**EXPERIMENT NO. 08**

|  |  |
| --- | --- |
| **DATE OF PERFORMANCE:** | **GRADE:** |
| **DATE OF ASSESSMENT:** | **SIGNATURE OF LECTURER/ TTA:** |

**AIM: To Study Inheritance in C++.**

**THEORY:**

**Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.**

**When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.**

**The idea of inheritance implements the is a relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.**

**BASE & DERIVED CLASSES:**

**A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form:**

***class derived-class: access-specifier base-class***

**Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.**

**Consider a base class Shape and its derived class Rectangle as follow:**

**#include <iostream.h>**

**// Base class**

**class Shape**

**{**

**public:**

**void setWidth(int w)**

**{**

**width = w;**

**}**

**void setHeight(int h)**

**{**

**height = h;**

**}**

**protected:**

**int width;**

**int height;**

**};**

**// Derived class**

**class Rectangle: public Shape**

**{**

**public:**

**int getArea()**

**{**

**return (width \* height);**

**}**

**};**

**int main(void)**

**{**

**Rectangle Rect;**

**Rect.setWidth(5);**

**Rect.setHeight(7);**

**// Print the area of the object.**

**cout << "Total area: " << Rect.getArea() << endl;**

**return 0;**

**}**

**When the above code is compiled and executed, it produces the following result:**

**Total area: 35**

**ACCESS CONTROL AND INHERITANCE:**

**A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.**

**We can summarize the different access types according to who can access them in the following way:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| **Same class** | **yes** | **yes** | **Yes** |
| **Derived classes** | **yes** | **yes** | **No** |
| **Outside classes** | **yes** | **no** | **No** |

**A derived class inherits all base class methods with the following exceptions:**

* **Constructors, destructors and copy constructors of the base class.**
* **Overloaded operators of the base class.**
* **The friend functions of the base class.**

**TYPE OF INHERITANCE:**

**When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access-specifier as explained above.**

**We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied:**

* **Public Inheritance: When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.**
* **Protected Inheritance: When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.**
* **Private Inheritance: When deriving from a private base class, public and protected members of the base class become private members of the derived class.**

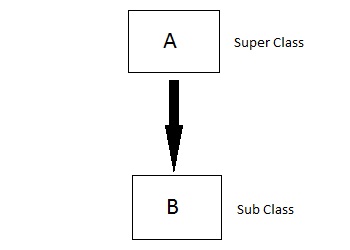
**TYPES OF INHERITANCE:**

**In C++, we have 5 different types of Inheritance. Namely,**

1. **Single Inheritance**
2. **Multiple Inheritance**
3. **Hierarchical Inheritance**
4. **Multilevel Inheritance**
5. **Hybrid Inheritance (also known as Virtual Inheritance)**

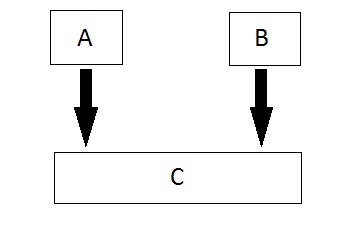
**SINGLE INHERITANCE:**

**In this type of inheritance one derived class inherits from only one base class. It is the simplest form of Inheritance.**

****

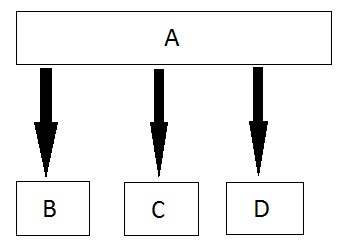
**MULTIPLE INHERITANCE:**

**In this type of inheritance a single derived class may inherit from two or more than two base classes.**

****

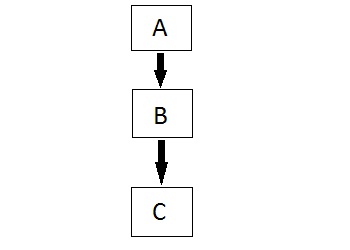
**HIERARCHICAL INHERITANCE:**

**In this type of inheritance, multiple derived classes inherit from a single base class.**

****

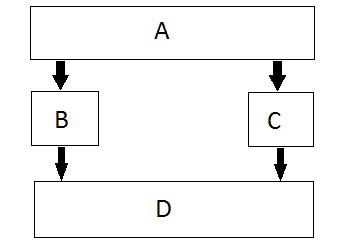
**MULTILEVEL INHERITANCE:**

**In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.**

****

**HYBRID (VIRTUAL) INHERITANCE:**

**Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.**

****

**MULTIPLE INHERITANCES:**

**A C++ class can inherit members from more than one class and here is the extended syntax:**

***class derived-class: access baseA, access baseB....***

**Where access is one of public, protected, or private and would be given for every base class and they will be separated by comma as shown above. Let us try the following example:**

**#include <iostream.h>**

**// Base class Shape**

**class Shape**

**{**

**public:**

**void setWidth(int w)**

**{**

**width = w;**

**}**

**void setHeight(int h)**

**{**

**height = h;**

**}**

**protected:**

**int width;**

**int height;**

**};**

**// Base class PaintCost**

**class PaintCost**

**{**

**public:**

**int getCost(int area)**

**{**

**return area \* 70;**

**}**

**};**

**// Derived class**

**class Rectangle: public Shape, public PaintCost**

**{**

**public:**

**int getArea()**

**{**

**return (width \* height);**

**}**

**};**

**int main(void)**

**{**

**Rectangle Rect;**

**int area;**

**Rect.setWidth(5);**

**Rect.setHeight(7);**

**area = Rect.getArea();**

**// Print the area of the object.**

**cout << "Total area: " << Rect.getArea() << endl;**

**// Print the total cost of painting**

**cout << "Total paint cost: $" << Rect.getCost(area) << endl;**

**return 0;**

**}**

**When the above code is compiled and executed, it produces the following result:**

**Total area: 35**

**Total paint cost: $2450**

**CONSTRUCTORS IN DERVIED CLASSES:**

**Base class constructors are always called in the derived class constructors. Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor finishes execution.**

**Whether derived class's default constructor is called or parameterised is called, base class's default constructor is always called inside them.**

**To call base classes parameterised constructor inside derived class's parameterised constructor, we must mention it explicitly while declaring derived class's parameterized constructor.**

**BASE CLASS DEFAULT CONSTRUCTOR IN DERIVED CLASS CONSTRUCTORS:**

**class Base**

**{**

**int x;**

**public:**

**Base() { cout << "Base default constructor"; }**

**};**

**class Derived : public Base**

**{**

**int y;**

**public:**

**Derived() { cout << "Derived default constructor"; }**

**Derived(int i) { cout << "Derived parameterized constructor"; }**

**};**

**int main()**

**{**

**Base b;**

**Derived d1;**

**Derived d2(10);**

**}**

**VIRTUAL BASE CLASS:**

**An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.**

**C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.**

**When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class’ name with the word virtual.**

**EXAMPLE:**

**class A**

**{**

**public:**

**int i;**

**};**

**class B : virtual public A**

**{**

**public:**

**int j;**

**};**

**class C: virtual public A**

**{**

**public:**

**int k;**

**};**

**class D: public B, public C**

**{**

**public:**

**int sum;**

**};**

**int main()**

**{**

**D ob;**

**ob.i = 10; //unambiguous since only one copy of i is inherited.**

**ob.j = 20;**

**ob.k = 30;**

**ob.sum = ob.i + ob.j + ob.k;**

**cout << “Value of i is : ”<< ob.i<<”\n”;**

**cout << “Value of j is : ”<< ob.j<<”\n”; cout << “Value of k is :”<< ob.k<<”\n”;**

**cout << “Sum is : ”<< ob.sum <<”\n”;**

**return 0;**

**}**

**ABSTRACT CLASS:**

**An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class.**

**The C++ interfaces are implemented using abstract classes and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.**

**A class is made abstract by declaring at least one of its functions as pure virtual function. A pure virtual function is specified by placing "= 0" in its declaration as follows:**

**class Box**

**{**

**public:**

**// pure virtual function**

**virtual double getVolume() = 0;**

**private:**

**double length; // Length of a box**

**double breadth; // Breadth of a box**

**double height; // Height of a box**

**};**

**The purpose of an abstract class (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be used to instantiate objects and serves only as an interface. Attempting to instantiate an object of an abstract class causes a compilation error.**

**Thus, if a subclass of an ABC needs to be instantiated, it has to implement each of the virtual functions, which means that it supports the interface declared by the ABC. Failure to override a pure virtual function in a derived class, then attempting to instantiate objects of that class, is a compilation error.**

**Classes that can be used to instantiate objects are called concrete classes.**

**PROGRAM 1: USE OF SINGEL INHERIHANCE.**

**#include<conio.h>**

**#include<iostream.h>**

**class b**

**{**

**int a;**

**public:**

**int b;**

**void set\_ab();**

**void get\_a(void);**

**void show\_a(void);**

**};**

**class d:public b**

**{**

**int c;**

**public:**

**void mul(void);**

**void display(void);**

**};**

**void b::set\_ab(void)**

**{**

**a=5;b=10;**

**};**

**void b::get\_a()**

**{**

**return a;**

**}**

**void b::show\_a()**

**{**

**cout<<"a="<<a<<"\n";**

**}**

**void d::mul()**

**{**

**c=b\*get\_a();**

**}**

**void d::display()**

**{**

**cout<<"a="<<get\_a()<<"\n";**

**cout<<"b="<<b<<"\n";**

**cout<<"c="<<c<<"\n";**

**}**

**void main()**

**{**

**d a;**

**d.set\_ab();**

**d.mul();**

**d.show\_a();**

**d.display();**

**d.b=20;**

**d.mul();**

**d.display();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 2: SINGEL INHERITANCE WITH PRIVATE CLASS.**

**#include<conio.h>**

**#include<iostream.h>**

**class b**

**{**

**int a;**

**public:**

**int b;**

**void get\_ab();**

**int get\_a(void);**

**void show\_a(void);**

**};**

**class d:private b**

**{**

**int c;**

**public:**

**void mul(void);**

**void display(void);**

**};**

**void b::get\_ab(void)**

**{**

**cout<<"Enter Values For a & b:";**

**cin>>a>>b;**

**}**

**int b::get\_a()**

**{**

**return a;**

**}**

**void b::show\_a()**

**{**

**cout<<"a="<<a<<"\n";**

**}**

**void d::mul()**

**{**

**get\_ab();**

**c=b\*get\_a();**

**}**

**void d::display()**

**{**

**show\_a();**

**cout<<"b="<<b<<"\n"<<"c="<<c<<"\n";**

**}**

**void main()**

**{**

**d b;**

**clrscr();**

**b.mul();**

**b.display();**

**b.mul();**

**b.display();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 3: USE OF MULTIPLE INHERITANCE.**

**#include<iostream.h>**

**#include<conio.h>**

**class alpha**

**{**

**int x;**

**public:**

**alpha(int i)**

**{**

**x=i;**

**cout<<"\n alpha initialized.";**

**}**

**void show\_x()**

**{**

**cout<<"\n x = "<<x;**

**}**

**};**

**class beta**

**{**

**float p,q;**

**public:**

**beta(float a,float b)**

**{**

**p=a;**

**q=b;**

**cout<<"\n beta initialized.";**

**}**

**void show\_pq()**

**{**

**cout<<"\n p = "<<p<<"\n q = "<<q;**

**}**

**};**

**class gamma:public beta,public alpha**

**{**

**int k;**

**public:**

**gamma(int a,float b,float c,int d):alpha(a\*d),beta(b,c)**

**{**

**k=d;**

**cout<<"\n gamma initialized.";**

**}**

**void show\_c()**

**{**

**cout<<"\n k = "<<k;**

**}**

**};**

**void main()**

**{**

**clrscr();**

**gamma g(2,2.5,1.5,7);**

**g.show\_x();**

**g.show\_pq();**

**g.show\_c();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 4: USE OF MULTILEVEL INHERITANCE.**

**#include<conio.h>**

**#include<iostream.h>**

**class student**

**{**

**protected:**

**int roll\_number;**

**public:**

**void get\_number(int);**

**void put\_number(void);**

**};**

**void student::get\_number(int a)**

**{**

**roll\_number=a;**

**}**

**void student::put\_number()**

**{**

**cout<<"Roll Number:"<<roll\_number<<"\n";**

**}**

**class test:public student**

**{**

**protected:**

**float sub1;**

**float sub2;**

**public:**

**void get\_marks(float x,float y)**

**{**

**sub1=x;**

**sub2=y;**

**}**

**void put\_marks()**

**{**

**cout<<"Marks In Sub1 = "<<sub1<<"\n";**

**cout<<"Marks In Sub2 = "<<sub2<<"\n";**

**}**

**};**

**class result:public test**

**{**

**float total;**

**public:**

**void display()**

**{**

**total=sub1+sub2;**

**put\_number();**

**put\_marks();**

**cout<<"Total = "<<total<<"\n";**

**}**

**};**

**void main()**

**{**

**clrscr();**

**result s;**

**s.get\_number(111);**

**s.get\_marks(75.0,59.5);**

**s.display();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 5: USE OF HYBRID INHERITANCE.**

**#include<iostream.h>**

**#include<conio.h>**

**class student**

**{**

**protected:**

**int roll\_no;**

**public:**

**void get\_no(int a)**

**{**

**roll\_no=a;**

**}**

**void put\_no()**

**{**

**cout<<"Roll no:"<<roll\_no<<"\n";**

**}**

**};**

**class test:public student**

**{**

**protected:**

**float part1,part2;**

**public:**

**void get\_marks(float x,float y)**

**{**

**part1=x;**

**part2=y;**

**}**

**void put\_marks(void)**

**{**

**cout<<"Marks obtained"<<"\n"**

**<<"part 1="<<part1<<"\n"**

**<<"part 2="<<part2<<"\n";**

**}**

**};**

**class sports**

**{**

**protected:**

**float score;**

**public:**

**void get\_score(float s)**

**{**

**score=s;**

**}**

**void put\_score(void)**

**{**

**cout<<"Sports wt:"<<score<<"\n";**

**}**

**};**

**class result:public test,public sports**

**{**

**float total;**

**public:**

**void display(void);**

**};**

**void result::display(void)**

**{**

**total=part1+part2+score;**

**put\_no();**

**put\_marks();**

**put\_score();**

**cout<<"Total Score:"<<total<<"\n";**

**}**

**void main()**

**{**

**result s1;**

**clrscr();**

**s1.get\_no(1234);**

**s1.get\_marks(27.5,33.0);**

**s1.get\_score(6.0);**

**s1.display();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 6: CONSTRUCTOR IN DERIVED CLASS.**

**#include<iostream.h>**

**#include<conio.h>**

**class alpha**

**{**

**int x;**

**public:**

**alpha(int i)**

**{**

**x=i;**

**cout<<"\n Alpha Initialized.";**

**}**

**void show\_x()**

**{**

**cout<<"\n x = "<<x;**

**}**

**};**

**class beta**

**{**

**float y;**

**public:**

**beta(float j)**

**{**

**y=j;**

**cout<<"\n Beta initialized.";**

**}**

**void show\_y()**

**{**

**cout<<"\n y = "<<y;**

**}**

**};**

**class gamma:public alpha,public beta**

**{**

**int m,n;**

**public:**

**gamma(int a,float b,int c,int d):alpha(a),beta(b)**

**{**

**m=c;**

**n=d;**

**cout<<"\n Gamma initialized.";**

**}**

**void show\_mn()**

**{**

**cout<<"\n m = "<<m<<"\n n = "<<n;**

**}**

**};**

**void main()**

**{**

**clrscr();**

**gamma g(10,2.5,2,6);**

**g.show\_x();**

**g.show\_y();**

**g.show\_mn();**

**getch();**

**}**

**OUTPUT:**

**PROGRAM 7: USE OF VIRTUAL IN INHERITANCE.**

**#include<iostream.h>**

**#include<conio.h>**

**class student**

**{**

**protected:**

**int roll\_no;**

**public:**

**void get\_no(int a)**

**{**

**roll\_no=a;**

**}**

**void put\_no(void)**

**{**

**cout<<"Roll no:"<<roll\_no<<"\n";**

**}**

**};**

**class test:virtual public student**

**{**

**protected:**

**float p1,p2;**

**public:**

**void get\_marks(float x,float y)**

**{**

**p1=x;**

**p2=y;**

**}**

**void put\_marks(void)**

**{**

**cout<<"Marks obtained:"<<"\n"**

**<<"part 1= "<<p1<<"\n"**

**<<"part 2= "<<p2<<"\n";**

**}**

**};**

**class sports:virtual public student**

**{**

**protected:**

**float score;**

**public:**

**void get\_score(float s)**

**{**

**score=s;**

**}**

**void put\_score(void)**

**{**

**cout<<"Sport wt:"<<score<<"\n";**

**}**

**};**

**class result:public test,public sports**

**{**

**float total;**

**public:**

**void display(void);**

**};**

**void result::display()**

**{**

**total=p1+p2+score;**

**put\_no();**

**put\_marks();**

**put\_score();**

**cout<<"Total score:"<<total<<"";**

**}**

**void main()**

**{**

**clrscr();**

**result s1;**

**s1.get\_no(678);**

**s1.get\_marks(30.5,25.5);**

**s1.get\_score(7.0);**

**s1.display();**

**getch();**

**}**

**OUTPUT:**