فصل چهارم

«تکرار»

مقدمه

تکرار[1]، 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***[2] نیز نامیده می‌شوند.

4-1 دستور 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

---

1 – Iteration 2 - Loop

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. توجه کنید که شرط باید درون پرانتز قرار بگیرد.

× مثال 1-4 محاسبهٔ حاصل جمع اعداد صحیح متوالی با حلقهٔ while

این برنامه مقدار 1 + 2 + 3 + … + n را برای عدد ورودی n محاسبه می‌کند:

```
int main()
{  int n, i=1;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   while (i <= n)
      sum += i++;
   cout << "The sum of the first " << n << " integers is "
        << sum;
}
```

برنامهٔ بالا از سه متغیر محلی n و i و sum استفاده می‌کند. متغیر i با مقدار 1 مقداردهی اولیه می‌شود و عددی که کاربر وارد می‌کند در متغیر n قرار می‌گیرد. متغیر sum نیز با 0 مقداردهی اولیه می‌شود. سپس حلقهٔ while آغاز می‌گردد: ابتدا مقدار i با n مقایسه می‌شود. اگر i<=n بود مقدار i با مقدار sum جمع شده و حاصل در sum قرار می‌گیرد. به i یکی افزوده شده و دوباره شرط حلقه بررسی می‌شود. هنگامی که i>n شود حلقه متوقف می‌شود. پس n آخرین مقداری است که به sum افزوده می‌شود. شکل زیر پاسخ برنامه را به ازای ورودی n=8 نشان می‌دهد. همچنین مقدار متغیرها در هر گام حلقه در جدول نشان داده شده است.

```
Enter a positive integer: 8
The sum of the first 8 integers is 36
```

i	0	1	2	3	4	5	6	7	8
sum	0	1	3	6	10	15	21	28	36

در دومین اجرا، کاربر عدد 100 را وارد می‌کند، لذا حلقهٔ while نیز 100 بار تکرار می‌شود تا محاسبهٔ 1+2+3+...+98+99+100=5050 را انجام دهد:

```
Enter a positive integer: 100
The sum of the first 100 integers is 5050
```

به تورفتگی دستور داخل حلقه توجه کنید. این شکل چینش سبب می‌شود که منطق برنامه راحت‌تر دنبال شود، خصوصا در برنامه‌های بزرگ.

× مثال 2-4 استفاده از حلقهٔ while برای تکرار یک محاسبه

برنامهٔ زیر جذر هر عددی که کاربر وارد کند را محاسبه می‌نماید. در این برنامه از حلقهٔ while استفاده شده تا مجبور نباشیم برای محاسبهٔ جذر عدد بعدی، برنامه را دوباره اجرا کنیم:

```
int main()
{  double x;
   cout << "Enter a positive number: ";
   cin >> x;
   while (x > 0)
   {  cout << "sqrt(" << x << ") = " << sqrt(x) << endl;
      cout << "Enter another positive number (or 0 to quit): ";
      cin >> x;
   }
}
```

```
Enter a positive number: 49
sqrt(49) = 7
Enter another positive number (or 0 to quit): 3.14159
sqrt(3.14159) = 1.77245
Enter another positive number (or 0 to quit): 100000
sqrt(100000) = 316.228
Enter another positive number (or 0 to quit): 0
```

در این مثال، شرط کنترل حلقه عبارت **(x > 0)** است. مقدار x درون حلقه با تغییر عدد ورودی تغییر می‌کند. بنابراین فقط وقتی حلقه خاتمه می‌یابد که عدد ورودی برابر با 0 یا کم‌تر از آن باشد. متغیری که به این شکل برای کنترل حلقه استفاده شود، ***متغیر کنترل حلقه*** نامیده می‌شود.

4-2 خاتمه دادن به یک حلقه

قبلا دیدیم که چگونه دستور break برای کنترل دستورالعمل switch استفاده می‌شود (به مثال 3-17 نگاه کنید). از دستور break برای پایان دادن به حلقه‌ها نیز می‌توان استفاده کرد.

× مثال 4-3 استفاده از دستور break برای خاتمه دادن به یک حلقه

این برنامه همان تاثیر مثال 4-1 را دارد:

```
int main()
{  int n, i=1;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   while (true)
   {  if (i > n) break;     // terminates the loop immediately
      sum += i++;
   }
   cout << "The sum of the first " << n << " integers is " << sum;
}
```

```
Enter a positive integer: 100
The sum of the first 100 integers is 5050
```

برنامهٔ بالا مانند مثال 4-1 کار می‌کند: همین که مقدار i به n برسد، حلقه خاتمه می‌یابد و دستور خروجی در پایان برنامه اجرا می‌شود.

توجه کنید که شرط کنترل حلقه true است. به این ترتیب حلقه برای همیشه تکرار می‌شود و هیچ‌گاه پایان نمی‌یابد اما در بدنهٔ حلقه شرطی هست که سبب پایان گرفتن حلقه می‌شود. به محض این که i>n شود دستور break حلقه را می‌شکند و کنترل به بیرون حلقه پرش می‌کند. وقتی قرار است حلقه از درون کنترل شود، معمولا شرط کنترل حلقه را true می‌گذارند. با این روش عملا شرط کنترل حلقه حذف می‌شود.

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

× مثال 4-4 اعداد فیبوناچی

اعداد فیبوناچی ... F_0, F_1, F_2, F_3, به شکل بازگشتی توسط معادله‌های زیر تعریف می‌شوند:

$$F_0 = 0 \quad , \quad F_1 = 1 \quad , \quad F_n = F_{n-1} + F_{n-2}$$

مثلا برای n=2 داریم:

$$F_2 = F_{2-1} + F_{2-2} = F_1 + F_0 = 0 + 1 = 1$$

یا برای n=3 داریم:

$$F_3 = F_{3-1} + F_{3-2} = F_2 + F_1 = 1 + 1 = 2$$

و برای n=4 داریم:

$$F_4 = F_{4-1} + F_{4-2} = F_3 + F_2 = 2 + 1 = 3$$

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

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)
   {  long f2 = f0 + f1;
      if (f2 > bound) break;    // terminates the loop immediately
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
```

حلقهٔ while شامل بلوکی از پنج دستور است. وقتی شرط (f2 > bound) درست باشد، دستور break اجرا شده و بدون این که سه دستور آخر حلقه اجرا شوند، حلقه فورا پایان می‌یابد.

توجه کنید که از کاراکتر خط جدید '\n' در رشتهٔ "\n0,1:" استفاده شده. این باعث می‌شود که علامت کولن : در پایان خط فعلی چاپ شود و سپس مکان‌نما به خط بعدی روی صفحه‌نمایش پرش نماید و رشتهٔ 0,1 را در شروع آن خط چاپ کند.

× مثال 5-4 استفاده از تابع exit(0)

تابع exit(0) روش دیگری برای خاتمه دادن به یک حلقه است. هرچند که این تابع بلافاصله اجرای کل برنامه را پایان می‌دهد:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)
   {  long f2 = f0 + f1;
      if (f2 > bound) exit(0); // terminates the program immediately
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
```

برنامهٔ بالا پس از بدنهٔ حلقه هیچ دستور دیگری ندارد. پس خاتمه دادن حلقه به معنی پایان دادن برنامه است. به همین دلیل این برنامه مانند مثال 4-4 اجرا می‌شود.

این مثال یک راه برای خروج از حلقهٔ نامتناهی را نشان داد. مثال بعدی روش دیگری را نشان می‌دهد. اما برنامه‌نویسان ترجیح می‌دهند از break برای خاتمه دادن به حلقه‌های نامتناهی استفاده کنند زیرا قابلیت انعطاف بیشتری دارد.

× مثال 6-4 متوقف کردن یک حلقهٔ نامتناهی

اگر از راهکارهای خاتمهٔ حلقه استفاده نکنید، حلقه برای همیشه ادامه پیدا می‌کند و به طبع آن برنامه هم هیچ‌گاه به پایان نمی‌رسد. ممکن است شرط کنترلی که برای حلقه می‌نویسید هنگام اجرای برنامه هیچ‌گاه «نادرست» نشود و حلقه تا بی‌نهایت ادامه یابد. در چنین مواردی از سیستم عامل کمک بگیرید. با فشردن کلیدهای Ctrl+C سیستم عامل یک برنامه را به اجبار خاتمه می‌دهد. کلید Ctrl را پایین نگه داشته و کلید C روی صفحه‌کلید خود را فشار دهید تا برنامهٔ فعلی خاتمه پیدا کند. به کد زیر نگاه کنید:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)       // ERROR: INFINITE LOOP! Press <Ctrl>+c.)
   {  long f2 = f0 + f1;
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
159781, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 5040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352,
24157817, 63245986, 102334155, 165580141, 267914296, 433494437,
```

چون هیچ شرط پایان حلقه‌ای در این برنامه وجود ندارد، اجرای برنامه تا بی‌نهایت ادامه خواهد یافت (تا وقتی حافظه سرریز شود). پس کلیدهای Ctrl+C را فشار دهید تا برنامه خاتمه یابد.

4-3 دستور do..while

ساختار do..while روش دیگری برای ساختن حلقه است. نحو آن به صورت زیر است:

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

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.

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 4-7 محاسبهٔ حاصل جمع اعداد صحیح متوالی با حلقهٔ do..while

این برنامه همان تأثیر مثال 4-1 را دارد:

```
int main()
{  int n, i=0;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   do
      sum += i++;
   while (i <= n);
   cout << "The sum of the first " << n << " integers is " << sum;
```

```
}
```

× مثال 8-4 اعداد فاکتوریال

اعداد فاکتوریل !0 و !1 و !2 و !3 و ... با استفاده از رابطه‌های بازگشتی زیر تعریف می‌شوند:

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

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:

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

برنامهٔ زیر همهٔ اعداد فاکتوریال را که از عدد داده شده کوچک‌ترند، چاپ می‌کند:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Factorial numbers < " << bound << ":\n1";
   long f=1, i=1;
   do
   {  cout << ", " << f;
      f *= ++i;
   }
   while (f < bound);
}
```

```
Enter a positive integer: 100000
Factorial numbers < 100000:
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880
```

حلقهٔ do..while تا وقتی که شرط کنترل (**f < bound**) نادرست شود، تکرار می‌گردد.

4-4 دستور for

نحو دستورالعمل **for** به صورت زیر است:

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

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. بنابراین زنجیرهٔ وقایعی که تکرار را ایجاد می‌کنند عبارتند از:

1 – ارزیابی عبارت initialization

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

3 – اجرای statement

4 – ارزیابی عبارت update

5 – تکرار گام‌های 2 تا 4

عبارت‌های initialization و *condition* وupdate عبارت‌های اختیاری هستند. یعنی می‌توانیم آن‌ها را در حلقه ذکر نکنیم.

× مثال 9-4 استفاده از حلقهٔ for برای محاسبهٔ مجموع اعداد صحیح متوالی

این برنامه همان تأثیر مثال 1-4 را دارد:

```
int main()
{  int n;
   cout << "Enter a positive integer: ";
```

```
   cin >> n;
   long sum=0;
   for (int i=1; i <= n; i++)
      sum += I;
   cout << "The sum of the first " << n << " integers is " << sum;
}
```

در حلقهٔ برنامهٔ فوق، عبارت مقداردهی اولیه **int i=1** است. شرط کنترل حلقه **i<=n** می‌باشد و عبارت پیش‌بری متغیر کنترل هم **i++** است. دقت کنید که این‌ها همان عباراتی هستند که در برنامهٔ مثال‌های 1–4 و 3–4 و 7–4 استفاده شده است.

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 10–4 استفادهٔ مجدد از اسامی متغیرهای کنترل حلقهٔ for

برنامهٔ زیر همان اثر برنامهٔ مثال 1–4 را دارد:

```
int main()
{  int n;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   for (int i=1; i < n/2; i++)
      // the scope of this i is this loop
      sum += i;
   for (int i=n/2; i <= n; i++)
      // the scope of this i is this loop
      sum += i;
   cout << "The sum of the first " << n << " integers is " << sum ;
}
```

دو حلقهٔ for در برنامهٔ بالا همان محاسبات حلقهٔ for در برنامهٔ مثال 9–4 را انجام می‌دهند. این دو حلقه، کار را به دو قسمت تقسیم می‌کنند: n/2 محاسبه در حلقهٔ اول

انجام می‌گیرد و مابقی در حلقهٔ دوم. هر حلقه به طور مستقل متغیر کنترلی i خودش را دارد.

اخطار: بیشتر کامپایلرهای قبل از C++ استاندارد، حوزهٔ متغیر کنترلی حلقهٔ for را تا بعد از پایان حلقه نیز گسترش می‌دهند.

× مثال 11-4 دوباره اعداد فیبوناچی

این برنامه همان تأثیر برنامهٔ مثال 8-4 را دارد:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Factorial numbers < " << bound << ":\n1";
   long f=1;
   for (int i=2; f <= bound; i++)
   {  cout << ", " << f;
      f *= i;
   }
}
```

```
Enter a positive integer: 100000
Factorial numbers < 100000:
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880
```

برنامهٔ بالا را با مثال 8-4 مقایسه کنید. هر دو کارهای مشابهی را انجام می‌دهند. در هر دو برنامه، کارهای زیر انجام می‌شود: مقدار اولیهٔ 1 در f قرار می‌گیرد. مقدار اولیهٔ 2 در i قرار داده می‌شود و سپس پنج گام تکرار رخ می‌دهد: چاپ f، ضرب f در i، افزایش i، بررسی شرط (f <= bound) و پایان دادن به حلقه در صورت نادرست بودن شرط. این برنامه با حلقهٔ for همان تأثیر برنامه با حلقهٔ do..while را دارد.

دستور for انعطاف‌پذیری بیشتری به برنامه می‌دهد. مثال‌های زیر این مطلب را آشکار می‌کنند.

× مثال 12-4 یک حلقهٔ for نزولی

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

```
int main()
{  for (int i=10; i > 0; i--)
      cout << " " << i;
}
```

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

× مثال 13-4 استفاده از حلقهٔ for با گام‌های بزرگ‌تر از یک

برنامهٔ زیر مشخص می‌کند که آیا یک عدد ورودی اول هست یا خیر:

```
int main()
{  long n;
   cout << "Enter a positive integer: ";
   cin >> n;
   if (n < 2) cout << n << " is not prime." << endl;
   else if (n < 4) cout << n << " is prime." << endl;
   else if (n%2 == 0) cout << n << " = 2*" << n/2 << endl;
   else
   {  for (int d=3; d <= n/2; d+=2)
         if (n%d == 0)
         {  cout << n << " = " << d << "*" << n/d << endl;
            exit(0);
         }
      cout << n << " is prime." << endl;
   };
}
```

```
Enter a positive integer: 101
101 is prime.
```

```
Enter a positive integer: 975313579
975313579 = 17*57371387
```

توجه کنید که حلقهٔ **for** در برنامهٔ بالا متغیر کنترلی خود یعنی d را دو واحد دو واحد افزایش می‌دهد. سعی کنید منطق برنامهٔ بالا را توضیح دهید.

× مثال 14-4 استفاده از نگهبان برای کنترل حلقهٔ for

این برنامه مقدار بیشینهٔ یک رشته از اعداد ورودی را پیدا می‌کند:

```
int main()
{  int n, max;
   cout << "Enter positive integers (0 to quit): ";
   cin >> n;
   for (max = n; n > 0; )
   {  if (n > max) max = n;
      cin >> n;
   }
   cout << "max = " << max << endl;
}
```

```
Enter positive integers (0 to quit): 44 77 55 22 99 33 11 66 88 0
max = 99
```

حلقهٔ for در برنامهٔ بالا به وسیلهٔ متغیر ورودی n کنترل می‌شود. این حلقه ادامه می‌یابد تا زمانی که n<=0 بشود. متغیر ورودی که به این شیوه برای کنترل حلقه نیز استفاده شود، ***نگهبان*** نامیده می‌شود.

به بخش کنترلی این حلقه که به صورت **(max = n; n > 0;)** است دقت کنید. بخش پیش‌بری در آن وجود ندارد و بخش مقداردهی آن نیز متغیر جدیدی را تعریف نمی‌کند بلکه از متغیرهایی که قبلا در برنامه تعریف شده استفاده می‌برد. علت این است که حلقهٔ مذکور نگهبان دارد و نگهبان از طریق ورودی پیش برده می‌شود و دیگر نیازی به بخش پیش‌بری در حلقه نیست. متغیر max نیز باید مقدار خود را پس از اتمام حلقه حفظ کند تا در خروجی چاپ شود. اگر متغیر max درون حلقه اعلان می‌شد، پس از اتمام حلقه از بین می‌رفت و دیگر قابل استفاده نبود.

× مثال 15-4 بیشتر از یک متغیر کنترل در حلقهٔ for

حلقهٔ for در برنامهٔ زیر دو متغیر کنترل دارد:

```
int main()
{  for (int m=95, n=11, m%n > 0; m -= 3, n++)
      cout << m << "%" << n << " = " << m%n << endl;
}
```

```
95%11 = 7
92%12 = 8
89%13 = 11
```

```
86%14 = 2
83%15 = 8
```

در بخش کنترل این حلقه، دو متغیر m و n به عنوان متغیر کنترل اعلان و مقداردهی شده‌اند. در هر تکرار حلقه، m سه واحد کاسته شده و n یک واحد افزایش می‌یابد. در نتیجه زوج‌های (m,n) به شکل (95,11) و (92,12) و (89,13) و (86,14) و (83,15) و (80,16) تولید می‌شوند. چون 80 بر 16 بخش‌پذیر است، حلقه با زوج (80,16) پایان می‌یابد.

× مثال 16-4 حلقه‌های for تودرتو

برنامهٔ زیر یک جدول ضرب چاپ می‌کند:

```
#include <iomanip>    // defines setw()
#include <iostream>   // defines cout
using namespace std;
int main()
{  for (int x=1; x <= 10; x++)
   {  for (int y=1; y <= 10; y++)
         cout << setw(4) << x*y;
      cout << endl;
   }
}
```

```
  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
```

در اولین تکرار از حلقهٔ بیرونی، وقتی که x=1 است، حلقهٔ درونی ده مرتبه تکرار می‌شود و به ازای y=1 تا 10 مقادیر 1*y را روی یک ردیف چاپ می‌کند. وقتی حلقهٔ درونی پایان یافت، با دستور cout << endl; مکان‌نما به سطر بعدی روی صفحه‌نمایش منتقل می‌شود. حالا دومین تکرار حلقهٔ بیرونی به ازای x=2 آغاز

می‌شود. دوباره حلقهٔ درونی ده مرتبه تکرار می‌شود و این دفعه مقادیر 2*y روی یک سطر چاپ می‌شود. دوباره با دستور cout << endl; مکان‌نما به سطر بعدی می‌رود و تکرار سوم حلقهٔ بیرونی شروع می‌شود. این رویه ادامه می‌یابد تا این که حلقهٔ بیرونی برای بار دهم تکرار شده و آخرین سطر جدول هم چاپ می‌شود و سپس برنامه خاتمه می‌یابد.

در این برنامه از شکل‌دهندهٔ فرایند setw استفاده شده. عبارت setw(4) به این معنی است که طول ناحیهٔ چاپ را برای خروجی بعدی به اندازهٔ چهار کاراکتر تنظیم کن. به این ترتیب اگر خروجی کم‌تر از چهار کاراکتر باشد، فضای خالی به خروجی مربوطه پیوند زده می‌شود تا طول خروجی به اندازهٔ چهار کاراکتر شود. نتیجه این است که خروجی نهایی به شکل یک جدول مرتب روی ده سطر و ده ستون زیر هم چاپ می‌شود. شکل‌دهنده‌های فرایند در سرفایل <iomanip> تعریف شده‌اند. بنابراین برای استفاده از شکل‌دهنده‌های فرآیند باید راهنمای پیش‌پردازندهٔ #include <iomanip> را به ابتدای برنامه بیافزایید. همچنین برنامه باید دارای راهنمای پیش‌پردازندهٔ #include<iostream> نیز باشد.

4-5 دستور break

دستور **break** یک دستور آشناست. قبلا از آن برای خاتمه دادن به دستور switch و همچنین حلقه‌های while و do..while استفاده کرده‌ایم. از این دستور برای خاتمه دادن به حلقهٔ for نیز می‌توانیم استفاده کنیم. دستور break انعطاف‌پذیری بیشتری را برای حلقه‌ها ایجاد می‌کند. معمولا یک حلقهٔ while ، یک حلقهٔ do..while یا یک حلقهٔ for فقط در شروع یا پایان مجموعهٔ کامل دستورالعمل‌های موجود در بلوک حلقه، خاتمه می‌یابد. دستور break در هر جایی درون حلقه می‌تواند جا بگیرد و در همان جا حلقه را خاتمه دهد.

× مثال 4-17 کنترل ورودی با یک نگهبان

این برنامه یک رشته اعداد صحیح مثبت را تا زمانی که صفر وارد شود، خوانده و معدل آن‌ها را محاسبه می‌کند:

```
int main()
{  int n, count=0, sum=0;
   cout << "Enter positive integers (0 to quit):" << endl;
   for (;;)    // "forever"
   {  cout << "\t" << count + 1 << ": ";
      cin >> n;
      if (n <= 0) break;
      ++count;
      sum += n;
   }
   cout << "The average of those " << count << " positive
            numbers is "<< float(sum)/count << endl;
}
```

```
Enter positive integers (0 to quit):
        1: 4
        2: 7
        3: 1
        4: 5
        5: 2
        6: 0
The average of those 5 positive numbers is 3.8
```

در برنامهٔ بالا وقتی که 0 وارد شود، دستور break اجرا شده و حلقه فورا خاتمه می‌یابد و اجرای برنامه به اولین دستور بعد از حلقه پرش می‌کند. به نحوهٔ نوشتن دستور for در این برنامه دقت کنید. هر سه بخش کنترلی در این حلقه، خالی است: **for(; ;)**. این ترکیب به معنای بی‌انتهایی است. یعنی بدون دستور break این حلقه یک حلقهٔ نامتناهی می‌شود.

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.

× مثال 18-4 استفاده از دستور break در حلقه‌های تودرتو

چون عمل ضرب جابجایی‌پذیر است (یعنی 3×4 = 4×3)، برای ایجاد یک جدول ضرب فقط کافی است اعداد قطر پایینی مشخص شوند. این برنامه، مثال 16-4 را برای چاپ یک جدول ضرب مثلثی تغییر می‌دهد:

```
int main()
{  for (int x=1; x <= 10; x++)
   {  for (int y=1; y <= 10; y++)
         if (y > x) break;
         else cout << setw(4) << x*y;
      cout << endl;
   }
}
```

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

وقتی y>x باشد، اجرای حلقهٔ y درونی خاتمه می‌یابد و تکرار بعدی حلقهٔ خارجی x شروع می‌شود. مثلا وقتی x=3 باشد، حلقهٔ y سه بار تکرار می‌شود و خروجی‌های 3 و 6 و 9 چاپ می‌شوند. در تکرار چهارم، شرط (y>x) برابر با درست ارزیابی می‌شود. پس دستور break اجرا شده و کنترل فورا به خط cout<<endl; منتقل می‌شود (زیرا این خط اولین دستور خارج از حلقهٔ درونی y است). آنگاه حلقهٔ بیرونی x تکرار چهارم را با x=4 آغاز می‌کند.

4-6 دستور 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.

× مثال 19-4 استفاده از دستورهای break و continue

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

```
int main()
{  int n = 1;
   char c;
   for( ; ;n++ )
   {  cout << "\nLoop no: " << n << endl;
      cout << "Continue? <y|n> ";
      cin >> c;
      if (c = = 'y') continue;
      break;
   }
   cout << "\nTotal of loops: " << n;
}
```

```
Loop no: 1
Continue? y
Loop no: 2
Continue? y
Loop no: 3
Continue? n
Total of loops: 3
```

برنامهٔ بالا تعداد تکرار حلقه را می‌شمارد. در ابتدای هر حلقه با چاپ n مشخص می‌شود که چندمین دور حلقه در حال اجراست. سپس از کاربر درخواست می‌شود تا یک کاراکتر را به عنوان انتخاب، وارد کند. اگر کاراکتر وارد شده 'y' باشد، شرط (c=='y') برابر با درست ارزیابی می‌شود و لذا دستور continue اجرا شده و دور جدید حلقه شروع می‌شود. اگر کاراکتر وارد شده هر چیزی غیر از 'y' باشد، دستور break این حلقه را خاتمه می‌دهد و کنترل اجرا به اولین دستور بیرون حلقه پرش می‌کند. سپس مجموع دفعاتی که حلقه تکرار شده چاپ می‌گردد و برنامه پایان می‌گیرد.

4-7 دستور goto

دستورهای break و continue و switch باعث می‌شوند که اجرای برنامه به مکان دیگری از جایی که به طور طبیعی باید می‌رفت، منتقل شود. مقصد انتقال را نوع دستور تعیین می‌کند: break به خارج از حلقه می‌رود، continue به شرط ادامهٔ حلقه(دور بعدی حلقه) می‌رود و switch به یکی از ثابت‌های case می‌رود. هر سهٔ این دستورها ***دستور پرش***[1] هستند زیرا باعث می‌شوند اجرای برنامه از روی دستورهای دیگر پرش کند.

دستور**goto** نوع دیگری از دستورهای پرش است. مقصد این پرش توسط یک برچسب معین می‌شود. ***برچسب***[2] شناسه‌ای است که جلوی آن علامت کولن(:) می‌آید و جلوی یک دستور دیگر قرار می‌گیرد. برچسب‌ها شبیه caseها در دستور switch هستند، یعنی مقصد پرش را مشخص می‌کنند. یک مزیت دستور **goto** این است که با استفاده از آن می‌توان از همهٔ حلقه‌های تودرتو خارج شد و به مکان دلخواهی در برنامه پرش نمود.

مثال زیر نشان می‌دهد که دستور break فقط درونی‌ترین حلقه را خاتمه می‌دهد ولی با دستور **goto** می‌توان چند حلقه یا همهٔ حلقه‌ها را یک‌جا خاتمه داد.

× مثال 4-20 استفاده از دستور goto برای خارج شدن از حلقه‌های تودرتو

```
int main()
{  const int N=5;
   for (int i=0; i<N; i++)
   {  for (int j=0; j<N; j++)
      {  for (int k=0; k<N; k++)
            if (i+j+k>N) goto esc;
            else cout << i+j+k << " ";
         cout << "* ";
      }
   esc: cout << "." << endl;   // inside the i loop, outside the j loop
   }
}
```

```
0 1 2 3 4 * 1 2 3 4 5 * 2 3 4 5 .
```

```
1 2 3 4 5 * 2 3 4 5 .
2 3 4 5 .
3 4 5 .
4 5 .
```

1 – Jump 2 - Label

اگر شرط (i+j+k>N) در درونی‌ترین حلقه درست شود، به دستور goto می‌رسیم. وقتی این دستور اجرا شود، اجرای برنامه به سطری که برچسب **esc** دارد منتقل می‌شود. این خط بیرون از حلقه‌های j و k است. پس با این پرش هر دوی این حلقه‌ها پایان می‌گیرند. وقتی i و j صفر هستند، حلقهٔ k پنج بار تکرار می‌گردد و 4 3 2 1 0 به همراه ستاره × چاپ می‌شود. آنگاه j به 1 افزایش می‌یابد و حلقهٔ k پنج بار دیگر تکرار شده و این بار 5 4 3 2 1 همراه یک ستاره × چاپ می‌شود. سپس j به 2 افزایش می‌یابد و حلقهٔ k چهار بار دیگر تکرار می‌شود و 5 4 3 2 چاپ می‌شود. اما در تکرار بعدی حلقهٔ k شرط (i+j+k>0) درست می‌شود زیرا حالا i+j+k = 6 است. پس دستور **goto** برای اولین بار اجرا شده و کنترل برنامه به سطر برچسب‌دار که یک دستور خروجی است پرش می‌کند، یک نقطه چاپ شده و مکان‌نما به سطر بعد منتقل می‌شود. توجه کنید که حلقه‌های j و k بدون این که تکرارهایشان را کامل کنند، ناتمام رها می‌شوند. دستور برچسب خورده، خود جزو بدنهٔ حلقهٔ i است. لذا پس از پایان گرفتن اجرای این سطر، تکرار بعدی حلقهٔ i آغاز می‌شود. حالا i=1 است و حلقهٔ j دوباره با j=0 شروع می‌شود و این خود باعث می‌گردد حلقهٔ k نیز با k=0 دوباره شروع شود. برنامه به همین ترتیب ادامه می‌یابد و خروجی نهایی حاصل می‌شود.

با استفاده از دستور **goto** می‌توان از هر قسمت برنامه به هر قسمت دیگری پرش کرد. گرچه این دستور باعث می‌شود راحت‌تر بتوانیم از تکرار حلقه‌ها خلاص شویم یا آسان‌تر به سطر دلخواه انشعاب کنیم اما تجربه نشان داده که استفادهٔ بی‌مهابا از دستور **goto** سبب افزایش خطاهای زمان اجرا و کاهش پایداری برنامه می‌شود.

× مثال 21–4 خارج شدن از چند حلقهٔ تودرتو بدون استفاده از goto

این برنامه همان اثر برنامهٔ مثال 20-4 را دارد:

```
int main()
{  const int N=5;
   bool done=false;
   for (int i=0; i<N; i++)
   {  for (int j=0; j<N && !done; j++)
      {  for (int k=0; k<N && !done; k++)
            if (i+j+k>N) done = true;
            else cout << i+j+k << " ";
         cout << "* ";
      }
      cout << "." << endl;    // inside the i loop, outside the j loop
      done = false;
   }
}
```

در برنامهٔ بالا از یک پرچم به نام done استفاده شده. وقتی done برابر با true شود، هر دو حلقهٔ درونی k و j خاتمه می‌یابد و حلقهٔ خارجی i تکرارش را با چاپ یک نقطه ادامه می‌دهد و پرچم را دوباره false می‌کند و دور جدید را آغاز می‌نماید. گرچه منطق این برنامه کمی پیچیده‌تر است اما قابلیت اطمینان بیشتری نسبت به برنامهٔ مثال 20-4 دارد زیرا هیچ حلقه‌ای نیمه‌کاره نمی‌ماند و هیچ متغیری بلاتکلیف رها نمی‌شود.

4-8 تولید اعداد شبه تصادفی[1]

یکی از کاربردهای بسیار مهم رایانه‌ها، «شبیه‌سازی[2]» سیستم‌های دنیای واقعی است. تحقیقات و توسعه‌های بسیار پیشرفته به این راهکار خیلی وابسته است. به وسیلهٔ شبیه‌سازی می‌توانیم رفتار سیستم‌های مختلف را مطالعه کنیم بدون این که لازم باشد واقعا آن‌ها را پیاده‌سازی نماییم. در شبیه‌سازی نیاز است «اعداد تصادفی» توسط رایانه‌ها تولید شود تا نادانسته‌های دنیای واقعی مدل‌سازی شود. البته رایانه‌ها «ثابت‌کار» هستند یعنی با دادن داده‌های مشابه به رایانه‌های مشابه، همیشه خروجی یکسان تولید می‌شود. با وجود این می‌توان اعدادی تولید کرد که به ظاهر تصادفی هستند؛ اعدادی که

به طور یکنواخت در یک محدودهٔ خاص گسترده‌اند و برای هیچ‌کدام الگوی مشخصی وجود ندارد. چنین اعدادی را «اعداد شبه‌تصادفی» می‌نامیم.

سرفایل <cstdlib> در c استاندارد دارای تابعی به نام rand() است که این تابع اعداد صحیح شبه‌تصادفی در محدودهٔ صفر تا RAND_MAX تولید می‌نماید.

---

1 – Pseudo Random 2 - Simulation

RAND_MAX ثابتی است که آن هم در سرفایل <cstdlib> تعریف شده. هر بار که تابع rand() فراخوانی شود، یک عدد صحیح متفاوت از نوع unsigned تولید می‌کند که این عدد در محدودهٔ ذکر شده قرار دارد.

× مثال 22-4 تولید اعداد شبه تصادفی

این برنامه از تابع rand() برای تولید اعداد شبه‌تصادفی استفاده می‌کند:

```
#include <cstdlib>  // defines the rand() and RAND_MAX
#include <iostream>

int main()
{   // prints pseudo-random numbers:
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
   cout << "RAND_MAX = " << RAND_MAX << endl;
}
```

```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

هر بار که برنامهٔ بالا اجرا شود، رایانه هشت عدد صحیح unsigned تولید می‌کند که به طور یکنواخت در فاصلهٔ 0 تا RAND_MAX گسترده شده‌اند. RAND_MAX در این

رایانه برابر با 2,147,483,647 است. خروجی زیر، اجرای دیگری از برنامهٔ بالا را نشان می‌دهد:

```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

متاسفانه هر بار که برنامه اجرا می‌شود، همان اعداد قبلی تولید می‌شوند زیرا این اعداد از یک هستهٔ مشترک ساخته می‌شوند.

هر عدد شبه‌تصادفی از روی عدد قبلی خود ساخته می‌شود. اولین عدد شبه‌تصادفی از روی یک مقدار داخلی که «هسته[1]» گفته می‌شود ایجاد می‌گردد. هر دفعه که برنامه اجرا شود، هسته با یک مقدار پیش‌فرض بارگذاری می‌شود. برای حذف این اثر نامطلوب که از تصادفی بودن اعداد می‌کاهد، می‌توانیم با استفاده از تابع srand() خودمان مقدار هسته را انتخاب کنیم.

× مثال 23-4 کارگذاری هسته به طور محاوره‌ای

این برنامه مانند برنامهٔ مثال 22-4 است بجز این که می‌توان هستهٔ تولیدکنندهٔ اعداد تصادفی را به شکل محاوره‌ای وارد نمود:

```
#include <cstdlib>  // defines the rand() and srand()
#include <iostream>

int main()
{   // prints pseudo-random numbers:
   unsigned seed;
   cout << "Enter seed: ";
   cin >> seed;
   srand(seed);      // initializes the seed
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
}
```

سه اجرای متفاوت از برنامهٔ بالا نشان داده شده است:

```
Enter seed: 0
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
```

1 – Seed

```
Enter seed: 1
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
```

```
Enter seed: 12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
```

خط `srand(seed);` مقدار متغیر `seed` را به هستهٔ داخلی تخصیص می‌دهد. این مقدار توسط تابع `rand()` برای تولید اعداد شبه‌تصادفی استفاده می‌شود. هسته‌های متفاوت، نتایج متفاوتی را تولید می‌کنند.

توجه کنید که مقدار متغیر `seed` که در سومین اجرای برنامه استفاده شده (`12345`) اولین عددی است که توسط تابع `rand()` در اجرای اول تولید شده بود. در نتیجه اعداد اول تا هشتم که در اجرای سوم تولید شده با اعداد دوم تا نهم که در اجرای اول تولید شده بود برابر است. همچنین دقت کنید که رشته اعداد تولید شده در

اجرای دوم مانند رشته اعداد تولید شده در مثال 22-4 است. این موضوع القا می‌کند که مقدار پیش‌فرض هسته در این رایانه، عدد یک است.

این که مقدار هسته باید به طور محاوره‌ای وارد شود مشکلی است که با استفاده از ساعت سیستم حل می‌شود. «ساعت سیستم[1]» زمان فعلی را بر حسب ثانیه نگه می‌دارد. تابع time() که در سرفایل <ctime> تعریف شده زمان فعلی را به صورت یک عدد صحیح unsigned برمی‌گرداند. این مقدار می‌تواند به عنوان هسته برای تابع rand() استفاده شود.

1 – System timer

× مثال 24-4 کارگذاری هسته از ساعت سیستم

برنامهٔ زیر، همان برنامهٔ مثال 23-4 است با این فرق که هستهٔ تولیدکنندهٔ اعداد شبه‌تصادفی را با استفاده از ساعت سیستم تنظیم می‌کند.

توجه: اگر کامپایلر شما سرفایل <ctime> را تشخیص نمی‌دهد، به جای آن از سرفایل <time.h> استفاده کنید.

```
#include <cstdlib>
#include <ctime>      // defines the time() function
#include <iostream>
//#include <time.h>   // use this if <ctime> is not recognized
int main()
{   // prints pseudo-random numbers:
   unsigned seed = time(NULL);      // uses the system clock
   cout << "seed = " << seed << endl;
   srand(seed);                     // initializes the seed
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
}
```

```
seed = 808148157
1877361330
352899587
1443923328
1857423289
```

```
200398846
1379699551
1622702508
715548277
```

```
seed = 808148160
892939769
1559273790
1468644255
952730860
1322627253
844657339
440402904
```

در اولین اجرا، تابع time() عدد صحیح 808,148,157 را برمی‌گرداند که به عنوان هستهٔ تولید کنندهٔ اعداد تصادفی استفاده شده است. دومین اجرا 3 ثانیه بعد انجام شده، بنابراین تابع time() عدد صحیح 808,148,160 را برمی‌گرداند که این مقدار، رشته اعداد کاملا متفاوتی را تولید می‌کند.

دو اجرای زیر روی یک pc با پردازندهٔ intel انجام شده است:

```
seed = 943364015
2948
15841
72
25506
30808
29709
13155
2527
```

```
seed = 943364119
17427
20464
13149
5702
12766
1424
16612
31746
```

در بیشتر برنامه‌های کاربردی، نیاز است که اعداد تصادفی در محدودهٔ مشخصی پخش شده باشند. مثال بعدی طریقهٔ انجام این کار را نشان می‌دهد.

× مثال 25-4 تولید اعداد تصادفی در یک محدودهٔ مشخص

برنامهٔ زیر مانند مثال 24-4 است به جز این که اعدادی که برنامهٔ زیر تولید می‌کند در یک ناحیهٔ مشخص محدود شده:

```
#include <cstdlib>
#include <ctime>      // defines the time() function
#include <iostream>
//#include <time.h>   // use this if <ctime> is not recognized
int main()
{  // prints pseudo-random numbers:
   unsigned seed = time(NULL);     // uses the system clock
   cout << "seed = " << seed << endl;
   srand(seed);                    // initializes the seed
   int min, max;
   cout << "Enter minimum and maximum: ";
   cin >> min >> max;           // lowest and highest numbers
   int range = max - min + 1;   // number of numbers in rsnge
   for (int i = 0; i < 20; i++)
   {  int r = rand()/100%range + min;
      cout << r << " ";
   }
   cout << endl;
}
```

```
seed = 808237677
Enter minimum and maximum: 1 100
85 57 1 10 5 73 81 43 46 42 17 44 48 9 3 74 41 4 30 68
```

```
seed = 808238101
Enter minimum and maximum: 22 66
63 29 56 22 53 57 39 56 43 36 62 30 41 57 26 61 59 26 28
```

اولین اجرا 20 عدد صحیح تصادفی بین 1 و 100 تولید می‌کند. دومین اجرا 20 عدد صحیح که بین 22 و 66 گسترش یافته را تولید می‌نماید.

در حلقهٔ for ابتدا مقدار rand() بر 100 تقسیم می‌شود تا دو رقم سمت راست عدد تصادفی حذف شود زیرا این مولد به طور متناوب اعداد زوج و فرد تولید

می‌کند. با حذف دو رقم سمت راست عدد تولید شده، این مشکل برطرف می‌شود. سپس عبارت rand()/100%range اعداد تصادفی در محدودهٔ 0 تا range-1 تولید نموده و عبارت rand()/100%range + min اعدادی تصادفی در محدودهٔ min تا max تولید می‌نماید.

پرسش‌های گزینه‌ای

1 – کدام یک از دستورات زیر، یک حلقه نیست؟

الف – while ب – switch

ج – for د – do..while

2 – در مورد کد `while (false) i++;` **کدام گزینه صحیح است؟**

الف – حلقهٔ فوق تا بی‌نهایت ادامه می‌یابد و مقدار i بی‌نهایت می‌شود.

ب – حلقهٔ فوق آن قدر ادامه می‌یابد تا این که i سرریز شود.

ج – حلقهٔ فوق فقط یک بار اجرا می‌شود و i فقط یک واحد اضافه می‌شود.

د – حلقهٔ فوق اصلا اجرا نمی‌شود.

3 – اگر i از نوع صحیح و مقدار آن 5 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار i برابر با کدام گزینه است؟

```
While (i<10) i+=2;
```

الف – 10 ب – 11 ج – 9 د – 12

4 – اگر n از نوع صحیح و مقدار آن 0 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار n برابر با کدام گزینه است؟

```
While (n<5)
{  if (n>3) break;
   n++;
}
```

الف – 4 ب – 5 ج – 6 د – 3

5 – اگر k از نوع صحیح و مقدار آن 5 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار k برابر با کدام گزینه است؟

```
do
   k++
while (k<5);
```

الف – 5 ب – 6 ج – k سرریز می شود د – k پاریز می شود.

6 – کدام گزینه صحیح نیست؟

الف– دستور break حلقه را خاتمه می‌دهد.

ب – دستور exit(0) حلقه را خاتمه می‌دهد.

ج – دستور continue حلقه را خاتمه می‌دهد.

د – غلط بودن شرط کنترل، حلقه را خاتمه می‌دهد.

7 – حلقهٔ مقابل چند بار تکرار می‌شود؟ `for (j=0; true; j++) {…}`

الف – اصلا تکرار نمی‌شود

ب – تا بی‌نهایت ادامه می‌یابد

ج – فقط یک بار تکرار می‌شود

د – اگر در بدنهٔ حلقه، دستور خاتمهٔ حلقه وجود نداشته باشد تا بی‌نهایت ادامه می‌یابد.

8 – در پایان حلقه‌های مقابل مقدار k چقدر است؟

```
k = 0;
for (i=0; i<5; i++)
   for (j=0; j<5; j++)
      k++;
```

الف – 5 ب – 10 ج – 25 د – 50

9 – کدام گزینه صحیح است؟

الف – حلقهٔ do..while دست کم یک بار اجرا می‌شود.

ب – حلقهٔ while دست کم یک بار اجرا می‌شود.

ج – حلقهٔ for دست کم یک بار اجرا می‌شود

د – حلقهٔ while و حلقهٔ for دست کم یک بار اجرا می‌شوند.

10 – در کدام حلقه، شرط کنترل حلقه در انتهای هر تکرار بررسی می‌شود؟

الف – حلقهٔ while ب – حلقهٔ do..while ج – حلقهٔ for د – هیچ‌کدام

11 – کدام حلقه نمی‌تواند تا بی‌نهایت ادامه یابد؟

الف – حلقهٔ while ب – حلقهٔ do..while ج – حلقهٔ for د – هیچ‌کدام

12 – در مورد حلقه‌های تودرتو کدام عبارت صحیح است؟

الف – دستور break فقط درونی‌ترین حلقه را خاتمه می‌دهد

ب – دستور break فقط بیرونی‌ترین حلقه را خاتمه می‌دهد

ج – دستور break فقط حلقه‌ای که این دستور در بدنهٔ آن قرار دارد را خاتمه می‌دهد

د – دستور break حلقه‌های درونی و بیرونی را یک‌جا خاتمه می‌دهد.

13 – کدام عبارت صحیح است؟

الف – حلقهٔ for را می‌توان به حلقهٔ while یا حلقهٔ do..while تبدیل کرد.

ب – حلقهٔ for را نمی‌توان به حلقهٔ while یا حلقهٔ do..while تبدیل کرد.

ج – حلقهٔ for را فقط می‌توان به حلقهٔ while تبدیل کرد.

د – حلقهٔ for را فقط می‌توان به حلقهٔ do..while تبدیل کرد.

14 – اگر به جای شرط کنترل اجرای یک حلقه، عبارت true قرار دهیم آنگاه:

الف – حلقه اصلا اجرا نمی‌شود

ب – حلقه حتما تا بی‌نهایت ادامه می‌یابد

ج – تعداد تکرارها بستگی به دستورات بدنه دارد

د – کامپایلر خطا می‌گیرد

15 – اگر i متغیری از نوع bool با مقدار false باشد آنگاه حلقهٔ مقابل چند بار تکرار می‌شود؟

```
while (!i) i=true;
```

الف – اصلا اجرا نمی‌شود ب – یک بار اجرا می‌شود

ج – تا بی‌نهایت ادامه می‌یابد د – تا وقتی حافظه سرریز شود ادامه می‌یابد

16 – دستور continue در حلقه‌ها چه کاری انجام می‌دهد؟

الف – حلقه را در همان محل خاتمه می‌دهد

ب – مابقی دستورات بدنهٔ حلقه را نادیده گرفته و تکرار بعدی حلقه را آغاز می‌کند

ج – مابقی دستورات بدنهٔ حلقه را نادیده گرفته و حلقه را خاتمه می‌دهد

د – تمام دستورات تکرار فعلی را اجرا نموده و سپس حلقه را خاتمه می‌دهد

17 – در رابطه با بخش initializing یا مقداردهی اولیه در حلقهٔ for کدام عبارت صحیح نیست؟

الف – این بخش فقط یک بار ارزیابی می‌شود

ب – این بخش قبل از این که تکرارها آغاز شوند ارزیابی می‌شود

ج – این بخش می‌تواند در حلقهٔ for قید نشود

د – در حلقه‌های for تودرتو این بخش حذف می‌شود

پرسش‌های تشریحی

1- در یک حلقهٔ while اگر شرط کنترل در ابتدا با مقدار false (یعنی صفر) مقداردهی شود، چه اتفاقی می‌افتد؟

2- چه وقت باید متغیر کنترل در حلقهٔ for قبل از حلقه اعلان گردد (به جای این که داخل بخش کنترلی آن اعلان گردد)؟

3- دستور break چگونه باعث کنترل بهتر روی حلقه‌ها می‌شود؟

4- حداقل تکرار در:

الف - یک حلقهٔ while چقدر است؟

ب - یک حلقهٔ do..while چقدر است؟

5- چه اشتباهی در حلقهٔ زیر است؟

```
while (n <= 100)
 sum += n*n;
```

6- چه خطایی در برنامهٔ زیر است؟

```
int main()
{  const double PI;
   int n;
   PI = 3.14159265358979
   n = 22;
}
```

7- «حلقهٔ بی‌پایان» چیست و چه فایده‌ای دارد؟

8- چطور می‌توان حلقه‌ای ساخت که با یک دستور در وسط بلوکش پایان یابد؟

9- چرا از به‌کارگیری متغیرهای ممیز شناور در مقایسه‌های برابری باید اجتناب شود؟

تمرین‌های برنامه‌نویسی

1- قطعه برنامهٔ زیر را دنبال نمایید و مقدار هر متغیر را در هر گام مشخص کنید:

```
float x = 4.15;
for (int i=0; i < 3; i++)
   x *= 2;
```

2- حلقهٔ for زیر را به حلقهٔ while تبدیل کنید:

```
for (int i=1; i <= n; i++)
 cout << i*i << " ";
```

3- خروجی این برنامه را توضیح دهید:

```
int main()
{  for (int i = 0; i < 8; i++)
      if ( i%2 == 0) cout << i + 1 << "\t";
      else if (i%3 == 0) cout << i*i << "\t";
      else if (i%5 == 0) cout << 2*i - 1 << "\t";
      else cout << i << "\t";
}
```

4- خروجی برنامهٔ زیر را توضیح دهید:

```
int main()
{  for (int i=0; i < 8; i++)
   {  if (i%2 == 0) cout << i + 1 << endl;
      else if (i%3 == 0) continue;
      else if (i%5 == 0) break;
      cout << "End of program.\n";
   }
   cout << "End of program.\n";
}
```

5- برنامه‌ای نوشته و اجرا کنید که عددی را از ورودی گرفته و با استفاده از یک حلقهٔ while مجموع مربعات اعداد متوالی تا آن عدد را پیدا کند. برای مثال اگر 5 وارد شود، برنامه مذکور عدد 55 را چاپ کند که معادل $5^2+4^2+3^2+2^2+1^2$ است.

6- پاسخ سوال 5 را با یک حلقهٔ for نوشته و اجرا کنید.

7- پاسخ سوال 5 را با یک حلقهٔ do..while نوشته و اجرا کنید.
8- برنامه‌ای را نوشته و اجرا کنید که اعمال تقسیم و باقیمانده را بدون استفاده از عملگرهای / و % برای تقسیم اعداد صحیح مثبت پیاده‌سازی می‌کند.
9- برنامه‌ای را نوشته و اجرا کنید که ارقام یک عدد مثبت داده شده را معکوس می‌کند. (به تمرین 13 فصل سوم نگاه کنید)
10- برنامه‌ای بنویسید که ریشهٔ صحیح یک عدد داده شده را پیدا کند. ریشهٔ صحیح، بزرگ‌ترین عدد صحیحی است که مربع آن کوچک‌تر یا مساوی عدد داده شده باشد.
11 – با استفاده از الگوریتم اقلیدس، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح داده شده را بیابید. این الگوریتم به وسیلهٔ تقسیم‌های متوالی، زوج (m,n) را به زوج (n,0) تبدیل می‌کند. به این صورت که عدد صحیح بزرگ‌تر را بر عدد کوچک‌تر تقسیم کرده و سپس به جای عدد بزرگ‌تر، عدد کوچک‌تر را قرار می‌دهد و به جای عدد کوچک‌تر، باقیمانده تقسیم را قرار می‌دهد و دوباره تقسیم را روی این زوج جدید تکرار می‌کند. وقتی باقیمانده برابر با صفر شود، عدد دیگر از آن زوج، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح اولیه است (و همچنین بزرگ‌ترین مقسوم‌علیه مشترک تمام زوج‌های میانی). برای مثال اگر m برابر با 532 و n برابر با 112 باشد، الگوریتم اقلیدس زوج (532,112) را به ترتیب زیر به زوج (28,0) تبدیل می‌کند:

(532,112) ⇨ (112,84) ⇨ (84,28) ⇨ (28,0).

برنامه‌ای بنویسید که با استفاده از الگوریتم اقلیدس، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح داده شده را بیابد.