Chapter 5:
Control Structures II (Repetition)

Objectives

- In this chapter, you will:
  - Learn about repetition (looping) control structures
  - Learn how to use a `while` loop in a program
  - Explore how to construct and use counter-controlled, sentinel-controlled, flag-controlled, and EOF-controlled repetition structures
  - Learn how to use a `for` loop in a program
  - Learn how to use a `do…while` loop in a program

Objectives (cont'd.)

- Examine break and continue statements
- Discover how to form and use nested control structures
- Learn how to avoid bugs by avoiding patches
- Learn how to debug loops

Why Is Repetition Needed?

- Repetition allows efficient use of variables
- Can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

`while` Looping (Repetition) Structure

- Syntax of the `while` statement:

```
while (expression)
    statement
```

- `statement` can be simple or compound
- `expression` acts as a decision maker and is usually a logical expression
- `statement` is called the body of the loop
- The parentheses are part of the syntax

`while` Looping (Repetition) Structure (cont'd.)

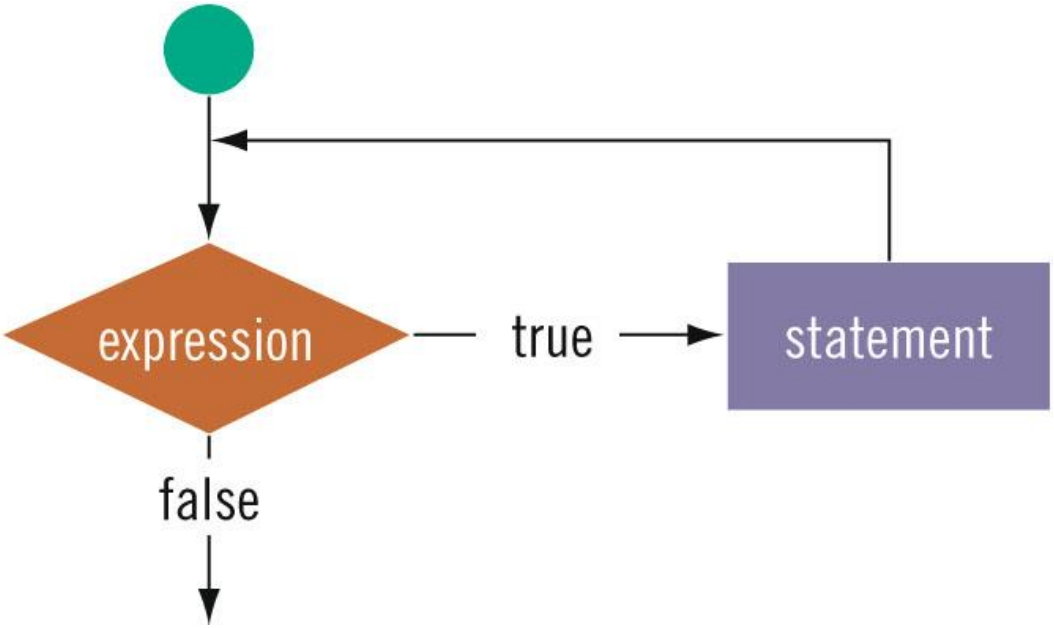

FIGURE 5-1 `while` loop

`while` Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-1

Consider the following C++ program segment: (Assume that `i` is an `int` variable.)

```
i = 0;                          //Line 1

while (i <= 20)                 //Line 2
{
    cout << i << " ";           //Line 3
    i = i + 5;                  //Line 4
}

cout << endl;
```

Sample Run:

```
0 5 10 15 20
```

`while` Looping (Repetition) Structure (cont'd.)

- The variable `i` in Example 5-1 is called the loop control variable (LCV)
- Infinite loop: continues to execute endlessly
  - Avoided by including statements in loop body that assure the exit condition is eventually false

`while` Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-2

Consider the following C++ program segment:

```
i = 20;                          //Line 1
while (i < 20)                   //Line 2
{
    cout << i << " ";            //Line 3
    i = i + 5;                   //Line 4
}
cout << endl;                    //Line 5
```

It is easy to overlook the difference between this example and Example 5-1. In this example, in Line 1, `i` is set to 20. Because `i` is 20, the expression `i < 20` in the `while` statement (Line 2) evaluates to `false`. Because initially the loop entry condition, `i < 20`, is `false`, the body of the `while` loop never executes. Hence, no values are output, and the value of `i` remains 20.

Case 1: Counter-Controlled `while` Loops

- When you know exactly how many times the statements need to be executed
  - Use a counter-controlled `while` loop

```
counter = 0;          //initialize the loop control variable

while (counter < N) //test the loop control variable
{
      .
      .
      .
      counter++;      //update the loop control variable
      .
      .
      .
}
```

Case 2: Sentinel-Controlled `while` Loops

- Sentinel variable is tested in the condition
- Loop ends when sentinel is encountered

```
cin >> variable;                    //initialize the loop control variable

while (variable != sentinel)  //test the loop control variable
{
      .
      .
      .
      cin >> variable;        //update the loop control variable
      .
      .
      .
}
```

Case 3: Flag-Controlled `while` Loops

- Flag-controlled `while` loop: uses a `bool` variable to control the loop

```
found = false;        //initialize the loop control variable

while (!found)        //test the loop control variable
{
    .
    .
    .
    if (expression)
        found = true; //update the loop control variable
    .
    .
    .
}
```

Number Guessing Game

- Example 5-6 implements a number guessing game using a flag-controlled while loop
- Uses the function `rand` of the header file `cstdlib` to generate a random number
  - `rand()` returns an `int` value between 0 and 32767
  - To convert to an integer >= 0 and < 100:
    - `rand() % 100`

Case 4: EOF-Controlled `while` Loops

- End-of-file (EOF)-controlled `while` loop: when it is difficult to select a sentinel value
- The logical value returned by `cin` can determine if there is no more input

Case 4: EOF-Controlled `while` Loops (cont'd.)

EXAMPLE 5-7

The following code uses an EOF-controlled `while` loop to find the sum of a set of numbers:

```
int sum = 0;
int num;

cin >> num;

while (cin)
{
    sum = sum + num;    //Add the number to sum
    cin >> num;         //Get the next number
}

cout << "Sum = " << sum << endl;
```

eof Function

- The function `eof` can determine the end of file status
- `eof` is a member of data type `istream`
- Syntax for the function `eof`:

```
istreamVar.eof()
```

- `istreamVar` is an input stream variable, such as `cin`

More on Expressions in `while` Statements

- The expression in a `while` statement can be complex
  - Example:

```
while ((noOfGuesses < 5) && (!isGuessed))
{
   . . .
}
```

`for` Looping (Repetition) Structure

- `for` loop: called a counted or indexed `for` loop
- Syntax of the `for` statement:

```
for (initial statement; loop condition; update statement)
    statement
```

- The `initial statement`, `loop condition`, and `update statement` are called `for` loop control statements

`for` Looping (Repetition) Structure (cont'd.)

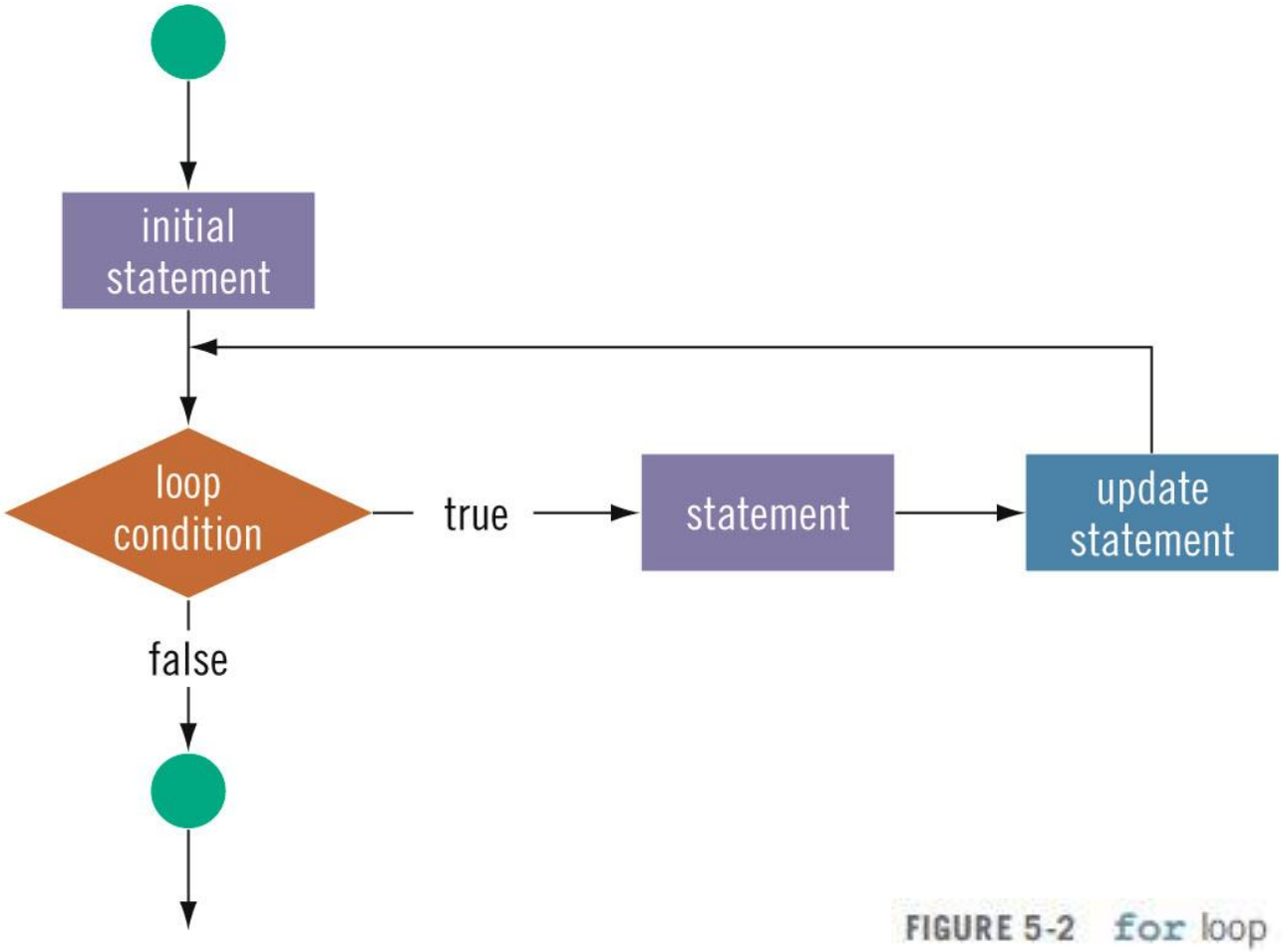

FIGURE 5-2 `for` loop

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-9

The following for loop prints the first 10 nonnegative integers:

```
for (i = 0; i < 10; i++)
    cout << i << " ";
cout << endl;
```

The initial statement, i = 0;, initializes the int variable i to 0. Next, the loop condition, i < 10, is evaluated. Because 0 < 10 is true, the print statement executes and outputs 0. The update statement, i++, then executes, which sets the value of i to 1. Once again, the loop condition is evaluated, which is still true, and so on. When i becomes 10, the loop condition evaluates to false, the for loop terminates, and the statement following the for loop executes.

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-10

1. The following `for` loop outputs `Hello!` and a star (on separate lines) five times:

```
for (i = 1; i <= 5; i++)
{
    cout << "Hello!" << endl;
    cout << "*" << endl;
}
```

2. Consider the following `for` loop:

```
for (i = 1; i <= 5; i++)
    cout << "Hello!" << endl;
    cout << "*" << endl;
```

This loop outputs `Hello!` five times and the star only once.

`for` Looping (Repetition) Structure (cont'd.)

- The following is a semantic error:

EXAMPLE 5-11

The following `for` loop executes five empty statements:

```
for (i = 0; i < 5; i++);      //Line 1
    cout << "*" << endl;      //Line 2
```

The semicolon at the end of the `for` statement (before the output statement, Line 1) terminates the `for` loop. The action of this `for` loop is empty, that is, null.

- The following is a legal (but infinite) `for` loop:

```
for (;;)
    cout << "Hello" << endl;
```

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-12

You can count backward using a for loop if the for loop control expressions are set correctly.

For example, consider the following for loop:

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

The output is:

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

In this for loop, the variable i is initialized to 10. After each iteration of the loop, i is decremented by 1. The loop continues to execute as long as i >= 1.

for Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-13

You can increment (or decrement) the loop control variable by any fixed number. In the following `for` loop, the variable is initialized to `1`; at the end of the `for` loop, `i` is incremented by `2`. This `for` loop outputs the first `10` positive odd integers.

```
for (i = 1; i <= 20; i = i + 2)
    cout << " " << i;
cout << endl;
```

do...while Looping (Repetition) Structure

- Syntax of a `do...while` loop:

```
do
    statement
while (expression);
```

- The `statement` executes first, and then the `expression` is evaluated
  - As long as `expression` is true, loop continues
- To avoid an infinite loop, body must contain a statement that makes the `expression` false

`do...while` Looping (Repetition) Structure (cont'd.)

- The statement can be simple or compound
- Loop always iterates at least once

do...while Looping (Repetition) Structure (cont'd.)

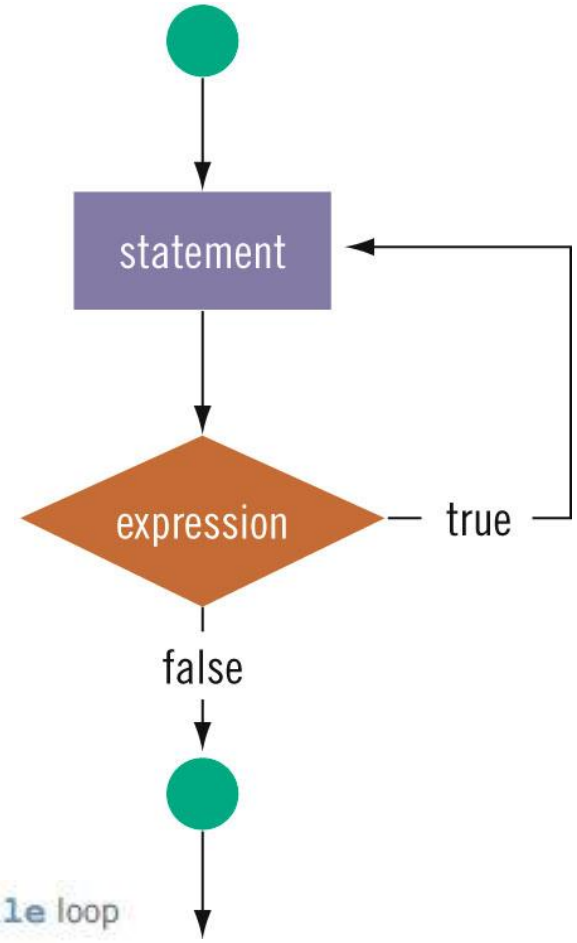

FIGURE 5-3 do...while loop

do...while Looping (Repetition) Structure (cont'd.)

EXAMPLE 5-18

```
i = 0;

do
{
    cout << i << " ";
    i = i + 5;
}
while (i <= 20);
```

The output of this code is:

```
0 5 10 15 20
```

After 20 is output, the statement:

```
i = i + 5;
```

changes the value of i to 25 and so i <= 20 becomes false, which halts the loop.

`break` and `continue` Statements

- `break` and `continue` alter the flow of control
- `break` statement is used for two purposes:
  - To exit early from a loop
    - Can eliminate the use of certain (flag) variables
  - To skip the remainder of a `switch` structure
- After `break` executes, the program continues with the first statement after the structure

`break` and `continue` Statements (cont'd.)

- `continue` is used in `while`, `for`, and `do…while` structures
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop

Nested Control Structures

- To create the following pattern:

```
*
**
***
****
*****
```

- We can use the following code:

```
for (i = 1; i <= 5 ; i++)
{
      for (j = 1; j <= i; j++)
            cout << "*";
      cout << endl;
}
```

Nested Control Structures (cont'd.)

- What is the result if we replace the first for statement with this?

```
for (i = 5; i >= 1; i--)
```

- Answer:

```
*****
****
***
**
*
```

Summary

- C++ has three looping (repetition) structures:
  - `while`, `for`, and `do…while`
- `while`, `for`, and `do` are reserved words
- `while` and `for` loops are called pretest loops
- `do...while` loop is called a posttest loop
- `while` and `for` may not execute at all, but `do...while` always executes at least once

Summary (cont'd.)

- `while: expression` is the decision maker, and `statement` is the body of the loop
- A `while` loop can be:
  - Counter-controlled
  - Sentinel-controlled
  - EOF-controlled
- In the Windows console environment, the end-of-file marker is entered using `Ctrl+z`

Summary (cont'd.)

- `for` loop: simplifies the writing of a counter-controlled `while` loop
  - Putting a semicolon at the end of the `for` loop is a semantic error
- Executing a `break` statement in the body of a loop immediately terminates the loop
- Executing a `continue` statement in the body of a loop skips to the next iteration