Repetition Control Structures-I

Objectives of the Lecture

- **Repetition (looping) control structures.**
- **while Looping (Repetition) Structure**
- **Counter-Controlled while Loops**

Repetition (looping) control structures

Why Is Repetition Needed?

- Repetition allows you to efficiently use 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

Kinds of Repetition control structures.

- 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**
- A while loop can be:
  - **Counter-controlled**
  - **Sentinel-controlled**
  - **Flag- controlled**
  - **EOF-controlled**
- for loop: simplifies the writing of a counter-controlled while loop

while Looping (Repetition) Structure

- The general form of the while statement is:

```
while (expression)
    statement
```

- **while** is a reserved word
- 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

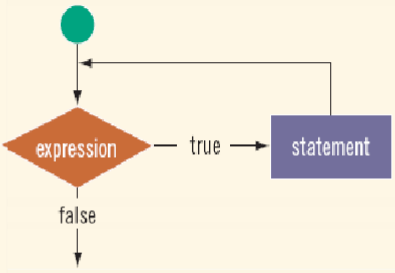

FIGURE 5-1 `while` loop

B **Infinite** loop: continues to execute **endlessly**:

- Avoided by including statements in loop body that assure exit condition is eventually false

EXAMPLE 5-1

Consider the following C++ program segment:

```
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
```

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.

Counter-Controlled while Loops

Counter-controlled repetition requires:

- The name of a control variable (or loop counter).
- The initial value of the control variable.
- The condition that tests for the final value of the control variable (i.e., whether looping should continue).
- The increment (or decrement) by which the control variable is modified each time through the loop.

Example:

```
int counter =1;           //initialization
while (counter <= 10){    //repetition condition
   cout << counter << endl;
   ++counter;             //increment
       }
```

- If you know exactly how many pieces of data need to be read,
  - **while loop** becomes a counter-controlled loop

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

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

```
//Program: Counter-Controlled Loop
#include <iostream>
using namespace std;
int main()
{
    int limit;    //store the number of data items
    int number;   //variable to store the number
    int sum;      //variable to store the sum
    int counter;  //loop control variable
    cout << "Line 1: Enter the number of "
         << "integers in the list: ";              //Line 1
    cin >> limit;                                  //Line 2
    cout << endl;                                  //Line 3
    sum = 0;                                       //Line 4
    counter = 0;                                   //Line 5
```

```
    cout << "Line 6: Enter " << limit
         << " integers." << endl;                    //Line 6
    while (counter < limit)                          //Line 7
    {
        cin >> number;                               //Line 8
        sum = sum + number;                          //Line 9
        counter++;                                   //Line 10
    }
    cout << "Line 11: The sum of the " << limit
         << " numbers = " << sum << endl;            //Line 11
    if (counter != 0)                                //Line 12
        cout << "Line 13: The average = "
             << sum / counter << endl;               //Line 13
    else                                             //Line 14
        cout << "Line 15: No input." << endl;        //Line 15

    return 0;                                        //Line 16
}
```