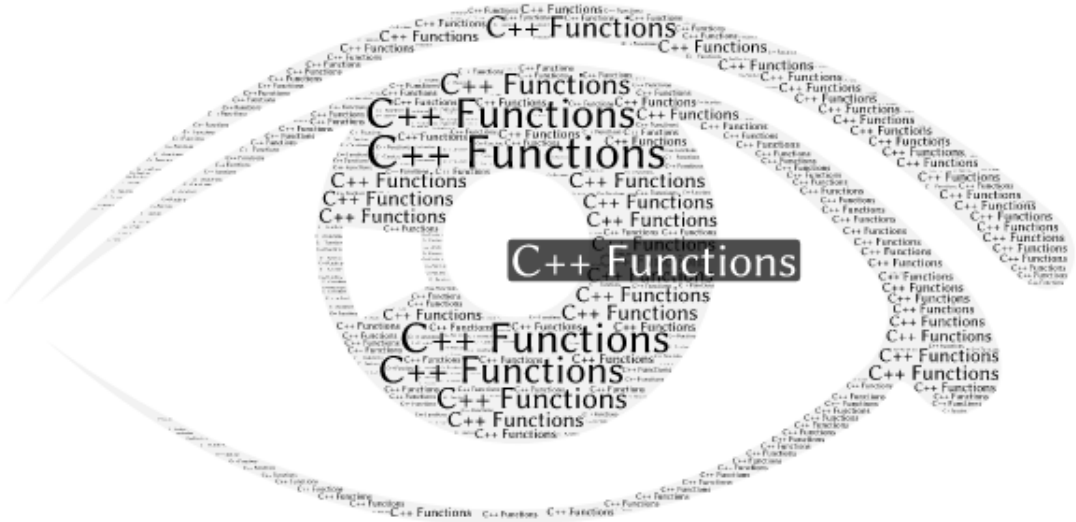
مبانی کامپیوتر و برنامه سازی

اسلاید دوازدهم

«برنامه نویسی مقدماتی به زبان C++»

بخش پنجم : توابع (بخش اول)



درس

مبانی کامپیوتر

۱۲

محمد سعید صفایی صادق

(استفاده از اسلایدها صرفا برای دانشجویان مجاز می باشد!)

۱۴۰۲

www.SaeidSafaei.ir

مقدمه توابع

تابع

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 زیر برنامه‌ها «تابع» نامیده می‌شوند.

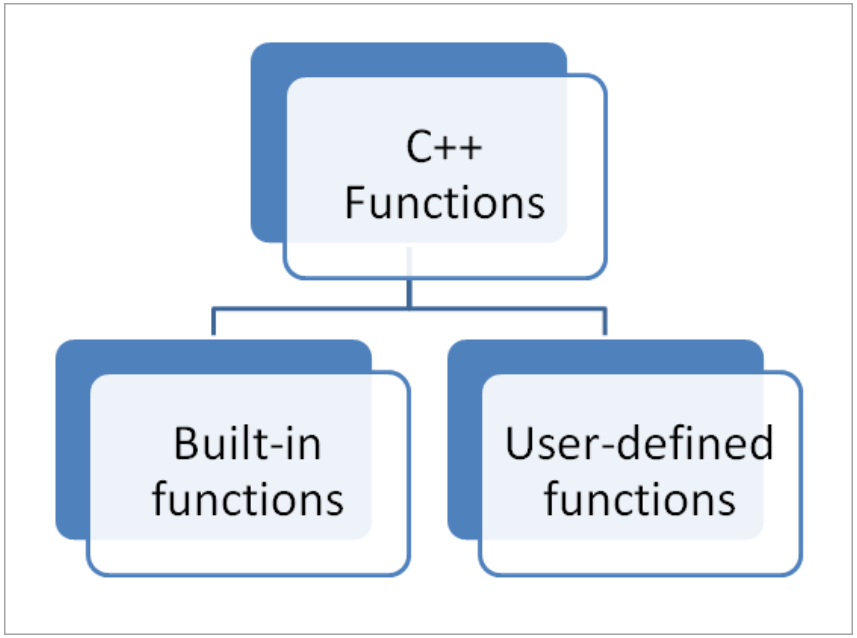
توابع را می‌توان به طور جداگانه کامپایل و آزمایش نمود و در برنامه‌های مختلف دوباره از آن‌ها استفاده کرد.

تابع

بسته به اینکه توابع توسط برنامه نویس نوشته شده باشند یا از قبل موجود باشند، به دو دسته تقسیم می شوند:

۱- توابع کتابخانه ای

۲- توابع نوشته شده توسط کاربر



تابع

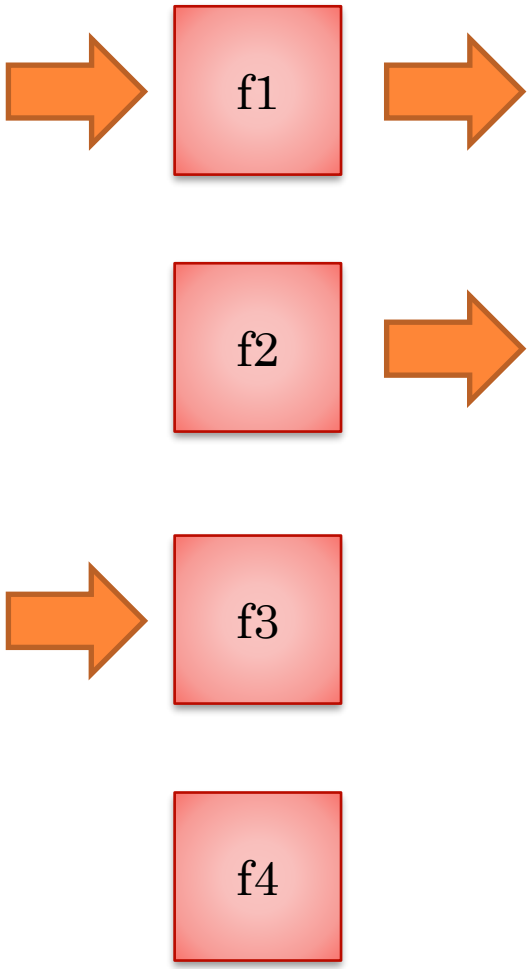
در حالت کلی دو نوع تابع وجود در برنامه c++ دارد.

یک نوع از توابع از قبل به صورت برنامه نویسی شده تهیه شده و قابل دسترس هستند و باید جهت استفاده فراخوانی شوند و توابع دوم، محصول عملکرد کاربر است.

توابع دسته اول را **توابع کتابخانه ای** و توابع قسمت دوم را **توابع نوشته شده** نامند.

اولین و مهمترین تابع از توابع c++ تابع اصلی یا همان تابع main() که اجرای برنامه منوط به وجود آن بوده و در سرخط‌های برنامه نوشته می‌شود.

انواع تابع از نظر ورودی و خروجی



ورودی / خروجی	ورودی دارد	ورودی ندارد void
خروجی دارد	گروه ۱	گروه ۲
خروجی ندارد void	گروه ۳	گروه ۴

توضیحات تکمیلی (صفحه ۴۱)

توابع کتابخانه ای

۱- توابع کتابخانه ای

«کتابخانهٔ C++ استاندارد» مجموعه‌ای است که شامل توابع از پیش تعریف شده و سایر عناصر برنامه است.

تابع‌های کتابخانه تابع‌های داخلی زبان برنامه‌نویسی C ++ هستند.

برنامه‌نویس می‌تواند با فراخوانی مستقیم تابع از این تابع‌های کتابخانه استفاده کند و نیازی نیست که آن‌ها را خودش بنویسد.

این توابع و عناصر از طریق «سرفایل‌ها» قابل دستیابی‌اند.

قبلا برخی از آن‌ها را استفاده کرده‌ایم:تابع sqrt()، pow() که در cmath یا math.h تعریف شده است و... .

مثال توابع کتابخانه ای

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double number, squareRoot;
    cout << "Enter a number: ";
    cin >> number;

    // sqrt() is a library function to calculate square root
    squareRoot = sqrt(number);
    cout << "Square root of " << number << " = " << squareRoot;
    return 0;
}
```

```
Enter a number: 26
Square root of 26 = 5.09902
```

۱- توابع کتابخانه ای

در مثال فوق، تابع کتابخانه sqrt() برای محاسبه ریشه مربع یک عدد استفاده می‌شود.

به کد include <cmath# > در برنامه فوق توجه کنید.

در این کد cmath یک فایل «هدر» است.

تعریف تابع sqrt() یعنی بدنه تابع در فایل هدر cmath قرار دارد.

زمانی که فایل cmath را برنامه خود بگنجانید، می‌توانید از همه تابع‌های تعریف شده در آن استفاده کنید.

این کار «فراخوانی تابع» یا «احضار تابع» گفته می‌شود.

بنابراین وقتی کد sqrt(x) اجرا شود، تابع sqrt() فراخوانی می‌گردد.

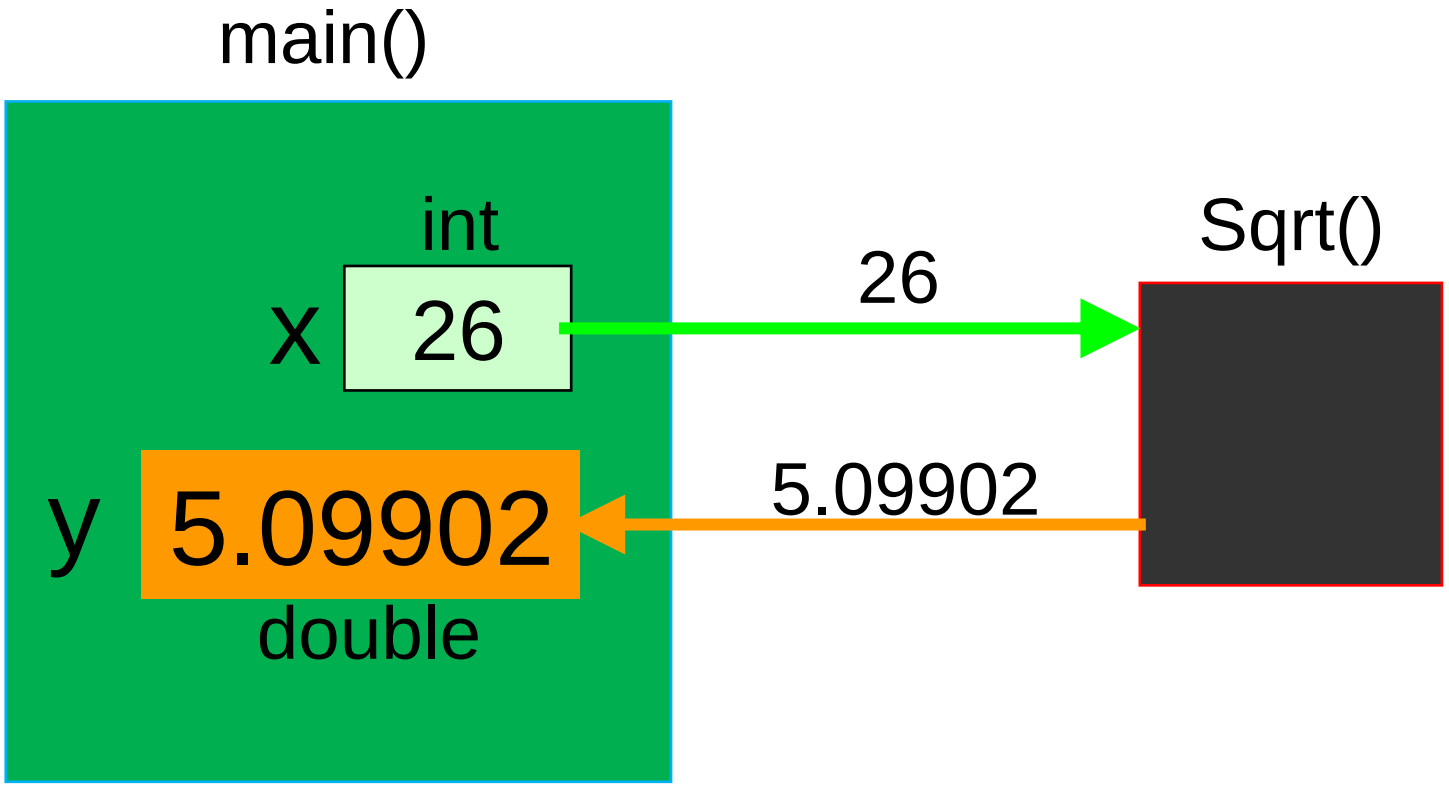
عبارت xدرون پرانتز «آرگومان» یا «پارامتر واقعی» فراخوانی نامیده می‌شود.

در چنین حالتی می‌گوییم که x توسط «مقدار» به تابع فرستاده می‌شود.

لذا وقتی x=26 است، با اجرای کد sqrt(x) تابع sqrt() فراخوانی شده و مقدار ۲۶ به آن فرستاده می‌شود.

تابع مذکور نیز حاصل **۵/۰۹۹۰۲** را به عنوان پاسخ برمی‌گرداند...

این فرایند در نمودار زیر نشان داده شده است.



چگونه از توابع کتابخانه ای استاندارد استفاده کنیم؟

۱. شروع

۲. برای وجود تابع استاندارد در C++ جستجو می کنیم

۳. اگر تابع استانداردی پیدا شد، اطلاعات زیر را مورد نیاز داریم:

- تعداد و نوع ورودی
- نوع خروجی
- نام تابع
- سرفایلی که تابع در آن تعریف شده است

۴. سرفایل را به برنامه اضافه می کنیم

۵. متناسب با اطلاعات تابع، تابع را در برنامه در محل مورد نظر فراخوانی می کنیم

۶. پایان

توابع کتابخانه ای

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:

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

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.

تابع	شرح	مثال
`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

توابع کتابخانه ای

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

https://en.cppreference.com/w/cpp/header

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 <stdlib.h> به کار ببریم، باید دستور پیش‌پردازندهٔ زیر را به ابتدای فایل برنامهٔ اصلی اضافه کنیم:

```
#include <stdlib.h>
```

مثال:

```
1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4  int main() {
5      int v1,v2,v3;
6      v1 = rand() % 100;          // v1 in the range 0 to 99
7      v2 = rand() % 100 + 1;      // v2 in the range 1 to 100
8      v3 = rand() % 30 + 1985;    // v3 in the range 1985-2014
9      cout<<v1<<" "<<v2<<" "<<v3<<" ";
10 }
```

v2 = rand() % 100 + 1; // v2 in the range 1 to 100

v3 = rand() % 30 + 1985; // v3 in the range 1985-2014

۳۰ تا عدد از خود ۱۹۸۵ بشمار برو جلو

توابع ساخت کاربر

user-defined functions

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.

یک تابع ساخت کاربر دو قسمت دارد: ۱-عنوان ۲- بدنه.

عنوان یک تابع به صورت زیر است:

نوع بازگشتی **نام تابع (فهرست پارامترهای ورودی)**

مثال:

```
int cube(int x)
{
...          بدنه تابع
return ...
}
```

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.

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.

تابع cube()

یک مثال ساده از توابع ساخت کاربر:

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

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

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.

برنامه آزمون:

وقتی یک تابع مورد نیاز را ایجاد کردید، فورا باید آن تابع را با یک برنامهٔ ساده امتحان کنید. چنین برنامه‌ای ***برنامهٔ آزمون*** نامیده می‌شود.

برنامهٔ آزمون یک برنامهٔ موقتی است که باید «سریع و ساده» باشد؛ **یعنی**:

لازم نیست در آن تمام ظرافت‌های برنامه‌نویسی – مثل پیغام‌های خروجی، برچسب‌ها و راهنماهای خوانا – را لحاظ کنید.

تنها هدف این برنامه، امتحان کردن تابع و بررسی صحت کار آن است.

یک برنامهٔ آزمون برای تابع cube()

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

```
1  #include <iostream>
2  using namespace std;
3
4  int cube(int x)
5  {  // returns cube of x:
6     return x*x*x;
7  }
8  int main()
9  {  // tests the cube() function:
10    int n=1;
11    while (n != 0)
12    {
13       cin >> n;
14       cout << "\tcube(" << n << ") = " << cube(n) << endl;
15    }
16 }
```

برنامهٔ حاضر اعداد صحیح را از ورودی می‌گیرد و مکعب آن‌ها را چاپ می‌کند تا این که کاربر مقدار 0 را وارد کند.

دو روش اعلان تابع (صفحه ۳۰)

یک برنامهٔ آزمون برای تابع cube()

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

```
1  #include <iostream>
2  using namespace std;
3
4  int cube(int) ;
5  int main()
6  {  // tests the cube() function:
7     int n=1;
8     while (n != 0)
9     {  cin >> n;
10       cout << "\tcube(" << n << ") = " << cube(n) << endl; }}
11 int cube(int x)
12 {  // returns cube of x:
13    return x*x*x;
14 }
```

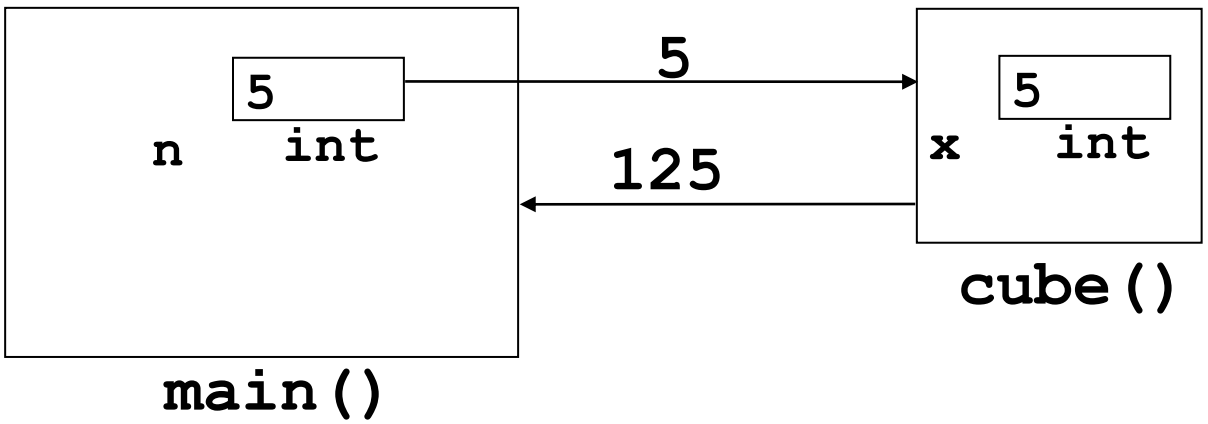
برنامهٔ حاضر اعداد صحیح را از ورودی می‌گیرد و مکعب آن‌ها را چاپ می‌کند تا این که کاربر مقدار 0 را وارد کند.

دو روش اعلان تابع (صفحه ۳۰)

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.

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.

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



یک برنامهٔ آزمون برای تابع max() : تابع زیر دو پارامتر دارد.

این تابع از دو مقدار فرستاده شده به آن، مقدار بزرگ‌تر را برمی‌گرداند:

```
1  #include <iostream>
2  using namespace std;
3  int max(int x, int y) // returns larger of the two given integers
4  { int z;
5     z = (x > y) ? x : y ;
6     return z; }
7  int main()
8  {
9     int m, n;
10    do {
11       cin >> m >> n;
12       cout << "\tmax(" << m << "," << n << ") = "  << max(m,n) << endl;
13    }
14    while (m != 0);
15 }
```

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.

به دو روش میتوان توابع را تعریف نمود:

۱-توابع قبل از تابع main() و بعد از سرفایل ها به طور کامل با بدنه مربوطه آورده شوند.

(شما این روش را خوانده اید، مانند مثال هایی که دیدید)

۲-راه دیگری که بیشتر رواج دارد این گونه است که ابتدا تابع *اعلان* شود، سپس متن برنامهٔ اصلیmain() بیاید، پس از آن *تعریف* کامل تابع قرار بگیرد.

اعلان تابع شبیه اعلان متغیرهاست.

اعلان تابع با تعریف تابع تفاوت دارد. اعلان تابع، فقط عنوان تابع است که یک سمیکولن در انتهای آن قرار دارد. تعریف تابع، متن کامل تابع است که هم شامل عنوان است و هم شامل بدنه.

یک متغیر قبل از این 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.

سپس در زمان اجرا به تعریف بدنهٔ تابع نیز احتیاج می‌شود که این بدنه در انتهای برنامه و پس از تابع main() قرار می‌گیرد.

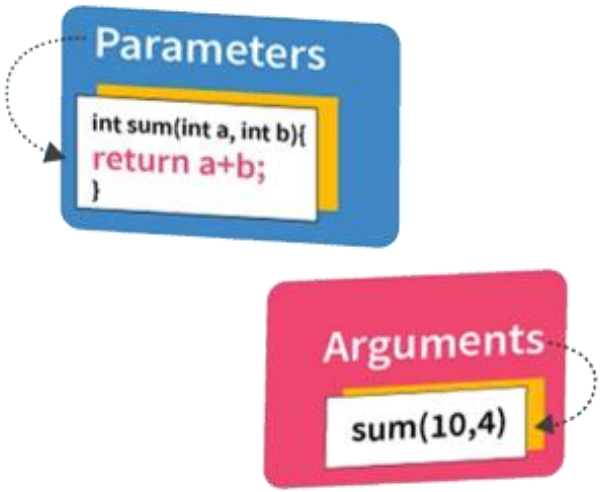
روش ۱	روش ۲
عنوان تابع متن اصلی تابع Main	اعلان تابع; Main عنوان تابع متن اصلی تابع

فرق بین «آرگومان» و «پارامتر» :

پارامترها متغیرهایی هستند که در فهرست پارامتر یک تابع نام برده می‌شوند.

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.



به مثال بعدی توجه کنید

تابع max() با اعلان جدا از تعریف آن

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

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

```
int main()
{    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;}
```

m و n آرگومان های ورودی هستند.

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

متغیر محلی، توابع محلی:

متغیر محلی، 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 برای آن تابع هستند.

متغیر های محلی درون یک تابع، در تابع اصلی تعریف نشده اند.

این متغیرها فقط تا وقتی که تابع در حال کار است وجود دارند.

پارامترهای تابع نیز متغیرهای محلی محسوب می‌شوند.

تابع فاکتوریل: 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)
{
   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 و متغیر **f** نیز محلی است زیرا درون بدنهٔ تابع اعلان شده است.

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.

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.

مثال تابع Void :

```
1  #include <iostream>
2  using namespace std;
3  void sayhi()
4  {
5      cout<<"Hi... "<<endl;
6      return;
7  }
8  int main()
9  {
10     int n;
11     cin >> n ;
12     do {
13         sayhi();
14         n--;
15     }
16     while (n != 0);
17     return 0;
18 }
```

```
5
Hi...
Hi...
Hi...
Hi...
Hi...


...Program finished with exit code 0
Press ENTER to exit console.
```

نکته: برخی کامپایلرها تابع void را پشتیبانی نمیکند از جمله DEV لذا برای تست گرفتن از این نوع توابع میتوانید از کامپایلر های آنلاین استفاده کنید.

https://www.onlinegdb.com/online_c++_compiler

تابع main() :

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.

تابع main() :

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.

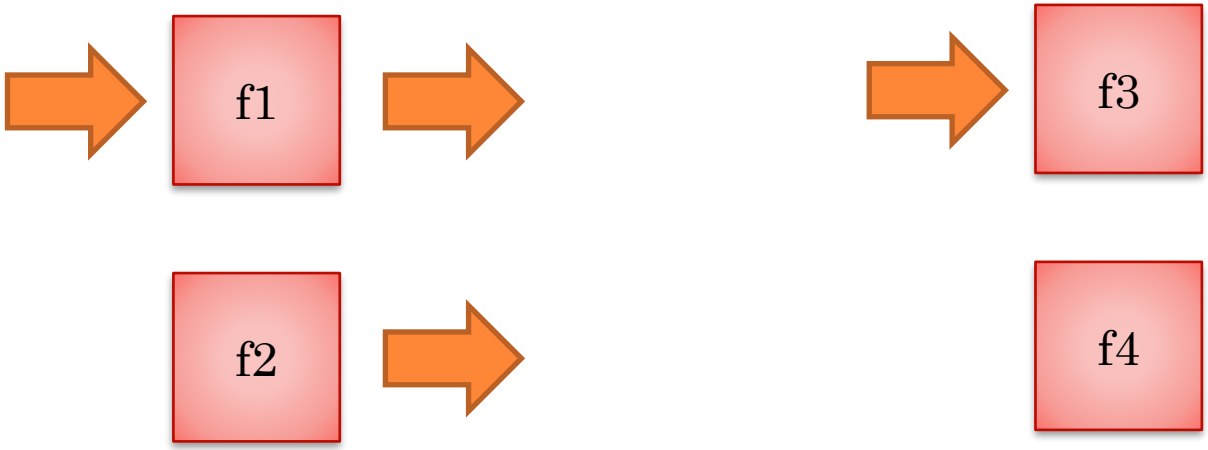
تابع main() :

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;
}
```

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

توابع ساخت کاربر



ورودی / خروجی	ورودی دارد	ورودی ندارد void

خروجی دارد	dType fName1 (dType){
	…
	return …
	}
		dType fName2
		(){
		…
		return …
		}

خروجی ندارد
void
	void fName3 (dType){
	…
	}
		void fName4 (){
		…
		}

پارامتر پیش فرض:

در C++ میتوان برای یک پارامتر مقدار پیشفرضی را تعیین کرد که هنگام فراخوانی در صورت ندادن آرگومان مربوط به آن پارامتر با مقدار پیش‌فرض مقدار بگیرد. این مقدار پیشفرض برای ساده کردن فراخوانی توابع پیچیده استفاده می شود.

نکته اول : اگر پارامتری مقدار پیشفرض داشته باشد تمام پارامتر های سمت راست آن نیز باید مقدار پیش فرض داشته باشند.

بنابراین در مثال بعد نمیشود که *num* مقدار پیشفرض داشته باشد ولی *ch* نداشته باشد.

پارامتر پیش فرض:

```
void myFunc(int num=1,char ch='A')
{
        .
        .
        .
}
```

```
myFunc(2,'B'); // num=2 & ch='B'
myFunc(2);     // num=2 & ch='A'
myFunc();      // num=1 & ch='A'
```

پارامتر پیش فرض:

نکته دوم : هنگام صدا کردن تابع ابتدا پارامتر هایی که مقدار پیش‌فرض ندارند داده میشوند

سپس میرسیم به ابتدای پارامتر هایی که مقدار پیشفرض دارند

سپس تا جایی دلخواه از چپ میتوانیم آرگومان مورد نظر را برای پارامتر های داری مقدار پیش‌فرض بفرستیم و دیگر ادامه ندهیم و در اینصورت پارامتر های باقی مانده با مقدار پیشفرض مقداردهی میشوند.

```
void func(T1 P1,T2 P2,T3 P3=D3,T4 P4=D4,T5 P5=D5){}
...
func(ARG1,ARG2);                  // P1=ARG1 & P2=ARG2 & P3=D3   & P4=D4   & P5=D5
func(ARG1,ARG2,ARG3);             // P1=ARG1 & P2=ARG2 & P3=ARG3 & P4=D4   & P5=D5
func(ARG1,ARG2,ARG3,ARG4);        // P1=ARG1 & P2=ARG2 & P3=ARG3 & P4=ARG4 & P5=D5
func(ARG1,ARG2,ARG3,ARG4,ARG5);   // P1=ARG1 & P2=ARG2 & P3=ARG3 & P4=ARG4 & P5=ARG5
```

با تشکر از همراهی شما

محمد سعید صفایی صادق

www.SaeidSafaei.ir