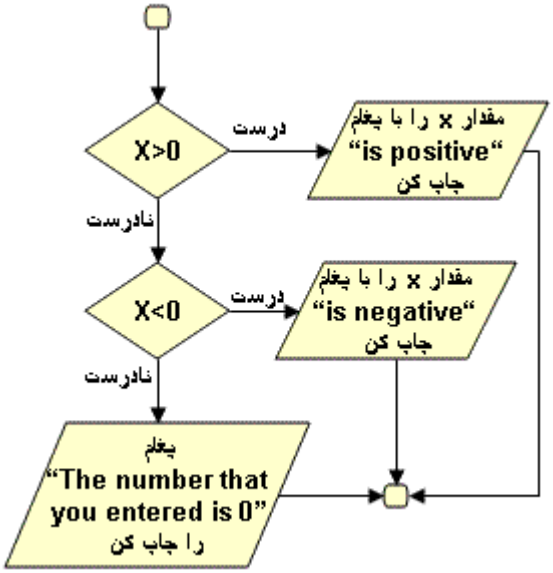
```
  else
     if ( x < 0 )
        cout << x << "is negative.";
     else
        cout << "The number that you entered is 0.";
  return 0;
}
```

برنامه فوق را سه بار با سه عدد مختلف اجرا می کنیم. خروجی ها به صورت زیر می باشند:

```
Please enter a number : 10
10 is positive.
Please enter a number : -5
-5 is negative.
Please enter a number : 0
The number that you entered is 0.
```



نکته : برای وضوح برنامه پیشنهاد می شود همانند برنامه فوق هنگام استفاده از **if** یا **if/else** و یا دیگر ساختارهای کنترلی از تورفتگی های مناسب استفاده کنید. یعنی به عنوان مثال دستور **if** را به صورت زیر:

```
if ( x > 0 )
    cout << x << "is positive.";
```

بنویسیم و نه به صورت زیر :

```
if ( x > 0 )
cout << x << "is positive.";
```

ساختار چند انتخابی switch

در دو مبحث قبلی ساختارهای **if** و **if/else** را بررسی کردیم. در برنامه نویسی گاهی به الگوریتمی نیاز پیدا می کنیم که در آن متغیری به ازای هر مقدار صحیح ثابتی، باعث اجرای یک دستور خاص شود و به ازای هر مقدار اعمال مختلف انجام پذیرد. برای نیل به این هدف C++ ساختار چند انتخابی **switch** را که به صورت زیر می باشد در اختیار ما قرار داده است:

```
switch  (عبارتی که باید مورد بررسی قرار گیرد  )
   {
      case  مقدار ثابت ۱  :
                  مجموعه دستورات ۱
                  break;
      case  مقدار ثابت ۲  :
                  مجموعه دستورات ۲
                  break;

      .
      .
      .

      case   n  مقدار ثابت  :
                  n  مجموعه دستورات
                  break;
      default  :
           مجموعه دستورات حالت پیش فرض
 }
```

ساختار **switch** به شیوه زیر عمل می کند:

switch ابتدا عبارت داخل پرانتز را مورد ارزیابی قرار می هد و سپس آن را با **مقدار ثابت ۱** مورد مقایسه قرار می دهد. اگر برابر بودند **مجموعه دستورات ۱** را اجرا خواهد شد، تا هنگامی که برنامه به دستور **break** برسد، هنگامی که برنامه به دستور **break** رسید از ساختار چند انتخابی **switch** خارج می شود. اگر عبارت داخل پرانتز با **مقدار ثابت ۱** برابر نبود ساختار **switch** عبارت داخل پرانتز را با **مقدار ثابت ۲** مورد مقایسه قرار می دهد، در صورت برابر بودن **مجموعه**

دستورات ۲ اجرا می گردد. این روال همینطور ادامه پیدا می کند. در صورتی که عبارت داخل پرانتز با هیچ یک از مقادیر ثابت برابر نباشد، مجموعه دستورات حالت **default** (پیش فرض) اجرا می گردد. به برنامه زیر توجه کنید:

```
#include <iostream.h>
int main( )
{
  int x;
  cout << "Please enter a number:";
  cin >> x;

  switch (x) {
    case 1:
      cout << "x is 1";
      break;
    case 2:
      cout << "x is 2";
      break;
    default:
      cout << "Unknown value";
  }
  return 0;
}
```

برنامه فوق را سه بار با سه عدد مختلف اجرا می کنیم. خروجی ها به صورت زیر می باشند:

```
Please enter a number:1
x is 1
Please enter a number:2
x is 2
Please enter a number:5
Unknown value
```

توجه داشته باشید که ساختار **switch** را می توان با ساختار **if/else** نیز پیاده سازی کرد. به عنوان مثال ساختار **switch** به کار رفته در مثال فوق معادل ساختار **if/else** زیر می باشد:

```
if (x == 1)
    cout << "x is 1";
else
    if (x == 2)
        cout << "x is 2";
    else
       cout << Unknown value";
```

ما الزامی به استفاده از حالت **default** در ساختار **switch** نداریم ولی توصیه می شود که حالت پیش فرض را به کار ببریم چون معمولاً امکان دارد که عبارت برابر با هیچ یک از مقادیر ثابت نباشد و با به کار بردن حالت پیش فرض می توانید پیغام مناسبی در این رابطه به صفحه نمایش بفرستید.

توجه داشته باشید اگر دستور **break** بعد از هر مجموعه از دستورات استفاده نکنیم برنامه از ساختار **switch** خارج نخواهد شد و مجموعه دستورات بعدی اجرا می گردد تا به اولین دستور **break** برسد. این مورد به ما امکان ایجاد حالتهای ترکیبی را می دهد. البته در به کار بردن این تکنیک دقت لازم را بکنید.

```
#include <iostream.h>
int main( )
{
  int x;
  cout << "Please enter a number:";
  cin >> x;

  switch (x) {
    case 1:
    case 2:
    case 3:
      cout << "x is (1 or 2 or 3)";
      break;
    default:
      cout << "x is not (1 or 2 or 3)";
    }
  return 0;
}
```

برنامه فوق را سه بار با سه عدد مختلف اجرا می کنیم. خروجی ها به صورت زیر می باشند:

```
Please enter a number:1
x is (1 or 2 or 3)
Please enter a number:2
x is (1 or 2 or 3)
Please enter a number:5
x is not (1 or 2 or 3)
```

ساختار تکرار while

ساختار تکرار (حلقه تکرار) به برنامه نویس این امکان را می دهد که برنامه ، قسمتی از دستورات را تا هنگامی که شرط خاصی برقرار است، را تکرار کند. به عنوان مثال :

تا وقتی که مورد دیگری در لیست خرید من هست.
آن را بخر و از لیست خرید حذفش کن.

مورد فوق روال یک خرید را انجام می دهد. شرط مورد نظر " مورد دیگری در لیست خرید من هست" می باشد، که ممکن است درست یا نادرست باشد. اگر شرط برقرار باشد (یعنی مورد دیگری در لیست خرید باشد) عمل "خرید آن و حذفش از لیست" انجام می گیرد. این عمل تا وقتی که شرط برقرار باشد ادامه می یابد. هنگامی که شرط برقرار نباشد (یعنی تمام موارد لیست خرید حذف شده باشند)، ساختار تکرار به پایان می رسد و اولین دستور بعد از حلقه تکرار، اجرا می گردد. ساختار تکرار **while** به صورت زیر می باشد.

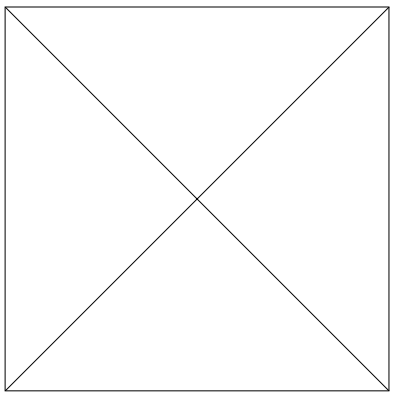
```
while (شرط مورد نظر )
      {
          مجموعه دستورات
      }
```

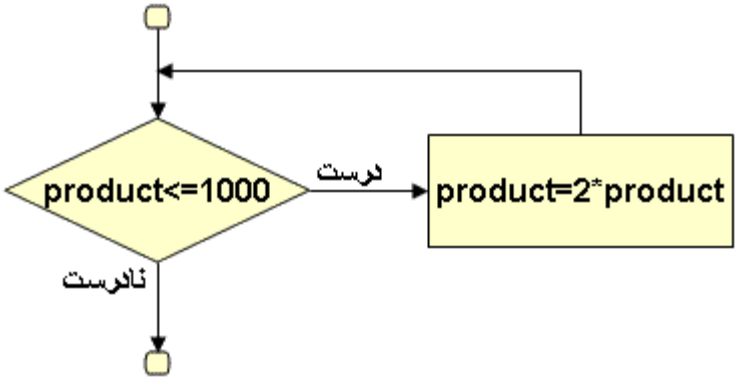
تا وقتی که شرط داخل پرانتز برقرار باشد مجموعه دستورات اجرا خواهند شد. برای درک بهتر شیوه کاربرد حلقه های تکرار فرض کنید می خواهیم اولین توانی از عدد **۲** که بزرگتر از **۱۰۰۰** می باشد را بیابیم. برنامه به صورت زیر خواهد بود.

```
#include <iostream.h>
int main( )
 {
  int product = 2;
  while (product <= 1000)
       product = 2 * product;

  cout << "The first power of 2 larger than 1000 is "
       <<product <<endl;
  return 0;
 }
```

در برنامه فوق ابتدا متغیری به نام **product** را با مقدار اولیه **۲** مقدار دهی کردیم. در حلقه تکرار **while** با هر بار اجرای دستور **product=2*product** مقدار متغیر **product** دو برابر می شود بدین ترتیب با پایان یافتن حلقه متغیر **product** حاوی عدد **۱۰۲۴** یعنی اولین توانی از **۲** که بزرگتر از **۱۰۰۰** می باشد، خواهد بود.

```
The first power of 2 larger than 1000 is 1024
```





نکته : در مثال فوق در حلقه **while** چون تنها از یک دستور استفاده شده بود از {} استفاده نشد، ولی اگر بیش از یک دستور داشتیم ملزم به استفاده از {} بودیم.

مثال : برنامه ای بنویسید تا مجموع اعداد یک تا صد را محاسبه کند.

```
#include <iostream.h>
int main( )
  {
    int n=1, sum=0;
    while (n <= 100)
      {
        sum += n;        // sum = sum + n;
        ++n;             // n = n + 1;
      }
    cout << "1 + 2 + ... + 100 =" <<sum << endl;
    return 0;
  }
```

در مثال فوق حلقه ۱۰۰ بار اجرا می گردد و هر بار عدد **n** به متغیر **sum** اضافه می گردد و عدد **n** نیز یک واحد افزایش می یابد تا در یکصدمین بار اجرای حلقه مقدار متغیر **n** برابر ۱۰۱ می شود و هنگام بررسی شرط (**n** <=۱۰۰) توسط حلقه **while** شرط نادرست می شود و اولین دستور بعد از حلقه یعنی دستور خروجی **cout** اجرا می گردد.

```
1 + 2 + ... + 100 =5050
```

نکته :

- در مثال فوق متغیر **n** به عنوان شمارنده دفعات تکرار حلقه بکار گرفته شد. برحسب مورد شمارنده ها معمولاً با یک یا صفر مقدار دهی اولیه می شوند.
- متغیر **sum** حاوی مجموع حاصلجمع بود. چنین متغیرهایی که برای محاسبه یک حصلجمع به کار می روند معمولاً با صفر مقدار دهی اولیه می شوند.

مثال : برنامه ای بنویسید که تعداد نامشخصی عدد مثبت را از ورودی دریافت نماید و میانگین آنها را محاسبه نماید. عدد **۱-** را برای مشخص کردن انتهای لیست اعداد در نظر بگیرید.

```
#include <iostream.h>
int main( )
{ int num, counter = 0;
  float average, sum = 0;

  cout << "Enter a number (-1 to end):";
  cin >>num;

  while (num != -1){
    sum += num ; // sum = sum + sum;
    ++counter;
    cout << "Enter a number (-1 to end):";
    cin >> num;
  }

  if (counter != 0){
    average = sum / counter;
    cout << "The average is " << average << endl;
  }
  else
    cout << "No numbers were entered." << endl;

  return 0;
}
```

خروجی برنامه به صورت زیر خواهد بود.

```
Enter a number (-1 to end):20
Enter a number (-1 to end):50
Enter a number (-1 to end):65
Enter a number (-1 to end):70
Enter a number (-1 to end):90
Enter a number (-1 to end):100
Enter a number (-1 to end):1
Enter a number (-1 to end):6
Enter a number (-1 to end):-1
The average is 50.25
```

در برنامه مثال قبل عدد ۱- به عنوان یک مقدار کنترلی به کار می رود و با وارد کردن این عدد اجرای برنامه به پایان می رسد و میانگین اعداد در خروجی به نمایش در می آید. متغیر **num** اعداد را از ورودی دریافت می کند. متغیر **counter** وظیفه شمارش تعداد اعداد وارد شده را دارا می باشد و متغیر **sum** مجموع حاصلجمع اعداد را در خود قرار می دهد و در نهایت متغیر **average**، میانگین را در خود قرار می دهد. ساختار کنترلی **if** به کار رفته در برنامه، جلوی بروز خطای زمان اجرای تقسیم بر صفر را می گیرد ، یعنی اگر در اولین دستور **cin** به کار رفته عدد ۱- وارد شود خروجی برنامه به صورت زیر خواهد بود :

```
Enter a number (-1 to end): -1
No numbers were entered.
```

ساختار تکرار do/while

ساختار تکرار **do/while** مشابه ساختار تکرار **while** می باشد. در ساختار تکرار **while** شرط حلقه در ابتدا بررسی می شود ولی در ساختار تکرار **do/while** شرط در انتهای حلقه مورد بررسی قرار می گیرد، بدین ترتیب در ساختار تکرار **do/while** دستورات حلقه حداقل یکبار اجرا خواهند شد. ساختار تکرار **do/while** به صورت زیر می باشد:

```
do {
      مجموعه دستورات
   }while ( شرط مورد نظر );
```

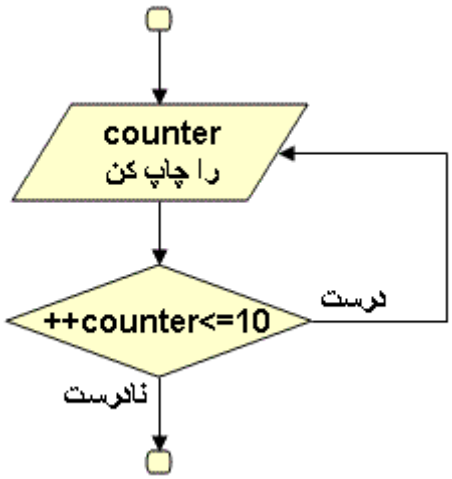
به عنوان مثال به برنامه زیر توجه نمایید:

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

```
int main()
{
   int counter = 1;
   do {
       cout << counter << " ";
      }while ( ++counter <= 10 );
   cout << endl;

   return 0;
}
```

در این برنامه اعداد ۱ تا ۱۰ با فاصله بر روی صفحه نمایش چاپ خواهند شد. دقت کنید که متغیر **counter** در قسمت شرط حلقه ، یک واحد اضافه می گردد سپس مقدارش با عدد ۱۰ مقایسه می گردد.

```
1 2 3 4 5 6 7 8 9 10
```



مثال: برنامه ای بنویسید که لیست نمرات یک کلاس را دریافت کرده و تعداد قبولی ها و مردودی ها را مشخص کند. ضمنا در ابتدای برنامه تعداد نمرات لیست پرسیده شود.

```
#include <iostream.h>
int main( )
{
  float mark;
  int howmany,counter=1;
  int passes=0,failures=0;

  cout << "How many marks : ";
  cin >> howmany;
```

```
  do {
      cout << "Enter mark "<<counter<<" : ";
      cin>>mark;
      if (mark>=10)
         ++passes;
      else
         ++failures;
     }while (++counter <= howmany);

  cout<<"Passed : "<<passes<<endl;
  cout<<"Failed : "<<failures<<endl;

  return 0;
}
```

خروجی برنامه به صورت زیر می باشد :

```
How many marks : 10
Enter mark 1 : 18
Enter mark 2 : 15
Enter mark 3 : 9
Enter mark 4 : 17.5
Enter mark 5 : 9.75
Enter mark 6 : 8
Enter mark 7 : 11
Enter mark 8 : 13
Enter mark 9 : 5
Enter mark 10 : 13
Passed : 6
Failed : 4
```

ساختار تکرار for

ساختار تکرار **for** نیز مانند دو ساختار قبلی یک حلقه تکرار می سازد. از ساختار تکرار **for** معمولاً هنگامی که دفعات تکرار حلقه براساس یک شمارنده می باشد استفاده می شود. ساختار تکرار **for** به صورت زیر می باشد:

```
for ( تعریف متغیر شمارنده و تعیین مقدار اولیه ; شرط حلقه ; افزایش یا کاهش مقدار شمارنده )
```

```
    {
        مجموعه دستورات
    }
```

ساختار تکرار **for** را با ساختار تکرار **while** نیز می توان پیاده سازی کرد به عنوان مثال دو برنامه زیر اعداد **۱** تا **۵** را بر روی صفحه نمایش چاپ می کنند:

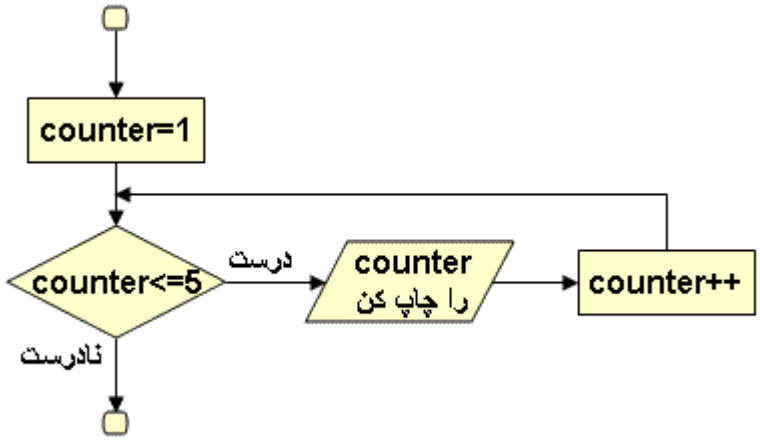
```
#include <iostream>

int main()
{
  int counter = 1;

  while ( counter <= 5 ) {
     cout << counter << endl;
     ++counter;
  }
  return 0;
}
```

برنامه فوق با حلقه **while** نوشته شده بود. در برنامه زیر معادل حلقه **while** فوق را با حلقه **for** پیاده سازی می کنیم:

```
#include <iostream>

int main()
{

  for ( int counter = 1; counter <= 5; counter++ )
     cout << counter << endl;

return 0;
}
```

در برنامه فوق هنگامی که دستور **for** اجرا می شود متغیر کنترلی **counter** تعریف می گردد و عدد **۱** در آن قرار می گیرد. سپس شرط حلقه مورد بررسی قرار می گیرد (**counter**<=**۵**) چون مقدار **counter** ، عدد **۱** می باشد پس شرط درست است و دستور حلقه، یعنی دستور **cout** اجرا می گردد و اولین عدد یعنی **۱** نمایش داده می شود. پس از آن دستور **counter**++ اجرا می گردد و مقدار متغیر **counter** یک واحد اضافه می شود. سپس مجدداً شرط حلقه بررسی و در صورت برقرار بودن شرط دستور **cout** اجرا می گردد. این روال تا وقتی که شرط برقرار باشد ادامه می یابد و به محض برقرار نبودن شرط یعنی هنگامی که **counter** حاوی عدد **۶** شود خاتمه می یابد و برنامه به پایان می رسد.

```
1 2 3 4 5
```



در برنامه قبلی اگر حلقه **for** را به صورت زیر بازنویسی کنیم:

```
for(int counter=10; counter>=1; counter=counter-2)
```

خروجی برنامه اعداد زوج **۱۰** تا **۱** به صورت معکوس می باشد، یعنی :

```
10 8 6 4 2
```

توجه داشته باشید که در حلقه فوق به جای استفاده از دستور **counter=counter-1** می توانستیم از دستور **counter -= 2** استفاده کنیم.

مثال : برنامه ای بنویسید که مجموع اعداد زوج **۱** تا **۱۰۰** را محاسبه کند.

```
#include <iostream.h>

int main ( )
{ int sum = 0;

  for (int num = 2; num <= 100; num += 2)
     sum += num;
  cout << "2 + 4 + 6 + ... + 100 =" <<sum<<endl;

  return 0;
}
```

```
2 + 4 + 6 + ... + 100 =2550
```

توجه داشته باشید که حلقه for در برنامه فوق را با کمک عملگر کاما (,) می توانیم به صورت زیر نیز بنویسیم:

```
for (int num = 2;
     num <= 100;
     sum += num, num +=2);
```

ضمناً شکستن حلقه به چند خط نیز مشکلی ایجاد نمی کند. البته دو مورد اخیر توصیه نمی شوند، چون از خوانایی برنامه می کاهند.

مثال : برنامه ای بنویسید که عددی را از ورودی دریافت کرده و **۲** به توان آن عدد را محاسبه و در خروجی چاپ نماید.

```
#include <iostream.h>
int main( )
{
  unsigned long int x=1;
  int power;

  cout << "Enter power:";
  cin >>power;

  for (int counter=1;counter<=power;counter++)
    x*=2;

  cout << "2 ^ " << power << " = " << x <<endl;

return 0;
}
```

```
Enter power:25
2 ^ 25 = 33554432
```

در مثال های فوق، دستورات حلقه **for** را داخل { } قرار ندادیم چون حلقه **for** تنها شامل یک دستور بود، توجه داشته باشید که اگر بیش از یک دستور در حلقه به کار رود ملزم به استفاده از { } می باشیم.

مثال : برنامه ای بنویسید که جدول ضرب 5X5 را ایجاد کند.

```
#include <iostream.h>
int main( )
{

  for (int x=1;x<=5;x++)
   {
```

```
      for (int y=1;y<=5;y++)
         cout <<x*y<<"\t";
      cout<<endl;
    }

  return 0;
}
```

خروجی برنامه به صورت زیر خواهد بود:

```
1     2     3     4     5
2     4     6     8     10
3     6     9     12    15
4     8     12    16    20
5     10    15    20    25
```

در برنامه فوق حلقه شامل متغیر **x** ، دارای دو دستور **for** و **cout** بود، به همین علت از { } استفاده شد. اما حلقه شامل متغیر y تنها دارای یک دستور **cout** بود. اگر دقت کرده باشید دستور **t\"<<x*y<<cout";** دارای علامت **t** بود. به کار بردن **t** باعث جدول بندی و مرتب شدن خروجی می شود. در حقیقت مکان نمای صفحه نمایش را در محل جدول بندی قرار می دهد. ضمناً در مثال فوق یک ساختار **for** در دل ساختار **for** دیگری استفاده شد به این شیوه استفاده حلقه های تودرتو گفته می شود که در برنامه نویسی ها به کرات از آنها استفاده می شود. در ضمن توجه داشته باشید که اگر از دستور **cout;** <<**endl** استفاده نشود، خروجی به صورت نا مرتب زیر خواهد بود:

```
1    2    3    4    5    2    4    6    8    10
3    6    9    12   15   4    8    12   16   20
5    10   15   20   25
```

نکته : در حلقه های تکرار ممکن است شرط حلقه را به اشتباه بنویسیم یا به عنوان مثال شمارنده حلقه را افزایش ندهیم در چنین حالاتی ممکن است پس از اجرای برنامه، برنامه به اتمام نرسد و حلقه همچنان تکرار شود. در صورت بروز چنین مشکلی با فشردن کلید **Ctrl** همراه با **Break** (**Ctrl+Break**) اجرای برنامه به صورت ناگهانی قطع می شود و به محیط ویرایشگر C++ باز می گردید و می توانید کد اشتباه را درست کنید. سپس برنامه را مجدداً اجرا کنید.

دستور های break و continue

دستور **break** هرگاه که در ساختارهای **while** و **do/while** و **for** یا **switch** اجرا گردد، باعث خروج فوری برنامه از آن ساختار خواهد شد و برنامه اولین دستور بعد از آن ساختار را اجرا خواهد کرد. به برنامه زیر توجه کنید:

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

```
int main()
{
  int n;
  for (n=10; n>0; n--)
  {
     cout << n << ",";
     if (n==3)
     {
       cout << "countdown aborted!";
       break;
     }
  }
  return 0;
}
```

خروجی برنامه به صورت زیر می باشد:

```
10,9,8,7,6,5,4,3,countdown aborted!
```

برنامه فوق اعداد ۱۰ تا ۴ را چاپ خواهد کرد و هنگامی که متغیر **n** عدد ۳ می شود، شمارش معکوس به پایان می رسد.

نکته : در برنامه فوق شمارنده حلقه یعنی **n** در خارج از دستور **for** تعریف شد. در چنین حالتی ، این متغیر خارج از حلقه نیز می تواند مورد استفاده قرار گیرد ولی اگر تنها در داخل حلقه تعریف شده بود ، تنها آنجا می توانستیم از آن استفاده کنیم و خارج حلقه تعریف نشده بود.

دستور **continue** هرگاه در ساختارهای **while** و **do/while** یا **for** اجرا گردد دستورات بعدی آن ساختار نادیده گرفته می شود و بار بعدی حلقه تکرار اجرا می شود. در دو ساختار **while** و **do/while** پس از اجرای دستور **continue** شرط حلقه مورد بررسی قرار می گیرد، اما در ساختار **for** ابتدا مقدار شمارنده افزایش یا کاهش می یابد، سپس شرط حلقه بررسی می شود. توجه داشته باشید که در حلقه **while** و**do/while** دستور **continue** همواره بعد از افزایش یا کاهش شمارنده به کار رود. به عنوان مثال برنامه زیر مجموع اعداد ۱ تا ۲۰ به جز ۱۰ را محاسبه می کند.

```
#include <iostream.h>

int main( )
{
  int n=0, sum=0;
  while (n < 20)
  {
    ++n; // n = n + 1;
    if (n==10) continue;
    sum += n; // sum = sum + n;
```