COMSATS University
Islamabad
(Lahore Campus)

CSC103- Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

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

```
#include <iostream>

using namespace std;

int main()
{
    int calBurnedDay1, calBurnedDay2, calBurnedDay3,
        calBurnedDay4, calBurnedDay5, calBurnedDay6,
        calBurnedDay7;
    int calBurnedInAWeek;

    cout << "Enter calories burned day 1: ";
    cin >> calBurnedDay1;
    cout << endl;

    cout << "Enter calories burned day 2: ";
    cin >> calBurnedDay2;
    cout << endl;

    cout << "Enter calories burned day 3: ";
    cin >> calBurnedDay3;
    cout << endl;

    cout << "Enter calories burned day 4: ";
    cin >> calBurnedDay4;
    cout << endl;

    cout << "Enter calories burned day 5: ";
    cin >> calBurnedDay5;
    cout << endl;

    cout << "Enter calories burned day 6: ";
    cin >> calBurnedDay6;
    cout << endl;
```

```
    cout << "Enter calories burned day 7: ";
    cin >> calBurnedDay7;
    cout << endl;

    calBurnedInAWeek = calBurnedDay1 + calBurnedDay2 + calBurnedDay3
                     + calBurnedDay4 + calBurnedDay5 + calBurnedDay6
                     + calBurnedDay7;

    cout << "Average number of calories burned each day: "
         << calBurnedInAWeek / 7 << endl;

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
Enter calories burned day 1: 375

Enter calories burned day 2: 425

Enter calories burned day 3: 270

Enter calories burned day 4: 190

Enter calories burned day 5: 350

Enter calories burned day 6: 200

Enter calories burned day 7: 365

Average number of calories burned each day: 310
```

Consider the following statements, in which `calBurnedInAWeek` and `calBurnedInOneDay` are variables of the type `int`.

1. `calBurnedInAWeek = 0;`
2. `cin >> calBurnedInOneDay;`
3. `calBurnedInAWeek = calBurnedInAWeek + calBurnedInOneDay;`

The first statement initializes `calBurnedInAWeek` to 0. Next, let us execute statements 2 and 3 three times.

St.	Execution of the Statement	Effect
2	`cin >> calBurnedInOneDay;`	`calBurnedInOneDay = 375`
3	`calBurnedInAWeek = calBurnedInAWeek + calBurnedInOneDay;`	`calBurnedInAWeek = 0 + 375` `= 375`
2	`cin >> calBurnedInOneDay;`	`calBurnedInOneDay = 425`
3	`calBurnedInAWeek = calBurnedInAWeek + calBurnedInOneDay;`	`calBurnedInAWeek = 375 + 425` `= 800`
2	`cin >> calBurnedInOneDay;`	`calBurnedInOneDay = 270`
3	`calBurnedInAWeek = calBurnedInAWeek + calBurnedInOneDay;`	`calBurnedInAWeek = 800 + 270` `= 1070`

`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.)

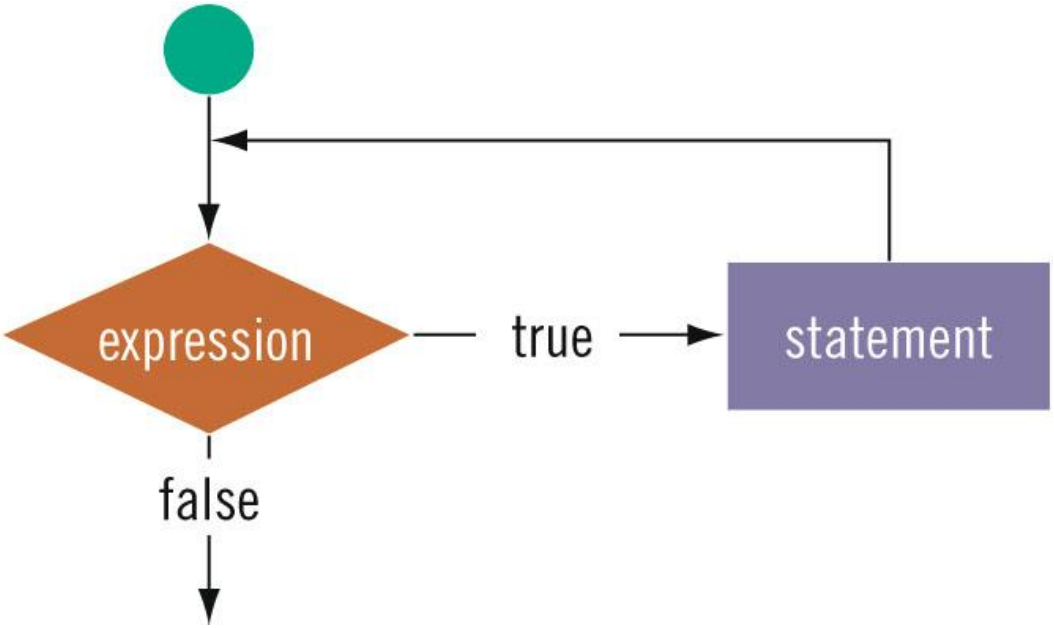


FIGURE 5-1 `while` loop

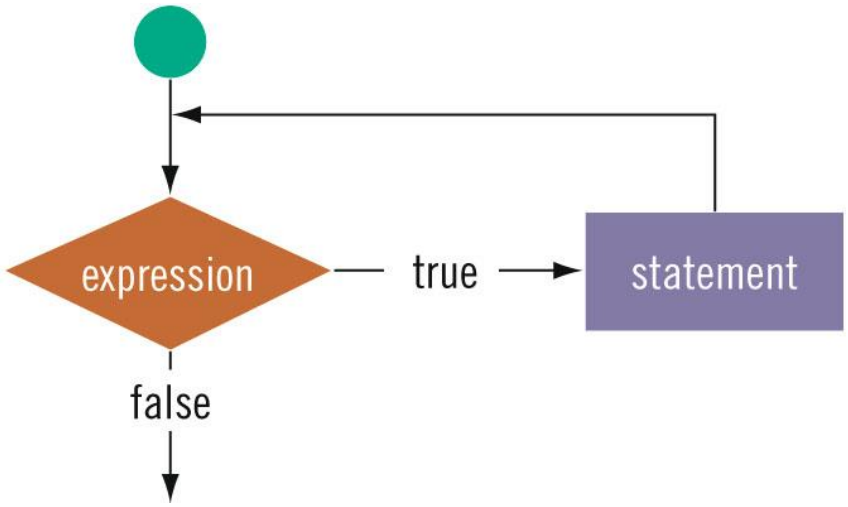
```
#include <iostream>

using namespace std;

int main()
{
    int calBurnedInADay;
    int calBurnedInAWeek;
    int day;

    day = 1;
    calBurnedInAWeek = 0;

    while (day <= 7)
    {
        cout << "Enter calories burned day " << day << ": ";
        cin >> calBurnedInADay;
        cout << endl;

        calBurnedInAWeek = calBurnedInAWeek + calBurnedInADay;
        day = day + 1;
    }

    cout << "Average number of calories burned each day: "
         << calBurnedInAWeek / 7 << endl;

    return 0;
}
```

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

EXAMPLE 5-1

Consider the following C++ program segment:

```
int i = 0;                          //Line 1

while (i <= 20)                     //Line 2
{                                   //Line 3
     cout << i << " ";              //Line 4
     i = i + 5;                     //Line 5
}                                   //Line 6

cout << endl;                       //Line 7
```



Iteration	Value of i	Expression in Line 2	Statements in Lines 4 and 5
1	i = 0	i <= 20 is true	Output: 0 i = i + 5 = 0 + 5 = 5
2	i = 5	i <= 20 is true	Output: 5 i = i + 5 = 5 + 5 = 10
3	i = 10	i <= 20 is true	Output: 10 i = i + 5 = 10 + 5 = 15
4	i = 15	i <= 20 is true	Output: 15 i = i + 5 = 15 + 5 = 20
5	i = 20	i <= 20 is true	Output: 20 i = i + 5 = 20 + 5 = 25
6	i = 25	i <= 20 is false	The loop terminates

The preceding `while` loop produces the following output:

```
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
      .
      .
      .
}
```

```
//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

       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
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 1: Enter the number of integers in the list: 12

Line 6: Enter 12 integers.
8 9 2 3 90 38 56 8 23 89 7 2
Line 11: The sum of the 12 numbers = 335
Line 13: The average = 27
```

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
     .
     .
     .
}
```

Suppose you want to read some positive integers and average them, but you do not have a preset number of data items in mind. Suppose the number **-999** marks the end of the data. You can proceed as follows.

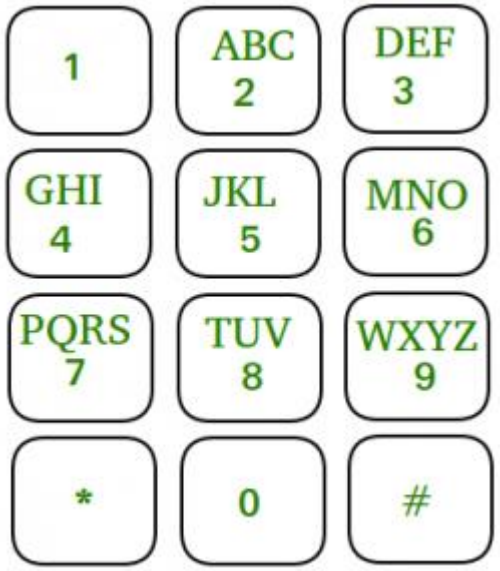
```
//Program:  Sentinel-Controlled Loop

#include <iostream>

using namespace std;

const int SENTINEL = -999;

int main()
{
    int number;       //variable to store the number
    int sum = 0;      //variable to store the sum
    int count = 0;    //variable to store the total
                      //numbers read

cout << "Line 1: Enter integers ending with "
     << SENTINEL << endl;                          //Line 1
cin >> number;                                     //Line 2

while (number != SENTINEL)                         //Line 3
{
    sum = sum + number;                            //Line 4
    count++;                                       //Line 5
    cin >> number;                                 //Line 6
}

cout << "Line 7: The sum of the " << count
     << " numbers is " << sum << endl;             //Line 7

if (count != 0)                                    //Line 8
    cout << "Line 9: The average is "
         <<  sum / count << endl;                  //Line 9
else                                               //Line 10
    cout << "Line 11: No input." << endl;          //Line 11

return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 1: Enter integers ending with -999
34 23 9 45 78 0 77 8 3 5 -999
Line 7: The sum of the 10 numbers is 282
Line 9: The average is 28
```

Example 5-5: Telephone Digits

- Example 5-5 provides an example of a sentinel-controlled loop
- The program converts uppercase letters to their corresponding telephone digit



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`

```
//Flag-controlled while loop.
//Number guessing game.

#include <iostream>                                                    //Line 1
#include <cstdlib>                                                     //Line 2
#include <ctime>                                                       //Line 3

using namespace std;                                                   //Line 4

int main()                                                             //Line 5
{                                                                      //Line 6
      //declare the variables                                          Line 7
    int num;              //variable to store the random
                          //number                                     Line 8
    int guess;            //variable to store the number
                          //guessed by the user                        Line 9
    bool isGuessed;       //boolean variable to control
                          //the loop                                   Line 10

    srand(time(0));                                                    //Line 11
    num = rand() % 100;                                                //Line 12

    isGuessed = false;                                                 //Line 13

    while (!isGuessed)                                                 //Line 14
    {                                                                  //Line 15
        cout << "Enter an integer greater"
             << " than or equal to 0 and "
             << "less than 100: ";                                     //Line 16

        cin >> guess;                                                  //Line 17
        cout << endl;                                                  //Line 18

        if (guess == num)                                              //Line 19
        {                                                              //Line 20
            cout << "You guessed the correct "
                 << "number." << endl;                                 //Line 21
            isGuessed = true;                                          //Line 22
        }                                                              //Line 23
        else if (guess < num)                                          //Line 24
            cout << "Your guess is lower than the "
                 << "number.\n Guess again!"
                 << endl;                                              //Line 25
        else                                                           //Line 26
            cout << "Your guess is higher than "
                 << "the number.\n Guess again!"
                 << endl;                                              //Line 27
    } //end while                                                     //Line 28

    return 0;                                                          //Line 29
}                                                                      //Line 30
```

Sample Run: In this sample run, the user input is shaded.

```
Enter an integer greater than or equal to 0 and less than 100: 45

Your guess is higher than the number.
 Guess again!
Enter an integer greater than or equal to 0 and less than 100: 20

Your guess is lower than the number.
 Guess again!
Enter an integer greater than or equal to 0 and less than 100: 35

Your guess is higher than the number.
 Guess again!
Enter an integer greater than or equal to 0 and less than 100: 28

Your guess is lower than the number.
 Guess again!
Enter an integer greater than or equal to 0 and less than 100: 32

You guessed the correct number.
```

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

The following is an example of an EOF-controlled **while** loop:

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

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

Notice that here, the variable **cin** acts as the loop control variable.

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))
{
    . . .
}
```