DHA SUFFA UNIVERSITY
Department of Computer Science
Object Oriented Programming
CS-1002L
Spring 2020

Lab 07

Composition , Aggregation, and Inheritance

Objectives

- To learn about composition, aggregation and inheritance.

Composition

The composition is a type of association where the relationship is restrictive i.e. if two objects are in composition, the composed object will not exist without the other.

Example 7.1

```
//FileName:Birthday.h
#ifndef BIRTHDAY_H
#define BIRTHDAY_H
class Birthday{
      public:
            Birthday();
            Birthday(int m, int d, int y);
            void printDate();
            ~Birthday();
      private:
            int month;
            int day;
            int year;
};
#endif
```

```
//FileName:Birthday.cpp
#include <iostream>
#include "Birthday.h"
Birthday::Birthday()
{
            std::cout << "Constructor of Birthday class" << std::endl;
}
Birthday::Birthday(int m,int d,int y):month(m),day(d),year(y){
      std::cout << "Constructor of Birthday class" << std::endl;
```

```
}
Birthday::~Birthday()
{
	std::cout << "Destructor of Birthday class" << std::endl;
}
void Birthday::printDate()
{
	std::cout << month << "/" << day << "/" << year << std::endl;
}


}

//FileName:Person.h
#ifndef Person_H
#define Person_H
#include <iostream>
#include "Birthday.h"
class Person{
	public:
		Person();
		Person(std::string ,const Birthday&);
		void printInfo();
		~Person();
	private:
		std::string name;
		Birthday dateofBirth;
};
#endif
```

```
//FileName:Person.cpp

#include "Person.h"
Person::Person(std::string n,const Birthday&
dateofBirth):name(n),dateofBirth(dateofBirth){};
Person::~Person()
{
	std::cout << "Destructor of Person class" << std::endl;
}
void Person::printInfo()
```

```
{
	std::cout << name << " was born on ";
	dateofBirth.printDate();
}
```

```
//FileName:main.cpp

#include "Person.h"
int main(){
	//Creating Object
	Birthday b(2, 2, 2000);
	Person p1("Ali",b);
	p1.printInfo();
	return 0;
}
```

Output

```
Constructor of Birthday class
Ali was born on 2/2/2000
Destructor of Person class
Destructor of Birthday class
Destructor of Birthday class
```

Aggregation

Aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

Example 7.2

```
//FileName:Teacher.h
#ifndef TEACHER_H
#define TEACHER_H
#include <string>
class Teacher
{
      public:
       Teacher(const std::string tname,const std::string email);
       ~Teacher();
       std::string getName();
       std::string getEmail();
      private:
       std::string tname;
       std::string email;
};
#endif
```

```
//FileName:Teacher.cpp
#include "Teacher.h"

Teacher :: Teacher(const std::string tname,const std::string email)
{
    this->tname = tname;
    this->email = email;
}

Teacher :: ~Teacher(){}

std::string Teacher::getName() {
    return this->tname;
}

std::string Teacher::getEmail() {
    return this->email;
}
```

```
//FileName:Department.h
#ifndef Department_H
```

```
#define Department_H
#include "Teacher.h"
#include <iostream>
class Department
{
    public:
       Department(int id, std::string name, Teacher* teach);
       ~Department();
         friend   std::ostream&   operator<<(std::ostream&   out,   const
Department& department);
    private:
       int id;
       std::string name;
       Teacher* teach;//Department HAS-A Teacher
};
#endif
```

```
//FileName:Department.cpp
#include "Department.h"
Department:: Department(int id, std::string name, Teacher* teach)
{
    this->id = id;
    this->name = name;
    this->teach = teach;
}

Department::~Department(){}

std::ostream& operator<<(std::ostream& out, const Department& department)
{
     out << "Department id: " << department.id << " Department Name: " <<
department.name
       << " Teacher name: " << department.teach->getName()
       << " Teacher email: " << department.teach->getEmail();
    return out;
}
```

```
//FileName:main.cpp
#include "Teacher.h"
#include "Department.h"
#include <iostream>
int main()
{
    Teacher t1("Mr.Ali Ahmed","ali@dsu.edu.pk");
    Department d1(1,"Computer Science",&t1);

    std::cout << d1 << std::endl;
    return 0;
}
```

Output

```
Department id: 1 Department Name: Computer Science Teacher name: Mr.Ali
Ahmed Teacher email: ali@dsu.edu.pk
```

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.

Role of Constructor in Inheritance

When a derived or child class object is instantiated its base or parent class constructor will be invoked first and then the constructor of derived or child class will be invoked.

Types of inheritance with respect to base class access control

- `public`
- `private`
- `protected`

Syntax of public inheritance

```
class child className: type parent className
{
};
```

Types of inheritance with respect to Structure

- Single
- Multiple
- Multi level
- Hierarchal

Single Inheritance

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

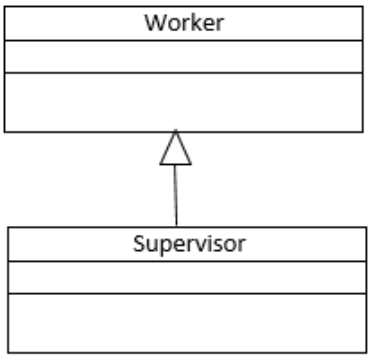


Fig 7.1:Single Inheritance

Example 7.3

```
//FileName:Worker.h
#ifndef WORKER_H
#define WORKER_H
#include<iostream>
//base class of worker
class Worker
{
    //member variables and functions
    protected:
```

```
        std::string name;
        int age;
    public:
        Worker();
        ~Worker();
        void showProfile();
};
#endif
```

```
//FileName:Worker.cpp

#include"Worker.h"

//function definitions
Worker::Worker():name(""), age(0){
    std::cout << "Worker Constructor" << std::endl;
};

//destructor
Worker::~Worker()
{
    std::cout << "Worker Destructor" << std::endl;
}

void Worker::showProfile()
{
    std::cout << "Worker's Profile" << std::endl;
}
```

```
//FileName:Supervisor.h

#ifndef SUPERVISOR_H
#define SUPERVISOR_H
#include"Worker.h"

//derived class inherit the base class using public derivation
class Supervisor : public Worker
{
     //member variables
     private:
        int subOrdinates;
```

```
     //member functions
     public:
        Supervisor();
        ~Supervisor();
        void input();
        void show();
};
#endif
```

```
//FileName:Supervisor.cpp

#include"Supervisor.h"

//function definitions
Supervisor::Supervisor():subOrdinates(0)
{
    std::cout << "Supervisor Constructor" << std::endl;
}

Supervisor::~Supervisor()
{
    std::cout<<"Supervisor Destructor" << std::endl;
}

void Supervisor::input()
{
    std::cout<< "Enter your Name :" << std::endl;
    std::cin >> name;

    std::cout << std::endl << "Enter the Age :" << std::endl;
    std::cin >> age;
    std::cout << std::endl << "Enter the number of workers under you : "
              << std::endl;
    std::cin >> subOrdinates;
}


void Supervisor::show()
{
    std::cout << std::endl << "Name: "<<name;
    std::cout << std::endl << "Age: "<<age;
    std::cout << std::endl << "Number of workers under you is: "
```

```
                << subOrdinates << std::endl;
}
```

```
//FileName:driver.cpp

#include"Supervisor.h"
int main ()
{
    //creating object of supervisor
    Supervisor s1;
    //calling function
    s1.input();
    s1.show();
    return 0;
}
```

Output

```
Worker Constructor
Supervisor Constructor
Enter your Name :
Aijaz
Enter the Age :
24
Enter the number of workers under you :
30
Name: Aijaz
Age: 24
Number of workers under you is: 30
Supervisor Destructor
Worker Destructor
```

Multilevel Inheritance

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

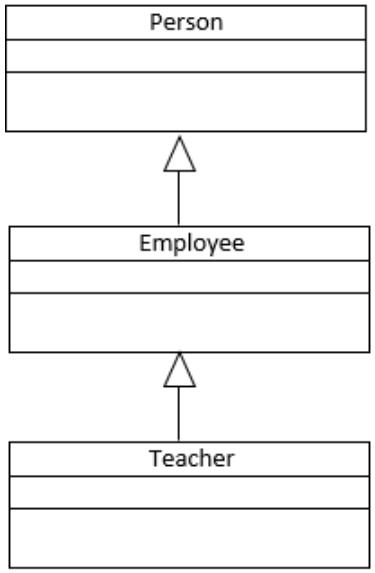


Fig 7.4:Multi level inheritance

Example 7.4

```
//FileName:Person.h

#ifndef PERSON_H
#define PERSON_H
#include <iostream>

//defining person class
class Person
{
     //member variables and functions
      private:
            std::string name;
            int age;
      public:
            Person();
            ~Person();
            void inputData();
            void display();
};
#endif
```

```
//FileName:Person.cpp

#include "Person.h"
#include <iostream>

//function definitions
Person::Person():name(""), age(0){}

Person::~Person(){}

void Person :: inputData()
{
      std::cout << "Name:" ;
      std::cin>>name;

      std::cout << " Age: ";
      std::cin >>age;
}

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

```
//FileName:Employee.h

//Deriving Employee class from person class
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include"Person.h"
//class of employee inherited from Person class
class Employee: public Person
{
      //member functions and variables
      protected:
            int empNo;
            float salary;
      public:
      Employee();
```

```
      ~Employee();
      void setData(int empNo, float salary);
      void display();
};
#endif
```

```
//FileName:Employee.cpp

#include "Employee.h"
//function definitions
Employee::Employee():empNo(0), salary(0.0){}

Employee::~Employee(){}

void Employee :: setData(int empNo, float salary)
{
      this->empNo = empNo;
      this->salary = salary;
}

void Employee :: display()
{
      std::cout <<"Employee Number: "<< empNo << std::endl;
      std::cout <<"Employee salary: "<< salary << std::endl;
}
```

```
//FileName:Teacher.h
#ifndef TEACHER_H
#define TEACHER_H
#include "Employee.h"
//Deriving Teacher class from employee class
class Teacher:public Employee
{
    //member functions
      public:
            Teacher();
            ~Teacher();
            void inputData();
            void display();
```

```
      //Declaring data members(instance variables)
      private:
            int numberOfCourses;
};
#endif
```

```
//FileName:Teacher.cpp

#include "Teacher.h"

//Defining member function
Teacher::Teacher():numberOfCourses(0){}

Teacher::~Teacher(){}

void Teacher :: inputData()
{
      std::cout << "Number of courses: ";
      std::cin >> numberOfCourses;
}

//Defining member function
void Teacher :: display()
{
      std::cout << "Number of courses: "<< numberOfCourses <<std::endl;
      std::cout << "Employee Number: "<< empNo << std::endl;
      std::cout << "Employee Salary: "<< salary << std::endl;
}



//FileName:Driver.cpp

#include "Teacher.h"
int main()
{
      int empNumber = 10;
      float salary = 55.5;

      //object creation
```

```
	Teacher t;
	std::cout <<"Enter data"<< std::endl;

	//calling functions
	t.inputData();
	t.setData(empNumber, salary);

	std::cout << std::endl <<"Displaying data"<< std::endl;
	t.display();

	return 0;
}
```

Output

```
Enter data
Number of courses: 3
Displaying data
Number of courses: 3
Employee Number: 10
Employee Salary: 55.5
```

Multiple Inheritance

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

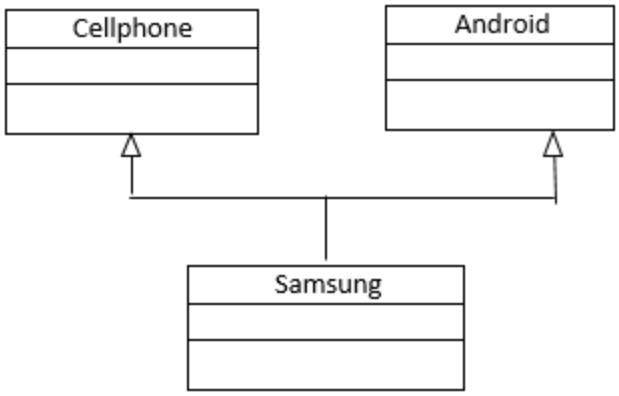


Fig 7.5:Multiple Inheritance

Example 7.5

```
//FileName:CellPhone.h
#ifndef CELLPHONE_H
#define CELLPHONE_H

//Base class of cellPhone
class CellPhone
{
      //member variables
      protected:
            int memory;
      //member functions
      public:
            CellPhone();
            ~CellPhone();
            void call();

};
#endif
```

```
//FileName: CellPhone.cpp

#include"CellPhone.h"
#include <iostream>

CellPhone::CellPhone():memory(0){}

CellPhone::~CellPhone(){}

void CellPhone::call()
{
      std::cout<<"Make Call" << std::endl;
}
```

```
//FileName:Android.h

#ifndef ANDROID_H
```

```
#define ANDROID_H
#include <iostream>

//base class of Android
class Android
{
	//member variables
	protected:
	std::string operatingSystem;
	//member functions
	public:
		Android();
		~Android();
	void AppStore();

};
#endif
```

```
//FileName:Android.cpp

#include"Android.h"
//defining member functions
Android::Android():operatingSystem(""){}

Android::~Android(){}

void Android :: AppStore()
{
	std::cout << "Open App Store" << std::endl;
}
```

```
//FileName:Samsung.h

#ifndef SAMSUNG_H
#define SAMSUNG_H
#include"Android.h"
#include"CellPhone.h"
//derived class of samsung from cellPhone and Android Class
class Samsung : public CellPhone, public Android
```

```
{
     //member functions
      public:
            Samsung();
            ~Samsung();
            void unlock();

     //member variables
      private:
            int mobilePassword;
};
#endif
```

```
//FileName:Samsung.cpp

#include"Samsung.h"
//function definitions
Samsung::Samsung():mobilePassword(0){}

Samsung::~Samsung(){}

void Samsung :: unlock()
{
      std::cout << "Unlock Phone" << std::endl;
}
```

```
//FileName:Driver.cpp

#include "Samsung.h"
int main()
{
      //object creation
      Samsung myPhone;

      //function calls
      myPhone.unlock();
      myPhone.call();
      myPhone.AppStore();
}
```

Output

```
Unlock Phone
Make Call
Open App Store
```

Data Access Types

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.

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes
Accessible from 2nd derived class?	no	yes	yes

Example 7.6

```
//FileName:Automobile.h

#ifndef AUTOMOBILE_H
#define AUTOMOBILE_H
#include<iostream>

class Automobile
{
     //member functions and variables
      protected:
            std::string make;
```

```
            int model;
      public:
            Automobile();
            ~Automobile();
            void setMake(std::string make);
            void setModel(int model);
};
#endif
```

```
//FileName:Automobile.cpp

#include "Automobile.h"

//function definitions
Automobile::Automobile():make(“”), model(1990){}

Automobile::~Automobile(){}

void Automobile::setMake(std::string make)
{
      this->make = make;
}

void Automobile::setModel(int model)
{
      this->model = model;
}
```

```
//FileName:FourWheeler.h

#ifndef FourWheeler_H
#define FourWheeler_H

class FourWheeler
{
     //member variables and functions
      protected:
            int wheels;
      public:
            FourWheeler();
            ~FourWheeler();
```

```
            void start();
            void stop();

};
#endif
```

```
//FileName:FourWheeler.cpp

#include "FourWheeler.h"
#include <iostream>

//member function definitions
FourWheeler::FourWheeler():wheels(0){}

FourWheeler::~FourWheeler(){}

void FourWheeler::start()
{
      std::cout << "start" << std::endl;
}

void FourWheeler::stop()
{
      std::cout << "stop" << std::endl;
}

//FileName:Car.h

#ifndef CAR_H
#define CAR_H
#include "Automobile.h"
#include "FourWheeler.h"

//inherited class of car

class Car : public Automobile, public FourWheeler
{
      //member variables
      protected:
            int doors;

      //member functions
```

```
      public:
            Car();
            ~Car();
            void setDoors(int doors);
            int getDoors();
            void showInfo();
};
#endif
```

```
//FileName:Car.cpp

#include "Car.h"

//function definitions
Car::Car():doors(0){}

Car::~Car(){}

void Car::setDoors(int doors)
{
      this->doors = doors;
}

int Car::getDoors()
{
      return doors;
}

void Car::showInfo()
{
      std::cout<< "Make:" << make << std::endl;
      std::cout<< "Model:" << model << std::endl;
}
```

```
//FileName:main.cpp

#include "Car.h"
#include "FourWheeler.h"

int main()
```

```
{
      //object creation
      Car c1;

     //calling function
      c1.setMake("japanese");
      c1.setModel(12);
      c1.showInfo();
      c1.start();

      return 0;
}
```

Output

```
Make:japanese
Model:2016
start
```

Protected Inheritance

When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes	yes (inherited as protected variables)
Accessible from 2nd derived class?	no	yes	yes

Example 7.7

```
//FileName:Mobile.h

#ifndef MOBILE_H
#define MOBILE_H
#include<iostream>

//base class of mobile
class Mobile
{
	//data members and functions
	protected:
	std::string type;
	public:
	Mobile();
	~Mobile();
	void call();

};
#endif

//FileName:Mobile.cpp

#include"Mobile.h"

//function definition
Mobile::Mobile():type(""){}

Mobile::~Mobile(){}

void Mobile::call()
{
	std::cout << "Make Call" << std::endl;
}
```

```
//FileName:Samsung.h

#ifndef SAMSUNG_H
#define SAMSUNG_H
#include "Mobile.h"
```

```
//inherited from Mobile class
class Samsung : protected Mobile
{
	public:
	Samsung();
	~Samsung();
	void openAppStore();

	//data member and member functions
	protected:
	std::string os;
	float megapixelCamera;
};
#endif
```

//FileName:**Samsung.cpp**

```
#include"Samsung.h"
//function definition
Samsung::Samsung():os(""), megapixelCamera(0.0){}

Samsung::~Samsung(){}

void Samsung::openAppStore()
{
	std::cout << "Open App Store" <<std::endl;
}
```

//FileName:**SamsungS9.h**

```
#ifndef SAMSUNGS9_H
#define SAMSUNGS9_H
#include "Samsung.h"

//inherited from samsung class
class SamsungS9 : protected Samsung
{
	public:
	SamsungS9();
	void setData(std::string t,std::string os,float mp);
	void display();
	~SamsungS9();
```

```
      //declaring member variables and functions
      protected:
      int screenSize;
};
#endif

//SamsungS9.cpp
#include"SamsungS9.h"
#include<iostream>
//function definitions
SamsungS9::SamsungS9():screenSize(0){}

SamsungS9::~SamsungS9(){}

void SamsungS9::setData(std::string type, std::string os,float
megapixelCamera)
{
      this->type = type;
      this->os = os;
      this->megapixelCamera = megapixelCamera;
}

void SamsungS9::display()
{
      std::cout << "Type: " << type << std::endl;
      std::cout << "OS: " << os << std::endl;
      std::cout << "Mega Pixel Camera: "<< megapixelCamera << std::endl;
}
```

```
//FileName:main.cpp

#include"SamsungS9.h"
int main()
{
   //object creation
   SamsungS9 phone;

   //calling function
   phone.setData("dual sim", "android", 13.3);
   phone.display();
```

```
   return 0;
}
```

Output

```
Type: dual sim
OS: android
Mega Pixel Camera: 13.3
```

Private Inheritance

When deriving from a private base class, public and protected members of the base class become private members of the derived class.

Accessibility	private variables	protected variables	public variables
Accessible from own class?	yes	yes	yes
Accessible from derived class?	no	yes (inherited as private variables)	yes (inherited as private variables)
Accessible from 2nd derived class?	no	no	no

Example 7.8

```
//FileName:Student.h

#ifndef STUDENT_H
#define STUDENT_H
#include<iostream>
//base class of student
class Student
{
```

```
      //member variables and functions
      protected:
         std::string name;
      public:
         Student();
         ~Student();
         void inputData();
         void showData();

};
#endif

//FileName:Student.cpp
//function definitions
#include "Student.h"
Student::Student():name(""){}

Student::~Student(){}

void Student::inputData()
{
      std::cin >> name;
}

void Student::showData()
{
      std::cout << name << std::endl;
}
```

```
//FileName:Marks.h

#ifndef MARKS_H
#define MARKS_H
//second base class of marks
class Marks
{
      protected:
          int marks1;
          int marks2;
      public:
          Marks();
          ~Marks();
```

```
          void inputMarks();
          void showMarks();

};
#endif

//FileName:Marks.cpp
//function definitions
#include "Marks.h"
#include <iostream>

Marks::Marks():marks1(0), marks2(0){}

Marks::~Marks(){}

void Marks::inputMarks()
{
   std::cin >> marks1;
   std::cin >> marks2;
}

void Marks::showMarks()
{
   std::cout << marks1 << std::endl;
   std::cout << marks2 << std::endl;
}
```

```
//FileName:Result.h
#ifndef RESULT_H
#define RESULT_H
#include "Student.h"
#include "Marks.h"

//derived class of result
class Result : private Student, private Marks
{
      //member variables and functions
      protected:
      int total;
      public:
            Result();
            ~Result();
```

```
      void CalculateResult();
      void printResult();

};
#endif

//FileName:Result.cpp
#include "Result.h"

//function definitions
Result::Result():total(0){}

Result::~Result(){}

void Result::CalculateResult()
{
      std::cout << "Enter your name:" << std::endl;
      std::cin >> name;
      std::cout<<"Enter Marks1:" << std::endl;
      std::cin >> marks1;
      std::cout <<"Enter Marks2:" << std::endl;
      std::cin >> marks2;
      total = marks1 + marks2;
}

//printing result
void Result::printResult()
{

      std::cout << "RESULT:" << std::endl;
      std::cout << "Name:" << name << std::endl;
      std::cout << "Total Marks:" << total << std::endl;
}
```

```
//FileName:main.cpp
#include"Result.h"
int main()
{
    //object creation
    Result r;

    //function calls
```

```
    r.CalculateResult();
    r.printResult();

    return 0;
}
```

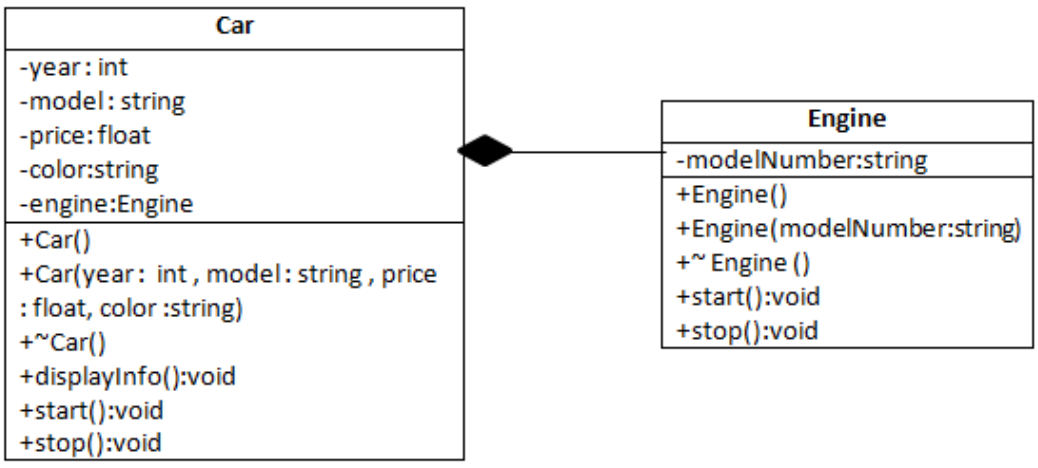
Output

```
Enter your name:
Ali
Enter Marks1:
23
Enter Marks2:
22
RESULT:
Name:Ali
Total Marks:45
```

Tasks

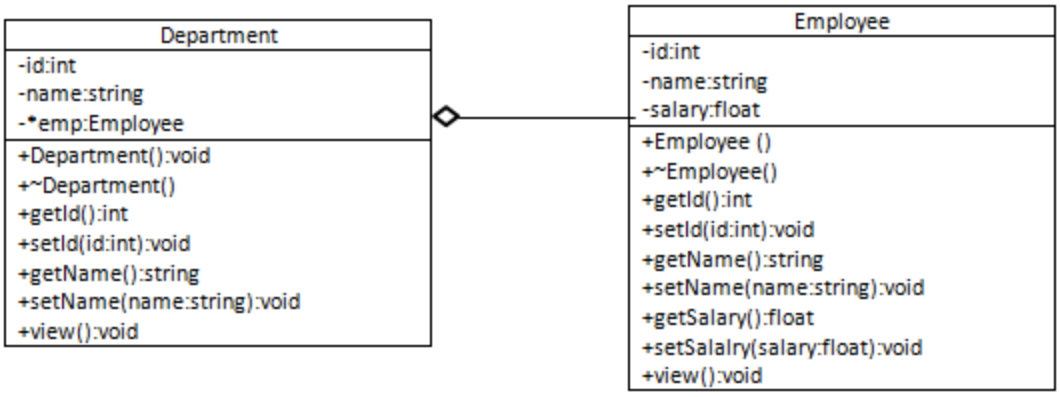
Task 7.1

Implement the following diagram in a c++ code.



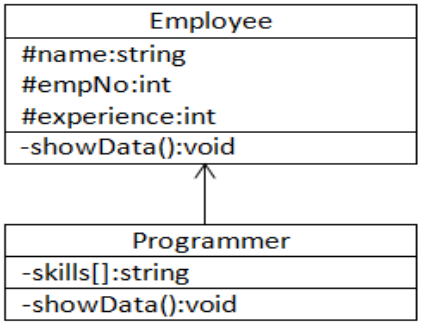
Task 7.2(Home Assignment)

Implement the following diagram in a c++ code



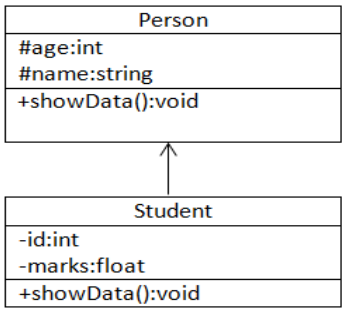
Task 7.3

Create a class Employee and derive a child class programer using public inheritance.



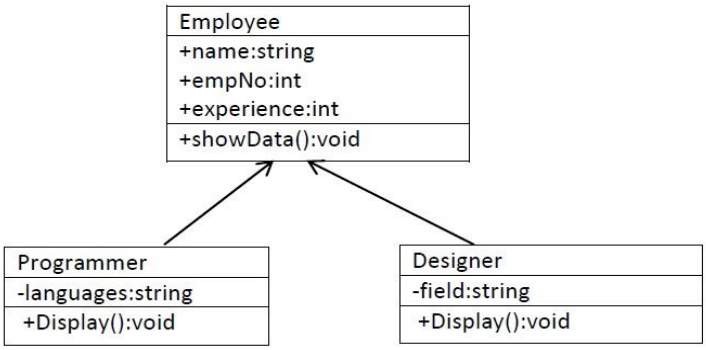
Task 7.4

Create a parent class Person , derive child class Student from it using private inheritance.



Task 7.5(Home Assignment)

Create a parent class Employee , derive child class Programmer and Designer from it using protected inheritance.



Task 7.6 (Home Assignment)

1. Define a class `Human` holding a `firstName` and a `lastName`.
2. Define a class `Student` derived from `Human` that has a field `EnrollmentNumber`.
3. Define a class Worker derived from Human with fields `weekSalary` and `workHoursPerDay` and method `moneyPerHour()` that returns the payment earned by hour in a week by the `worker`.

Submission Instructions

1. Number your solution folders as question number e.g. Q1, Q2, etc. (Q is in upper case)
2. Every folder should contain three files (one header, one implementation and one driver)
3. Create a new folder named cs192abc where abc is your 3 digit roll #. e.g. cs192111.
4. Copy all the project folders into this folder.
5. Now make sure a zip file named cs192abc.zip is created e.g. cs192111.zip
6. Upload the assignment solution on LMS under the assignment named Lab 07
7. Assignment – XX, where XX is your section name.