CMT 210 – OBJECT ORIENTED PROGRAMMING 1

Lecture Three Notes

CONTROL STATEMENTS

3.1 INTRODUCTION

This module discusses the statements that control a program's flow of execution. There are three categories of:

a) **Selection statements**, which include the if, the if-else and the switch.
b) **Iteration statements**, which include the for, while, and do-while loops.
c) **Jump statements**, which include break, continue, return, and goto.

3.2 THE IF STATEMENT

The complete form of the if statement is

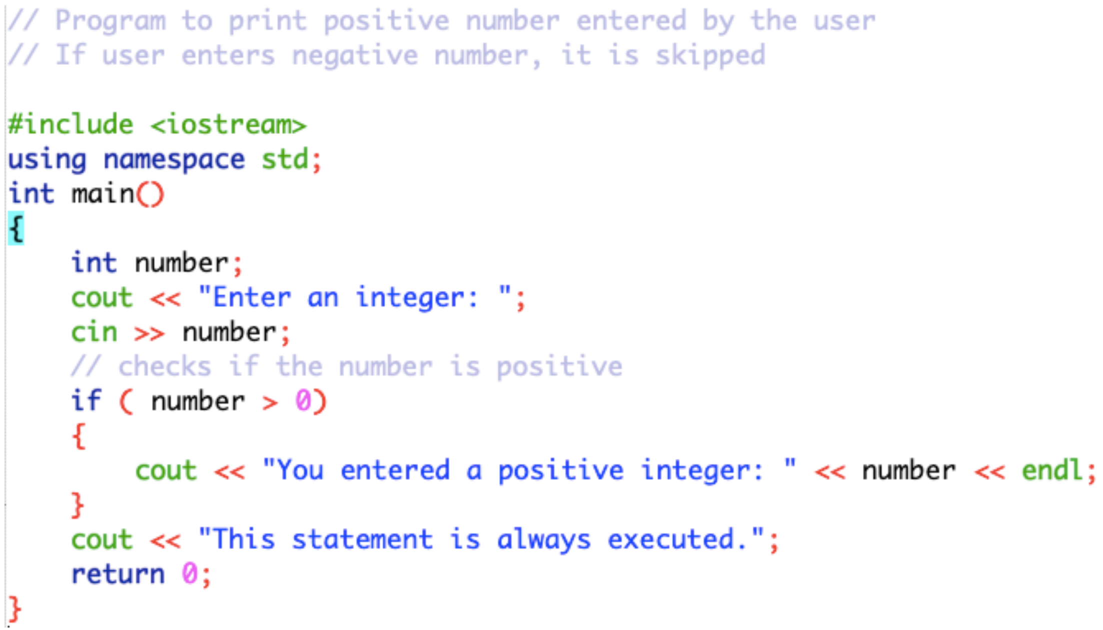
```
if (expression)
        statement
```

The target of the if can also be blocks of statements.

The general form of the if using blocks of statements is

```
if(expression)
{
statement sequence
}
```

If the conditional expression is true, the target of the if will be executed;



```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;
int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    // checks if the number is positive
    if ( number > 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }
    cout << "This statement is always executed.";
    return 0;
}
```

3.3 THE IF-ELSE STATEMENT

The complete form of the if – else statement is

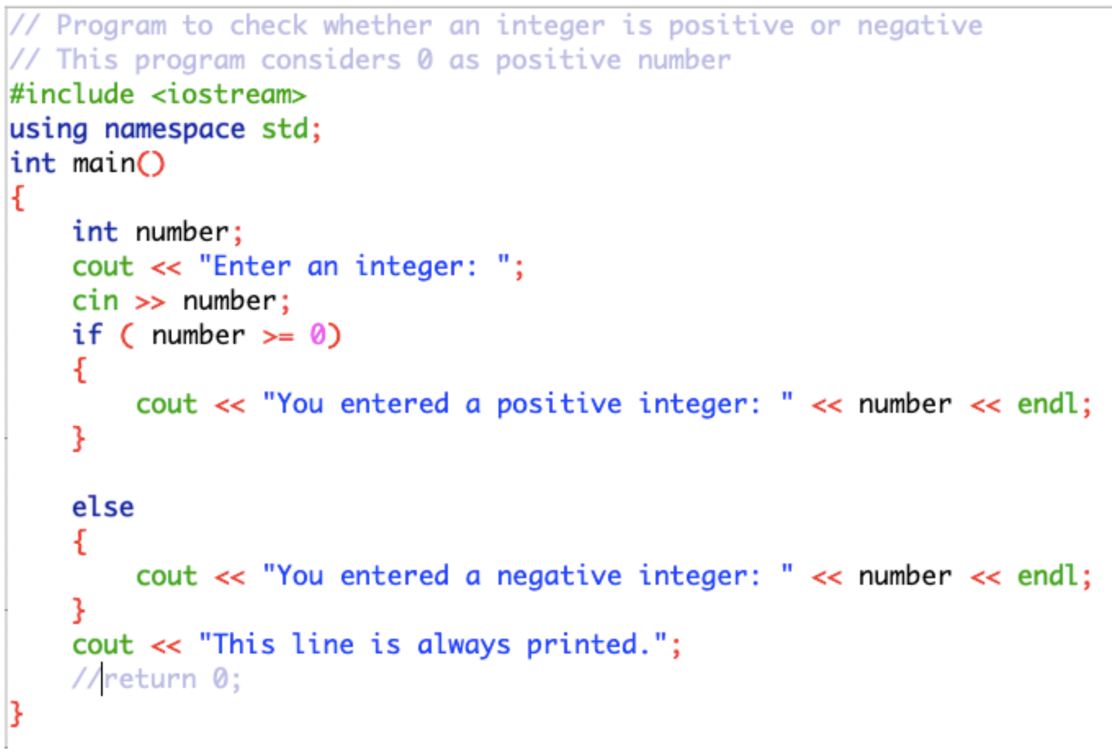
```
if (expression)
        statement
else
        statement
```

The targets of both the if and else can also be blocks of statements.

The general form of the if using blocks of statements is

```
if(expression)
{
statement sequence
}
else
{
statement sequence
}
```

If the conditional expression is true, the target of the if will be executed; otherwise, the target of the else, if it exists, will be executed. At no time will both be executed.



```
// Program to check whether an integer is positive or negative
// This program considers 0 as positive number
#include <iostream>
using namespace std;
int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    if ( number >= 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }

    else
    {
        cout << "You entered a negative integer: " << number << endl;
    }
    cout << "This line is always printed.";
    //return 0;
}
```

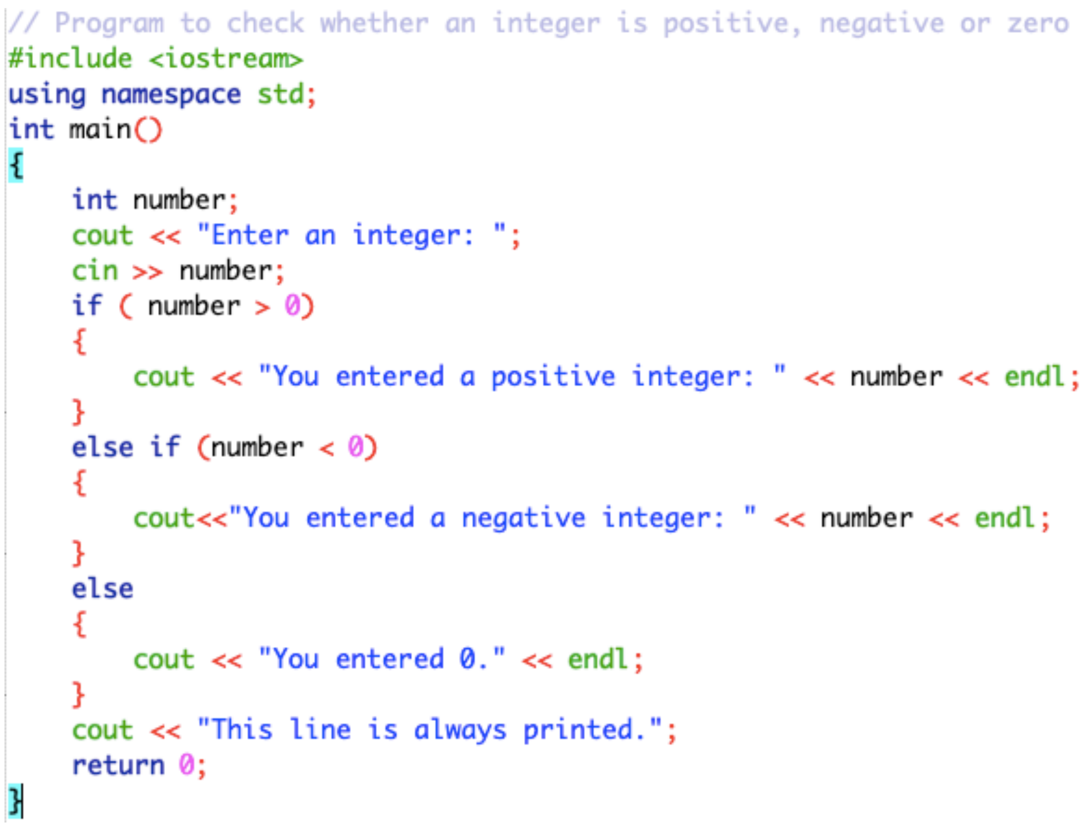
3.4 NESTED IFS:

A nested if is an if statement that is the target of another if or else. Nested ifs are very common in programming. The main thing to remember about nested ifs in C++ is that an else statement always refers to the nearest if statement that is within the same block as the else and not already associated with an else.

The general form of the nested if using blocks of statements is

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
  // statements to be executed if testExpression1 and testExpression2 is false and testExpression3 is true
}
.
.
```

else
{
  // statements to be executed if all test expressions are false
}



```
// Program to check whether an integer is positive, negative or zero
#include <iostream>
using namespace std;
int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    if ( number > 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }
    else if (number < 0)
    {
        cout<<"You entered a negative integer: " << number << endl;
    }
    else
    {
        cout << "You entered 0." << endl;
    }
    cout << "This line is always printed.";
    return 0;
}
```

3.5 THE SWITCH STATEMENT:

The second of C++'s selection statements is the **switch**. *The switch provides for a multiway branch*. Thus, it enables a program to *select among several alternatives.*

Although a series of nested if statements can perform multiway tests, for many situations the switch is a more efficient approach. It works like this: the value of an expression is successively tested against a list of constants.

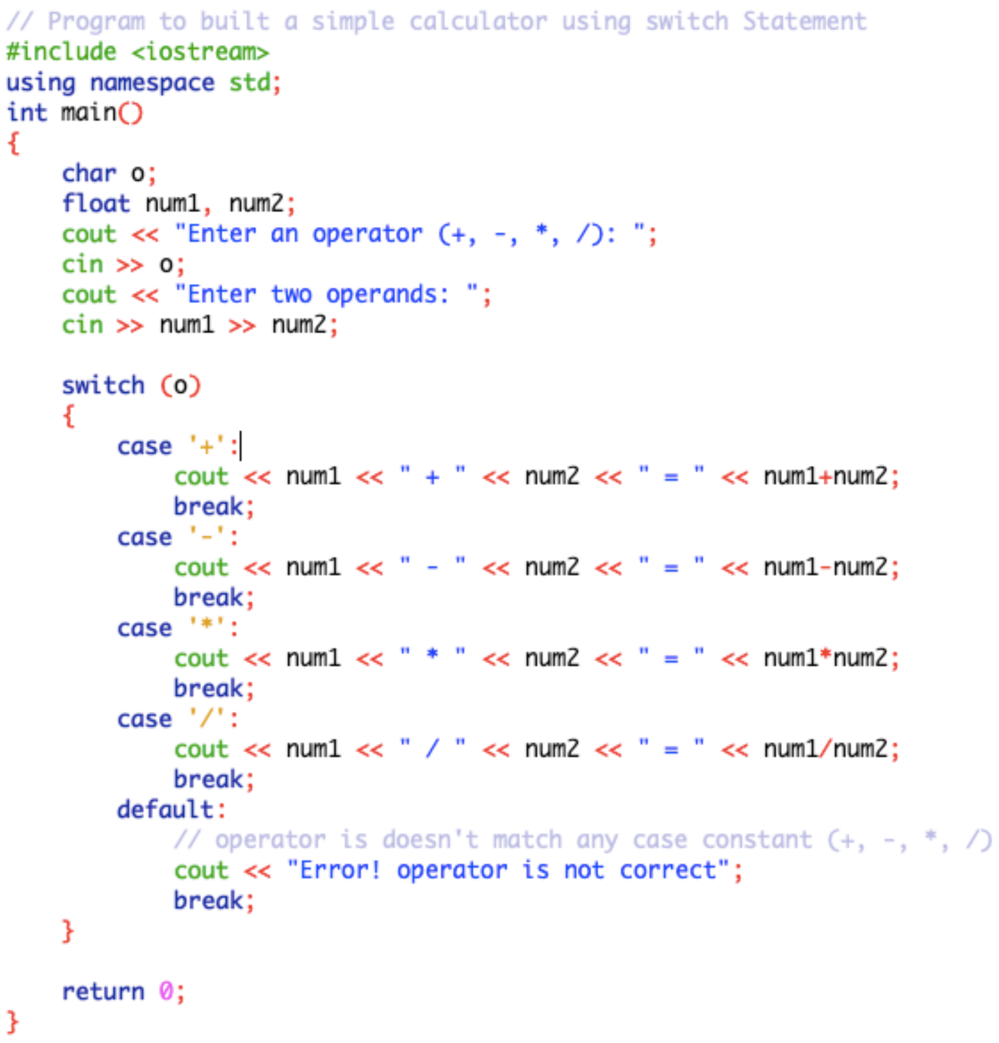
When a match is found, the statement sequence associated with that match is executed. The general form of the switch statement is

```
switch (n)
{
   case constant1:
      // code to be executed if n is equal to constant1;
      break;
```

```
    case constant2:
      // code to be executed if n is equal to constant2;
      break;
      .
      .
      .
    default:
      // code to be executed if n doesn't match any constant
}
```

The switch expression must evaluate to either a character or an integer value. The case constants must be integer or character literals.

The default statement sequence is performed if no matches are found. The default is optional; if it is not present, no action takes place if all matches fail. When a match is found, the statements associated with that case are executed until the break is encountered or, in a concluding case or default statement, until the end of the switch is reached.



```cpp
// Program to built a simple calculator using switch Statement
#include <iostream>
using namespace std;
int main()
{
    char o;
    float num1, num2;
    cout << "Enter an operator (+, -, *, /): ";
    cin >> o;
    cout << "Enter two operands: ";
    cin >> num1 >> num2;

    switch (o)
    {
        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! operator is not correct";
            break;
    }

    return 0;
}
```

3.5 THE FOR LOOP:

The general form of the for loop for repeating a single statement is

for(initialization; expression; increment) statement;

For repeating a block, the general form is

for(initialization; expression; increment)
{
statement sequence
}

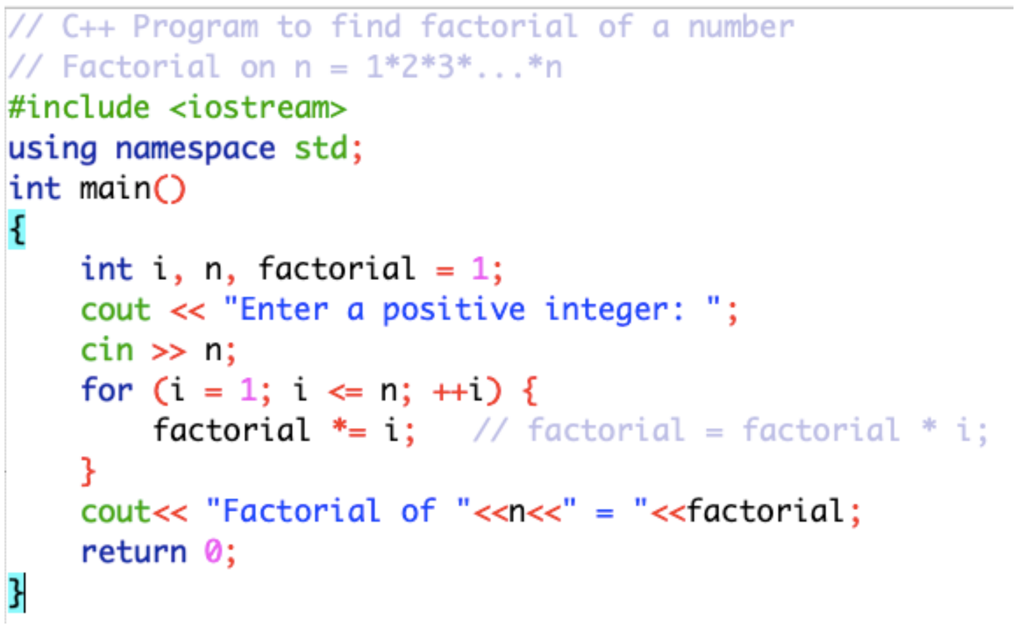
The **initialization** is usually an assignment statement that sets the initial value of the loop control variable, which acts as the counter that controls the loop.

The **expression** is a conditional expression that determines whether the loop will repeat.

The **increment** defines the amount by which the loop control variable will change each time the loop is repeated.

Notice that these three major sections of the loop must be separated by semicolons.

The for loop will continue to execute as long as the conditional expression tests true. Once the condition becomes false, the loop will exit, and program execution will resume on the statement following the for block.



```
// C++ Program to find factorial of a number
// Factorial on n = 1*2*3*...*n
#include <iostream>
using namespace std;
int main()
{
    int i, n, factorial = 1;
    cout << "Enter a positive integer: ";
    cin >> n;
    for (i = 1; i <= n; ++i) {
        factorial *= i;   // factorial = factorial * i;
    }
    cout<< "Factorial of "<<n<<" = "<<factorial;
    return 0;
}
```

3.6 THE WHILE LOOP:

The general form of the while loop is

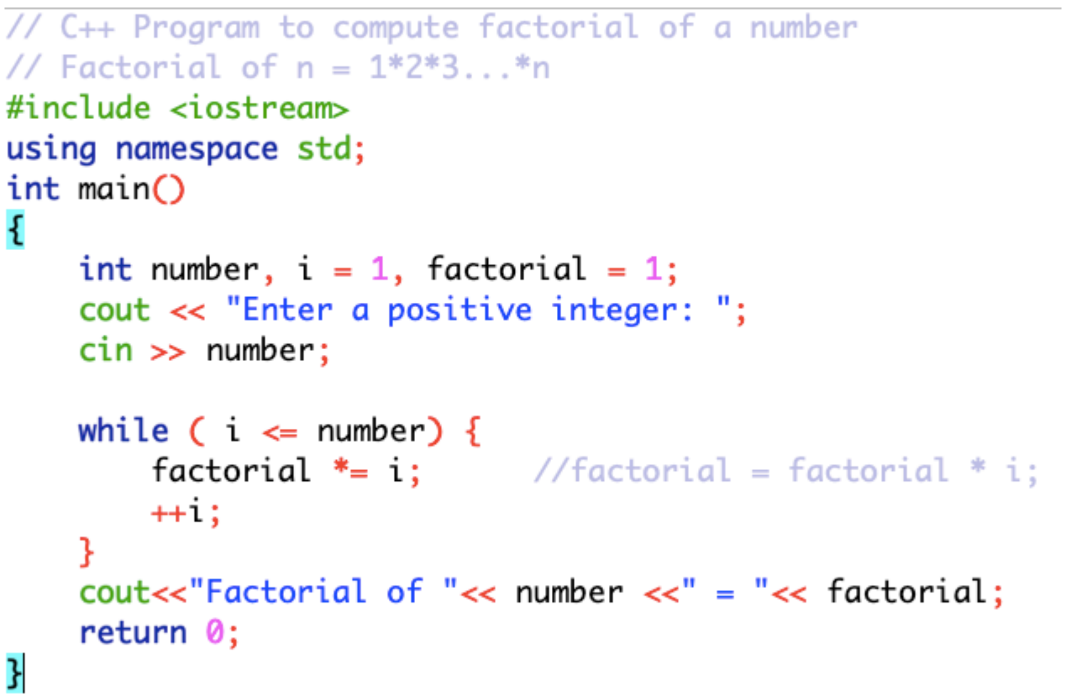
while(expression) statement;

Where **statement** may be a single statement or a block of statements.

The **expression** defines the condition that controls the loop, and it can be any valid expression.

The statement is performed while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

The following program illustrates this characteristic of the while loop. It displays a line of periods. The number of periods displayed is equal to the value entered by the user. The program does not allow lines longer than 80 characters. The test for a permissible number of periods is performed inside the loop's conditional expression, not outside of it.



```
// C++ Program to compute factorial of a number
// Factorial of n = 1*2*3...*n
#include <iostream>
using namespace std;
int main()
{
    int number, i = 1, factorial = 1;
    cout << "Enter a positive integer: ";
    cin >> number;

    while ( i <= number) {
        factorial *= i;      //factorial = factorial * i;
        ++i;
    }
    cout<<"Factorial of "<< number <<" = "<< factorial;
    return 0;
}
```

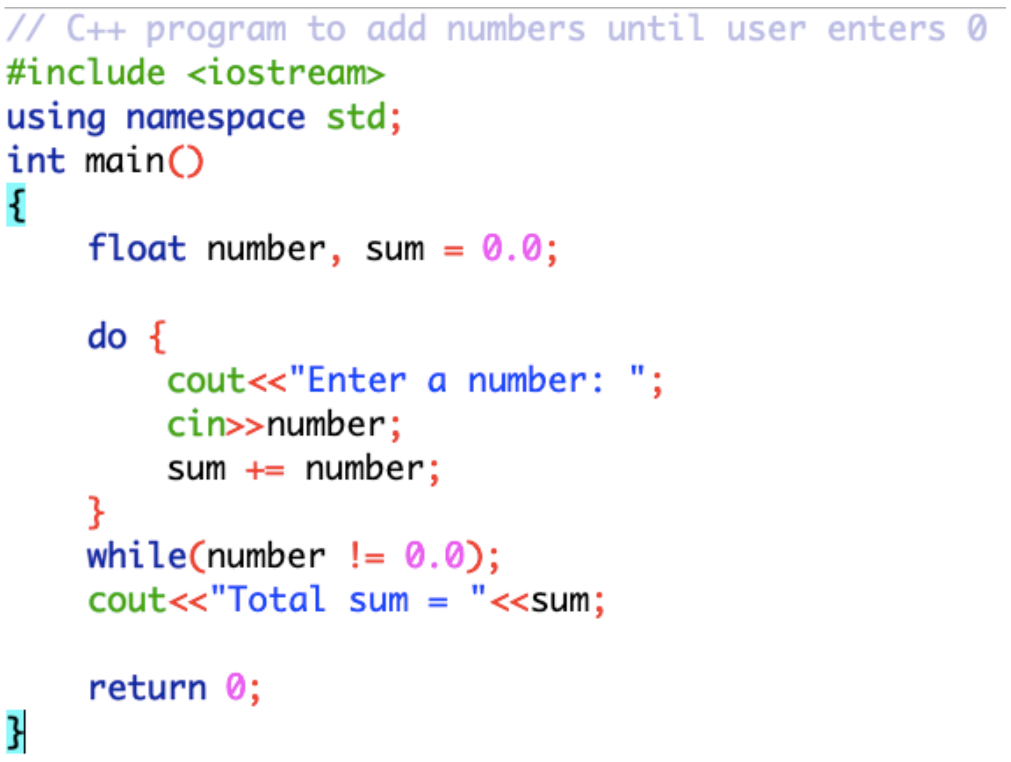
3.7 THE DO-WHILE LOOP:

The last of C++'s loops is the do-while. Unlike the for and the while loops, in which the condition is tested at the top of the loop, the do-while loop checks its condition at the bottom of the loop. This means that a do-while loop will always execute at least once.

The general form of the do-while loop is

```
do
{
        statements;
}
while(condition);
```

Although the braces are not necessary when only one statement is present, they are often used to improve readability of the do-while construct, thus preventing confusion with the while. The do-while loop executes as long as the conditional expression is true.

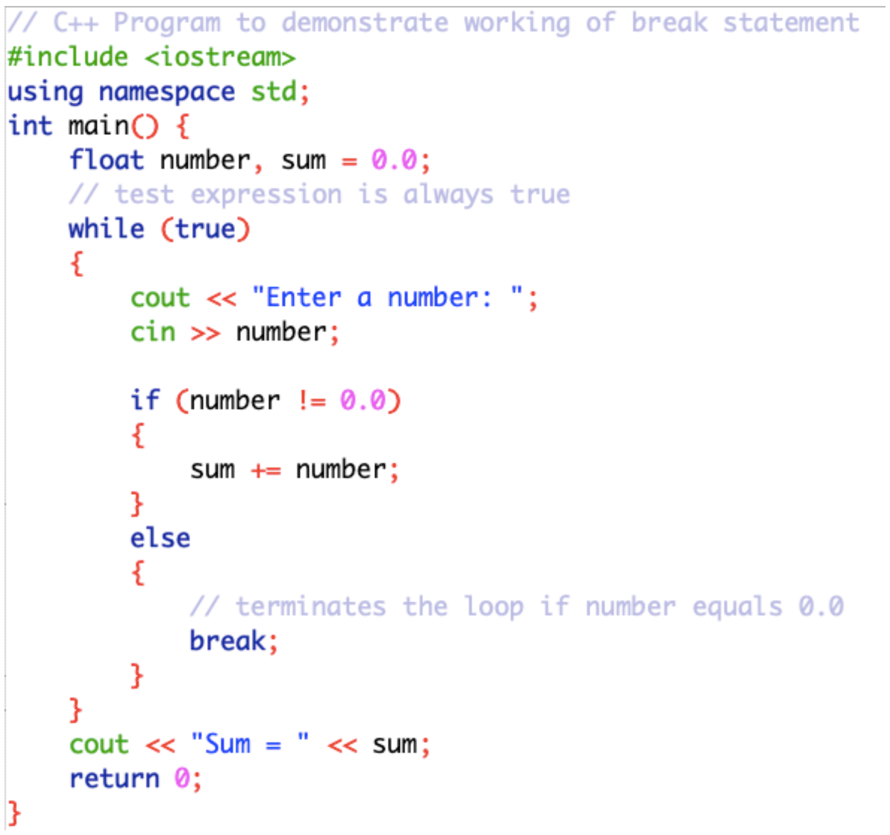


```
// C++ program to add numbers until user enters 0
#include <iostream>
using namespace std;
int main()
{
    float number, sum = 0.0;

    do {
        cout<<"Enter a number: ";
        cin>>number;
        sum += number;
    }
    while(number != 0.0);
    cout<<"Total sum = "<<sum;

    return 0;
}
```

3.8 USING BREAK TO EXIT A LOOP:

It is possible to force an immediate exit from a loop, bypassing the loop's conditional test, by using the break statement.

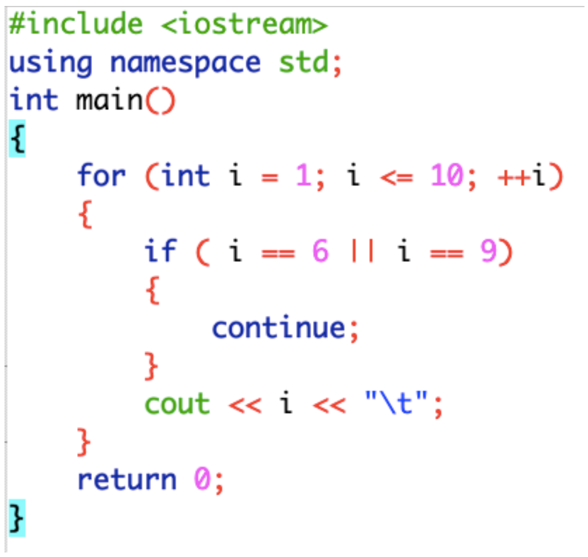
When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop. Here is a simple example



```
// C++ Program to demonstrate working of break statement
#include <iostream>
using namespace std;
int main() {
    float number, sum = 0.0;
    // test expression is always true
    while (true)
    {
        cout << "Enter a number: ";
        cin >> number;

        if (number != 0.0)
        {
            sum += number;
        }
        else
        {
            // terminates the loop if number equals 0.0
            break;
        }
    }
    cout << "Sum = " << sum;
    return 0;
}
```

<u>USING CONTINUE</u>

It is possible to force an early iteration of a loop, bypassing the loop's normal control structure. This is accomplished using **continue**. The continue statement *forces the next iteration of the loop to take place*, skipping any code between itself and the conditional expression that controls the loop.



```
#include <iostream>
using namespace std;
int main()
{
    for (int i = 1; i <= 10; ++i)
    {
        if ( i == 6 || i == 9)
        {
            continue;
        }
        cout << i << "\t";
    }
    return 0;
}
```

3.9 USING THE GOTO STATEMENT:

The goto is C++'s unconditional jump statement. Thus, when encountered, program flow jumps to the location specified by the goto.

There are times when the use of the goto can clarify program flow rather than confuse it. The goto requires a label for operation. A label is a valid C++ identifier followed by a colon. Furthermore, the label must be in the same function as the goto that uses it.

```
// This program calculates the average of numbers entered by user.
// If user enters negative number, it ignores the number and
// calculates the average of number entered before it.
# include <iostream>
using namespace std;
int main()
{
    float num, average, sum = 0.0;
    int i, n;
    cout << "Maximum number of inputs: ";
    cin >> n;
    for(i = 1; i <= n; ++i)
    {
        cout << "Enter n" << i << ": ";
        cin >> num;

        if(num < 0.0)
        {
           // Control of the program move to jump:
            goto jump;
        }
        sum += num;
    }

jump:
    average = sum / (i - 1);
    cout << "\nAverage = " << average;
    return 0;
}
```