UNIT-1

C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

Advantage of OOPs over Procedure-oriented programming language

1. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
2. OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.
4. Both Procedural Oriented Programming (POP) and Object Oriented Programming (OOP) are the high level languages in programming world and are widely used in development of applications. On the basis of nature of developing the code both languages have different approaches on basis of which both are differentiate from each other.
5. Following are the important differences between Procedural Oriented Programming (POP) and Object Oriented Programming (OOP)

Sr. No.	Key	Object Oriented Programming (OOP)	Procedural Oriented Programming (POP)
1	Definition	Object-oriented Programming is a programming language that uses classes and objects to create models based on the real world environment. In OOPs it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing ones.	On other hand Procedural Oriented Programming is a programming language that follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do.
2	Approach	In OOPs concept of objects and classes is introduced and hence the program is divided into small chunks called objects which are instances of classes.	On other hand in case of POP the main program is divided into small parts based on the functions and is treated as separate program for individual smaller program.
3	Access	In OOPs access modifiers are introduced namely as Private,	On other hand no such modifiers are

Sr. No.	Key	Object Oriented Programming (OOP)	Procedural Oriented Programming (POP)
	modifiers	public, and Protected.	introduced in POP.
4	Security	Due to abstraction in OOPs data hiding is possible and hence it is more secure than POP.	On other hand POP is less secure as compare to OOPs.
5	Complexity	OOPs due to modularity in its programs is less complex and hence new data objects can be created easily from existing objects making object-oriented programs easy to modify	On other hand there is no simple process to add data in POP at least not without revising the whole program.

C++ history

History of C++ language is interesting to know. Here we are going to discuss brief history of C++ language.

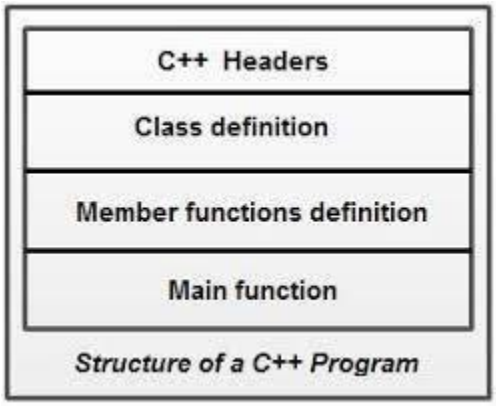
C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language.**

It was develop for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

C++ programming is "relative" (called a superset) of C, it means any valid C program is also a valid C++ program.

Structure of C++ program:



Structure of a C++ Program

C++ Features

C++ is object oriented programming language. It provides a lot of **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. Structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible
11. Object Oriented
12. Compiler based

1) Simple

C++ is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

3) Mid-level programming language

C++ is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.

4) Structured programming language

C++ is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library

C++ provides a lot of inbuilt functions that makes the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation. In C++ language, we can free the allocated memory at any time by calling the free() function.

7) Speed

The compilation and execution time of C++ language is fast.

8) Pointer

C++ provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

9) Recursion

In C++, we can call the function within the function. It provides code reusability for every function.

10) Extensible

C++ language is extensible because it can easily adopt new features.

C++ Basic Input/Output

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

I/O Library Header Files

Let us see the common header files used in C++ programming are:

Header File	Function and Description
<iostream.h>	It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.**
<fstream>	It is used to declare services for user-controlled file processing.

Standard output stream (cout)

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Standard input stream (cin)

The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

Standard end line (endl)

The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

Let's see the simple example of standard end line (endl):

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

int main( ) {
cout << "C++ Tutorial";
cout << " Javatpoint"<<endl;
cout << "End of line"<<endl;
}
```

Output:

```
C++ Tutorial Javatpoint
End of line
```

C++ Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

There are 4 types of data types in C++ language.

Types	Data Types
Basic Data Type	int, char, float, double, etc
Derived Data Type	array, pointer, etc
Enumeration Data Type	enum
User Defined Data Type	structure

Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

Let's see the basic data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 32,767
int	2 byte	-32,768 to 32,767
signed int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 32,767
short int	2 byte	-32,768 to 32,767
signed short int	2 byte	-32,768 to 32,767
unsigned short int	2 byte	0 to 32,767
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 18,446,744,073,709,551,615
float	4 byte	
double	8 byte	
long double	10 byte	

C++ Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring variable is given below:

```
int x;
float y;
char z;
```

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

```
int x=5,b=10; //declaring 2 variable of integer type
float f=30.8;
char c='A';
```

Rules for defining variables

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

Invalid variable names:

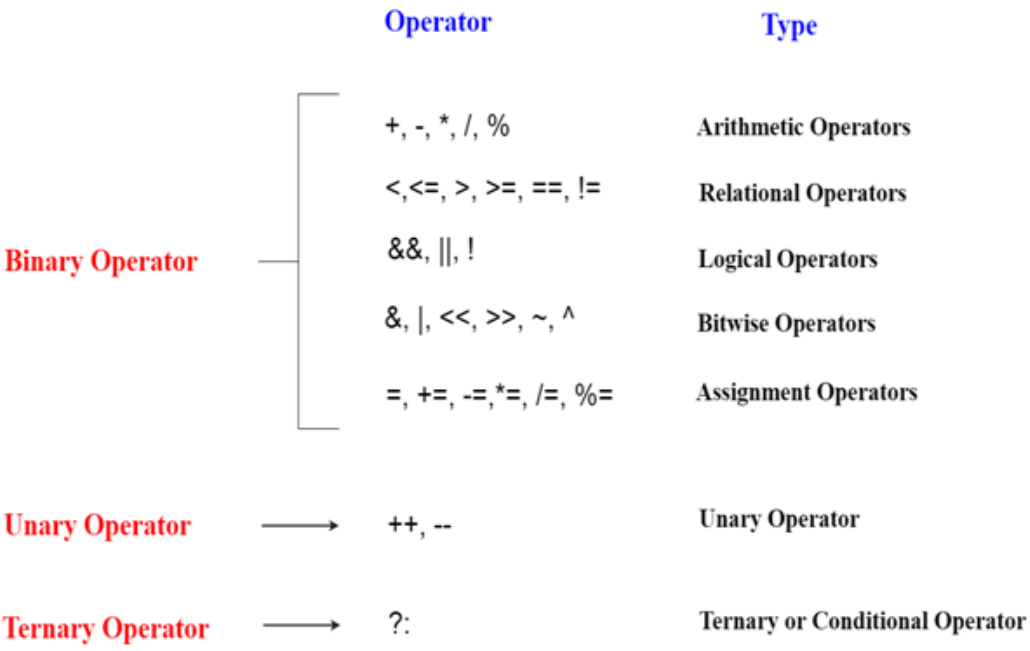
```
int 4;
int x y;
int double;
```

C++ Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operator
- Unary operator
- Ternary or Conditional Operator
- Misc Operator



Precedence of Operators in C++

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.

Let's understand the precedence by the example given below:

1. **int** data=5+10*10;

The "data" variable will contain 105 because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C++ operators is given below:

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

C++ Identifiers

C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

- **Constants**
- **Variables**
- **Functions**
- **Labels**
- **Defined data types**

C++ Control Statement

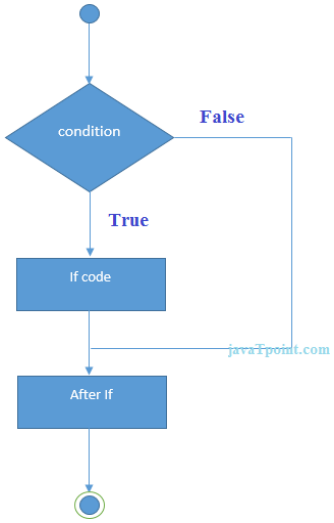
In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

```
if(condition){
//code to be executed
}
```



C++ If Example

```
#include <iostream.h>
 void main () {
  int num = 10;
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
  return 0;
}
```

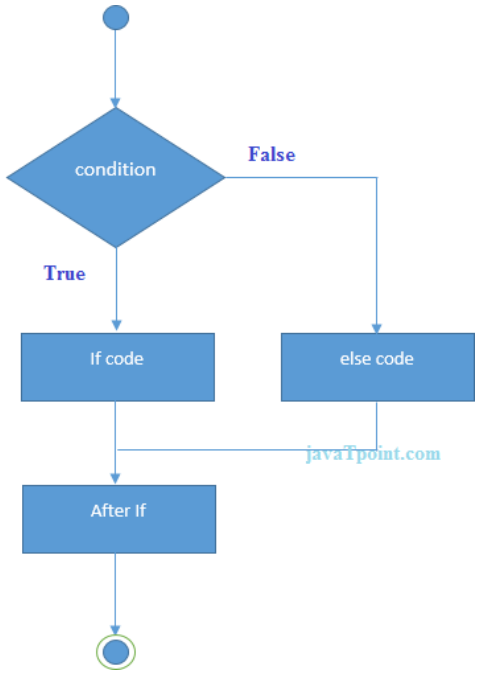
Output:

```
It is even number
```

C++ IF-else Statement

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```



C++ If-else Example

```
#include <iostream.h>
 void main () {
  int num = 11;
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
        else
        {
           cout<<"It is odd number";
        }
  return 0;
}
```

Output:

```
It is odd number
```

C++ If-else Example: with input from user

```
#include <iostream.h>
void main () {
   int num;
   cout<<"Enter a Number: ";
   cin>>num;
         if (num % 2 == 0)
         {
            cout<<"It is even number"<<endl;
         }
         else
         {
            cout<<"It is odd number"<<endl;
         }
  return 0;
}
```

Output:

```
Enter a number:11
It is odd number
```

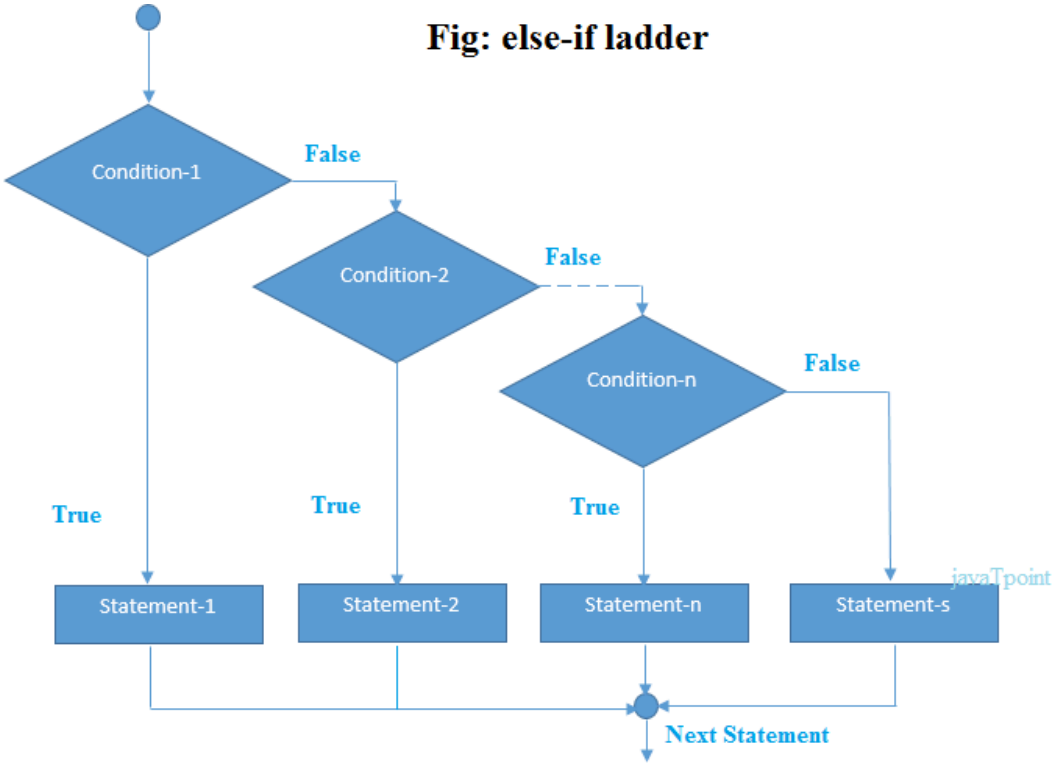
Output:

```
Enter a number:12
It is even number
```

C++ IF-else-if ladder Statement

The C++ if-else-if ladder statement executes one condition from multiple statements.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```



C++ If else-if Example

```
#include <iostream.h>
 void main () {
                    int num;
cout<<"Enter a number to check grade:";
cin>>num;
    if (num <0 || num >100)
    {
       cout<<"wrong number";
    }
    else if(num >= 0 && num < 50){
       cout<<"Fail";
    }
    else if (num >= 50 && num < 60)
    {
       cout<<"D Grade";
    }
    else if (num >= 60 && num < 70)
    {
       cout<<"C Grade";
    }
    else if (num >= 70 && num < 80)
    {
       cout<<"B Grade";
```

```
    }
    else if (num >= 80 && num < 90)
    {
       cout<<"A Grade";
    }
    else if (num >= 90 && num <= 100)
    {
       cout<<"A+ Grade";
    }
}
```

Output:

```
Enter a number to check grade:66
C Grade
```

Output:

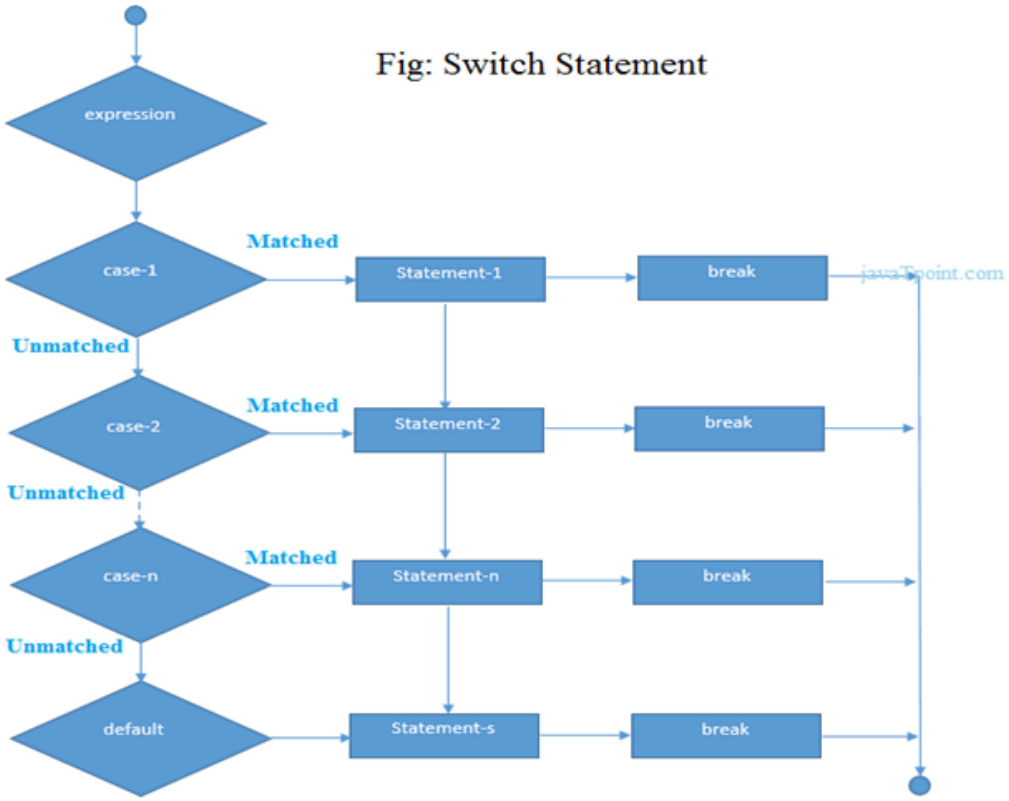
```
Enter a number to check grade:-2
wrong number
```

C++ switch

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

```
switch(expression){
case value1:
 //code to be executed;
 break;
case value2:
 //code to be executed;
 break;
.....

default:
 //code to be executed if all cases are not matched;
 break;
}
```



C++ Switch Example

```
#include <iostream.h>
 void main () {
    int num;
    cout<<"Enter a number to check grade:";
    cin>>num;
       switch (num)
      {
         case 10: cout<<"It is 10"; break;
         case 20: cout<<"It is 20"; break;
         case 30: cout<<"It is 30"; break;
         default: cout<<"Not 10, 20 or 30"; break;
      }
  }
```

Output:

```
Enter a number:
10
It is 10
```

Output:

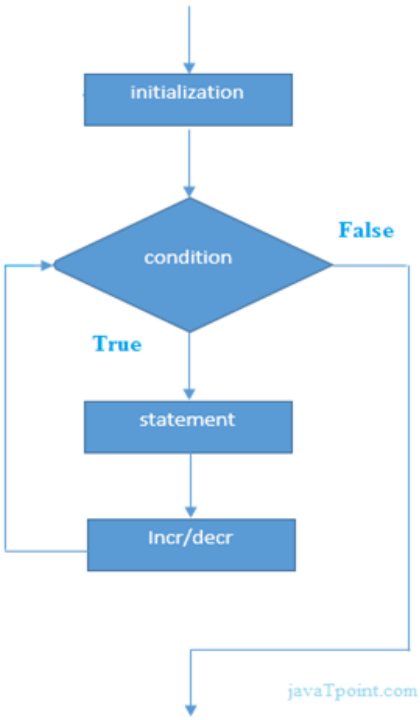
```
Enter a number:
55
Not 10, 20 or 30
```

C++ For Loop

The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

```
for(initialization; condition; incr/decr){
//code to be executed
}
```




```
#include <iostream.h>
void main() {
        for(int i=1;i<=10;i++){
          cout<<i <<"\n";
         }
    }
```

Output:

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

C++ Nested For Loop Example

Let's see a simple example of nested for loop in C++.

```
#include <iostream.h>

void main () {
      for(int i=1;i<=3;i++){
           for(int i=1;j<=3;j++){
            cout<<i<<" "<<j<<"\n";
       }
      }
  }
```

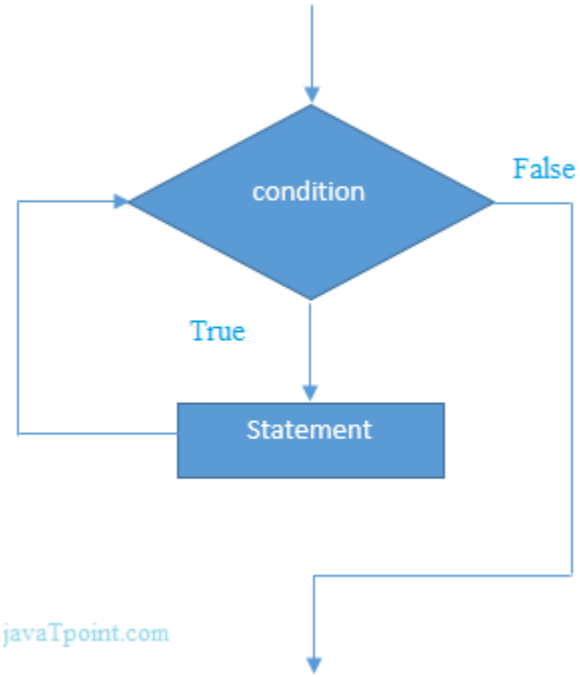
Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

C++ While loop

In C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

```
while(condition){
//code to be executed
}
```



C++ While Loop Example

Let's see a simple example of while loop to print table of 1.

```
#include <iostream.h>

void main() {
 int i=1;
       while(i<=10)
      {
          cout<<i <<"\n";
          i++;
        }
   }
```

Output:

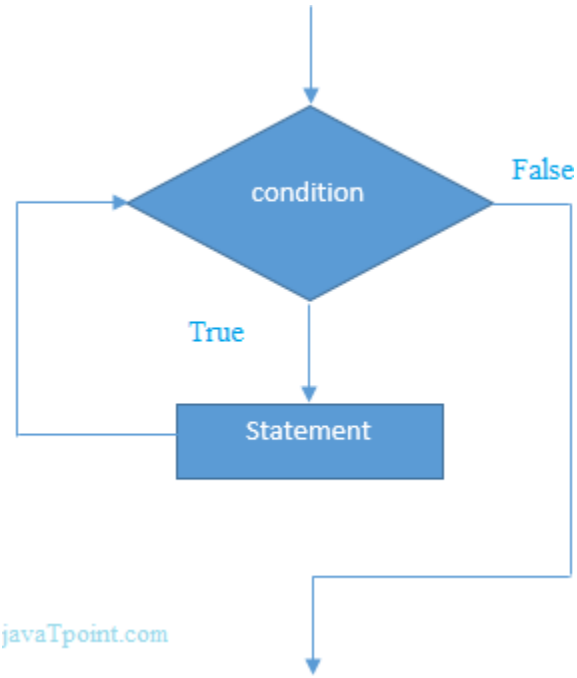
```
1
2
3
4
5
6
7
8
9
10
```

C++ Do-While Loop

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The C++ do-while loop is executed at least once because condition is checked after loop body.

```
do{
//code to be executed
}while(condition);
```



C++ do-while Loop Example

Let's see a simple example of C++ do-while loop to print the table of 1.

```
#include <iostream.h>

void main() {
    int i = 1;
        do{
            cout<<i<<"\n";
            i++;
        } while (i <= 10) ;
}
```

Output:

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

C++ Break Statement

The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

```
jump-statement;
```

```
break;
```

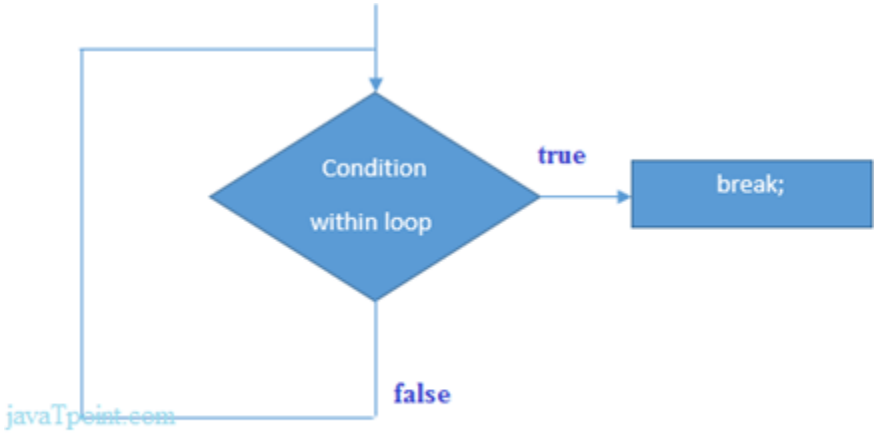


Figure: Flowchart of break statement

C++ Break Statement Example

Let's see a simple example of C++ break statement which is used inside the loop.

```
#include <iostream.h>

void main() {
    for (int i = 1; i <= 10; i++)
       {
          if (i == 5)
          {
             break;
          }
      cout<<i<<"\n";
       }
}
```

Output:

```
1
2
3
4
```

C++ Continue Statement

The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

```
jump-statement;
continue;
```

C++ Continue Statement Example

```
#include <iostream.h>
void main()
{
    for(int i=1;i<=10;i++){
        if(i==5){
            continue;
        }
        cout<<i<<"\n";
      }
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

C++ Goto Statement

The C++ goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

C++ Goto Statement Example

Let's see the simple example of goto statement in C++.

```
#include <iostream.h>

void main()
{
ineligible:
      cout<<"You are not eligible to vote!\n";
    cout<<"Enter your age:\n";
    int age;
    cin>>age;
    if (age < 18){
        goto ineligible;
    }
    else
    {

    }
```

```
cout<<"You are eligible to vote!";
```

```
}
```

Output:

```
You are not eligible to vote!
Enter your age:
16
You are not eligible to vote!
Enter your age:
7
You are not eligible to vote!
Enter your age:
22
You are eligible to vote!
```

C++ Functions

The function in C++ language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

Advantage of functions in C

There are many advantages of functions.

1) Code Reusability

By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

2) Code optimization

It makes the code optimized, we don't need to write much code.

Suppose, you have to check 3 numbers (531, 883 and 781) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

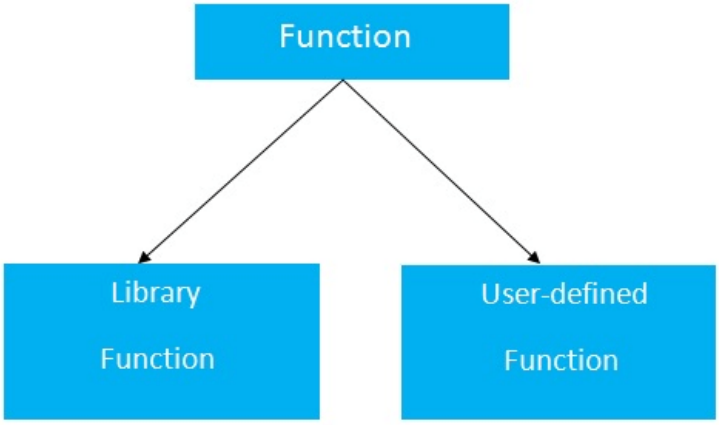
But if you use functions, you need to write the logic only once and you can reuse it several times.

Types of Functions

There are two types of functions in C programming:

1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.



Declaration of a function

The syntax of creating function in C++ language is given below:

```
return_type function_name(data_type parameter...)

{
//code to be executed
}
```

C++ Function Example

Let's see the simple example of C++ function.

```
#include <iostream.h>
void func() {
   static int i=0; //static variable
   int j=0; //local variable
   i++;
   j++;
   cout<<"i=" << i<<" and j=" <<j<<endl;
}
void main()
{
 func();
 func();
```

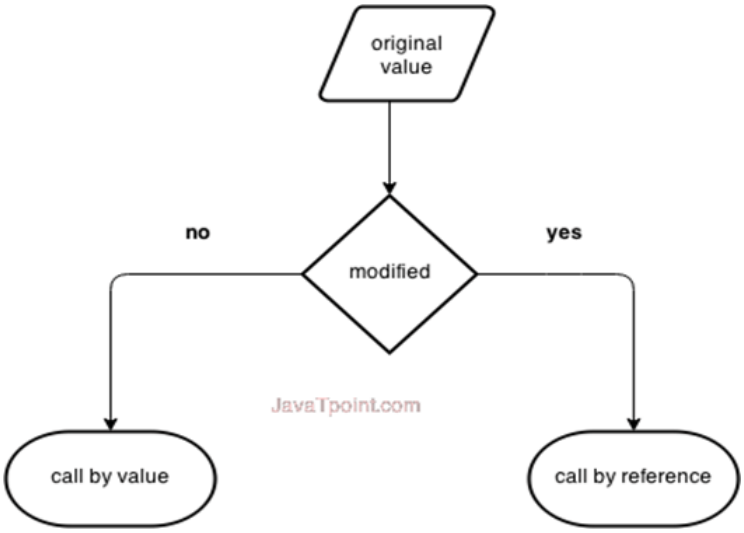
```
 func();
}
```

Output:

```
i= 1 and j= 1
i= 2 and j= 1
i= 3 and j= 1
```

Call by value and call by reference in C++

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.



Let's understand call by value and call by reference in C++ language one by one.

Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().
Let's try to understand the concept of call by value in C++ language by the example given below:

```
#include <iostream.h>

void change(int data);
int main()
{
int data = 3;
change(data);
cout << "Value of the data is: " << data<< endl;
return 0;
}
```

```
void change(int data)
{
data = 5;
}
```

Output:

```
Value of the data is: 3
```

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.
Let's try to understand the concept of call by reference in C++ language by the example given below:

```
#include<iostream.h>

void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
{
 int x=500, y=100;
 swap(&x, &y); // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

Difference between call by value and call by reference in C++

No.	Call by value	Call by reference

1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

C++ Recursion

When function is called within the same function, it is known as recursion in C++. The function which calls the same function, is known as recursive function.

A function that calls itself, and doesn't perform any task after function call, is known as tail recursion. In tail recursion, we generally call the same function with return statement.
Let's see a simple example of recursion.

```
recursionfunction(){

recursionfunction(); //calling self function
}
```

C++ Recursion Example

Let's see an example to print factorial number using recursion in C++ language.

```
#include<iostream.h>

void main()
{
int factorial(int);
int fact,value;
cout<<"Enter any number:
"; cin>>value;
fact=factorial(value);
cout<<"Factorial of a number is: "<<fact<<endl;
}
int factorial(int n)
{
if(n<0)
return(-1); /*Wrong value*/
if(n==0)
return(1); /*Terminating condition*/
else
{
return(n*factorial(n-1));
```

```
}
}
```

Output:

```
Enter any number: 5 Factorial
of a number is: 120
```

We can understand the above program of recursive method call by the figure given below:

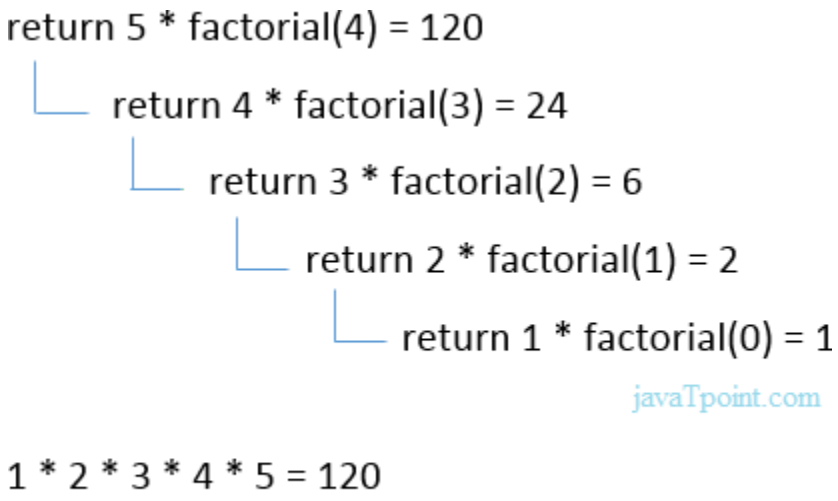


Fig: Recursion