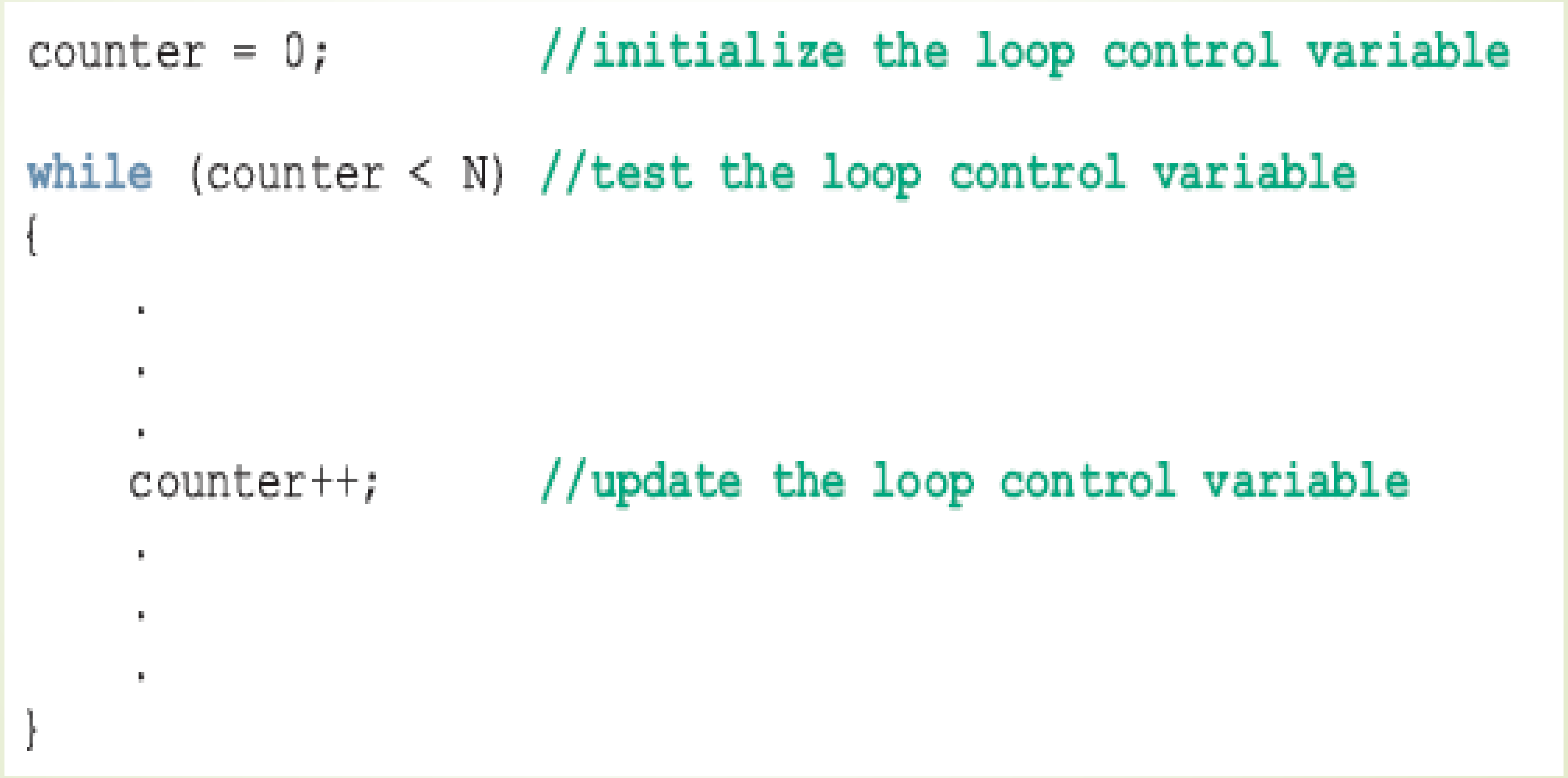
Programming Fundamentals

Aamina Batool

Counter-Controlled `while` Loops



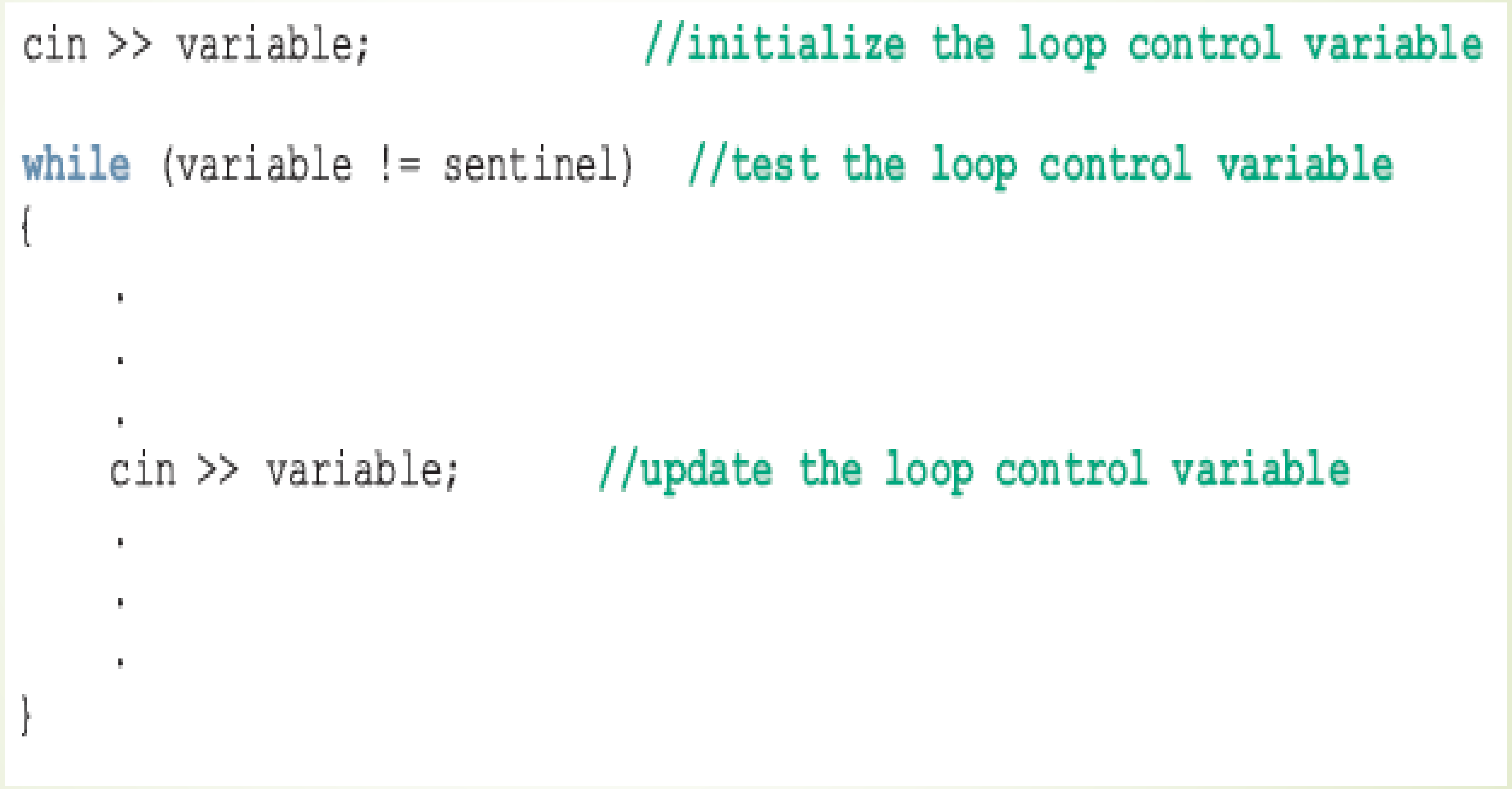
```
counter = 0;         //initialize the loop control variable

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

Example – Print 1000 Natural Numbers

- int counter = 1;
- while (counter <= 1000)
  - Cout << counter;

Sentinel-Controlled `while` Loops



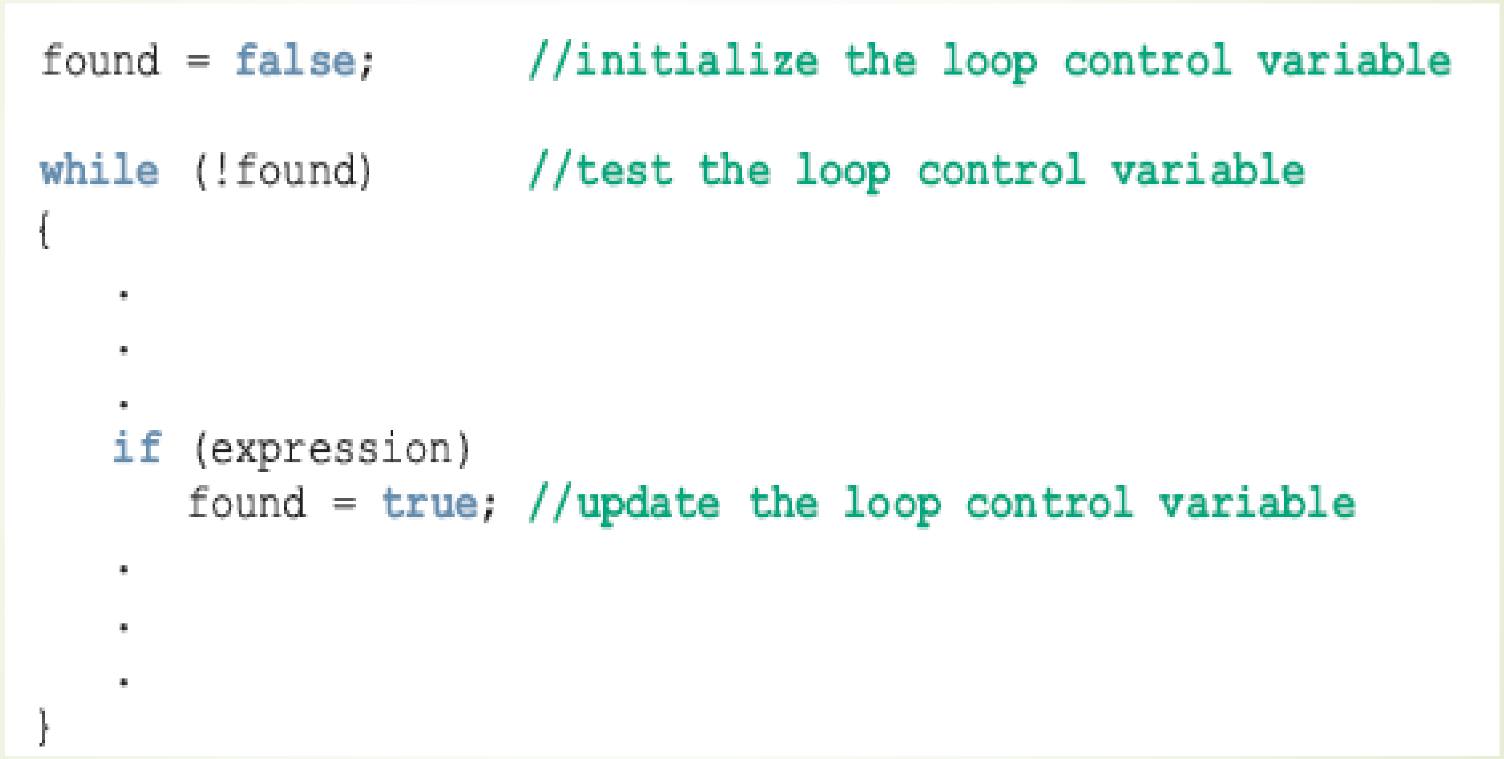
```
cin >> variable;            //initialize the loop control variable

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

Example – Validate an input

- Problem: Ask the user to enter -1.
- int number = 0;
- cin >> number;
- while (number != -1)
  - cin >> number;

Flag-Controlled `while` Loops



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

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

Example – input validation

```
int number;
bool flag = false;
while (flag != true)
{
    cout << “Enter number”;
    cin >> number;
    if (number >= 0)
        flag = true
}
```

The `for` Loop

- The general form of the `for` statement is:

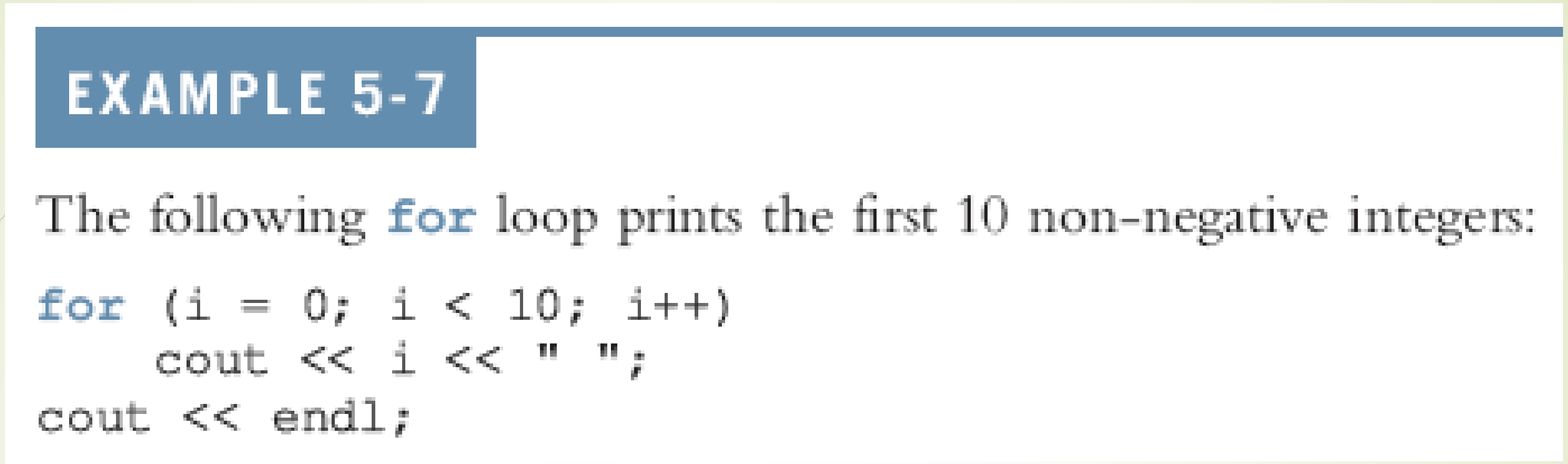
```
for (initial statement; loop condition; update statement)
    statements
```

The `for` Loop

The for loop executes as follows:

1. The initial statement executes.
2. The loop condition is evaluated. If the loop condition evaluates to true
    i. Execute the for loop statement.
    ii. Execute the update statement (the third expression in the parentheses).
3. Repeat Step 2 until the loop condition evaluates to false.

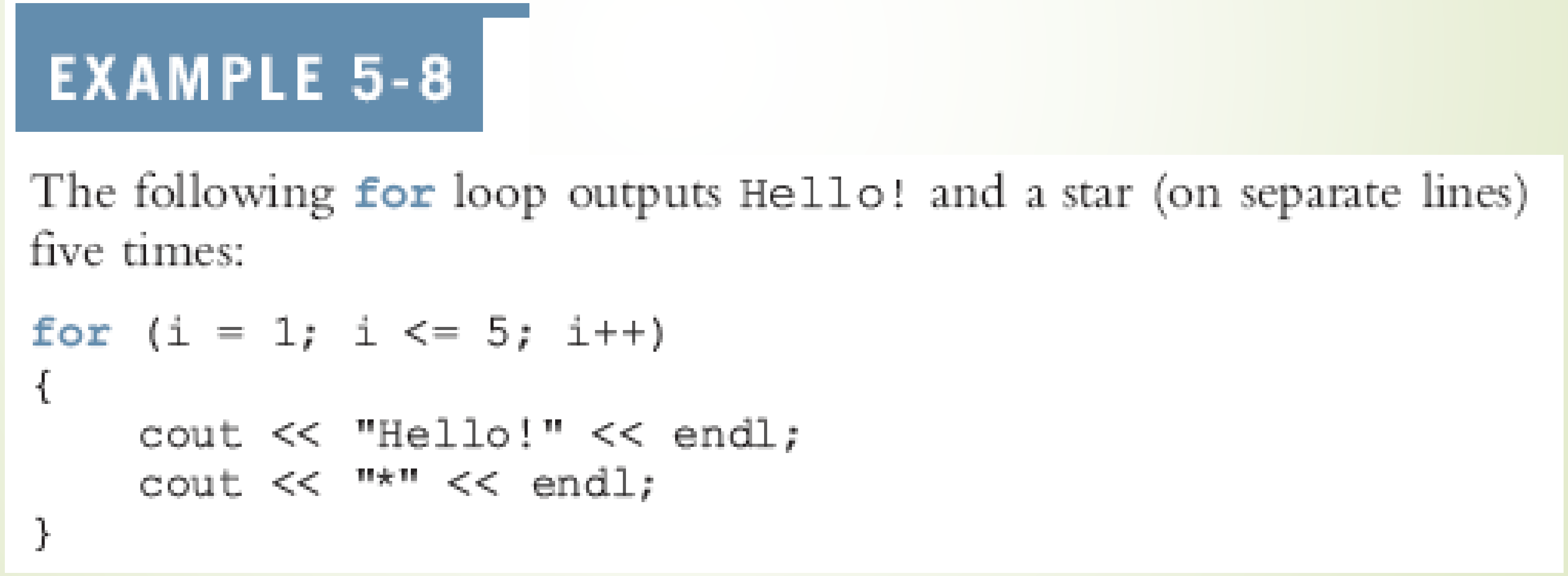
The initial statement usually initializes a variable (called the for **loop control**, or for **indexed**, **variable**).
In C++, for is a reserved word.



EXAMPLE 5-7

The following for loop prints the first 10 non-negative integers:

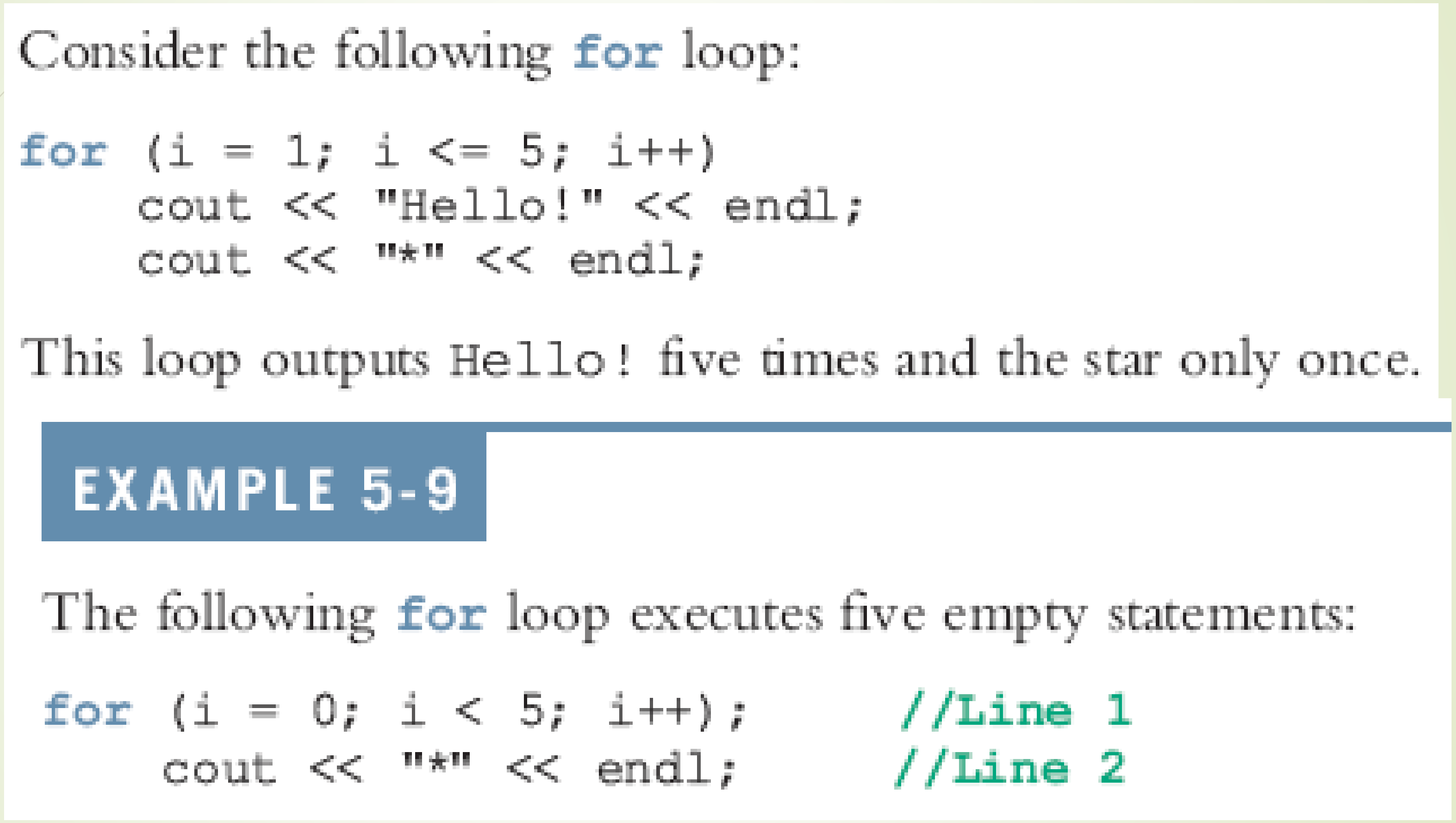
```
for (i = 0; i < 10; i++)
    cout << i << " ";
cout << endl;
```



EXAMPLE 5-8

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;
}
```



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.

EXAMPLE 5-9

The following **for** loop executes five empty statements:

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

The for Loop

- A semicolon at the end of the for statement (just before the body of the loop) is a semantic error. In this case, the action of the for loop is empty.
- In the for statement, if the loop condition is omitted, it is assumed to be true.
- In a for statement, you can omit all three statements—initial statement, loop condition, and update statement. The following is a legal for loop:

```
for (;;)

    cout << "Hello" << endl;
```

The do…while Loop

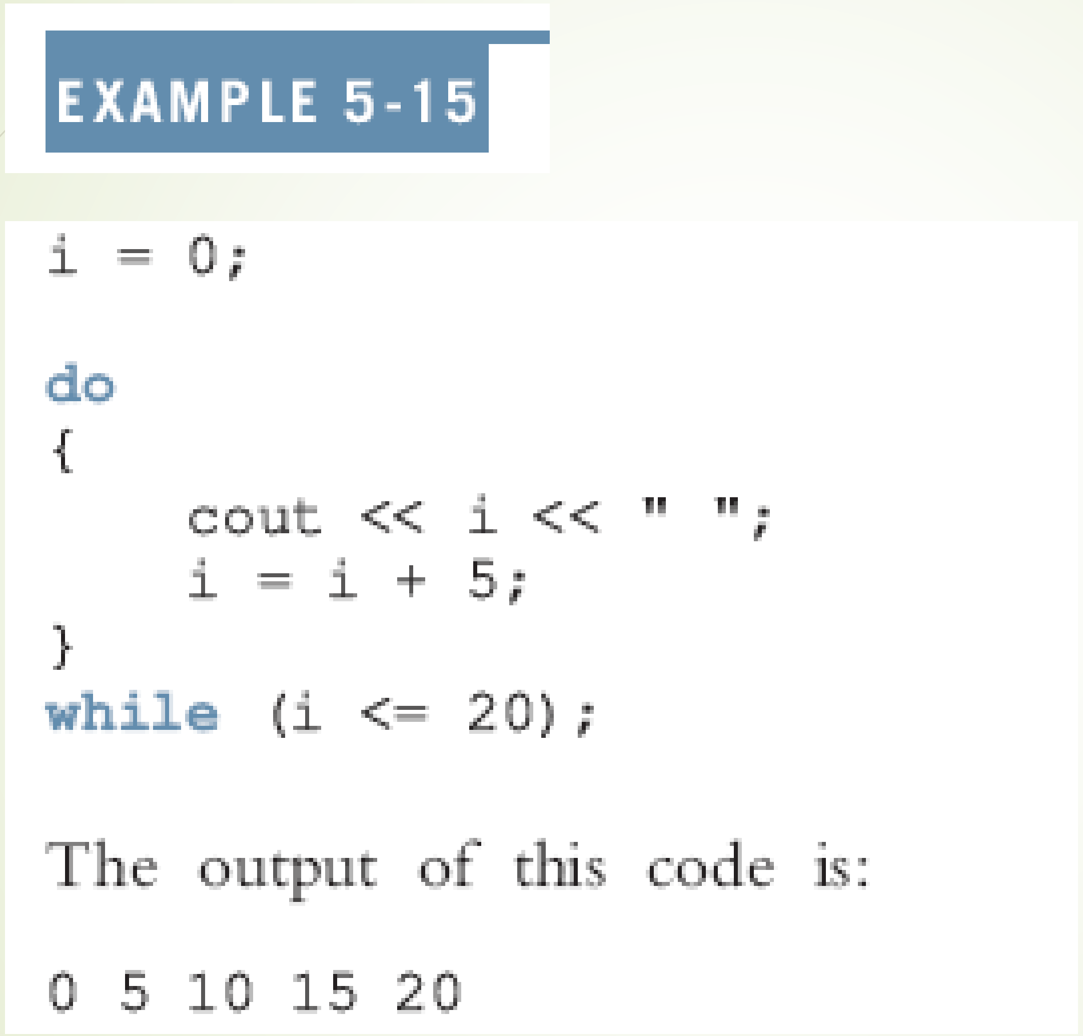
- The general form of a do…while statement is:

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

- The statement executes first, and then the expression is evaluated
- If the expression evaluates to `true`, the statement executes again
- As long as the expression in a do…while statement is `true`, the statement executes

The `do...while` Loop (continued)

- To avoid an infinite loop, the loop body must contain a statement that makes the expression `false`
- The statement can be simple or compound
- If compound, it must be in braces
- `do...while` loop has an exit condition and always iterates at least once (unlike `for` and `while`)



EXAMPLE 5-15

```
i = 0;

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

The output of this code is:

```
0 5 10 15 20
```

`break` & `continue` Statements

- `break` and `continue` alter the flow of control
- When the `break` statement executes in a repetition structure, it immediately exits
- The `break` statement can be used in `while`, `for`, and `do`…`while` loops

`break` & `continue` Statements

- The `break` statement is used for two purposes:
  1. To exit early from a loop
  2. To skip the remainder of the switch structure
- After the `break` statement executes, the program continues with the first statement after the structure
- The use of a `break` statement in a loop can eliminate the use of certain (flag) variables

`break` & `continue` Statements

- `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

`break` & `continue` Statements

- In a `while` and `do…while` structure
  - Expression (loop-continue test) is evaluated immediately after the continue statement
- In a `for` structure, the update statement is executed after the `continue` statement
  - Then the loop condition executes

Nested Control Structures

- Suppose we want to create the following pattern

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

- In the first line, we want to print one star, in the second line two stars and so on

Nested Control Structures (continued)

- Since five lines are to be printed, we start with the following for statement

```
for (i = 1; i <= 5 ; i++)
```

- The value of `i` in the first iteration is `1`, in the second iteration it is `2`, and so on
- Can use the value of `i` as limit condition in another for loop nested within this loop to control the number of starts in a line

Nested Control Structures (continued)

- The syntax is:

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

Nested Control Structures (continued)

- What pattern does the code produce if we replace the first for statement with the following?

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

- Answer:

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