INHERITANCE

Classes can share, obtain or "inherit" properties and methods that belong to existing classes.

Base class: It is the class from which features are to be inherited into another class.

Derived class: It is the class in which the base class features are inherited. A derived class can have additional properties and methods not present in the parent class that distinguishes it and provides additional functionality.

REAL WORLD EXAMPLE

Let's consider the Windows operating system. Windows 98 had certain properties and methods that were used in Windows XP. Windows XP was derived from Windows 98, but it was still different from it. Windows 7 was derived from Windows XP, but they were both different. They both followed a certain basic template and shared properties, however.

SYNTAX (INHERITANCE)

```
class derived-class-name : access base-class-name {
// body of class
};
```

TYPES OF INHERITANCE WITH RESPECT TO BASE CLASS ACCESS CONTROL

- Public
- Private
- Protected

PUBLIC INHERITANCE

With public inheritance, every object of a derived class is also an object of that derived class's base class. However, base class objects are not objects of their derived classes.

For example, if we have Flower as a base class and Rose as a derived class, then all Roses are Flowers, but not all Flowers are Roses—for example, a Flower could also be Jasmine or a Tulip.

IS-A RELATIONSHIP

Inheritance is represented by an is-a relationship which means that an object of a derived class also can be treated as an object of its base class for example, a Rose is a Flower, so any attributes and behaviors of a Flower are also attributes and behaviors of a Rose.

PUBLIC INHERITANCE SYNTAX

```
class Rose : public Flower
```

TABLE SHOWING BASE CLASS ACCESS CONTROL (PUBLIC PRIVATE AND PROTECTED)

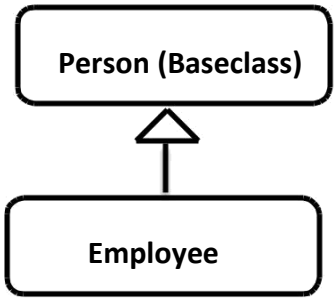
Visibility Of Base Class Members	TYPE OF INHERITANCE		
	PUBLIC INHERITANCE	PRIVATE INHERITANCE	PROTECTED INHERITANCE
PUBLIC	PUBLIC in derived class	PRIVATE in derived class	PROTECTED in derived class
PRIVATE	HIDDEN in derived class	HIDDEN in derived class	HIDDEN in derived class
PROTECTED	PROTECTED in derived class	PRIVATE in derived class	PROTECTED in derived class

TYPES OF INHERITANCE WITH RESPECT TO DERIVED CLASSES

- SINGLE INHERITANCE
- HIERARICAL
- MULTILEVEL
- MULTIPLE INHERITANCE
- HYBRID INHERITANCE

1) SINGLE INHERITANCE

It is the type of inheritance in which there is one base class and one derived class.



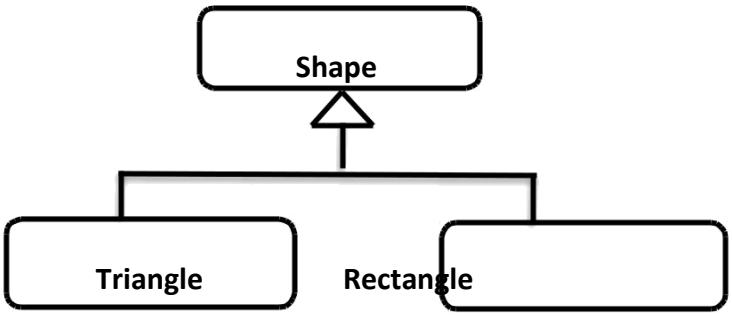
SINGLE INHERITANCE

```
#include <iostream>
using namespace std;

class Person
{
        char name[100],gender[10];
        int age;
        public: void
        getdata()
        {
                cout<<"Name: ";
                cin>>name;
                cout<<"Age: ";
                cin>>age;
                cout<<"Gender: ";
                cin>>gender;
        }

        void display()
        {
                cout<<"Name: "<<name<<endl;
                cout<<"Age: "<<age<<endl;
                cout<<"Gender: "<<gender<<endl;
        }
};
```

```
class Employee: public Person
{
        char
        company[100];
        float salary; public:
        void getdata()
        {
                Person::getdata();
                cout<<"Name of Company:
                "; cin>>company; cout<<"
                Salary: Rs."; cin>>salary;
        }
void display()
        {
                Person::display(); cout<<"Name of
        Company:"<<company<<endl; cout<<"Salary:
        Rs."<<salary<<endl;
        }
};

int main()
{
        Employee emp;
        cout<<"Enter data"<<endl;
        emp.getdata();
        cout<<endl<<"Displaying data"<<endl;
        emp.display();
        return 0;
}
```

2) HIERARICAL INHERITANCE

This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.

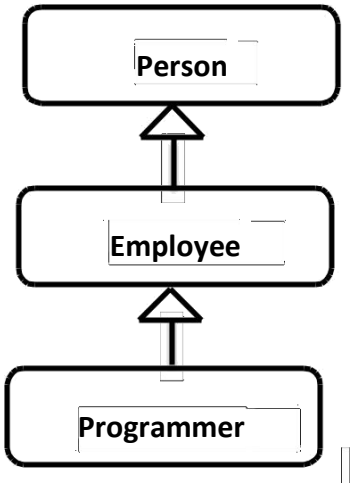


HIERARICAL INHERITANCE

```
#include <iostream>
using namespace std;


class Shape
{
        protected: float
        width, height;
        public:
        void set_data (float a, float b)
        { width = a; height
                = b;
        }
};
class Rectangle: public Shape
   { public: float
          area ()
        { return (width * height);
        }
};


class Triangle: public Shape
{ public:
        float area ()
        { return (width * height / 2);
        }
};
int main ()
{
        Rectangle rect; Triangle
        tri; rect.set_data (5,3);
        tri.set_data (2,5); cout <<
        rect.area() << endl; cout
        << tri.area() <<
        endl; return 0;
}
```

3) MULTI LEVEL INHERITANCE

When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.

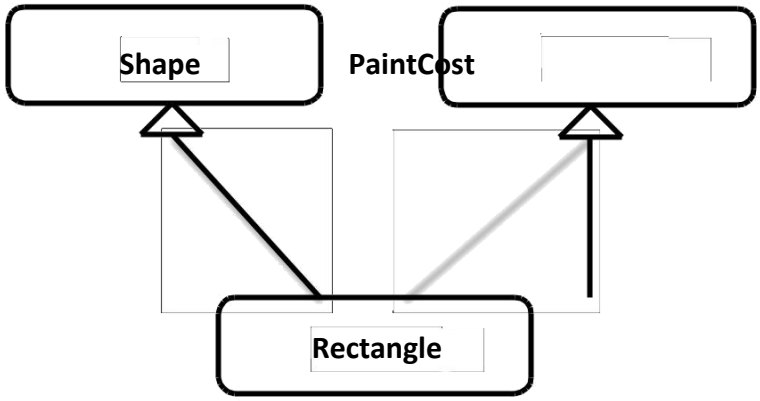


MULTI LEVEL IHERITANCE

```
#include <iostream>
using namespace std;

class Person
{
        char name[100],gender[10];
        int age;
        public: void
        getdata()
        {
                cout<<"Name: ";
                cin>>name;
                cout<<"Age: ";
                cin>>age;
                cout<<"Gender: ";
                cin>>gender;
        }

        void display()
        {
                cout<<"Name: "<<name<<endl;
                cout<<"Age: "<<age<<endl;
                cout<<"Gender: "<<gender<<endl;
        }
};

class Employee: public Person
{
        char
        company[100];
        float salary;
        public: void
        getdata()
        {
                Person::getdata();
                cout<<"Name of Company:
                "; cin>>company; cout<<"
                Salary: Rs."; cin>>salary;
        }
        void display()
                {
                        Person::display();
                        cout<<"Name of Company:"<<company<<endl;
                        cout<<"Salary: Rs."<<salary<<endl;
                }
};

 class Programmer: public Employee
{
        int number;
        public: void
        getdata()
        {
                Employee::getdata();
                cout<<"Number of programming language known: ";
                cin>>number;
        }
        void display()
        {
                Employee::display();
               cout<<"Number of programming language known:"<<number;
        }
};

int main()
{
        Programmer p;
        cout<<"Enter data"<<endl;
        p.getdata(); cout<<endl<<"Displaying
        data"<<endl; p.display();
        return 0;
}
```

4) MULTIPLE INHERITANCE

In multiple inheritance, one class inherits the features of multiple classes simultaneously, hence this type of inheritance is called Multiple Inheritance.

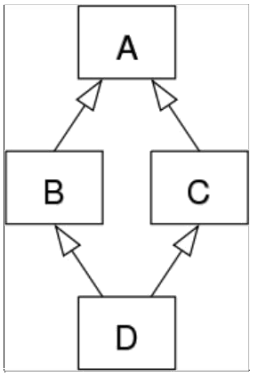


MULTIPLE INNHERITANCE

```
#include <iostream>
using namespace std;

class Shape  // Base class Shape
{ public:
        void setWidth(int w)
        { width = w;
        }
        void setHeight(int h)
        { height = h;
        }
        protected:
        int width;
        int height;
};

class PaintCost // Base class PaintCost
{ public: int getCost(int area)
        { return area * 70;
        }
};
```

```
class Rectangle: public Shape, public PaintCost // Derived class
{ public: int
        getArea()
        { return (width * height);
        }
};

int main()
{
        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;
}
```

5) HYBRID INHERITANCE

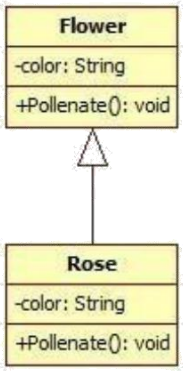


HYBRID INHERITANCE

```
#include<iostream> using
namespace std;



class Student
{
        protected:
        int rno; public:
        void get_no(int a)
        { rno=a;
        }
        void put_no(void)
        {
                cout<<"Roll no"<<rno<<"\n";



        }
};

class Test:public Student
{
        protected:
        float part1,part2; public: void
        get_mark(float x,float y)
        {
                part1=x; part2=y;
        }
        void put_marks()
        {
cout<<"Marksobtained:\npart1="<<part1<<"\n"<<"part2="<<part2
<<"\n";


        }
};
```

```
class Sports
{
        protected: float score;
        public:         void
        getscore(float s)
        { score=s;
        }
        void putscore()
        { cout<<"sports:"<<score<<"\n";
        } };



class Result: public Test, public Sports
{ float total; public:
        void display()
        {
                total=part1+part2+score;
                put_no(); put_marks();
                putscore();
                  cout<<"Total Score="<<total<<"\n";
        } };

int main()
{
        Result stu; stu.get_no(123);
        stu.get_mark(27.5,33.0);
        stu.getscore(6.0);
        stu.display();
        return 0;
}
```

UML REPRESENTATION FOR INHERITANCE (GENERALIZATION)




```
class Flower
{ string color;
        public:
        void Pollenate();
};
class Rose : public Flower
{ };
```

CONSTRUCTOR CALLS

The constructor of a derived class is required to create an object of the derived class type. As the derived class contains all the members of the base class, the base sub-object must also be created and initialized. The base class constructor is called to perform this task. Unless otherwise defined, this will be the default constructor.

The order in which the constructors are called is important. The base class constructor is called first, then the derived class constructor. The object is thus constructed from its core outwards.

BASE CLASS INITIALIZER

C++ requires that a derived-class constructor call its base-class constructor to initialize the base-class data members that are inherited into the derived class. In a derive class, constructor uses a base class initialize list which uses a member initializer to pass arguments to the base-class. When a derived class constructor calls a base-class constructor, the arguments passed to the base-class constructor must be consistent with the number and types of parameters specified in one of the base-class constructors; otherwise, a compilation error occurs.

BASE CLASS INITIALIZER WITH SIINGLE INHERITANCE

```
#include<iostream>

#include<string>

using namespace std;

class Account
{ private:
        long accountNumber;             // Account number

        protected:
        string name;                    // Account holder

        public:                         //Public interface:
        const string accountType;       // Account Type

Account(long accNumber, string accHolder, const string& accType)
: accountNumber(accNumber), name(accHolder), accountType(accType)
{ cout<<"Account's constructor has been called"<<endl<<endl;
}

~ Account()                             //Destructor
{
```

```
        cout<<endl<<"Object Destroyed";
}

const long getAccNumber() const  //accessor for privately defined data menber; accountNumber
{ return accountNumber;
}

void DisplayDetails()
{
        cout<<"Account Holder: "<<name<<endl; cout<<"Account
        Number: "<<accountNumber<<endl; cout<<"Account Type:
        "<<accountType<<endl;
}
};

class CurrentAccount : public Account //Single Inheritance
{ private:
        double balance; public:

        CurrentAccount(long accNumber, const string& accHolder, string accountType, double accBalance) :
        Account(accNumber, accHolder, accountType), balance(accBalance)
        { cout<<"CurrentAccount's constructor has been called"<<endl<<endl;
        }


void deposit_currbal()
   {
    float deposit;
    cout<<"Enter amount to Deposit : ";
    cin>>deposit; cout<<endl; balance =
    balance + deposit;
   }

void Display()
{
        name = "Dummy"; //can change protected data member of Base class DisplayDetails();
        cout<<"Account Balance: "<<balance<<endl<<endl;
}
};

int main()
{
        CurrentAccount currAcc(7654321,"Dummy1", "Current Account",
        1000); currAcc.deposit_currbal(); currAcc.Display();

        return 0;
}
```

BASE CLASS INITIALIZER WITH MULTIPLE INHERITANCE

```
#include<iostream> using
namespace std;

class FirstBase
{
        protected:
        int a; public:
        FirstBase(int x)
        {
cout<<"Constructor of FirstBase is called: "<<endl; a=x;
        }
};

class SecondBase
{
        protected:
        string b; public:
        SecondBase(string x)
        {
cout<<"Constructor of SecondBase is called: "<<endl;
                b=x;
        } };
```

```
class Derived : public FirstBase, public SecondBase
{ public:
        Derived(int a,string b):
FirstBase(a),SecondBase(b)
        {
        cout<<"Child Constructor is called: "<<endl;
        }

        void display()
        { cout<<a<<" "<<b<<endl;
        }
};

int main()
{
        Derived obj(24,"Multiple Inheritance");
        obj.display();
}
```

DESTROYING OBJECTS

When an object is destroyed, the destructor of the derived class is first called, followed by the destructor of the base class. The reverse order of the constructor calls applies. You need to define a destructor for a derived class if actions performed by the constructor need to be reversed. The base class destructor need not be called explicitly as it is executed implicitly.

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's time and effort. Since the main code written can be reused in various situations as needed. **Maintainability**
- It is easy to debug a program when divided in parts.

LAB 06 EXERCISES

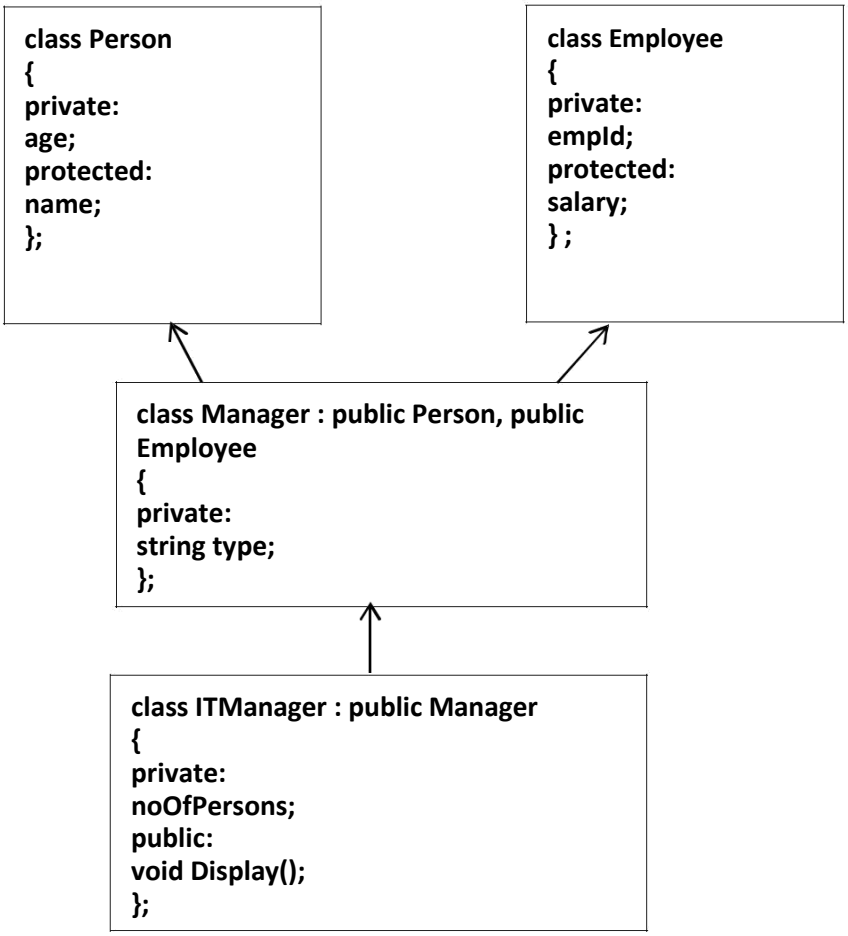
INSTRUCTIONS:

NOTE: Violation of any of the following instructions may lead to the cancellation of your submission.

1) Create a folder and name it by your student id (k16-1234).
2) Paste the .cpp file for each question with the names such as Q1.cpp, Q2.cpp and so on into that folder.
3) Submit the zipped folder on slate.

QUESTION#1

Implement the following scenario in C++:



1) The Display() function in "ITManager" should be capable of displaying the values of all the data members declared in the scenario (age,name,empId,salary,type,noOfPersons) without being able to alter the values.

2) The "int main()" function should contain only three program statements which are as follows:

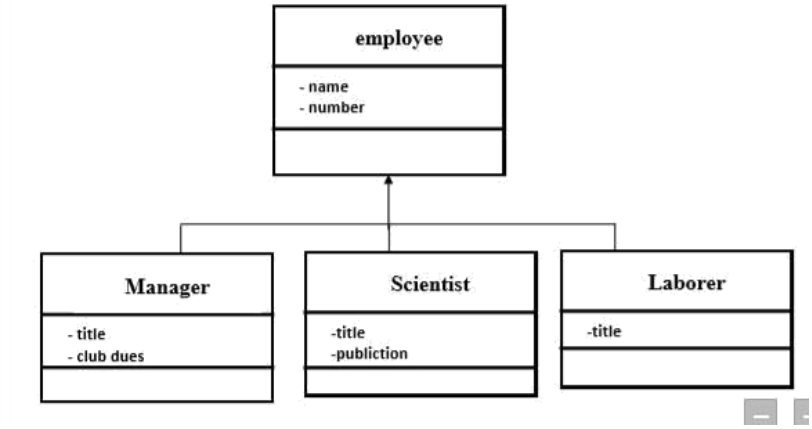
a) In the first statement, create object of "ITManager" and pass the values for all the data members:
ITManager obj(age,name,empId,salary,type,noOfPersons);
b) In the second statement, call the Display() function.
c) In the third statement, return 0.

Question#2

In the database only three kinds of employees are represented. Managers manage, scientists perform research to develop better widgets, and laborers operate the dangerous widget-stamping presses.

The database stores a name and an employee identification number for all employees, no matter what their category is. However, for managers, it also stores their titles and golf club dues. For scientists, it stores the number of scholarly articles they have published and their title. Laborers store the title only.

You must start with a base class employee. This class handles the employee's last name and employee number. From this class three other classes are derived: manager, scientist, and laborer. All three classes contain additional information about these categories of employee, and member functions to handle this information as shown in figure.



Question#3

A supermarket chain has asked you to develop an automatic checkout system. All products are identifiable by means of a barcode and the product name. Groceries are either sold in packages or by weight. Packed goods have fixed prices. The price of groceries sold by weight is calculated by multiplying the weight by the current price per kilo.

Develop the classes needed to represent the products first and organize them hierarchically. The Product class, which contains generic information on all products (barcode, name, etc.), can be used as a base class.

The Product class contains two data members for storing barcodes and the product name. Define a constructor with parameters for both data members. Add default values for the parameters to provide a default constructor for the class. In addition to the access methods setCode() and getCode(), also define the methods scanner() and printer(). For test purposes, these methods will simply output product data on screen or read the data of a product from the keyboard.

The next step involves developing special cases of the Product class. Define two classes derived from Product, PrepackedFood and Fresh- Food. In addition to the product data, the PrepackedFood class should contain the unit price and the FreshFood class should contain a weight and a price per kilo as data members. In both classes define a constructor with parameters providing default-values for all data members. Use both the base and member initializer. Define the access methods needed for the new data members.Also redefine the methods scanner() and printer() to take the new data members into consideration.

Test the various classes in a main function that creates three objects each of the types Product, PrepackedFood and FreshFood. One object of each type is fully initialized in the object definition. Use the default constructor to create the other object and test the scanner() method. For the third object, test the get and set methods. Display the products on screen for all objects.