COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 5:
Control Structures II (Repetition)

Programming Example: Fibonacci Number

- Consider the following sequence of numbers:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Called the Fibonacci sequence
- Given the first two numbers of the sequence (say, a1 and a2)
  - *n*th number a_n, n >= 3, of this sequence is given by: $a_n = a_{n-1} + a_{n-2}$

Programming Example: Fibonacci Number (cont'd.)

- Fibonacci sequence
  - n^{th} Fibonacci number
  - $a_2 = 1$
  - $a_1 = 1$
  - Determine the n^{th} number a_n, n >= 3

Programming Example: Fibonacci Number (cont'd.)

- Suppose a_2 = 6 and a_1 = 3
  - $a_3 = a_2 + a_1 = 6 + 3 = 9$
  - $a_4 = a_3 + a_2 = 9 + 6 = 15$
- Write a program that determines the *n*th Fibonacci number, given the first two numbers

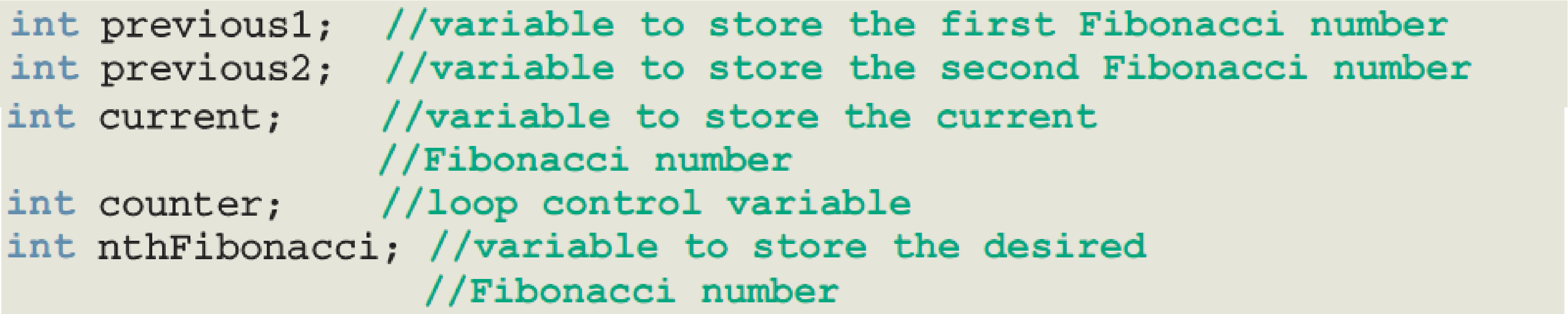
Programming Example: Input and Output

- Input: first two Fibonacci numbers and the desired Fibonacci number
- Output: n^{th} Fibonacci number

Programming Example: Problem Analysis and Algorithm Design

- Algorithm:
  - Get the first two Fibonacci numbers
  - Get the desired Fibonacci number
    - Get the position, *n*, of the number in the sequence
  - Calculate the next Fibonacci number
    - Add the previous two elements of the sequence
  - Repeat Step 3 until the *n*th Fibonacci number is found
  - Output the *n*th Fibonacci number

Programming Example: Variables



```
int previous1;  //variable to store the first Fibonacci number
int previous2;  //variable to store the second Fibonacci number
int current;    //variable to store the current
                //Fibonacci number
int counter;    //loop control variable
int nthFibonacci; //variable to store the desired
                  //Fibonacci number
```

Programming Example: Main Algorithm

1. Prompt the user for the first two numbers—that is, `previous1` and `previous2`
2. Read (input) the first two numbers into `previous1` and `previous2`
3. Output the first two Fibonacci numbers
4. Prompt the user for the position of the desired Fibonacci number

Programming Example: Main Algorithm (cont'd.)

5. Read the position of the desired Fibonacci number into `nthFibonacci`

```
6. a.if (nthFibonacci == 1)
        The desired Fibonacci number is the first Fibonacci
        number; copy the value of previous1 into current
   b. else if (nthFibonacci == 2)
        The desired Fibonacci number is the second Fibonacci
        number; copy the value of previous2 into current
```

Programming Example: Main Algorithm (cont'd.)

c. else calculate the desired Fibonacci number as follows:

Start by determining the third Fibonacci number

C1. Initialize counter to 3 to keep track of the calculated Fibonacci numbers.

C2. Calculate the next Fibonacci number, as follows:

```
current = previous2 + previous1;
```

C3. Assign the value of previous2 to previous1

C4. Assign the value of current to previous2

C5. Increment counter by 1

Repeat steps C2 through C5 until Fibonacci number is calculated:

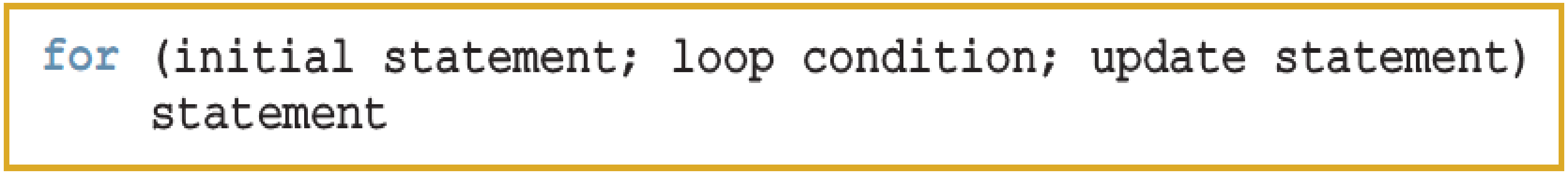
```
while (counter <= nthFibonacci)
{
     current = previous2 + previous1;
     previous1 = previous2;
     previous2 = current;
     counter++;
}
```

Programming Example: Main Algorithm (cont'd.)

7. Output the `nthFibonacci` number, which is current

`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.)

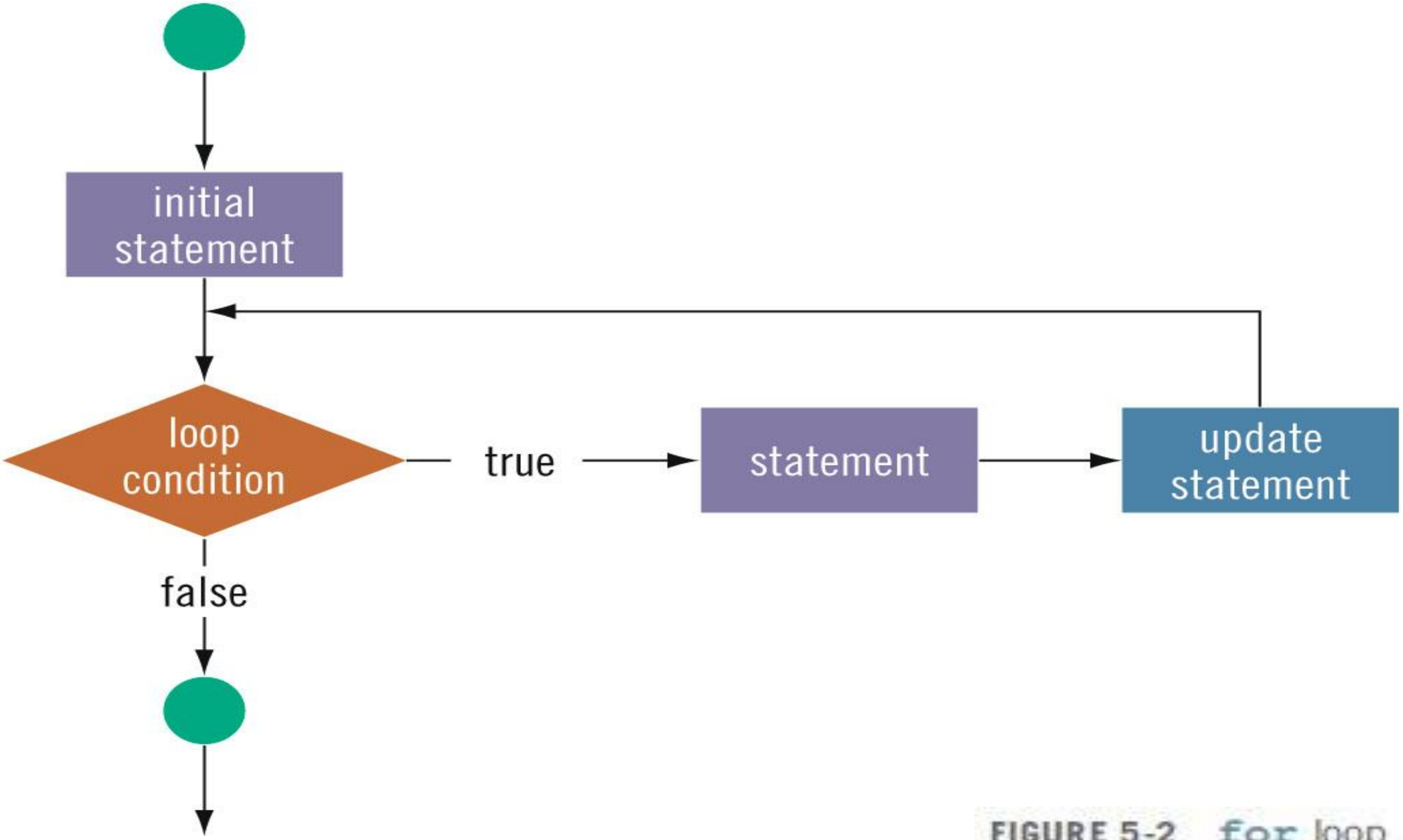
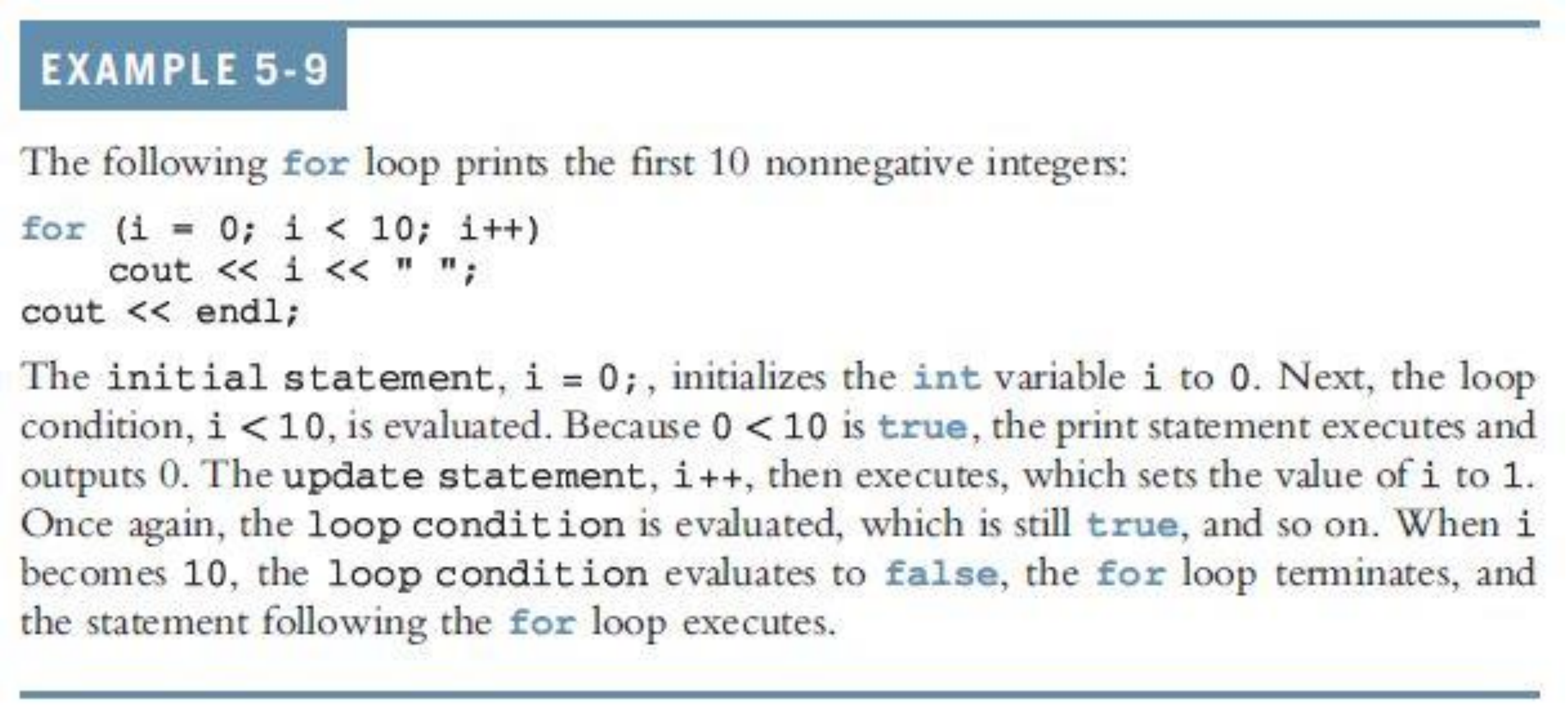


FIGURE 5-2 `for` loop

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



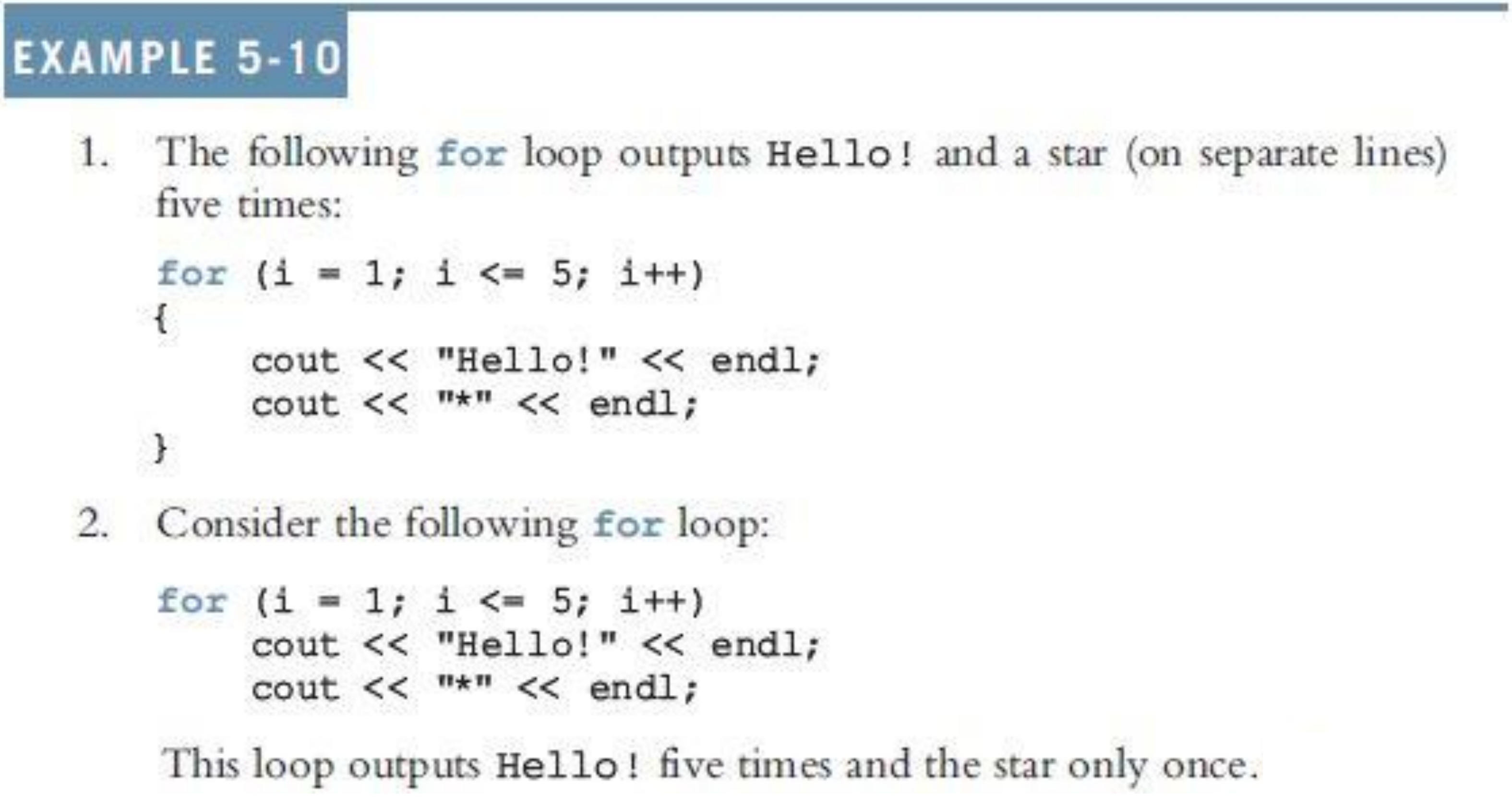
EXAMPLE 5-9

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;
}
```

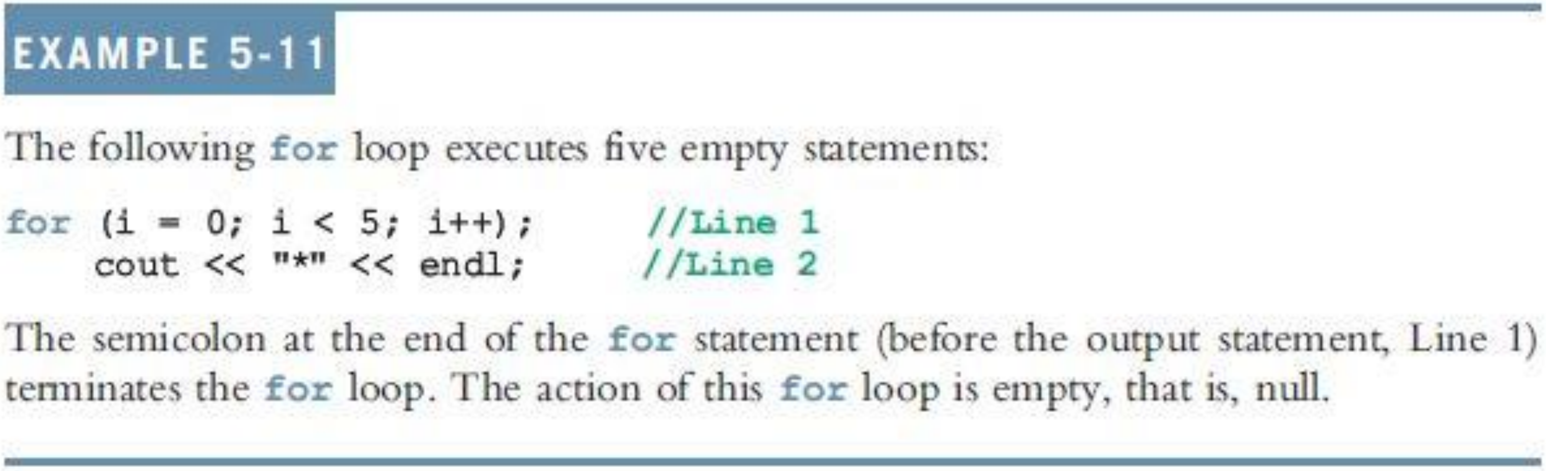
2. Consider the following **for** loop:

```
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:

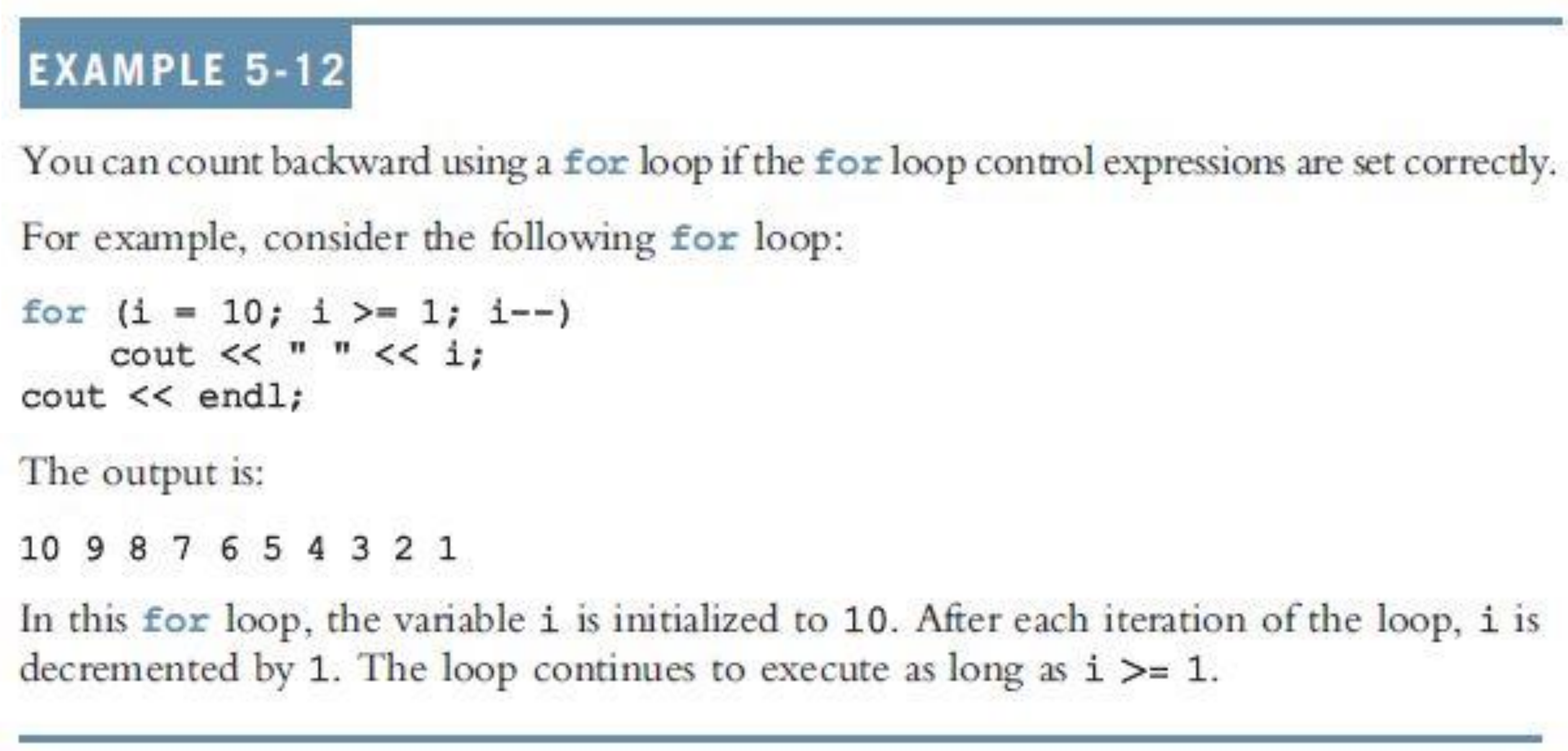
```
for (i = 0; i < 5; i++);          //Line 1
    cout << "*" << endl;          //Line 2
```

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:

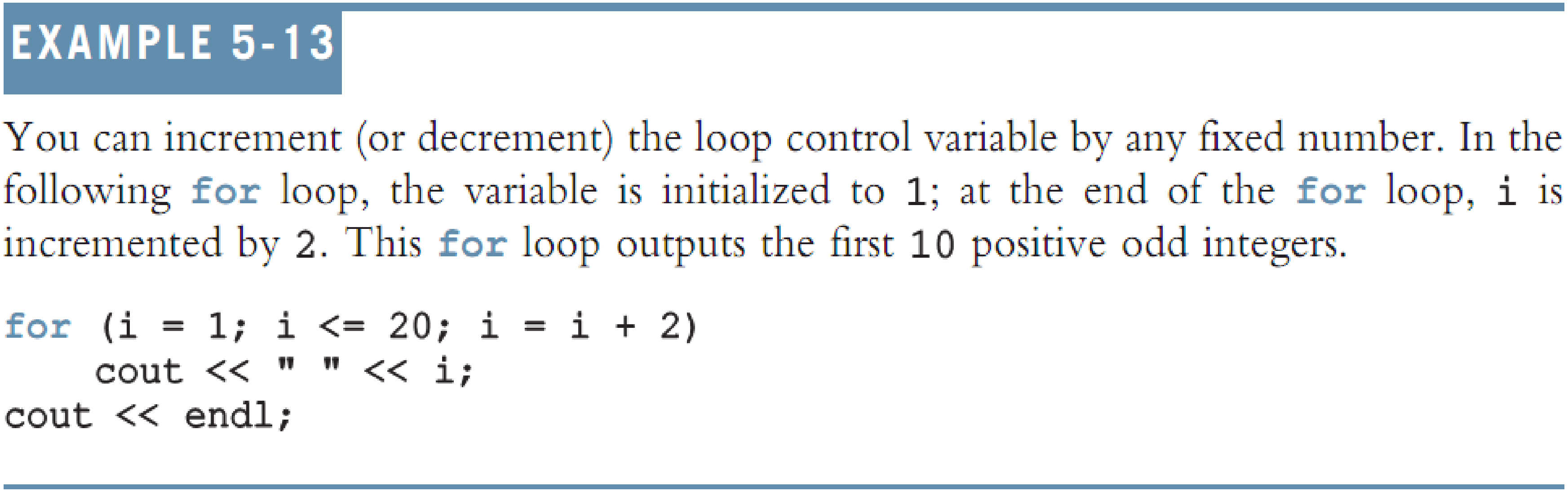
```
for (i = 10; i >= 1; i--)
    cout << " " << i;
cout << endl;
```

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.)



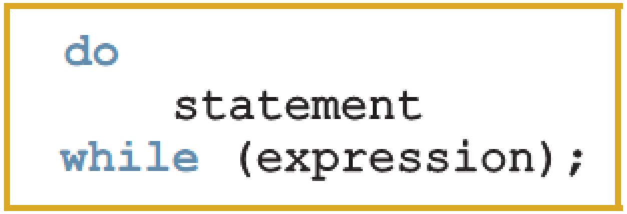
EXAMPLE 5-13

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.)

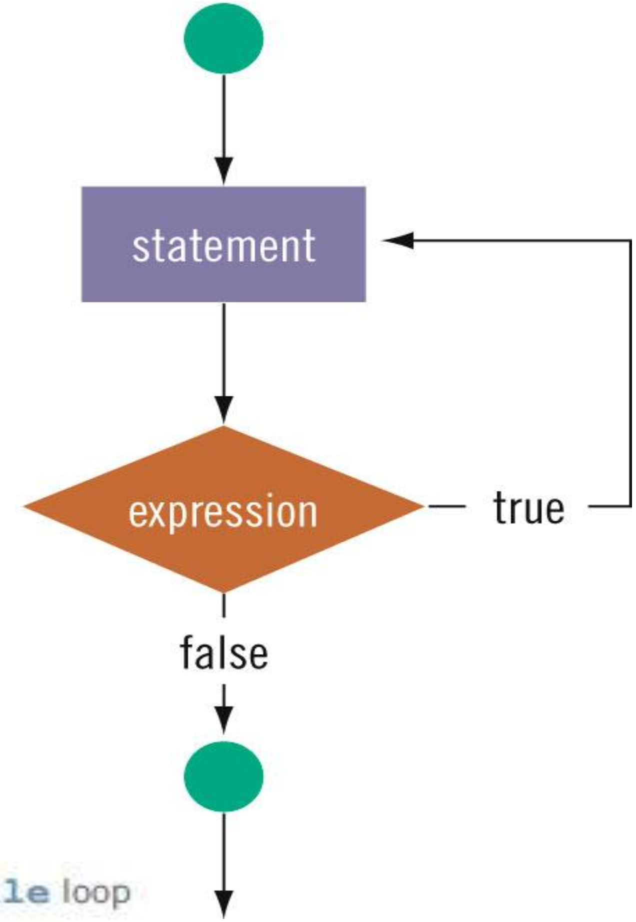
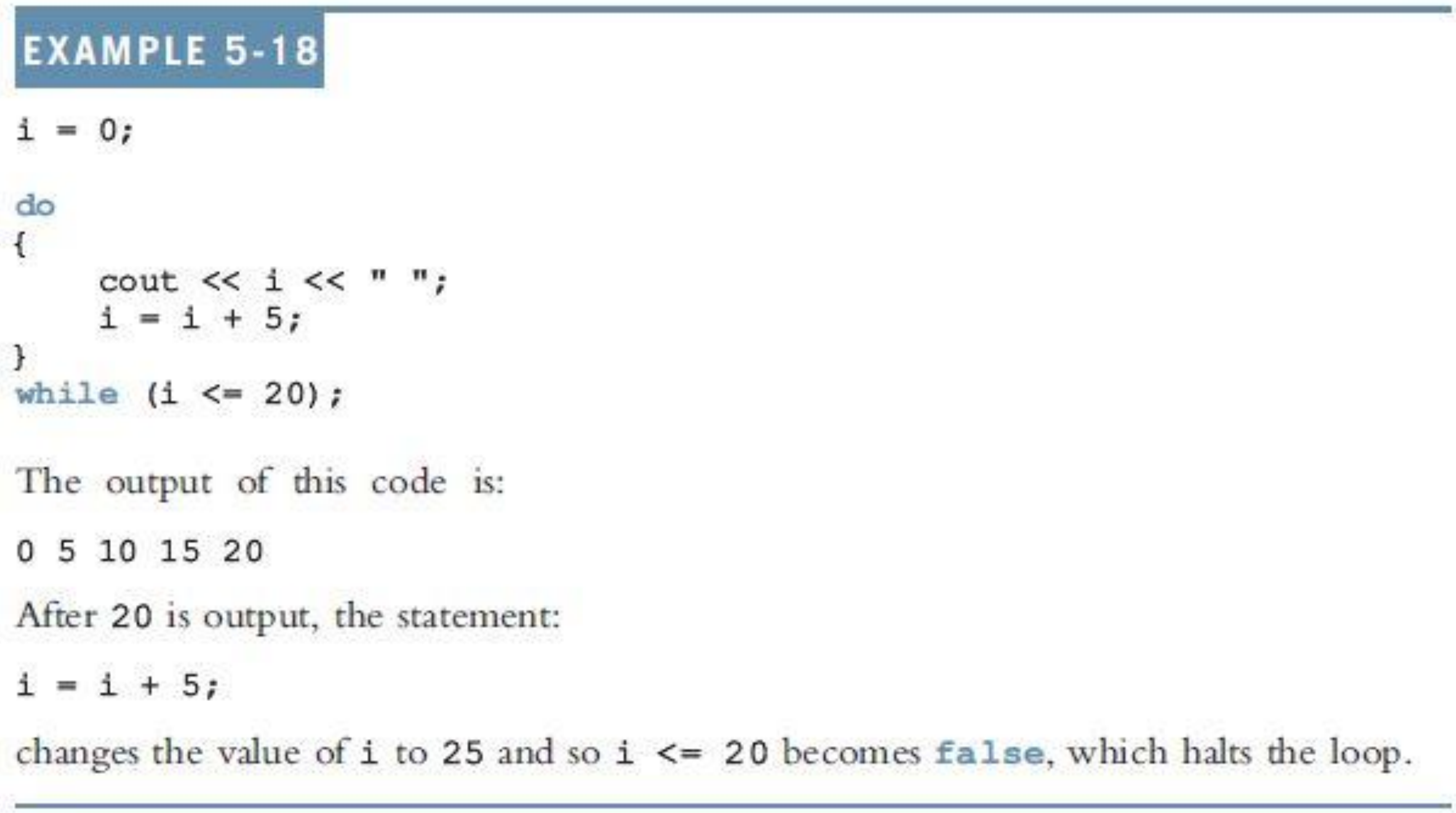


FIGURE 5-3 do...while loop

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:

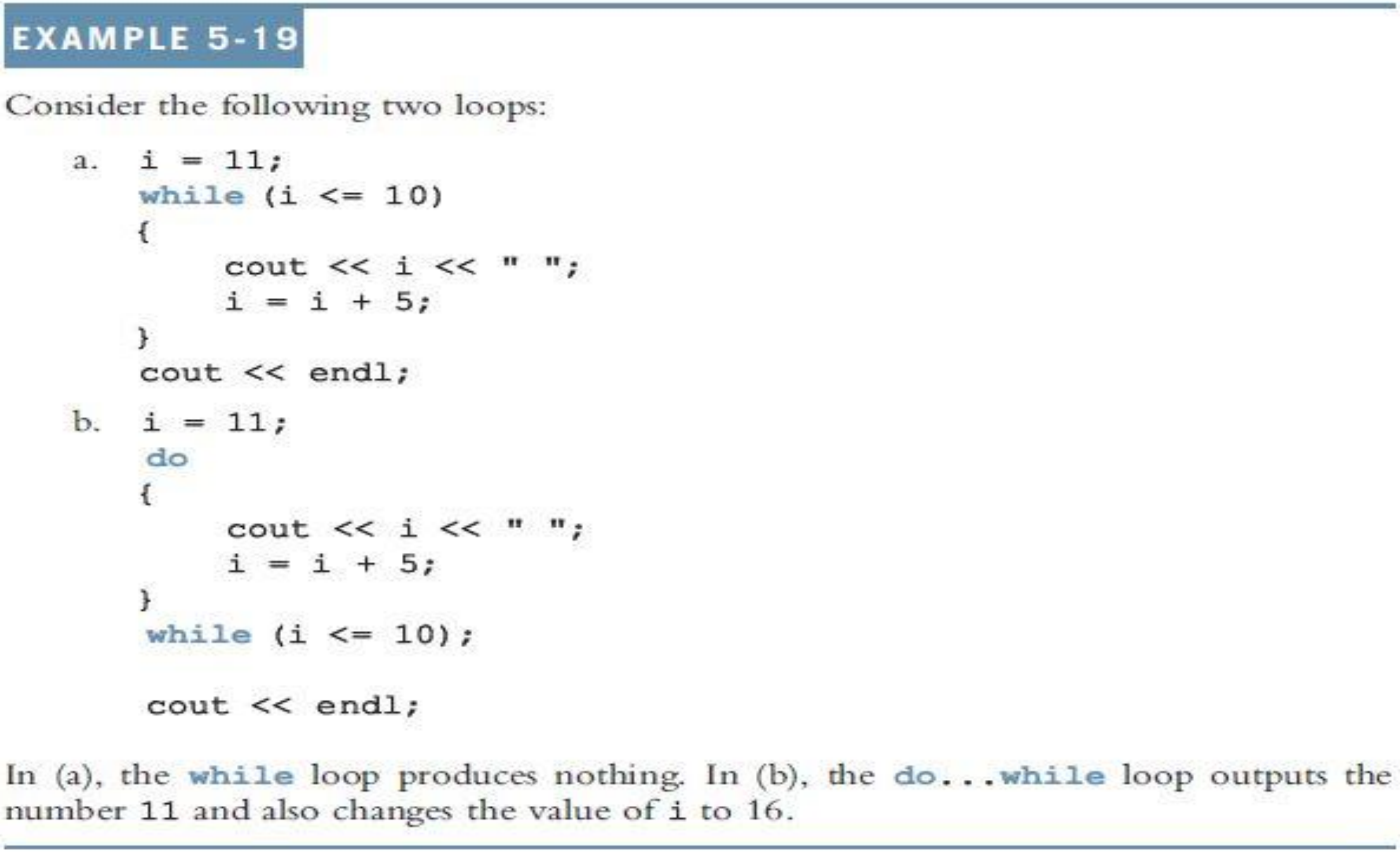
```
0 5 10 15 20
```

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.

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



EXAMPLE 5-19

Consider the following two loops:

a.
```
i = 11;
while (i <= 10)
{
    cout << i << " ";
    i = i + 5;
}
cout << endl;
```

b.
```
i = 11;
do
{
    cout << i << " ";
    i = i + 5;
}
while (i <= 10);

cout << endl;
```

In (a), the while loop produces nothing. In (b), the do...while loop outputs the number 11 and also changes the value of i to 16.

Choosing the Right Looping Structure

- All three loops have their place in C++
  - If you can determine in advance the number of repetitions needed, the `for` loop is the correct choice, i.e. *counter-controlled loop.*
  - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a `while` loop, e.g. *sentinel-controlled, EOF-controlled and flag-controlled loops*
  - If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a `do...while` loop, e.g. input-validation loop (coming next…)

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
  - Flag-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