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

Lab Manual # 7&8

Prepared by Eng. Enas Aldahbali

Student Name			
Department			
Roll #			
Section			
Lab Date	-11-2024		
Submission Date			
Lab Grade:	10	**Obtained Grade**	
Instructor's Signature:			

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

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

- Looping with Variable Number of Loops
- Learn about Simple Flow of Control Event-controlled *while* and *do ... while* loops
- Learn about the *for* statement as another means to control the flow
- Learn about break statement with examples
- Learn about continue statement with examples

Activity 7.1 - *Looping with Variable Number of Loops*

There are times that you do not know, up front, how many times a computation must be done. For example, you do not know how many numbers are going to be entered every time that you run the program. Thus, you want to make the loop flexible. Here is another version of the above program with variable number of loops. Suppose the user first provides number of values for which he/she wants to find the average, then he/she would enter those values.

Code:-

```
// P6_1.cpp - Read and average N integers, print the result.
#include <iostream>
using namespace std;
int main(void)
{     int x;
      int count = 0;   // (1) initialize a counter to 0 to count number of values
      int N; // Number of values for which the average must be computed.
      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 number of values, N, to be read in <Enter>:" << endl;
      cin >> N;
      while( count < N) // (2) read N grades and compute their sum, count ensures N entries
     {  // read each number and compute the sum:
          cout << "\n Enter a grade <Enter>: ";
          cin >> x;
          sum = sum + x;
          count++;  // (3) update the count
      }
      if(N == 0)
           cout << "You haven't enter 0 number, no average will be computed, bye \n";
      else{
        average = average = sum/N;
         cout << "The average of these " << N << " grades is " << average << endl;
      }
      return 0;
               }
```

In the previous parts, you learned to use a while loop with fix number of loops and then with variable number of loops. In both cases, you used a *counting* mechanism to stop the looping. There are times that you want to use a condition to stop the looping. For example, in the previous program, suppose you want to ask the user to see whether he/she wishes to enter a new grade. For now, we will use an integer to control the looping number. Here, the user will enter a 1 if he/she wishes to loop one more time and will enter a 0 or any other number to stop the looping.

Code:-

```
// P6_2.cpp - Read and average some integers, print the result.
// This program continue asking for a new number until the user enters a 0 to terminate the program
#include <iostream>
using namespace std;
int main(void)
{
      int x;
      int count = 0;   // (1) initialize a counter to 0 to count number of values
      int choice = 1; // This is the choice that controls the looping continuation or termination
      double sum = 0; // initialize the sum to 0 to make sure the sum at the beginning is 0
      double average;
      while( choice == 1) // (2) read N grades and compute their sum, count ensures N entries
      {
          // read each number and compute the sum:
          cout << "\n Enter a grade <Enter>: ";
          cin >> x;
          sum = sum + x;
          count++;  // (3) update the count
         // prompt the user:
          cout << "Do you wish to enter another grade? (1 for yes and 0 or other key for no): " << endl;
          cin >> choice;
      }
      if(count == 0)
          cout << "You haven't entered any number, no average will be computed, bye \n";
      else{
        average = sum/count;  //Notice that we have divided by count this time
          cout << "The average of these " << count << " grades is " << average << endl;
      }
      return 0;
}
```

Exercise 7-1

In program 6_1.cpp, right at the beginning you have initialized the choice to 1. You did that to get the *while loop* to run at least once. If you use a *do ... while* instead, you wouldn't need to initialize the choice. Re-write the above program, call the new program ex24_1.cpp, so that is uses *do ... while*. Do not initialize the choice to 1 this time and compile and run the program. Does the program work the same way?

Code Answer:-

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

Activity 7.2 Infinite while loop:-

If the `condition` of a loop is always `true`, the loop runs for infinite times (until the memory is full). For example,

```
// infinite while loop
while(true) {
   // body of the loop
}
```

Here is an example of an infinite `do...while` loop.

Code:-

```
#include <iostream>
using namespace std;
int main(){
  int i=1;
  /* The loop would continue to print
   * the value of i until the given condition
   * i<=6 returns false.
   */
  while(i<=6){
    cout<<"Value of variable i is: "<<i<<endl;
 i++;
return 0;
  }
```

In the above programs, the condition is always true. Hence, the loop body will run for infinite times.

Code:-

```
```

Activity 7.3 C++ for loop

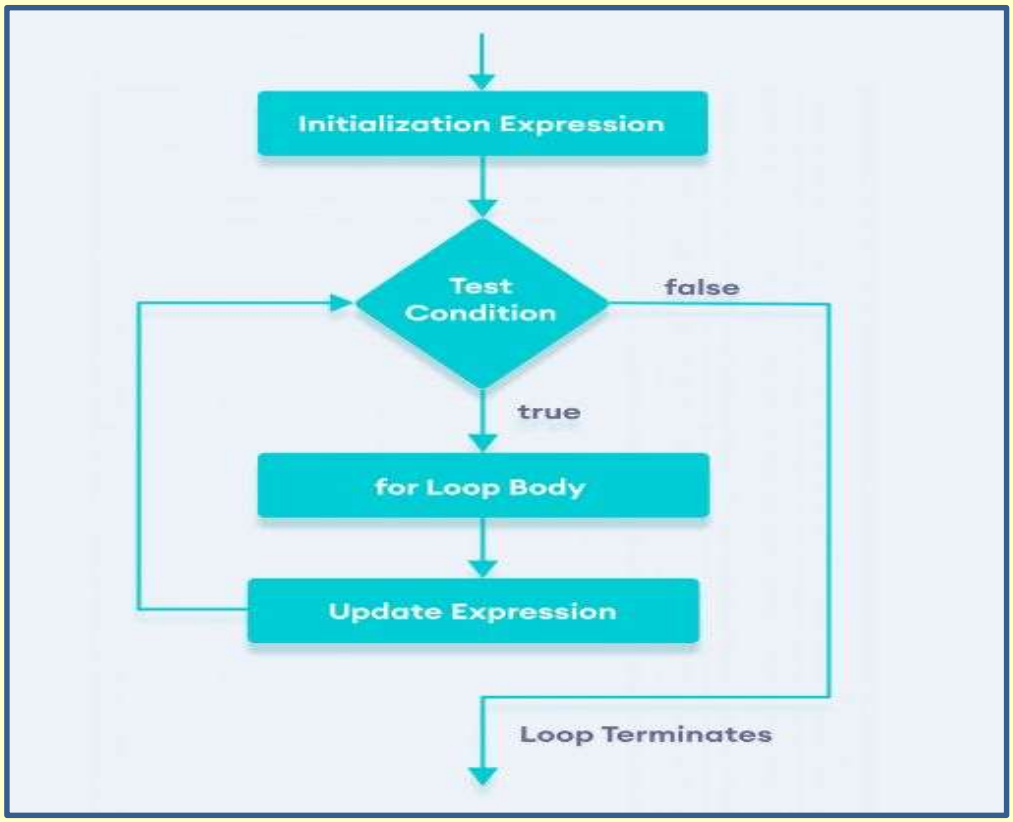
The syntax of for-loop is:

```
for (initialization; condition; update) {
   // body of-loop
}
```

Here,

- `initialization` - initializes variables and is executed only once
- `condition` - if `true`, the body of `for` loop is executed
  if `false`, the for loop is terminated
- `update` - updates the value of initialized variables and again checks the condition

Flowchart of for Loop in C++



Example 1: Printing Numbers From 1 to 5

Code:-

```
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 5; ++i) {
    cout << i << " ";
  }
  return 0;
}
```

result 1displayed on the screen

```
1 2 3 4 5
```

Here is how this program works

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

Example 2: Display a text 5 times

Code:-

```
// C++ Program to display a text 5 times
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 5; ++i) {
        cout <<  "Hello World! " << endl;
    }
    return 0;
}
```

result 1displayed on the screen

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Here is how this program works

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

Example 3: Find the sum of first n Natural Numbers

Code:-

```
// C++ program to find the sum of first n natural numbers
// positive integers such as 1,2,3,...n are known as natural numbers
#include <iostream>
using namespace std;
int main() {
   int num, sum;
   sum = 0;
   cout << "Enter a positive integer: ";
   cin >> num;
   for (int count = 1; count <= num; ++count) {
      sum += count;
   }
cout << "Sum = " << sum << endl;

   return 0;
}
```

result 1displayed on the screen

```
Enter a positive integer: 10
Sum = 55
```

In the above example, we have two variables *num* and *sum*. The *sum* variable is assigned with *0* and the *num* variable is assigned with the value provided by the user.

Note that we have used a `for` loop.

```
for(int count = 1; count <= num; ++count)
```

Here,

- `int count = 1`: initializes the *count* variable
- `count <= num`: runs the loop as long as *count* is less than or equal to *num*
- `++count`: increase the *count* variable by 1 in each iteration

When *count* becomes `11`, the `condition` is `false` and *sum* will be equal to `0 + 1 + 2 + ... + 10`.

C++ Infinite for loop:-

If the condition in a for loop is always true, it runs forever (until memory is full). For example,

```
// infinite for loop
for(int i = 1; i > 0; i++) {
   // block of code
}
```

In the above program, the `condition` is always `true` which will then run the code for infinite times.

Activity 8.1 C++ break Statement

In C++, the `break` statement terminates the loop when it is encountered.

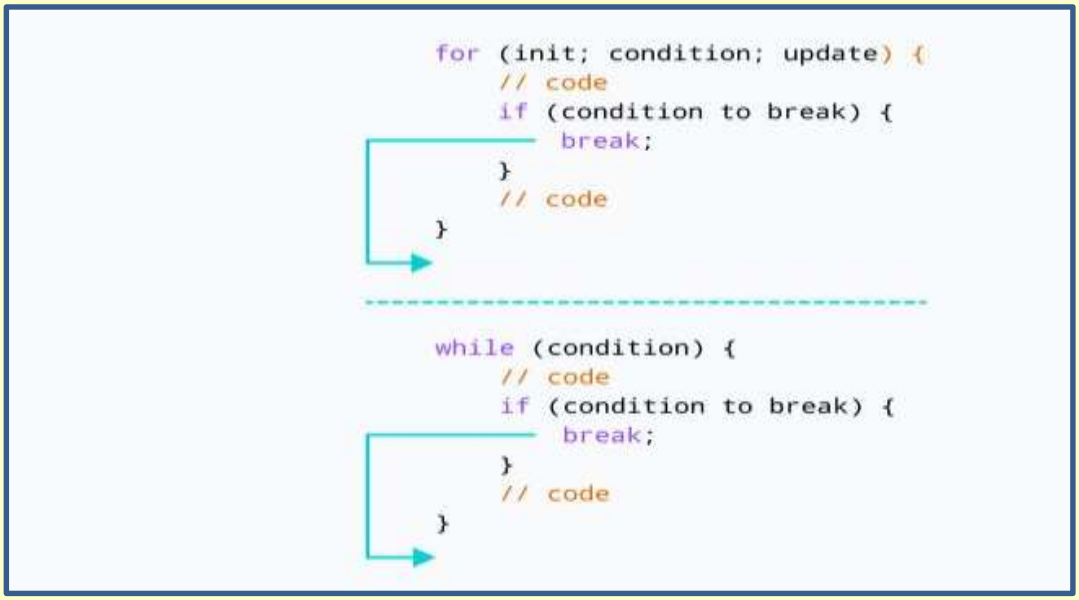
The syntax of the break statement is:

```
break;
```

Before you learn about the `break` statement, make sure you know about:

- C++ for loop
- C++ if...else
- C++ while loop

Working of C++ break Statement



Example 1: break with for loop

Code:-

```
// program to print the value of i
#include <iostream>
using namespace std;
int main() {
 for (int i = 1; i <= 5; i++) {
  // break condition
      if (i == 3) {
         break;
      }
    cout << i << endl;
  }
return 0;
}
```

result displayed on the screen

```
1
2
```

In the above program, the for loop is used to print the value of *i* in each iteration. Here, notice the code:

```
if (i == 3) {
   break;
}
```

his means, when *i* is equal to **3**, the `break` statement terminates the loop. Hence, the output doesn't include values greater than or equal to 3.

Note: The **`break`** statement is usually used with decision-making statements

Example 2: break with while loop

Code:-

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

        // break condition
        if (number < 0) {
            break;
        }
 // add all positive numbers
        sum += number;
    }
// display the sum
    cout << "The sum is " << sum << endl;
  return 0;
}
```

result displayed on the screen

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: -5
The sum is 6.
```

The `break` statement is also used with the `switch` statement.

In the above program, the user enters a number. The `while` loop is used to print the total sum of numbers entered by the user. Here, notice the code,

```
if (number < 0) {
   break;
}
```

This means, when the user enters a negative number, the `break` statement terminates the loop and codes outside the loop are executed.

The `while` loop continues until the user enters a negative number.

break with Nested loop

When `break` is used with nested loops, `break` terminates the inner loop. For example,

Example :-

Code:-

```
// using break statement inside
// nested for loop
#include <iostream>
using namespace std;
int main() {
   int number;
   int sum = 0;
 // nested for loops
 // first loop
   for (int i = 1; i <= 3; i++) {
      // second loop
      for (int j = 1; j <= 3; j++) {
         if (i == 2) {
            break;
         }
         cout << "i = " << i << ", j = " << j << endl;
      }
   }
return 0;
}
```

Result 1 displayed on the screen

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

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.

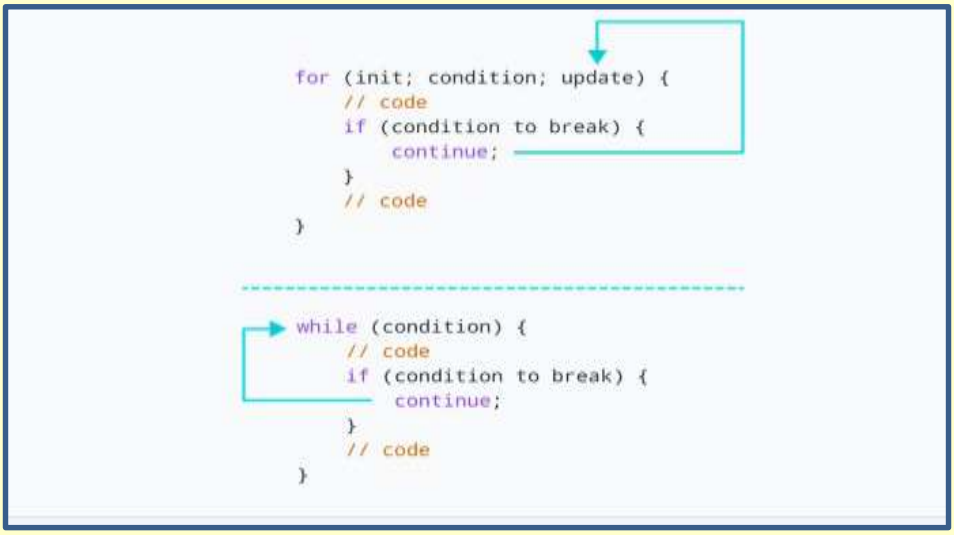
Activity 8.2 C++ continue Statement

In computer programming, the continue statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.

The syntax of the continue statement is:

```
continue;
```

Working of C++ continue Statement



Example 1: continue with for loop

In a for loop, continue skips the current iteration and the control flow jumps to the update expression.

Code:-

```
// program to print the value of i
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 5; i++) {
        // condition to continue
        if (i == 3) {
            continue;
        }
        cout << i << endl;
    }
    return 0;
}
```

Result 1 displayed on the screen

```
1
2
4
5
```

In the above program, we have used the the `for` loop to print the value of *i* in each iteration. Here, notice the code,

```
if (i == 3) {
   continue;
}
```

This means

- When *i* is equal to 3, the continue statement skips the current iteration and starts the next iteration
- Then, *i* becomes 4**,** and the condition is evaluated again.
- Hence, 4 and 5 are printed in the next two iterations.

Note: The continue statement is almost always used with decision-making statements.

Note: The break statement terminates the loop entirely. However, the continue statement only skips the current iteration.

Example 2: continue with while loop

In a `while` loop, `continue` skips the current iteration and control flow of the program jumps back to the `while condition`.

Code:-

```
// program to calculate positive numbers till 50 only
// if the user enters a negative number,
// that number is skipped from the calculation
// negative number -> loop terminate
// numbers above 50 -> skip iteration
#include <iostream>
using namespace std;
int main() {
   int sum = 0;
   int number = 0;
  while (number >= 0) {
      // add all positive numbers
      sum += number;
// take input from the user
      cout << "Enter a number: ";
      cin >> number;
 // continue condition
      if (number > 50) {
         cout << "The number is greater than 50 and won't be   calculated." << endl;
         number = 0;  // the value of number is made 0 again
         continue;
      }
  }// display the sum
   cout << "The sum is " << sum << endl;
  return 0;
}
```

Result 1 displayed on the screen

```
Enter a number: 12

Enter a number: 0

Enter a number: 2

Enter a number: 30

Enter a number: 50
```

In the above program, the user enters a number. The while loop is used to print the total sum of positive numbers entered by the user, as long as the numbers entered are not greater than 50.

Notice the use of the continue statement.

```
if (number > 50){
   continue;
}
```

- When the user enters a number greater than 50, the continue statement skips the current iteration. Then the control flow of the program goes to the condition of while loop.
- When the user enters a number less than 0, the loop terminates.

Note: The continue statement works in the same way for the do...while loops

continue with Nested loop

When continue is used with nested loops, it skips the current iteration of the inner loop. For example,

Code:-

```
// using continue statement inside
// nested for loop
#include <iostream>
using namespace std;
int main() {
   int number;
   int sum = 0;
  // nested for loops
    // first loop
   for (int i = 1; i <= 3; i++) {
      // second loop
      for (int j = 1; j <= 3; j++) {
         if (j == 2) {
            continue;   }
         cout << "i = " << i << ", j = " << j << endl;
      }
}return 0;
        }
```

In the above program, when the `continue` statement executes, it skips the current iteration in the inner loop. And the control of the program moves to the **update expression** of the inner loop.

Hence, the value of `j = 2` is never displayed in the output.

Result 1 displayed on the screen

```
i = 1, j = 1
i = 1, j = 3
i = 2, j = 1
i = 2, j = 3
i = 3, j = 1
i = 3, j = 3
```

Assignment #1:-

1. Given the run of the following C++ code fragment if the user inputs 3, 7, 12, and 5

```
int n, times, total, curr;
cout << "How many times: ";
cin >> times;
total = 0;
for (n = 1; n <= times; n = n + 1)
{
        cout << "Enter an int: ";
        cin >> curr;
        total = total + curr;
        cout << curr << " " << total << endl;
}
```

Answer :Result 1 displayed on the screen

2. What is printed by the following C++ program fragment if the user enters 27?

```
 divisor = 1;
flag = 0;
cout << "Please enter a number: ";
cin >> num; //num is 27
while ( !flag)
{   if ((num % divisor) == 0)
        flag = 1;
        divisor = divisor + 1;
        if (divisor == num) flag = 1;
        cout << "Help me please!" <<endl;
       }
if (divisor == num) cout << "I hate computers!" << endl;
else
cout << "I love computers!" << endl;
```

Answer :Result 1 displayed on the screen

3. What is printed by each of the following program fragments?

a)
```
for (j = 20; j <= 5; j = j -4)
        cout << "Again\n";
```

Answer :Result 1 displayed on the screen

b)
```
for (j = 10; j > 4; j--)
        cout << j << ' ';
```

Answer :Result 1 displayed on the screen

c)
```
n = 5;
r = 1;
do{
        cout << r << "**";
        r = r + 2;
}while (r <= n);
cout << r;
```

Answer :Result 1 displayed on the screen

d)
```
for (i = 1; i <= 5; i++)
  { for (j = 0; j < i; j++)\
      cout << '*';
      cout << endl;
  }
```

Answer :Result 1 displayed on the screen

4. Write the code fragment (not a complete program) that prints the first n positive integers, such that each integer is preceded by two asterisks.

For example, if n = 3 then the output would be **1**2**3

if n = 5 then the output would be **1**2**3**4**5

Answer Code:-

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

5. What is printed by the following while loops?

a)
```
int m = 5;
while (m >= 1)
{ cout << "m = " << m << endl;
        m = m - 2;
        }
```

Answer :Result 1 displayed on the screen

b)
```
sum = 8;
  while (sum < 10)
{
                sum--;
        cout << "sum = " << sum;
           }
```

Answer :Result 1 displayed on the screen