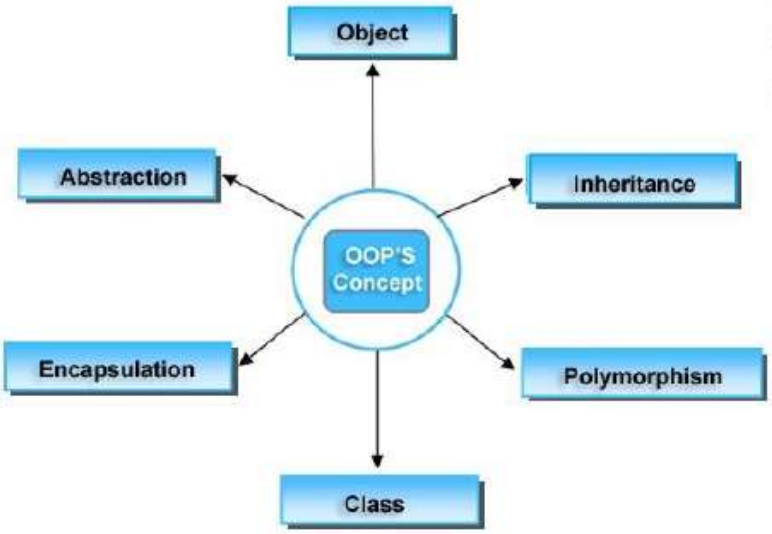
Chapter 01
Principles of Object Oriented Programming

1.1. Introduction

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 such as Object, Class, Inheritance, Polymorphism, Abstraction, and Encapsulation. 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. The main purpose of C++ programming was to add object oriented extension to the C programming language. If any programming language follows oops concept then that language called object oriented programming language.

A. Features of OOPS:



1. Object

Object is an entity that has state and behavior. For example: chair, pen, table, keyboard, bike etc. It can be physical or logical entity. Objects are basic run-time entities in an object oriented system, objects are instances of a class.

2. Class

Class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties. **Collection of objects** is called class. It is a logical entity. Class is also called as user defined data type or template for members such as data and functions.

3. Encapsulation

Encapsulation is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in C++ language by class concept. The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class. Binding (or wrapping) code and data together into a single unit is known as encapsulation.

4. Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. **Abstraction** is the concept of exposing only the required essential characteristics and behavior with respect to a context.
Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object or abstract classes can be used.

5. Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism. The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

6. Polymorphism

Ability to represent more than one form is known as **Polymorphism**. Polymorphism means one entity can represent many different behaviors. **One task is performed by different ways.** In C++, we use Function overloading and Function overriding to achieve polymorphism.

7. Dynamic Binding:

Binding means to link a function call with function definition. Binding can be of two types:
(a) Static binding. (b) Dynamic binding

8. Message Passing:

In Object Oriented Programming the objects communicate with each other by sending and receiving message to each another.

B. Characteristics of OOPS:

(1) It follows bottom up approach.
(2) Primary focus is given to data rather than function.
(3) Programs are divided into small parts known as objects.
(4) Data and functions are defined inside the class so it can be accessed only by the object of the class.
(5) Objects of the same class communicate with each other using member functions.
(6) New data and function can be easily added.

C. What is C++:

C++ is an **object oriented programming language**. It was developed by **Bjarne Stroustrup** in early **1980** at AT&T bell laboratories. Bjarne Stroustrup developed C++ by adding the feature of class so initially it was known as "C with Class". In 1983 it was given name C++. Because C++ is an incremented version of c it is called C++ using the concept of increment operator (++).

D. Object Oriented Languages:

C++, Java, VB.net, C#.net, SmallTalk, Objective C, Ruby, Perl, Python, JADE, LAVA, PHP5, Simula, Swift, and UberCode etc.....

1.2. Benefits of OOPS

- **Data encapsulation** benefit provided by the OOP using the concept of class to bind data and its associated functions together.
- **Data hiding** benefit provided by the OOP using the concept of public, private and protected visibility mode.
- The problem can be easily divided into smaller part using the concept of **object**.
- Using the concept of **function overloading** we can create same function with different number and type of arguments to perform different task.
- We can give additional meaning to the existing operator using the concept of **operator overloading**.
- **Code Reusability** benefit provided by the OOP using the concept of **inheritance**.
- Object of the same class communicates with each other using the concept of **message passing**.
- Complexity of the program can be easily managed using OOP.

1.3. Applications of OOPS

- User interface design such as windows, menu.
- Real Time Systems
- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation systems etc.

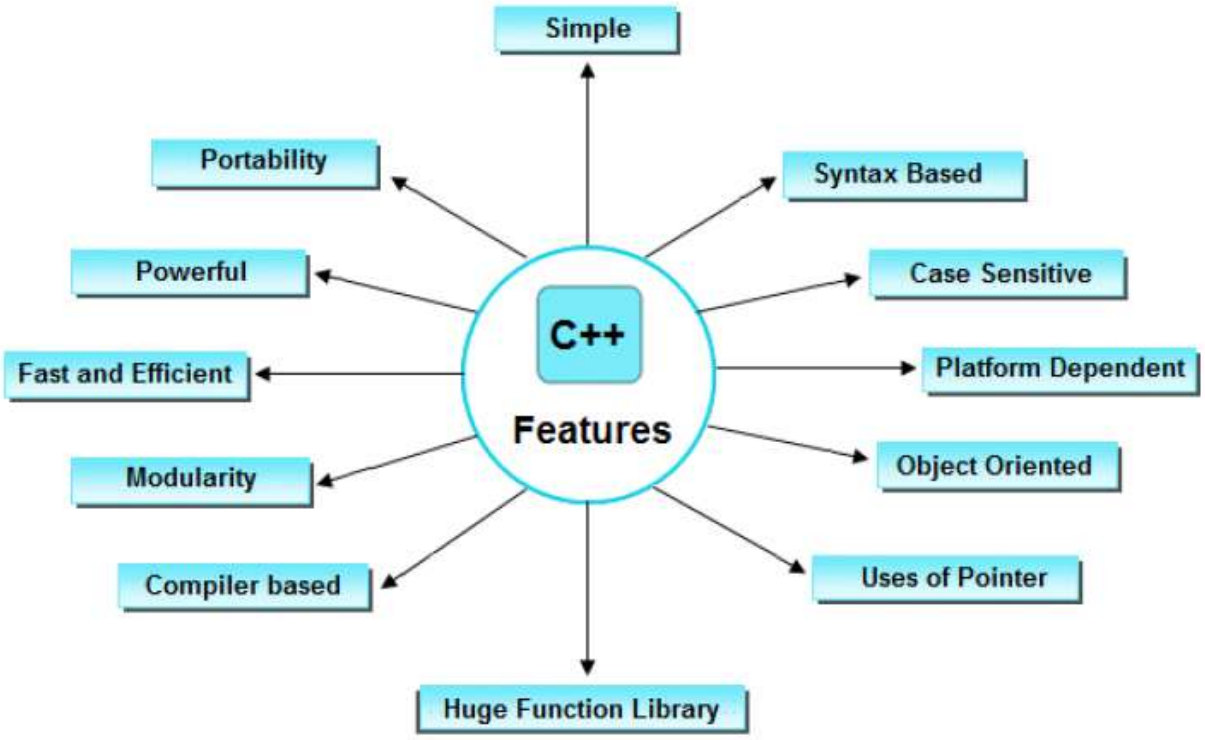
1.4. Differences

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP	OOP provides an easy way to add new

	is not so easy.	data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Examples of POP are: Algol, C, FORTRAN, and Pascal.	Examples of OOP are: C++, JAVA, VB.NET, C#.NET.

C	C++
C is Procedural Language.	C++ is non Procedural i.e Object oriented Language.
No virtual Functions are present in C	The concept of virtual Functions are used in C++.
In C, Polymorphism is not possible.	The concept of polymorphism is used in C++. Polymorphism is the most Important Feature of OOPS.
Operator overloading is not possible in C.	Operator overloading is one of the greatest Feature of C++.
Top down approach is used in Program Design.	Bottom up approach adopted in Program Design.
No namespace Feature is present in C Language.	Namespace Feature is present in C++ for avoiding Name collision.
Multiple Declaration of global variables are allowed.	Multiple Declaration of global varioables are not allowed.
In C • scanf() Function used for Input. • printf() Function used for output.	In C++ • Cin>> Function used for Input. • Cout<< Function used for output.
Mapping between Data and Function is difficult and complicated.	Mapping between Data and Function can be used using "Objects"
In C, we can call main() Function through other Functions	In C++, we cannot call main() Function through other functions.
C requires all the variables to be defined at the starting of a scope.	C++ allows the declaration of variable anywhere in the scope i.e at time of its First use.
No inheritance is possible in C.	Inheritance is possible in C++
In C, malloc() and calloc() Functions are used for Memory Allocation and free() function for memory Deallocating.	In C++, new and delete operators are used for Memory Allocating and Deallocating.
It supports built-in and primitive data types.	It support both built-in and user define data types.
In C, Exception Handling is not present.	In C++, Exception Handling is done with Try and Catch block.

1.5. Features of C++



Simple

Every C++ program can be written in simple English language so that it is very easy to understand and developed by programmer.

Platform dependent

A language is said to be platform dependent whenever the program is execute in the same operating system where that was developed and compiled but not run and execute on other operating system. C++ is platform dependent language.

Portability

It is the concept of carrying the instruction from one system to another system. In C++ Language **.cpp** file contain source code, we can edit also this code. **.exe** file contain application, only we can execute this file. When we write and compile any C++ program on window operating system that program easily run on other window based system.

Powerful

C++ is a very powerful programming language, it have a wide verity of data types, functions, control statements, decision making statements, etc.

Object oriented Programming language

This main advantage of C++ is, it is object oriented programming language. It follow concept of oops like polymorphism, inheritance, encapsulation, abstraction.

Case sensitive

C++ is a case sensitive programming language. In C++ programming 'break and BREAK' both are different. If any language treats lower case latter separately and upper case latter separately than they can be called as case sensitive programming language [Example c, c++, java, .net are sensitive programming languages.] otherwise it is called as **case insensitive** programming language [Example HTML, SQL is case insensitive programming languages].

Compiler based

C++ is a compiler based programming language that means without compilation no C++ program can be executed. First we need compiler to compile our program and then execute.

Syntax based language

C++ is a strongly tight syntax based programming language. If any language follows all rules and regulation very strictly then it is known as strongly syntax based language.

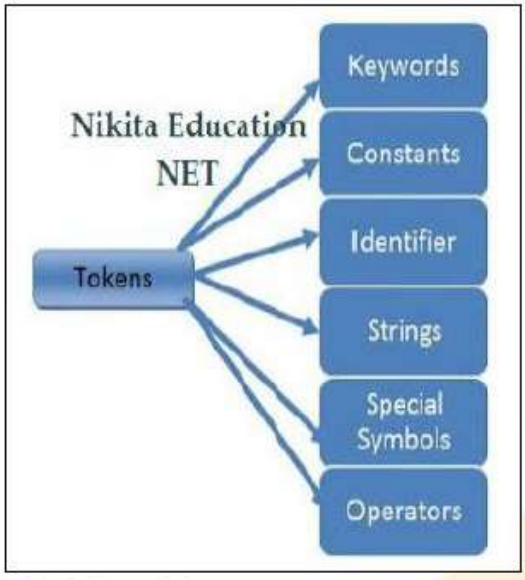
1.6. Expressions and Control Structures

A. Expressions

In programming, an expression is any **legal combination of symbols that represents a value or operation**. C++ Programming provides its own rules of Expression, whether it is legal expression or illegal expression. For example, in the C++ language x+5 is a legal expression. Every expression consists of at least one operand and can have one or more operators. Operands are values and Operators are symbols that represent particular actions. In C++ programming language, expressions are divided into THREE types. They are as follows...

1. Infix Expression
2. Postfix Expression
3. Prefix Expression

B. Tokens or basic building blocks



No	Token Type	Example 1	Example 2
1	Keyword	do	while
2	Constants	05	-14
3	Identifier	var1	sum
4	String	"NET"	"NSPL"
5	Special Symbol	*	@
6	Operators	++	/

Token	Meaning
Keyword	Keywords are reserved words used by compiler.
Constant	Constants are expressions with a fixed value
Identifier	The term identifier is usually used for variable names
String	Sequence of characters
Special Symbol	Symbols other than the Alphabets and Digits and white-spaces
Operators	A symbol that represent a specific mathematical or non mathematical action

C. Data types and variables

C++ Data Types

Data type is a keyword used to identify type of data. It is used for storing the input of the program into the main memory (RAM) of the computer by allocating sufficient amount of memory space in the main memory of the computer. In general every programming language is containing three categories of data types. They are

- Fundamental or primitive data types
- Derived data types
- User defined data types

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

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.

Syntax:

1. type variable_list;

Example:

1. **int** x;
2. **float** y;
3. **char** z;

Scope of Variable in C++

In C++ language, a variable can be either of global or local scope.

1. *Global variable*

Global variables are defined outside of all the functions, generally on top of the program. The global variables will hold their value throughout the life-time of your program.

2. *Local variable*

A local variable is declared within the body of a function or a block. Local variable only use within the function or block where it is declare.

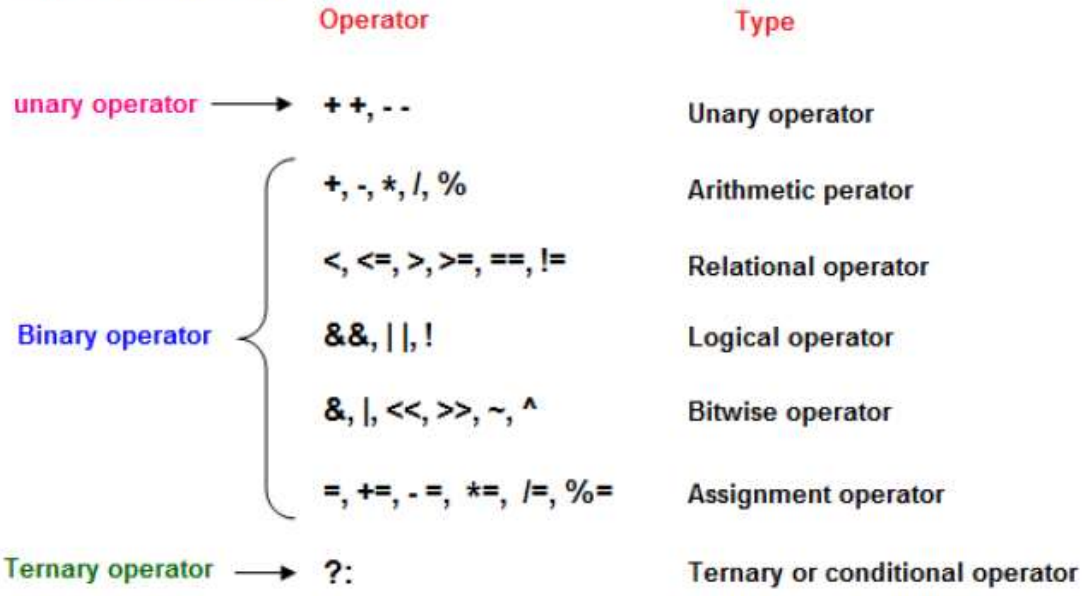
```
#include<iostream.h>
#include<conio.h>

int a;   // global variable
void main()
{
      int b;    // local variable
      a=10, b=20;
      cout<<"Value of a: "<<a;
      cout<<"Value of b: "<<b;
      getch();
}
```

D. Operators

Operator is a special symbol that tells the compiler to perform specific mathematical or logical or Bitwise Operation.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Ternary or Conditional Operators



E. Basic Structure of C++ Program

Documentation
Header files section
Namespaces [optional]
Global declarations
Class definitions
Main function

```
#include<headerfilename.h>                    //include section
using namespace namespacename;
global variables;
class class_name                              //class definition
{
        Access specifier:                     //private or public
        data members;                         //instance variables
        user defined method;                  //functions
        {
                .....
                .....
        }
};
```

```
returntype main()                                    //main method
{
.....
.....
}
```

Example:

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Welcome to Nikita Education for C++ Programming.";
   return 0;
}
```

F. Input / Output Functions

1. cout

Standard output stream. The cout is a predefined object of ostream class. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console.

```
#include <iostream>
using namespace std;
int main( ) {
   char ary[] = "Welcome to C++ tutorial";
   cout << "Value of ary is: " << ary << endl;
}
```

2. cin

Standard input stream. The cin is a predefined object of istream class. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

```
#include <iostream>
using namespace std;
int main( ) {
   int age;
   cout << "Enter your age: ";
   cin >> age;
   cout << "Your age is: " << age << endl;
}
```

G. Array & String in C++

An array is a collection of similar data type value in a single variable. It is a derived data type in C++, which is constructed from fundamental data type of C++ language.

Advantage of array

- **Code Optimization:** Less code is required; one variable can store numbers of value.

- **Easy to traverse data:** By using array easily retrieve the data of array.
- **Easy to sort data:** Easily short the data using swapping technique
- **Random Access:** With the help of array index you can randomly access any elements from array.

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

Syntax: Datatype arrayName[SIZE]; or Datatype ArrayName[SIZE][SIZE];

Example: int myarray[5]; or int arr[]={10,20,30,40,50};

H. Decision Making Statements

C++ conditional statements allow you to make a decision, based upon the result of a condition. These statements are called as Decision Making Statements or Conditional Statements.

- if statement
- if-else statement
- else-if statement
- switch statement
- Conditional Operator

I. Loop Control Statements

Sometimes it is necessary for the program to execute the statement several times, and C++ loops execute a block of commands a specified number of times until a condition is met.

- while loops
- do while loops
- for loops

J. Jump Control Statements

Jump control statements allow us to jump from one statement to another statement during program execution. C++ supports following jump control statements.

- goto statement
- break statement
- continue statement

K. Functions in C++

Functions are used for divide a large code into module, due to this we can easily debug and maintain the code. Functions are used for Code Re-usability, Develop an application in module format, Easily to debug the program, Code optimization**:** No need to write lot of code. There are two types of functions in C++ i.e. *Library function or pre-define function and User defined function.*

Syntax:

```
return_type  function_name(parameter)
{
        function body;
}
```

Chapter 02
Classes & Objects

1.1. Classes

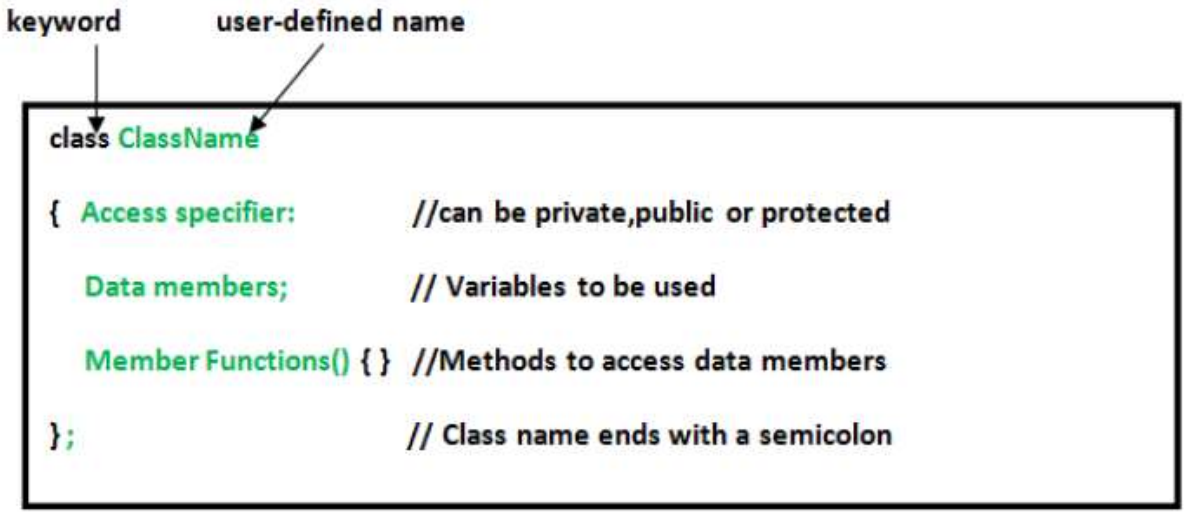
A. Class:

A class is a **user defined data type** that allows us to bind data and its associated functions together as a **single unit**. Thus class provides the facility of data encapsulation. Class provides the facility of data hiding using the concept of visibility mode such as **public, private and protected**. Once a class is defined we can create an object of the class to access variables and functions defined inside the class.

A class is **an abstract data type** similar to 'C structure'. The class is **a container** for data members and methods. Class is **a group of similar objects** or it is a template from which objects are created. A class is a blueprint for the object.

A class in C++ contains, following properties;

- Data Member
- Method
- Constructor
- Block



Syntax:

```
class Class_Name
{
      Private:
            Data-Type Variable_Name;
            Function declaration or Function Definition;
      Public:
            Data-Type Variable_Name;
            Function declaration or Function Definition;
};
```

Class can be created using the **class keyword**. The class definition starts with curly bracket and ends with curly bracket followed by semicolon. We can declare variables as well as functions inside the curly bracket as shown in the syntax. The variables defined inside class are known as **data member** and the function declared inside the class are known as **member function**. In order to provide **data hiding** facility class provides the concept of **visibility mode** such as **private, public or protected**. If you don't specify any visibility mode for the member of the class then **by default all the members of the class are considered as private**.

Example:

```
class test
{
      int a, b;                              //data member
      public:                                //access specifier
            void input ();                   //member function
            {
                  cout<<"Enter Value of a and b";
                  cin>>a>>b;
            }
            void output ()                   //member function
            {
                  cout<<"A="<<a<<endl<<"B="<<b;
            }
};
```

1.2. Objects

In C++, Object is a **real world entity**, for example, chair, car, pen, mobile, laptop etc. In other words, object is an entity that **has state and behavior**. Here, **state means data** and **behavior means functionality**. Object is a **runtime entity**, it is created at runtime. Object is **an instance** of a class. All the members of the class can be accessed through object. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

An Object in C++ has three characteristics:

- State: Represents data (value) of an object.
- Behavior: Represents the behavior (functionality).
- Identity: Object identity is typically implemented via a unique ID.

Syntax: `ClassName ObjectName;`

Example: `Employee e1, e2, e3;`

In real world many examples of object and class like dog, cat, and cow are belong to animal's class. Each object has state and behaviors. For example a dog has state:- color, name, height, age as well as behaviors:- barking, eating, and sleeping.

Note: Objects are created like variables using class name as its type.

1.2.1. Create Object in C++

There are two ways to create objects in C++:

a. Create object with class definition

```
class test
{
   int a,b;
   public:
        void input ();
        void ouput ();
}t1,t2,t3;
```

Here, t1, t2 and t3 are the objects of class test. Similar to creating references of structure in C.

b. Create object inside any function

```
class test
{
   int a,b;
   public:
        void input (){ …… }
        void ouput (){ …… }
};

 void main()
 {
   test t1, t2, t3;          //object of class test
 }
```

1.2.2. Accessing Class Members Using Objects:

The data member and member function declared as a public can be accessed by **dot (.) or pointer (-->)** operator directly using the object of the class. But the data member and member function declared as private cannot be accessed directly using the object of the class.

```
Object_Name . Data_Member = value;

Object_Name . Member_Function (Argument_List);
```

Example:

```
t1. input ();

t1. output ();
```

Example:

```
class Test
{
      int b;                  //by default b is private variable of class
      Public:
            int a;            //a is public variable of class
            void inputb()     //member function of a class
            {
                  b=20;
            }
};

int main()
{
      Test T1;
      T1.a=10;                //works because a is public
      T1.b=20;                //error because b is private
      T1.inputb ();           //works because inputb () is public
      return 0;
 }
```

1.3. Access Specifiers/Modifiers in C++

Access modifiers are used to implement important feature of Object Oriented Programming known as **Data Hiding**. Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions. There are 3 types of access modifiers available in C++:

1. Public
2. Private
3. Protected

Note: If we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be **Private**.

Examples:

```
class PublicAccess {
      public:           // public access specifier
      int x;            // Data Member Declaration
      void display();   // Member Function decaration
}

class PrivateAccess {
      private:          // private access specifier
      int x;            // Data Member Declaration
      void display();   // Member Function decaration
}
```

```
class ProtectedAccess {
        protected:          // protected access specifier
        int x;              // Data Member Declaration
        void display();     // Member Function decaration
}
```

a. Public: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

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

class Circle              // class definition
{
    public:
        double radius;

        double  compute_area()
        {
            return 3.14*radius*radius;
        }

};

int main()          // main function
{
    Circle obj;

    // accessing public datamember outside class
    obj.radius = 5.5;

    cout << "Radius is:" << obj.radius << "\n";
    cout << "Area is:" << obj.compute_area();
    return 0;
}

Output:
Radius is:5.5
Area is:94.985
```

b. Private: The class members declared as **private** can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

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

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double  compute_area()
        {   // member function can access private
            // data member radius
            return 3.14*radius*radius;
        }

};
// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;

    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

Output:

```
In function 'int main()':
11:16: error: 'double Circle::radius' is private
       double radius;
              ^
31:9: error: within this context
    obj.radius = 1.5;
      ^
```

However we can access the private data members of a class indirectly using the public member functions of the class. Below program explains how to do this:

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

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double  compute_area(double r)
        {   // member function can access private
            // data member radius
            radius = r;

            double area = 3.14*radius*radius;

            cout << "Radius is:" << radius << endl;
            cout << "Area is: " << area;
        }

};

// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);

    return 0;
}
```

Output:
Radius is:1.5
Area is: 7.065

c. **Protected**: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class.

```
#include <bits/stdc++.h>
using namespace std;

// base class
class Parent
{
    // protected data members
    protected:
    int id_protected;

};

// sub class or derived class
class Child : public Parent
{


    public:
    void setId(int id)
    {

        // Child class is able to access the inherited
        // protected data members of base class

        id_protected = id;

    }

    void displayId()
    {
        cout << "id_protected is:" << id_protected << endl;
    }
};

// main function
int main() {

    Child obj1;

    // member function of derived class can
    // access the protected data members of base class

    obj1.setId(81);
```

```
    obj1.displayId();
    return 0;
}
```

Output:
id_protected is:81

1.4. Member Functions

Function is a set of instructions or self contained block of statements used to perform specific task. Functions are used to divide complex task into smaller manageable units called as modules of the application. The primary aim of functions is to avoid repetitive code in main program.

Syntax:
```
return_type function_name( parameter list )
{
    body of the function
}
```

Member function can be defined in two different way:
(1) Inside class
(2) Outside class

(1) Inside Class:

When we declare the function in the class at the same time we can also give the definition of the function in the class as shown below. The function defined inside class becomes **inline** by default.

```
class Test
{
    int a,b;
public:
    //member function declared and defined inside the class
    void input ()
    {
        cout<<"Enter Value of a";
        cin>>a>>b;
    }
};
```

(2) Outside Class:

We can also define the member function outside the class. But at that time we have to instruct compiler this function belongs to which class using scope resolution operator as follow:

Syntax:
```
Return-Type Class_Name :: Function_Name (parameter list)
{
    Function definition
}
```

Example:

```
class Test
{
      int a,b;
      Public:
            void input ();
};
void test :: input ()
{
      cout<<"Enter Value of a";
      cin>>a>>b;
}
```

1.4.1. Private Member Function

Like data member of the class a member function can also be declared as private so that it cannot be accessed outside the class. If we declare private member function then it cannot be accessed directly using the object of the class so we have to access it from the public member function of the same class.

Example:

```
class Circle
{
      int r;
      float area ();

      public:
            void getRadius ();
            void DisplayArea ();
};

int Circle ::area ()
{
      return (3.14 * r * r);
}

void Circle :: getRadius ()
{
      cout<<"Enter Radius";
      cin>>r;
}

void Circle ::DisplayArea ()
{
      cout<<"Area="<<area ();
}
```

```
int main()
{
      Circle C1;
      C1.getRadius();
      C1.DisplayArea ();
      return 0;
}
```

In the above example we have declared three member function getRadius (), DisplayArea () and area (). In which area () is defined as private. We have called only two member functions getRadius () and DisplayArea () using the name of the object inside main function. We have called the third member function area () from inside the DisplayArea () member function. This concept is known as private member function.

1.4.2. Types of Member Functions in Class

1. Simple functions
2. Static functions
3. Const functions
4. Inline functions
5. Friend functions
6. Virtual functions

Simple Member functions

These are the basic member function, which don't have any special keyword like static etc as prefix. All the general member functions, which are of below given form, are termed as simple and basic member functions.

```
      return_type functionName(parameter_list)
      {
            function body;
      }
```

Static member functions

Static member functions are designed to work with static data members. In order to make a member function as static we need to precede the function declaration with **static keyword**. Static member function can access only static member of the class in which it is declared. Static member function is not a part of class object so it cannot be called using object of the class. Static member function can be called using class name and scope resolution operator as shown below:

```
                  Class_Name :: Function_Name ();
class X
{
      public:
      static void f(){};
};
int main()
{
      X::f();   // calling member function directly with class name
}
```

Const Member functions

When used with member function, such member functions can never modify the object or its related data members.

```
//Basic Syntax of const Member Function
void fun() const {}
```

Inline functions

All the member functions defined inside the class definition are by default declared as Inline. We will study Inline Functions in details in the next topic.

Friend functions

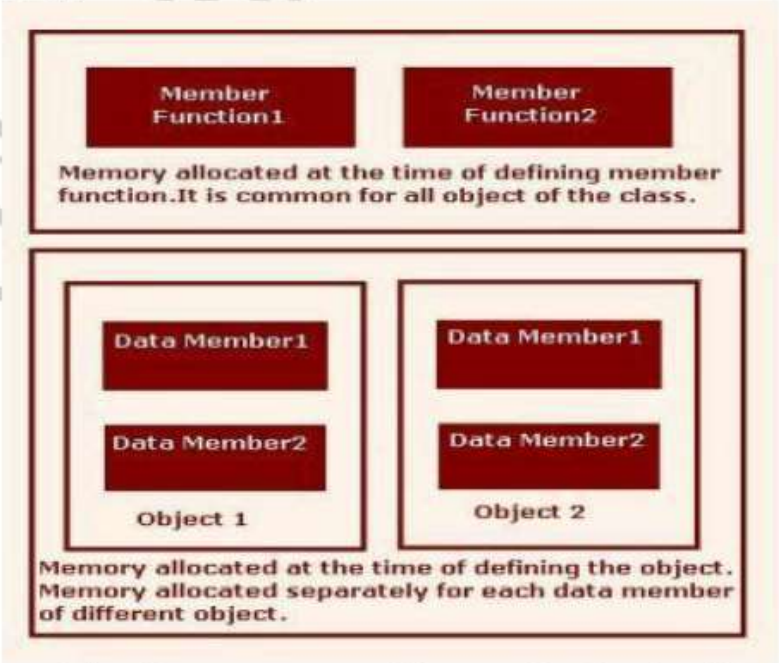
Friend functions are actually not class member function. Friend functions are made to give **private** access to non-class functions. You can declare a global function as friend, or a member function of other class as friend.

Virtual Functions

Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform **Late Binding** on this function. Virtual Keyword is used to make a member function of the base class Virtual.

1.5. Memory Allocation

Once you define class it will not allocate memory space for the data member of the class. The memory allocation for the data member of the class is performed separately each time when an object of the class is created. Since member functions defined inside class remains same for all objects, only memory allocation of member function is performed at the time of defining the class. Thus memory allocation is performed separately for different object of the same class. All the data members of each object will have separate memory space. The memory allocation of class members is shown below:



Hence data member of the class can contain different value for the different object; memory allocation is performed separately for each data member for different object at the time of creating an object. Member function remains common for all objects. Memory allocation is done only once for member function at the time of defining it.

1.5.1. Dynamic Memory Allocation

C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory. C++ supports these functions and also has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a better and easier way.

Allocation of heap memory using new operator

Syntax: datatype pointername = new datatype

Example:
```
int *new_op = new int;
//Allocating block of memory
int *new_op = new int[10];
//allocating memory for object of class Stud
Stud * S = new Stud();
```

Deallocation of memory using delete operator

Syntax: delete pointer_variable

Example: delete new_op; or delete S;

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

class Box
{
public:
   int a,b;
   int * c=new int;
   void show()
   {
      cout<<a<<" , "<<b;
   }
};

int main()
{
   Box * p=new Box();
   p->a=50;
   p->b=10;
   p->show();
   delete p;
}
```

1.6. Arrays

An array is a collection of similar data type value in a single variable. It is a derived data type in C++, which is constructed from fundamental data type of C++ language. Array in C++ is a group of similar types of elements that have contiguous memory location. Arrays can be declared as the data members of a class. The arrays can be declared as private, public or protected members of the class.

Advantages of C++ Array

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

C++ Array Types

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

1.6.1. Arrays within a Class

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

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

const int size=5;

class Student{
   int roll_no;
   int marks[size];
public:
   void getdata ();
   void tot_marks ();
};

void Student :: getdata()
{
   cout<<"\nEnter roll no: ";
   cin>>roll_no;
   for(int i=0; i<size; i++)
   {
      cout<<"Enter marks in subject"<<(i+1)<<": ";
      cin>>marks[i] ;
   }
}
void Student :: tot_marks() //calculating total marks
{
   int total=0;
   for(int i=0; i<size; i++)
      total+= marks[i];
   cout<<"\n\nTotal marks "<<total;
}
```

```
int main()
{
   Student stu;
   stu.getdata() ;
   stu.tot_marks() ;
   return 0;
}
```

1.6.2. Array of Objects

- Like array of other user-defined data types, an array of type class can also be created.
- The array of type class contains the objects of the class as its individual elements.
- Thus, an array of a class type is also known as an array of objects.
- An array of objects is declared in the same way as an array of any built-in data type.

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

class Student
{
   int rollno;
   char name[20];
public:
   void Input();
   void Output();
};

void Student::Input()
{
   cout<<"Enter Roll Number:";
   cin>>rollno;
   cout<<"Enter Name:";
   cin>>name;
}

void Student::Output()
{
   cout<<"Roll Number:"<<rollno<<endl;
   cout<<"Name:"<<name<<endl;
}
int main()
{
   Student S[3];
   int i;
   for(i=0;i<3;i++)
      S[i].Input();
   for(i=0;i<3;i++)
      S[i].Output();
   return 0;
}
```

1.7. Object as a Function Argument

In C++ we can pass object as a function argument and also return an object from the function. An object can be passed to the function using two methods:
(1) Call By value: In which copy of the object is passed to the function.
(2) Call By reference: In which an address of the object is passed to the function.

1.7.1. Passing object as an argument

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

class Student
{
   int a,b,c;
   int d;
public:
   void add()
   {
      a=b=23;
      c=a+b;
   }
   void mul(Student d1)
   {
      int res;
      d=5;
      res=d*d1.c;
      cout<<"result is :"<<res<<endl;
   }
};

int main()
{
   Student d1;
   d1.add();
   Student d2;
   d2.mul(d1);
   return 0;
}
```

1.7.2. Returning object from function

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

class Demo
{
   int roll;
public:
   Demo setRoll(int r);
```

```
    void show(Demo ob2);
};

Demo Demo::setRoll(int r)
{
    Demo ob2;
    ob2.roll=r;
    return ob2;
}

void Demo::show(Demo ob2)
{
    cout<<ob2.roll;
}

int main()
{
    Demo ob1;
    Demo ob2=ob1.setRoll(175105);
    ob1.show(ob2);
    return 0;
}
```

1.8. Static Members

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following:

- Static Data Members
- Static Member Functions

1.8.1. Static data members

- To declare **static** keyword is used
- Static creates a single common variable means it allocates memory only once to variable
- Static variable is commonly shared by all the objects of same class
- Static data members are initialized before the objects of class are created
- *Static data member is declared in the class but it must be defined outside the class using class name and scope resolution operator (::)*

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

class item
{
    static int count;
public:
    void DisplayCounter()
    {
        count++;
```

```
        cout<<"count:"<<count<<endl;
    }
};

int item::count;        //static data member must be defined outside the class

int main()
{
    item a,b,c;
    a.DisplayCounter();
    b.DisplayCounter();
    c.DisplayCounter();
    return 0;
}
```

1.8.2. Static member functions

- Static member functions are designed to work with static data members.
- In order to make a member function as static we need to precede the function declaration with **static keyword**.
- Static member function can access only static member of the class in which it is declared.
- Static member function is not a part of class object so it can not be called using object of the class.
- Static member function can be called using class name and scope resolution operator as shown below:

Class_Name :: Function_Name ();

```
#include <iostream>
using namespace std;
class test
{
private:
    static int count;
public:
    void setCount(void);
    static void DisplayCounter(void);
};

void test :: setCount(void)
{
    count++;
}

void test :: DisplayCounter(void)
{
    cout<<"Count:"<< count << endl;
}

int test::count;
```

```
int main(void)
{
   test t1, t2;
   t1.setCount();
   test :: DisplayCounter();
   t2.setCount();
   test :: DisplayCounter();
   return(0);
}
```

**********Difference Between**********

Object Oriented	Procedure Oriented
In Object Oriented Programming primary focus is on object.	In Procedure Oriented Programming primary focus is on function.
It follows bottom up approach.	It follows top down approach.
Primary importance is given to data instead of function.	Primary importance is given to function instead of data.
The program is divided into small parts Known as object.	The program is divided into small parts Known as function.
Data can not move freely from one function to another function.	Data moves freely from one function to another function.
It provides data hiding facility.	It does not provides data hiding facility.
Data and functions can be added easily Whenever required.	Data and functions cannot be added easily.
C++, JAVA is the example of Object Oriented Programming Language.	C, COBOL, FORTRAN is the example of Procedure Oriented Programming.

Data Abstraction	Data Encapsulation
Data abstraction refers to the process of providing only essential information to the outside word and hiding their background details.	The process of binding data and its associated function into a single unit is known as data encapsulation.

cout	cin
cout is known as an output statement.	cin is known as an input statement.
It is used to display the content of the variable or string on the output screen.	It is used to input data into the program.
cout is a predefined object defined in ostream.h which represents the standard output stream in C++ which is monitor.	cin is a predefined object defined in istream.h file which represents the standard input stream in C++ which is keyboard.
Syntax: cout << Variable_Name or "String"; Example cout << "Welcome To C++";	Syntax: cin >> Variable_Name ; Example cin >>a;

Single Line Comment	Multi Line Comment
It was introduced in C++ so it is also known as C++ Style comment.	It was introduced in C so it is also known as C Style comment.
It is starts with //.	It is starts with /* and ends with */ symbols.

Example:
// This Program Implements Class
// and Objects
	Example:
	/* This Program Implements Class And Objects
	*/

We can not use this comment inside C++ statements.	We can also use this comment inside C++ statements.

While Loop	Do While Loop
In while loop, condition is checked at the beginning of loop.	In do...while loop, condition is checked at the end of loop.
It is known as entry controlled loop.	It is known as exit controlled loop.
If condition is false at the first trial body of the loop never executes.	If condition is false at the first trial body of the loop executes at least once.

Syntax:
while(condition)
{
Body of loop
}
	Syntax:
	do
	{
	Body of loop
	} while(condition);

Break	Continue
Break statement is used to transfer control of the program immediately after the loop or switch case statement.	Continue statement is used to skip some part of the loop and transfer control of the program to the next iteration in the loop.

Normal Function	Inline Function
Normal Function decrease execution speed of the program.	Inline Function increase execution speed of the program.
It occupies less memory during runtime as compared to inline function.	It occupies more memory during runtime as compared to normal function.
Normal function can contain any number of statements.	Inline function works fine if it contains only two or three statements.

Public	Private
Public members of the class can be accessed directly using object of the class.	Private members of the class can not be accessed directly using object of the class.
Inheritance of public members is possible.	Inheritance of private members is not possible.

Call By Value	Call By Reference
In this method of calling the function values of the original variables is passed to the function.	In this method of calling the function addresses of the original variables is passed to the function using concept of pointer.
As you pass the values of original variables it is copied in the formal parameters of the function. Thus function works with the copy of the original variables. Any changes that are made to the variables inside function do not affect the original variables.	As you pass the addresses of original variables, the function works with the original variables. Any changes that are made to the variables inside function also affect the original variables.
Call by value method can not alter the value of the original variable.	Call by reference method alters the value of the original variable.

Compile Time Polymorphism	Run Time Polymorphism
Compile time polymorphism is also known as static binding or early binding.	Run time polymorphism is also known as dynamic binding or late binding.
Function overloading and Operator overloading are the example of compile time polymorphism.	Virtual function is the example of run time polymorphism.
In compile time polymorphism which function to invoke is determined at compile time so it is called static or early binding.	In run time polymorphism which function to invoke is determined at runtime so it is called dynamic binding or late binding.

Default Constructor	Parameterized Constructor
Default Constructor does not accept any arguments.	Parameterized Constructor accept one or more arguments.
It initialize same value for data member of different objects.	It can initialize different value for data member of different objects.