Learning Objectives

After learning this chapter, the students will be able to

- Understand the different kinds of statements.
- Construct different flow of control statements in C++.

10.1 Introduction

In the previous chapters you learnt the basic concepts of C++ programming such as variables, constants, operators, data types etc. Generally a program executes its statements sequentially from beginning to end. However, such a strict sequential ordering is restrictive and less useful. There are lot of situations where it is useful to decide the code block executed on the basis of a certain condition. In such situations, the flow of control jumps from one part of the code to another segment of code. Program statements that cause such jumps are called as **"Control flow"**. This chapter deals with the basics of control structures such as "Selection", "Iteration" and "Jump" statement.

10.2 Statements

A computer program is a set of statements or instructions to perform a specific task. These statements are intended to perform specific action. The action may be of variable declarations, expression evaluations, assignment operations, decision making, looping and so on.

Flow of Control

There are two kinds of statements used in C++.

(i) Null statement

(ii) Compound statement

10.2.1 Null statement

The "**null** or **empty statement**" is a statement containing only a semicolon. It takes the flowing form:

```
; // it is a null statement
```

Null statements are commonly used as placeholders in iteration statements or as statements on which to place labels at the end of compound statements or functions.

10.2.2 Compound (Block) statement

C++ allows a group of statements enclosed by pair of braces {}. This group of statements is called as a compound statement or a block.

The general format of compound statement is:

```
{
        statement1;
        statement2;
        statement3;
}
```

For example

```
{
        int x, y;
        x = 10;
        y = x + 10;
}
```

The compound statement or block is a treated as a single unit and may appear anywhere in the program.

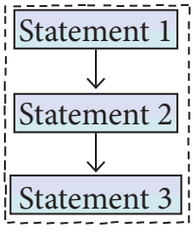
10.3 Control Statements

Control statements are statements that alter the sequence of flow of instructions.

In a program, statements may be executed sequentially, selectively or iteratively. Every programming languages provide statements to support sequence, selection (branching) and iteration.

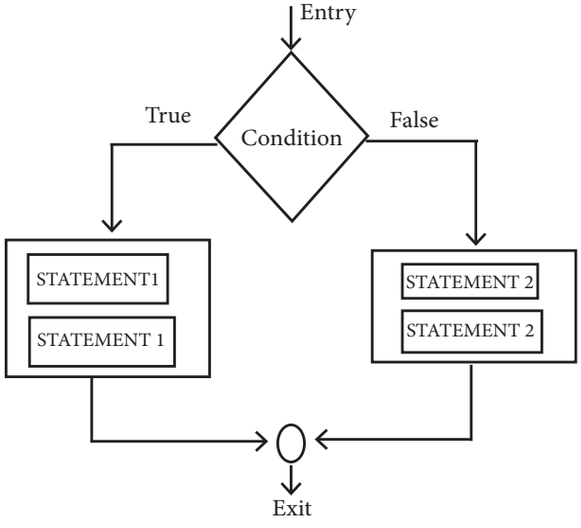
If the statements are executed sequentially, the flow is called as sequential flow. In some situations, if the statements alter the flow of execution like branching, iteration, jumping and function calls, this flow is called as control flow.

Sequence statement



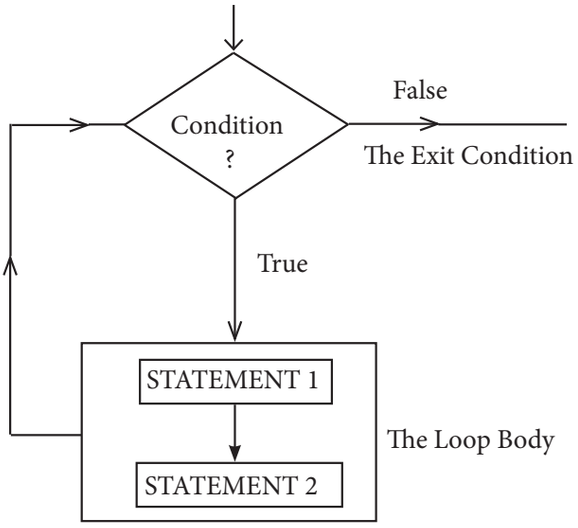
The **sequential statement** are the statements, that are executed one after another only once from top to bottom. These statements do not alter the flow of execution. These statements are called as sequential flow statements. They always end with a semicolon (;).

Selection statement



The selection statement means the statement (s) executed depend upon a condition. If a condition is true, a true block (a set of statements) is executed otherwise a false block is executed. This statement is also called **decision statement** or **selection statement** because it helps in making decision about which set of statements are to be executed.

Iteration statement



The **iteration statement** is a set of statement that are repetitively executed based upon a conditions. If a condition evaluates to true, the set of statements (true block) is executed again and again. As soon as the condition becomes false, the repetition stops. This is also known as **looping statement** or iteration statement.

The set of statements that are executed again and again is called the **body of the loop.**The condition on which the execution or exit from the loop is called **exit-condition** or **test-condition.**

Generally, all the programming languages support this type of statements to write programs depending upon the problem. C++ also supports this type of statements. These statements will be discussed in coming sections.

Note

In C++, any **non zero is treated as true including negative numbers and zero is treated as false.**

Selection statements and iteration statements are executed depending upon the conditional expression. The conditional expression evaluates either true or false.

10.4 Selection statements

In a program a decision causes a one time jump to a different part of a program. Decisions in C++ are made in several ways, most importantly with if .. else … statement which chooses between two alternatives. Another decision statement, switch creates branches for multiple alternatives sections of code, depending on the value of a single variable.

10.4.1 if statement

The if statement evaluates a condition, if the condition is true then a true-block (a statement or set of statements) is executed, otherwise the true-block is skipped.The general syntax of the if statement is:

```
if (expression)
        true-block;
statement-x;
```

In the above syntax, **if** is a keyword that should contain expression or condition which is enclosed within parentheses. If the expression is true (nonzero) then the true-block is executed and followed by statement-x are also executed, otherwise, the control passes to statement-x. The true-block may consists of a single statement, a compound statement or empty statement. The control flow of **if** statement and the corresponding flow chart is shown below.

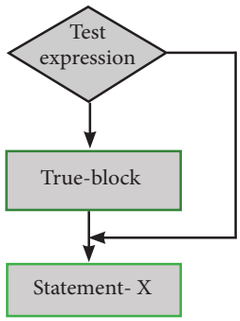


Illustration 10.1 C++ program to check whether a person is eligible to vote using if statement

```
#include <iostream>
using namespace std;
int main()
{
        int age;
        cout<< "\n Enter your age: ";
        cin>> age;
        if(age>=18)
                cout<< "\n You are eligible for voting ....";
        cout<< "This statement is always executed.";
        return 0;
}
```

The pair of braces is not required because if condition followed by only one statement

Output

```
Enter your age: 23
You are eligible for voting….
This statement is always executed.
```

10.4.2 if-else statement

In the above examples of **if**, you have seen that, a block of statements are excecuted only if the condition evaluates to true. What if there is another course of action to be followed if the condition evaluates to false. There is another form of **if** that allows for this kind of either or condition by providing an else clause. The syntax of the if-else statement is given below:

```
if ( expression)
{
        True-block;
}
else
{
        False-block;
}
Statement-x
```

In if-else statement, first the expression or condition is evaluated to either true of false. If the result is true, then the statements inside true-block is executed and false-block is skipped. If the result is false, then the statement inside the false-block is executed i.e., the true-block is skipped.

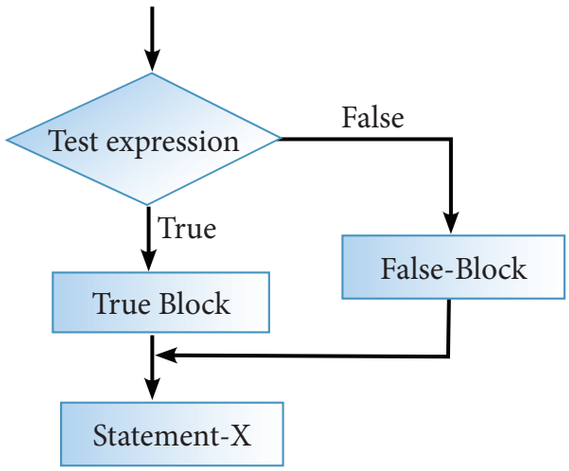


Illustration 10.2 C++ program to find whether the given number is even number or odd number using if-else statement

```
#include <iostream>
using namespace std;
int main()
{
        int num, rem;
        cout<< "\n Enter a number: ";
        cin>>num;
        rem = num % 2;
        if (rem==0)
                cout<< "\n The given number" <<num<< " is Even";
        else
                cout<< "\n The given number "<<num<< " is Odd";
        return 0;
}
```

Output

```
Enter number: 10
The given number 10 is Even
```

In the above program, the remainder of the given number is stored in rem. If the value of rem is zero, the given number is inferred as an even number otherwise, it is inferred as on odd number.

10.4.3 Nested if

An if statement which contains another if statement is called nested if. The nested can have one of the following three forms.

1. If nested inside if part
2. If nested inside else part
3. If nested inside both if part and else part

The syntax of the nested if:

If nested inside if part

```
if (expression-1)
{
    if (expression-2)
    {
     True_Part_Statements;
    }
    else
    {
     False_Part_Statements;
    }
}
else
     body of else part;
```

If nested inside else part

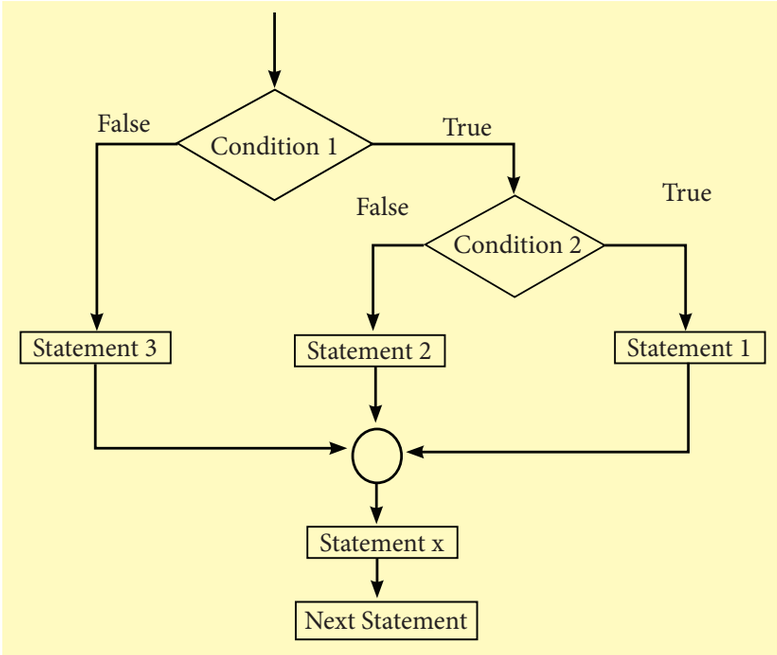
```
if (expression-1)
{
     body of true part;
}
else
{
     if (expression-2)
     {
      True_Part_Statements;
     }
     else
     {
      False_Part_Statements;
     }
}
```

If nested inside both if part and else part

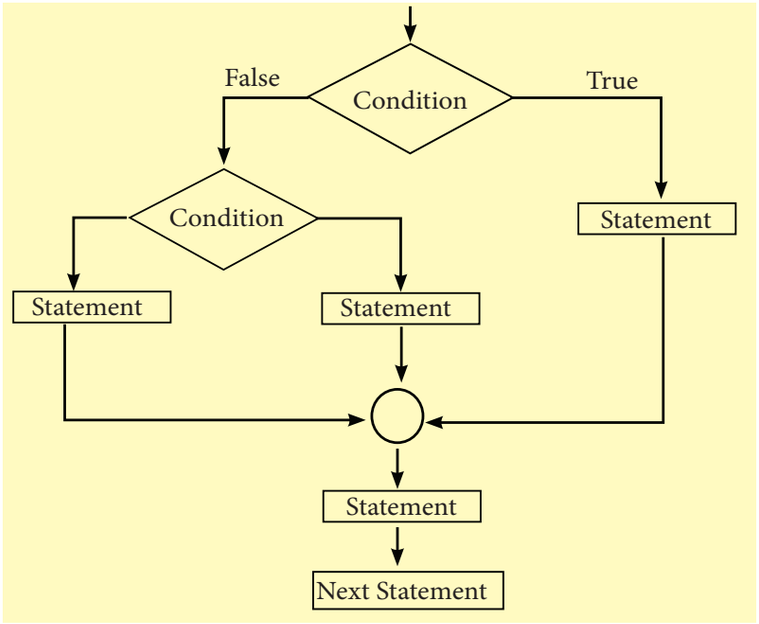
```
if (expression)
{
     if (expression)
     {
      True_Part_Statements;
     }
     else
     {
      False_Part_Statements;
     }
}
else
{
     if (expression)
     {
      True_Part_Statements;
     }
     else
     {
      False_Part_Statements;
     }
}
```

In the first syntax of the nested if mentioned above the expression-1 is evaluated and the expression result is false then control passes to statement-m. Otherwise, expression-2 is evaluated,if the condition is true, then Nested-True-block is executed, next statement-n is also executed. Otherwise Nested-False-Block, statement-n and statement-m are executed.

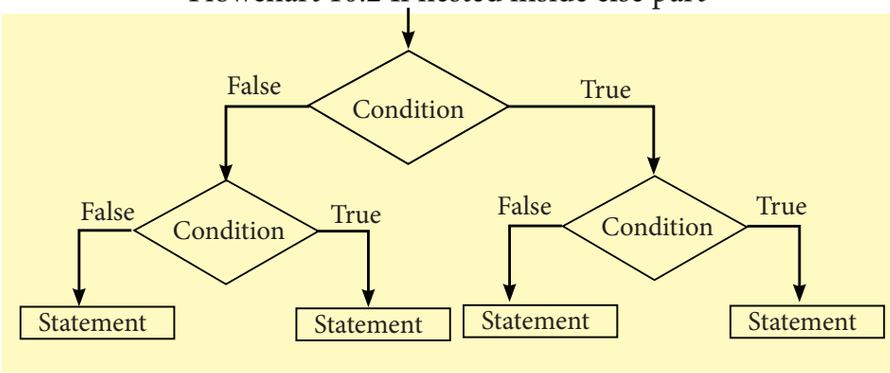
The working procedure of the above said if..else structures are given as flowchart below:



Flowchart 10.1 if nested inside if Part



Flowchart 10.2 If nested inside else part



Flowchart 10.3 If nested inside both if part and else part

Illustration 10.3 – C++ program to calculate commission according to grade using if..else statement

```
#include <iostream>
using namespace std;
int main()
{
        int sales, commission;
        char grade;
        cout << "\n Enter Sales amount: ";
        cin >> sales;
        cout << "\n Enter Grade: ";
        cin >> grade;
        if (sales > 5000)
        {
                commission = sales * 0.10;
                cout << "\n Commission: " << commission;
        }
```

```
        else
        {
                commission = sales * 0.05;
                cout << "\n Commission: " << commission;
        }
        cout << "\n Good Job ..... ";
        return 0;
}
```

Output:

```
Enter Sales amount: 6000
Enter Grade: A
Commission: 600
Good Job .....
```

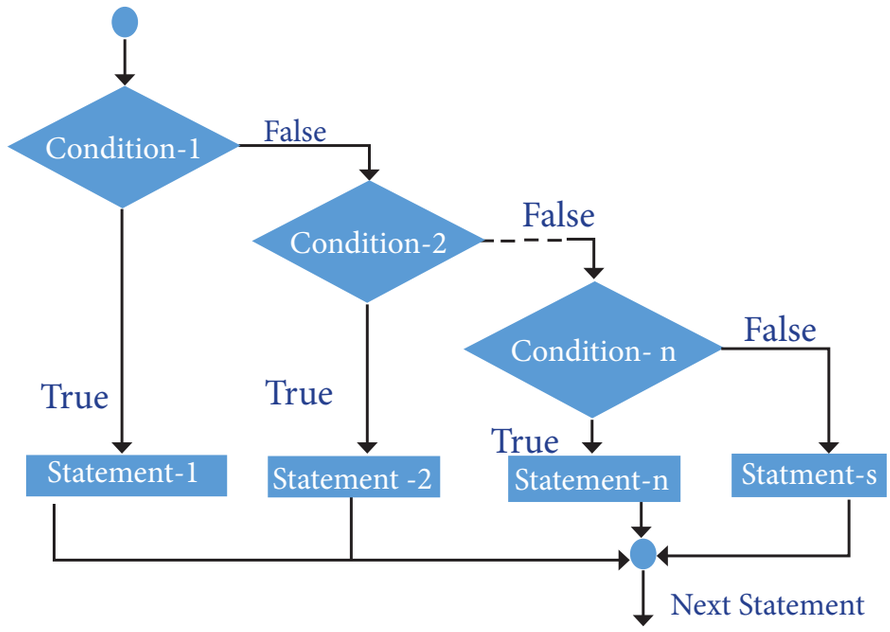
10.4.4 if -else-if ladder

The if-else ladder is a multi-path decision making statement. In this type of statement '**if**' is followed by one or more **else if** statements and finally end with an **else** statement.

The syntax of if-else ladder:

```
if (expression 1)
{
  Statement-1
}
else
        if( expression 2)
        {
                Statement-2
        }
        else
                if ( expression 3)
                {
                        Statement-3
                }
                else
                {
                        Statement-4
                }
```

When the respective expression becomes true, the statement associated with block is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



Flowchart 10.4 if-else ladder flow chart

Illustration 10.4 C++ program to find your grade using if-else ladder.

```
#include <iostream>
using namespace std;
int main ()
{
int marks;
cout<<" Enter the Marks  :";
cin>>marks;
if( marks >= 60 )
        cout<< "Your grade is 1st class !!" <<endl;
                else if( marks >= 50 && marks < 60)
                        cout<< "your grade is 2nd class !!" <<endl;
                                else if( marks >= 40 && marks < 50)
                                        cout<< "your grade is 3rd class !!" <<endl;
else
        cout<< "You are fail !!" <<endl;
return 0;
}
```

Output

```
Enter the Marks  :60
Your grade is 1st class !!
```

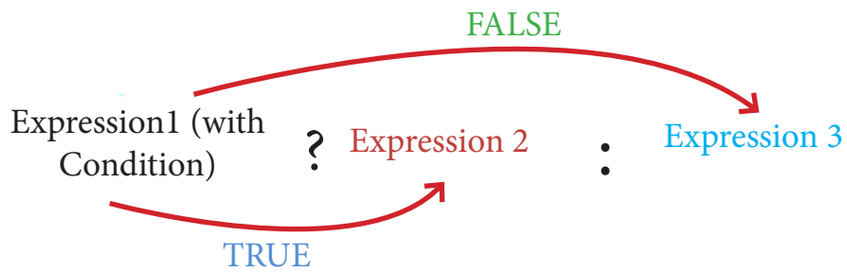
When the marks are greater than or equal to 60, the message **"Your grade is 1st class !!"** is displayed and the rest of the ladder is bypassed. When the marks are between 50 and 59, the message **"Your grade is 2nd class !!"** is displayed, and the other ladder is bypassed. When the mark between 40 to 49, the message **"Your grade is 3rd class !!"** is displayed, otherwise, the message "You are fail !!" is displayed.

10.4.5 The ?: Alternative to if- else

The conditional operator (or Ternary operator) is an alternative for 'if else statement'. The conditional operator that consists of two symbols (?:). It takes three arguments. The control flow of conditional operator is shown below:

The syntax of the conditional operator is:

```
expression 1? expression 2 : expression 3
```



In the above syntax, the expression 1 is a condition which is evaluated, if the condition is true (Non-zero), then the control is transferred to expression 2, otherwise, the control passes to expression 3.

Illustration 10.5 – C++ program to find greatest of two numbers using conditional operator

```
#include <iostream>
using namespace std;
int main()
{
        int a, b, largest;
        cout << "\n Enter any two numbers: ";
        cin >> a >> b;
         largest = (a>b)? a : b;
        cout << "\n Largest number : " << largest;
        return 0;
}
```

Output:

```
Enter any two numbers: 12 98
Largest number : 98
```

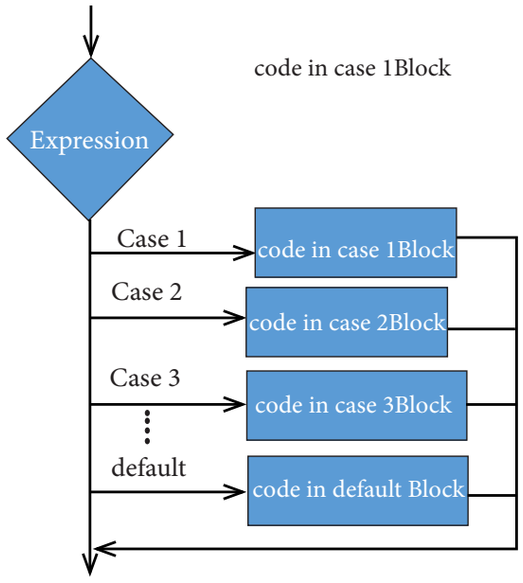
10.4.6 Switch statement

The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. The switch statement replaces multiple if-else sequence.

The syntax of the switch statement is;

```
switch(expression)
{
        case constant 1:
                statement(s);
                break;
        case constant 2:
                statement(s);
                break;
        .
        .
        .
        .
        default:
                statement(s);
}
```

In the above syntax, the expression is evaluated and if its value matches against the constant value specified in one of the case statements, that respective set of statements are executed. Otherwise, the statements under the default option are executed. The workflow of switch statement and flow chart are shown below.



Flowchart10.5: workflow of switch and flow chart

Rules:

1. The expression provided in the switch should result in a constant value otherwise it would not be valid.
2. Duplicate case values are not allowed.
3. The default statement is optional.
4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

6. Nesting of switch statements is also allowed.

Illustration 10.6 – C++ program to demonstrate switch statement

```
#include <iostream>
using namespace std;
int main()
{
        int num;
        cout << "\n Enter week day number: ";
        cin >> num;
        switch (num)
        {
                case 1 : cout << "\n Sunday"; break;
                case 2 : cout << "\n Monday"; break;
                case 3 : cout << "\n Tuesday"; break;
                case 4 : cout << "\n Wednessday"; break;
                case 5 : cout << "\n Thursday"; break;
                case 6 : cout << "\n Friday"; break;
                case 7 : cout << "\n Saturday"; break;
                default: cout << "\n Wrong input....";
        }
}
```

Output:

```
Enter week day number: 6
Friday
```

10.4.7 Switch vs if-else

"if-else" and "switch" both are selection statements. The selection statements, transfer the flow of the program to the particular block of statements based upon whether the condition is "true" or "false". However, there are some differences in their operations. These are given below:

Key Differences Between if-else and switch

S.No	if-else	Switch
1	Expression inside if statement decide whether to execute the if block or under else block.	expression inside switch statement decide which case to execute.
2	An if-else statement uses multiple statements for multiple choices	switch statement uses single expression for multiple choices.
3	If-else statement checks for equality as well as for logical expression.	switch checks only for equality.
4	The **if** statement evaluates integer, character, pointer or floating-point type or Boolean type.	switch statement evaluates only character or a integer data type.
5	If the condition is false the else block statements will be executed	If the condition is false then the default statements are executed.

The if statement is more flexible than switch statement.

10.5 Iteration statements

An iteration (or looping) is a sequence of one or more statements that are repeatedly executed until a condition is satisfied. These statements are also called as control flow statements. It is used to reduce the length of code, to reduce time, to execute program and takes less memory space. C++ supports three types of iteration statements;

- for statement
- while statement
- do-while statement

All looping statements repeat a set statements as long as a specified condition is remains true. The specified condition is referred as a loop control. For all three loop statements, a true condition is any nonzero value and a zero value shows a false condition.

10.5.1 Parts of a loop

Every loop has four elements that are used for different purposes. These elements are

- Initialization expression
- Test expression
- Update expression
- The body of the loop

Initialization expression(s): The control variable(s) must be initialized before the control enters into loop. The initialization of the control variable takes place under the initialization expressions. The initialization expression is executed only once in the beginning of the loop.

Test Expression: The test expression is an expression or condition whose value decides whether the loop-body will be executed or not. If the expression evaluates to true (i.e., 1), the body of the loop gets executed, otherwise the loop is terminated.

In an entry-controlled loop, the test-expression is evaluated before the entering into a loop whereas in an exit-controlled loop, the test-expression is evaluated before exit from the loop.

Update expression: It is used to change the value of the loop variable. This statement is executed at the end of the loop after the body of the loop is executed.

The body of the loop: A statement or set of statements forms a body of the loop that are executed repetitively. In an entry-controlled loop, first the test-expression is evaluated and if it is nonzero, the body of the loop is executed otherwise the loop is terminated. In an exit-controlled loop, the body of the loop is executed first then the test-expression is evaluated. If the test-expression is true the body of the loop is repeated otherwise loop is terminated

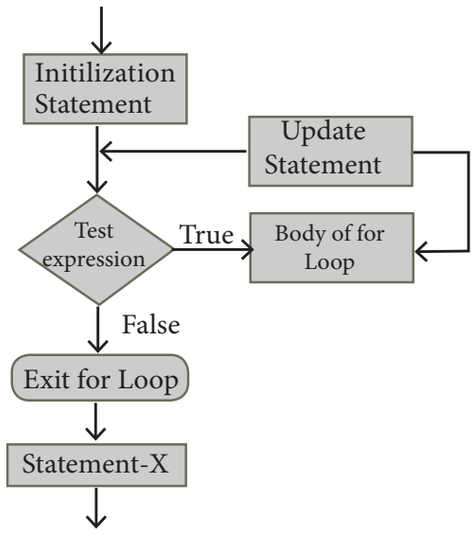
10.5.2 for loop

The for loop is a entry- controlled loop and is the easiest looping statement which allows code to be executed repeatedly. It contains three different statements (initialization, condition or test-expression and update expression(s)) separated by semicolons.

The general syntax is:

```
for (initialization(s); test-expression; update expression(s))
{
        Statement 1;
        Statement 2;
        ..............
}
Statement-x;
```

The initialization part is used to initialize variables or declare variable which are executed only once, then the control passes to test-expression. After evaluation of test-expression, if the result is false, the control transferred to statement-x. If the result is true, the body of the for loop is executed, next the control is transferred to update expression. After evaluation of update expression part, the control is transferred to the test-expression part. Next the steps 3 to 5 is repeated. The workflow of for loop and flow chart are shown below.



Flowchart 10.6: Workflow of for loop and flow chart

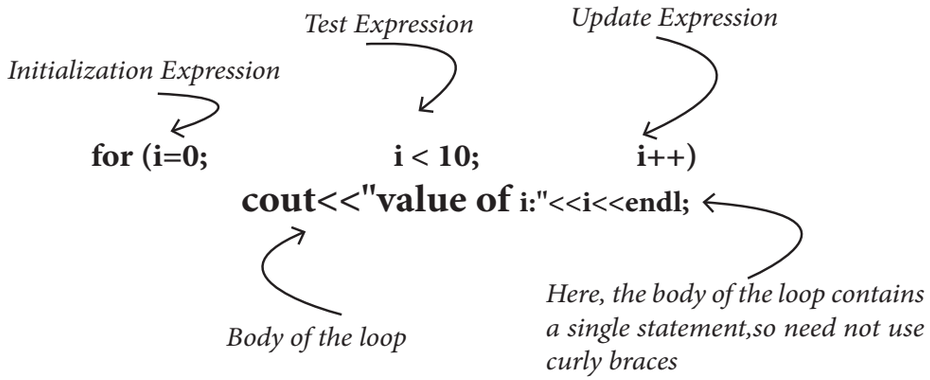
Illustration 10.7 C++ program to display numbers from 0 to 9 using for loop

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

Output

```
value of i : 0
value of i : 1
value of i : 2
value of i : 3
value of i : 4
value of i : 5
value of i : 6
value of i : 7
value of i : 8
value of i : 9
```

The following lines describes the working of the above given for loop:



In the above program, first the variable i is initialized, next i is compared with 10, if i is less than ten, the value of i is incremented. In this way, the numbers 0 to 9 are displayed. Once i becomes 10, it is no longer < 10. So, the control comes out of the for loop.

Illustration 10.8 C++ program to sum the numbers from 1 to 10 using for loop

```
#include <iostream>
using namespace std;
int main ()
{
int i,sum=0;
for(i=1; i<=10;i++)
 {
        sum=sum+i;
 }
cout<<"The sum of 1 to 10 is "<<sum;
return 0;
}
```

Output

The sum of 1 to 10 is 55

Variations of for loop

The **for** is one of the most important looping statement in C++ because it allows a several variations. These variations increase the flexibility and applicability of **for** loop. These variations will be discussed below:

Multiple initialization and multiple update expressions

Multiple statements can be used in the initialization and update expressions of **for** loop. These multiple initialization and multiple update expressions are separated by commas. For example,

```
#include<iostream>
using namespace std;
int main()
{
        int i, j;
            for(i=0, j=10 ;  i<j  ;  i++,j--)
            {
            cout<<"\nThe value of i is"<<i<<" The value of j is "<<j;
```

Multiple initialization expressions (separated by commas)

Multiple update expressions (separated by commas)

```
}
return 0;
}
```

Output

```
The value of i is 0  The value of j is 10
The value of i is 1  The value of j is 9
The value of i is 2  The value of j is 8
The value of i is 3  The value of j is 7
The value of i is 4  The value of j is 6
```

In the above example, the initialization part contains two variables i and j and update expression contains i++ and j++. These two variables are separated by commas which is executed in sequential order i.e., during initialization firstly i=0 followed by j=10. Similarly, in update expression, firstly i++ is evaluated followed by j++ is evaluated.

Prefer prefix operator over postfix

Generally, the update expression contains increment/decrement operator (++ or --). In this part, always prefer prefix increment/decrement operator over postfix when to be used alone. The reason behind this is that when used alone, prefix operators are executed faster than postfix.

Optional expressions

Generally, the for loop contains three parts, i.e., initialization expressions, test expressions and update expressions. These three expressions are optional in a for loop.

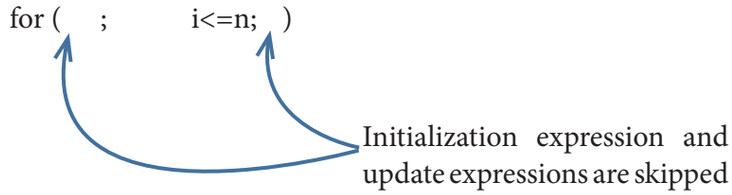
Illustration 10.9 C++ program to sum the numbers from 1 to n

```
#include <iostream>
using namespace std;
int main ()
{
int i, sum=0, n;
cout<<"\n Enter The value of n";
cin>>n;
i =1;
for (  ; i<=n;  )
 {
        sum += i;
        ++i;
 }
cout<<"\n The sum of 1 to " <<n<<"is "<<sum;
return 0;
}
```

Output

```
Enter the value of n 5
The sum of 1 to 5 is 15
```

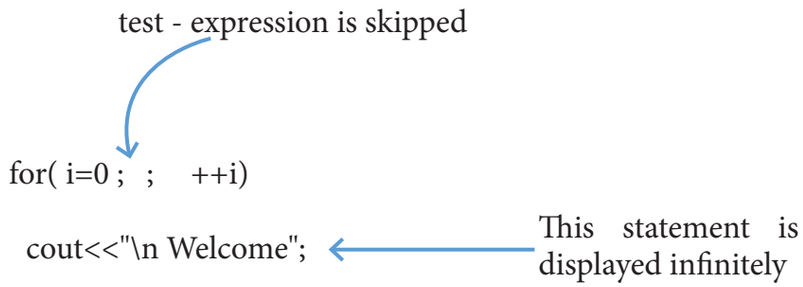
In the above code, the update expression is not given, but a semicolon is necessary before the update expression.



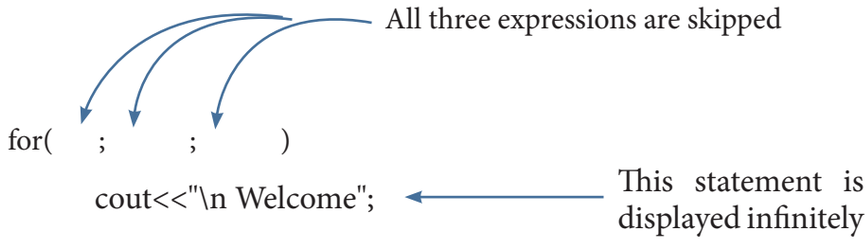
In the above code, neither the initialization nor the update expression is given in the for loop. If both or any one of expressions are absent then the control is transferred to conditional part.

infinite loop

An infinite loop will be formed if a test-expression is absent in a for loop. For example,

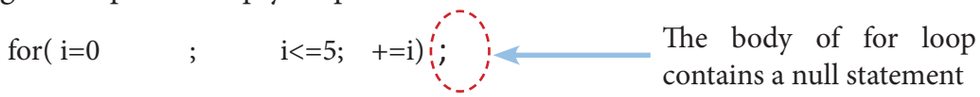


Similarly, the following for loop also forms an infinite loop.



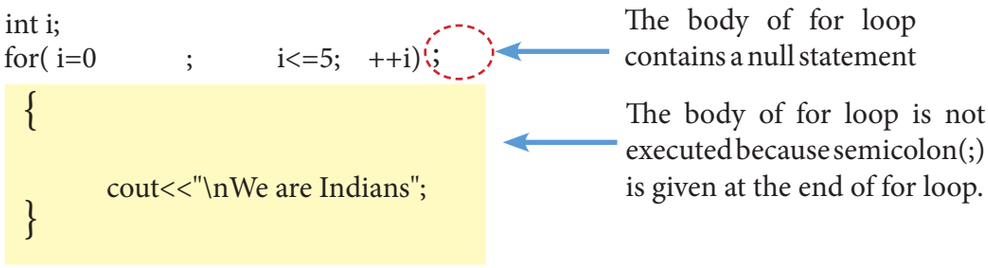
Empty loop

Empty loop means a loop that has no statement in its body is called an empty loop. Following for loop is an empty loop:



In the above code, the for loop contains a null statement, it is an empty loop.

Similarly, the following for loop also forms an empty loop.



In the above code, the body of a for loop enclosed in braces is not executed because a semicolon is given after the for loop.

Declaration of variable in a for loop

In C++, the variables can also be declared within a for loop. For instance,

```
int main ()
{
   int sum = 0;
   for(int i=0;  i<=5;  ++i )
   {
         sum = sum + i;
   }
   cout<<"\nThe variable i cannot be accessed here";
   cout<<"\n The variable sum can be accessed here";
}
```

Variable (i)is declared within the for loop.

The variable i can be accessed only within the body of loop.

A variable declared inside the block of main() can be accessed anywhere inside main() i.e., the scope of variable in main()

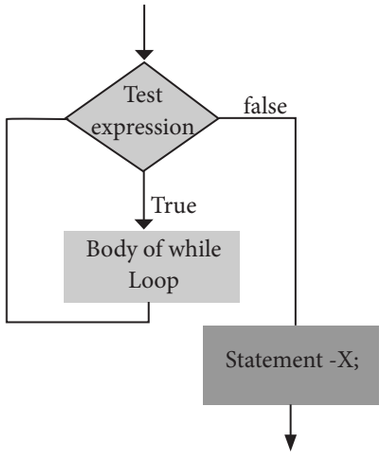
10.5.3 While loop

A while loop is a control flow statement that allows the loop statements to be executed as long as the condition is true. The while loop is an entry-controlled loop because the test-expression is evaluated before entering into a loop.

The while loop syntax is:

```
while ( Test expression )
{
        Body of the loop;
}
Statement-x;
```

The control flow and flow chart of the while loop is shown below.



Flowchart 10.7: while loop control flow and flowchart

In while loop, the test expression is evaluated and if the test expression result is true, then the body of the loop is executed and again the control is transferred to the while loop. When the test expression result is false the control is transferred to statement-x.

Illustration 10.10 C++ program to sum numbers from 1 to 10 using while loop

```
#include <iostream>
using namespace std;
int main ()
{
int i=1,sum=0;
while(i<=10)
{
        sum=sum+i;
        i++;
}
cout<<"The sum of 1 to 10 is "<<sum;
return 0;
}
```

Output

```
The sum of 1 to 10 is 55
```

In the above program, the integer variable i is initialized to 1 and the variable sum to 0. The while loop checks the condition, i <= 10, if the condition is true, the value of i, which is added to sum and i is incremented by 1. Again, the condition i <= 10 is checked. Since 2 <= 10, 2 is added to the earlier value of sum. This continues until i becomes 11. At this point in time, 11 <= 10 evaluates to false and the while loop terminates. After the loop termination, the value of sum is displayed.

Illustration 10.11 C++ program to find sum and average of 5 numbers using while loop

```
#include <iostream>
using namespace std;
int main ()
{
int i=1,num,avg,sum=0;
while(i<=5)
{
        cout<<"Enter the number : ";
        cin>>num;
        sum=sum+num;
        i++;
}
avg=sum/5;
cout<<"The sum is "<<sum<<endl;
cout<<"The average is "<<avg;
return 0;
}
```

Output

```
Enter the number : 1
Enter the number : 2
Enter the number : 3
Enter the number : 4
Enter the number : 5
The sum is 15
The average is 3
```

In the above program, integer variables **num** and **avg** are declared and variable i is initialized to 1 and sum to 0. The while loop checks the condition, since i <= 5 the condition is true, a number is read from the user and this is added to sum and i is incremented by 1. Now, the condition is i <= 5 is again checked. Since 2 <=5, the second number is obtained from the user and it is added to sum. This continues, until i becomes 6, at which point the while loop terminates. After the loop termination, the avg is computed and both sum and avg are displayed.

While loop variation

A while loop may contain several variations. It can be an empty loop or an infinite loop. An empty while loop does not have any statement inside the body of the loop except null statement i.e., just a semicolon.

For example

```
int main()
{

    int i=0;
    while(++i < 10000 ):
    return 0;

}
```

This is an empty loop because the while loop does not contain any statement

In the above code, the loop is a time delay loop. A time delay loop is useful for pausing the program for some time.

A while loop may be infinite loop when no update statement is given inside the body of the loop. For example,

```
int main()
{

    int i = 0;
    while(i < =10)
      cout <<"The value of i is "<<i;
      i++;

    return 0;

}
```

This statement will be displayed infinitely because no update statement inside the body of the loop

This is not a part of the while loop statement because of missing curly braces

10.5.4 do-while loop

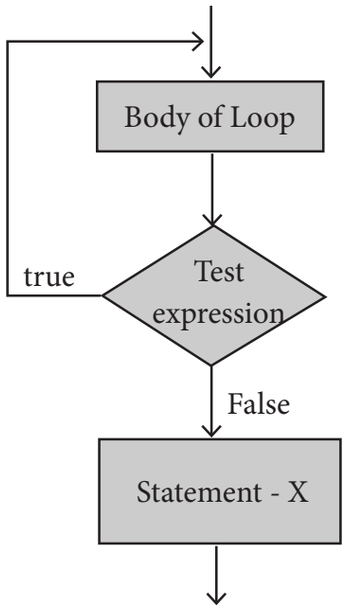
The do-while loop is an exit-controlled loop. In do-while loop, the condition is evaluated at the bottom of the loop after executing the body of the loop. This means that the body of the loop is executed at least once, even when the condition evaluates false during the first iteration.

The do-while loop syntax is:

```
do
{
        Body of the loop;

} while(condition);
```

The flow control and flowchart for do-while loop is shown below



Flowchart 10.8 : do-while loop control flow and flowchart

Illustration 10.12 C++ program to display number from 10 to 1 using do-while loop

```
#include <iostream>
using namespace std;
int main ()
{
int n = 10;
do
{
        cout<<n<<", ";
        n--;
}while (n>0) ;
}
```

Output

10, 9, 8, 7, 6, 5, 4, 3, 2, 1

In the above program, the integer variable **n** is initialized to **10**. Next the value of **n** is displayed as **10** and **n** is decremented by **1**. Now, the condition is evaluated, since **9 > 0**, again **9** is displayed and **n** is decremented to **8**. This continues, until **n** becomes equal to **0**, at which point, the condition **n > 0** will evaluate to false and the do-while loop terminates.

10.5.5 Nesting of loops

A loop which contains another loop is called as a nested loop.
The syntax is given below:

```
for (initialization(s); test-expression; update expression(s))
{
        for (initialization(s); test-expression; update expression(s)
        {
        statement(s);
         }
statement(s);
}
```

```
while(condition)
{
        while(condition)
        {
        statement(s);
        }
statement(s);
}
```

```
do
{
statement(s);
        do
        {
        statement(s);
        }while(condition);
} while( condition );
```

Illustration 10.13 C++ program to display matrix multiplication table using nested for loop

```
#include<iostream>
using namespace std;
int main(void)
{
    cout<< "A multiplication table:" <<endl <<"  1\t2\t3\t4\t5\t6\t7\t8\t9" <<endl<< "" <<endl;
    for(int c = 1; c < 10; c++)
    {
    cout<< c << "| ";
    for(int i = 1; i< 10; i++)
    {
    cout<<i * c << '\t';
    }
    cout<<endl;
    }
return 0;
}
```

Output

A multiplication table:

1	2	3	4	5	6	7	8	9
1 | 1	2	3	4	5	6	7	8	9
2 | 2	4	6	8	10	12	14	16	18
3 | 3	6	9	12	15	18	21	24	27
4 | 4	8	12	16	20	24	28	32	36
5 | 5	10	15	20	25	30	35	40	45
6 | 6	12	18	24	30	36	42	48	54
7 | 7	14	21	28	35	42	49	56	63
8 | 8	16	24	32	40	48	56	64	72
9 | 9	18	27	36	45	54	63	72	81

10.6 Jump statements

Jump statements are used to interrupt the normal flow of program. Types of Jump Statements are

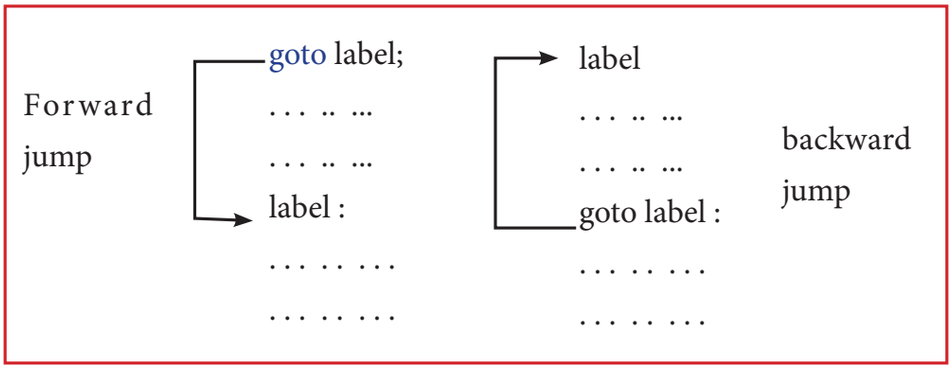
- goto statement
- break statement
- continue statement

10.6.1 goto statement

The goto statement is a control statement which is used to transfer the control from one place to another place without any condition in a program.

The syntax of the goto statement is;

Syntax1	**Syntax2**
goto label; ---------- ---------- ---------- **label:**	**label:** ---------- ---------- ---------- **goto label;**



In the syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.

Illustration 10.14 C++ program to display the first five odd numberts using goto statement

```
# include <iostream>
using namespace std;
int main()
{
int n=1;
jump:
{
 if(n<10)
 { // Control of the program move to jump:
        cout<<n<<'\t';
        n+=2;
        goto jump;
 }
 else
        return 0;
}  }
```

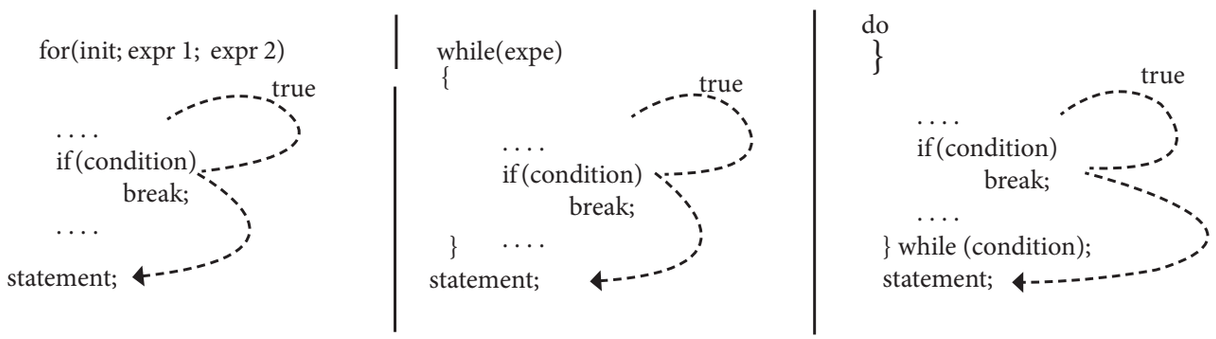
Output

1 3 5 7 9

In the above program the first five odd numbers are displayed.if n is is less than 10 goto transfers the control to jump statement .If n is greater than 10 the control comes out of the loop.

10.6.2 break statement

A break statement is a jump statement which terminates the execution of loop and the control is transferred to resume normal execution after the body of the loop. The following Figure. shows the working of break statement with looping statements;



break statement in for, while and do-while loop

Illustration 10.15 C++ program to count N numbers using break statement

```
#include <iostream>
using namespace std;
int main ()
{
int count = 1;
do
{
cout<< "Count : " << count <<endl;
if( count >= 3)
 {
        break;
  }
count ++;
}while( count < 20 );
return 0;
}
```

Output

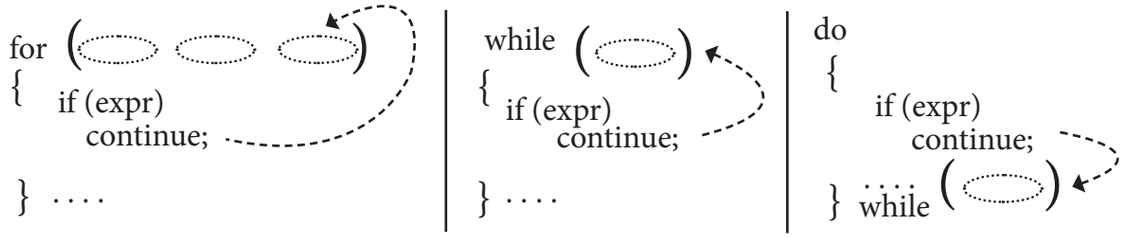
```
Count : 1
Count : 2
Count : 3
```

In the above example, the while loop will iterate for 20 times, but as soon as the count reaches 3, the loop is terminated, because of the break statement.

10.6.3 continue statement

The continue statement works quite similar to the break statement. Instead of terminating the loop (break statement), continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and next iteration of the loop will begin.

The following Figure describes the working flow of the continue statement



The workflow of the continue statement

In the above example, the loop will iterate 10 times but, if i reaches 6, then the control is transferred to for loop, because of the continue statement.

Illustraion 10.16 C++ program to display numbers from 1 to 10 except 6 using continue statement

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

Output

1 2 3 4 5 7 8 9 10

Difference between Break and Continue

Break	Continue
Break is used to **terminate** the execution of the loop.	Continue is not used to terminate the execution of loop.
It **breaks** the iteration.	It **skips** the iteration.
When this statement is executed, control will come out from the loop and executes the statement immediate after loop.	When this statement is executed, it will not come out of the loop but moves/jumps to the next iteration of loop.
Break is used with loops as well as switch case.	Continue is only used in loops, it is not used in switch case.

Hands on practice: 

Write C++ program to solve the following problems :

1. Program to input a character and to print whether a given character is an alphabet, digit or any other character.
2. Program to print whether a given character is an uppercase or a lowercase character or a digit or any other character. use ASCII codes for it. The ASCII codes are as given below:

Characters	ASCII Range
0' - '9'	48 - 57
'A' - 'Z'	65 - 90
'a' - 'z'	97 - 122
other characters	0- 255 excluding the above mentioned codes.

3. Program to calculate the factorial of an integer.
4. Program to print fibonacci series i.e., 0 1 1 2 3 5 8.....

5. Programs to produce the following design using nested loops

(a)	(b)
A A B A B C A B C D A B C D E A B C D E F	5 4 3 2 1 5 4 3 2 5 4 3 5 4 5

Points to Remember:

- A computer program is a set of statements or instructions to perform a specific task.
- There are two kinds of statements used in C++, viz Null and Compound Statement.
- Control Statement are statements that alter the sequence of flow of instructions.
- There are three kinds of statement used in C++. (1) Sequence Statement (2) Selection Statement (3) Iteration Statement
- *If* and *Switch* are Selection Statements.
- The Conditional Operator is an alternative for 'if else Statement'.
- The Switch Statment is a multi-way branching statement.
- Iteration Statement (looping) is use to execute a set of statements repeatedly until a condition is satisfied.
- There are three kinds Iteration Statements supported. (1) *for* (2) *While* (3) *do-While*.
- In C++ three Jump Statment are used (1) *goto* (2) *break* (3) *continue*

Evaluation

SECTION – A

Choose the correct answer

1. What is the alternate name of null statement?
   (A) No statement
   (B) Empty statement
   (C) Void statement
   (D) Zero statement

2. In C++, the group of statements should be enclosed within:
   (A) { } (B) [ ]
   (C) ( ) (D) < >

3. The set of statements that are executed again and again in iteration is called as:
   (A) condition (B) loop
   (C) statement (D) body of loop

4. The multi way branch statement:
   (A) if (B) if … else
   (C) switch (D) for

5. How many types of iteration statements?
   (A) 2 (B) 3
   (C) 4 (D) 5

6. How many times the following loop will execute? for (int i=0; i<10; i++)
   (A) 0 (B) 10
   (C) 9 (D) 11