# UNIT-5

Dr. Shipra Saraswat
UNIT-5 Notes
OOSD

## Private and Public Members

**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.

**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.

## Static data members in C++

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are :-

1) only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
2) It is initialized before any object of this class is being created, even before main starts.

3) It is visible only within the class, but its lifetime is the entire program.

Syntax

```
Static data-type data-member;
```

## Static member Functions in C++

The static member functions are special functions used to access the static data members or other static member functions. A member function is defined using the static keyword. A static member function shares the single copy of the member function to any number of the class objects. We can access the static member function using the class name or class objects. If the static member function accesses any non static data member or non static member function, it throws an error.

Syntax

```
class-name :: function-name (parameter);
```

## Constructor in C++

It is a special method that is invoked automatically at the time of the object creation. It is used to initialize the data members of new objects generally. The Constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e provides data for the object which is why it is known as Constructors.

Constructor does not have a return value, hence they do not have a return type. The prototype of Constructors is as follows:

<class-name> (list-of-parameters);

Constructors can be defined inside or outside the class declaration :-

1) The syntax for defining the constructor within the class :

<class-name> (list-of-parameters) {// Constructor definition}

2) The syntax for defining the constructor outside the class :

<class-name>:: <class-name> (list-of-parameters)
{// Constructor definition }

## Destructor

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

→ Destructor is also a special member function like Constructor. Destructor destroys the class objects created by Constructor.

→ Destructor has the same name as their class name preceded by a tilde (~) symbol.

→ It is not possible to define more than one destructor.

→ The destructor is only one way to destroy the object create by Constructor. Hence destructor can not be overloaded.

→ Destructor neither requires any argument nor returns any value.

→ It is automatically called when object goes out of scope.

→ Destructor release memory space occupied by the objects created by Constructor.

→ In Destructor, objects are destroyed in the reverse of an object creation.

Syntax :-

Syntax for defining the destructor within the class

```
~ <class-name> ()
{
}
```

Syntax for defining the destructor outside the class

```
<class-name> :: ~ <class-name> ()
{
}
```

## Operator Overloading

In C++, we can make operators work for user defined classes. This means C++ has the ability to provide the operators with a special

meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like string so that we can concatenate two strings by just using +.

→ operator overloading is a Compile time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

Example:

```
int a;
float b, Sum;
Sum = a+b;
```

Here variables 'a' and 'b' are of types 'int' and 'float' which are built in data types. Hence the addition operator '+' can easily add the contents of 'a' and 'b'. This is because the addition operator '+' is predefined to add variables of built in data type only.

Write a program of ++ operator (unary operator) overloading

```
// overload ++ when used as prefix
#include <iostream>
using namespace std;
class count {
  private:
  int value;

public:
// Constructor to initialize count to 5
```

```
    Count () : value (5) { }

// overload ++ when used as prefix
  void operator ++ () {
      ++ value;
  }
  void display () {
    Cout << " count : " << value << end l;
    }
};
int main () {
  Count count1;
// Call the " void operator ++ () " function
  ++ Count1;
Count1. display ();
return 0;
}
```

Output
Count : 6

## Overriding

Function overriding in C++ is a feature that allows us to use a function in the child class that is already present in its parent class. The child class inherits all the data members, and the member functions present in the parent class.

write a program to demonstrate function overriding

```
#include <iostream>
using namespace std;
class Base {
  public:
  void print() {
    cout << "Base Function" << endl;
  }
};
class Derived : public Base {
public:
void print() {
    cout << "Derived Function" << endl;
  }
};
int main() {
  Derived derived1;
  derived.print();
  return 0;
}
```

Output

Derived Function

## Virtual Base Class

Virtual base classes are used in virtual inheritance in a way of preventing multiple 'instances' of a given class appearing in an inheritance hierarchy when using multiple inheritances.

the ... at a variable occupies.

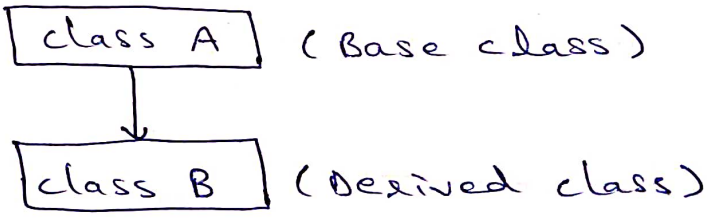
## Inheritance in C++

The capability of a class to derive properties and characteristics from another class is called inheritance. Inheritance is one of the most important features of object oriented programming.

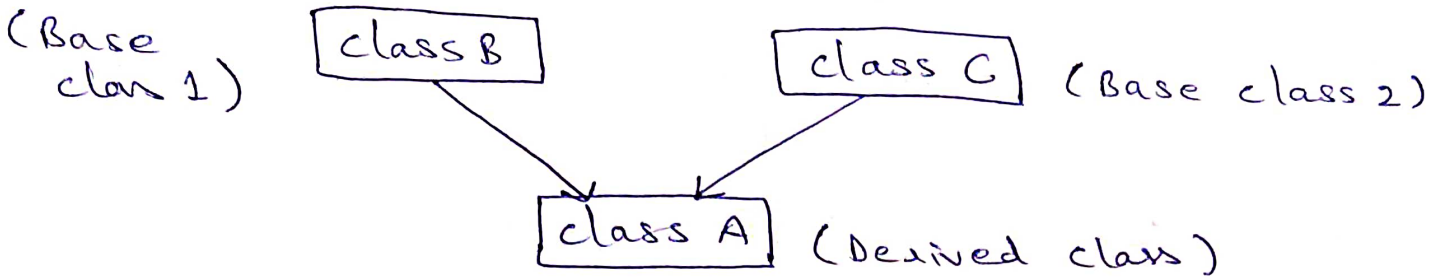
Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called "derived class" or "child class" and the existing class is known as the "base class" or "parent class". The derived class now is said to be inherited from the base class.

## Types of Inheritance

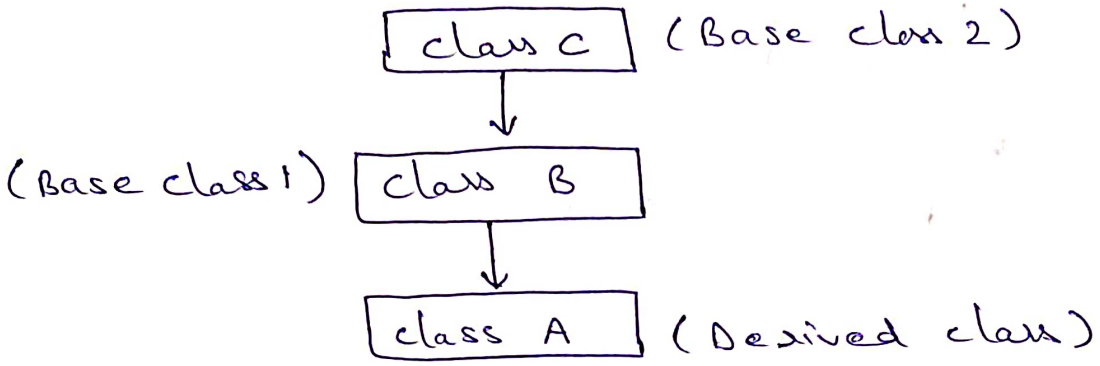
1) **Single Inheritance :-** In single inheritance, a class is allowed to inherit from only one class i.e one subclass is inherited by one base class only.



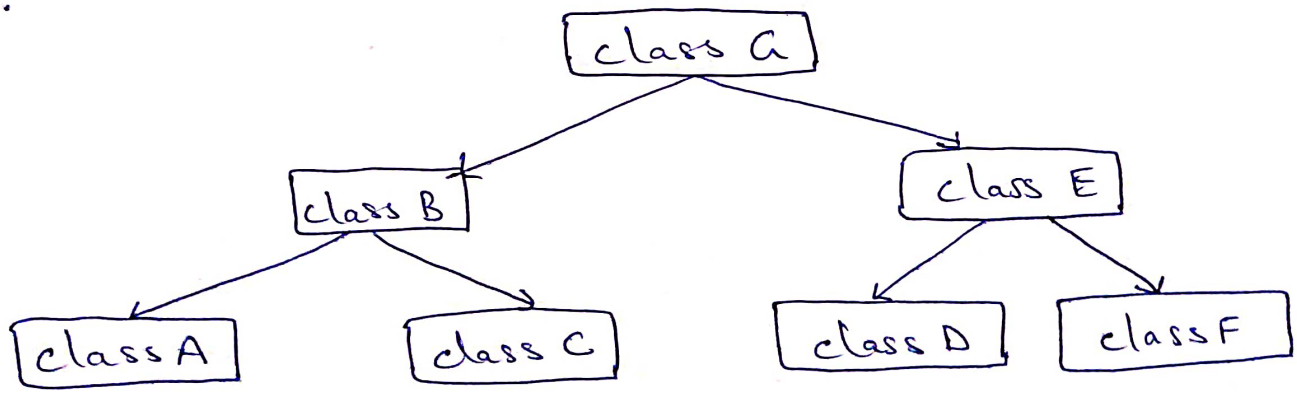

2) **Multiple Inheritance :-** Multiple inheritance is a feature of C++ where a class can inherit from more than one class i.e one subclass is inherited from more than one base class.



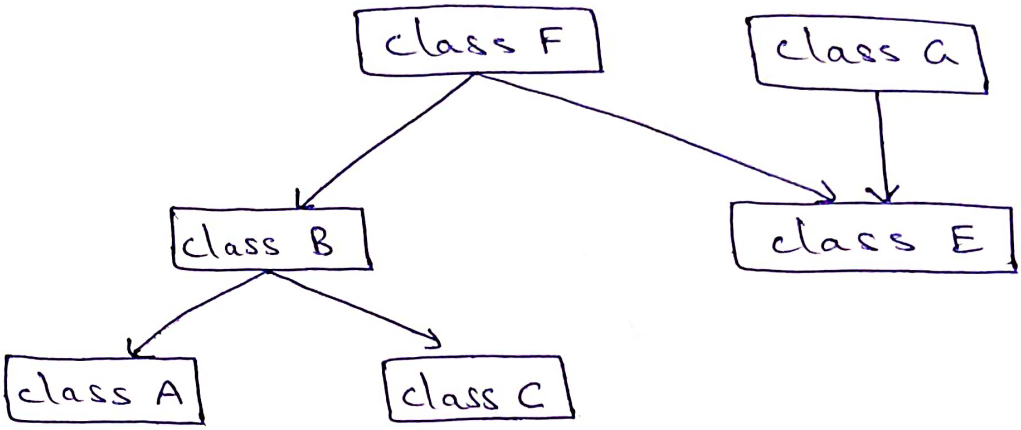

3) Multilevel Inheritance :- In this type of inheritance, a derived class is created from another derived class.




4) Hierarchial Inheritance :- In this type of inheritance, more than one subclass is inherited from a single base class i.e more than one derived class is created from a single base class.



5) Hybrid Virtual Inheritance :- Hybrid inheritance is implemented by combining more than one type of inheritance. For example : Combining hierarchial inheritance and multiple inheritance.




## Pointers in C++

In C++, a pointer refers to a variable that holds the address of another variable. Like regular variables, pointers have a data type. For example, a pointer of type integer can hold the address of a variable of type integer. A pointer of character type can hold the address of a variable of character type.

## Addresses in C++

When you create a variable in your C++ program, it is assigned some space the Computer memory. The value of this variable is stored in the assigned location.

To know the location in the computer memory where the data is stored, C++ provides the & (reference) operator. The operator returns

the address that a variable occupies.

For example, if x is a variable, &x returns the address of the variable.

## "This Pointer"

"This pointer" is a pointer accessible only within the non static member functions of a class, struct, or union type. It points to the object for which the member function is called. Static member functions don't have a this pointer.

→ "This pointer" is passed as a hidden argument to all non static member function calls and is available as a local variable within the body of all non static functions.

→ "This pointer" is not available in static member functions as static member functions can be called without any object.

## Virtual Function Vs Pure Virtual Function

### Virtual Function

1) A virtual function is a member function of base class which can be redefined by derived class.

### Pure Virtual Function

1) A pure virtual function is a membefunction of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.

2) classes having virtual functions are not abstract.

3) Syntax :-

```
virtual <func-type> <func-name> ()
{
  // code
}
```

4) Definition is given in base class.

5) Base class having virtual function can be instantiated i.e its object can be made.

6) If derived class do not redefine virtual function of base class, then it does not affect compilation.

7) All derived class may or may not redefine virtual function of base class.

2) Base class containing pure virtual function becomes abstract.

3) Syntax :-

```
virtual <func-type> <func-name>() = 0;
```

4) No definition is given in base class.

5) Base class having pure virtual function becomes abstract i.e it cannot be instantiated.

6) If derived class do not redefine virtual function of base class, then no compilation error but derived class also becomes abstract just like the base class.

7) All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.