Program Name	**B.Tech in Computer Engineering/ Computer Science and Engineering**
Semester	**FOURTH**
Course Title	**Object Oriented Programming Using C++ (Elective-I)**
Course Code	**BTCOE404 (A)**

UNIT -1 Introduction to Object Oriented Programming & Class and Objects

1.1 Need of Object Oriented Programming

- As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming.
- Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

1.2 The Object Oriented Approach

- In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.
- In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and

maintenance. It identifies the objects in problem domain, classifying them in terms of data and behavior.

- The OO model is beneficial in the following ways −
    - It facilitates changes in the system at low cost.
    - It promotes the reuse of components.
    - It simplifies the problem of integrating components to configure large system.
    - It simplifies the design of distributed systems.

1.3 Characteristics of Object Oriented Languages

Object-oriented language (OOL) is a high-level computer programming language that implements objects and their associated procedures within the programming context to create software programs.

- Simula, the first object-oriented programming language.
- Java.
- Python.
- Ruby.
- C++
- Smalltalk.
- Visual Basic .NET.
- Objective-C: OOP is a core tenet of iOS mobile app programming, and Objective-C is essentially the C language with an object-oriented layer.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

1.4 Procedure Oriented Programming versus Object Oriented Programming

Structured Approach (POP)	Object Oriented Approach (OOP)
It works with Top-down approach.	It works with Bottom-up approach.
Program is divided into number of submodules or functions.	Program is organized by having number of classes and objects.
Function call is used.	Message passing is used.
Software reuse is not possible.	Reusability is possible.

Structured design programming usually left until end phases.	Object oriented design programming done concurrently with other phases.
Structured Design is more suitable for offshoring.	It is suitable for in-house development.
It shows clear transition from design to implementation.	Not so clear transition from design to implementation.
It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction.	It is suitable for most business applications, game development projects, which are expected to customize or extended.
DFD & E-R diagram model the data.	Class diagram, sequence diagram, state chart diagram, and use cases all contribute.
In this, projects can be managed easily due to clearly identifiable phases.	In this approach, projects can be difficult to manage due to uncertain transitions between phase.

1.5 Applications of Object Oriented Programming (OOP)

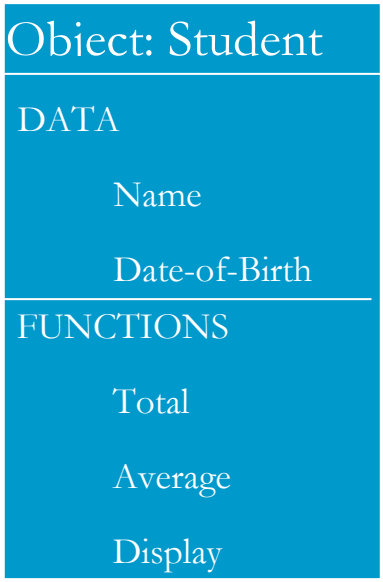
- Real Time Systems
- Simulation and modeling
- Object-oriented databases.
- Hypertext, hypermedia & expertext.
- Al 7 expert systems.
- Neural networks & programming.
- Decision support & office automation systems.
- CIM/CAM/CAD systems.

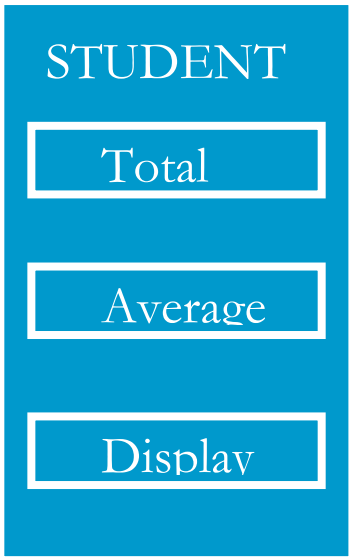
1.6 Basic Concepts of OOP

1) OBJECTS: -

- Objects are the basic run time entities in an object oriented system
- They may represent a person a person, a bank account, a table of data or any item that the program has to handle .
- Programming problem is analyzed in terms of objects and nature of communication between them

- Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in C
- When a program is executed the objects interact by sending messages to one another. Ex: "Customer" and "Account" objects
- Each object contains data and code to manipulate the data





2) CLASSES: -

- The entire set of data and code of an object can be made a user defined data type with the help of a class
- Thus, objects are variable of type class
- Once a class has been defined we can create any number of objects belonging to that class thus, a class is a collection of objects of similar type
- Each object is associate with the data of the class for which that object is created
- Classes are user defined data type and behave like built-in-types of a programming language

- Syntax for class : -

```
Class class-name
{
        Data members;
        member functions;
}
```

Syntax for Object :-

```
        Class-name   Object-name;
```

Ex: -

```
Class Student
{
Int rno, marks;
Void total( );
Void display( );
};
Student s1;
```

3) OOP-Data Abstraction and Encapsulation

- The wrapping of data and functions into a single unit is known as encapsulation.
- The data is not accessible to outside world only those functions which are wrapped in class can access it.
- This concept is known as Data hiding or information hiding
- Abstraction refers to the act of representing essential features without including the background details or explanations
- Classes encapsulate all the essential properties of the objects that are to be created.
- Since the classes use the concept of data abstraction, they are known as Abstract Data Type (ADT).

4) Inheritance: -

- Inheritance is the process by which object of one class acquire the properties of objects of another class.
- Each derived class share some common characteristics with the class from it is derived
- The inheritance provides the idea of Reusability
- We can add additional features to an existing class without modifying it, This is possible by deriving a new class from the existing one
- The new (Derived) classes have the feature of both the classes.

5) Polymorphism: -

- is the ability to take more than one form
- An operation may exhibit different behaviors in different instances
- Addition operation for two nos it will generate sum and also if operands are strings then generate third string by concatenation
- Thus, the process of making an operator to exhibit different behaviors in different instances is known as operator overloading

- Using single function name to perform different types of tasks is known as function overloading
- Polymorphism is extensively used in implementing inheritance

6) Dynamic Binding: -

- is the linking of the code to be executed in response to the call
- Dynamic Binding (Late Binding) means that the code associated with a given procedure call is not known until the time of the call at run time
- It is associated with polymorphism and inheritance

7) Message Passing:-

- Objects can communicate with each other
- - Steps: creating class that defines object and their behavior
- Creating objects from class definition, and Establishing communication among objects
- Objects communicate with each other by sending and receiving information
- A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result
- Message passing involves specifying the name of the object, the name of the function (message) and the information to be sent
- Ex: employee.salary (name);

1.7 A Structure of C++ Program and Simple C++ Program

What is C++: -

- C++ is Object-Oriented programming language. It was developed by BAJARNE STROUSTRUP at AT&T Bell Laboratories in USA early 1980's.
- C++ is an extension of C with a major addition of the class construct feature
- C++ = C+1 , C++ is a superset of C so all C programs are also C++ programs
- The object oriented features in C++ allow programmers to buile large programs with clearity, extensibility and ease of maintenance, incorporating the spirit and efficiency.

A structure of C++ Program: -

The general structure of C++ program with classes is shown as:

1. Documentation Section

2. Preprocessor Directives or Compiler Directives Section

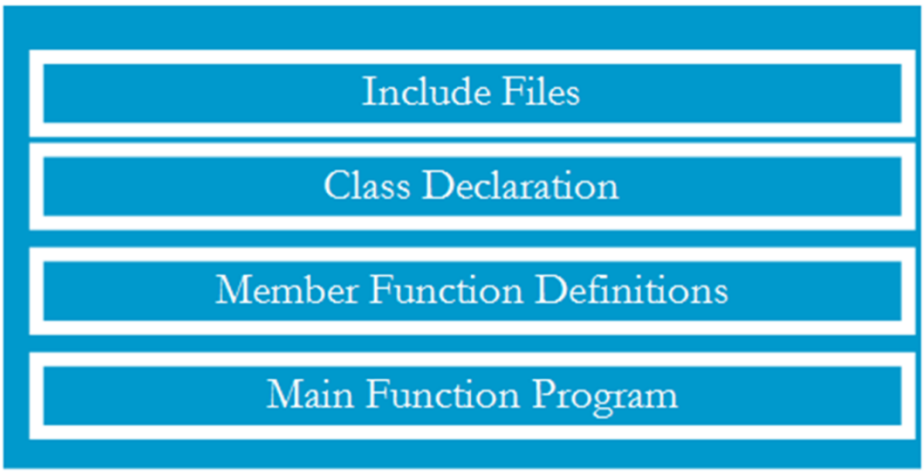
(i) Link Section

(ii) Definition Section

3. Global Declaration Section

4. Class declaration or definition

5. Main C++ program function called main ( )



A Simple C++ Program: -

```
#include<iostream.h>          // include header file
using_namespace std;
void main()
{
        cout<<"C++ is Better Than C. \n";    // C++ statement
        getch();
}
```

1.8 Basic Concepts of C++

1. Keywords: -

Keywords are words that are a part of the C++ language and should be used only in the context as defined.

Such names cannot be used as user-defined identifiers as each of these keywords has a specific function to perform in C++. Following is the list of keyword, which have been inherited from the C language:

auto	extern	short	break	float	size of
case	for	static	char	goto	struck
continue	if	switch	default	int	type def
do	while	long	union	double	register
unsigned	else	return	const	enum	signed
volatile	void				

The following are the addition keywords, which have been added in C++:

class	friend	virtual	inline	private	public
protected	this	new	delete	operate	

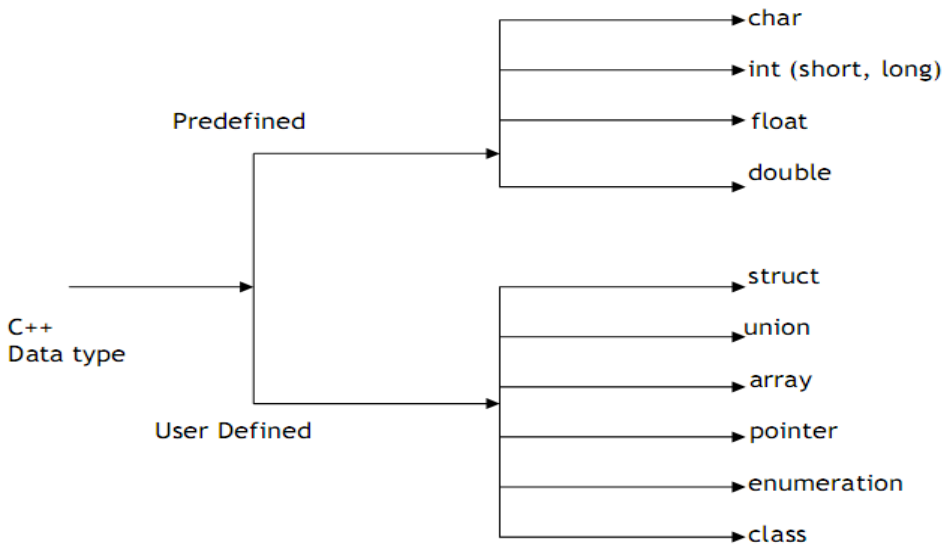
2. Identifiers: -

- The names of variable, functions, tables and various other user-defined references are called identifiers.
- These identifiers consist of one or more characters. It is necessary that the first character of an identifier is an alphabet (A-Z or a-z) or an underscore (_). The subsequent characters can be alphabets, numbers or underscores.
- It is necessary to note the importance of meaningful identifiers in C++ programming. The following factors should be kept in mind while naming identifiers:
- Naming Identifiers : -
    - A standard naming convention should be followed for all the identifiers throughout a program.
    - The identifier name should adequately define the purpose of its existence within the system.
    - It is not advisable to use single character identifier names such as u, I, j etc. for the sake of programming.

3. Data Types in C++: -

The data type is one of the fundamental properties of a variable. It may be considered as the definition of the set of values that a given variable can accept.

C++ data types can be divided into two groups:



Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 4,294,967,295
float	4bytes	3.4 E^{-38} to 3.4 E^{38}
double	8bytes	1.7 E^{-308} 1.7 E^{308}
long double	12bytes	3.4 E^{-4932} to 3.4 E^{4932}

4. Variable Declaration and Initialization:

- Variable is a data name which is used to store some data value or symbolic names for storing program computations and results.
- The value of the variable can be change during the execution. The rule for naming the variables is same as the naming identifier.
- Before used in the program it must be declared.
- **Declaration of variables:**
    - **Syntax: data_type Name_of_variable;**
    - It is declared with the data_type so that compiler will assign memory and also knows what type of data is going to store in it.
    - Ex: int a;

      char c;

      float f;
- **Variable initialization**
    - When we assign any initial value to variable during the declaration, is called initialization of variables. The variable is initialized with the assignment operator such as
    - **Data_type variable_name=value;**
    - Example: int a=20;

      Or
    - int a;

      a=20;

5. Constants in C++ :-

- C++ also has a feature for declaring constants in addition to the standard set of variables.
- A constant is a data set that holds a value, which is set when initialized and which cannot be changed later on anywhere during the course of a program.
- The only difference being that the const keyword has to be added before the data type.
- Constants may be declared as:

const float pi = 3.146 ;

const char three = '3' ;

const int number = 5 ;

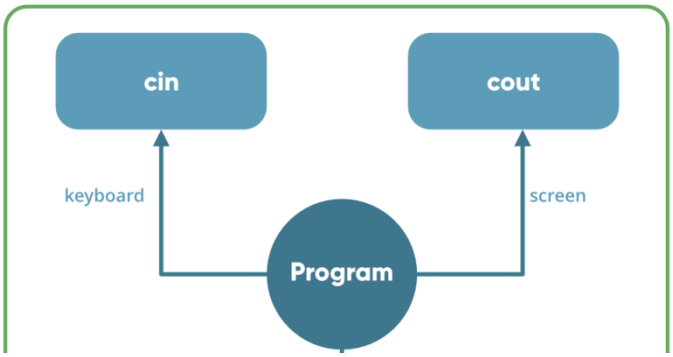
7. Basic Input and Output in C++

C++ comes with libraries that provide us with many ways for performing input and output.

In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams.

Input Stream: If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.

Output Stream: If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.



Header files available in C++ for Input/Output operations are:

iostream: iostream stands for standard input-output stream.

This header file contains definitions to objects like cin, cout, cerr etc.

The two keywords cout and cin in C++ are used very often for printing outputs and taking inputs respectively.

These two are the most basic methods of taking input and printing output in C++.

To use cin and cout in C++ one must include the header file iostream in the program.

cout: - The C++ cout statement is the instance of the ostream class.

It is used to produce output on the standard output device which is usually the display screen.

The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

cin: - cin statement is the instance of the class istream and is used to read input from the standard input device which is usually a keyboard.

The extraction operator(>>) is used along with the object cin for reading inputs.

The extraction operator extracts the data from the object cin which is entered using the keyboard.

Class and Objects

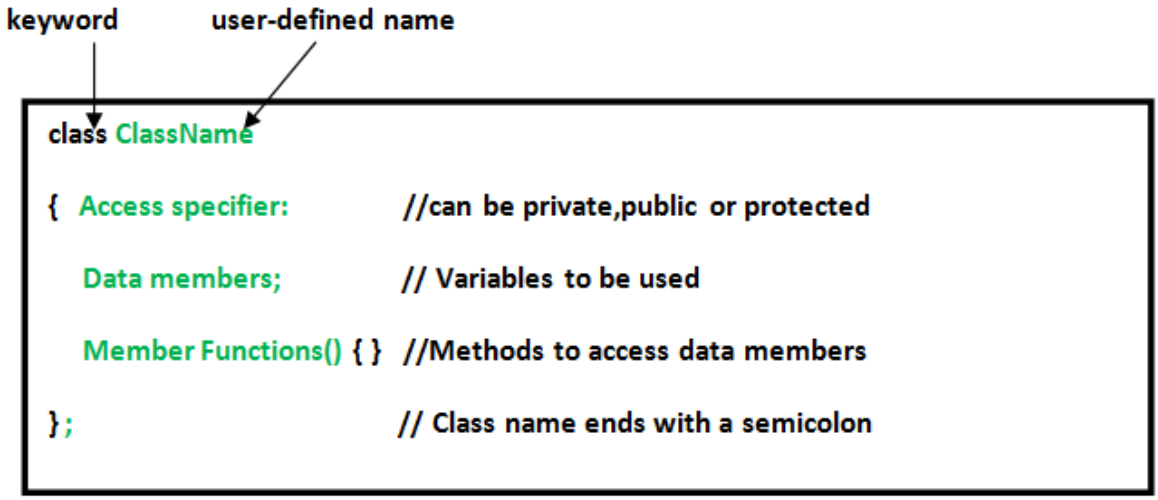
2.1 Class & Objects

1] Class:-

- Class is a syntactic unit which has the data and function.
- Class is a collection of data member and member function.
- Class is users define data type and act as a built-in data type.
- A class in C++ is the building block that leads to Object-Oriented programming.
- **It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance(object) of that class.**
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.
- In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.
- Classes are created using the keyword class.
- A class declaration is syntactically similar to a structure.

Syntax-

```
Class class_name
{
    Access specifier1;
        data member;
        member function( );
    Access specifier2;
        data member;
        member function( );
        _____________
        _____________
        _____________
    Access specifier n;
        data member;
        member function( );
};
```



- As in syntax class is created with the keyword class followed by the class name and data and function are declared within the class{ }

2] Declaring data member:-

- The variable declared inside the class are called as data member of that class.
- **Generally data members are declared inside private section so it is called as private element** and it can't be accessed outside the class.
- You cannot initialize the data member at the time of declaration.

3] Defining member function:-

- Functions that are declared within a class are called as member function. **Member function may access any element of the class in which it is present**.
- Generally member functions are declared inside public section of class to access its private data member.
- There are 2 ways to define a member function:
  - **Inside class definition**
  - **Outside class definition**

Defining member function: Inside class definition

Example:-

```
class Student
{
private:
    int rollno;
    char name[20];
public:
    void accept( )
```

```
    {
        cout<<"Enter roll no. and name,";
        cin>>rollno>>name;
    }
    void display( )
    {
        cout<<"Roll no. ="<<rollno<<endl;
        cout<<"name="<<name<<endl;
    }
};
```

In the above example **rollno and name are the private data members of class.**

The accept( ) and display( ) are the public member function of the class.

The private data member are access through a public member functions.

Defining member function : Outside class definition

A member function of a class can be defined outside the class.

To define a member function outside the class it must be declared inside and **define outside the class definition**, we have to use the scope resolution **::** operator along with class name and function name.

Syntax-

```
    <return type><class_name><scope resolution operator><Fun_name>( )
    {
        Statement;
    }
```

- A member function is defined outside the class by using the scope resolution operator.

- The scope resolution operator tells that the member function is bind to that particular class.
- Thus we can only declare the member function and define it outside the class.

Example: -

```
class Student
{
        private:
            int rollno;
            char name[20];
        public:
            void accept( );
            void display( );
};
void student::accept()
{
        cout<<"Enter roll no. and name,";
        cin>>rollno>>name;
}
void student::display( )
{
        cout<<"Roll no. ="<<rollno<<endl;
        cout<<"name="<<name<<endl;
}
```

As in above example the member function accept and display are declared inside the class student. These member function are defining outside the class.

Note:- The member function inside and outside the class have the same meaning.

4] Access specifier:-

Access specifier specifies the area in which the particular data and function are accessed. There are three types of access specifier in C++ as follows—

1] Private

2] Public

3] Protected

1] Private-

- It is most securing that because only part of that class accesses its data members and functions.
- **The data and functions declared inside private are not accessible outside the class.**
- **By default data and functions declared within a class are private to that class**.
- Thus private access specifier gives the data security feature.

Example:

```
Class student
{
    private:
        int rollno;
        char name[20];
};
```

2] Public:-

- Public access specifier allows their data and function to access everywhere in the program.

- The data and function declared inside the public are accessible within the class and also outside the class.

Example:

```
Class student
{
    public:
        int rollno;
        char name[20];
        void accept( );
        void display( );
};
```

In the above example the data and function are accessible everywhere in the program.

3] Protected:-

- The protected access specifier is needed only when inheritance is used.
- **The data and function declared inside the protected are accessible in the class and also in the derived class only.**
- So protected is used in the inheritance which act as private because protected data and functions are not accessible outside the class but only accessible in the derived class.

```
Class student
{
    protected:
        int rollno;
        char name[20];
        void accept( );
```

```
        void display( );
};
```

5] Object:-

- An Object is an instance of a Class also called as variable of the class.
- When a class is defined, no memory is allocated but when it is created memory is allocated.
- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:-

```
ClassName ObjectName;
```

Object is declared by using the name of the class followed by the name of the object.

Example: Student S1;

- Here student is the class and S1 is the object of class student
- Each object has occupied its own memory space after declaring the object and at runtime. So Object is a basic run time entity of a class.
- You can declare any number of objects for a particular class.
- Each object has its own memory space so any two objects do not share the memory space.

6) Accessing data members and member functions:

Accessing Member Function:

- The data members and member functions of class can be accessed using the dot('.') operator with the object.
- For example if the name of object is S1 and you want to access the member function with the name accept() then you will have to write **S1.accept()** .

Accessing Data Members

- The public data members are also accessed in the same way by using dot(' .') operator however the private data members are not allowed to be accessed directly

by the object. Accessing a data member depends solely on the access control of that data member.

- For example if the name of object is S1 and you want to access the public data member with the name rollno then you will have to write **S1.rollno**.

Example:

```
#include <iostream>

#include <cstring>   //<string.h>

using namespace std;

class student

{

private:

        int rollno;

        char name[20];

public:

        void accept( );

        void display( );

};

void student::accept()

{

cout<<"Enter roll no. and name,";

cin>>rollno;

gets(name);

}

void student::display( )

{
```

```
cout<<"Roll no. ="<<rollno<<endl;

cout<<"name="<<name<<endl;

}

int main( )

{

student S1;

        S1.accept();

        S1.display();

return(0);

}
```

2.2 Static Data Members and Static Member Functions

2.2.1 Static Data Member

- A data member of a class can be declared as static.

Syntax:-

<Static><Data Type><Var_name>;

- Here static is a keyword.

  EX: - static int a;

- Static data member is initializing to zero when the first object of its class is created.
- Only one copy for the static data member is shared by all objects of the class no matter how many objects are created.
- It is visible only within the class but its life time is the entire program
- The type and scope of each static member variable must be defined outside the class definition.

Syntax:-

<Data type><Class name>::<Var_name>;

e.g. - **int item :: a;**

- Here item is a class name and ‘a’ is the static variable.
- This is necessary because the static data member is stored separately rather than as a part of an object. Since they are associated with the class itself not with any class object.
- The static data member stored separately with only one copy and this copy is shared by all the object of that class.
- While defining a static variable some initial value can also be assign to the variable.

  e.g. **int item :: a= 10;**

  Syntax :-

  <Data type><class_name>::<var_name>= <value>

  Here the initial value of the static variable is 10.

2.2.2 Static Member Function

Like static variable we can declare static member function.

Syntax:-

<static><return _value><function_name>( )

{

Statement;

}

- Here static is a keyword to know the complier that it is static member function.
- **A static function can access only static data members declared within the same class.**

- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called by using object and also even if no objects of the class exist, then the static functions are accessed using only the class name and the scope resolution operator :: as follows:

  Class_name :: staticmemberfunction( );

- A static member function can only access static data member, other static member functions and any other functions from outside the class.

Example:

```
#include <iostream.h>
#include <conio.h>
using namespace std;
class Point
{
  static int count;              //Static variable
  public:
  static void increment()     //static member function
  {
      count++;                          //count=count+1
  }
  void display()
  {
    cout<<"The count value is:"<<count<<"\n";
  }
};
```

```
int Point::count = 0;

int main()

{

  Point p1;

  Point p2;

  Point p3;

   p1.increment();                  //point::increment();

   p2.increment();                  //point::increment();

   p3.increment();                  //Point::increment();

        p1.display();

        p2.display();

        p3.display();

  return 0;

}
```

Here all 3 objects share only one copy of static variable so value of count variable is incremented for all objects.

2.3.1 Array of object-

We can declare object of class as an array that is called as array of object.

Array of object is used to store the group of similar data element

Syntax-

<class_name><obj_name>[size];

- Here class_name indicates the name of class whose array of object we have to declare.
- Object_name indicates name of array

- The object array having the same rules of general variable array.
- The object name must follow the rule for declaring variable name.

Size indicate how many elements are present in the array of object. All these elements are of same type that is same class type.

e.g.

```
#include <iostream>
#include <cstring>   //<string.h>
using namespace std;
class student
{
private:
        int rollno;
        char name[20];
public:
        void accept();
        {
                cout<<"Enter roll no. and name,";
                cin>>rollno;
                gets(name);
        }
        void display();
        {
                cout<<"Roll no. ="<<rollno<<endl;
                cout<<"name="<<name<<endl;
```

```
        }
};
int main( )
{
student S1[5];
        for (int i=0;i<=4;i++)
        {
                s1[i].accept();
                s1[i].display();
        }
return(0);
}
```

2.3.2 Object as a function argument –

- It is possible to have function which accepts the object of the class as argument.
- The object can be pass as an argument to the function by value, by reference, by pointer.

e.g.

```
        //Object as a Function Argument
#include <iostream>
using namespace std;
class Distance
{
        float feet;
        float inches;
```

```
        public:
        void get( )
         {
         cout<< "enter feet and inches";
         cin>>feet>>inches;
         }
        void show()
        {
         cout<<feet<<inches;
         }

void add(Distance d1,Distance d2)
         {
          feet= d1.feet+d2.feet;
          inches= d1.inches+d2.inches;
         }
};
int main()
{
        Distance S1,S2,S3;
        S1.get();
        S2.get();
        S3.add(S1,S2);
        S3.show();
```

return(0);

} The above program perform the addition of two distance in feet and inches in main function the statement S3.add(S1,S3); envoke call the member function add of the class distance by the object d3 with the object d1 and d2 as argument.

- It can directly access the variable feet and inches of object d3.
- The member of S1 and S2 are access by using dot operator within the add member function.
- The add function perform addition of two objects S1 and S2 and store it the object S3.
- So S1 and S2 are copied into d1 and d2 which is the argument of the function add and S3 is passed implicitly.
- Which directly access feet and inches from the function add.
- As this is pass by value so any modification made to the data member of the object d1 and d2 are not change the actual data member of the object S1 and S2.

2.3.2 Returning Objects

Syntax:

object = return object_name;

Example: In the above example we can see that the add function does not return any value since its return-type is void.

In the following program the add function returns an object of the Distance class whose value is stored in S3.

When the object S3 calls the add function it passes the other two objects namely S1 & S2 as arguments.

Inside the function, another object is declared which calculates the sum of all the three variables and returns it to S3.

The add function returns an object whose value is stored in another object of the same class Distance S3.

```
#include <iostream>
using namespace std;
class Distance
{
        float feet;
        float inches;
        public:
        void get( )
         {
         cout<< "enter feet and inches";
         cin>>feet>>inches;
         }
        void show()
        {
         cout<<feet<<inches;
         }
Distance add(Distance d1,Distance d2)
         {
          Distance D3;
          D3.feet= d1.feet+d2.feet;
          D3.inches= d1.inches+d2.inches;
          return D3;
         }
};
```

```
int main()
{
    Distance S1,S2,S3;
    S1.get();
    S2.get();
    S3=S3.add(S1,S2);
    S3.show();
    return(0);
}
```

2.4 Constructor-

A constructor is a special member function which is a member of the class.

A constructor has the same name as class name.

Syntax-

```
Class_name( )
{
    Statements;
}
```

- **A constructor does not have any return type.**
- **A constructor is a member function which is executed automatically whenever an object is created.**
- So no need to call the constructor explicitly. **Constructor will call automatically when the object is created**. **This is done by complier implicitly itself.**

Constructor does not return any value because it doesn't have any return type.

e.g.-

```
class count
{
 Int counter;
 Public:
 count( )
 {
        counter++;
  }
 };
```

Calling Constructor-

- Constructor is called automatically whenever object is created.
- e.g. count C1,C2;
- Here constructor is executed twice once for C1 and once for C2 so every time we create an object of this class a constructor is called by default. This is known as default constructor or constructor without argument.
- Note: Every class has its default constructor implicitly. You can call that default constructor in your program. Otherwise complier itself called default constructor implicitly which is hidden from user / programmer for allocate memory to object.

Characteristics of constructor

- Constructor should be declared in public section.
- Constructor called automatically when object is created.
- Constructor cannot return any value even void also, so it cannot have any return type.
- Constructor cannot be virtual.
- Constructor cannot be inherited.
- Constructor can have default arguments.

2.4.1 Type of Constructor

1) Default Constructor

- A Constructor that accept no parameter is called as default Constructor i.e. Constructor without argument.
- If such a Constructor is define in a class then it is automatically called when object is created.
- If no such default Constructor is defined in a class then complier itself supplies a default Constructor of that class which is hidden from user. E.g.

```
Class student
 {
int Rno ;
Public :
Student ( )              // Default Constructor
{
Rno = 0;
}
};
Void main ( )
{
Student s1;              // Default Constructor called.
getch ( );
}
```

- In above program we define a default Constructor student in which we set the default value for roll no.

- At the statement student s1; i.e. object creation the default Constructor will called & it assign initial value to data member Rno.
- Thus we can use default Constructor to set the initial value for data members.

Note: For each class there must be a default Constructor is present either complier or explicitly that is by complier or by user.

2] Parameterized Constructor:-

- We can create a constructor which has the number of arguments.
- This type of constructor is called as parameterized constructor.
- A copy constructor is a member function which initializes an object using another object of the same class

Syntax:-

```
Classname (arg1, arg2, ….., argn)
{
 Statement;
}
```

- Here we can declare the constructor with many number of argument so when we create the object, it should pass these number of argument present in the constructor
- When we create the parameterized constructor a care should be taken that the object must pass the same type and number of arguments to the constructor.
- When we create a parameterized constructor in your program then we cannot create the object without passing any argument.

e.g.

```
Class Count
{
```

```
int counter;
Public:
Count ( )
{
counter=10;
}
Count (float con)
{
Counter= con;
}
Void display ( )
{
Cout<<counter;
}
};
Void main( )
{
Count C1;
Count C2(20.40);
C2.display ( );
C1.display( );
getch( );
}
```

3) Copy Constructor

- Copy Constructor is used to initialize the object by using another object of same class.
- A copy Constructor can accept a reference to its own class as a parameter which is called as a copy Constructor.

Syntax :-

```
Class classname
{
…….
……
Public :
Class name(class name &obj-name)
{
……
…...
}};
```

- A copy Constructor always contains reference of object as argument

e.g.

```
class code
{
int   id;
Public:
Code ( )
{
```

```
}
 Code (int a )
 {
 id=a;
 }
  Code (code & co)
  {
id = co.id
}
  Void display ( )
  {
  Cout <<id;
   }};
  Void main ( )
   {
      Code a (100);
      Code b (a);                    //copy
       Code c;                       //copy
       C=A;
       A .display ( );
        B .display ( );
        C .display ( );
        getch( );
         }
```

- In above example the statement “code B(A);” would define the object B & at the same time initialize it to the values of A. i.e. the object B copy’s the value from object A this process of initializing through a copy Constructor is known as copy initialization.
- The statement “C=A;”will call the copy Constructor but if `C’ & `A’ are the object then this is legal & assigh the values of A to object `C’ by calling copy Constructor.
- This a coy Constructor takes or accept a reference to an object of the same class as the object itself as an argument.

2.5 Constructor with default argument

- We can define a Constructor with default argument i.e. we can provide a default value to the Constructor argument i.e. called as Constructor with default arguments.
- A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values.
- e.g. addition (int a, int b=20)
- Here the Constructor addition has the 2 arguments a&b among which b has the default initial value20.
- When we call that Constructor by supplying actual values.

i.e. addition a1(10,40); Then default value of b is over write with the actual value that is 40.

- If we not specify the value for b. i.e. addition a1 (10); Then it uses the default value of b that is 20.
- When we declare a Constructor which only has all the default argument then it is act as a default Constructor.
- In other case default Constructor in totally different than Constructor with default argument.
- So when default argument Constructor is called with no argument then it becomes default Constructor.

e.g. addition (int a =10,int b=20)

addition a( );

Here that Constructor with default argument is act as a default Constructor which we called with no argument e.g.

```
Class addition
{
int x,y,sum;
Public:
Addition ( int a=10, int b=20)
{
X=a;
Y=b;
}
Void sum ( )
{
Sum = x+y;
cout  < < “sum = “ < < sum;
}
} ;
Void main ( )
{
Addition   a1 (30, 40);
A1.sum ( );
Addition a2 (50);
```

```
A2.sum ( );
Addition a3 ( );
A3.sum ( );
 getch ( );
  }
```

2.5.1 Multiple Constructor (constructor Overloading)

- In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading.
- Overloaded constructors essentially have the same name (name of the class) and **different number of arguments**.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Example:

```
#include <iostream>
using namespace std;
class construct
{
public:
   float area;
   construct()                    // Constructor with no parameters
   {
      area = 0;
   }
   construct(int a, int b)              // Constructor with two parameters
```

```
    {
        area = a * b;
    }
    void disp()
    {
        cout<< area<< endl;
    }
};
int main()
{
    // Constructor Overloading
    // with two different constructors
    // of class name
    construct o1;
    construct o2( 10, 20);
    o1.disp();
    o2.disp();
    return(0);
}
```

2.6 Destructor: -

- Destructor is a special function which is opposite of Constructor.
- In a program we need to perform some action when object is destroyed or goes out of scope then we can use destructor.
- Local object are created when they entered into block & destroyed when the block is left.
- Global the objects are destroyed when the program terminates.
- These when an object is destroyed, destructor has automatically called.

Syntax : -

```
~ class name ( )
{
……
……
}
```

- Destructor has the same of class name precede by tilde operator (~).
- For every object, when it is goes out of scope or destroyed they destructor is called.
- If you not specify the destructor then complier itself provide & call the destructor.
- We can specify the destructor in program to perform some action like closing a file when the object is destroyed.
- The main reason of destructor is to dislocate the memory of the object.

//Destructor Demo

```
#include <iostream>
#include <cstring>   //<string.h>
using namespace std;
```

```
class Count
{
static int counter;
public:
Count()
{
        counter ++;
        cout<<"constructor called\n";
        cout<<"count = "<<counter<<"\n";
}
~Count ( )
{
        counter --;
        cout<<" destrouctor called.\n";
        cout<<" count = "<<counter<<"\n";
}
};
int Count::counter ;
int main()
{
Count c1,c2;

return(0);
}
```

Outout –

Constructor called

count=1

Constructor called

count=2

Destructor called

count=1

destructor called

count=0

- In above program destructor is called twice once for object c1 & once for object c2.
- At the end of program first object c1 is destroyed and its destructor is called that object c2 is destroyed and then its destructor is called.

NOTE: - The objects are destroyed is the same sequence they are created.