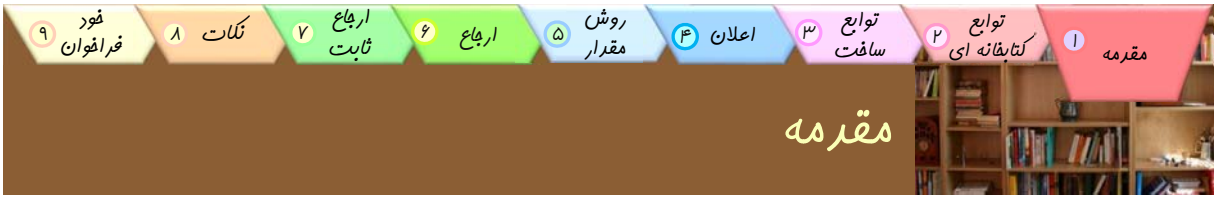
فصل هشتم: «توابع»



برنامه ها: بخش بندی به زیر برنامه‌هایی با نام «تابع»
توابع را می توان به طور جداگانه کامپایل و آزمایش نمود و در برنامه های مختلف دوباره از آنها استفاده کرد.
این بخش بندی در موفقیت یک نرم افزار شی گرا بسیار موثر است.

«کتابخانه 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.

انواع تابع:
۱) ورودی دارند و خروجی هم دارند مثل **y= sqrt (x);**
۲) ورودی ندارند وخروجی دارند. مثل **ch=getch( );**
۳) ورودی دارند و خروجی ندارند. مثال **circle(x, y, r);**
۴) ورودی ندارند خروجی ندارند. مثل **cleardevice( );** صفحه گرافیکی را پاک می‌کند

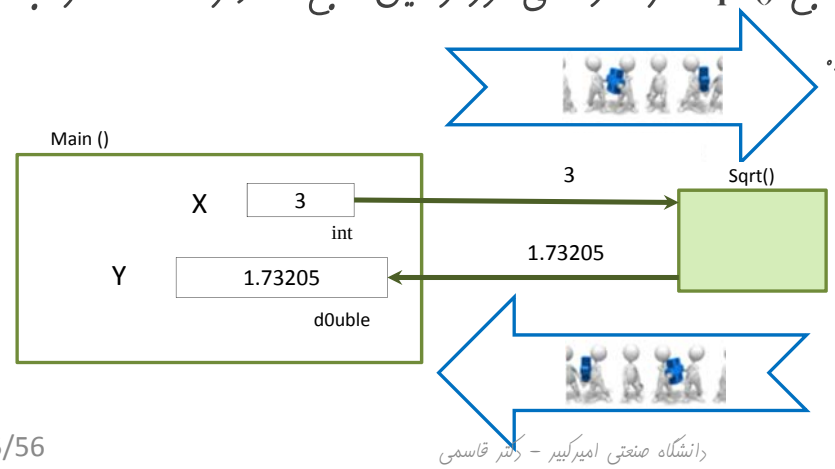
توابع: افزایش خوانایی برنامه
افزایش قابلیت اشکال‌زدایی
ماژول‌ها باید به تنهایی معنا دهند.

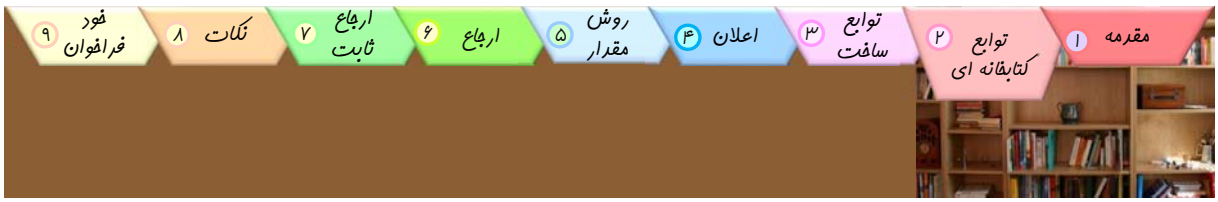
sqrt(x): اجرا ⟸ فراخوانی تابع **sqrt()**

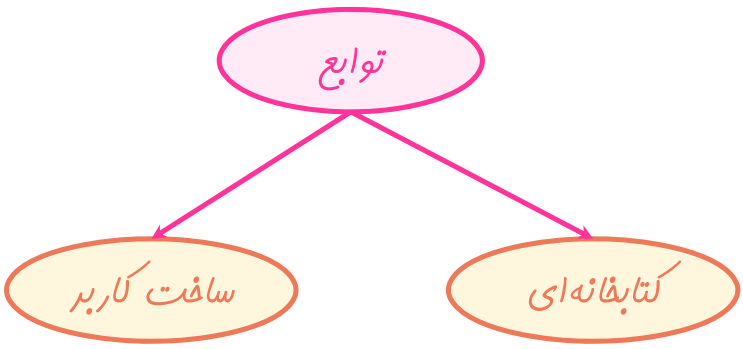
x: «آرگومان» یا «پارامتر واقعی» فراخوانی

x توسط «مقدار» به تابع فرستاده می‌شود.

مقدار **x** (برابر ۳) به تابع **sqrt()** فرستاده می‌شود و این تابع مقدار **1.73205** را به تابع **main()** برمی‌گرداند.







بعضی از توابع تعریف شده در سرفایل <cmath>

نوع double

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 -0.41614 را برمی‌گرداند
exp (x)	تابع نمایی x(درپایه 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 (گردشده)	Floor (3.141593) مقدار 3.0 را برمی‌گرداند
log (x)	لگاریتم طبیعی x (درپایه e)	log (2) مقدار 0.693147 را برمی‌گرداند
log10 (x)	لگاریتم عمومی x (درپایه 10)	log10 (x) 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

بعضی از سرفایل‌های کتابخانه C++ استاندارد

سرفایل	شرح
<ctype>	توابعی را برای بررسی کاراکترها تعریف می‌کند
<cfloat>	ثابت‌های مربوط به اعداد ممیز شناور را تعریف می‌کند
<climits>	محدوده 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 استاندارد را تعریف می‌کند
<cstring>	توابعی را برای پردازش رشته‌ها تعریف می‌کند
<ctime>	توابع تاریخ و ساعت را تعریف می‌کند

مثال ۱) تابع جذر sqrt()

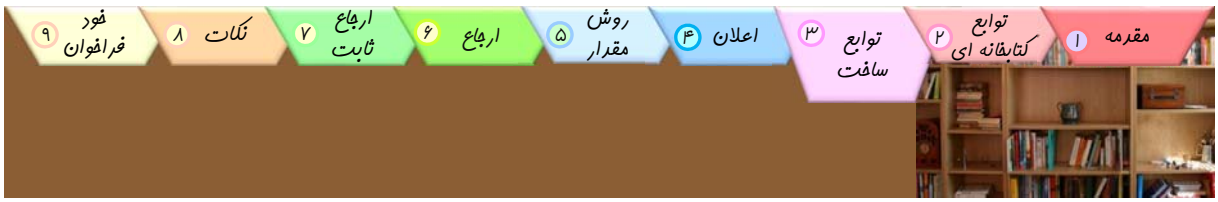
```
#include <cmath>     // defines the sqrt() function
#include <iostream>  // defines the cout object
using namespace std;
int main()
{ //tests the sqrt() function:
  for (int x=0; x < 6; x++)
    cout << "\t" << x << "\t" << sqrt(x) << endl;
}
```

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


مثال ۲) آزمایش یک رابطه مثلثاتی
Sin2x=2SinxCosx

```
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"<< 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
```



عنوان یک تابع :

(فهرست پارامترها) نام نوع بازگشتی

بدنه تابع: یک بلوک کد است که در ادامه عنوان آن می آید.
بدنه شامل دستوراتی است که باید انجام شود تا نتیجه موردنظر به دست آید.

همچنین بدنه شامل دستور **return** است که پاسخ نهایی را به مکان فراخوانی تابع برمی گرداند.

مثال ۳) تابع محاسبه مکعب یک عدد

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

دستور **return** :

۱- اجرای تابع را خاتمه می‌دهد.

۲- مقدار نهایی را به برنامه فراخوان بازمی‌گرداند. **return expression;**

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

int main(): تعریف تابع با نوع بازگشتی از نوع **int** و نام آن **main** است. 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 برنامه ساده امتحان کنید.

برنامه آزمون: هدف آن امتحان کردن تابع و بررسی صحت کار آن است.

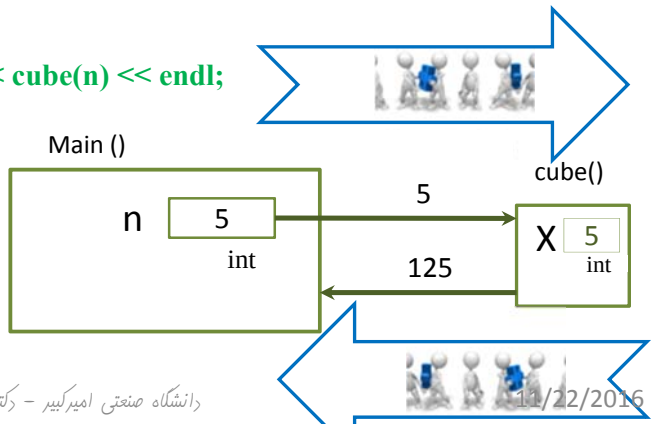
یک برنامه موقتی «سریع و کثیف»

وقتی با استفاده از برنامه آزمون تابع را آزمایش کردید دیگر به آن احتیاجی نیست و می‌توانید برنامه آزمون را دور بریزید.

مثال۴) یک برنامه آزمون برای تابع cube()

```
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
```



نحوه تعریف و استفاده از توابع:

```
# include <iostream>
# include <conio.h>
using namespace std ;
unsigned long long fact (unsigned int n)
{
unsigned long long r=1:
for (unsigned int i=2 ; i<=n ; i++)
{
r*=i ;
}
return r ; برگشت دادن خروجی
}
int main ( )
{
int n ;
cin>>n ;
cout<<n<<"!="<<fact(n) ;
getch ( ) ;
return 0 ;
}
```

مثال ۵) برنامه‌ای بنویسید که دو عدد دریافت کند و با تابعی بزرگترین را تشخیص داده و برگرداند.

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

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

```
int max(int x, int y)
{int z;
  z = (x > y) ? x : y ;
  return z;
}
int main()
{ 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
```


اعلان‌ها و تعاریف تابع

۱- ابتدا تعریف کامل تابع و زیر آن متن برنامه اصلی.

۲- ابتدا تابع اعلان شود، سپس متن برنامه اصلی بیاید، پس از برنامه اصلی تعریف کامل تابع قرار بگیرد.

اعلان تابع: فقط عنوان تابع با یک سمیکولن در انتها (شبیه اعلان متغیرها؛ متغیر را در هر جایی از برنامه می‌توان اعلان کرد اما تابع را باید قبل از برنامه اصلی اعلان نمود.) در اعلان تابع فقط بیان می‌شود که نوع بازگشتی تابع چیست و نوع پارامترهای تابع چیست.

تعریف تابع: متن کامل تابع که هم شامل عنوان است و هم شامل بدنه.

فرق بین «آرگومان» و «پارامتر»

پارامترها: متغیرهایی که در فهرست پارامتر یک تابع نام برده می‌شوند و متغیرهای محلی برای تابع محسوب می‌شوند؛ یعنی فقط در طول اجرای تابع وجود دارند. در مثال قبلی **x** و **y** پارامترهای تابع **max()** هستند.

آرگومان‌ها: متغیرهایی که از برنامه اصلی به تابع فرستاده می‌شوند. در مثال قبلی **n** و **m** آرگومان‌های تابع **max()** هستند.

وقتی یک تابع فراخوانی می‌شود، مقدار آرگومان‌ها درون پارامترهای تابع قرار می‌گیرد تا تابع پردازش را شروع کند.

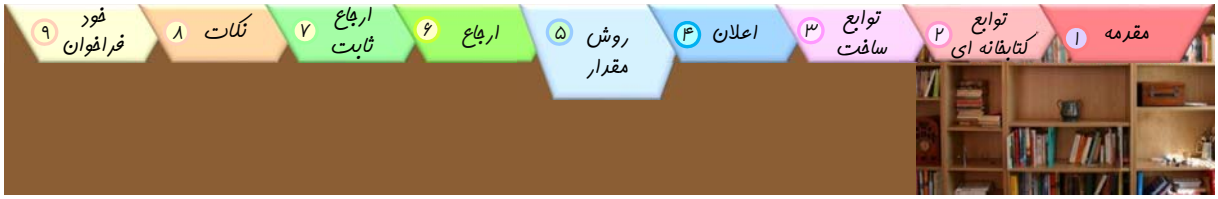
ارسال آرگومان‌ها «به روش مقدار»

مقدار آرگومان‌ها جایگزین پارامترهای متناظرشان می‌شوند.

می‌توان به جای آرگومان‌ها، یک مقدار ثابت را به تابع فرستاد (مثل **max(22,44)**) یا می‌توان یک عبارت 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).

یعنی ابتدا مقدار متغیری که در فراخوانی تابع ذکر شده برآورد می‌شود و سپس این مقدار به پارامترهای محلی تابع فرستاده می‌شود.

مثال: در فراخوانی **cube(n)** ابتدا مقدار **n** برآورد شده و سپس این مقدار به متغیر محلی **x** در تابع فرستاده می‌شود و پس از آن تابع کار خویش را آغاز می‌کند. در طی اجرای تابع ممکن است مقدار **x** تغییر کند اما چون **x** محلی است هیچ تغییری روی مقدار **n** نمی‌گذارد. پس خود **n** به تابع نمی‌رود بلکه مقدار آن درون تابع کپی می‌شود. تغییر دادن این مقدار کپی شده درون تابع هیچ تاثیری بر **n** اصلی ندارد.

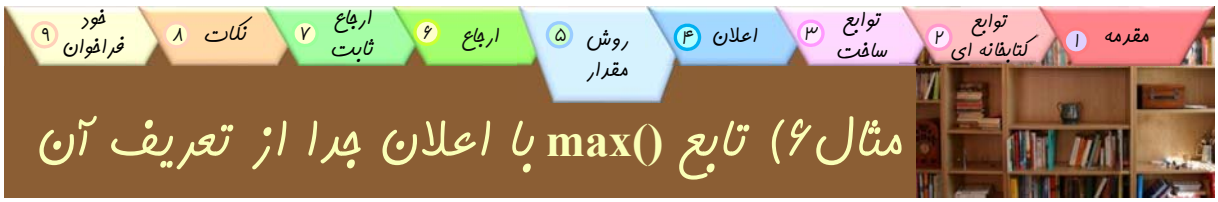


به این ترتیب تابع می‌تواند مقدار n را بخواند اما نمی‌تواند مقدار n را تغییر دهد. به همین دلیل به n یک پارامتر «فقط خواندنی» می‌گویند.

در ارسال به وسیله مقدار: هنگام فراخوانی تابع می‌توان از عبارات استفاده کرد.

مثال: تابع cube() را می‌توان به صورت cube(2*x-3) فراخوانی کرد یا به شکل cube(2*sqrt(x)-cube(3)) فراخوانی نمود. در هریک 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 برنامه اصلی از تغییرات ناخواسته در توابع مصون بمانند.



مثال۶) تابع max() با اعلان جدا از تعریف آن

```
int max(int,int);
int main()
{  // 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;
}
```

متغیرهای محلی، توابع محلی

متغیرمحلی: اعلان در داخل یک بلوک ⟸ قابل دستیابی فقط در همان بلوک

چون بدنه تابع، خودش یک بلوک است پس متغیرهای اعلان شده در یک تابع متغیرهای محلی برای آن تابع هستند. این متغیرها فقط تا وقتی که تابع در حال کار است وجود دارند. پارامترهای تابع نیز متغیرهای محلی محسوب می‌شوند.

* کلیه توابع و متغیرهای تعریف شده در حوزه‌ی آن‌ها از هم مستقل بوده و اولویت همواره با متغیر محلی **(local)** است.

مثال ۷) تابعی بنویسید که فاکتوریل پارامتر ورودی‌اش را محاسبه نماید.

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

```
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;
}
```

```
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;
}
```

1 2 6 24 120

مثال ۸) تابع جایگشت

این تابع بیان می‌کند که به چند طریق می‌توان k عنصر دلخواه از یک مجموعه n عنصری را کنار یکدیگر قرار داد:

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

```
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);
}
```

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

برنامه آزمون برای تابع perm() :

```
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.

تابع void

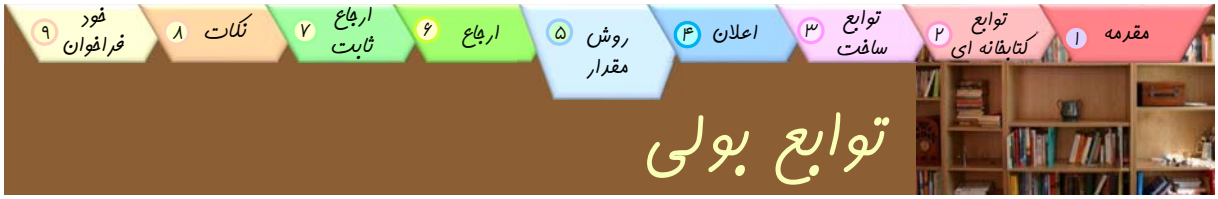
مثال ۹) تابعی که به جای شماره ماه‌ها، نام آنها را می‌نویسد.

لازم نیست یک تابع حتما مقداری را برگرداند. یک تابع void تابعی است که هیچ مقدار بازگشتی ندارد.

```
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)
{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.
```



بررسی شرط با دستورات زیاد ← یک تابع
مخصوصا برای حلقه‌ها

توابع بولی فقط دو مقدار را برمی‌گردانند: **true** یا **false**.
اسم توابع بولی: شکل سوالی؛ همیشه به یک سوال مفروض پاسخ بلی یا خیر می‌دهند

درک آسانتر برنامه و یادآوری وظیفه تابع
در کتابخانه c++ استاندارد: **isLower()** یا **isUpper()**

مثال ۱۰) تابعی که اول بودن اعداد را بررسی می‌کند.

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

```
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
}
```

```
#include <cmath>     // defines the sqrt() function
#include <iostream>  // defines the cout object
using namespace std;
bool isPrime(int);
int main()
{  for (int n=0; n < 80; n++)
     if (isPrime(n)) cout << n << " ";
   cout << endl; }
```

ارسال به طریق ارجاع (آدرس)

ارسال به طریق مقدار: متغیرهای برنامه اصلی از تغییرات ناخواسته در توابع مصون می‌مانند.

گاهی هدف دستکاری محتویات متغیر فرستاده شده به تابع است.

تابع به جای این که یک کپی محلی از آن آرگومان ایجاد کند، **خود آرگومان محلی را به کار میگیرد**. 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 برنامه اصلی اعمال می‌شود.

ارسال یک پارامتر به طریق ارجاع

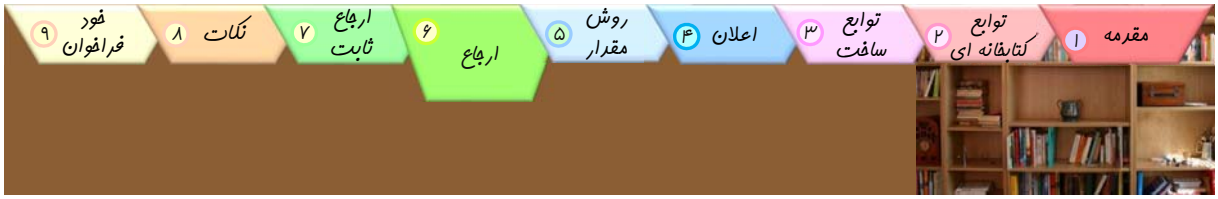
float& x, float& y

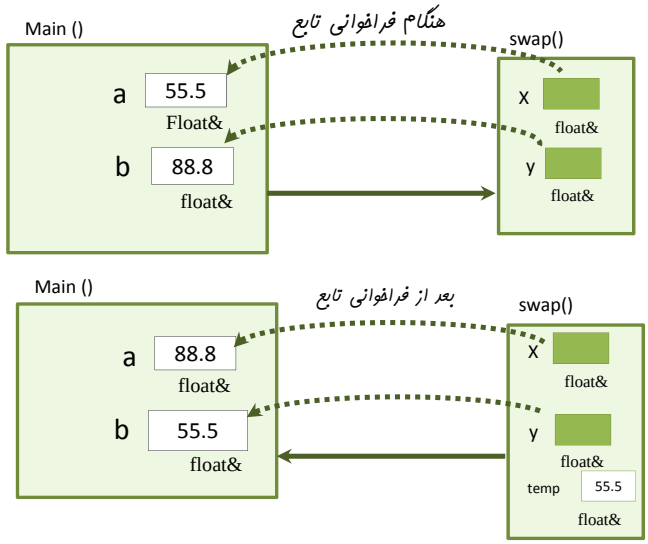
مثال۱۲) تابع swap()

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

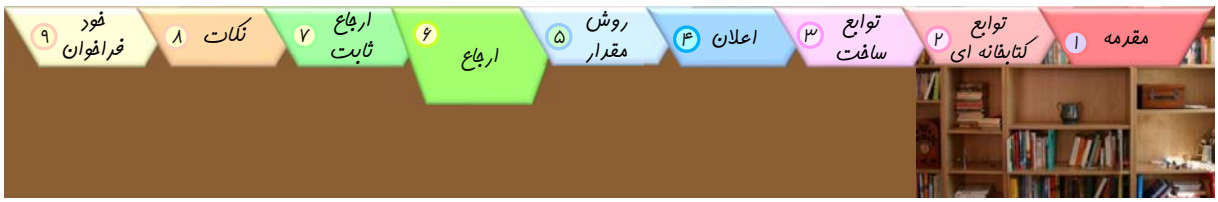
```
#include <iostream>   // defines the cout object
using namespace std;
void swap(float&, float&) ;// exchanges the values of x and y:
int main()
{  // tests the swap() function:
   float a = 55.5, b = 88.8;
   cout << "a = " << a << ", b = " << b << endl;
   swap(a,b);
   cout << "a = " << a << ", b = " << b << endl;

}
```

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





کامپایلر هیچ فرقی بین این دو اعلان نمی‌گذارد (float& x) (float &x)



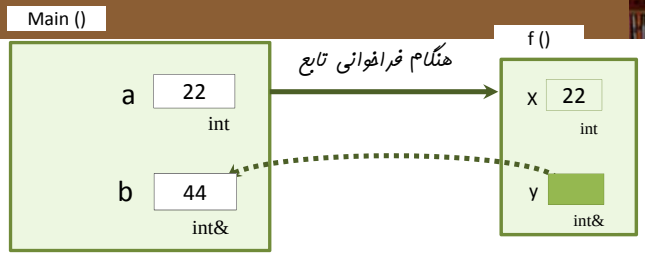
مقایسه ارسال از طریق مقدار با ارسال از طریق ارجاع

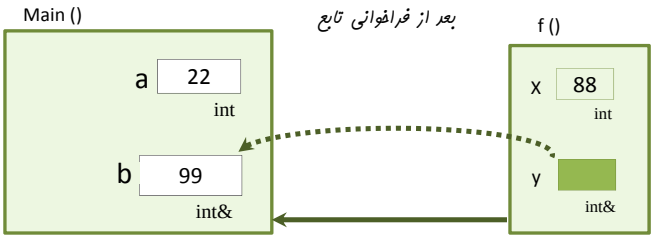
ارسال از طریق مقدار	ارسال از طریق ارجاع
int x ;	int& x ;
پارامتر X یک متغیر محلی است	پارامتر X یک متغیر ارجاع است
X یک کپی از آرگومان است	X مترادف با آرگومان است
تغییر محتویات آرگومان ممکن نیست	می‌تواند محتویات آرگومان را تغییر دهد
آرگومان ارسال شده از طریق مقدار می‌تواند یک ثابت، یک عبارت یا یک متغیر باشد	آرگومان ارسال شده از طریق ارجاع فقط باید یک متغیر باشد
آرگومان فقط خواندنی است	آرگومان خواندنی – نوشتنی است

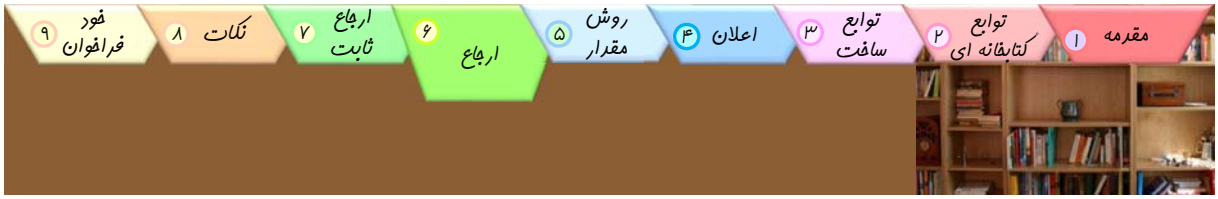
مثال۱۳) ارسال به طریق مقدار و ارسال به طریق ارجاع

```
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
```





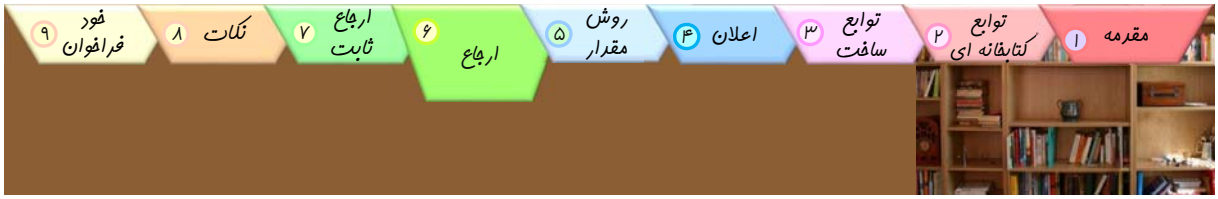
یکی از مواقعی که پارامترهای ارجاع مورد نیاز هستند جایی است که تابع باید بیش از یک مقدار را بازگرداند.

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.

مثال۱۴) تابعی بنویسیدکه دو مقدار محیط و مساحت را برای دایره‌ای با شعاع **r** برگرداند.

```
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;
}
void ComputeCircle (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
```



پارامترهایی که از طریق ارجاع ارسال می‌شوند در ابتدای فهرست پارامترها قرار داده می‌شوند. 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.

ارسال از طریق ارجاع ثابت

مزایای ارسال پارامترها به طریق ارجاع:

۱- تابع می‌تواند روی آرگومان واقعی تغییراتی بدهد

۲- 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 اندازه آرگومان اصلی حافظه اشغال می‌کند.

اگر این شیء حجیم را از طریق ارجاع به تابع ارسال کنیم دیگر نسخه محلی ساخته نمی‌شود و حافظه ای هم هدر نمی‌رود. ایراد: تابع می‌تواند مقدار پارامتر ارجاع را دست کاری کند.

پیشنهاد: ارسال از طریق ارجاع ثابت، مانند ارسال از طریق ارجاع ولی تابع نمی‌تواند محتویات پارامتر ارجاع را دستکاری نماید و فقط اجازه 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.


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

```
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 است.

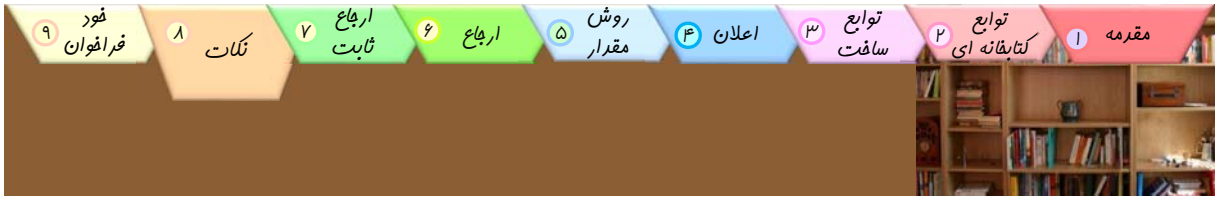
مزیت‌های تعریف و بدنه توابع در فایل‌های جداگانه:

۱- «مخفی سازی اطلاعات»

۲- «بسته بندی نرم افزار»، توابع موردنیاز را می‌توان قبل از برنامه اصلی، جداگانه آزمایش نمود. تولید توابع موردنیاز و تولید برنامه اصلی، همزمان و مستقل از هم پیش می‌رود بدون این که یکی منتظر دیگری بماند.

۳- در هر زمانی به راحتی می‌توان تعریف توابع را عوض کرد بدون این که لازم باشد برنامه اصلی تغییر یابد.

۴- می‌توانید یک بار یک تابع را کامپایل و ذخیره کنید و از آن پس در برنامه‌های مختلفی از همان تابع استفاده ببرید. این کار سرعت تولید نرم‌افزار را افزایش می‌دهد.



max.cpp

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

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);
}
```

توابع بی واسطه

وقتی تابعی درون یک برنامه فراخوانی می‌شود، ابتدا باید مکان فعلی اجرای برنامه 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 در نهایت تابع شروع به کار کند. پس از پایان کار تابع نیز باید همین مسیر به شکل معکوس پیموده شود تا برنامه اصلی ادامه یابد.

انجام همه این کارها هم زمان‌گیر است و هم حافظه اضافی می‌طلبد. در اصطلاح می‌گویند که فراخوانی و اجرای تابع «سربار» دارد.

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 ۱۶) تابع cube() به شکل بی واسطه

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

```
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);
}
```

چندشکلی توابع

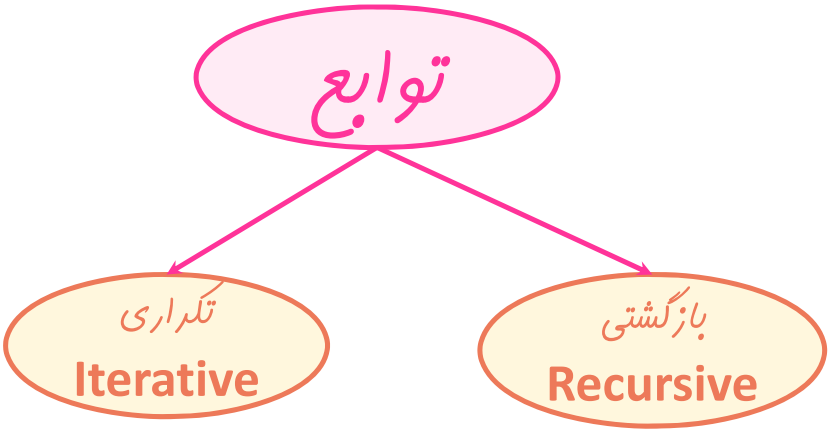
در c++ می توانیم چند تابع داشته باشیم که همگی یک نام دارند. در این حالت می‌گوییم که تابع مذکور، چند شکلی دارد.

شرط: فهرست پارامترهای این توابع با یکدیگر تفاوت داشته باشد. یعنی تعداد پارامترها متفاوت باشد یا دست کم یکی از پارامترهای متناظر هم نوع نباشند.

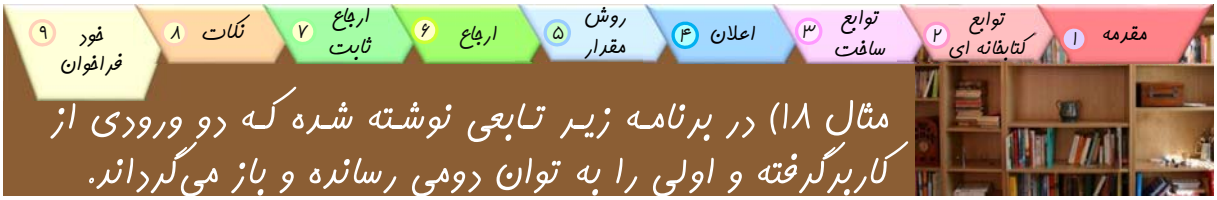
مثال ۱۷) چندشکلی تابع max()

```
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



در برنامه‌نویسی به زبان **C++** این امکان وجود دارد که در بدنه تابع نوشته شده مجددا همان تابع را فراخوانی کرد. به چنین توابعی، توابع خود فراخوان(بازگشتی) گفته می‌شود. استفاده از این روش می‌تواند جایگزینی برای حلقه‌ها باشد اما نیازمند دقت بیشتری در برنامه نویسی‌است.




```
float raised_to_power(float number, int power)
{
  if (power < 0)
  {
     cout << "\nError - can't raise to a negative power\n";
     exit(1);
  }
  else if (power == 0)
     return (1.0);
  else
     return (number * raised_to_power(number, power-1));
}
```

مثال ۱۹) برنامه‌ای بنویسید که با استفاده از تابع خودفراخوان فاکتوریل عدد داده شده را محاسبه کند.

اعداد فاکتوریل 0!,1!,2!,3! و ... با استفاده از رابطه های بازگشتی زیر تعریف می شوند:

$0! = 1 \quad , \quad n! = n(n-1)!$

برای مثال به ازای n=1 در معادله دوم داریم:

$1!= 1((1-1)!)=1(0!)=1(1)=1$

همچنین برای n=2 داریم:

$2!=2((2-1)!)=2(1!)=2(1)=2$

و به ازای n=3 داریم:

$3!=3((3-1)!)=3(2!)=3(2)=6$


```
#include<iostream>
using namespace std;
int fact (int);
int main()
{  long n;
   cout << "Enter a positive integer: ";
   cin >> n;
   cout << n << "!="<< fact(n) << endl;
   return 0;
}
int fact (int n)
{ if (n < 0) return 0;
   if (n<2) return 1;
   return n*fact(n-1);
}
```

```
Enter a positive integer: 5
5!=120
```

Fact (۵)

```
if (۵<=۱)
    return ۱ ;
else
    return ۵*Fact (۵-۱) ;
```

Fact (۴)

```
if (۴<=۱)
    return ۱ ;
else
    return ۴*Fact (۴-۱) ;
```

Fact (۳)

```
if (۳<=۱)
    return ۱ ;
else
    return ۳*Fact (۳-۱) ;
```

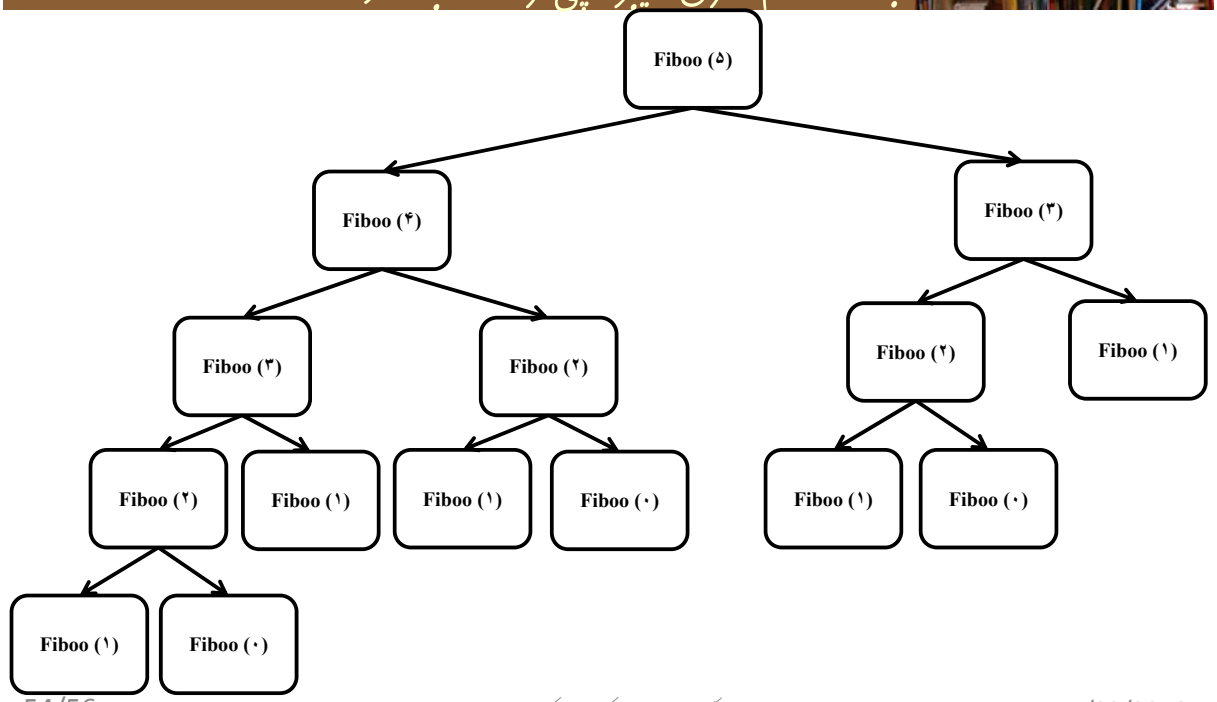
Fact (۲)

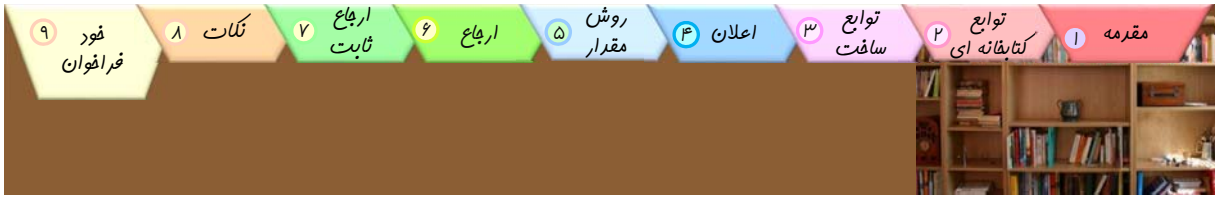
```
if (۲<=۱)
    return ۱ ;
else
    return ۲*Fact (۲-۱) ;
```

Fact (۱)

```
if (۱<=۱)
    return ۱ ;
else
    return ۱*Fact (۱-۱) ;
```


مثال ۲۰) برنامه‌ای بنویسید که با استفاده از تابع خودفراخوان جمله nام سری فیبوناچی را محاسبه کند.






```
#include<iostream>
using namespace std;
int fibo (int);
int main()
{  long n;
   cout << "Enter a positive integer: ";
   cin >> n;
   cout << "Term " <<n<<" is:"<< fibo(n) << endl;
   return 0;
}
int fibo(int n)
{ if (n < 0) return 0;
  if (n<3) return 1;;
   return fibo(n-1) + fibo(n-2);
}
```

Enter a positive integer: 12
Term 12 is: 144

پرسش و پاسخ ؟