```
  }
  cout << "1+2+ ...(except 10)...+20=" <<sum << endl;

  return 0;
}
```

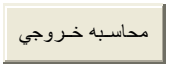
خروجی برنامه به صورت زیر می باشد.

```
1+2+ ...(except 10)...+20=200
```

برنامه عمل جمع را تا رسیدن به عدد ۱۰ ادامه می دهد به محض اینکه **n** برابر ۱۰ می شود دوباره به شرط حلقه منتقل می شود و چون شرط همچنان برقرار است وارد حلقه شده و **n** یک واحد افزایش می یابد و جمع اعداد ادامه می یابد.

توابع ریاضی

مفهوم تابع یکی از مهمترین مفاهیم در ریاضیات و علوم کامپیوتر و نیز سایر علوم می باشد. تابع را می توان به عنوان دستگاهی در نظر گرفت که ورودیهای مجازش را با تغییراتی که وظیفه آن دستگاه می باشد به خروجی متناظر با ورودی تبدیل می کند. معادله خط **y = 2x + 1** را در نظر بگیرید، اگر به جای **f(x**)، **y** را قرار دهیم معادله خط فوق به صورت **f(x) = 2x + 1** در خواهد آمد. در اینجا دستگاه ما تابع **f** خواهد بود که هر ورودی (هر عدد حقیقی) را در ۲ ضرب می کند و سپس یک واحد به آن اضافه می کند. شکل دستگاه به صورت زیر می باشد.



به عنوان مثال :

```
f(0) = 2 * (0) + 1 = 1
f(1) = 2 * (0) + 1 = 3
f(-1) = 2 * (-1) + 1 = -1
```

همانطور که در مثالهای فوق دیدید تابع **f** به هر ورودی تنها یک خروجی را نظیر می کند.

مثال : تابعی بنویسید که شعاع یک دایره به عنوان ورودی باشد و خروجی ، مساحت دایره باشد.

می دانیم که فرمول مساحت دایره $s = 3.14*r^2$ می باشد پس تابع را به صورت زیر تعریف می کنیم:

```
s(r) = 3.14*r^2
```

بعضی از توابع ممکن است بر اساس شرط خاصی خروجی متفاوتی داشته باشند، اینگونه توابع معمولاً به صورت چند ضابطه ای تعریف می شوند. به عنوان مثال تابع قدر مطلق به صورت زیر می باشد:

$$abs(x) = |x| = \begin{cases} x & , x \geq 0 \text{ اگر} \\ -x & , x < 0 \text{ اگر} \end{cases}$$

abs(1)= محاسبه خروجی

به عنوان نمونه :

```
abs(1) = 1
abs(-1) = 1
abs(0) = 0
```

توابع همواره یک مقدار را به عنوان ورودی دریافت نمی کنند، بلکه ممکن است بیش از یک مقدار را به عنوان ورودی دریافت کنند. به عنوان مثال تابع زیر را در نظر بگیرید :

f(x,y) = x + y

تابع فوق هر دو عددی که به عنوان ورودی به آن داده می شوند را با هم جمع کرده و به عنوان خروجی بر می گرداند. به عنوان مثال :

f(1 , 1)=x+y= محاسبه خروجي

```
f(1,2) = 1+2 = 3
f(2,3) = 2+3 = 5
f(1,-1) = 1+(-1) = 0
```

به توابعی مانند تابع فوق توابع دو متغیره گفته می شود و اگر تعداد متغیرها سه تا باشد، تابع سه متغیره و ... و چند متغیره گفته می شود. به متغیرهای ورودی تابع آرگومان نیز گفته می شود.

زبان **C++** برای انجام محاسبات ریاضیاتی ، توابع کاربردی فراوانی را در اختیار ما قرار داده است ، به عنوان مثال فرض کنید که می خواهید جذر یک عدد را بدست آورید، تابعی که زبان **C++** برای اینکار در اختیار ما قرار داده است، تابع **sqrt** می باشد. به عنوان مثال دستور زیر :

```
cout << sqrt (900);
```

عدد ۳۰ را چاپ خواهد کرد. در اینجا عدد ۹۰۰ آرگومان تابع **sqrt** می باشد. برای استفاده از توابع ریاضی در برنامه ملزم به استفاده از دستور:

```
#include <math.h>
```

در ابتدای برنامه می باشیم، چون توابع ریاضی در فایل کتابخانه ای **math.h** قرار دارند. آرگومانهای توابع می توانند شامل اعداد ثابت، متغیرها و یا ترکیبی از آنها باشند؛ به عنوان مثال به برنامه زیر توجه کنید :

```
#include <iostream.h>
#include <math.h>
int main ( )
{
    int x = 30;
    double y = 5;
    cout << sqrt (x+2*y+9)<<endl;
    return 0;
}
```

خروجی برنامه فوق عدد ۷ خواهد بود چون تابع **sqrt** جذر عبارت **30+2*5+9=49** را محاسبه خواهد کرد.

تعدادی از توابع ریاضی مورد استفاده در جدول زیر توضیح داده شده اند.

fabs(x)	این تابع، قدر مطلق **x** را حساب می کند. **fabs(1.2) = 1.2** **fabs(-1.2) = 1.2** **fabs(0) = 0** fabs(-1)= محاسبه خروجي
sqrt(x)	این تابع ، جذر **x** را محاسبه می کند. **sqrt(9) = 3** **sqrt(2) = 1.414214** sqrt(9)= محاسبه خروجي
pow(x,y)	این تابع ، **x** به توان **y** را محاسبه می کند. **pow(2, 5) = 32** **pow(2, 0.5) = 1.414214** pow(2 , 3)= محاسبه خروجي
exp(x)	این تابع ، مقدار عبارت **e** را محاسبه می کند.**e** یک ثابت ریاضی با مقدار تقریبی ۲,۷۱۸۲۸ می باشد. **exp(2) = 7.38906** exp(2)= محاسبه خروجي
log(x)	این تابع ، لگاریتم طبیعی عدد **x** را حساب می کند . لگاریتم طبیعی، لگاریتم بر مبنای **e** می باشد. این تابع بر عکس تابع **exp** عمل می کند. **log(2.71828) = 1** **log(7.389050) = 2** log(2.718)= محاسبه خروجي
log10(x)	این تابع ، لگاریتم بر مبنای ۱۰ عدد **x** را محاسبه می کند.مثلاً (**log10(100** یعنی ۱۰ به توان چه عددی برسد تا حاصل برابر ۱۰۰ شود که جواب ۲ خواهد بود پس **log10(100)=2** **log10(10) = 1** **log10(1000) = 3** log10(10)= محاسبه خروجي

fmod(x,y)	این تابع ، باقیمانده تقسیم دو عدد اعشاری را حساب می کند. **x/y** **fmod(13.657,2.333) = 1.992** fmod(5.1 , 4.2)= محاسبه خروجی
ceil(x)	این تابع ، کوچکترین عدد صحیح بزرگتر مساوی **x** را به عنوان خروجی بر می گرداند. **ceil(9) = 9** **ceil(9.2) = 9** **ceil(-9.8) = -9** ceil(2.4)= محاسبه خروجی
floor(x)	این تابع ، همانند تابع جزء صحیح ریاضیات عمل می کند.یعنی بزرگترین عدد صحیح کوچکتر مساوی **x** را بر می گرداند. **floor(7) = 7** **floor(7.3) = 7** **floor(-7.2) = -8** floor(2.4)= محاسبه خروجی
sin(x)	این تابع ، **sin** عدد **x** را حساب می کند.(**x** باید بر اساس رادیان باشد) **sin(0) = 0** **sin(3.14/2) = 1** sin(1.57( )= محاسبه خروجی
cos(x)	این تابع ، **cos** عدد **x** را محاسبه می کند.(**x** باید بر اساس رادیان باشد) **cos(0) = 1** **cos(3.14/2) = 0** cos(1.57( )= محاسبه خروجی
tan(x)	این تابع ، **tan** زاویه **x** را محاسبه می کند.(**x** باید بر اساس رادیان باشد) **tan(0) = 0** **tan(3.14/4) = 1**

tan(0.785)= محاسبه خروجی	
این تابع ، **arcsin** عدد **x** را محاسبه می کند. توجه داشته باشید که عدد **x** بین ۱- تا ۱ باشد. ضمناً خروجی تابع زاویه ای بر حسب رادیان می باشد.	**asin(x)**
asin(0) = 0 **asin(1) = 1.570796** asin(1)= محاسبه خروجی	
این تابع ، **arccos** عدد **x** را محاسبه می کند. توجه داشته باشید که عدد **x** بین ۱- تا ۱ باشد. ضمناً خروجی تابع زاویه ای بر حسب رادیان می باشد.	**acos(x)**
acos(1) = 0 **acos(0) = 1.570796** acos(1)= محاسبه خروجی	
این تابع ، **arctan** عدد **x** را محاسبه می کند. خروجی تابع زاویه ای بر حسب رادیان می باشد.	**atan(x)**
atan(1) = 0.785398 **atan(0) = 0** **atan(21584.891) = 1.570795** atan(1)= محاسبه خروجی	

نکته : در جدول فوق گفتیم که زاویه ها بر حسب رادیان می باشند. رادیان یکی از واحدهای اندازه گیری زاویه می باشد و با یک تناسب ساده می توان هر زاویه ای که بر حسب درجه می باشد را بر حسب رادیان محاسبه کرد.

$$\frac{R}{3.14} = \frac{D}{180^\circ}$$

در تناسب فوق به جای **D** اندازه زاویه مورد نظر را بنویسید و سپس با حل تناسب، **R** حاوی اندازه درجه بر حسب رادیان می باشد و بر عکس. به مثال های زیر توجه کنید :

$$R \qquad 45^\circ$$

```
R       D
--- = ---    ====> R=0.785
3.14    180°

1.57     D
---- = ---    ====> D=90°
3.14    180°
```

مثال : برنامه ای بنویسید که **sin** و **cos** و**tan** زاویه های زوج **۱** تا **۹۰** درجه را در خروجی به صورت جدول بندی شده تا سه رقم اعشار چاپ نماید.

```
#include <iostream.h>
#include <math.h>

int main( )
{
  float r;
  for (int d=2;d<=90;d+=2)
  {
    r = 3.1415 * d / 180;
    cout<<"sin("<<d<<")="
        <<floor(sin(r)*1000 + 0.5)/1000;
    cout<<"\tcos("<<d<<")="
        <<floor(cos(r)*1000 + 0.5)/1000;
    cout<<"\ttan("<<d<<")="
        <<floor(tan(r)*1000 + 0.5)/1000;
    cout<<endl;
  }
  return 0;
}
```

خروجی برنامه به صورت زیر می باشد:

```
sin(2)=0.035    cos(2)=0.999    tan(2)=0.035
sin(4)=0.07     cos(4)=0.998    tan(4)=0.07
sin(6)=0.105    cos(6)=0.995    tan(6)=0.105
sin(8)=0.139    cos(8)=0.99     tan(8)=0.141
sin(10)=0.174   cos(10)=0.985   tan(10)=0.176
sin(12)=0.208   cos(12)=0.978   tan(12)=0.213
sin(14)=0.242   cos(14)=0.97    tan(14)=0.249

       .               .               .
       .               .               .
       .               .               .
```

```
sin(86)=0.998    cos(86)=0.07     tan(86)=14.292
sin(88)=0.999    cos(88)=0.035    tan(88)=28.599
sin(90)=1        cos(90)=0        tan(90)=21584.891
```

در برنامه فوق توسط فرمول **r = 3.1415 * d / 180** زاویه بر حسب درجه را به رادیان تبدیل کردیم و توسط فرمول **floor(sin(r)*1000 + 0.5)/1000** خروجی را تا سه رقم اعشار محاسبه کردیم. همانطور که می بینید ورودی تابع، علاوه بر متغیر و عدد ثابت خروجی تابع دیگری می باشد. به عنوان مثال **sqrt(pow(2,2))** برابر با **۲** خواهد بود و ترکیب توابع به این شکل کاملاً مجاز می باشد و بر حسب نیاز می توانید از این شیوه استفاده کنید.

تعریف توابع

اکثر برنامه های کاربردی ، خیلی بزرگتر از برنامه هایی می باشند که ما تا اکنون در مباحث قبلی نوشته ایم. تجربه نشان داده است که بهترین راه برای گسترش و نگهداری و ارتقاء برنامه های بزرگ، تقسیم کردن آنها به قطعات کوچکتر می باشد. هر کدام از قطعات راحتتر از برنامه اصلی مدیریت می شوند و اعمال تغییرات و خطایابی در آنها نیز ساده تر می باشد. قطعات مورد نظر ما در زبان **C++** همان توابع می باشند. اگر بیاد داشته باشید در مباحث قبلی گفتیم که **main** نیز یک تابع می باشد و این تابع نقطه ای است که برنامه اجرای دستورات را از آن آغاز می کند. زبان **C++** توابع آماده زیادی را در اختیار ما قرار داده که در فایلهای کتابخانه ای این زبان موجود می باشند، که گوشه ای از آنها را در مبحث قبل دیدید. در برنامه نویسی ممکن است که نیاز داشته باشیم مجموعه دستوراتی را عیناً در چند جای برنامه استفاده کنیم، در چنین حالتی بهتر است این مجموعه دستورات را در تابعی قرار دهیم و تابع را در برنامه چندین باز صدا بزنیم و از تکرار دستورات که تنها حجم برنامه اصلی را زیاد می کنند و از خوانایی آن می کاهند خودداری کنیم.

در زبان **C++** توابع به شیوه زیر تعریف می شوند :

```
نوع داده خروجی    نام تابع    (آرگومانهای تابع)
{
    تعریف متغیرها
    دستورات تابع
}
```

نام تابع از قواعد نام گذاری متغیرها پیروی می کند. برای آشنا شدن با نحوه تعریف توابع و شیوه به کار گیری آنها به برنامه زیر توجه کنید:

```
#include <iostream.h>

long int power2 (int x)
{
  long int y;
```

```
  y=x*x;

  return y;
}

int main ()
{
  for (int i=1;i<=10;i++)
    cout<<power2(i)<<" ";

  return 0;
}
```

خروجی برنامه مربع اعداد ۱ تا ۱۰ می باشد.

```
1 4 9 16 25 36 49 64 81 100
```

تابع **power2(x)** که در این برنامه نوشتیم تقریباً شبیه تابع **pow(x,2)** از توابع کتابخانه ای **C++** عمل می کند. ضابطه ریاضیاتی این تابع $f(x) = x^2$ می باشد. ورودی این تابع اعداد صحیح (**int**) و خروجی آن اعداد بزرگ (**long int**) می باشد. هنگامی که برنامه به تابع **power2(i)** می رسد تابع فراخوانی می شود و مقدار آرگومان **i** را دریافت می کند و در متغیر**x** قرار می دهد. سپس متغیر **y** تعریف می گردد و مقدار **x*x** در **y** قرار می گیرد و سرانجام مقدار **y** به عنوان خروجی تابع برگردانده می شود و توسط دستور **cout** چاپ می گردد. توجه داشته باشید که تابع تغییری در مقدار متغیر **i** ایجاد نمی کند و حلقه **for** تابع را ۱۰ بار فراخوانی می کند، ضمناً متغیرهای **x** و**y** در تابع **main** قابل استفاده نمی باشند و نیز متغیر **i** در تابع **power2** تعریف نشده است.

نکته : توجه داشته باشید که توابع داخل یکدیگر قابل تعریف نمی باشند و جدا از هم باید تعریف گردند.

مثال : تابعی بنویسید که سه عدد را به عنوان ورودی دریافت کرده و بزرگترین آنها را به عنوان خروجی برگرداند. این تابع را در برنامه ای به کار ببرید.

```
#include <iostream.h>

int maximum (int x,int y, int z)
{
  int max=x;

  if (y>max)
     max=y;
  if (z>max)
```

```
    max=z;

  return max;
}

int main ()
{
  int num1,num2,num3;

  cout << "Enter three numbers: ";
  cin >> num1 >> num2 >> num3;
  cout << "Max is :"
       << maximum(num1,num2,num3)<<endl;
  cout << "Max of 5,20,1 is "
       << maximum(5,20,1)<<endl;

  return 0;
}
```

```
Enter three numbers: -5 20 150
Max is :150
Max of 5,20,1 is 20
```

تابع **maximum** دارای سه آرگومان بود. هنگامی که اعداد **num1** و **num2** و **num3** در برنامه از ورودی دریافت می شوند با فراخوانی تابع **maximun(num1,num2,num3)** اعداد آرگومانهای **num1** و **num2** و **num3** به ترتیب در متغیر های **x** و **y** و **z** قرار می گیرند و مقادیر توسط تابع مقایسه می شوند و نهایتاً بزرگترین عدد در متغیر **max** قرار می گیرد که توسط دستور **return max;** به عنوان خروجی تابع برگردانده می شود. سپس دستور **cout** خروجی تابع را بر روی صفحه نمایش چاپ می کند.

پیش تعریف توابع

تا به حال توابع مورد استفاده در برنامه هایمان را قبل از اولین فراخوانی آنها تعریف کردیم و این فراخوانی معمولا در تابع **main** بود. لذا تابع **main** را به عنوان آخرین تابع در برنامه نوشتیم. اگر بخواهید که تابع **main** را قبل از هر تابع دیگری در برنامه بنویسید. هنگام اجرای برنامه یک پیغام خطا دریافت خواهید کرد. دلیل وقوع خطا این است که هنگامی که تابعی فراخوانی می شود باید قبلا تعریف شده باشد، مانند شیوه ای که ما در برنامه های قبلی استفاده کردیم.

یک راه چاره برای اجتناب از نوشتن کد همه توابع قبل از استفاده آنها در تابع **main** یا سایر توابع وجود دارد. این راهکار پیش تعریف توابع می باشد. پیش تعریف تابع به صورت زیر می باشد:

```
نوع داده خروجی    نام تابع    ( نوع آرگومانهای تابع ) ;
```

توجه داشته باشید که پیش تعریف تابع شامل دستورات تابع نمی شود و تنها شامل نوع داده خروجی ، نام تابع و نوع آرگومانها می باشد و در پایان نیاز به علامت (;) دارد. به عنوان مثال پیش تعریف تابع **power2** در مبحث قبلی به صورت زیر می باشد:

```
long int power2( int );
```

ویا پیش تعریف تابع **maximum** به صورت زیر است :

```
int maximum( int, int, int );
```

در اینجا برنامه تابع **maximum** در مبحث قبلی را با روش پیش تعریف تابع باز نویسی می کنیم :

```
#include <iostream.h>

int maximum (int ,int,int);

int main ()
{
  int num1,num2,num3;

  cout << "Enter three numbers: ";
  cin >> num1>>num2>>num3;
  cout << "Max is :"
       << maximum(num1,num2,num3)<<endl;
  cout << "Max of 5,20,1 is "
       << maximum(5,20,1)<<endl;

  return 0;
}

int maximum (int x,int y, int z)
{
  int max=x;

  if (y>max)
      max=y;
```

```
  if (z>max)
      max=z;

  return max;
}
```

```
Enter three numbers: -5 20 150
Max is :150
Max of 5,20,1 is 20
```

همانطور که در برنامه می بینید، تابع **main** قبل از تابع **maximum** نوشته شده است و این امکانی است که پیش تعریف تابع **maximum** به ما داده است.

نکته: در بعضی از برنامه ها، نیاز پیدا می کنیم که دو تابع یکدیگر را فراخوانی کنند. در چنین حالتی ملزم به استفاده از پیش تعریف تابع می باشیم. اما به عنوان یک توصیه برنامه نویسی همواره از پیش تعریف توابع استفاده کنید ، حتی اگر ملزم به استفاده از آن نبودید.

توابع بازگشتی

برنامه هایی که تا کنون نوشتیم یک تابع، تابع دیگری را فراخوانی می کرد. در برنامه نویسی ممکن است نیاز پیدا کنیم که تابعی خودش را به صورت مستقیم یا غیر مستقیم فراخوانی کند. به چنین توابعی، توابع بازگشتی گفته می شود . ابتدا از دید ریاضیاتی توابع بازگشتی را بررسی می کنیم. دنباله اعداد زیر را در نظر بگیرید :

2 , 5 , 11 , 23 , ...

جمله پنجم دنباله اعداد فوق چه عددی می باشد؟ حدس شما چیست؟ اگر کمی دقت کنید متوجه خواهید شد که هر جمله از دنباله فوق برابر است با دو برابر جمله قبلی بعلاوه یک. پس جمله پنجم برابر است با 2*23+1=47 دنباله فوق را توسط فرمول زیر نیز می توان مشخص کرد :

$$d_1 = 2$$
$$d_n = 2*d_{n-1}+1$$

همانطور که متوجه شده اید در این دنباله هر جمله به جملات قبلی خود وابسته است و برای بدست آوردن آن نیاز به بازگشت روی جملات قبلی داریم تا اینکه سرانجام به جمله اول که عدد ۲ می باشد برسیم. فرمول فوق را به صورت تابعی زیر بازنویسی می کنیم :

$$d(1) = 2$$
$$d(n) = 2*d(n-1)+1$$

همانطور که در تابع فوق می بینید یک حالت پایه وجود دارد که همان **d(1)=2** می باشد و یکه حالت بازگشتی که تابع با یک واحد کمتر دوباره فراخوانی می شود **d(n) = 2*d(n-1)+1** . توابع بازگشتی به طور کلی دارای یک یا چند حالت پایه و یک بخش بازگشتی می باشند. که معمولاً در بخش بازگشتی تابع با مقداری کمتر مجدداً فراخوانی می شود.
تابع بازگشتی فوق به زبان **C++** به صورت زیر می باشد :

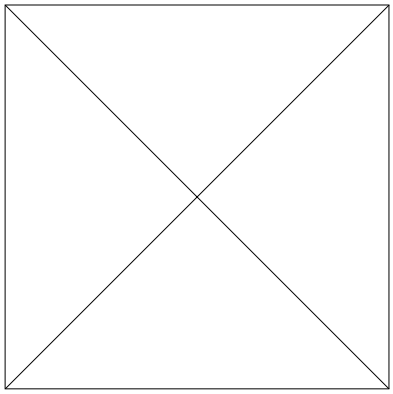
```
long int d(long int n)
{
   if (n == 1)
      return 2;
   else
      return 2*d(n-1)+1;
}
```

در زیر برنامه ای می نویسیم تا با استفاده از تابع فوق ۲۰ جمله اول دنباله مذکور را نمایش دهد.

```
#include <iostream.h>

long int d(long int);
int main( )
{
   for (int i=1;i<=20;i++)
     {
      cout<<d(i)<<"\t";
      if (i%5==0) cout<<endl;
     }

   return 0;
}

long int d(long int n)
{
   if (n == 1)
      return 2;
   else
      return 2*d(n-1)+1;
}
```

```
2       5       11      23      47
95      191     383     767     1535
3071    6143    12287   24575   49151
98303   196607  393215  786431  1572863
```

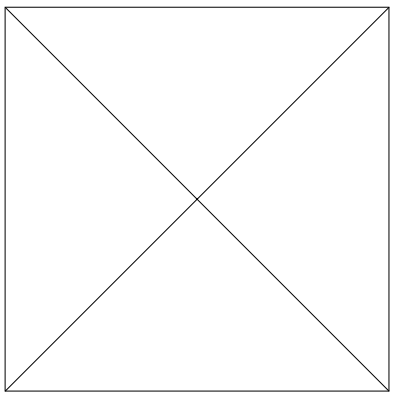
به عنوان مثال برنامه فوق جمله پنجم را به شیوه زیر محاسبه می کند :



تابع، بازگشت را تا رسیدن به حالت پایه ادامه می دهد و به محض رسیدن به حالت پایه محاسبات بازگشتی پی در پی موجب رسیدن به جواب مورد نظر می شود.

مسئله برجهای هانوی (Towers of Hanoi)

هر برنامه نویسی باید به نحوی با مسائل کلاسیک دست وپنجه نرم کرده باشد . یکی از معروفترین مسائل کلاسیک ، مسئله برجهای هانوی می باشد. طبق افسانه ای در معبدی در شرق دور، کاهنان معبدی تعدادی دیسک را از یک ستون به ستون دیگر جا به جا می کردند . ستون اول در ابتدا دارای ۶۴ دیسک با اندازه های مختلف می باشد، که بزرگترین دیسک در پایین ستون و کوچکترین دیسک در بالای ستون قرار دارد. کاهنان باید همه دیسکها را از یک ستون به ستون دوم منتقل می کردند. با این شرط که در هر بار جا به جایی تنها یک دیسک منتقل شود و نیز دیسک بزرگتری روی دیسک کوچکتر قرار نگیرد. ضمناً ستون سومی به عنوان ستون کمکی در اختیار آنها می باشد. گویند هنگامی که کاهنان معبد همه ۶۴ دیسک را با روش گفته شده از ستون اول به ستون دوم منتقل کنند جهان به پایان می رسد. شکل زیر نحوه انتقال سه دیسک را نشان می دهد.



برای راحتی کار کاهنان و برای اینکه دچار اشتباه و دوباره کاری در انتقال نشوند می خواهیم برنامه ای بنویسیم که ترتیب انتقال دیسکها را چاپ کند.

برای نوشتن این برنامه ، مسئله را باید با دید بازگشتی نگاه کنیم . انتقال **n** دیسک را به شیوه زیر انجام می دهیم :

۱- ابتدا **n-1** دیسک را از ستون اول به ستون دوم به کمک ستون سوم منتقل کن.
۲- دیسک آخر (بزرگترین دیسک) را از ستون اول به ستون سوم منتقل کن.
۳- **n-1** دیسک قرار گرفته در ستون دوم را به کمک ستون اول به ستون سوم منتقل کن.

مراحل انجام کار هنگام انتقال آخرین دیسک یعنی وقتی که **n=1** می باشد، یعنی حالت پایه به اتمام می رسد. در حالت **n=1** یک دیسک بدون کمک ستون کمکی به ستون دیگر منتقل می شود.

تابع بازگشتی مورد استفاده برای حل مسئله برجهای هانوی را با چهار آرگومان می نویسیم.

۱-	تعداد	دیسکها
۲-	ستون	مبدأ
۳-	ستون	کمکی
۴- ستون مقصد		

تابع هانوی و برنامه ای که در آن این تابع مورد استفاده قرار گرفته است به صورت زیر می باشد :

```
#include <iostream.h>

int hanoi(int, char, char, char);

int main( )
{ int disks;

  cout<<"Moving disks form tower A to C."<<endl;
  cout<<"How many disks do you want to move?";
  cin>>disks;
  cout<<hanoi(disks,'A','B','C')<<endl;

  return 0;
}

int hanoi(int n, char first, char help, char second)
{
  if (n == 1) {
    cout << "Disk " << n << " from tower " << first
         << " to tower " << second << endl;
  }
  else {
     hanoi(n-1, first, second, help);
     cout << "Disk " << n << " from tower " << first
          << " to tower " << second << endl;
     hanoi(n-1, help, first, second);
```