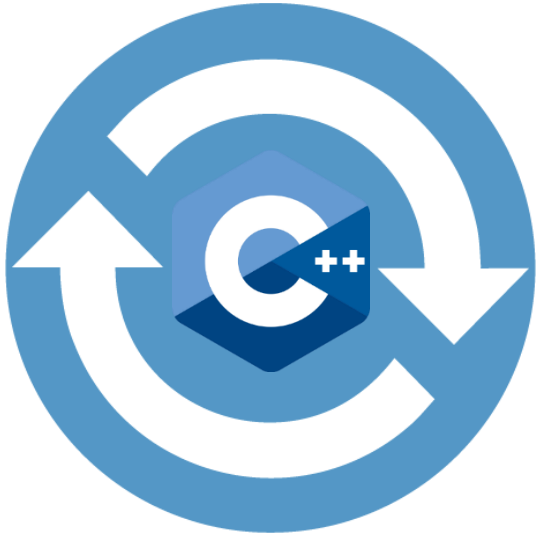
مبانی کامپیوتر و برنامه سازی

اسلاید دهم

«برنامه نویسی مقدماتی به زبان C++»

بخش سوم : دستورات حلقه و تکرار



درس

مبانی کامپیوتر

۱۰

محمد سعید صفایی صادق

۱۴۰۲

حلقه و تکرار

حلقه و تکرار

یکی از ابزارهای پر استفاده و مهم هر زبان برنامه‌نویسی حلقه‌های تکرار هستند.

وجود چنین ابزاری به برنامه‌نویس این امکان را می‌دهد که ساختارهای نیازمند به تکرار مجموعه دستورات (مانند جستجو، گزارش‌گیری، محاسبات، دریافت اطلاعات از کاربر یا فایل) را پیاده‌سازی کند.

هر زبانی عموما شامل چندین نوع حلقه تکرار است که هر کدام به نحوی به برنامه‌نویس در نوشتن کدهای مختصر و با مفهوم کمک می‌کنند. در این فرصت با انواع حلقه‌های تکرار در زبان برنامه‌نویسی C ++ آشنا می‌شویم.

حلقه و تکرار

تکرار، اجرای پی در پی یک دستور یا بلوکی از دستورالعمل‌ها در یک برنامه است.

با استفاده از تکرار می‌توانیم کنترل برنامه را مجبور کنیم تا به خطوط قبلی برگردد و آن‌ها را دوباره اجرا نماید.

C++ دارای سه دستور تکرار است: دستور while، دستور do_while و دستور for .

دستورهای تکرار به علت طبیعت چرخه‌مانندشان، **حلقه** نیز نامیده می‌شوند.

اجزای دستورات حلقه و تکرار

۱- مقداردهی اولیه

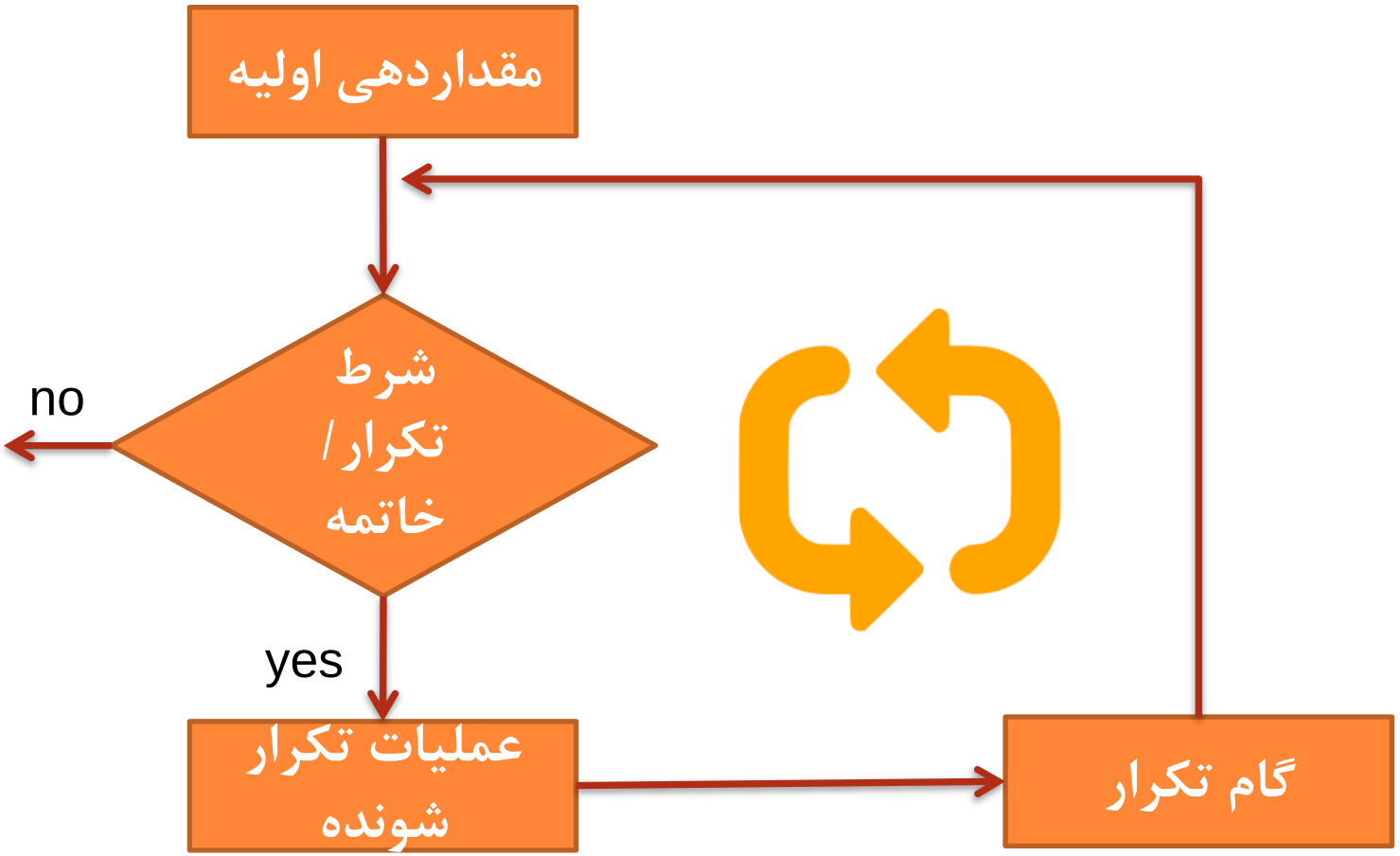
۲- شرط تکرار / شرط خاتمه

۳- عملیات تکرار شونده

۴- گام تکرار

دستور for

دستور for



دستور for

ساختار تکرار forیکی از امکانات ایجاد حلقه است و معمولا در حالتی بکار میرود که تعداد دفعات تکرار حلقه از قبل مشخص باشد.

در این ساختار متغیری وجود دارد که تعداد دفعات تکرار را کنترل میکند.

این متغیر شمانده یا اندیس حلقه تکرار نام دارد.

اندیس حلقه دارای مقدار اولیه است و در هربار اجرای دستورات حلقه، مقداری به آن اضافه میشود که گام حرکت حلقه نام دارد.

گام حرکت حلقه میتواند عددی صحیح و اعشاری، مثبت یا منفی، و یا کاراکتری باشد.

دستور for

یکی دیگر از اجزای حلقه for، شرط حلقه است.

شرط حلقه مشخص میکند که دستورات داخل حلقه تا کی باید اجرا شوند.

اگر شرط دارای ارزش درستی باشد دستورات حلقه اجرا میشوند وگرنه کنترل برنام از حلقه خارج میشود.

دستور for را میتوان به دو صورت بکار برد.

for

- نحو دستورالعمل for به صورت زیر است:

- **for** (initialization; *condition*; update) statement;

for (نام متغیر مقدار اولیه ; عبارت منطقی ; تغییر مقدار متغیر)

دستور ;

- سه قسمت داخل پرانتز، حلقه را کنترل می‌کنند.

عبارت initialization برای اعلان یا مقداردهی اولیه به متغیر کنترل حلقه استفاده می‌شود.این عبارت اولین عبارتی است که ارزیابی می‌شود پیش از این که نوبت به تکرارها برسد.

عبارت *condition* برای تعیین این که آیا حلقه باید تکرار شود یا خیر به کار می‌رود. یعنی این عبارت، شرط کنترل حلقه است. اگر این شرط درست باشد دستور statement اجرا می‌شود.

عبارت update برای پیش‌بردن متغیر کنترل حلقه به کار می‌رود.

این عبارت پس از اجرای statement ارزیابی می‌گردد.

نکته

به علامت (;) در دستور for توجه کنید. بعد از علامت پرانتز علامت(;) نگذارید، زیرا دستور for هنوز تمام نشده است. علامت ; باید در انتهای دستور نوشته شود.

for (نام متغیر مقدار اولیه ; عبارت منطقی ; تغییر مقدار متغیر)

دستور ;

بنابراین زنجیرهٔ وقایعی که تکرار را ایجاد می‌کنند عبارتند از:

۱ – ارزیابی عبارت **initialization**

۲ – بررسی شرط ***condition*** . اگر نادرست باشد، حلقه خاتمه می‌یابد.

۳ – اجرای **statement**

۴ – ارزیابی عبارت **update**

۵ – تکرار گام‌های ۲ تا ۴

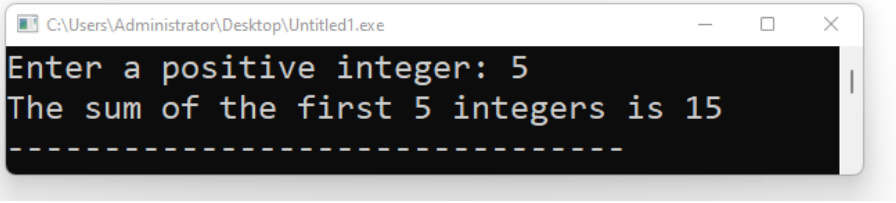
عبارت‌های **initialization** و ***condition*** و **update** عبارت‌های اختیاری هستند.

یعنی می‌توانیم آن‌ها را در حلقه ذکر نکنیم.

مثال

استفاده از حلقهٔ for برای محاسبهٔ مجموع اعداد صحیح متوالی

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {   int n;
5       cout << "Enter a positive integer: ";
6       cin >> n;
7       long sum=0;
8       for (int i=1; i <= n; i++)
9           sum += i;
10      cout << "The sum of the first " << n << " integers is " << sum;
11  }
```



در C++ استاندارد وقتی یک متغیر کنترل درون یک حلقهٔ for اعلان می‌شود (مانند i در مثال بالا) حوزهٔ آن متغیر به همان حلقهٔ for محدود می‌گردد.

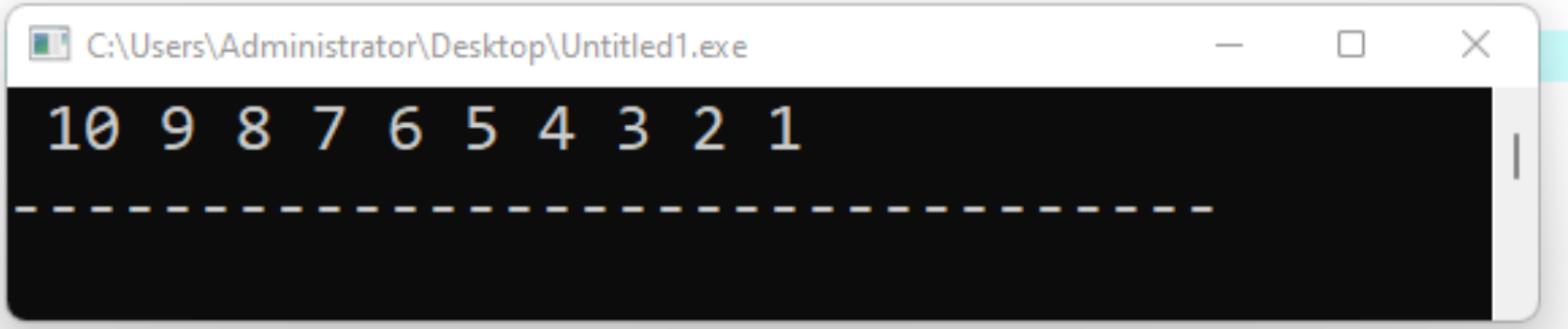
یعنی آن متغیر نمی‌تواند بیرون از آن حلقه استفاده شود.

نتیجهٔ دیگر این است که می‌توان از نام مشابهی در خارج از حلقهٔ for برای یک متغیر دیگر استفاده نمود.

مثال استفاده از حلقهٔ for نزولی

برنامهٔ زیر ده عدد صحیح مثبت را به ترتیب نزولی چاپ می‌کند:

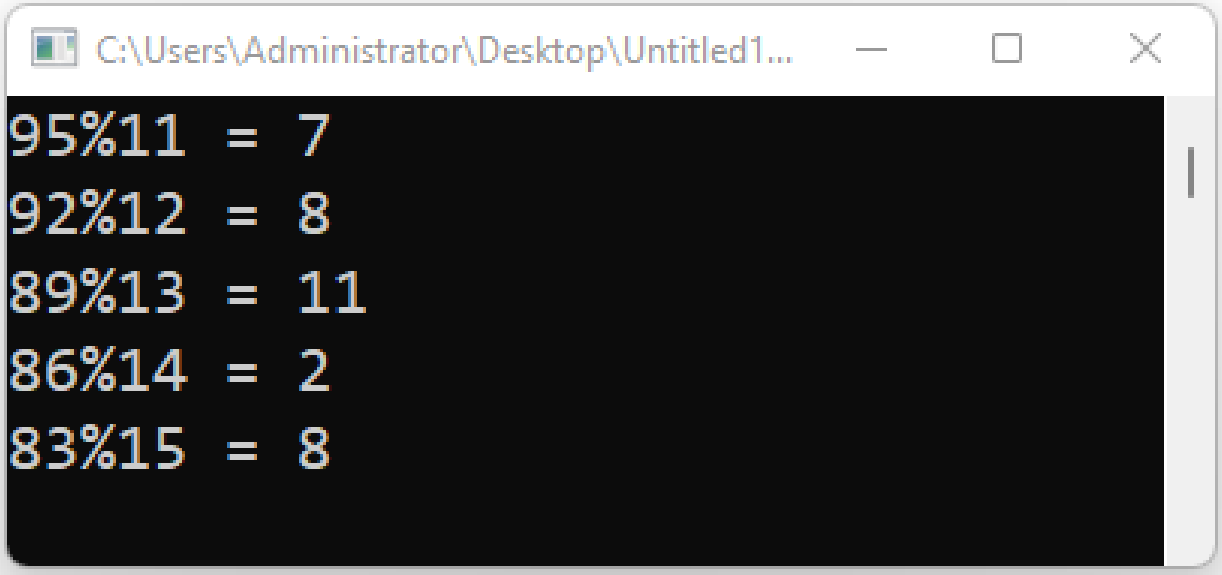
```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      for (int i=10; i > 0; i--)
6          cout << " " << i;
7  }
8
9
```



بیشتر از یک متغیر کنترل در حلقهٔ for

حلقهٔ **for** در برنامهٔ زیر دو متغیر کنترل دارد:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      for (int m=95, n=11; m%n > 0; m -= 3, n++)
6          cout << m << "%" << n << " = " << m%n << endl;
7  }
8
9
10
```



حلقه for تو در تو

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      for (int x=1; x <= 10; x++)
6     {  for (int y=1; y <= 10; y++)
7            cout << "\t" << x*y;
8        cout << endl;
9     }
10 }
```

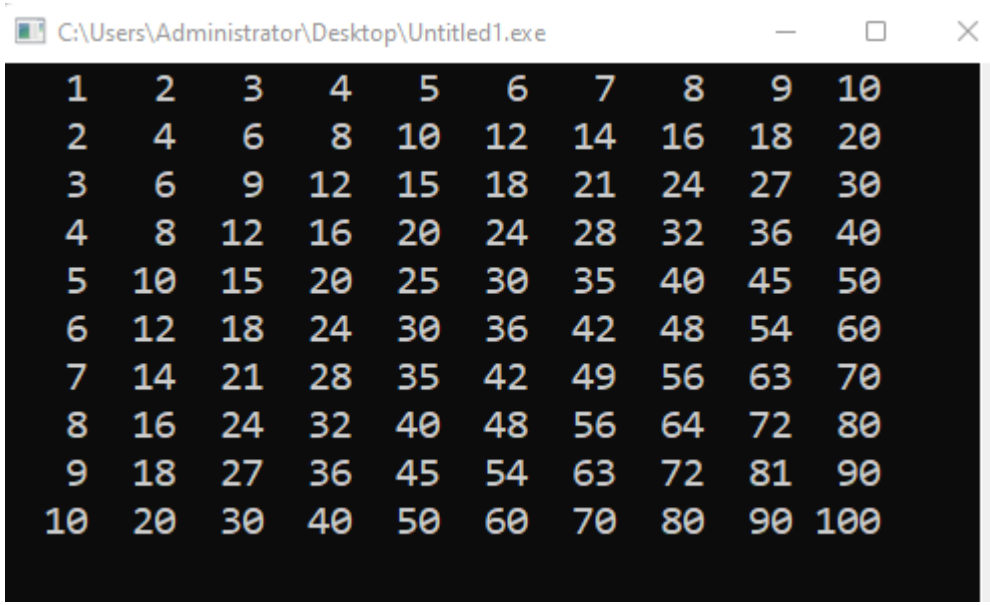
C:\Users\Administrator\Desktop\Untitled1.exe

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

حلقه for تو در تو

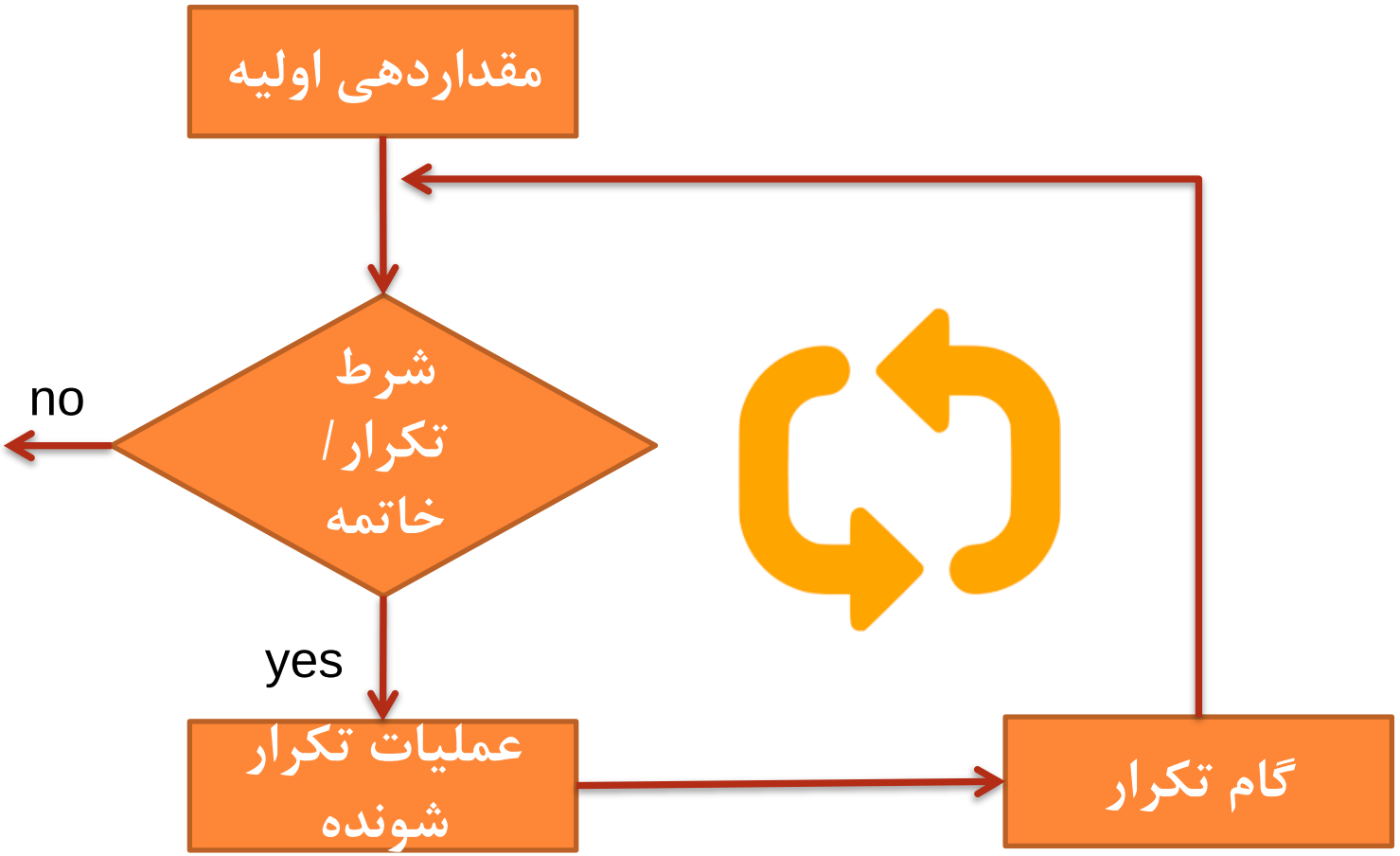
```
1  #include <iomanip>
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6      for (int x=1; x <= 10; x++)
7     {  for (int y=1; y <= 10; y++)
8           cout << setw(4) << x*y;
9        cout << endl;
10    }
11 }
```

تابع setw() این تابع در سر فایل iomanip قرار داره و کارش فاصله بندی خروجی های کنسول است.



دستور While

دستور While



While

نحو دستور **while** به شکل زیر است:

```
while (condition) statement;
```

به جای **condition**، یک شرط قرار می‌گیرد و به جای **statement** دستوری که باید تکرار شود قرار می‌گیرد.

اگر مقدار شرط، صفر (یعنی نادرست) باشد، **statement** نادیده گرفته می‌شود و برنامه به اولین دستور بعد از **while** پرش می‌کند.

اگر مقدار شرط ناصفر (یعنی درست) باشد، **statement** اجرا شده و دوباره مقدار شرط بررسی می‌شود.

این تکرار آن قدر ادامه می‌یابد تا این که مقدار شرط صفر شود.

While

نکته

به علامت نقطه ویرگول در دستور while توجه کنید. بعد از علامت پرانتز علامت نقطه ویرگول نگذارید، زیرا دستور while هنوز تمام نشده است. علامت نقطه ویرگول باید در انتهای دستور نوشته شود.

نقطه ویرگول ندارد

```
while (عبارت منطقی)
; دستور
```

نکته

اگر بخواهید بیش از یک دستور تکرار گردد، باید آنها را به‌صورت یک بلاک بنویسید. یعنی آنها را در داخل علامت‌های آکولاد باز و بسته قرار دهید.

While

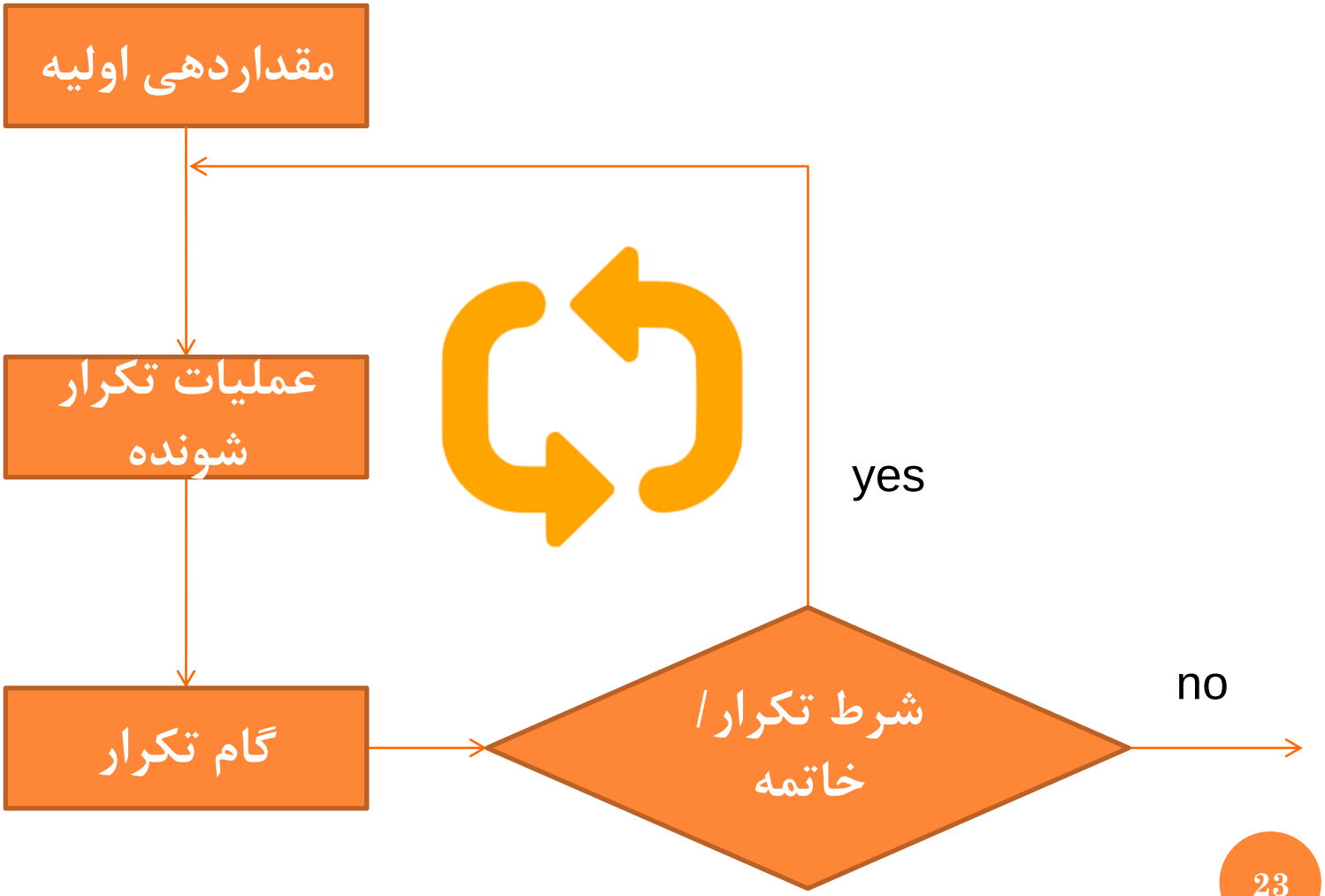
برنامه ای بنویسید که مقدار ۱ + ۲ + ۳ + ... + n را برای عدد ورودی n محاسبه کند. با استفاده از دستور while

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {   int n, i=1;
5      cout << "Enter a positive integer: ";
6      cin >> n;
7      long sum=0;
8      while (i <= n)
9          sum += i++;
10     cout << "The sum of the first " << n << " integers is "
11         << sum;
12 }
```

```
C:\Users\Administrator\Desktop\Untitled1.exe
Enter a positive integer: 5
The sum of the first 5 integers is 15
--------------------------------
```

دستور Do-While

دستور do-While



Do-While

ساختار **do..while** روش دیگری برای ساختن حلقه است.

نحو آن به صورت زیر است:

```
do
    statement
while (condition);
```

```
do
    دستور;
while (عبارت منطقی);
```

به جای **condition** یک شرط قرار می‌گیرد و به جای **statement** دستور یا بلوکی قرار می‌گیرد که قرار است تکرار شود.

این دستور ابتدا **statement** را اجرا می‌کند و سپس شرط **condition** را بررسی می‌کند.

اگر شرط درست بود حلقه دوباره تکرار می‌شود وگرنه حلقه پایان می‌یابد.

Do-While

دستور **do..while** مانند دستور **while** است.

با این فرق که شرط کنترل حلقه به جای این که در ابتدای حلقه ارزیابی گردد، در انتهای حلقه ارزیابی می‌شود.

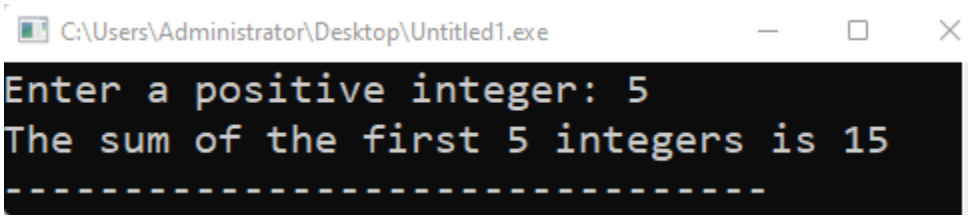
یعنی هر متغیر کنترلی به جای این که قبل از شروع حلقه تنظیم شود، می‌تواند درون آن تنظیم گردد.

نتیجهٔ دیگر این است که حلقهٔ **do..while** همیشه بدون توجه به مقدار شرط کنترل، لااقل یک بار اجرا می‌شود اما حلقهٔ **while** می‌تواند اصلا اجرا نشود.

مثال

محاسبهٔ حاصل جمع اعداد صحیح متوالی با حلقهٔ do..while

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {  int n, i=0;
6     cout << "Enter a positive integer: ";
7     cin >> n;
8     long sum=0;
9     do
10       sum += ++i;
11    while (i < n);
12    cout << "The sum of the first " << n << " integers is " << sum;
13     return 0;
14 }
```



اعداد فاکتوریل

اعداد فاکتوریل ۰! و ۱! و ۲! و ۳! و ... با استفاده از رابطه‌های بازگشتی زیر تعریف می‌شوند:

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

برای مثال، به ازای $n = 1$ در معادلهٔ دوم داریم:

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

همچنین برای $n = 2$ داریم:

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

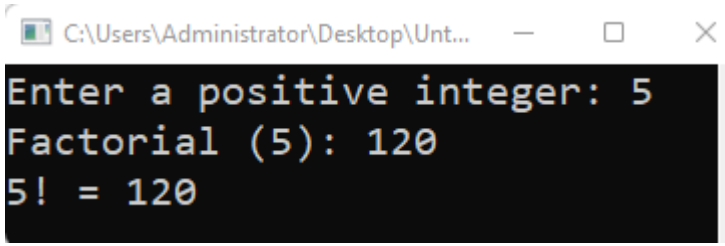
و به ازای $n = 3$ داریم:

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

مثال:

برنامهٔ زیر فاکتوریل N را محاسبه و چاپ می‌کند:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  int N;
5     cout << "Enter a positive integer: ";
6     cin >> N;
7     cout << "Factorial (" << N << "): ";
8     long long f=1, i=1;
9     do
10    {
11       f *= i++;
12    }
13    while (i <= N);
14    cout<<f<<endl;
15    cout<<N<<"! = " << f << endl;
16 }
```



مثال:

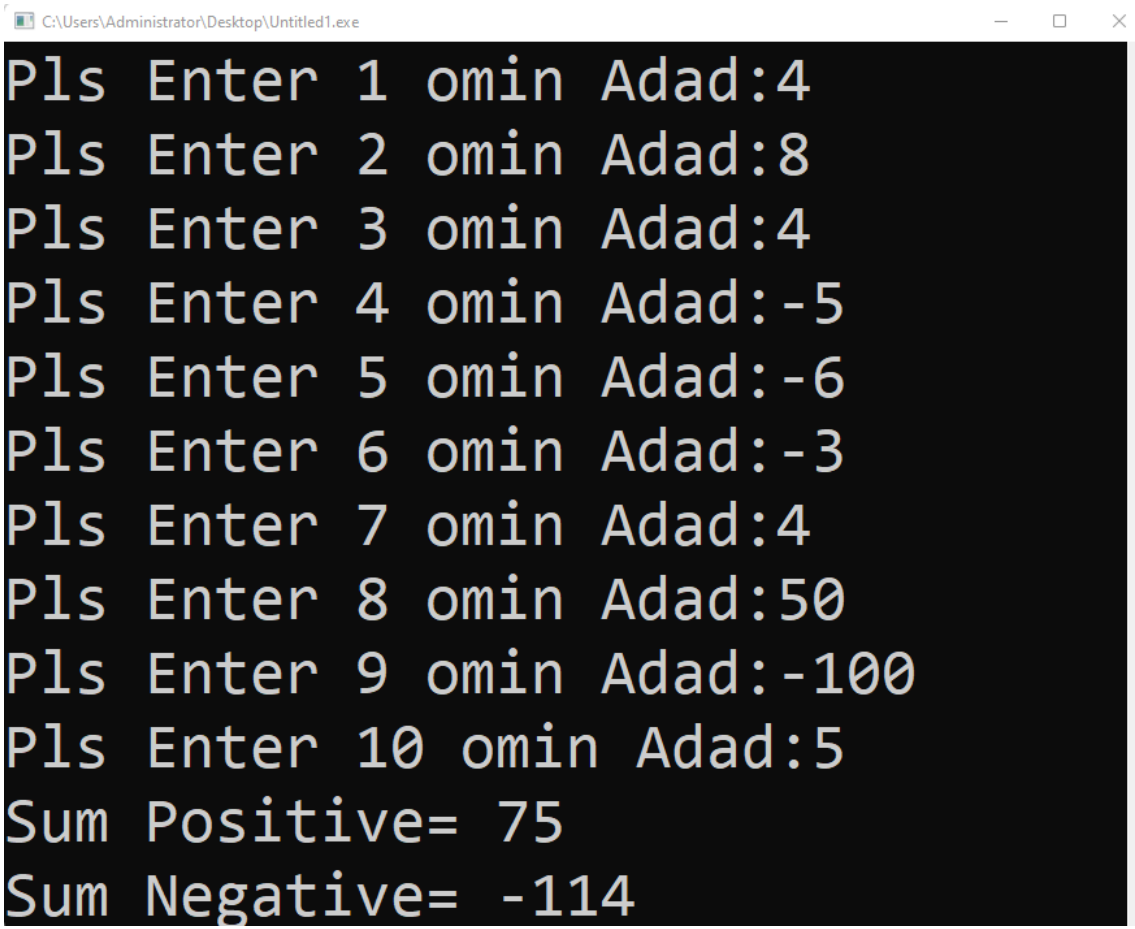
برنامه ای بنویسید که ۱۰ عدد از کاربر خوانده مجموع اعداد مثبت و منفی را جداگانه محاسبه کرده و نمایش دهد.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n,SN=0,SP=0,i=1;
6      do
7      {
8          cout << "Pls Enter "<<i<<" omin Adad:";
9          cin >> n;
10         (n>0)?SP=SP+n:SN=SN+n;
11         i++;
12     }
13     while (i <= 10);
14     cout<<"Sum Positive= "<<SP<<endl;
15     cout<<"Sum Negative= "<<SN<<endl;
16 }
```

مثال

خروجی:

برنامه ای بنویسید که ۱۰ عدد از کاربر خوانده مجموع اعداد مثبت و منفی را جداگانه محاسبه کرده و نمایش دهد.



مثال

مثال:

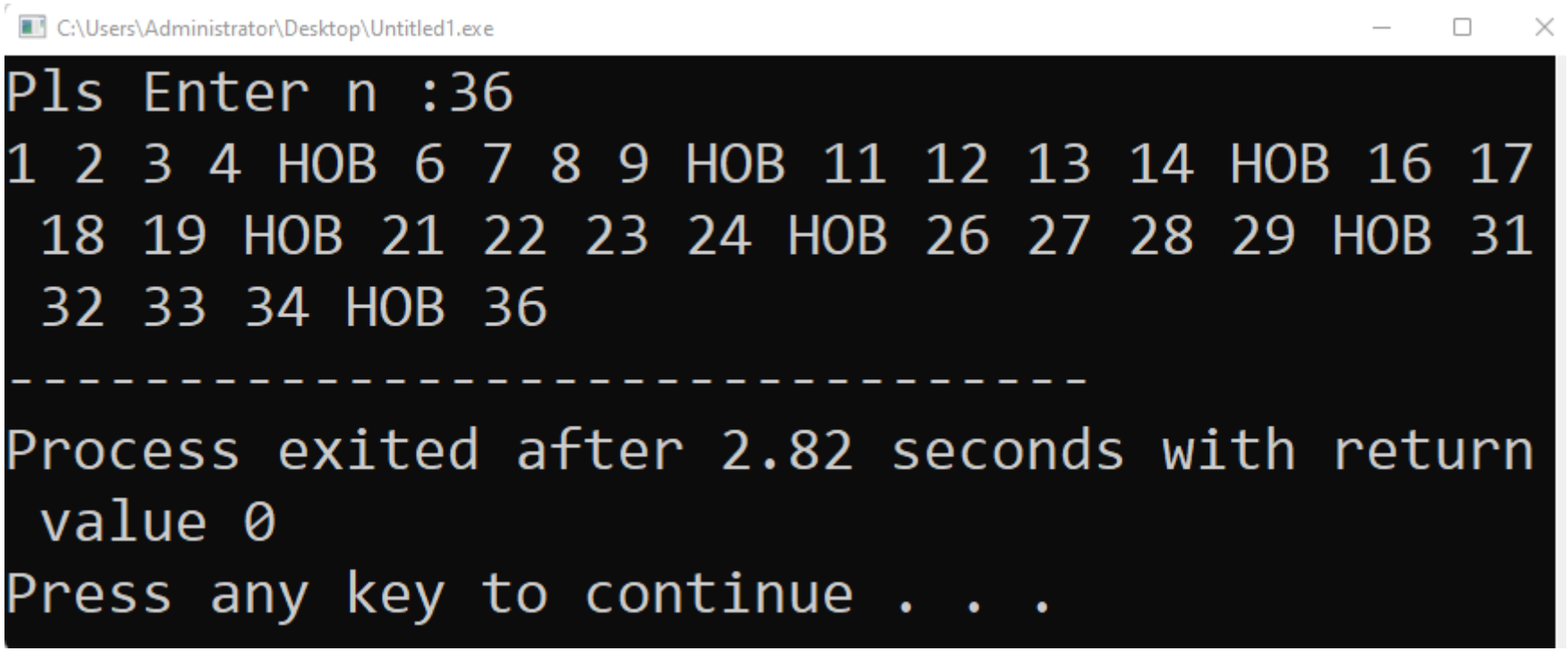
برنامه ای بنویسید مثل بازی HOB هر جا مضرب ۵ بود HOB بنویسد

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n;
6      string m;
7      cout<<"Pls Enter n :";
8      cin>>n;
9      for (int i=1;i<=n;i++){
10         (i%5)?cout<<i<<" ":cout<<"HOB ";
11     }
12 }
```

مثال

خروجی:

برنامه ای بنویسید مثل بازی HOB هر جا مضرب ۵ بود HOB بنویسد

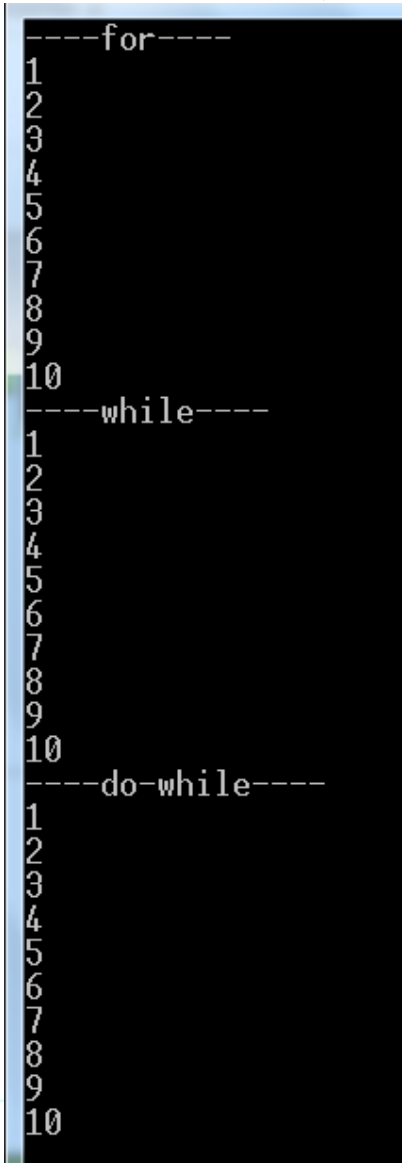


```
cout<<"----for----"<<endl;
for (int i=1;i<=10;i++)
    cout<<i<<endl;

cout<<"----while----"<<endl;
int i=1;
while(i<=10)
    cout<<i++<<endl;

cout<<"----do-while----"<<endl;
i=1;
do
    cout<<i++<<endl;
while(i<=10);
```

مقایسه:



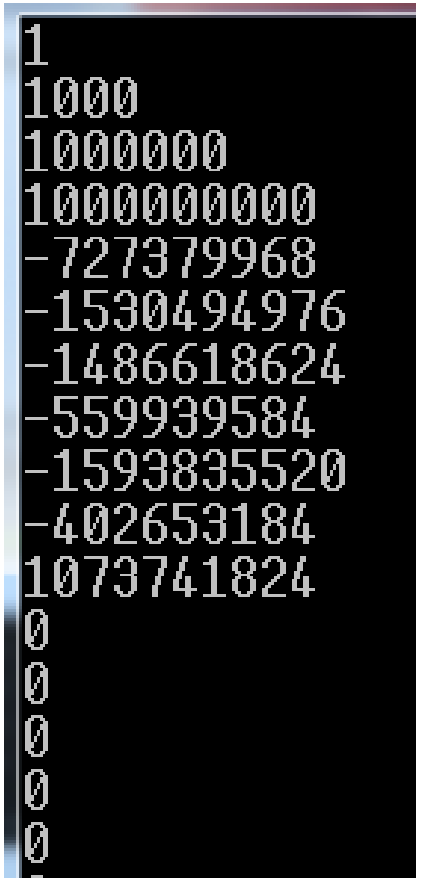
حلقه بی نهایت
Break & Continue

کدهای زیر موجب ایجاد حلقه بی نهایت می شوند!

بنابراین باید در نوشتن بدنه دستورات حلقه/تکرار دقت کرد

```
for (int i=1; ; i*=1000)
        cout<<i<<endl;
```

```
for (int i=1; true; i*=1000)
        cout<<i<<endl;
```



کدهای زیر موجب ایجاد حلقه بی نهایت می شوند!

بنابراین باید در نوشتن بدنه دستورات حلقه/تکرار دقت کرد

```
for (int i=1; i<1000; )
        cout<<i<<endl;
```

نکته

توجه داشته باشید، که هیچ کدام از سه قسمت داخل پرانتز، در دستور for اجباری نیستند. حتی دستور for، می‌تواند به‌صورت زیر نوشته شود که در این صورت، یک حلقه تمام نشدنی و بی‌نهایت ایجاد می‌شود.

```
for (; ; )

        cout<<“Hello"<<endl;
```

برنابراین اگر بخواهید حلقه متناهی و محدود داشته باشید، باید ساختار for را کامل بنویسید یا با استفاده از دیگر دستورات محدودیت لازم را ایجاد کنید.

Break

دستور **break** یک دستور آشناست. قبلا از آن برای خاتمه دادن به دستور **switch** استفاده کرده‌ایم.

از این دستور برای خاتمه دادن به حلقهٔ **for** نیز می‌توانیم استفاده کنیم.

دستور **break** در هر جایی درون حلقه می‌تواند جا بگیرد و در همان جا حلقه را خاتمه دهد.

وقتی دستور **break** درون حلقه‌های تودرتو استفاده شود، فقط روی حلقه‌ای که مستقیما درون آن قرار گرفته تاثیر می‌گذارد.

حلقه‌های بیرونی بدون هیچ تغییری ادامه می‌یابند.

Break

قبلا دیدیم که چگونه دستور **break** برای کنترل دستورالعمل **switch** استفاده می‌شود. از دستور **break** برای پایان دادن به حلقه‌ها نیز می‌توان استفاده کرد.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {   int n, i=1;
5      cout << "Enter a positive integer: ";
6      cin >> n;
7      long sum=0;
8      while (true)
9      {
10         if (i > n) break;
11         sum += i++;
12     }
13     cout << "The sum of the first " << n << " integers is " << sum;
14 }
```

یکی از مزیت‌های دستور break این است که فورا حلقه را خاتمه می‌دهد بدون این که مابقی دستورهای درون حلقه اجرا شوند.

Continue

دستور **break** بقیهٔ دستورهای درون بلوک حلقه را نادیده گرفته و به اولین دستور بیرون حلقه پرش می‌کند.

دستور **continue** نیز شبیه همین است اما به جای این که حلقه را خاتمه دهد، اجرا را به تکرار بعدی حلقه منتقل می‌کند.

این دستور، ادامهٔ چرخهٔ فعلی را لغو کرده و اجرای دور بعدی حلقه را آغاز می‌کند.

Continue

استفاده از دستورهای **break** و **continue**

این برنامهٔ کوچک، دستورهای **break** و **continue** را شرح می‌دهد:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {   int n = 1;
5      char c;
6      for( ; ;n++ )
7      {   cout << "\nLoop no: " << n << endl;
8          cout << "Continue? <y|n> ";
9          cin >> c;
10         if (c == 'y') continue;
11         break;
12     }
13     cout << "\nTotal of loops: " << n;
14 }
```

Continue

استفاده از دستورهای break و continue

این برنامهٔ کوچک، دستورهای break و continue را شرح می‌دهد:

```
Loop no: 1
Continue? <y!n> y

Loop no: 2
Continue? <y!n> y

Loop no: 3
Continue? <y!n> y

Loop no: 4
Continue? <y!n> y

Loop no: 5
Continue? <y!n> n

Total of loops: 5Press any key to continue . . .
```

اعداد فیبوناچی

اعداد فیبوناچی F0, F1, F2, F3, ... به شکل بازگشتی توسط معادله‌های زیر تعریف می‌شوند:

$$F0 = 0 \quad , \quad F1 = 1 \quad , \quad Fn = Fn-1 + Fn-2$$

مثلا برای $n=2$ داریم:

$$F2 = F2-1 + F2-2 = F1 + F0 = 0 + 1 = 1$$

یا برای $n=3$ داریم:

$$F3 = F3-1 + F3-2 = F2 + F1 = 1 + 1 = 2$$

و برای $n=4$ داریم:

$$F4 = F4-1 + F4-2 = F3 + F2 = 2 + 1 = 3$$

فیبوناچی

برنامهٔ زیر، همهٔ اعداد فیبوناچی را تا یک محدودهٔ مشخص که از ورودی دریافت می‌شود، محاسبه و چاپ می‌کند:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      long bound;
6      cout << "Enter a positive integer: ";
7      cin >> bound;
8      cout << "Fibonacci numbers < " << bound << ":\n0, 1";
9      long f0=0, f1=1;
10     while (true)
11     {
12       long f2 = f0 + f1;
13       if (f2 > bound) break;
14       cout << ", " << f2;
15       f0 = f1;
16       f1 = f2;
17     }
18     cout<<"end";
19 }
```

```
C:\Users\Administrator\Desktop\Untitled1.exe
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987end
--------------------------------
```

با تشکر از همراهی شما

محمد سعید صفایی صادق

www.SaeidSafaei.ir