برنامه، ریشهٔ دوم اعداد صفر تا پنج را چاپ می‌کند. هر وقت اجرای برنامه به عبارت sqrt(x) می‌رسد، تابع sqrt() اجرا می‌گردد. گرچه کد اصلی تابع مذکور درون کتابخانهٔ C++ پنهان شده اما می‌توانیم مطمئن باشیم که به جای عبارت sqrt(x) مقدار جذر x قرار می‌گیرد. به دستور #include <cmath> در اولین

خط برنامه توجه کنید. کامپایلر به این خط نیاز دارد تا بتواند تعریف تابع sqrt() را پیدا کند. این خط به کامپایلر می‌گوید که تعریف تابع جذر در سرفایل <cmath> وجود دارد.

برای اجرای یک تابع مانند تابع sqrt() کافی است نام آن تابع به صورت یک متغیر در دستورالعمل مورد نظر استفاده شود، مانند زیر:

```
y=sqrt(x).
```

این کار «فراخوانی تابع[1]» یا «احضار تابع» گفته می‌شود. بنابراین وقتی کد sqrt(x) اجرا شود، تابع sqrt() فراخوانی می‌گردد. عبارت x درون پرانتز «آرگومان[2]» یا «پارامتر واقعی» فراخوانی نامیده می‌شود. در چنین حالتی می‌گوییم که x توسط «مقدار»[3] به تابع فرستاده می‌شود. لذا وقتی x=3 است، با اجرای کد sqrt(x) تابع sqrt() فراخوانی شده و مقدار 3 به آن فرستاده می‌شود. تابع مذکور نیز حاصل 1.73205 را به عنوان پاسخ برمی‌گرداند. این فرایند در نمودار مقابل نشان داده شده. متغیرهای x و y در تابع main() تعریف شده‌اند. مقدار x که برابر با 3 است به تابع sqrt() فرستاده می‌شود و این تابع مقدار 1.73205 را به تابع main() برمی‌گرداند. جعبه‌ای که تابع sqrt() را نشان می‌دهد به رنگ تیره است، به این معنا که فرایند داخلی و نحوهٔ کار آن قابل رویت نیست.



× مثال 2-5 آزمایش یک رابطهٔ مثلثاتی

این برنامه هم از سرفایل <cmath> استفاده می‌کند. هدف این است که صحت رابطهٔ Sin2x=2SinxCosx به شکل تجربی بررسی شود.

```
int main()
{  // tests the identity sin 2x = 2 sin x cos x:
   for (float x=0; x < 2; x += 0.2)
      cout << x << "\t\t" << sin(2*x) << "\t"
```

1 – Function call 2 – Argument 3 – Pass by value

```
            << 2*sin(x)*cos(x) << endl;
}
```

```
0               0       0
0.2             0.389418        0.389418
0.4             0.717356        0.717356
0.6             0.932039        0.932039
0.8             0.999574        0.999574
1               0.909297        0.909297
1.2             0.675463        0.675463
1.4             0.334988        0.334988
1.6             -0.0583744      -0.0583744
1.8             -0.442521       -0.442521
```

برنامهٔ بالا مقدار x را در ستون اول، مقدار Sin2x را در ستون دوم و مقدار 2SinxCosx را در ستون سوم چاپ می‌کند. خروجی نشان می‌دهد که برای هر مقدار آزمایشی x، مقدار Sin2x با مقدار 2SinxCosx برابر است. البته این نتایج به طور کلی اثبات نمی‌کند که رابطهٔ مذکور صحیح است اما به طور تجربی نشان می‌دهد که این رابطه درست می‌باشد. توجه کنید که x از نوع float تعریف شده است. این امر سبب می‌شود که کد x += 0.2 به درستی کار کند و خطای گردکردن رخ ندهد.

حاصل تابع را می‌توانیم مانند یک متغیر معمولی در هر عبارتی به کار ببریم. یعنی می‌توانیم بنویسیم:

```
y = sqrt(2);
cout << 2*sin(x)*cos(x);
```

همچنین می‌توانیم توابع را به شکل تودرتو فراخوانی کنیم:

```
y = sqrt(1 + 2*sqrt(3 + 4*sqrt(5)))
```

بیشتر توابع معروف ریاضی که در ماشین‌حساب‌ها هم وجود دارد در سرفایل <cmath> تعریف شده است. بعضی از این توابع در جدول زیر نشان داده شده:

بعضی از توابع تعریف شده در سرفایل <cmath>

تابع	شرح	مثال
acos(x)	کسینوس معکوس x (به رادیان)	acos(0.2) مقدار 1.36944 را برمی‌گرداند
asin(x)	سینوس معکوس x (به رادیان)	asin(0.2) مقدار 0.201358 را برمی‌گرداند
atan(x)	تانژانت معکوس x (به رادیان)	atan(0.2) مقدار 0.197396 را برمی‌گرداند

ceil(x)	مقدار سقف x (گرد شده)	ceil(3.141593) مقدار 4.0 را برمی‌گرداند
cos(x)	کسینوس x (به رادیان)	cos(2) مقدار 0.416147- را برمی‌گرداند
exp(x)	تابع نمایی x (در پایه e)	exp(2) مقدار 7.38906 را برمی‌گرداند
fabs(x)	قدر مطلق x	fabs(-2) مقدار 2.0 را برمی‌گرداند
floor(x)	مقدار کف x (گرد شده)	floor(3.141593) مقدار 3.0 را برمی‌گرداند
log(x)	لگاریتم طبیعی x (در پایه e)	log(2) مقدار 0.693147 را برمی‌گرداند
log10(x)	لگاریتم عمومی x (در پایه 10)	log10(2) مقدار 0.30103 را برمی‌گرداند
pow(x,p)	x به توان p	pow(2,3) مقدار 8.0 را برمی‌گرداند
sin(x)	سینوس x (به رادیان)	sin(2) مقدار 0.909297 را برمی‌گرداند
sqrt(x)	جذر x	sqrt(2) مقدار 1.41421 را برمی‌گرداند
tan(x)	تانژانت x (به رادیان)	tan(2) مقدار 2.18504- را برمی‌گرداند

توجه داشته باشید که هر تابع ریاضی یک مقدار از نوع double را برمی‌گرداند. اگر یک نوع صحیح به تابع فرستاده شود، قبل از این که تابع آن را پردازش کند، مقدارش را به نوع double ارتقا می‌دهد.

بعضی از سرفایل‌های کتابخانهٔ C++ استاندارد که کاربرد بیشتری دارند در جدول زیر آمده است:

بعضی از سرفایل‌های کتابخانهٔ C++ استاندارد

سرفایل	شرح
<assert>	تابع <assert> را تعریف می‌کند
<ctype>	توابعی را برای بررسی کاراکترها تعریف می‌کند
<cfloat>	ثابت‌های مربوط به اعداد ممیز شناور را تعریف می‌کند
<climits>	محدودهٔ اعداد صحیح را روی سیستم موجود تعریف می‌کند
<cmath>	توابع ریاضی را تعریف می‌کند
<cstdio>	توابعی را برای ورودی و خروجی استاندارد تعریف می‌کند
<cstdlib>	توابع کاربردی را تعریف می‌کند
<cstring>	توابعی را برای پردازش رشته‌ها تعریف می‌کند
<ctime>	توابع تاریخ و ساعت را تعریف می‌کند

این سرفایل‌ها از کتابخانهٔ C استاندارد گرفته شده‌اند. استفاده از آن‌ها شبیه استفاده از سرفایل‌های C++ استاندارد (مانند <iostream>) است. برای مثال اگر

بخواهیم تابع اعداد تصادفی rand() را از سرفایل <cstdlib> به کار ببریم، باید دستور پیش‌پردازندهٔ زیر را به ابتدای فایل برنامهٔ اصلی اضافه کنیم:

```
#include <cstdlib>
```

5-3 توابع ساخت کاربر

گرچه توابع بسیار متنوعی در کتابخانهٔ C++ استاندارد وجود دارد ولی این توابع برای بیشتر وظایف برنامه‌نویسی کافی نیستند. علاوه بر این برنامه‌نویسان دوست دارند خودشان بتوانند توابعی را بسازند و استفاده نمایند.

× مثال 5-3 تابع cube()

یک مثال ساده از توابع ساخت کاربر:

```
int cube(int x)
{  // returns cube of x:
   return x*x*x;
}
```

این تابع، مکعب یک عدد صحیح ارسالی به آن را برمی‌گرداند. بنابراین فراخوانی cube(2) مقدار 8 را برمی‌گرداند.

یک تابع ساخت کاربر دو قسمت دارد: عنوان و بدنه. عنوان یک تابع به صورت زیر است:

نوع بازگشتی **نام** (*فهرست پارامترها*)

نوع بازگشتی تابع cube() که در بالا تعریف شد، int است. نام آن cube می‌باشد و یک پارامتر از نوع int به نام x دارد. یعنی تابع cube() یک مقدار از نوع int می‌گیرد و پاسخی از نوع int تحویل می‌دهد. پس عنوان تابع فوق عبارت است از:

```
int cube(int x)
```

بدنهٔ تابع، یک بلوک کد است که در ادامهٔ عنوان آن می‌آید. بدنه شامل دستوراتی است که باید انجام شود تا نتیجهٔ مورد نظر به دست آید. بدنه شامل دستور return

است که پاسخ نهایی را به مکان فراخوانی تابع برمی‌گرداند. بدنهٔ تابع cube عبارت است از:

```
{  // returns cube of x:
   return x*x*x;
}
```

این تقریبا ساده‌ترین بدنه‌ای است که یک تابع می‌تواند داشته باشد. توابع مفیدتر معمولا بدنهٔ بزرگ‌تری دارند اما عنوان تابع اغلب روی یک سطر جا می‌شود.

دستور return دو وظیفهٔ عمده دارد. اول این که اجرای تابع را خاتمه می‌دهد و دوم این که مقدار نهایی را به برنامهٔ فراخوان باز می‌گرداند. دستور return به شکل زیر استفاده می‌شود:

return *expression*;

به جای *expression* هر عبارتی قرار می‌گیرد که بتوان مقدار آن را به یک متغیر تخصیص داد. نوع آن عبارت باید با نوع بازگشتی تابع یکی باشد.

عبارت int main() که در همهٔ برنامه‌ها استفاده کرده‌ایم یک تابع به نام «تابع اصلی» را تعریف می‌کند. نوع بازگشتی این تابع از نوع int است. نام آن main است و فهرست پارامترهای آن خالی است؛ یعنی هیچ پارامتری ندارد.

4-5 برنامهٔ آزمون

وقتی یک تابع مورد نیاز را ایجاد کردید، فورا باید آن تابع را با یک برنامهٔ ساده امتحان کنید. چنین برنامه‌ای ***برنامهٔ آزمون***[1] نامیده می‌شود. تنها هدف این برنامه، امتحان کردن تابع و بررسی صحت کار آن است. برنامهٔ آزمون یک برنامهٔ موقتی است که باید «سریع و کثیف» باشد؛ یعنی لازم نیست در آن تمام ظرافت‌های برنامه‌نویسی – مثل پیغام‌های خروجی، برچسب‌ها و راهنماهای خوانا – را لحاظ کنید. وقتی با استفاده از برنامهٔ آزمون، تابع را آزمایش کردید دیگر به آن احتیاجی نیست و می‌توانید برنامهٔ آزمون را دور بریزید.

1 – Testing program

× مثال 4-5 یک برنامهٔ آزمون برای تابع cube()

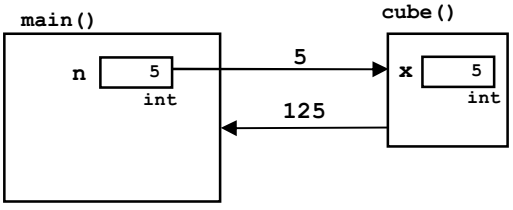
کد زیر شامل تابع cube() و برنامهٔ آزمون آن است:

```
int cube(int x)
{  // returns cube of x:
   return x*x*x;
}


int main()
{  // tests the cube() function:
   int n=1;
   while (n != 0)
   {  cin >> n;
      cout << "\tcube(" << n << ") = " << cube(n) << endl;
   }
}
```

```
5
        cube(5) = 125
-6
        cube(-6) = -216
0
        cube(0) = 0
```

برنامهٔ بالا اعداد صحیح را از ورودی می‌گیرد و مکعب آن‌ها را چاپ می‌کند تا این که کاربر مقدار 0 را وارد کند. هر عدد صحیحی که خوانده می‌شود، با استفاده از کد cube(n) به تابع cube() فرستاده می‌شود. مقدار بازگشتی از تابع، جایگزین عبارت cube(n) گشته و با استفاده از cout در خروجی چاپ می‌شود.



می‌توان رابطهٔ بین تابع main() و تابع cube() را شبیه این شکل تصور نمود:

تابع main() مقدار 5 را به تابع cube() می‌فرستد و تابع

cube() مقدار 125 را به تابع main() بازمی‌گرداند. آرگومان n به وسیلهٔ مقدار به پارامتر صوری x فرستاده می‌شود. به بیان ساده‌تر وقتی تابع فراخوانی می‌شود، x مقدار n را می‌گیرد.

دقت کنید که تابع cube() در بالای تابع main() تعریف شده زیرا قبل از این که تابع cube() در تابع main() به کار رود، کامپایلر C++ باید در بارهٔ آن اطلاع حاصل کند.

مثال بعدی یک تابع ساخت کاربر به نام max() را نشان می‌دهد که این تابع از دو عدد ارسال شده به آن، عدد بزرگ‌تر را برمی‌گرداند. این تابع دو پارامتر دارد.

× مثال 5-5 یک برنامهٔ آزمون برای تابع max()

تابع زیر دو پارامتر دارد. این تابع از دو مقدار فرستاده شده به آن، مقدار بزرگ‌تر را برمی‌گرداند:

```
int max(int x, int y)
{  // returns larger of the two given integers:
   int z;
   z = (x > y) ? x : y ;
   return z;
}

int main()
{   // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
   }
   while (m != 0);
}
```

```
3 9
      max(3,9) = 9
2 -2
```

```
        max(2,-2) = 2
0 0
        max(0,0) = 0
```

تابع max() یک متغیر محلی به نام z دارد که مقدار بزرگ‌تر در آن نگهداری شده و سپس این مقدار با استفاده از دستور return به تابع main() می‌گردد.

توابع می‌توانند بیش از یک دستور return داشته باشند. مثلا تابع max() را مانند این نیز می‌توانستیم بنویسیم:

```
int max(int x, int y)
{  // returns larger of the two given integers:
   if (x < y) return y;
   else return x;
}
```

در این کد هر دستور return که زودتر اجرا شود مقدار مربوطه‌اش را بازگشت داده و تابع را خاتمه می‌دهد.

دستور return نوعی دستور پرش است (شبیه دستور break) زیرا اجرا را به بیرون از تابع هدایت می‌کند. اگرچه معمولا return در انتهای تابع قرار می‌گیرد، می‌توان آن را در هر نقطهٔ دیگری از تابع قرار داد.

5-5 اعلان‌ها و تعاریف تابع

در دو مثال آخر، تعریف کامل تابع در ابتدای برنامه آمد و زیر آن متن برنامهٔ اصلی قرار گرفت. این یک روش تعریف توابع است که اغلب برای برنامه‌های آزمون از آن استفاده می‌شود. راه دیگری که بیشتر رواج دارد این گونه است که ابتدا تابع ***اعلان***[1] شود، سپس متن برنامهٔ اصلی بیاید، پس از برنامهٔ اصلی ***تعریف***[2] کامل تابع قرار بگیرد. این روش در مثال بعدی نشان داده شده است.

اعلان تابع با تعریف تابع تفاوت دارد. اعلان تابع، فقط عنوان تابع است که یک سمیکولن در انتهای آن قرار دارد ولی تعریف تابع، متن کامل تابع است که هم شامل عنوان است و هم شامل بدنه. اعلان تابع شبیه اعلان متغیرهاست. یک متغیر قبل از این

1 – Declaration 2 - Definition

که به کار گرفته شود باید اعلان شود. تابع هم همین طور است با این فرق که متغیر را در هر جایی از برنامه می‌توان اعلان کرد اما تابع را باید قبل از برنامهٔ اصلی اعلان نمود. در اعلان تابع فقط بیان می‌شود که نوع بازگشتی تابع چیست، نام تابع چیست و نوع پارامترهای تابع چیست. همین‌ها برای کامپایلر کافی است تا بتواند کامپایل برنامه را آغاز کند. بعدا در زمان اجرا به تعریف بدنهٔ تابع نیز احتیاج می‌شود که این بدنه در انتهای برنامه و پس از تابع main() قرار می‌گیرد.

اکنون باید فرق بین «آرگومان[1]» و «پارامتر[2]» را بدانیم. ***پارامترها*** متغیرهایی هستند که در فهرست پارامتر یک تابع نام برده می‌شوند. در مثال قبلی x و y پارامترهای تابع max() هستند. پارامترها متغیرهای محلی برای تابع محسوب می‌شوند؛ یعنی فقط در طول اجرای تابع وجود دارند. ***آرگومان‌ها*** متغیرهایی هستند که از برنامهٔ اصلی به تابع فرستاده می‌شوند. در مثال قبلی m و n آرگومان‌های تابع max() هستند. وقتی یک تابع فراخوانی می‌شود، مقدار آرگومان‌ها درون پارامترهای تابع قرار می‌گیرد تا تابع پردازش را شروع کند. به این ترتیب می‌گوییم که آرگومان‌ها «به روش مقدار» ارسال شده اند. یعنی مقدار آرگومان‌ها جایگزین پارامترهای متناظرشان می‌شوند. در مثال بالا وقتی تابع max() فراخوانی می‌شود، مقدار آرگومان‌های m و n به ترتیب جایگزین پارامترهای x و y می‌شود و سپس تابع کارش را شروع می‌کند. می‌توان به جای آرگومان‌ها، یک مقدار ثابت را به تابع فرستاد (مثل max(22,44)) یا می‌توان یک عبات را به تابع فرستاد (مثل max(2*m, 3-n) که مقدار 2*m در x قرار می‌گیرد و مقدار 3-n در y قرار می‌گیرد).

× مثال 6-5 تابع max() با اعلان جدا از تعریف آن

این برنامه همان برنامهٔ آزمون تابع max() در مثال 5-5 است. اما این‌جا اعلان تابع بالای تابع اصلی ظاهر شده و تعریف تابع بعد از برنامهٔ اصلی آمده است:

```
int max(int,int);
// returns larger of the two given integers:

int main()
```

1 – Argument 2 - Parameter

```
{  // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
   }
   while (m != 0);
}

int max(int x, int y)
{  if (x < y) return y;
   else return x;
}
```

توجه کنید که پارامترهای x و y در بخش عنوان تعریف تابع آمده‌اند (طبق معمول) ولی در اعلان تابع وجود ندارند.

6-5 کامپایل جداگانهٔ توابع

اغلب این طور است که تعریف و بدنهٔ توابع در فایل‌های جداگانه‌ای قرار می‌گیرد. این فایل‌ها به طور مستقل کامپایل[1] می‌شوند و سپس به برنامهٔ اصلی که آن توابع را به کار می‌گیرد الصاق[2] می‌شوند. توابع کتابخانهٔ C++ استاندارد به همین شکل پیاده‌سازی شده‌اند و هنگامی که یکی از آن توابع را در برنامه‌هایتان به کار می‌برید باید با دستور راهنمای پیش‌پردازنده، فایل آن توابع را به برنامه‌تان ضمیمه کنید. این کار چند مزیت دارد. اولین مزیت «مخفی‌سازی اطلاعات» است. یعنی این که توابع لازم را در فایل جداگانه‌ای تعریف و کامپایل کنید و سپس آن فایل را به همراه مشخصات توابع به برنامه‌نویس دیگری بدهید تا برنامهٔ اصلی را تکمیل کند. به این ترتیب آن برنامه‌نویس از جزییات توابع و نحوهٔ اجرای داخلی آن‌ها چیزی نمی‌داند (نباید هم بداند) و فقط می‌داند که چطور می‌تواند از آن‌ها استفاده کند. در نتیجه اطلاعاتی که دانستن آن‌ها برای برنامه‌نویس ضروری نیست از دید او مخفی می‌ماند. تجربه نشان

1 – Compiling 2 - Linking

داده که پنهان‌سازی اطلاعات، فهمیدن برنامهٔ اصلی را آسان می‌کند و پروژه‌های بزرگ با موفقیت اجرا می‌شوند.

مزیت دیگر این است که توابع مورد نیاز را می‌توان قبل از این که برنامهٔ اصلی نوشته شود، جداگانه آزمایش نمود. وقتی یقین کردید که یک تابع مفروض به درستی کار می‌کند، آن را در یک فایل ذخیره کنید و جزییات آن تابع را فراموش کنید و هر وقت که به آن تابع نیاز داشتید با خیالی راحت از آن در برنامه‌هایتان استفاده نمایید. نتیجه این است که تولید توابع مورد نیاز و تولید برنامهٔ اصلی، هم‌زمان و مستقل از هم پیش می‌رود بدون این که یکی منتظر دیگری بماند. به این دیدگاه «بسته‌بندی نرم‌افزار» می‌گویند.

سومین مزیت این است که در هر زمانی به راحتی می‌توان تعریف توابع را عوض کرد بدون این که لازم باشد برنامهٔ اصلی تغییر یابد. فرض کنید تابعی برای مرتب کردن فهرستی از اعداد ایجاد کرده‌اید و آن را جداگانه کامپایل و ذخیره نموده‌اید و در یک برنامهٔ کاربردی هم از آن استفاده برده‌اید. حالا هرگاه که الگوریتم سریع‌تری برای مرتب‌سازی یافتید، فقط کافی است فایل تابع را اصلاح و کامپایل کنید و دیگر نیازی نیست که برنامهٔ اصلی را دست‌کاری نمایید.

چهارمین مزیت هم این است که می‌توانید یک بار یک تابع را کامپایل و ذخیره کنید و از آن پس در برنامه‌های مختلفی از همان تابع استفاده ببرید. وقتی شروع به نوشتن یک برنامهٔ جدید می‌کنید، شاید برخی از توابع مورد نیاز را از قبل داشته باشید. بنابراین دیگر لازم نیست که آن توابع را دوباره نوشته و کامپایل کنید. این کار سرعت تولید نرم افزار را افزایش می‌دهد.

تابع `max()` را به خاطر بیاورید. برای این که این تابع را در فایل جداگانه‌ای قرار دهیم، تعریف آن را در فایلی به نام `max.cpp` ذخیره می‌کنیم. فایل `max.cpp` شامل کد مقابل است:

max.cpp

```
int max(int x, int y)
{  if (x < y) return y;
   else return x;
}
```

حالا سراغ برنامهٔ اصلی می‌رویم. متن برنامهٔ اصلی را در فایلی به نام test.cpp ذخیره می‌نماییم. این فایل شامل کد زیر است:

test.cpp

```
int max(int,int);
// returns larger of the two given integers:

int main()
{  // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
  }
   while (m != 0);
}
```

در برنامهٔ اصلی، تابع max() فقط اعلان شده است. در اولین خط از برنامهٔ اصلی اعلان شده که تابع max() دو پارامتر از نوع int دارد و یک مقدار از نوع int برمی‌گرداند.

نحوهٔ کامپایل کردن فایل‌ها و الصاق آن‌ها به یکدیگر به نوع سیستم عامل و نوع کامپایلر بستگی دارد. در سیستم عامل ویندوز معمولا توابع را در فایل‌هایی از نوع DLL[1] کامپایل و ذخیره می‌کنند و سپس این فایل را در برنامهٔ اصلی احضار می‌نمایند. فایل‌های DLL را به دو طریق ایستا و پویا می‌توان مورد استفاده قرار داد. برای آشنایی بیشتر با فایل‌های DLL به مرجع ویندوز و کامپایلرهای c++ مراجعه کنید.

6-5 متغیرهای محلی، توابع محلی

متغیر محلی[2]، متغیری است که در داخل یک بلوک اعلان گردد. این گونه متغیرها فقط در داخل همان بلوکی که اعلان می‌شوند قابل دستیابی هستند. چون بدنهٔ تابع، خودش یک بلوک است پس متغیرهای اعلان شده در یک تابع متغیرهای محلی

1 – Data Link Library

1 – Local variable

برای آن تابع هستند. این متغیرها فقط تا وقتی که تابع در حال کار است وجود دارند. پارامترهای تابع نیز متغیرهای محلی محسوب می‌شوند.

× مثال 7–5 تابع فاکتوریل

اعداد فاکتوریل را در مثال 8–4 دیدیم. فاکتوریل عدد صحیح n برابر است با:

```
n! = n(n-1)(n-2)..(3)(2)(1)
```

تابع زیر، فاکتوریل عدد n را محاسبه می‌کند:

```
long fact(int n)
{  //returns n! = n*(n-1)*(n-2)*...*(2)*(1)
   if (n < 0) return 0;
   int f = 1;
   while (n > 1)
      f *= n--;
   return f;
}
```

این تابع دو متغیر محلی دارد: n و f. پارامتر n یک متغیر محلی است زیرا در فهرست پارامترهای تابع اعلان شده. متغیر f نیز محلی است زیرا درون بدنهٔ تابع اعلان شده. تابع فاکتوریل را با استفاده از برنامهٔ آزمون زیر می‌توان آزمایش کرد:

```
long fact(int);
// returns n! = n*(n-1)*(n-2)*...*(2)*(1)

int main()
{  // tests the factorial() function:
   for (int i=-1; i < 6; i++)
      cout << " " << fact(i);
   cout << endl;
}
```

```
0 1 1 2 6 24 120
```

برای این که برنامهٔ بالا قابل اجرا باشد، یا باید فایل تابع فاکتوریل را به آن الصاق کنیم و یا این که تعریف تابع فاکتوریل را به انتهای آن اضافه نماییم.

همان گونه که متغیرها می‌توانند محلی باشند، توابع نیز می‌توانند محلی باشند. یک ***تابع محلی***[1] تابعی است که درون یک تابع دیگر به کار رود. با استفاده از چند تابع ساده و ترکیب آن‌ها می‌توان توابع پیچیده‌تری ساخت. به مثال زیر نگاه کنید.

× مثال 8-5 تابع جایگشت

در ریاضیات، تابع جایگشت را با p(n,k) نشان می‌دهند. این تابع بیان می‌کند که به چند طریق می‌توان k عنصر دلخواه از یک مجموعهٔ n عنصری را کنار یکدیگر قرار داد. برای این محاسبه از رابطهٔ زیر استفاده می‌شود:

$$P(n,k)=\frac{n!}{(n-k)!}$$

برای مثال:

$$P(4,2)=\frac{4!}{(4-2)!}=\frac{4!}{2!}=\frac{24}{2}=12$$

پس به 12 طریق می‌توانیم دو عنصر دلخواه از یک مجموعهٔ چهار عنصری را کنار هم بچینیم. برای دو عنصر از مجموعهٔ {1, 2, 3, 4} حالت‌های ممکن عبارت است از:

```
12, 13, 14, 21, 23, 24, 31, 32, 34, 41, 42, 43
```

کد زیر تابع جایگشت را پیاده‌سازی می‌کند:

```
long perm(int n, int k)
{  // returns P(n,k), the number of the permutations of k from n:
   if (n < 0) || k < 0 || k > n) return 0;
   return fact(n)/fact(n-k);
}
```

این تابع، خود از تابع دیگری که همان تابع فاکتوریل است استفاده کرده. شرط به کار رفته در دستور if برای محدود کردن حالت‌های غیر ممکن استفاده شده است. در این حالت‌ها، تابع مقدار 0 را برمی‌گرداند تا نشان دهد که یک ورودی اشتباه وجود داشته است. برنامهٔ آزمون برای تابع perm() در ادامه آمده است:

---

1 – Local function

```
long perm(int,int);
// returns P(n,k), the number of permutations of k from n:

int main()
{  // tests the perm() function:
   for (int i = -1; i < 8; i++)
   {  for (int j= -1; j <= i+1; j++)
         cout << " " << perm(i,j);
      cout << endl;
  }
}
```

```
0 0
0 1 0
0 1 1 0
0 1 2 2 0
0 1 3 6 6 0
0 1 4 12 24 24 0
0 1 5 20 60 120 120 0
0 1 6 30 120 360 720 720 0
0 1 7 42 210 840 2520 5040 5040 0
```

البته ضروری است که تعریف دو تابع perm() و fact() در یک فایل باشد.

5-7 تابع void

لازم نیست یک تابع حتما مقداری را برگرداند. در C++ برای مشخص کردن چنین توابعی از کلمهٔ کلیدی void به عنوان نوع بازگشتی تابع استفاده می‌کنند. یک تابع void تابعی است که هیچ مقدار بازگشتی ندارد.

× مثال 5-9 تابعی که به جای شمارهٔ ماه‌ها ، نام آن‌ها را می‌نویسد

```
void PrintDate(int,int,int);
// prints the given date in literal form:

int main()
{  // tests the PrintDate() function:
   int day, month, year;
   do
```

```
   {  cin >> day >> month >> year;
      PrintDate(day,month,year);
   }
   while (month > 0);
}

void PrintDate(int d, int m, int y)
{  // prints the given date in literal form:
 if (d < 1 || d > 31 || m < 1 || m > 12 || y < 0)
   {  cout << "Error: parameter out of range.\n";
      return;
    }
   Cout << d;
   switch (m)
   {  case  1:  cout << "Farvardin ";   break;
      case  2:  cout << "Ordibehesht "; break;
      case  3:  cout << "Khordad ";     break;
      case  4:  cout << "Tir ";         break;
      case  5:  cout << "Mordad ";      break;
      case  6:  cout << "Shahrivar ";   break;
      case  7:  cout << "Mehr ";        break;
      case  8:  cout << "Aban ";        break;
      case  9:  cout << "Azar ";       break;
      case 10:  cout << "Dey ";        break;
      case 11:  cout << "Bahman ";     break;
      case 12:  cout << "Esfnad ";     break;
   }
   cout << y << endl;
}
```

```
7 12 1383
7 Esfand 1383
15 8 1384
15 Aban 1384
0 0 0
Error: parameter out of range.
```

تابع PrintDate() هیچ مقداری را برنمی‌گرداند. تنها هدف این تابع، چاپ تاریخ است. بنابراین نوع بازگشتی آن void است. اگر پارامترها خارج از محدوده باشند، تابع بدون این که چیزی چاپ کند خاتمه می‌یابد. با این وجود باز هم احتمال دارد مقادیر غیرممکنی مانند 31 Esfand 1384 چاپ شوند. اصلاح این ناهنجاری را به عنوان تمرین به دانشجو وا می‌گذاریم.

از آن‌جا که یک تابع void مقداری را برنمی‌گرداند، نیازی به دستور return نیست ولی اگر قرار باشد این دستور را در تابع void قرار دهیم، باید آن را به شکل تنها استفاده کنیم بدون این که بعد از کلمهٔ return هیچ چیز دیگری بیاید:

```
return;
```

در این حالت دستور return فقط تابع را خاتمه می‌دهد.

توابع void معمولا برای انجام یک کار مشخص استفاده می‌شوند مثل تابع بالا که تاریخ عددی را گرفته و شکل حرفی آن را چاپ می‌کند. به همین دلیل برنامه‌نویسان معمولا اسم این توابع را به شکل یک گزارهٔ فعلی انتخاب می‌کنند. مثلا نام تابع فوق PrintDate است که نشان می‌دهد این تابع کار چاپ تاریخ را انجام می‌دهد. رعایت این قرارداد به خوانایی و درک بهتر برنامه‌تان کمک می‌کند.

5-8 توابع بولی

در بسیاری از اوقات لازم است در برنامه، شرطی بررسی شود. اگر بررسی این شرط به دستورات زیادی نیاز داشته باشد، بهتر است که یک تابع این بررسی را انجام دهد. این کار مخصوصا هنگامی که از حلقه‌ها استفاده می‌شود بسیار مفید است. توابع بولی فقط دو مقدار را برمی‌گردانند: true یا false .

× مثال 10-5 تابعی که اول بودن اعداد را بررسی می‌کند

کد زیر یک تابع بولی است که تشخیص می‌دهد آیا عدد صحیح ارسال شده به آن، اول است یا خیر:

```
bool isPrime(int n)
{  // returns true if n is prime, false otherwise:
```

```
   float sqrtn = sqrt(n);
   if (n < 2) return false;       // 0 and 1 are not primes
   if (n < 4) return true;        // 2 and 3 are the first primes
   if (n%2 == 0) return false;    // 2 is the only even prime
   for (int d=3; d <= sqrtn; d += 2)
      if (n%d == 0) return false; // n has a nontrivial divisor
   return true;                   // n has no nontrivial divisors
}
```

تابع فوق برای عدد n به دنبال یک مقسوم‌علیه می‌گردد. اگر پیدا شد مقدار false را برمی‌گرداند یعنی n اول نیست. اگر هیچ مقسوم‌علیهی یافت نشد مقدار true را برمی‌گرداند که یعنی n اول است. این تابع برای یافتن مقسوم‌علیه فرض‌های زیر را در نظر می‌گیرد: 1 – اعداد کوچک‌تر از دو اول نیستند. 2– عدد دو اول است. 3 – هر عدد زوج غیر از دو اول نیست. 4 – حداقل یکی از مقسوم‌علیه‌های عدد از جذر آن عدد کوچک‌تر است. این فرض‌ها یکی یکی بررسی می‌شوند. اگر n از دو کوچک‌تر باشد مقدار false برمی‌گردد. اگر n برابر با 2 یا 3 باشد مقدار true برمی‌گردد یعنی n اول است. در غیر این صورت اگر n زوج باشد باز هم false برمی‌گردد زیرا هیچ عدد زوجی غیر از دو اول نیست. اگر n زوج هم نبود آنگاه حلقهٔ for شروع می‌شود و در این حلقه بررسی می‌شود که آیا عددی هست که از جذر n کوچک‌تر بوده و مقسوم‌علیه n باشد یا خیر. دقت کنید که در حلقهٔ for فقط کافی است اعداد فرد کوچک‌تر از جذر n را بررسی کنیم (چرا؟)

یک برنامهٔ آزمون و خروجی آن در ادامه آمده است:

```
#include <cmath>      // defines the sqrt() function
#include <iostream>   // defines the cout object
using namespace std;

bool isPrime(int);
// returns true if n is prime, false otherwise;
int main()
{  for (int n=0; n < 80; n++)
      if (isPrime(n)) cout << n << " ";
   cout << endl;
}
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
```

بهتر است بدانید که این تابع، بهینه نیست. هر عدد مرکب(غیر اول) را می‌توان به صورت ضرب یک عدد اول در عدد دیگری نوشت. به همین دلیل برای تشخیص اول بودن یک عدد کافی است بررسی شود که آیا این عدد به اعداد اول قبل از خودش قابل تقسیم است یا خیر. برای این منظور هم باید تمامی اعداد اول یافته شده را در آرایه‌ای ذخیره کنیم. آرایه‌ها را در فصل بعدی بررسی می‌کنیم.

اسم توابع بولی را معمولا به شکل سوالی انتخاب می‌کنند زیرا توابع بولی همیشه به یک سوال مفروض پاسخ بلی یا خیر می‌دهند. تابعی که در مثال بالا توضیح داده شد `isPrime` نام گرفته زیرا پاسخ می‌دهد که آیا عدد مذکور اول است یا خیر. این نحو نام‌گذاری گرچه اجباری نیست اما درک برنامه را آسان‌تر می‌کند و به یادآوری وظیفهٔ تابع نیز کمک می‌نماید. در کتابخانهٔ C++ استاندارد توابع بولی مثل `isLower()` یا `isUpper()` به همین شیوه نام‌گذاری شده‌اند.

9-5 توابع ورودی/خروجی[1] (I/O)

بخش‌هایی از برنامه که به جزییات دست و پا گیر می‌پردازد و خیلی به هدف اصلی برنامه مربوط نیست را می‌توان به توابع سپرد. در چنین شرایطی سودمندی توابع محسوس‌تر می‌شود. فرض کنید نرم‌افزاری برای سیستم آموزشی دانشگاه طراحی کرده‌اید که سوابق تحصیلی دانشجویان را نگه می‌دارد. در این نرم‌افزار لازم است که سن دانشجو به عنوان یکی از اطلاعات پروندهٔ دانشجو وارد شود. اگر وظیفهٔ دریافت سن را به عهدهٔ یک تابع بگذارید، می‌توانید جزییاتی از قبیل کنترل ورودی معتبر، یافتن سن از روی تاریخ تولد و ... را در این تابع پیاده‌سازی کنید بدون این که از مسیر برنامهٔ اصلی منحرف شوید.

قبلا نمونه‌ای از توابع خروجی را دیدیم. تابع `PrintDate()` در مثال 9-5 هیچ چیزی به برنامهٔ اصلی برنمی‌گرداند و فقط برای چاپ نتایج به کار می‌رود. این تابع نمونه‌ای از توابع خروجی است؛ یعنی توابعی که فقط برای چاپ نتایج به کار می‌روند و هیچ مقدار بازگشتی ندارند. توابع ورودی نیز به همین روش کار می‌کنند اما

1 – Input/Output functions

در جهت معکوس. یعنی توابع ورودی فقط برای دریافت ورودی و ارسال آن به برنامهٔ اصلی به کار می‌روند و هیچ پارامتری ندارند. مثال بعد یک تابع ورودی را نشان می‌دهد.

× مثال 11–5 تابعی برای دریافت سن کاربر

تابع سادهٔ زیر، سن کاربر را درخواست می‌کند و مقدار دریافت شده را به برنامهٔ اصلی می‌فرستد. این تابع تقریبا هوشمند است و هر عدد صحیح ورودی غیر منطقی را رد می‌کند و به طور مکرر درخواست ورودی معتبر می‌کند تا این که یک عدد صحیح در محدودهٔ 7 تا 120 دریافت دارد:

```
int age()
{  // prompts the user to input his/her age and returns that value:
   int n;
   while (true)
   {  cout << "How old are you: ";
      cin >> n;
      if (n < 0) cout << "\a\tYour age could not
                         be negative.";
      else if (n > 120) cout << "\a\tYou could not
                         be over 120.";
      else return n;
      cout << "\n\tTry again.\n";
  }
}
```

شرط کنترل حلقه، true است و این حلقه به ظاهر بی‌پایان به نظر می‌رسد. اما دستور return درون حلقه نه تنها مقدار ورودی معتبر را به برنامهٔ اصلی می‌فرستد بلکه هم حلقه را خاتمه می‌دهد و هم تابع را. به محض این که ورودی دریافت شده از cin معتبر باشد، دستور return اجرا شده و مقدار مذکور به برنامهٔ اصلی ارسال می‌شود و تابع خاتمه می‌یابد. اگر ورودی قابل قبول نباشد (n<7 یا n>120) آنگاه یک بوق اخطار پخش می‌شود (که این بوق حاصل چاپ کاراکتر \a است) و سپس یک توضیح روی صفحه‌نمایش درج می‌شود که کاربر می خواهد دوباره تلاش کند.

توجه کنید که این مثالی است که دستور return در انتهای تابع قرار نگرفته. علاوه بر این فهرست پارامترهای تابع خالی است زیرا از برنامهٔ اصلی چیزی دریافت نمی‌کند و فقط یک عدد صحیح را به برنامهٔ اصلی برمی‌گرداند. با این وجود لازم است که پرانتز هم در اعلان تابع و هم در فراخوانی تابع قید شود.

یک برنامهٔ آزمون و خروجی حاصل از آن در ادامه آمده است:

```
int age()

int main()
{  // tests the age() function:
   int a = age();
   cout << "\nYou are " << a << " years old.\n";
}
```

```
How old are you? 125
  You could not be over 120
  Try again.
How old are you? -3
  Your age could not be negative
  Try again.
How old are you? 99

You are 99 years old.
```

14-5 ارسال به طریق ارجاع[1] (آدرس)

تا این لحظه تمام پارامترهایی که در توابع دیدیم به طریق ***مقدار*** ارسال شده‌اند. یعنی ابتدا مقدار متغیری که در فراخوانی تابع ذکر شده برآورد می‌شود و سپس این مقدار به پارامترهای محلی تابع فرستاده می‌شود. مثلا در فراخوانی cube(x) ابتدا مقدار x برآورد شده و سپس این مقدار به متغیر محلی n در تابع فرستاده می‌شود و پس از آن تابع کار خویش را آغاز می‌کند. در طی اجرای تابع ممکن است مقدار n تغییر کند اما چون n محلی است هیچ تغییری روی مقدار x نمی‌گذارد. پس خود x به تابع نمی‌رود بلکه مقدار آن درون تابع کپی می‌شود. تغییر دادن این مقدار کپی شده درون تابع هیچ تاثیری بر x اصلی ندارد. به این ترتیب تابع می‌تواند مقدار x را بخواند

1 – Reference

اما نمی‌تواند مقدار x را تغییر دهد. به همین دلیل به x یک پارامتر «فقط خواندنی» می‌گویند. وقتی ارسال به وسیلهٔ مقدار باشد، هنگام فراخوانی تابع می‌توان از عبارات استفاده کرد. مثلا تابع cube() را می‌توان به صورت cube(2*x-3) فراخوانی کرد یا به شکل cube(2*sqrt(x)-cube(3)) فراخوانی نمود. در هر یک از این حالات، عبارت درون پرانتز به شکل یک مقدار تکی برآورد شده و حاصل آن مقدار به تابع فرستاده می‌شود.

ارسال به طریق مقدار باعث می‌شود که متغیرهای برنامهٔ اصلی از تغییرات ناخواسته در توابع مصون بمانند. اما گاهی اوقات عمدا می‌خواهیم این اتفاق رخ دهد. یعنی می‌خواهیم که تابع بتواند محتویات متغیر فرستاده شده به آن را دست‌کاری کند. در این حالت از ***ارسال به طریق ارجاع*** استفاده می‌کنیم.

برای این که مشخص کنیم یک پارامتر به طریق ارجاع ارسال می‌شود، علامت **&** را به نوع پارامتر در فهرست پارامترهای تابع اضافه می‌کنیم. این باعث می‌شود که تابع به جای این که یک کپی محلی از آن آرگومان ایجاد کند، خود آرگومان محلی را به کار بگیرد. به این ترتیب تابع هم می‌تواند مقدار آرگومان فرستاده شده را بخواند و هم می‌تواند مقدار آن را تغییر دهد. در این حالت آن پارامتر یک پارامتر «خواندنی–نوشتنی» خواهد بود. هر تغییری که روی پارامتر خواندنی–نوشتنی در تابع صورت گیرد به طور مستقیم روی متغیر برنامهٔ اصلی اعمال می‌شود. به مثال زیر نگاه کنید.

× مثال 12–5 تابع swap()

تابع کوچک زیر در مرتب کردن داده‌ها کاربرد فراوان دارد:

```
void swap(float& x, float& y)
{  // exchanges the values of x and y:
   float temp = x;
   x = y;
   y = temp;
}
```

هدف این تابع جابجا کردن دو عنصری است که به آن فرستاده می‌شوند. برای این منظور پارامترهای x و y به صورت پارامترهای ارجاع تعریف شده‌اند:

float& x, float& y

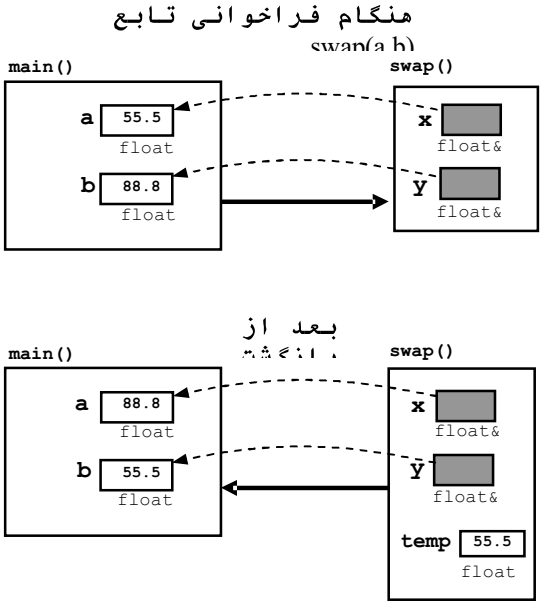
عملگر ارجاع **&** موجب می‌شود که به جای x و y آرگومان‌های ارسالی قرار بگیرند. برنامهٔ آزمون و اجرای آزمایشی آن در زیر آمده است:

```
void swap(float&, float&)
// exchanges the values of x and y:

int main()
{  // tests the swap() function:
   float a = 55.5, b = 88.8;
   cout << "a = " << a << ", b = " << b << endl;
   swap(a,b);
   cout << "a = " << a << ", b = " << b << endl;
}
```

```
a = 55.5, b = 88.8
a = 88.8, b = 55.5
```

وقتی فراخوانی swap(a,b) اجرا می‌شود، x به a اشاره می‌کند و y به b. سپس متغیر محلی temp اعلان می‌شود و مقدار x (که همان a است) درون آن قرار می‌گیرد. پس از آن مقدار y (که همان b است) درون x (یعنی a) قرار می‌گیرد و آنگاه مقدار temp درون y (یعنی b) قرار داده می‌شود. نتیجهٔ نهایی این است که مقادیر a و b با یکدیگر جابجا می شوند. شکل مقابل نشان می‌دهد که چطور این جابجایی رخ می‌دهد:



به اعلان تابع swap() دقت کنید:

```
void swap(float&, float&)
```

این اعلان شامل عملگر ارجاع & برای هر پارامتر است. برنامه‌نویسان C عادت دارند که عملگر ارجاع & را به عنوان پیشوند نام متغیر استفاده کنند (مثل float &x) در C++ فرض می‌کنیم عملگر ارجاع & پسوند نوع است (مثل float& x) به هر حال کامپایلر هیچ فرقی بین این دو اعلان نمی‌گذارد و شکل نوشتن عملگر ارجاع کاملا اختیاری و سلیقه‌ای است.

× مثال 13-5 ارسال به طریق مقدار و ارسال به طریق ارجاع

این برنامه، تفاوت بین ارسال به طریق مقدار و ارسال به طریق ارجاع را نشان می‌دهد:

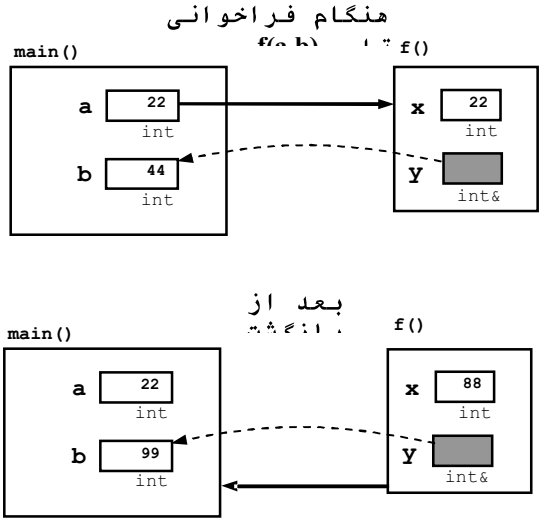
```
void f(int,int&);
// changes reference argument to 99:

int main()
{  // tests the f() function:
   int a = 22, b = 44;
   cout << "a = " << a << ", b = " << b << endl;
   f(a,b);
   cout << "a = " << a << ", b = " << b << endl;
   f(2*a-3,b);
   cout << "a = " << a << ", b = " << b << endl;
}

void f(int x , int& y)
{  // changes reference argument to 99:
   x = 88;
   y = 99;
}
```

```
a = 22, b = 44
a = 22, b = 99
a = 22, b = 99
```

تابع f() دو پارامتر دارد که اولی به طریق مقدار و دومی به طریق ارجاع ارسال می‌شود. فراخوانی f(a,b) باعث می‌شود که a از طریق مقدار به x ارسال شود و b از طریق ارجاع به y فرستاده شود. بنابراین x یک متغیر محلی است که مقدار 22 به آن فرستاده می‌شود در حالی که y یک ارجاع به متغیر b است که مقدار فعلی آن 44 می‌باشد. در تابع f() مقدار 88 در x قرار می‌گیرد که این تاثیری بر a ندارد. همچنین مقدار 99 در y قرار می‌گیرد که چون y در حقیقت یک نام مستعار برای b است، مقدار b به 99 تغییر می‌کند. هنگامی که تابع خاتمه یابد، a هنوز مقدار 22 را دارد ولی مقدار b به 99 تغییر یافته است. آرگومان a فقط خواندنی است و آرگومان b خواندنی-نوشتنی است. شکل زیر نحوهٔ کار تابع f() را نشان می‌دهد.



در جدول زیر خلاصهٔ تفاوت‌های بین ارسال از طریق مقدار و ارسال از طریق ارجاع آمده است.

ارسال از طریق مقدار در مقایسه با ارسال از طریق ارجاع

ارسال از طریق مقدار	ارسال از طریق ارجاع
int x;	int& x;
پارامتر x یک متغیر محلی است	پارامتر x یک ارجاع است
x یک کپی از آرگومان است	x مترادف با آرگومان است
تغییر محتویات آرگومان ممکن نیست	می‌تواند محتویات آرگومان را تغییر دهد

آرگومان ارسال شده از طریق مقدار می‌تواند یک ثابت، یک متغیر یا یک عبارت باشد	آرگومان ارسال شده از طریق ارجاع فقط باید یک متغیر باشد
آرگومان فقط خواندنی است	آرگومان خواندنی-نوشتنی است

یکی از مواقعی که پارامترهای ارجاع مورد نیاز هستند جایی است که تابع باید بیش از یک مقدار را بازگرداند. دستور return فقط می‌تواند یک مقدار را برگرداند. بنابراین اگر باید بیش از یک مقدار برگشت داده شود، این کار را پارامترهای ارجاع انجام می‌دهند.

× مثال 14-5 بازگشت بیشتر از یک مقدار

تابع زیر از طریق دو پارامتر راجاع، دو مقدار را بازمی‌گرداند: area و circumference (محیط و مساحت) برای دایره‌ای که شعاع آن عدد مفروض r است:

```
void ComputeCircle(double& area, double& circumference, double r)
{  // returns the area and circumference of a circle with radius r:
   const double PI = 3.141592653589793;
   area = PI*r*r;
   circumference = 2*PI*r;
}
```

برنامهٔ آزمون تابع فوق و یک اجرای آزمایشی آن در شکل زیر نشان داده شده است:

```
void ComputerCircle(double&, double&, double);
// returns the area and  circumference of a circle with radius r;

int main()
{  // tests the ComputeCircle() function:
   double r, a, c;
   cout << "Enter radius: ";
   cin >> r;
   ComputeCircle(a, c, r);
   cout << "area = " << a << ", circumference = "
        << c << endl;
}
```

```
Enter radius: 100
area = 31415.9, circumference = 628.319
```

در اعلان و تعریف تابع فوق، پارامترهایی که از طریق ارجاع ارسال می‌شوند در ابتدای فهرست پارامترها قرار داده شده‌اند. رعایت این قاعده باعث می‌شود که نظم برنامه حفظ شود و به سادگی بتوانید پارامترهای تابع را از یکدیگر تمیز دهید. البته این فقط یک قرارداد است و رعایت آن اجباری نیست.

5-11 ارسال از طریق ارجاع ثابت

ارسال پارامترها به طریق ارجاع دو خاصیت مهم دارد: اول این که تابع می‌تواند روی آرگومان واقعی تغییراتی بدهد و دوم این که از اشغال بی‌مورد حافظه جلوگیری می‌شود. وقتی یک آرگومان از طریق مقدار به تابع فرستاده شود، یک کپی محلی از آن آرگومان ایجاد شده و در اختیار تابع قرار می‌گیرد. این کپی به اندازهٔ آرگومان اصلی حافظه اشغال می‌کند. حال اگر آرگومان اصلی خیلی حجیم باشد (مثل یک تصویر گرافیکی) آنگاه ارسال از طریق مقدار باعث می‌شود که حافظه به میزان دوبرابر مصرف شود؛ بخشی برای آرگومان اصلی و بخشی دیگر برای نسخه محلی که در تابع به کار می‌رود. حال اگر این شیء حجیم را از طریق ارجاع به تابع ارسال کنیم دیگر نسخهٔ محلی ساخته نمی‌شود و حافظه‌ای هم هدر نمی‌رود. اما این کار یک عیب بزرگ دارد: تابع می‌تواند مقدار پارامتر ارجاع را دست‌کاری کند. اگر تابع نمی‌بایست پارامتر مذکور را دست‌کاری کند، آنگاه ارسال از طریق ارجاع مخاطره‌آمیز خواهد بود.

برای این که عیب مذکور برطرف شود و شیء اصلی از تغییرات ناخواسته درون تابع مصون باشد، C++ روش سومی را برای ارسال آرگومان پیشنهاد می‌کند: ارسال از طریق ***ارجاع ثابت***[1]. این روش مانند ارسال از طریق ارجاع است با این فرق که تابع نمی‌تواند محتویات پارامتر ارجاع را دست‌کاری نماید و فقط اجازهٔ خواندن آن را دارد. برای این که پارامتری را از نوع ارجاع ثابت اعلان کنیم باید عبارت `const` را به ابتدای اعلان آن اضافه نماییم.

1 – Constant reference

× مثال 15-5 ارسال از طریق ارجاع ثابت

سه طریقه ارسال پارامتر در تابع زیر به کار رفته است:

```
void f(int x, int& y, const int& z)
{  x += z;
   y += z;
   cout << "x = " << x << ", y = " << y << ", z = "
        << z << endl;
}
```

در تابع فوق اولین پارامتر یعنی x از طریق مقدار ارسال می‌شود، دومین پارامتر یعنی y از طریق ارجاع و سومین پارامتر نیز از طریق ارجاع ثابت. برنامهٔ آزمون و یک اجرای آزمایشی از آن در ذیل آمده است:

```
void f(int, int&, const int&);
int main()
{  // tests the f() function:
   int a = 22, b = 33, c = 44;
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
   f(a,b,c);
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
   f(2*a-3,b,c);
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
}
```

```
a = 22, b = 33, c = 44
x = 66, y = 77, z = 44
a = 22, b = 77, c = 44
x = 85, y = 121, z = 44
a = 22, b = 121, c = 44
```

تابع فوق پارامترهای x و y را می‌تواند تغییر دهد ولی قادر نیست پارامتر z را تغییر دهد. تغییراتی که روی x صورت می‌گیرد اثری روی آرگومان a نخواهد داشت زیرا a از طریق مقدار به تابع ارسال شده. تغییراتی که روی y صورت می‌گیرد روی آرگومان b هم تاثیر می‌گذارد زیرا b از طریق ارجاع به تابع فرستاده شده.

ارسال به طریق ارجاع ثابت بیشتر برای توابعی استفاده می‌شود که عناصر بزرگ را ویرایش می‌کنند مثل آرایه‌ها یا نمونهٔ کلاس‌ها که در فصل‌های بعدی توضیح آن‌ها آمده است. عناصری که از انواع اصلی هستند (مثل int یا float) به طریق مقدار ارسال می‌شوند به شرطی که قرار نباشد تابع محتویات آن‌ها را دست‌کاری کند.

5-12 توابع بی‌واسطه[1]

وقتی تابعی درون یک برنامه فراخوانی می‌شود، ابتدا باید مکان فعلی اجرای برنامهٔ اصلی و متغیرهای فعلی آن در جایی نگهداری شود تا پس از اتمام تابع، ادامه برنامه پی‌گیری شود. همچنین باید متغیرهای محلی تابع ایجاد شوند و حافظه‌ای برای آن‌ها تخصیص یابد و همچنین آرگومان‌ها به این متغیرها ارسال شوند تا درنهایت تابع شروع به کار کند. پس از پایان کار تابع نیز باید همین مسیر به شکل معکوس پیموده شود تا برنامهٔ اصلی ادامه یابد. انجام همهٔ این کارها هم زمان‌گیر است و هم حافظهٔ اضافی می‌طلبد. در اصطلاح می‌گویند که فراخوانی و اجرای تابع «سربار» دارد. در بعضی حالت‌ها بهتر است با تعریف تابع به شکل ***بی‌واسطه*** از سربار اجتناب کنیم. تابعی که به شکل بی‌واسطه تعریف می‌شود، ظاهری شبیه به توابع معمولی دارد با این فرق که عبارت inline در اعلان و تعریف آن قید شده است.

× مثال 5-16 تابع cube() به شکل بی‌واسطه

این همان تابع cube() مثال 3-5 است:

```
inline int cube(int x)
{  // returns cube of x:
   return x*x*x;
}
```

تنها تفاوت این است که کلمهٔ کلیدی inline در ابتدای عنوان تابع ذکر شده. این عبارت به کامپایلر می‌گوید که در برنامه به جای cube(n) کد واقعی (n) * (n) * (n) را قرار دهد. به برنامهٔ آزمون زیر نگاه کنید:

---

1 – Inline

```
int main()
{  // tests the cube() function:
   cout << cube(4) << endl;
   int x, y;
   cin >> x;
   y = cube(2*x-3);
}
```

این برنامه هنگام کامپایل به شکل زیر درمی‌آید، گویی اصلا تابعی وجود نداشته:

```
int main()
{  // tests the cube() function:
   cout << (4) * (4) * (4) << endl;
   int x, y;
   cin >> x;
   y = (2*x-3) * (2*x-3) * (2*x-3);
}
```

وقتی کامپایلر کد واقعی تابع را جایگزین فراخوانی آن می‌کند، می‌گوییم که تابع بی‌واسطه، ***باز می‌شود***.

احتیاط: استفاده از توابع بی‌واسطه می‌تواند اثرات منفی داشته باشد. مثلا اگر یک تابع بی‌واسطه دارای 40 خط کد باشد و این تابع در 26 نقطه مختلف از برنامهٔ اصلی فراخوانی شود، هنگام کامپایل بیش از هزار خط کد به برنامهٔ اصلی افزوده می‌شود. همچنین تابع بی‌واسطه می‌تواند قابلیت انتقال برنامهٔ شما را روی سیستم‌های مختلف کاهش دهد.

5-13 چندشکلی توابع

در C++ می‌توانیم چند تابع داشته باشیم که همگی یک نام دارند. در این حالت می‌گوییم که تابع مذکور، ***چندشکلی*** دارد. شرط این کار آن است که فهرست پارامترهای این توابع با یکدیگر تفاوت داشته باشد. یعنی تعداد پارامترها متفاوت باشد یا دست کم یکی از پارامترهای متناظر هم نوع نباشند.

× مثال 17-5 چندشکلی تابع ()max

در مثال 3-5 تابع ()max را تعریف کردیم. حالا توابع دیگری با همان نام ولی شکلی متفاوت تعریف می‌کنیم و همه را در یک برنامه به کار می‌گیریم:

```
int max(int, int);
int max(int, int, int);
int max(double, double);

int main()
{  cout << max(99,77) << " " << max(55,66,33) << " " <<
max(44.4,88.8);
}

int max(int x, int y)
{  // returns the maximum of the two given integers:
   return (x > y ? x : y);
}

int max(int x, int y, int z)
{  // returns the maximum of the three given integers:
   int m = (x > y ? x : y);  // m = max(x , y)
   return ( z > m ? z : m);
}

int max(double x, double y)
{  // return the maximum of the two given doubles:
   return (x>y ? x : y);
}
```

```
99 66 88.0
```

در این برنامه سه تابع با نام ()max تعریف شده است. وقتی تابع ()max در جایی از برنامه فراخوانی می‌شود، کامپایلر فهرست آرگومان آن را بررسی می‌کند تا بفهمد که کدام نسخه از max باید احضار شود. مثلا در اولین فراخوانی تابع ()max دو آرگومان int ارسال شده، پس نسخه‌ای که دو پارامتر int در فهرست پارامترهایش

دارد فراخوانی می‌شود. اگر این نسخه وجود نداشته باشد، کامپایلر intها را به double ارتقا می‌دهد و سپس نسخه‌ای که دو پارامتر double دارد را فرا می‌خواند.

توابعی که چندشکلی دارند بسیار فراوان در C++ استفاده می‌شوند. چندشکلی در کلاس‌ها اهمیت فراوانی دارد که این موضوع در فصل 12 بحث خواهد شد.

5-14 تابع main()

اکنون که با توابع آشنا شده‌ایم، نگاه دقیق‌تری به برنامه‌ها بیاندازیم. برنامه‌هایی که تا کنون نوشتیم همه دارای تابعی به نام main() هستند. منطق C++ این طور است که هر برنامه باید دارای تابعی به نام main() باشد. در حقیقت هر برنامه کامل، از یک تابع main() به همراه توابع دیگر تشکیل شده است که هر یک از این توابع به شکل مستقیم یا غیر مستقیم از درون تابع main() فراخوانی می‌شوند. خود برنامه با فراخوانی تابع main() شروع می‌شود. چون این تابع یک نوع بازگشتی int دارد، منطقی است که بلوک تابع main() شامل دستور return 0; باشد هرچند که در برخی از کامپایلرهای C++ این خط اجباری نیست و می‌توان آن را ذکر نکرد. مقدار صحیحی که با دستور return به سیستم عامل برمی‌گردد باید تعداد خطاها را شمارش کند. مقدار پیش‌فرض آن 0 است به این معنا که برنامه بدون خطا پایان گرفته است. با استفاده از دستور return می‌توانیم برنامه را به طور غیرمعمول خاتمه دهیم.

× مثال 5-18 استفاده از دستور return برای پایان دادن به یک برنامه

```
int main()
{  // prints the quotient of two input integers:
   int n, d;
   cout << "Enter two integers: ";
   cin >> n >> d;
   if (d = = 0) return 0;
   cout << n << "/" << d << " = " << n/d << endl;
}
```

```
Enter two integers: 99 17
99/17 = 5
```

اگر کاربر برای ورودی دوم 0 را وارد کند، برنامه بدون چاپ خروجی پایان می‌یابد:

```
Enter two integers: 99 0
```

دستور return تابع فعلی را خاتمه می‌دهد و کنترل را به فراخوانده بازمی‌گرداند. به همین دلیل است که اجرای دستور return در تابع main() کل برنامه را خاتمه می‌دهد.

چهار روش وجود دارد که بتوانیم برنامه را به شکل غیرمعمول (یعنی قبل از این که اجرا به پایان بلوک اصلی برسد) خاتمه دهیم:

1 – استفاده از دستور return

2 – فراخوانی تابع exit()

3 – فراخوانی تابع abort()

4 – ایجاد یک حالت استثنا[1]

طریقهٔ به‌کارگیری تابع exit() در مثال زیر شرح داده شده. این تابع در سرفایل <cstdlib> تعریف شده است. تابع exit() برای خاتمه دادن به کل برنامه در هر تابعی غیر از تابع main() مفید است. به مثال بعدی توجه کنید.

× مثال 19-5 استفاده از تابع exit() برای پایان دادن به برنامه

```
#include <cstdlib>     // defines the exit() function
#include <iostream>    // defines thi cin and cout objects
using namespace std;
double reciprocal(double x);

int main()
{  double x;
   cin >> x;
   cout << reciprocal(x);
}

double reciprocal(double x)
{  // returns the reciprocal of x:
```

1 – Exception

```
   if (x = = 0) exit(1);     // terminate the program
   return 1.0/x;
}
```

در برنامهٔ بالا اگر کاربر عدد 0 را وارد کند، تابع reciprocal() خاتمه می‌یابد و برنامه بدون هیچ مقدار چاپی به پایان می‌رسد.

15-5 آرگومان‌های پیش‌فرض[1]

در C++ می‌توان تعداد آرگومان‌های یک تابع را در زمان اجرا به دلخواه تغییر داد. این امر با استفاده از آرگومان‌های اختیاری و مقادیر پیش‌فرض امکان‌پذیر است.

× مثال 20-5 آرگومان‌های پیش‌فرض

برنامهٔ زیر حاصل چند جمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ را پیدا می‌کند. برای محاسبهٔ این مقدار از الگوریتم هورنر استفاده شده. به این شکل که برای کارایی بیشتر، محاسبه به صورت $a_0 + (a_1 + (a_2 + a_3x)x)x$ دسته‌بندی می‌شود:

```
double p(double, double, double=0, double=0, double=0);

int main()
{  // tests the p() function:
   double x = 2.0003;
   cout << "p(x,7) = " << p(x,7) << endl;
   cout << "p(x,7,6) = " << p(x,7,6) << endl;
   cout << "p(x,7,6,5) = " << p(x,7,6,5) << endl;
   cout << "p(x,7,6,5,4) = " << p(x,7,6,5,4) << endl;
}
double p(double x, double a0, double a1=0, double a2=0, double a3=0)
{  // returns a0 + a1*x + a2*x^2 + a3*x^3:
   return a0 + (a1 + (a2 + a3*x)*x)*x;
}
```

```
p(x,7) = 7
p(x,7,6) = 19.0018
p(x,7,6,5) = 39.0078
p(x,7,6,5,4) = 71.0222
```

1 – Default

هنگامی که p(x, a0, a1, a2, a3) فراخوانی شود، چندجمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ محاسبه می‌شود اما چون a1 و a2 و a3 مقدار پیش‌فرض 0 را دارند، تابع مذکور را می‌توان به صورت p(x,a0) نیز فراخوانی نمود. این فراخوانی معادل فراخوانی p(x,a0,0,0,0) است که برابر با a0 ارزیابی خواهد شد. همچنین می‌توان تابع فوق را به صورت p(x,a0,a1) فراخوانی نمود . این فراخوانی معادل فراخوانی p(x,a0,a1,0,0) است که چندجمله‌ای درجه اول $a_0 + a_1x$ را محاسبه می‌نماید. به همین ترتیب فراخوانی p(x,a0,a1,a2) چندجمله‌ای درجه دوم $a_0 + a_1x + a_2x^2$ را محاسبه می‌کند و همچنین فراخوانی p(x,a0,a1,a2,a3) چند جمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ را محاسبه می‌کند. پس این تابع را می‌توان با 2 یا 3 یا 4 یا 5 آرگومان فراخوانی کرد.

برای این که به یک پارامتر مقدار پیش‌فرض بدهیم باید آن مقدار را در فهرست پارامترهای تابع و جلوی پارامتر مربوطه به همراه علامت مساوی درج کنیم. به این ترتیب اگر هنگام فراخوانی تابع، آن آرگومان را ذکر نکنیم، مقدار پیش‌فرض آن در محاسبات تابع استفاده می‌شود. به همین خاطر به این گونه آرگومان‌ها، آرگومان اختیاری می‌گویند.

دقت کنید که پارامترهایی که مقدار پیش‌فرض دارند باید در فهرست پارامترهای تابع بعد از همهٔ پارامترهای اجباری قید شوند مثل:

```
void f( int a, int b, int c=4, int d=7, int e=3);     // OK
void g(int a, int b=2, int c=4, int d, int e=3);      // ERROR
```

همچنین هنگام فراخوانی تابع، آرگومان‌های ذکر شده به ترتیب از چپ به راست تخصیص می‌یابند و پارامترهای بعدی با مقدار پیش‌فرض پر می‌شوند. مثلا در تابع p() که در بالا قید شد، فراخوانی p(8.0,7,6) باعث می‌شود که پارامتر x مقدار 8.0 را بگیرد سپس پارامتر a0 مقدار 7 را بگیرد و سپس پارامتر a1 مقدار 6 را بگیرد. پارامترهای a2 و a3 مقدار پیش‌فرض‌شان را خواهند داشت. این ترتیب را نمی‌توانیم به هم بزنیم. مثلا نمی‌توانیم تابع را طوری فرا بخوانیم که پارامترهای x و a0 و a3 مستقیما مقدار بگیرند ولی پارامترهای a1 و a2 مقدار پیش‌فرض‌شان را داشته باشند.

پرسش‌های گزینه‌ای

1 – توابع ریاضی C++ استاندارد در کدام سرفایل تعریف شده‌اند؟

الف – سرفایل <iostream>　　ب – سرفایل <cmath>

ج – سرفایل <cstdlib>　　د – سرفایل <iomanip>

2 – در تعریف `int f(float a)` کدام گزینه صحیح نیست؟

الف – این کد تابعی به نام f را تعریف می‌کند

ب – تابع فوق متغیری از نوع float دارد

ج – نوع بازگشتی این تابع از نوع int است

د – پارامتر این تابع از نوع int است.

3 – دستور `return` در تابع چه کاری انجام می‌دهد؟

الف – تابع را خاتمه می‌دهد

ب – مقدار نهایی را به فراخواننده برمی‌گرداند

ج – نوع بازگشتی تابع را مشخص می‌کند

د – الف و ب

4 – کدام عبارت صحیح نیست؟

الف – پارامترهای تابع، متغیرهای محلی برای آن تابع محسوب می‌شوند

ب – متغیرهای اعلان شده در یک تابع، متغیرهای محلی آن تابع محسوب می‌شوند

ج – متغیرهای محلی تابع، فقط در طول اجرای تابع موجودند

د – متغیرهای محلی تابع در سراسر برنامه معتبرند

5 – تابعی که مقداری را برنمی‌گرداند، نوع بازگشتی آن چگونه اعلان می‌شود؟

الف – از نوع void　　ب – از نوع null

ج – از نوع پیش فرض int　　د – لازم نیست نوع بازگشتی قید شود

6 – نوع بازگشتی یک تابع بولی چیست؟

الف – int　　ب – void　　ج – bool　　د – const

7 – عملگر ارجاع کدام یک از گزینه‌های زیر است؟

الف – *　　ب – &　　ج – ->　　د – :

8 – چه زمانی یک آرگومان «خواندنی–نوشتنی» است؟

الف – وقتی از طریق مقدار ارسال شود

ب – وقتی از طریق ارجاع ثابت ارسال شود

ج – وقتی از طریق ارجاع ارسال شود

د – وقتی با پیشوند `const` اعلان شود

9 – کدام گزینه صحیح است؟

الف – فقط یک مقدار را می‌توان به تابع فرستاد و تابع فقط می‌تواند یک مقدار را بازگرداند

ب – فقط یک مقدار را می‌توان به تابع فرستاد ولی تابع می‌تواند چند مقدار را بازگرداند

ج – چند مقدار را می‌توان به تابع فرستاد ولی تابع می‌تواند فقط یک مقدار را بازگرداند

د – چند مقدار را می‌توان به تابع فرستاد و تابع می‌تواند چند مقدار را بازگرداند

10 – برای تعریف یک تابع به شکل بی‌واسطه از چه کلمهٔ کلیدی استفاده می‌کنیم؟

الف – `const` ب – `inline` ج – `void` د – `int`

11 – کدام عبارت در رابطه با چندشکلی توابع صحیح است؟

الف – یک تابع چندشکلی باید نام‌های متفاوت ولی بدنه‌های یکسان داشته باشد

ب – یک تابع چندشکلی باید نام‌های یکسان ولی فهرست پارامترهای متفاوت داشته باشد

ج – یک تابع چندشکلی باید نام‌های متفاوت ولی فهرست پارامترهای یکسان داشته باشد

د – یک تابع چندشکلی باید نام‌های یکسان و فهرست پارامترهای یکسان داشته باشد

12 – اگر تابع `f` به شکل `void f(int k, int x=0, int y=1)` اعلان شده باشد آنگاه:

الف – پارامتر `k` دارای مقدار پیش‌فرض نیست.

ب – پارامتر `x` دارای مقدار پیش‌فرض `0` است.

ج – پارامتر `y` دارای مقدار پیش‌فرض `1` است.

د – همه موارد فوق صحیح است.

13 – چرا در برنامه‌های بزرگ تعریف توابع را در فایل جداگانه‌ای قرار می‌دهند؟

الف – به این دلیل که مدیریت برنامه آسان شود

ب – به این دلیل که اصل پنهان‌سازی اطلاعات رعایت شود

ج – به این دلیل که بتوان در برنامه‌های دیگر هم از آن توابع استفاده کرد

د – همه موارد فوق

14 – اگر تابع g به شکل `void g(int m, int& n)` اعلان شده باشد آنگاه:

الف – پارامتر m به طریق ارجاع ارسال شده

ب – پارامتر n به طریق ارجاع ارسال شده

ج – پارامتر m به طریق ارجاع ثابت ارسال شده

د – پارامتر n به طریق ارجاع ثابت ارسال شده

15 – اگر تابع سوال 14 به شکل `g(x, y);` فراخوانی شود آنگاه کدام عبارت صحیح است؟

الف – تابع مقدار x را می‌تواند تغییر دهد

ب – تابع مقدار y را می‌تواند تغییر دهد

ج – تابع مقدار x و مقدار y را می‌تواند تغییر دهد

د – تابع مقدار هیچ کدام را نمی‌تواند تغییر دهد.

پرسش‌های تشریحی

1- استفاده از تابع برای بخش‌بندی برنامه چه مزایایی دارد؟
2- چه تفاوتی بین اعلان یک تابع و تعریف آن است؟
3- اعلان یک تابع کجا می‌تواند قرار بگیرد؟
4- برای استفاده از چه توابعی به دستور include نیاز است؟
5- گذاشتن تعریف یک تابع در یک فایل جداگانه چه مزیتی دارد؟
6- کامپایل کردن یک تابع به طور جداگانه چه مزیتی دارد؟
7- چه تفاوت‌هایی بین ارسال یک پارامتر از طریق مقدار و ارسال آن از طریق ارجاع وجود دارد؟
8- چه تفاوت‌هایی بین ارسال یک پارامتر از طریق ارجاع و ارسال آن از طریق ارجاع ثابت وجود دارد؟
9- چرا به پارامتری که از طریق مقدار ارسال می‌شود «فقط خواندنی» گفته می‌شود؟ چرا به پارامتری که از طریق ارجاع ارسال می‌شود «خواندنی-نوشتنی» گفته می‌شود؟
10- چه اشتباهی در اعلان زیر هست؟

```
int f(int a, int b=0, int c);
```

تمرین‌های برنامه‌نویسی

1- توضیح دهید چگونه یک تابع void با یک پارامتر ارجاع می‌تواند به یک تابع غیر void با یک پارامتر مقدار تبدیل گردد.
2- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت رابطهٔ مثلثاتی $\cos 2x = 2\cos^2 x - 1$ را تحقیق کند.
3- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت رابطهٔ مثلثاتی $\cos^2 x + \sin^2 x = 1$ را تحقیق کند.
4- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت تساوی $b^n = e^{(n \log b)}$ را تحقیق کند.
5- تابع min() را که به شکل زیر اعلان می‌شود، نوشته و آزمایش کنید. این تابع از میان چهار عدد صحیح ارسال شده، کوچک‌ترین عدد را برمی‌گرداند:

```
int min(int,int,int,int);
```

6- تابع max() را که به شکل زیر اعلان می‌شود نوشته و آزمایش کنید. این تابع با استفاده از تابع max(int,int) مثال 5-5 بزرگ‌ترین عدد در بین چهار عدد صحیح داده شده را برمی‌گرداند.

```
int max(int,int,int,int);
```

7- تابع min() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید. این تابع با استفاده از تابع min(int,int) کوچک‌ترین عدد را از میان چهار عدد صحیح ارسال شده به آن، پیدا کرده و برمی‌گرداند.

```
int main(int,int,int,int);
```

8- تابع average() را که میانگین چهار عدد را برمی‌گرداند، نوشته و آزمایش کنید:

```
float average(float x1, float x2, float x3, float x4)
```

9- تابع average() را که میانگین حداکثر چهار عدد را بر می‌گرداند، نوشته و آزمایش کنید:

```
float average(float x1, float x2 =0, float x3=0, float x4=0)
```

10- تابع فاکتوریال fact() را با یک حلقهٔ for پیاده‌سازی کنید(مثال 9-4 را ببینید). مشخص کنید که چه مقداری از n موجب می‌شود که fact(n) سرریز شود.

11- موثرترین طریق برای محاسبهٔ جایگشت تابع p(n,k) فرمول زیر است:

$$P(n,k) = n(n-1)(n-2)...(n-k+2)(n-k+1)$$

یعنی حاصل‌ضرب k عدد صحیح از n تا n-k+1 . با استفاده از این رابطه، تابع perm() مثال 10-5 را بازنویسی و آزمایش کنید.

12- تابع ترکیب c(n,k) تعداد زیرمجموعه‌های متفاوت (نامرتب) kعنصری که ممکن است از یک مجموعهٔ n عنصری ساخته شود را نشان می‌دهد. این تابع با رابطهٔ زیر بیان می‌شود:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

این تابع را پیاده‌سازی و آزمایش کنید.

13- تابع ترکیب c(n,k) را می‌توان با استفاده از رابطهٔ زیر بیان نمود:

$$C(n,k) = \frac{P(n,k)}{k!}$$

با استفاده از این رابطه، برنامهٔ مسالهٔ 13-5 را بازنویسی کرده و آزمایش کنید.

14- روش موثرتر برای محاسبهٔ c(n,k) رابطهٔ زیر است:

$$C(n,k) = ((((((n/1)(n-1))/2)(n-2))/3)...(n-k+2))/(k-1))(n-k+1))/k$$

این تابع به طور متناوب ضرب و تقسیم می‌شود، هر دفعه با یک واحد کم‌تر از مقدار فعلی n ضرب می‌شود و بر یک واحد بیشتر از مقدار قبلی با شروع از یک، تقسیم می‌شود. با استفاده از رابطهٔ فوق، تابع مسالهٔ 13-5 را بازنویسی و آزمایش کنید.

راهنمایی: مانند مسالهٔ 12-5 از حلقهٔ for استفاده کنید.

15- مثلث خیام یک آرایهٔ سه‌گوش از اعداد به شکل زیر است:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
  1   7  21  35  35  21   7   1
1   8  28  56  70  56  28   8   1
```

هر عدد در مثلث خیام یکی از ترکیبات c(n,k) است (به مساله 13-5 نگاه کنید). اگر ردیف‌ها وستون‌ها را با شروع از 0 شمارش کنیم، عدد واقع در ردیف n و ستون k برابر با c(n,k) است. به عنوان مثال عدد15 = c(6,2) در ردیف شماره 6 و ستون شماره 2 است. برنامه‌ای بنویسید که با استفاده از تابع مساله 14-5 یک مثلث خیام دوازده ردیفی چاپ کند.

16- تابع digit() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید:

```
int digit(int n, int k);
```

این تابع رقم kام عدد صحیح n را برمی‌گرداند. برای مثال اگر n عدد صحیح 29415 باشد، تابع digit(n,0) رقم 5 را بازمی‌گرداند و فراخوانی digit(n,2) رقم 4 را برمی‌گرداند. توجه کنید که رقم‌ها از راست به چپ و با شروع از 0 شمارش می‌شوند.

17- تابعی را نوشته و آزمایش کنید که الگوریتم اقلیدس را برای بازگرداندن بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح مثبت داده شده به کار می‌گیرد (به مسألهٔ 11 از فصل چهارم نگاه کنید)

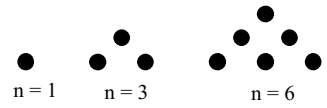
18- تابعی را نوشته و آزمایش کنید که با استفاده از تابع بزرگ‌ترین مقسوم‌علیه مشترک (مسألهٔ 17) کوچک‌ترین مضرب مشترک دو عدد صحیح مثبت را برگرداند.

19- تابعی به نام power() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید:

```
double power(double x, int p);
```

این تابع x را به توان p می‌رساند که p می‌تواند هر عدد صحیحی باشد. از الگوریتمی استفاده کنید که برای محاسبهٔ x^{20} مقدار x را 20 مرتبه در خودش ضرب می‌کند.

20- یونانی‌های باستان اعداد را به صورت هندسی طبقه‌بندی می‌کردند. برای مثال به یک عدد مثلثی می‌گفتند اگر آن عدد می‌توانست با ریگ‌ها در یک تقارن مثلثی چیده شود. ده عدد مثلثی اول اعداد 0 و 1 و 3 و 6 و 10 و15 و 21 و 28 و 36 و 45 هستند. تابع بولی زیر را نوشته و آزمایش کنید. اگر n یک عدد مثلثی باشد این تابع مقدار 1 را برمی‌گرداند وگرنه 0 برگشت داده می‌شود:




```
int isTriangular(int n);
```

21- تابع issquare() را نوشته و آزمایش کنید. این تابع تشخیص می‌دهد که آیا عدد داده شده یک عدد مربعی است یا خیر:


```
int isSquare(int n);
```

اولین ده عدد مربعی اعداد 0 و 1 و 4 و 9 و 16 و 25 و 36 و 49 و 64 و 81 هستند.

22- تابع ComputeCircle() که مساحت a و محیط c یک دایره با شعاع داده شدهٔ r را برمی‌گرداند، نوشته وامتحان کنید:

```
void computeCircle(float& a, float& c, float r);
```

23- تابع ComputeTriangle() که مساحت a و محیط p از یک مثلث با اضلاع به طول a و b و c را محاسبه می‌نماید، نوشته و آزمایش کنید:

```
void computeTriangle(float& a, float& p, float a, float b, float c);
```

24- تابع computeSphere() که حجم v و مساحت سطح s را برای یک کره با شعاع داده شدهٔ r برمی‌گرداند، نوشته و آزمایش کنید:

```
void ComputeSphere(float& v, float& s, float r);
```