Pros and Cons of Inheritance V.pravallika

The advantages of Inheritance are as follows

(1) whan a class is derived from another class the code of base class can be directly used by the derived class. The derived class need not rewrite the code that has already been written. This leads to code reusability.

(2) when a class is derived from more than one class all derived classes have similar properties to those of base classes

(3). The derived classes extend the properties of base classes to generate more dominant objects.

(4) The same base classes can be cued by a number of derived classes in class hierarchy.

(5) Inheritance helps programmers to avoid code redundancy

(6). As exiting code is reued it leads to less development and maintainance cost

Disadvantages of Inheritance!

(1) overuse or Improper use of Inheritance may lead to poorly designed or even wrong solutions

(2) often data members defined in base class are not used in derived class. This affects program's performance as unnecessarily memory is allocated to them

(3) In inheritance base and derived classes are tightly coupled. Therefore any change in base class affect all the derived classes

(4) A program requires more memory space to store its own code and code of others classes that it is using.

OBJECT as a class member!-

properties of one class can be used in another class using Inheritance or using the object of a class as a member in another class. Declaring object as a class data member in another class is also known as delegation. when a class has an object

: of another class as its member such a class is

ad known as container class.

In Inhertance the derived class can use members of base class. Here derived class is a kind of base class. The programmers can also add new members to derived class..

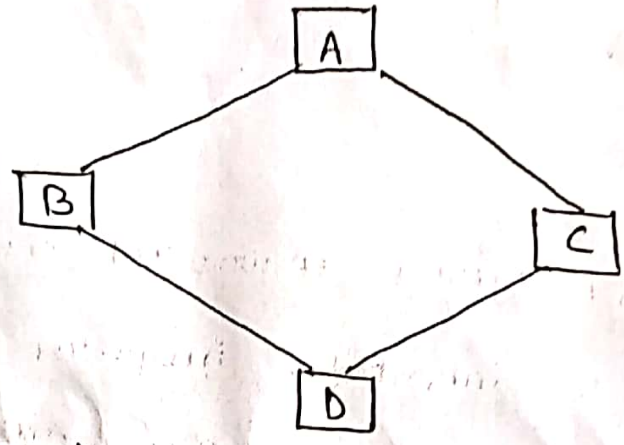
In delegation the class consists of objects from other classes. The composed class uses properties of other class through objects. This kind of relationship is known as has-a relationship or containership

```
class Birth
 {
        public:
           int dd, mm, yy;
           ~~Birth()~~
           void show()
              {
                 cout<< "Enter day, month, year";
                 cin>>dd>>mm>>yy;
                 cout<<dd<<"-"<<mm<<"-"<<yy;
              }
      };
```

```
class    Student
 {      Public:
        char   name[20];

        Birth   dob;

        char   gender;

       void    print()
          {
               cout<< " Enter name and gender"<<endl;

               cin>> name >> gender ;

               cout<< "Name =" << name <<endl;
               cout<< "gender =" << gender<< endl ;
               cout<< "Date of birth =" ;<<
                 dob. show();
          }

  };

  int    main()
       {
            student  s;
                s. print();
       }
```

multipath Inheritance:-

Deriving a class from other derived classes that are in turn derived from the same base class is called multipath Inheritance. In other words, its an combination of more than one Inheritance (multilevel, Hierarchial, multiple Inheritance)



a) multipath Inheritance

when derived class inherits the members of the base class twice through class B and class C This results in ambiguity because a duplicate set of members is created.

The solution to the problem of ambiguity in multipath Inheritance is avoided by making the common base class or grand parent class into virtual base class. This is done by using the keyword virtual while deriving the base class

Syntax!

```
class   derived_class : virtual public baseclass
     {
        - - - -
        - - - -
     };

                    (or)

class   derived_class :  public virtual base_class
     {  - - - -
        - - - -
        - - - -
     };
```

The keyword virtual ensures that only one copy of the base class is inherited, irrespective of the number of the inheritance paths that exist between the virtual base class and derived class.

Note:- The order of keywords virtual and public is interchangeable while defining parent derived classes B and C.

Example!

```
#include <iostream>
 using namespace std;
  class  A
   {
      public:
        void showA()
          {
             cout<< "Base class A"<<endl;
          }
```

```
class B: public virtual A
 {
    public:
        void showB()
         {
            cout<<" Derived class B" <<endl;
         }
  };

Class C: public virtual A
   {
      public:
        void showC()
        {
          cout<<"Derived class C" << endl;
        }
     };
 Class D: public B, C
    {
       public:
       void showD()
         {
           cout<<" child class D" <<endl;
         }
     };
   int main()
     {
        D d;
        d.showA();
     }
```

Rules for virtual functions:

1. The virtual function Should not be static

2. The virtual function must be member of a class

3. A constructor cannot be declared as virtual, but a destructor can be declared as virtual

4. It is also possible return a return a value from virtual functions similar to other functions

5. Arithematic operations cannot be used with base class pointers

6. If a base class contains virtual function and it same function is not refined in the derived classes in such a case the base class function is invoked

7. The virtual functions should be defined in the public section of the class. It is also possible to define the virtual function outside the class.

8. virtual functions are accessed using pointers.

pointer

C++ uses the "this" keyword to represent the object that invoked the member-function of the class. this pointer stores the address of an object used to call a non-static member function. It should be noted that each non-static member function name must be preceeded with ~~function~~ an object name when calling the function. the address of that object is passed to the function and stored in "this". In order to access the data stored in the object the function uses the object address stored in this. most of the time this pointer is hidden from programmers and processed implicitly by the compiler.

Ex:- consider an object obj 'calling one of its member function say get() as : obj.get(). Then this pointer will hold the address of object obj inside member function get(). this pointer acts as an implicit argument to all the member functions.

mainly this pointer is used to distinguish datamembers from local variables of member functions if they have same name

Program.

```
        Class   Rectangle
          {
             Public:
                 int  l, b;
               Rectangle ( int l, int b)
                 {
                 this → l = L;
                 this → b = b;
                 }
               Void   Area()
                       {
                          cout << "Area =" << l*b;
                       }
          };

           int  main()
             {
                  Rectangle   R( 15, 30);
                     R. Area();
             }
```

Ex-2   program to enter name & age of two persons find elder person using this pointer

```
        Class   name
          {
               char  str[15];
                Int age;
```

```
public:
void   input()
  {
        cout<<"Enter name & Age";
         cin>> name>>age;
  }

   void  show()
     {
        cout<<"Name="<<name<<endl;
        cout<<"Age="<<Age<<endl;
     }

   name     max (name  n)
          {
                if( this->age> n.age)
                     Return *this;
                else
                     return n;
          }
};

int     main()
   {
        name  n,n1,n2;
         n1.input();
         n2.input()
     n= n1.max(n2);
        n.show();
     }
```