International University of Technology Twintech
جامعة تونتك الدولية للتكنولوجيا

Faculty of Computer Science and Information Technology
Department of Business Information Technology
Sana'a - Yemen

Lab Manual # 6

Prepared by Eng. Enas Aldahbali

Student Name			
Department			
Roll #			
Section			
Lab Date	-10-2024		
Submission Date			
Lab Grade:	10	**Obtained Grade**	
Instructor's Signature:			

A. ***Title:*** **More Flow of Control**

B. ***Objectives of this lab:***

- Learn about Boolean Expressions
- Learn about the *switch* statement as means to perform multi-way branches
- Learn about the *while ....do while* statement as another means to control the flow

Activity 5.1 - Using Boolean Expression

The order in which the statements in a program are performed is called **flow of control**. In the previous labs, you saw programs in which *if, if...else, and in this lab you will learn while,* and *do ... while* statements which will b used. These statements will used to specify flow of control in your programs. The concept behind all these methods is the same, your program comes to a statement that is either **true** or **false**. Such a statement is called a **Boolean expression**. The program then will perform a different task depending on the statement being true or false. For example: Suppose somewhere in your program you have:

*if(**x < 10 && y > 12**).*

In this statement, the Boolean expression is ***x < 10 && y > 12***. This expression is true or false. Of course, there are several possibilities here:

Possibility 1: x >= 10 and y <= 12	the expression will be: **false**
Possibility 2: x >= 10 and y > 12	the expression will be: **false**
Possibility 3: x < 10 and y <= 12	the expression will be: **false**
Possibility 4: x < 10 and y > 12	the expression will be: **true**

So your program will go one way when the expression evaluates to false and another way when the expression evaluates to true. So its flow is controlled by this Boolean expression. Note that the expression inside ( ) in the *if statement* can be very simple or very complicated. But, in either case that statement will evaluate to true or false, regardless of its complexity. When an expression is complex, your compiler uses the **precedence rules** to decide the order in which each different part of the expression should be executed. In the following expression, for example, the order of execution is decided by these rules:

if ((x + 2) > 3 || (x + 1) < -3)

will be executed in this order:

*if (**(**(x + 2) > 3**)** || **(**(x + 1) < -3**)**)*

The rules are listed in Display 7.2 of the textbook. Here, first the expression ***(***(x + 2) > 3***)*** will be evaluated and then the expression ***(***(x + 1) < -3***)***. Because there is an OR, "| |", between the two expressions, if the first one evaluates to true, the second one never gets evaluated. This is because, in an OR expression, once either one of the two expressions is true, the entire expression evaluates to true.

Precedence Rules

The unary operators +, –, ++, --, and !.
The binary arithmetic operations *, /, %
The binary arithmetic operations +, –
The Boolean operations <, >, <=, >=
The Boolean operations ==, !=
The Boolean operations &&
The Boolean operations ||

Highest precedence (done first)

Lowest precedence (done last)

Exercise 5.1

Following is a list of some Boolean expressions. Carefully go through each one of them and determine whether they evaluate to true or false.

Expression	Answer
A: ((count == 0) && (limit < 20))	
B: (count == 0 && limit < 20)	
C: ((limit > 12) || (count < 5))	
D: (!(count == 5))	
E: ((count == 1) && (x < y))	
F: ((count < 10) || (x < y))	
G: (!(((count < 10) || (x < y)) && (count >= 0)))	
H: (((limit/count) > 7) || (limit < 20))	
I: ((limit < 20) || ((limit/count) > 7))	
J: (((limit/count) > 7) && (limit < 0))	
K: ((limit < 0) && ((limit/count) > 7))	
L: ((5 && 7) + (!6))	

Now, create a file ex71.cpp and cut and paste the following program in it. Compile and run the program and see how well your answers to the above expressions match the program's

Code:-

```
// ex51.cpp - This program illustrates the Boolean expressions and the
// order of precedence
#include<iostream>
using namespace std;
int main()
{
   int count = 0, limit = 10;
   int x,y;
   cout << "a  " << ( (count == 0) && (limit < 20)) << "\n";
   cout << "b  " << ( count == 0 && limit < 20 ) << "\n";
   cout << "c  " << ( (limit > 12) || (count < 5) ) << "\n";
   cout << "d  " << ( !(count == 5) ) << "\n";
   cout << "e  " << ( (count == 1) && (x < y) ) << "\n";
   cout << "f  " << ( (count < 10) || (x < y) ) << "\n";
   cout << "g  " << ( !( (count < 10) || (x < y)) && (count >= 0)) ) << "\n";
   cout << "h  " << ( ((limit/count) > 7) || (limit < 20) ) << "\n";
   cout << "i  " << ( (limit < 20) || ((limit/count) > 7) ) << "\n";
   cout << "j  " << ( ((limit/count) > 7) && (limit < 0) ) << "\n";
   cout << "k  " << ( (limit < 0) && (( limit/count) > 7) ) << "\n";
   cout << "l  " << ( (5 && 7) + (!6) ) << "\n";
   return 0;
}
```

Note that some of the statements may cause run-time errors. Find those statements and explain why the errors have happened. To make the program go to the next statement, simply comment (//) the line that causes the run-time error. Recompile and run the program again until all lines that cause a run-time error are commented.

Code:-

```
1
2
3
4
```

What is the difference between h and i?

Would it make any difference if in "j" we switch the first and the second statements, i.e., we use:

```
cout << "j  " << ( (limit < 0) && ((limit/count) > 7)  ) << "\n";
```

Answers:-

Activity 5.2 : C++ switch..case

The switch statement allows us to execute a block of code among many alternatives.

The syntax of the `switch` statement in C++ is:

```
switch (expression)  {
    case constant1:
        // code to be executed if
        // expression is equal to constant1;
        break;
    case constant2:
        // code to be executed if
        // expression is equal to constant2;
        break;
        .
        .
        .
    default:
        // code to be executed if
        // expression doesn't match any constant
}
```

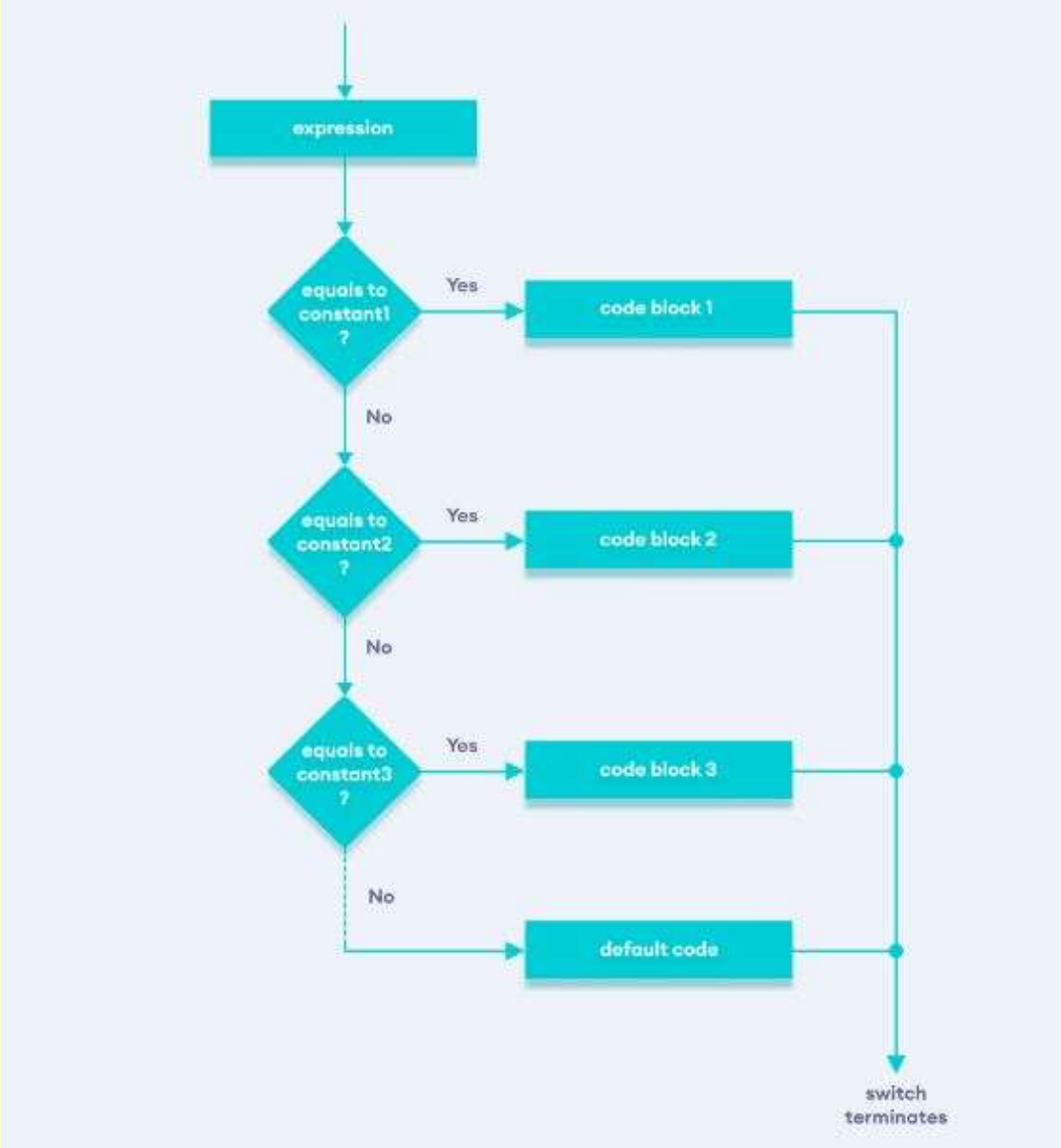
How does the switch statement work?

The `expression` is evaluated once and compared with the values of each `case` label.

- If there is a match, the corresponding code after the matching label is executed. For example, if the value of the variable is equal to `constant2`, the code after `case constant2:` is executed until the `break` statement is encountered.
- If there is no match, the code after `default:` is executed.

Note: We can do the same thing with the `if...else..if` ladder. However, the syntax of the `switch` statement is cleaner and much easier to read and write.

Flowchart of switch Statement:-



Example: Create a Calculator using the switch Statement

Code:-

```
// Program to build a simple calculator using switch Statement
#include <iostream>
using namespace std;

int main() {
    char oper;
    float num1, num2;
    cout << "Enter an operator (+, -, *, /): ";
    cin >> oper;
    cout << "Enter two numbers: " << endl;
    cin >> num1 >> num2;

    switch (oper) {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1 + num2;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1 - num2;
            break;
        case '*':
            cout << num1 << " * " << num2 << " = " << num1 * num2;
            break;
        case '/':
            cout << num1 << " / " << num2 << " = " << num1 / num2;
            break;
        default:
            // operator is doesn't match any case constant (+, -, *, /)
            cout << "Error! The operator is not correct";
            break;
    }

    return 0;
```

result 2 displayed on the screen

```
Enter an operator (+, -, *, /): -
Enter two numbers:
2.3
4.5
2.3 - 4.5 = -2.2
```

result 3 displayed on the screen

```
Enter an operator (+, -, *, /): *
Enter two numbers:
2.3
4.5
2.3 * 4.5 = 10.35
```

result 4 displayed on the screen

```
Enter an operator (+, -, *, /): /
Enter two numbers:
2.3
4.5
2.3 / 4.5 = 0.511111
```

result 5 displayed on the screen

```
Enter an operator (+, -, *, /): ?
Enter two numbers:
2.3
4.5
Error! The operator is not correct.
```

In the above program, we are using the `switch...case` statement to perform addition, subtraction, multiplication, and division.

How This Program Works:-

1. We first prompt the user to enter the desired operator. This input is then stored in the `char` variable named *oper*.
2. We then prompt the user to enter two numbers, which are stored in the float variables *num1* and *num2*.
3. The `switch` statement is then used to check the operator entered by the user:
    - If the user enters `+`, addition is performed on the numbers.
    - If the user enters `-`, subtraction is performed on the numbers.
    - If the user enters `*`, multiplication is performed on the numbers.
    - If the user enters `/`, division is performed on the numbers.
    - If the user enters any other character, the default code is printed.

Notice that the `break` statement is used inside each `case` block. This terminates the `switch` statement.

If the `break` statement is not used, all cases after the correct `case` are executed.

Activity 5.3 - loops

A **loop** is any program construction that repeats a statement or sequence of statements a number of times. The simple *while* loops and *do-while* loops that we have already seen are examples of loops. The statement (or group of statements) to be repeated in a loop is called the **body** of the loop, and each repetition of the loop body is called an **iteration** of the loop. The two main design questions when constructing loops are: What should the loop body be? How many times should the loop body be iterated?

For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop.

That was just a simple example; we can achieve much more efficiency and sophistication in our programs by making effective use of loops.

There are **3** types of loops in C++.

1. While loop
2. do...while loop
3. for loop

1.C++ while Loop:-

The syntax of the while loop is:

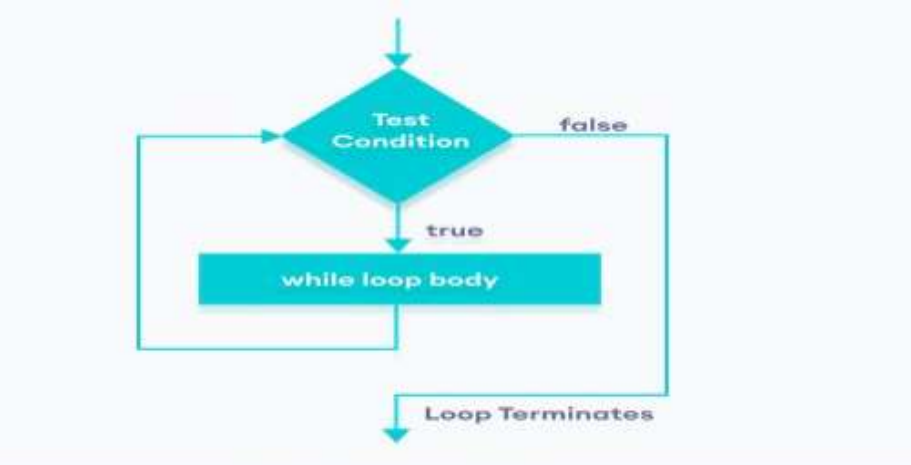
```
while( Boolean_Expression)
{
   Statement 1;
   Statement 2;
   ...
   Statement Last;

 update Action;
}
```

Here,

- A `while` loop evaluates the `condition`
- If the `condition` evaluates to `true`, the code inside the `while` loop is executed.
- The `condition` is evaluated again.
- This process continues until the `condition` is `false`.
- When the `condition` evaluates to `false`, the loop terminates.

Flowchart of while Loop:-



Flowchart of C++ while loop

Example 1: Display Numbers from 1 to

Code:-

```
1  #include <iostream>
   // C++ Program to print numbers from 1 to 5
2  #include <iostream>
   using namespace std;
3  int main() {
       int i = 1;
4    // while loop from 1 to 5
   while (i <= 5) {
         cout << i << " ";
         ++i;
    }
```

result displayed on the screen

```
1 2 3 4 5
```

Here is how the program works.

Iteration	Variable	i <= 5	Action
1st	`i = 1`	`true`	`1` is printed and `i` is increased to `2`.
2nd	`i = 2`	`true`	`2` is printed and `i` is increased to `3`.
3rd	`i = 3`	`true`	`3` is printed and `i` is increased to `4`
4th	`i = 4`	`true`	`4` is printed and `i` is increased to `5`.
5th	`i = 5`	`true`	`5` is printed and `i` is increased to `6`.
6th	`i = 6`	`false`	The loop is terminated

Example 2: Sum of Positive Numbers Only:-

Code:-

```
// program to find the sum of positive numbers
// if the user enters a negative number, the loop ends
// the negative number entered is not added to the sum
#include <iostream>
using namespace std;
int main() {
    int number;
    int sum = 0;
// take input from the user
    cout << "Enter a number: ";
    cin >> number;
   while (number >= 0) {
        // add all positive numbers
        sum += number;
 // take input again if the number is positive
        cout << "Enter a number: ";
        cin >> number;
    }
// display the sum
    cout << "\nThe sum is " << sum << endl;
    return 0;
}
```

Result 1 displayed on the screen

```
Enter a number: 6
Enter a number: 12
Enter a number: 7
Enter a number: 0
Enter a number: -2
The sum is 25
```

In this program, the user is prompted to enter a number, which is stored in the variable *number*.

In order to store the sum of the numbers, we declare a variable *sum* and initialize it to the value of `0`.

The `while` loop continues until the user enters a negative number. During each iteration, the number entered by the user is added to the *sum* variable.

When the user enters a negative number, the loop terminates. Finally, the total sum is displayed.

The following C++ program computes the average of 6 numbers.

Code:-

```
// P5_6.cpp - Read and compute the average for 6 integers, display the result.
#include <iostream.h>
using namespace std;
int main(void)
{
      int x,y,z, p, q, r;  // (A)
      double average;
      // prompt the user:
      cout << "Enter six grades separated by spaces, then press <Enter>:" << endl;
      // read and store six integers:
      cin >> x >> y >> z >> p >> q >> r;  // (B)
      average = (x + y + z + p + q + r)/6;  // (C)
      cout << "The average is " << average << endl;
      return 0;
}
```

If we wanted to compute the average of three numbers, we could make some changes in this program. The major changes will made in the lines that are in blue font, also, marked with (A), (B), and (C). Imagine, computing the average of 1000 numbers using the above program and method. That would make things more complicated and the code more tedious. We can simplify this computation using a *loop*. Following is a new version of the above program written using a *while loop*.

An Example for *while loop:-*

Code:-

```
// P5.6.cpp - Read and average 6 integers, print the result.
#include <iostream.h>
using namespace std;
int main(void)
{  int x;
      int count = 0;   // (1) initialize a counter to 0 to count number of grades
      double sum = 0; // initialize the sum to 0 to make sure the sum at the beginning is 0
      double average;
      // prompt the user:
      cout << "Enter six grades separated by a single space, then press <Enter>: ";
      while( count < 6) // (2) read six grades and compute their sum, count ensures 6 entries
     { // read each number and compute the sum:
          cin >> x;
          sum = sum + x;
          count++;  // (3) update the count  }
      cout << endl;
      average = sum/6;  // compute the average, total divided by the number of grades
      cout << "The average is " << average << endl;
      return 0; }
```

Remarks

In either case, if you wish to have a good working loop (any loop), there are three things that you have to always remember:

1) initialization, whatever controls the loop and is checked by the boolean expression,

2) valid condition, the boolean expression must correctly check the controlling variable, and

3) change, the controlling variable must be changed so that the boolean expression checks a new condition each time. If the condition does not change in the body of the loop, then the loop checks the same thing infinite number of times and you will have an infinite loop. Your program never stops and you have to terminate it using Ctrl-C or by killing the process. I am sure we will run into this problem in the lab, so you will learn how.

2.C++ do...while Loop

The `do...while` loop is a variant of the `while` loop with one important difference: the body of `do...while` loop is executed once before the `condition` is checked.

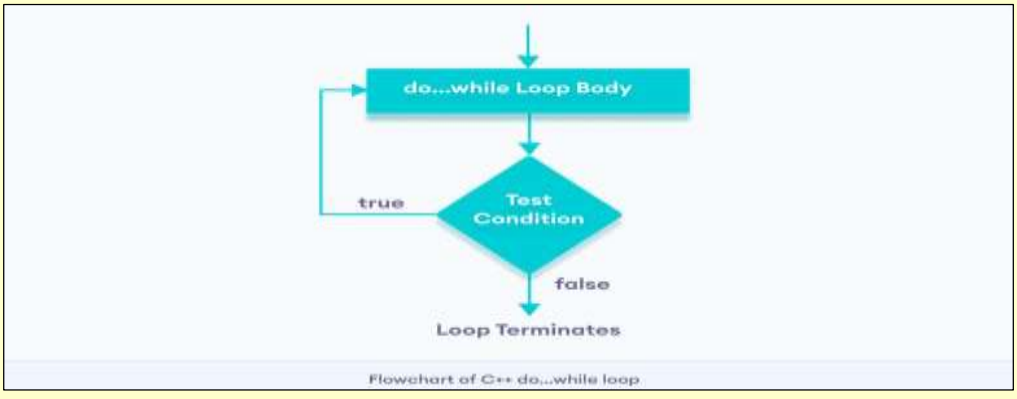
```
do

{
   Statement 1;
   Statement 2;
   ...
   Statement Last;
   update Action;
} while( Boolean_Expression) ;
```

Here,

- The body of the loop is executed at first. Then the condition is evaluated.
- If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- The condition is evaluated once again.
- If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- This process continues until the condition evaluates to false. Then the loop stops.

Flowchart of do...while Loop:-



Example 5.9: Display Numbers from 1 to 5

Code:-

```
/ C++ Program to print numbers from 1 to 5
#include <iostream>
using namespace std;

int main() {
   int i = 1;

   // do...while loop from 1 to 5
   do {
      cout << i << " ";
      ++i;
   }
   while (i <= 5);

   return 0;
}
```

Result 1 displayed on the screen

```
1 2 3 4 5
```

Here is how the program works.

Iteration	Variable	i <= 5	Action
	`i = 1`	not checked	`1` is printed and `i` is increased to 2
1st	`i = 2`	`true`	`2` is printed and `i` is increased to 3
2nd	`i = 3`	`true`	`3` is printed and `i` is increased to 4
3rd	`i = 4`	`true`	`4` is printed and `i` is increased to 5
4th	`i = 5`	`true`	`5` is printed and `i` is increased to **6**
5th	`i = 6`	`false`	The loop is terminated

Exercise #1:-

Write a C++ program that print phrase hello 10 times?

Write Your code Here

```
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
```

Exercise #2:-

Write a C++ program that enter 10 numbers then print the sum only of odd numbers ?

Code:-

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Assignment #1:-

Write C++ program that print these series 5 10 15 20……100.

Write Your code Here

```
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
```

Assignment #2:-

write a C++ program that enter 15 number and print the product of numbers?

Write Your code Here

```
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
```