

Chapter 20 – For Statement Loops

1. Repetition

The fundamental reason that computers are used to do so many things is their ability to repeatedly apply a set of instructions to each item of new data, no matter how many new items of data are presented. Word processing software will patiently echo all typed user input to the monitor screen for a 100-word paper or a 20,000-word novel. Banking software will perform the same actions on each check presented to the bank, thousands of times per day, millions of times per year. Repetition is such an intrinsic part of computer programming that it is an axiom in computer science that *any process that involves tedious repetition is a candidate for computation*. The act of repetition is referred to as a *loop* or *iteration*.

2. Counters

Loops are often executed until some variable reaches a pre-determined value. C++ contains several forms of statements called counters to increment variables in a given direction.

++ & --

To increment an integer variable object by one, use one of the following statement forms:

```
integer-variable++;
++integer-variable;
```

If *a* is an integer variable with the value of 4, the statement

```
a++;
```

increases the value of *a* to 5.

If *b* is an integer variable with the value of 9, the statement

```
++b;
```

increases the value of *b* to 10.

To decrement an integer variable object by one, use one of the following statement forms:

```
integer-variable--; // notice the use of two minus signs
--integer-variable;
```

If *a* is an integer variable with the value of 4, the statement

```
a--;
```

decreases the value of *a* to 3.

If b is an integer variable with the value of 9, the statement

`--b`

decreases the value of b to 8.

Traditional Counters

Almost all languages allow counters in the following forms:

variable = variable operator expression

where *variable* is the same object on both sides of the assignment operator

This is possible because everything to the right of the assignment operator is evaluated first, then the result assigned to the variable on the left. Here are some examples:

```
a = a + 3;    // increases the value of a by 3
b = b - 1;    // decreases the value of b by 1
c = c * 2;    // doubles the value of c
d = d / 5;    // reduces the value of d to one fifth of the original value
```

C++ Counter Abbreviations

C++ allows traditional counters to be written in the following shorter form:

variable operator= expression;

The four examples of traditional counters can be written as follows:

```
a += 3;      // increases the value of a by 3
b -= 1;      // decreases the value of b by 1
c *= 2;      // doubles the value of c
d /= 5;      // reduces the value of d to one fifth of the original value
```

3. The For Statement

The *for* statement (often called a *for-loop*) is one of three statements in C++ that act as control structures for *loops*. Any statement following a *for* statement is repeated until the *for* statement terminates. The *for* statement has the following form:

```
for(statement-1; Boolean-expression; expression)
    statement-2;
```

where *statement-1* is usually the declaration of a variable (often called the for-variable) and assignment of its initial value

Boolean-expression will halt the for statement when it is no longer true
expression is the amount that a variable, usually the for-variable, is incremented each time the for loop is executed

statement-2 is the body of the loop that is performed until the loop terminates, this statement may be a compound statement

This sounds very complex, but is very simple in practice. The following *for-loop* outputs a simple message 3 times.

```
for(int j=0; j<3; j++)
    cout << "One pass through the body of the loop." << endl;
```

When run, this program produces the following output:

```
One pass through the body of the loop.
One pass through the body of the loop.
One pass through the body of the loop.
```

When this for-loop is run, the following happens.

Pass One:

- 1) j is declared and initialized to 0
- 2) the Boolean expression is evaluated and found to be true (0<3)
- 3) the statement 'cout << "One pass through the body of the loop." << endl;' is executed
- 4) j++ is executed and the value in j increases to 1

Pass Two:

- 5) the Boolean expression is evaluated and found to be true (1<3)
- 6) the statement 'cout << "One pass through the body of the loop." << endl;' is executed
- 7) j++ is executed and the value in j increases to 2

Pass Three:

- 8) the Boolean expression is evaluated and found to be true (2<3)
- 9) the statement 'cout << "One pass through the body of the loop." << endl;' is executed
- 10) j++ is executed and the value in j increases to 3

Pass Four:

11) the Boolean expression is evaluated and found to be false and the loop halts

Notice that the statement following the *for-loop* is performed three times, but the *for* statement itself is executed four times.

4. Counters with For Loops

For variables can be used inside of the body of *for-loops*. The following example uses the *for variable* to output the squares of the integers from 1 to 5.

```
for(int x=1; x<6; x++)
    cout << "The square of " << x << " is " << x * x << endl;
```

When run, this *for-loop* produces the following output:

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

The following *for-loop* outputs the squares of 1.0, 0.8, 0.6, 0.4, and 0.2.

```
for(float f=1.0; f>0; f-=0.2)
    cout << "The square of " << f << " is " << f * f << endl;
```

When run, this *for-loop* produces the following output:

```
The square of 1 is 1
The square of 0.8 is 0.64
The square of 0.6 is 0.36
The square of 0.4 is 0.16
The square of 0.2 is 0.04
```

5. Counters in the Body of the Loop

Counters can be used for a variety of purposes, not just for controlling *for-loops*. The following example program uses a counter (the integer variable *sum*) to sum five grades so that the average of the grades may be computed. Note that the counter variable *sum* is initialized to zero before the *for-loop* starts, just as the counter variable *x* must be set to an initial value.

```

#include <iostream>
using namespace std;

void main( ) {
    int grade, sum=0;
    for(int x=0; x<5; x++) {
        cout << "Enter the numeric grade: ";
        cin >> grade;
        sum += grade;
    }
    cout << endl << "Average = " << sum/5 << endl;
}

```

If the user enters 85, 77, 96, 63 and 83 when prompted during the program run, the run of the program will look like the following:

```

Enter the numeric grade: 85
Enter the numeric grade: 77
Enter the numeric grade: 96
Enter the numeric grade: 63
Enter the numeric grade: 83

Average = 80

```

6. For with Break

The previous example program can be altered to allow for an unknown number of grades. To do this, several changes must be made to the program. The counter variable *x* will be used to count the number of grades entered and will replace the constant 5 in the calculation of the average.

The Boolean expression could be changed to detect a terminal value, such as any number less than 0. Unfortunately, this would mean that the terminal grade would be added into the counter variable *sum* and would result in an incorrect average calculation. Instead, the Boolean expression can be omitted, and the loop limiting function given to an *if* statement inside of the loop. The *if* statement is placed after the *cin* statement, but before the *sum* counter statement. If the program used enters a grade value less than 0, the Boolean expression within the *if* statement will become true and the *break* statement will be performed. This will cause the *sum* counter to be skipped and the loop halted. The terminal value in the variable *grade* will not be included in *sum*. Control will pass to the statements after the for-loop. (*Note the change to the prompt.*)

```

#include <iostream>
using namespace std;

void main() {
    int grade, sum=0;
    for(int x=0; ; x++)
    {
        cout << " Enter the numeric grade (<0 quits): ";
        cin >> grade;
        if (grade<0)
            break;
        sum += grade;
    }
    cout << endl << "Average = " << sum/x << endl;
}

```

If the user enters the grades 80, 66, 89, 59, 77, 85, 85, and the terminal value -5, the run will appear as follows:

```

Enter the numeric grade (<0 quits): 80
Enter the numeric grade (<0 quits): 66
Enter the numeric grade (<0 quits): 89
Enter the numeric grade (<0 quits): 59
Enter the numeric grade (<0 quits): 77
Enter the numeric grade (<0 quits): 85
Enter the numeric grade (<0 quits): 85
Enter the numeric grade (<0 quits): -5

```

```

Average = 77

```

7. Nested For Loops

Like any other control structure, *for-loops* may be nested inside of other *for-loops*. In a nesting of two *for-loops*, the inner loop must complete before the counter of the outer loop can change the value of the *for variable* of the outer loop. In fact, the inner loop will start over and complete each time the outer loop for variable changes. The following example will illustrate these points by outputting the for values of two nested for loops each time the value of one of the for variables changes.

```

for(int x=1; x < 4; x++) {
    cout << x << endl;
    for(int y=7; y<9;y++)
        cout << "\t" << y << endl;
}

```

When the example is run inside of a program, the output will be as follows:

```

1
  7
  8
2
  7
  8
3
  7
  8

```

If this example is rewritten with the value of *x* output as part of the *cout* statement of the inner loop,

```

for(int x=1; x < 4; x++)
    for(int y=7; y<9;y++)
        cout << x << y << endl;
}

```

the output will be as follows;

```

17
18
27
28
37
38

```

8. Leaving Things Out

For statements will work correctly with a variety of things missing. For example, the following for statement produces an infinite loop:

```
for( ; ; )
```

For variables can be declared before the for statement and can be assigned in the statement or before:

```
int x=0
for( ; x<27; x++)
```

or

```
int x;
for(x=0 ; x<27; x++)
```

For variables can be incremented in the loop. For example:

```
for(int x=0; x<27; )
    x++;
```

For loops can be exited (like any other compound statement) by use of the *break* statement. For example:

```
for(int x=0; x<27; x++) {
    cout << "Enter a number: ";
    cin >> n;
    if (n == 0)
        break;
    cout << m / n << endl;
}
```

For loops can be short-circuited (like any other compound statement) by use of the *continue* statement. When executed, the *continue* statement causes the remaining portion of the loop for be skipped, but the loop will continue from the for statement. For example:

```
for(int x=0; x<27; x++) {
    cout << "Enter a number: ";
    cin >> n;
    if (n == 0)
        continue;
    cout << m / n << endl;    // statement skipped if n is 0, but
                             // loop continues
}
```

Programming Exercise 20.1

Write a program to output your name 200 times.

Programming Exercise 20.2

Write a program to output every integer from –100 to 100.

Programming Exercise 20.3

Write a program to compute and output the cubes of the integers from –3 to 3.

Programming Exercise 20.4

Write a program to output all possible solutions to the expression $X^2 - y^2$ where X is an integer from 3 to 9 and Y is an integer from 2 to 4.

Programming Exercise 20.5

Write a program to output all possible solutions for the expression $6X^3 - 5Y^2 - 4Z + 2$ where X is an integer from 1 to 20, Y is an integer from -5 to 5, and Z is an integer from 3 to 17.

Programming Exercise 20.6

Write and use a function that receives a positive integer n and returns the sum of the integers from 1 to n.

Programming Exercise 20.7

Write and use a function that receives a positive integer n and returns the factorial of the n. (factorial of n: written n!, is: $1 * 2 * 3 * 4 * \dots * n$)

Programming Exercise 20.8

Write and use a function that receives two integer numbers n and m, then returns the sum of the integers from n to m.

Programming Exercise 20.9

Write and use a function that receives an integer n and a positive integer power p, computes n raised to the power p with a counter and for-loop, then returns n raised to the power p.

Programming Exercise 20.10

Write a program that sums the heights of any number of people, then outputs the average height.

Programming Exercise 20.11

Write a and use a function that when given a prime number P, returns its Mersenne number. (Mersenne numbers are $2^P - 1$ where P is a prime number.) Hint: Use the function from assignment #20.9.

Programming Exercise 20.12

Characters are a special class of integer. Each character in the class char is a representation of a numeric value from 0 to 255. If a char variable is assigned an integer value, the equivalent character will be created in the char variable. For example, if 65 is assigned to the char variable c, when c is output, the character A will appear.

```
int i = 65;
char c = i;
cout << c << " is " << i;
```

Output: A is 65

Write a program that produces all 256 characters along side the numeric value of each character. Use the function getch() (include <conio.h>) to pause between each character.

Challenge

Use text characters to output an old style graphics text box with your name in it.

Example:

George Washington

Programming Exercise 20.13

Write a program that uses a for loop to stage a race using an object of class track.

Programming Exercise 20.14

Certificates of Deposit (CDs) are purchased by a banking customer as an investment. A CD offers a higher rate of return (higher interest rate paid) than other types of bank accounts. In exchange for this higher return on the original investment, the customer purchasing a CD must agree to not cash in the CD for a certain length of time.

The First National Piggy Bank offers various types of CDs. The FNPB 90-day CD pays an annual rate of 5.25% with interest compounded every 30 days. The 120-day CD also pays an annual rate of 5.25%, but interest is compounded every day. The 180-day CD pays an annual rate of 5.50%. with interest compounded every day. CDs may be purchased in any multiple of \$5,000.

Design, create, and use a class that functions as a CD sales outlet for FNPB. Customers should be allowed to specify the CD amount and duration. The FNPB object must then show the customer the total return on the investment. The class should handle invalid amounts by explaining to the customer correct amounts to enter.

Programming Exercise 20.15

Write a program to simulate 1000 roles of a pair of dice. Keep count of the number of times the result of the role results in a two, three, four, etc. Output the counts.

Note: How to Generate Random Die Roles

- 1) include the standard header files <time> and <stdlib>
- 2) before the loop, seed the random number algorithm

```
srand( (unsigned)time(NULL) );
```

- 3) use the rand() functions and the remainder operator (%) to generate numbers from 1 to 6 for each die

```
die1 = rand() % 6 + 1
```

(rand() generates integers from 0 to 32,767. % 6 is used to limit the numbers generated to 0 to 5. Adding one moves the number range to 1 to 6.)

- 4) add the role for both dies together to get the role of the dice

Challenge

Read the chapter on graphics in Windows and change the output of the program to a bar graph.

Programming Exercise 20.16

Fibonacci numbers are the number series 0, 1, 1, 2, 3, 5, 8, 13, 21, ... The function Fibonacci is defined as:

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n-1) + f(n-2), \text{ where } n \text{ is a integer greater than } 1$$

In English, this means that:

giving 0 to the Fibonacci function returns a 0,
giving 1 to the Fibonacci function returns 1,
giving 2 to the Fibonacci function returns $0+1 = 1$,
giving 3 to the Fibonacci function returns $1 + 1 = 2$,
giving 4 to the Fibonacci function returns $1 + 2 = 3$,
giving 5 to the Fibonacci function returns $2 + 3 = 5$,
etc.

In other words, each Fibonacci number n where n is an integer greater than 1 is the sum of the two previous Fibonacci numbers.

Write and use a function that receives an integer n and returns the n^{th} Fibonacci number.
(Challenge on next page)

Challenge (for 20.16)

Read the chapter on beginning graphics in Windows and change the output of the program to a bar graph of the first 10 Fibonacci numbers.

Programming Exercise 20.17

The number PI can be approximated in decimal numbers by the following function.

$$f(1) = 4 * (1 / (2*3*4)) + 3$$

$$f(2) = 4 * (1 / (2*3*4 - 1 / (4*5*6)) + 3$$

$$f(3) = 4 * (1 / (2*3*4 - 1 / (4*5*6) + 1 / (6*7*8)) + 3$$

$$f(4) = 4 * (1 / (2*3*4 - 1 / (4*5*6) + 1 / (6*7*8) - 1 / (8*9*10)) + 3$$

$$f(5) = 4 * (1 / (2*3*4 - 1 / (4*5*6) + 1 / (6*7*8) - 1 / (8*9*10) + 1 / (10*11*12)) + 3$$

$$f(6) = 4 * (1 / (2*3*4 - 1 / (4*5*6) + 1 / (6*7*8) - 1 / (8*9*10) + 1 / (10*11*12) - 1 / (12*13*14)) + 3$$

etc.

Write a function that receives an integer n and returns an approximation of PI. What is the best PI that can be calculated this way and what is the value of n? (*Hint: Be sure and specify the number of decimals to output the best possible answer.*)

Programming Exercise 20.18

The function sine takes radians and returns a number between -1 and 1. Write a program that outputs the sine of 100 radian values between 0 and 2π .

Challenge

Read the chapter on graphics in Windows and change the output of the program to a sine curve.