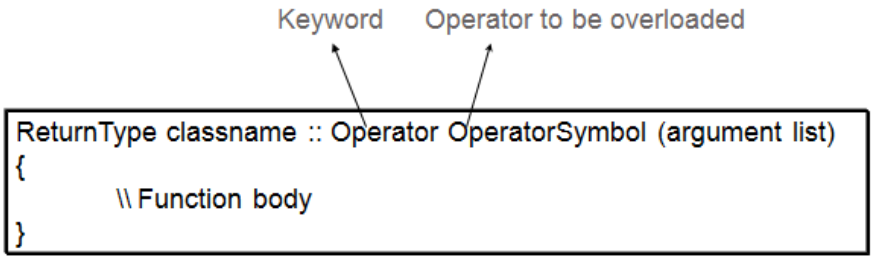
UNIT – 3
Operator overloading and type conversion, Inheritance

Q -1 What is operator overloading?

✓ Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.
✓ You 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 operator is used to perform operation on user-defined data type.

For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.
✓ Operator Overloading Syntax:



✓ Return type specifies the type of value returned by the specified operation.
✓ Operator is the keyword. Operator should be preceded which want to be overloaded.
✓ Operator function must be either member function or friend function.
✓ A difference between them is that a friend function will have only one argument for unary operators and two for binary operators, while member function has no argument for unary operator and one argument for binary operator.

- ✓ Arguments may be passed either by value or by reference.
- ✓ The operator can be classified into unary and binary operators based on number of arguments on which they operate.
- ✓ C++ allow almost all operators to be overloaded but at least one operand must be an instance of a class(object).

Q-2 Give the rules for operator overloading.

1) Overloaded operators must perform operation similar to those defined for basic data types. Meaning of operator can not be change.
2) Only existing operator can be overloaded. New operator can not be created like $.
3) At least one operand should be class object.
4) Operator function can be either member function or friend function.
5) In case of member function, it will take no explicit argument and return no explicit values for unary operator. It will take one explicit argument and may be return explicit values for binary argument.
6) In case of friend function, it will take one explicit argument which is the reference of the relevant class for unary operator, it will take two explicit argument for binary operator.
7) When binary operators overloaded, through a member function, the left hand operand must be an object of the relevant class.
8) There are some operators cannot be overloaded. They are listed as follow:
   . member accessing operator.
   .* pointer to member operator
   :: scope resolution operator
   ?: conditional operator
   Sizeof() sizeof operator
9) Some of the operators can not be overloaded by friend function but member function can overloaded it. They are listed as follows:
   = assignment operator
   ( ) function call operator
   [ ] subscripting operator
   ➔ Class member access operator
10) Binary arithmetic operators such as +,-,*,/ must explicitly return a value. They must not attempt to change their own arguments.

Q-3 Explain Unary Operator.

- ✓ Overloading without explicit arguments to an operator function is known as unary operator overloading.
- ✓ The unary operators like unary +, unary -, increment and decrement operator.
- ✓ Syntax:

```
Returntype operator OP()
{
    //body of operator function
}
```

- ✓ Where, function return type can be either from following : primitive, void, or user defined operator is the keyword OP specifies the operator symbol to be overloaded.
- ✓ For Ex:

```
Int operator – (); Void
operator ++ ();
```

Q-4 Explain Binary operator.

- ✓ The concept of overloading unary operators applies also to the binary operator.
- ✓ Syntax:

```
Returntype operator OP(arg)
{
    //body of function
}
```

- ✓ Here the function takes the first object as an implicit operand and the second operand must pass explicitly.
- ✓ The data members of first object are passed without using dot operator while second argument members can be accessed using the object and dot operator.
- ✓ For Ex:

```
Void operator / (data d1)
```

Q-5 Explain operator overloading with friend function.

- ✓ The operator function can also be used to overload operator.
- ✓ It provides a flexibility to use object of different class.
- ✓ The difference between a friend function and member function is that friend function require the arguments to be passed explicitly to the operator function while member function consider the first argument implicitly.
- ✓ Friend function can be used for both unary and binary operators.
- ✓ Syntax:

```
Friend returntype operator OP(arg1, arg2)
{
     //body of function
}
```

- ✓ According to above syntax the friend function must be prefixed with keyword friend.
- ✓ The body of the friend function can be either inside class or outside the class.
- ✓ When the definition of friend function is outside class then it defined as normal function and not prefixed with the friend keyword.
- ✓ It can return and data type wither or following: void, primitive or user defined.
- ✓ OP is the operator symbol to be overloaded and prefixed with operator keyword.
- ✓ For Ex:

```
Friend data operator +  (data d1,data d2);
```

Q-6 Explain this pointer (*this).

- ✓ Normally the member function of a class is invoked, by some object of the class.
- ✓ C++ has keyword this pointer to represent an object that invokes a member function. Thus, member function of every object has access to a pointer named this, which points to the object itself. It represented as *this.
- ✓ This pointer can be treated like any other pointer to an object.
- ✓ Using this pointer, any member function can find out the address of the object of which it is a member.

Q-7 Explain Type conversion.

- ✓ The assignment operator assigns the contents of a variable, the results of an expression, or a constant to other variable.
- ✓ There are three types of data conversion

1) Basic type to class type
2) Class type to basic type
3) One type class to another class type

1) Basic type to class type

- ✓ To convert data from a basic type to user defined type.
- ✓ The conversion function should be defined in user defined object class in the form of constructor.
- ✓ The constructor can take a single argument whose type is to be converted.
- ✓ The syntax:

```
Constructor (basic type)
{
//body for conversion
}
```

- ✓ Where constructor specifies the same name as class in which it belongs.
- ✓ It has only one argument of basic type.

2) Basic type to class type

- ✓ Constructor function does not support the operation of class to basic type.
- ✓ In case of user defined data type conversion.
- ✓ The operator function is defined as an overloaded basic data type which has no any arguments.
- ✓ It returns converted data member object to basic data type.
- ✓ Syntax:

```
Operator basic type()
{
    //code for conversion
}
```

- ✓ According to above syntax the operator is the keyword.
- ✓ Basic type specifies any primitive data type such as int, char, float double etc.

3) One class to another class type

- ✓ The c++ compiler does not support data conversion between object to two user define classes.
- ✓ However the conversion of class type to basic type and basic type to class type can be performed either by operator conversion function or constructor.
- ✓ Ex: Classa obja;

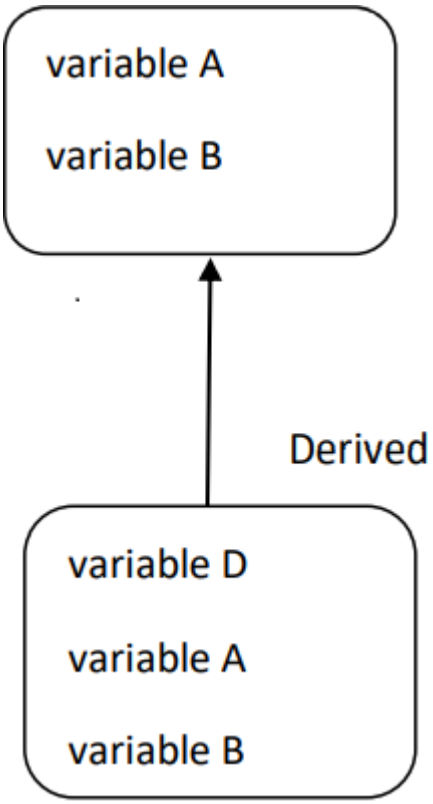
Classb objb;

Obja=objb;

- ✓ Where obja is the object of classa and objb is the object classb.
- ✓ The class classa type is converted to the class classb and converted value is assigned to the obja.

Q-8 What is inheritance?

- ✓ Inheritance is the process of creating new classes, called derived classes, from existing classes, which are often called base classes.
- ✓ The derived class inherits all the capabilities of the base class and it can also add new features of its own. But here base class remain unchanged.



- ✓ As shown in above figure the base class has there features (A,B,C). The derived class has three features of base class and adds its own features (D). The arrow represents the direction of derivation.
- ✓ Here, its direction from the derived class towards the base class. It represents that derived class access features of the base class not vice-versa.
- ✓ This inheritance mode is used mostly. In this the protected member of Base class becomes protected members of Derived class and public becomes public.
- ✓ class derivedclass: public baseclass

Accessing Base class members	public	protected	private
From Base class	Yes	Yes	Yes
From object of a Base class	Yes	No	No
From Derived classes	Yes (As Public)	Yes (As Protected)	No
From object of a Derived class	Yes	No	No
From Derived class of Derived Classes	Yes (As Public)	Yes (As Protected)	No

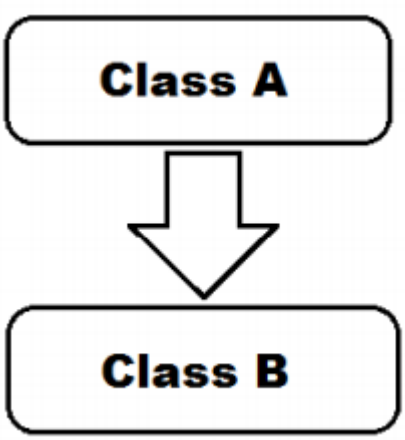
- ✓ Derived class of Derived Classes: If we are inheriting a derived class using a public inheritance as shown below
- ✓ class B : public A
- ✓ class C : public B
- ✓ then public and protected members of class A will be accessible in class C as public and protected respectively.
- ✓ There are different types of inheritance:

1) Single Inheritance
2) Multiple Inheritance
3) Multilevel Inheritance
4) Hierarchical Inheritance
5) Hybrid (Virtual) Inheritance

1) Single Inheritance :

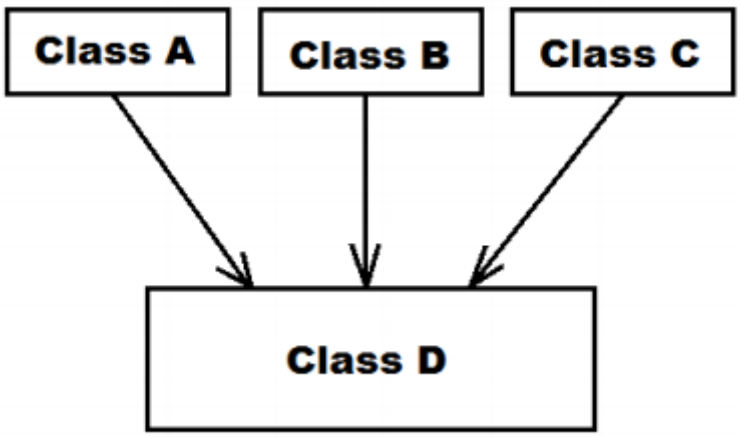
- ✓ Single inheritance represents a form of inheritance when there is only one base class and one derived class.

- ✓ For example, a class describes a Person:



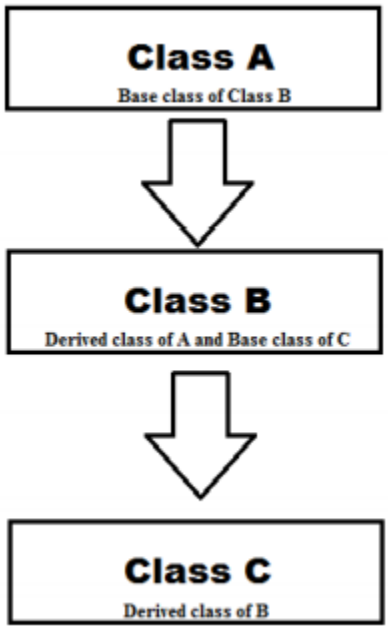
2. Multiple Inheritance

- ✓ Multiple inheritance represents a kind of inheritance when a derived class inherits properties of multiple classes. For example, there are three classes A, B and C and derived class is D as shown below:



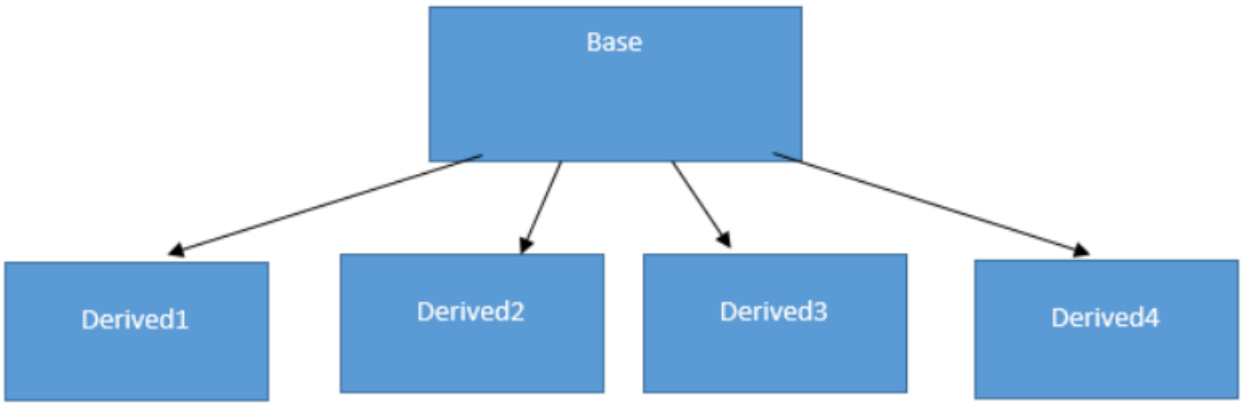
3. Multilevel Inheritance

- ✓ Multilevel inheritance represents a type of inheritance when a Derived class is a base class for another class.
- ✓ In other words, deriving a class from a derived class is known as multi-level inheritance.
- ✓ Simple multi-level inheritance is shown in below image where Class A is a parent of class B and Class B is a parent of Class C.



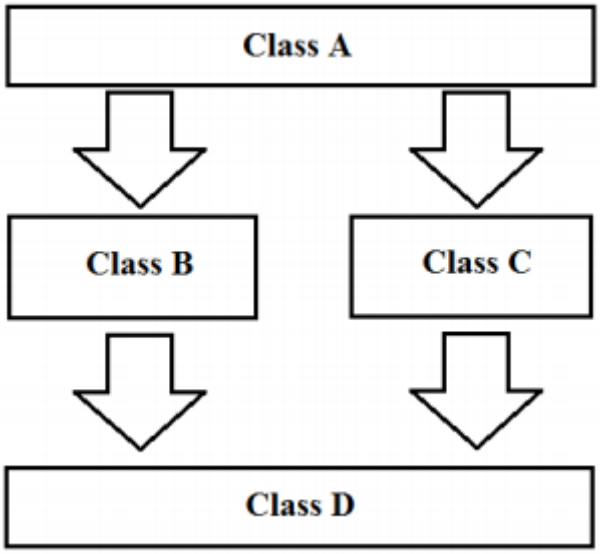
4. Hierarchical Inheritance :

- ✓ When there is a need to create multiple Derived classes that inherit properties of the same Base class is known as Hierarchical inheritance.



5.Hybrid Inheritance

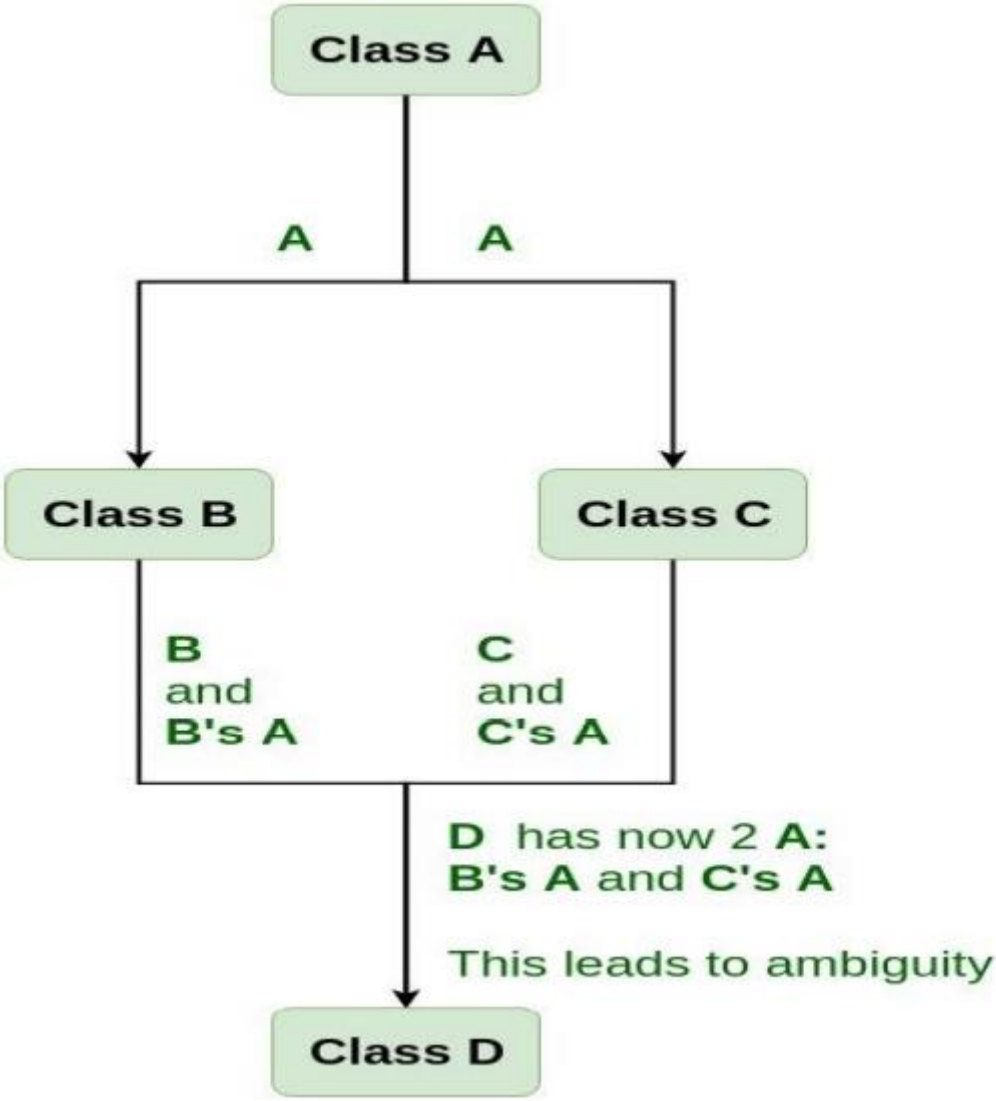
✓ Combination of Multi-level and Hierarchical inheritance will give you Hybrid inheritance.



- ✓ Virtual Inheritance
- ✓ We can avoid Diamond problem easily with Virtual Inheritance. Child classes in this case should inherit Grandparent class by using virtual inheritance:
- ✓ Constructor in derived class
- ✓ When a default or parameterized constructor of a derived class is called, the default constructor of a base class is called automatically.
- ✓ As you create an object of a derived class, first the default constructor of a base class is called after that constructor of a derived class is called.
- ✓ When multiple inheritance is used, default constructors of base classes are called in the order as they are in inheritance list. For example, when a constructor of derived class is called:
- ✓ Destructor in derived class.
- ✓ Deleting a derived class object using a pointer to a base class that has a non-virtual destructor results in undefined behaviour.
- ✓ To correct this situation, the base class should be defined with a virtual destructor.
- ✓ For example, following program results in undefined behaviour.
- ✓ Making base class destructor virtual guarantees that the object of derived class is destructed properly, i.e., both base class and derived class destructors are called.

Topic: 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.
- ✓ Need for Virtual Base Classes:
- ✓ Consider the situation where we have one class A .This class is A is inherited by two other classes B and C. Both these class are inherited into another in a new class D as shown in figure below.



- ✓ As we can see from the figure that data members/function of class A are inherited twice to class D.
- ✓ One through class B and second through class C. When any data / function member of class A is accessed by an object of class D, ambiguity arises as to which data/function member would be called?
- ✓ One inherited through B or the other inherited through C. This confuses compiler and it displays error.

Example: To show the need of Virtual Base Class in C++

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


class A
{
    public:
    void show()
    {
        cout << "Hello form A \n";
    }
};
class B : public A
{
};

class C : public A
{
};
```

```
class D : public B, public C
{

};

int main()
{
        D object;
        object.show();
}
```

Compile Errors:

```
prog.cpp: In function 'int main()':
prog.cpp:29:9: error: request for member 'show' is ambiguous
 object.show();
         ^
 prog.cpp:8:8: note: candidates are: void A::show()
 void show()
         ^
 prog.cpp:8:8: note:                 void A::show()
```

- ✓ How to resolve this issue?
  To resolve this ambiguity when class A is inherited in both class B and class C, it is declared as virtual base class by placing a keyword virtual as :
- ✓ Syntax for Virtual Base Classes:

Syntax 1:

```
class B : virtual public A
{

};
```

Syntax 2:

```
class C : public virtual A
{
};
```

- ✓ Note: virtual can be written before or after the public. Now only one copy of data/function member will be copied to class C and class B and class A becomes the virtual base class.
- ✓ Virtual base classes offer a way to save space and avoid ambiguities in class hierarchies that use multiple inheritances.
- ✓ When a base class is specified as a virtual base, it can act as an indirect base more than once without duplication of its data members.
- ✓ A single copy of its data members is shared by all the base classes that use virtual base.

Topic: Abstract Classes

- ✓ Abstract classes act as expressions of general concepts from which more specific classes can be derived.
- ✓ You cannot create an object of an abstract class type; however, you can use pointers and references to abstract class types.
- ✓ A class that contains at least one pure virtual function is considered an abstract class.
- ✓ Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes.
- ✓ Consider the example presented in Virtual Functions. The intent of class Account is to provide general functionality, but objects of type Account are too general to be useful.
- ✓ Therefore, Account is a good candidate for an abstract class:

```
// deriv_AbstractClasses.cpp
// compile with: /LD
class Account {
 public:
        Account( double d );   // Constructor.
        virtual double GetBalance();   // Obtain balance.
        virtual void PrintBalance() = 0;   // Pure virtual function.
private:
        double _balance;
}
```

- ✓ The only difference between this declaration and the previous one is that Print Balance is declared with the pure specifier (= 0).

Topic: Constructor in derived class

- ✓ The derived class need not have a constructor as long as the base class has no-argument constructor.
- ✓ However, if the base class has constructors with arguments (one or more), then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor.
- ✓ When an object of a base class is created, the constructor of the base class is executed first and later the constructor of the derived class.

3. Constructor only in the derived class

```
 class A
{
public:
};
class B:public A
{
        public:
B()
{
cout<<"No-argument constructor of the base class B is executed";
}
        };
        void main()
        {
        B ob;    //accesses derived constructor
        }
```

Topic: Containership

- ✓ We can create an object of one class into another and that object will be a member of the class.
- ✓ This type of relationship between classes is known as containership or has_a relationship as one class contain the object of another class.
- ✓ And the class which contains the object and members of another class in this kind of relationship is called a container class.
- ✓ The object that is part of another object is called contained object, whereas object that contains another object as its part or attribute is called container object.
- ✓ **Difference between containership and inheritance**
- ✓ **Containership**

➔ When features of existing class are wanted inside your new class, but, not its interface

- ✓ for eg. ->
    1. computer system has a hard disk.
    2. car has an Engine, chassis, steering wheels.
- ✓ **Inheritance**

➔ When you want to force the new type to be the same type as the base class.

- ✓ for eg. ->
    1. computer system is an electronic device
    2. Car is a vehicle