UNIT-1

Difference between Class and Structure :-

	Class	Structure
Definition	A class in C++ can be defined as a collection of related variables and functions encapsulated in a single structure.	A structure can be referred to as a user defined data type possessing its own operations.
Keyword for the declaration	Class	Struct
Default access specifier	Private	Public

Example	class myclass
	{
	private:
	int data;
	public:
	myclass(int data_):
	data(data_)
	{}
	virtual void foo()=0;
	virtual ~class()
	{}
		struct myclass
		{
		private:
		int data;
		public:
		myclass(int data_):
		data(data_)
		{}
		virtual void foo()=0;
		virtual ~class()
		{}

	};	};
Purpose	Data abstraction and further inheritance	Generally, grouping of data
Type	Reference	Value
Usage	Generally used for large amounts of data.	Generally used for smaller amounts of data.

Class and Object:-

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.

A class in C++ contains, following properties;

- Data Member
- Method
- Constructor
- Block
- Class and Interface

Object: - It is the physical as well as logical entity whereas **class** is the only logical entity. Object is an instance of class, object has state and behaviours. An Object in C++ has three characteristics:

- State
- Behaviour
- Identity

Syntax

```
class_ name object_reference;
```

Example

```
Employee e;
```

State: Represents data (value) of an object.
Behaviour: Represents the behaviour (functionality) of an object such as deposit, withdraw etc.
Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Class can also be used to achieve user defined data types.

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 behaviours. For example a dog has state:- colour, name, height, age as well as behaviours:- barking, eating, and sleeping.

Vehicle class

Car, bike, truck these all are belongs to vehicle class. These Objects have also different states and behaviours. For Example car has state - colour, name, model, speed, Mileage. as we;; as behaviours - distance travel.

Difference between Class and Object in C++

	Class	Object
1	Class is a container which is collection of variables and methods.	object is a instance of class

2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.

Syntax to declare a Class

Syntax

```
class Class_Name
{
  data member;
  method;
} ;
```

Simple Example of Object and Class

In this example, we have created a Employee class that have two data members eid and ename. We are creating the object of the Employee class and printing the objects value.

Example

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

class Employee
{
public:
int salary;  // data member
void sal()
{
 cout<<"Enter salary: ";
 cin>>salary;
```

```
 cout<<"Salary: "<<salary;
}
};
 void main()
 {
  clrscr();
  Employee e; //creating an object of Employee
  e.sal();
  getch();
 }
```

Output

```
Enter salary: 4500

Salary: 4500
```

❖ Class Access Modifiers :-

The access restriction to the class members is specified by the labelled public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers. The default access for members and classes is private.

```
class Base
 {
public:
// public members go here
protected:
// protected members go here
private:
// private members go here
};
```

1) The public Members :-

A public member is accessible from anywhere outside the class but within a program.

2) The private Members :-

A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

3) The Protected Members :-
A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

Function Overloading :-

Whenever same method name exists multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading or function overloading**.

For example above two methods we can write sum(int, int) and sum(int, int, int) using method overloading concept.

Syntax

```
class  class_Name
{
Returntype  method()
{
.....
}
Returntype  method(datatype1 variable1)
{
.....
}
Returntype  method(datatype1 variable1, datatype2 variable2)
{
.....
```

```
}
};
```

Program Function Overloading in C++

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

class Addition
{
public:
void sum(int a, int b)
{
cout<<a+b;
}
void sum(int a, int b, int c)
{
cout<<a+b+c;
}
};
void main()
{
clrscr();
Addition obj;
obj.sum(10, 20);
obj.sum(10, 20, 30);
}
```

Output

```
30
60
```

Operators Overloading

We can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Box operator+(const Box&);

It declares the addition operator that can be used to add two Box objects and returns final Box object.

Inline Function :-

Inline Function is powerful concept in C++ programming language. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

To make any function inline function just preceded that function with **inline** keyword.

Why use Inline function

Whenever we call any function many time then, it take a lot of extra time in execution of series of instructions such as saving the register, pushing arguments, returning to calling function. For solve this problem in C++ introduce inline function.

Syntax

```
inline function_name()
{
  //function body
}
```

Example

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

inline void show()
{
 cout<<"Hello world";
}

void main()
{
 show();  // Call it like a normal function
 getch();
}
```

Output

```
Hello word
```

Friend Function :-

In C++ a **Friend Function** that is a "friend" of a given class is allowed access to private and protected data in that class.

A function can be made a friend function using keyword friend. Any friend function is preceded with **friend** keyword. The declaration of friend function should be made inside the body of class (can be anywhere inside class either in private or public section) starting with keyword friend.

Syntax

```
class class_name
{
 ......
 friend returntype function_name(arguments);
}
```

Example

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

class employee
{
private:
     friend void sal();
};

void sal()
{
int salary=4000;
cout<<"Salary: "<<salary;
}
void main()
{
sal();
getch();
}
```

Output

```
Salary: 4000
```

Constructor :-

A **Constructor** is a special member method which will be called implicitly (automatically) whenever an object of class is created. In

other words, it is a member function which initializes a class which is called automatically whenever a new instance of a class is created.

- **Types of Constructors :-**

There are three types of constructor :

1. Default Constructor :-

   Default constructor is the constructor which doesn't take any argument. It has no parameter.

2. Parameterized Constructor :-

   These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

3. Copy Constructor :-

   These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

Syntax

```
Class classname
{
----
classname()
{
....
}
};
```

Example of Constructor in C++

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

```
class sum
{
public:
int a,b,c;
sum()
{
a=10;
b=20;
c=a+b;
cout<<"Sum: "<<c;
}
};

void main()
{
sum s;
getch();
}
```

Output

```
Sum: 30
```

Destructor :-

Destructor is a member function which deletes an object. A destructor function is called automatically when the object goes out of scope:

Syntax

```
Class classname
{
~classname()
{
.....
```

```
}
};
```

Example of Destructor in C++

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

class sum
{
public:
int a,b,c;
sum()
{
a=10;
b=20;
c=a+b;
cout<<"Sum: "<<c;
}
~sum()
{
cout<<<<endl;"call destructor";
}
// delay(500);
};

void main()
{
sum s;
cout<<<<endl;"call main";
getch();
}
```

Output

```
Sum: 30

call main

call destructor
```

Virtual Function :-

A **virtual function** is a member function of class that is declared within a base class and re-defined in derived class.

When you want to use same function name in both the base and derived class, then the function in base class is declared as virtual by using the **virtual** keyword and again re-defined this function in derived class without using virtual keyword.

Syntax

```
virtual return_type function_name()
{
 .....
}
```

Virtual Function Example

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

class A
{
public:
 virtual void show()
 {
  cout<<"Hello base class";
 }
};

class B : public A
{
```

```
 public:
  void show()
  {
   cout<<"Hello derive class";
 }
};

void main()
{
clrscr();
A aobj;
B bobj;
aobj.show();  // call base class function

bobj.show();  // call derive class function
getch();
}
```

Output

```
Hello base class

Hello derive class
```

C++ Qualifiers and Storage Classes

Qualifiers and storage class are smaller but important programming concept that helps to make the quality of a variable more accurate for using that variable within the program.

A qualifier is a token added to a variable which adds an extra *quality*, such as specifying volatility or constant-ness to a variable. Qualifiers just specify something about how it may be accessed or where it is stored.
Three qualifiers are there in C++. These are:

- const: This is used to define that the type is constant.

- volatile: This is used to define that the type is volatile.
- mutable: applies to non-static class members of the non-reference non-const type. Mutable members of const classes are modifiable.

Storage Class in C++

Storage classes are used to classify the lifetime and scope of variables. It tells how storage is allocated for variables and how a variable can be treated by the compiler; everything depends on these storage classes. To fully define a variable, storage classes are required apart from the data type. Till now you haven't used any of the storage class, but the default storage class was there playing its role secretly. *Auto* is the default storage class.
These storage classes are divided into four types. They are:

- Auto Variables
- Register Variables
- Static Variables
- Extern Variables

auto

This is the C++'s default storage class you have been using or the compiler is automatically assigning it to the variables. It is applied to local variables and those variables are visible only within the function inside which it is declared and it gets terminated as soon as the function execution gets over. The variables having "auto" as their storage class contains garbage value if they are not assigned any value. The keyword used is "auto".

Example:

```
int var1;        // by default, storage class is auto

auto int var;
```

register

The register storage class is used to classify local variables that gets stored in the CPU register instead of primary memory (RAM) which means that the variable has a maximum size equal to the register size. The storage class register is used only for those variables that require

quick access such as counters. It is not a must that a variable declared with register will always be stored in the CPU register; it means it might get stored in CPU register which fully depends on hardware and implementation restriction. The keyword "register" is used for declaring register variables.

Example of declaration:

Example:

```
register int var2;
```

static

The scope of a static variable is local to the function in which it is defined but it doesn't get terminated when the function execution gets over. In other words, the static storage class tells the compiler to keep the local variable in existence during the lifetime of the program instead of destroying it with the end of function scope. In between the function calls, the value of the static variable persists. By default, the static variable stores zero (0) as its initial value. The static modifier may also be applied to global variables.

Example:

```
static int var3 = 6;
```

extern

Variables having extern storage class have a global scope. Extern variables are used when programmers want a variable to be visible outside the file in which it is declared. Variables with storage class extern can be shared across multiple files. Extern variables do not end until the program execution gets terminated.

Example:

```
extern int var4; // declaration of variable 'var4'

int var4;        // definition of variable 'var4'
```