Unit

3

LOOPS AND DECISIONS

Lesson Structure

3.0 Objective

After going through this unit you will understand:

- Relational and Logical operators that are used in C++ programs
- Operator Precedence
- Looping Statements and their usage
- Conditional Statements and their usage

3.1 Introduction

Operators are the foundation of any programming language. Thus the functionality of C++ programming language is incomplete without the use of operators. We can define operators as symbols that helps us to perform specific mathematical and logical computations on operands. Loops in programming comes into use when we need to repeatedly execute a block of statements. In general, statements are executed sequentially, i.e., the first statement is executed initially, followed by the second statement and so on. However, if a block of statements is to be executed a number of times, then loops are used. A conditional statement is such a type of statement that is used to make some decision based on the condition given. If the condition is true, then a block of statements is executed. Otherwise another block may be executed.

The rest of the unit is organized as follows. Section 3.2 explains the Relational operators in C++. Sections 3.3 and 3.4 describe the usage of Looping statements and Conditional statements respectively. Section 3.5 discusses the logical operators and Section 3.6 shows the Operator precedence. Section 3.7 gives a brief summary on the unit. Section 3.8 contains some questions for the students to workout. Section 3.9 shows some suggested readings.

3.2 Relational Operators

In C++, the following relational operators are used. Suppose that A variable holds the value 10 and B variable holds the value 20.

Table 3.1: Relational operators in C++

Operator	Description	Example
>=	Checks whether the left operand's value is more than or equal to that of the right operand. If yes, then condition is true.	(A >= B) is false.
==	Checks whether the two operands have the same value or not. If they have same value then condition is true.	(A == B) is false.
<=	Checks whether the left operand's value is less than or equal to that of the right operand. If yes, then condition is true.	(A <= B) is true.

>	Checks whether the left operand's value is more than that of the right operand. If yes, then condition is true.	(A > B) is false.
!=	Checks whether the two operands have the same value or not. If they don't have same value then condition is true.	(A != B) is true.
<	Checks whether the left operand's value is less than that of the right operand. If yes, then condition is true.	(A < B) is true.

3.3 Looping Statements

Generally, statements are executed sequentially, i.e. one after another. However, sometimes a block of statements may have to be executed over and over again till some condition is satisfied. This task is accomplished by the help of looping statements. In C++, we use the following types of loops for this purpose.

Table 3.2: Looping statements in C++

Loop Type	**Description**
for loop	If condition is true, then code block is executed. Next, the statement is modified and the condition is checked again. When condition is false the loop is exited.
while loop	As long as the given condition is true, the target statement is executed. The condition is checked after each execution of the target statement.
do...while loop	It is similar to the while loop with the only difference that the condition is checked at the bottom of the loop. Also the statements in do-while loop will execute at least once for sure.
nested loops	One or more loops can be used inside any other for loop, while loop or do-while loop.

Loop Control Statements

These statements change the normal sequence of the execution. The subsequent control statements are used in C++.

Table 3.3: Loop control statements in C++

Control Statement	Description
continue statement	This statement forces the next iteration of the loop to take place, skipping the execution of the statements that are remaining in the current iteration of the loop.
goto statement	This statement allows the control of the program to be transferred to a given label within the same function.
break statement	This statement is used to terminate the current loop or switch and transfer the program control to the statement following the loop or switch

The Infinite Loop

If the condition never becomes false, then the execution of the loop continues infinitely. Such a loop is called an Infinite Loop. By leaving the conditional expression empty, the loop can be made an infinite loop.

Program: infinite loop

```
#include <iostream>
using namespace std;
int main ()
{
        for( ; ; )
        {
                printf("This loop will run forever.\n");
        }
        return 0;
}
```

For Loop

In C++, for loop is used to execute a block of statements over and over again till the given condition is true. After each execution the condition is checked again to see if it is true or not. If the condition is still true, then the statements are executed. Otherwise the loop is exited.

Syntax:

```
for ( init; condition; increment ) {
        statement1;
        statement2;
        statement3;
        .
        .
        .
        statementN;
        }
```

The flow of control in a for loop has been explained below:

At first the **init** step is executed. It is executed only once. In this step, loop control variables are declared and initialized.

In the next step, the condition is checked to see it is true or false. If the condition is true, then the statements within that loop are executed. If the condition is false, then the statements in the loop are skipped and the control of the program is transferred to the statement immediately following the for loop.

After the execution of all the statements within the for loop, the flow of control is transferred back to the increment statement. This statement modifies the value of the loop control variables.

After executing the increment statement, the condition is checked again. If the condition is true, then the whole process is repeated again (statements within loop → increment statement → condition check). When the condition becomes false, then the for loop is exited.

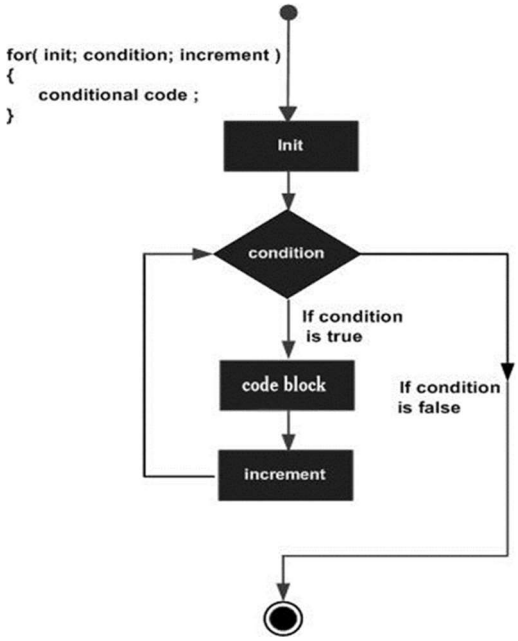

Figure 3.1: Flow Diagram of For Loop

Program: usage of for loop

```
#include <iostream>
using namespace std;
int main ()
{
        // execution of for loop
        for( int m = 3; m < 8; m = m + 1 )
        {
                cout << "value of m: " << m << endl;
        }
        return 0;
}
```

Ouput:

```
value of m: 3
value of m: 4
value of m: 5
value of m: 6
value of m: 7
```

Program: print n consecutive numbers starting from 1 using for Loop

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int m;
        cout<<"Enter the value of m:";
        cin>>m;
        for (int k = 1; k <= m; k++)
        {
                cout<< "Number is:" << k << endl;
        }
        getch();
        return 0;
}
```

Output:

```
Enter the value of m:5
Number is:1
Number is:2
Number is:3
Number is:4
Number is:5
```

Program: find Prime Number Using For Loop

```
#include<iostream>
#include<conio.h>
#include<math.h>
using namespace std;
int main()
{
        int m;
        cout<<"Enter the Number :";
        cin>>m;
        cout<<"List of Prime Numbers Below "<<m<<endl;
        for (int k=2; k<m; k++)
        for (int p=2; p*p<=k; p++)
        {
                if (k % p == 0)
                break;
                else if (p+1 > sqrt(k))
                {
                        cout << k << "\n";
                }
        }
        getch();
        return 0;
}
```

Output:

```
Enter the Number :50
List of Prime Numbers Below 50
5
7
11
13
17
19
23
29
31
37
41
43
47
```

Program: creating table Value Using For Loop

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int table, range;
        cout << "Enter the multiplication table you want to print : ";
        cin >> table;
        cout << "Enter the range: ";
        cin >> range;
        for (int i = 1; i <= range; ++i)
        {
                cout << table << " * " << i << " = " << table * i << endl;
        }
        getch();
        return 0;
}
```

Output:

```
Enter the multiplication table you want to print: 2
Enter the range: 10
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

While Loop

The while loop is a control statement that makes the code block execute again and again till the given condition holds true. As long as the given condition is true, the target statement is executed. The condition is checked after each execution of the target statement.

Syntax:

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

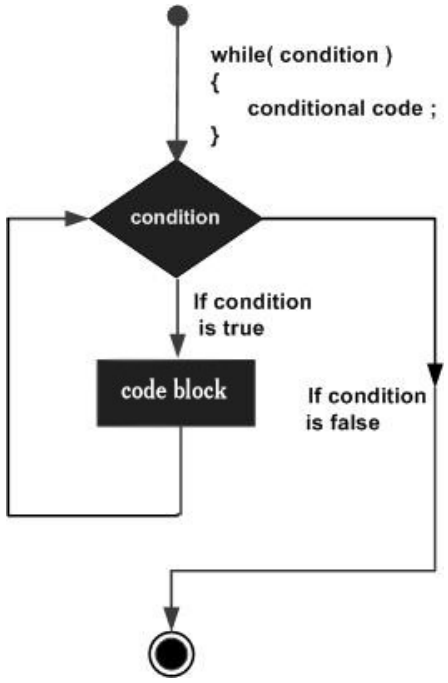

Figure 3.2: Flow Diagram of While Loop

It may so happen that the while loop doesn't execute at all. This happens when the condition holds false during the first test. In such a case the statements within the loop are skipped and the flow control is passed to the statement immediately following the while loop.

Program: print n consecutive numbers starting from 1 using while loop

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        // Variable Declaration
        int n;
        cout<<"Enter the Number :";
        cin>>n;
        int i = 1;
        //while Loop Block
        while (i <= n)
        {
                cout<< i << "\n";
                i++;
        }
        getch();
```

```
        return 0;
}
```

Output:

```
Enter the Number :10
1
2
3
4
5
6
7
8
9
10
```

Program: usage of while loop

```
#include <iostream>
using namespace std;
int main ()
{
        int m = 5;
        while( m < 10 )
        {
                cout << "value of m: " << m << endl;
                m++;
        }
        return 0;
}
```

Output:

```
value of m: 5
value of m: 6
value of m: 7
value of m: 8
value of m: 9
```

Program: generate Fibonacci Series using While Loop

```
#include<iostream>
#include<conio.h>
```

```
using namespace std;
int main()
{
        int counter, n;
        long last=1,next=0,sum;
        cout<<"Enter the Number :";
        cin>>n;
        //Fibonacci Series Calculation
        while(next<n/2)
        {
                cout<<last <<"  ";
                sum=next+last;
                next=last;
                last=sum;
        }
        getch();
        return 0;
}
```

Output:

Enter the Number :300

1 1 2 3 5 8 13 21 34 55 89 144 233

Do While statement

A do-while loop does not test the condition at the top of the loop. In fact, here the condition is tested at the bottom of the loop. Unlike the while loop, where the code may not be executed even once, in the do-while loop the code is executed at least once.

Syntax:

```
do {
  statement1;
  statement2;
  .
  .
  statementN;
}while( condition );
```

The condition for the code is specified after the code itself. The code is executed only if the specified condition holds true. If the condition does not hold true, then the loop is exited as shown in figure 3.3 below.

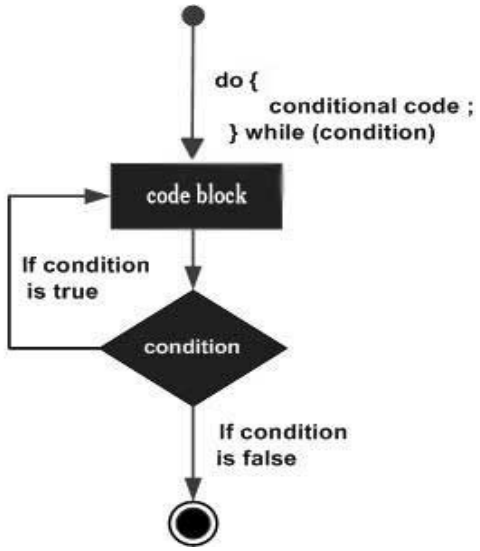

Figure 3.3: Flow Diagram of Do-While Loop

Program: example 1 of do-while loop

```
#include <iostream>
using namespace std;
int main ()
{
        int m = 5;
        do
        {
                cout << "value of m: " << m << endl;
                m = m + 1;
        }
        while( m < 10 );
        return 0;
}
```

Output:

```
value of m: 5
value of m: 6
value of m: 7
value of m: 8
value of m: 9
```

Program: example 2 of do-while loop

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int n;
        cout<<"Enter the Number :";
        cin>>n;
        int i = 1;
        //Do while Loop Block
        do
        {
                cout<<i<< "\n";
                i++;
        }
        while (i <= n);
        // Wait For Output Screen
        getch();
        return 0;
}
```

Output:

```
Enter the Number :5
1
2
3
4
5
```

Nested Loop

A loop can be nested inside of another loop. In C++, at least 256 levels of nesting are allowed.

Syntax for nested for loop statement:

```
for ( init; condition; increment ) {
  for ( init; condition; increment ) {
    statement1;
    statement2;
    ….
    statementN;
  }
```

```
    statement1;
    statement2;
    ….
    statementM;

}
```

Syntax for nested while loop statement:

```
while(condition) {
  while(condition) {
    statement1;
    statement2;
    ….
    statementN;

  }
    statement1;
    statement2;
    ….
    statementM;

}
```

Syntax for nested do-while loop statement:

```
do {
    statement1;
    statement2;
    ….
    statementN;

  do {
    statement1;
    statement2;
    ….
    statementM;

  }while( condition );

}while( condition );
```

Program: nested for loop to find the prime numbers between 10 and 50

```
#include <iostream>
using namespace std;
int main ()
{
        int m, n;
        for(m = 10; m<50; m++)
        {
                for(n = 2; n <= (m/n); n++)
                if(!(m%n)) break;
                if(n > (m/n)) cout << m << " is prime\n";
        }
        return 0;
}
```

Output:

```
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
```

Break Statement

This statement is used to terminate the current loop or switch and transfer the program control to the statement following the loop or switch.

Syntax:

```
break;
```

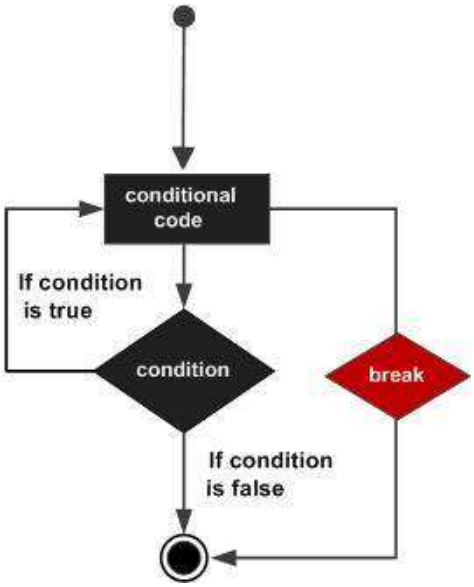

Figure 3.4: Flow Diagram of Break statement

Program: usage of break statement

```
#include <iostream>
using namespace std;
int main ()
{
        int m = 5;
        do
        {
                cout << "value of m: " << m << endl;
                m = m + 1;
                if( m > 10)
                {
                        break;
                }
        }
        while( m < 15 );
        return 0;
}
```

Output:

```
value of m: 5
value of m: 6
value of m: 7
value of m: 8
value of m: 9
value of m: 10
```

Continue Statement

This statement forces the next iteration of the loop to take place, skipping the execution of the statements that are remaining in the current iteration of the loop.

Syntax:

```
continue;
```

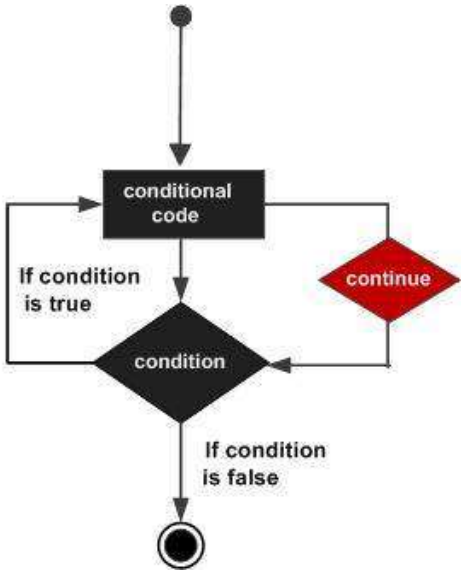

Figure 3.5: Flow Diagram of Continue statement

Program: usage of continue statement

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

int main ()
{
        int m = 5;

        do
        {
                if( m == 10)
                {
                        m = m + 1;
                        continue;
                }

                cout << "value of m: " << m << endl;
                m = m + 1;
```

```
        }
        while( m < 15 );
        return 0;
}
```

Output:

```
value of m: 5
value of m: 6
value of m: 7
value of m: 8
value of m: 9
value of m: 11
value of m: 12
value of m: 13
value of m: 14
```

Goto Statement

This statement allows the control of the program to be transferred to a given label within the same function.

Syntax:

```
goto label;
..
.
label: statement;
```

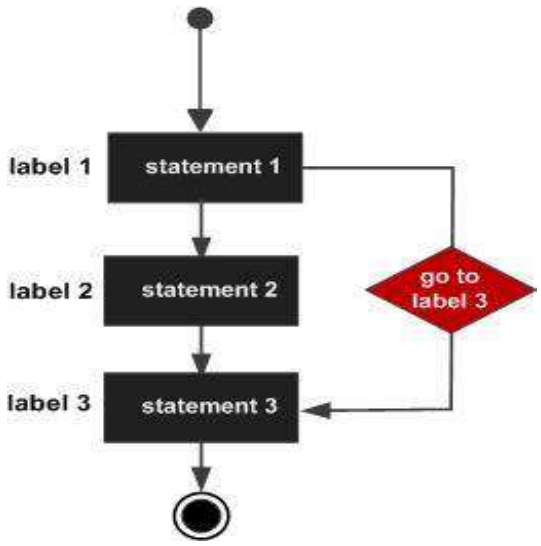

Figure 3.6: Flow Diagram of GoTo Loop

Program: usage of goto loop

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

int main ()
{
        int m = 5;
        LOOP:do
        {
                if( m == 10)
                {
                        m = m + 1;
                        goto LOOP;
                }

                cout << "value of m: " << m << endl;
                m = m + 1;
        }
        while( m < 15 );
        return 0;
}
```

Output:

```
value of m: 5
value of m: 6
value of m: 7
value of m: 8
value of m: 9
value of m: 11
value of m: 12
value of m: 13
value of m: 14
```

3.4 Conditional Statements

Conditional statements are used to control the execution flow in the program. For e.g., based on the condition given, it is decided whether a statement or a block of statements should be executed or not. Followings are conditional statements in C++:

I. if statement
II. if – else statement
III. switch statement

if statement

The *if statement* first checks the condition specified. If the condition holds true, then the concerned statements are executed. Otherwise, those statements are skipped.

Syntax:

```
if(condition)
{
Statement(s)
}
```

Look at the sample code:

```
if (number < 0)
{
   cout << "Error! You must enter a positive number!" << endl;
}
```

In the above code, the condition is checked first. If the variable number is less than 0, then the statement "Error! You must enter a positive number!" is printed on the computer screen. But if the number is not less than 0, then the given statement is skipped and the control is transferred to the statement that immediately follows the closing curly bracket (not shown in the code).

The comparison operators are shown below:

Conditions

A condition is basically an expression that can evaluate to either *true* or *false*. Both true and false are keywords in C++. A simple condition consists of two operands, and one operator that compares those two operands.

Table 3.4: Conditional operators in C++

>	if value of left operand is more than value of right operand, then condition is true
>=	if value of left operand is more than or equal to value of right operand, then condition is true
!=	if value of left operand is not equal to value of right operand, then condition is true
<	if value of left operand is less than value of right operand, then condition is true

==	if value of left operand is equal to value of right operand, then condition is true
<=	if value of left operand is less than or equal to value of right operand, then condition is true

Composite Conditions

We often face situations where simple conditions are not sufficient. For e.g., if we have to check if the given number lies within a range of values say 10 to 20. In such a case, we need to verify two conditions at the same time. We need to verify if the given number is greater than or equal to 10 and if the number is less than or equal to 20. The logical AND operator (&&) is used to combine both the conditions.

Syntax:

```
if (10 <= number && number <= 20)
```

Just as the logical AND operator, the logical OR operator also needs two expressions on either side.

Syntax:

```
if (result == 'p' || result == 'P')
```

The above statement uses a composite condition to check if the result is equal to p in a case-insensitive way.

The NOT operator can also be used with composite conditions as shown in the statement below.

Syntax:

```
if (! (answer == 'q' || answer == 'Q'))
```

Program: show the use of if statement.

```
#include<iostream>
using namespace std;
int main()
{
    int a;
```

```
    cout<<"Enter the Number :";
    cin>>a;
    if(a > 10)      //If Condition to Check Is number is greater than 10
    {
      cout<<a<<" Is Greater than 10";
      cout<<"Yes";
    }
    return 0;
}
```

Output:

```
Enter the Number: 15
15 Is Greater than 10
Yes
```

if-else Statement

Initially the condition specified in the if statement is evaluated. If the condition is true, then the code block enclosed within the opening and closing braces of the if statement is executed. If the condition in the if statement is not true, then the code block within the opening and closing braces of the else statement is executed.

Syntax:

```
if (condition)
{
   block 1 of statement(s)
}
else
{
   block 2 of statement(s)
}
```

If the if statement has multiple decisions, then the if statement is followed by multiple else-if statements as shown in the code below. An else statement may follow the multiple else-if statements. The else statement executes only if all the previous conditions evaluate to false.

Syntax:

```
if (1st condition)
{
```

```
    block 1 of statement(s)
}
else if (2nd condition)
{
    block 2 of statement(s)
}
...
else if (Nth condition)
{
    block N of statement(s)
}
else
{
    Optional block of statement(s)
}
```

Out of all the blocks, only one block of statements is executed. After the execution of that block, the control is transferred to the statement following the else block, or the statement following the last else-if block, provided there is no else block.

Suppose that the examination department of a college wants to convert the marks obtained by students into grade according to the following rule:

90 to 100	→	Grade A
80 to 89	→	Grade B
70 to 79	→	Grade C
60 to 69	→	Grade D
Less than 60	→	Fail

The following code can be used to implement the above rule.

```
if (marks >= 90)
{
    grade = 'A';
}
else if (marks >= 80)
```

```
{
   grade = 'B';
}
else if (marks >= 70)
{
   grade = 'C';
}
else if (marks >= 60)
{
   grade = 'D';
}
else
{
   grade = 'Fail';
}
```

Program: show the use of if-else statement in C++.

```
#include<iostream>
using namespace std;
int main()
{
   int a;
   cout<<"Enter the Number :";
   cin>>a;
   if(a > 10)
   {
      cout<<a<<" Is Greater than 10"; //If Condition to Check Is number is greater than 10
    }
   else
   {
        cout<<a<<" Is Less than/Equal 10"; // else block for Condition Fail
   }
   return 0;
}
```

Output 1:

Enter the Number :20
20 Is Greater than 10

Output 2:

Enter the Number :5

5 Is Less than/Equal 10

The switch Statement

The switch statement allows us to test a variable against multiple values, where each value is called a case and the variable is tested for each case.

A basic illustration of a switch statement in real life is how to handle menu selection. For e.g., the user is prompted by the program to select one option out of many. The options can be numbered 1 through N. The user enters a value and depending on that value the program executes a block of code among the block of codes available.

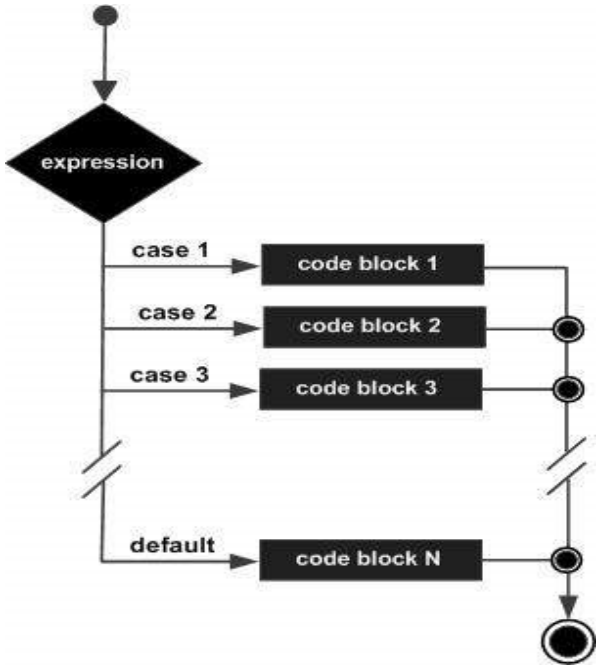

Figure 3.7: Flow Diagram of Switch statement

The code implementing the switch case for the above scenario is shown below.

```
int option;
cin >> option;
switch (option)
{
        case 1:
            cout << "First item selected!" << endl;
            break;
        case 2:
            cout << "Second item selected!" << endl;
            break;
```

```
        case 3:
            cout << "Third item selected!" << endl;
            break;
        default:
            cout << "Invalid selection!" << endl;
            break;
}
```

The break statement is used because it ends the switch statement. If the break statement is not used, then even after the execution of the correct case the execution continues for all the following cases. If the value entered by the user does not match any case given under the switch statement, then the block of code under the default case is executed.

Program: example 1 of switch statement

```
#include <iostream>
using namespace std;
int main ()
{
        char grade = 'B';
        switch(grade)
        {
            case 'A' :
                cout << "Excellent" << endl;
                break;
            case 'B' :
                cout << "Good" << endl;
                break;
            case 'C' :
                cout << "You passed" << endl;
                break;
            case 'D' :
                cout << "You failed" << endl;
                break;
            default :
                cout << "Invalid grade" << endl;
        }
        cout << "Your grade is " << grade << endl;
        return 0;
}
```

Output:

```
Good
Your grade is B
```

Program: example 2 of switch statement

```
#include <iostream>
using namespace std;
int main()
{
	int k=3;
	switch(k)
	{
		case 1:
			cout<<"1st Case "<<endl;
			break;
		case 2:
			cout<<"2nd Case "<<endl;
			break;
		case 3:
			cout<<"3rd Case "<<endl;
			break;
		case 4:
			cout<<"4th Case "<<endl;
			break;
		default:
			cout<<"Default Case"<<endl;
	}
	return 0;
}
```

Output: 3rd Case

3.5 Logical Operators

C++ supports the logical operators given in Table 3.5 below. Suppose that X variable holds the value 1 and Y variable holds the value 0.

Table 3.5: Logical operators in C++

Operator	Description	Example
!	It is called the Logical NOT Operator. Using this the logical state of the operand can be reversed. For e.g., if a condition is false, then the Logical NOT operator makes it true.	!(X && Y) is true.
||	It is called the Logical OR Operator. The condition holds true if any one of the two operands given is non-zero.	(X || Y) is true.
&&	It is called the Logical AND operator. The condition holds true if both the operands given are non-zero.	(X && Y) is false.

3.6 Operator Precedence

In C++, certain operators are given more precedence over the others. This thing affects the evaluation of expressions in C++. For e.g., the addition operator is given lower precedence as compared to the multiplication operator.

Consider the expression a = 5 + 2 * 4. Here the final value that is allotted to a is 13 and not 28. This is because * is given more preference than + in C++. So, the multiplication is performed prior to the addition operation.

Table 3.6 below lists the operators in the decreasing order of their precedence along the rows. Operators with higher precedence are located above the operators with lower precedence in the table.

Table 3.6: Operator precedence in C++

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right

Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

3.7 Summary

Operators are symbols that are used along with operands to perform mathematical and logical computations. Relational operators are used to compare the values of two operands with each other and execute and block of statements if the condition is true or false. The relational operators that are used in C++ are ==, !=, >, <. >= and <=. Logical operators allow us to logically combine Boolean variables (that is, variables of type bool, with true or false values). The various logical operators in C++ are &&, ||, and !. In C++, certain operators are given more precedence over the others. This thing affects the evaluation of expressions in C++. For e.g., the addition operator is given lower precedence as compared to the multiplication operator. Looping statements are used to execute a block of statements repeatedly for a number of times. Usually statements are executed sequentially, but with the help of loops, the statements in any block can be executed over and over again. The various looping statements in C++ are while loop, for loop, do-while loop and nested loop. Conditional statements are used to control the execution flow in the program. For e.g., based on the condition given, it is decided whether a statement or a block of statements should be executed or not.

3.8 Questions

1. Write a C++ program to find out factorial program in C++ using for Loop.

2. Write a C++ program to find out 10 prime numbers starting from 1 using while loop.

3. Describe the Relational operators in C++. Give suitable examples in each case.

4. What is the usage of the Break statement. Give an example.

5. What do you mean by Composite Conditions? Write a program to show its implementation.

6. What is a Switch statement? Write down the rules that apply to a Switch statement.

7. Write a program in C++ that takes as input an age, and prints on the screen whether the user is a child, a teen, an adult or an old person.

8. Explain in detail the Logical operators in C++.

9. Write a program in C++ to show the usage of the IF-ELSE statement.

10. What is the Infinite Loop? Draw a flowchart to explain it.

3.9 Suggested Readings

1. Lafore, Robert. Object-oriented programming in C++. Pearson Education, 1997.

2. https://www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm