#### MCA 205 Object Oriented Programming with C++Module 4

#### Inheritance

#### Introduction

Inheritance is the process by which new classes called derived classes are created from existing classes called base classes. The derived classes have all the features of the base class and the user can add new features specific to the newly created derived class.

For example, a programmer can create a *base* class named fruitand define *derived* classes as mango,orange,banana,etc. Each of these derived classes, (mango,orange,banana, etc.) has all the features of the *base* class (fruit) with additional attributes or features specific to these newly created derived classes.

**Features or Advantages of Inheritance:**

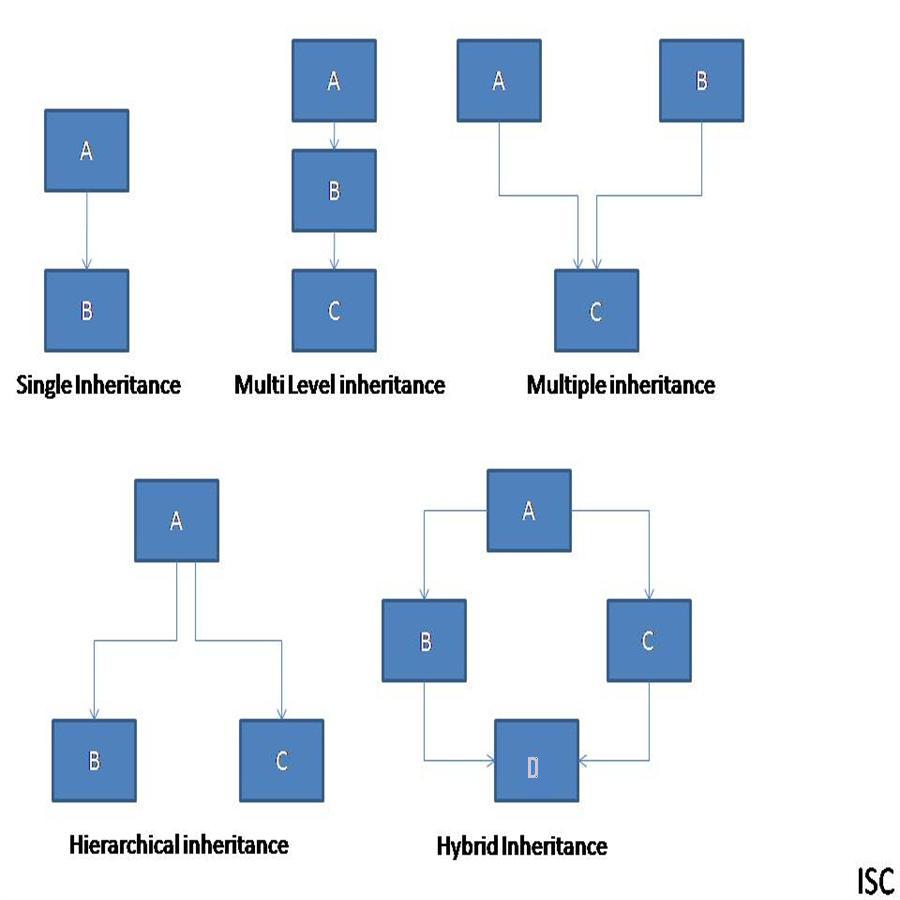
* **Reusability:**

Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it need not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

* **Saves Time and Effort:**

The above concept of reusability achieved by inheritance saves the programmer time and effort. Since the main code written can be reused in various situations as needed.

**Types of Inheritance**



To inherit the class from another class we need 2 class one class is called the Base class and another class is called the Derived class. Base class also called the super class and derived class also called sub class.

1. **Single Inheritance:**

Derivation of class from only one base class is called single inheritance.

1. **Multiple Inheritance**

Derivation of class from several base classes is called multiple inheritance.

1. **Hierarchical Inheritance:**

Derivation of several classes from a single class is called hierarchical inheritance.

1. **Multilevel Inheritance:**

Derivation of a class from another derived class is called multiple inheritance.

1. **Hybrid Inheritance:**

Derivation of a class involving more then one form of inheritance is called hybrid inheritance.

**Protected Member**

A member that is declared as protected is accessible by the member functions with in the class and any class immediately deriving from it. It cannot be accessed by the functions outside of these 2 classes.

#### General Format for implementing the concept of Inheritance:

class derived\_classname: visibility mode base class name

{

}

Public, private and protected are the visibility modes

There are mainly three visibility modes Public, Private, Protected. The default visibility mode is private.

* **Publicly Inherited:** When a Base Class is publicly Inherited by a derived class
  + - Private Members are not Inherited
    - Public members of the base class become Public members of the derived class and they are accessible to the objects of the derived class.
    - Protected members of the base class become protected members of the derived class and they are accessible by the member functions of the derived class and is ready for further inheritance.
* **Privately Inherited:** When a Base Class is privately Inherited by a derived class
  + - Private Members are not Inherited
    - Public members of the base class become Private members of the derived class and they are not accessible to the objects of the derived class. They can be only accessed by the member functions of the derived class.
    - Protected members of the base class become private members of the derived class and they are accessible by the member functions of the derived class, it is not available for further inheritance.
* **Protectely Inherited:** When a Base Class is Protectely Inherited by a derived class
  + - Private Members are not Inherited
    - Public members of the base class become protected members of the derived class and they are accessible by the member functions of the derived class and are ready for further inheritance.
    - Protected members of the base class become protected members of the derived class and they are accessible by the member functions of the derived class and are ready for further inheritance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Base Class Visibility** | **Derived Class Visibility** | | |
| **Public derivation** | **Private derivation** | **Protected derivation** |
| **Public Members** | Public | Private | Protected |
| **Private Members** | Not inherited | Not inherited | Not inherited |
| **Protected Members** | Protected | Private | Protected |

**Virtual base class**

Multipath inheritance introduces ambiguity problem because of derived child class inherits properties from the same base class multiple times.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class(ancestor class) as virtual base class while declaring direct or intermediate base classes as shown below.

Class A

{

……..

};

Class B1 : virtual public A

{

………..

};

Class B2 : public virtual A

{

………….

};

Class C : public B1, public B2

{

……..

};

When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how may inheritance paths exist between the virtual base class and a derived class.

The key words Virtual and public can be used in either order.

Virtual Public.

Public Virtual.

**Abstract classes**

An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes)

Eg.

class M

{

public:

void dis (void)

{

cout<<“class m”;

}

};

class N : public m

{

public:

void display(void)

{

dis();

cout<<“class N”;

}

};

void main()

{

N n;

n.display();

};

**Constructors in the Derived Class**

If a Base class contains a constructor with one or more arguments then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor. When both the derived and base class contains constructors, the base constructor is executed first and then the constructor in the derived class is executed.

In case of multiple inheritances, the base class is constructed in the same order in which they appear in the declaration of the derived class.

Similarly, in a multilevel inheritance, the constructor will be executed in the order of inheritance.

The general format of defining a derived constructor is

DerivedClassCon ( arglist1, arglist2,…….arglist n) : Base1(arglist1), Base2(arglist2),…Basen(arglist3)

{

DerivedClass constructor body

}

The header line of the derived-constructor function contains two parts separated by a colon (:). The first part provides the declaration of the arguments that are passed to the derived class constructor and the second part lists the function calls to the base class.

**Example:**

D(int a1, int a2,int b1,int b2, int d1):A(a1, a2), B(b1,b2)

{

d=d1;

}

A(a1, a2) invokes the base constructor A() and B(b1, b2) invokes another base class constructor B().

The constructor D() supply the values i.e. a1, a2, b1, b2 (to A() and B()) and to one of its own variables d1.

Hence, the assignment of the values will be as follows When an object of D is created,

D object-name(5, 12, 7, 8, 30);

**Nesting of Classes- Member Classes**

It’s a method of inheriting properties of one class into another. In this method an object is a collection of objects of other classes. That is a class can contain objects of other classes as its members.

Example:

Class A

{

:::

}

Class B

{

A obj1 // obj1 is an object of class A

};

All the objects of Class B will contain the object obj1. This kind of relationship is known as container relationship or nesting.

In Nesting constructors of all the member objects should be called before its own constructor body is executed. This is accomplished by the initialization list in the constructor of the nested class.

Example:

#include<iostream.h>

class base

{

int a;

public:

base(int x)

{

a=x;

}

void basedis()

{

cout<<a;

}

};

class sub

{

int b;

base obj1; // obj1 is an object of class base

public:

sub(int y,int z):obj1(y) // obj1(y) is the argument list for the constructor of obj1.

{

b=z;

}

void subdis()

{

obj1.basedis();

cout<<b;

}

};

int main()

{

sub obj2(5,6);

obj2.subdis();

return(0);

}