Salman Ahmad

04072113050

BSCS 6th Sem

CS-121 OOP Assignment 3

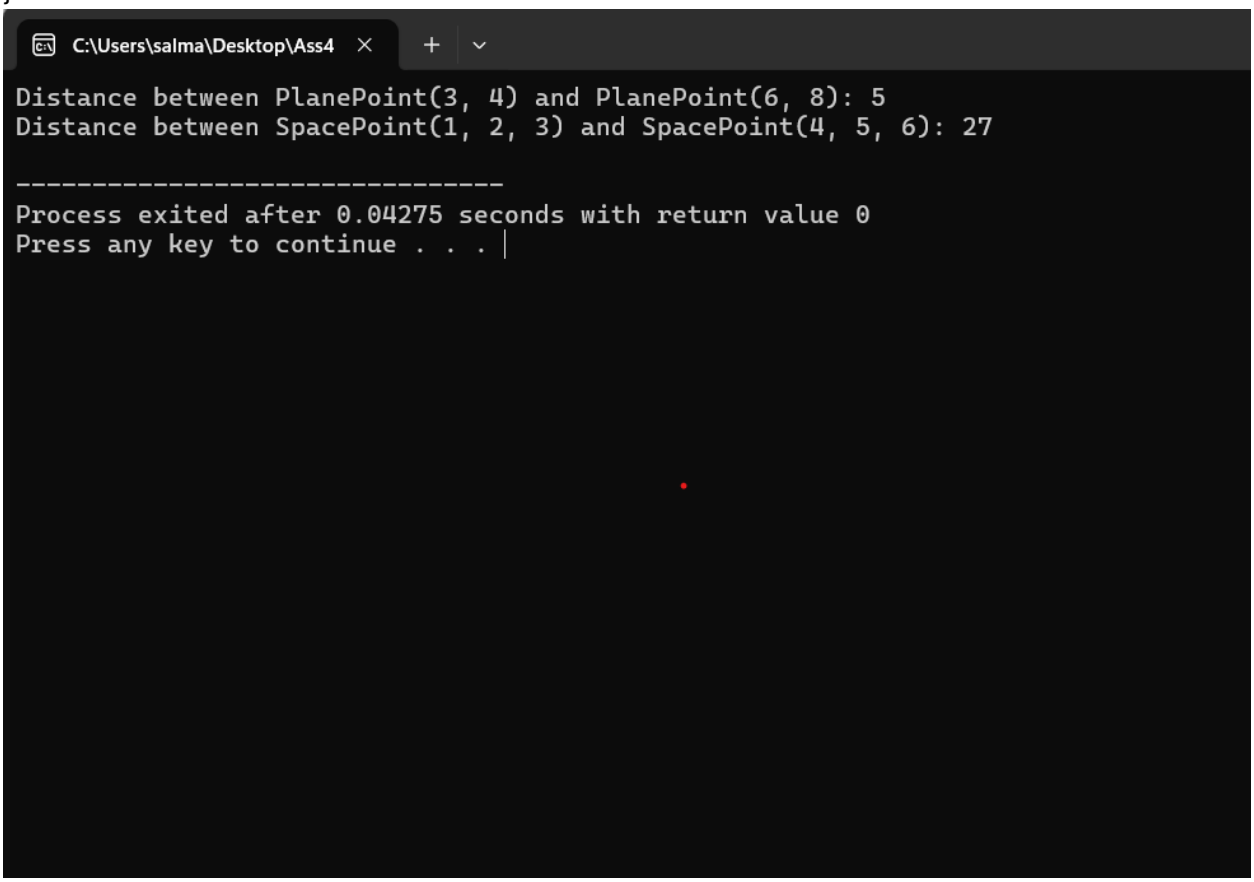
Q1.

```
#include <iostream>
#include <cmath>

using namespace std;



class PlanePoint {
protected:
   int X;
   int Y;

public:
   PlanePoint() : X(0), Y(0) {}

   PlanePoint(int x, int y) : X(x), Y(y) {}

   // Accessor functions
   int getX() const
                        { return X; }
```

```
    int getY() const
                        { return Y; }

    //distance calculation
    double planeDistance(PlanePoint& other)  {
        return sqrt(pow(X - other.X, 2) + pow(Y - other.Y, 2));
    }
};

// Derived class to represent a point in a 3D space
class SpacePoint : public PlanePoint {
private:
    int Z;

public:
    SpacePoint() : PlanePoint(), Z(0) {}
    SpacePoint(int x, int y, int z) : PlanePoint(x, y), Z(z) {}

    // Accessor function for Z
    int getZ() const { return Z; }

    // Function to calculate the distance between two SpacePoint objects
    double spaceDistance(const SpacePoint& other) const {
        return pow(X - other.X, 2) + pow(Y - other.Y, 2) + pow(Z - other.Z, 2);
    }
};

int main() {
    // Creating two points in the plane
```

```
PlanePoint point1(3, 4);
PlanePoint point2(6, 8);

// Calculating and displaying the distance between the two PlanePoint objects
double distance2D = point1.planeDistance(point2);
cout << "Distance between PlanePoint(3, 4) and PlanePoint(6, 8): " << distance2D << endl;

// Creating two points in the space
SpacePoint sp1(1, 2, 3);
SpacePoint sp2(4, 5, 6);

// Calculating and display the distance between the two SpacePoint objects
double distance3D = sp1.spaceDistance(sp2);
cout << "Distance between SpacePoint(1, 2, 3) and SpacePoint(4, 5, 6): " << distance3D << endl;

return 0;
```

}



Q2.

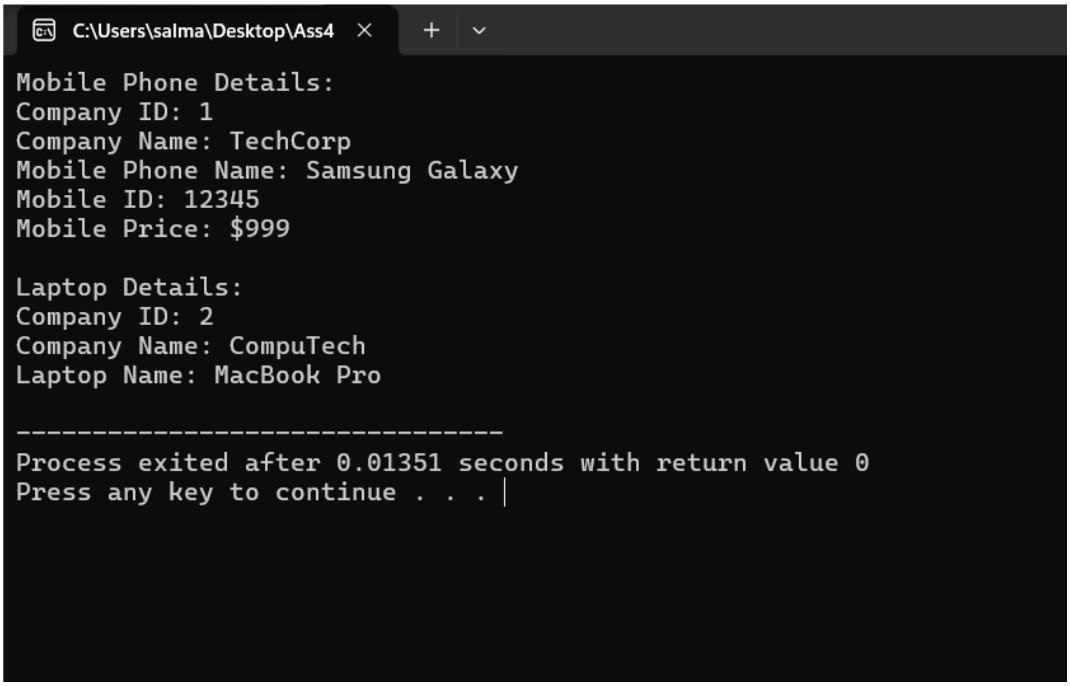
```
#include <iostream>

#include <string>


using namespace std;


// Base class Company

class Company {

protected:  // protected to allow derived classes access

   int companyID;

   string companyName;
```

```
public:
    // constructor
    Company(int ID, const string& name) : companyID(ID), companyName(name) {}

    // Setters
    void setCompanyID(int id) { companyID = id; }
    void setCompanyName(const string& name) { companyName = name; }

    // Getters
    int getID() const { return companyID; }
    string getCompanyName() const { return companyName; }
};

// Derived class MobilePhone
class MobilePhone : public Company {
private:
    string mobilePhoneName;
    int mobileID;
    int mobilePrice;

public:
    // Constructor
    MobilePhone(int id, const string& companyName, const string& phoneName, int phoneID, int
price)
        : Company(id, companyName), mobilePhoneName(phoneName), mobileID(phoneID),
mobilePrice(price) {}

    // Setters
    void setMobilePhoneName(const string& name) { mobilePhoneName = name; }
```

```
    void setMobileID(int id) { mobileID = id; }
    void setMobilePrice(int price) { mobilePrice = price; }

    // Getters
    string getMobilePhoneName() const { return mobilePhoneName; }
    int getMobileID() const { return mobileID; }
    int getMobilePrice() const { return mobilePrice; }

    // Display function implementation
    void display() const {
        cout << "Company ID: " << companyID << endl;
        cout << "Company Name: " << companyName << endl;
        cout << "Mobile Phone Name: " << mobilePhoneName << endl;
        cout << "Mobile ID: " << mobileID << endl;
        cout << "Mobile Price: $" << mobilePrice << endl;
    }
};

// Derived class Laptop
class Laptop : public Company {
private:
    string laptopName; // Data member to hold the laptop name

public:
    // Constructor
    Laptop(int id, const string& companyName, const string& name)
        : Company(id, companyName), laptopName(name) {}

    // Setter
```

```
    void setLaptopName(const string& name) { laptopName = name; }

    // Getter
    string getLaptopName() const { return laptopName; }

    // Display function implementation
    void display() const {
        cout << "Company ID: " << companyID << endl;
        cout << "Company Name: " << companyName << endl;
        cout << "Laptop Name: " << laptopName << endl;
    }
};

// Main function
int main() {

    MobilePhone phone(1, "TechCorp", "Samsung Galaxy", 12345, 999);
    Laptop laptop(2, "CompuTech", "MacBook Pro");
    cout << "Mobile Phone Details:" << endl;

    //so the advantage of using protected is that we can access private variables of base class
directly
    phone.display();

    // Display details for Laptop
    cout << "\nLaptop Details:" << endl;

    //so the advantage of using protected is that we can access private variables of base class
directly
```

```
    laptop.display();

    return 0;
}
```

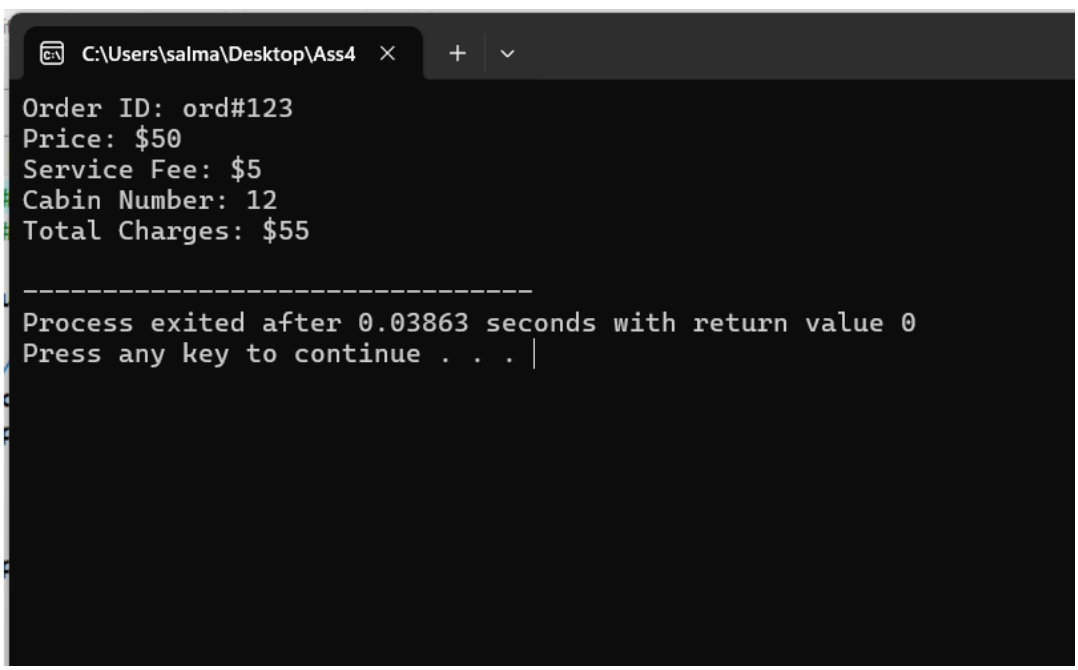


Q3.

```
#include <iostream>
#include <string>

using namespace std;

// Base class CafeService
class CafeService {
protected:
    string orderID;
```

```
    double price;

public:
    // No-argument constructor
    CafeService() : orderID("ord#0"), price(0.0) {}

    // Parameterized constructor
    CafeService(const string& id, double p) : orderID(id), price(p) {}



};

// Derived class StaffService
class StaffService : public CafeService {
private:
    double serviceFee;
    int cabinNumber;

public:
    StaffService(const string& id, double p, double fee, int cabin)
        : CafeService(id, p), serviceFee(fee), cabinNumber(cabin) {}

    // Function to calculate total charges
    double totalCharges() const {
        return price + serviceFee;
    }
```

```
  // Display function
  void display() const {
    cout << "Order ID: " << orderID << endl;
    cout << "Price: $" << price << endl;
    cout << "Service Fee: $" << serviceFee << endl;
    cout << "Cabin Number: " << cabinNumber << endl;
    cout << "Total Charges: $" << totalCharges() << endl;
  }
};

// Main function
int main() {

  StaffService service("ord#123", 50.0, 5.0, 12);
  service.display();

  return 0;
}
```



Q4.

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

class Automobile {
protected:
   double currentSpeed;

public:

   void setCurrentSpeed(double speed) {
      currentSpeed = speed;
   }

   double getCurrentSpeed() const {
      return currentSpeed;
   }
};

class Car : public Automobile {
protected:
   string color;

public:
   // Parameterized constructor
```

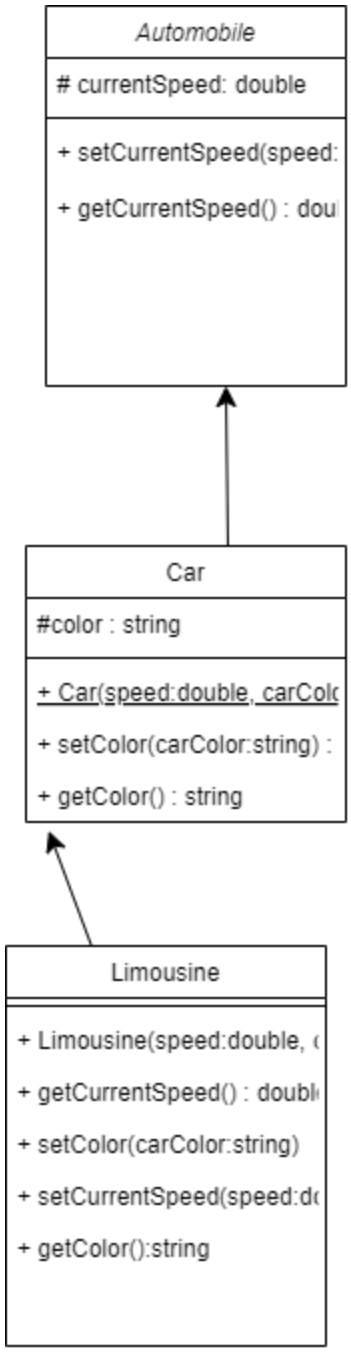
```
    Car(double speed, string& carColor)
    {
            currentSpeed = speed;
            color = carColor;
            }

    void setColor(string& carColor) {
        color = carColor;
    }

    string getColor() const {
        return color;
    }
};


class Limousine : public Car {
public:
    // Parameterized constructor
    Limousine(double speed, string carColor) : Car(speed, carColor) {}


//overRiding all these functions below
    void setCurrentSpeed(double speed) {
        Automobile::setCurrentSpeed(speed);
    }

    double getCurrentSpeed() const {
        return Automobile::getCurrentSpeed();
```

```
    }


    void setColor(string carColor) {
        Car::setColor(carColor);
    }


    string getColor() const {
        return Car::getColor();
    }
};

// Main function
int main() {

    Limousine limo(80.0, "Black");

    cout << "Current Speed: " << limo.getCurrentSpeed() << " km/h" << endl;
    cout << "Color: " << limo.getColor() << endl;

    // Modifying and displaying the details again
    limo.setCurrentSpeed(100.0);
    limo.setColor("Silver");
    cout << "\nUpdated Details:" << endl;
    cout << "Current Speed: " << limo.getCurrentSpeed() << " km/h" << endl;
    cout << "Color: " << limo.getColor() << endl;

    return 0;
```

}

```
C:\Users\salma\Desktop\Ass4

Current Speed: 80 km/h
Color: Black

Updated Details:
Current Speed: 100 km/h
Color: Silver

--------------------------------
Process exited after 0.01315 seconds with return value 0
Press any key to continue . . .
```

UML Diagram of Q4 Down Here:


Automobile
currentSpeed: double
+ setCurrentSpeed(speed:
+ getCurrentSpeed() : dou
Car
#color : string
+ Car(speed:double, carColo
+ setColor(carColor:string) :
+ getColor() : string
Limousine
+ Limousine(speed:double, (
+ getCurrentSpeed() : doubl
+ setColor(carColor:string)
+ setCurrentSpeed(speed:d
+ getColor():string

Q5.

```
#include <iostream>
#include <string>

using namespace std;

class Laboratory {
protected:
   string name;
   string location;

public:
   // No-argument constructor
   Laboratory() : name("NoName"), location("NULL") {}

   // Member function to input data
   virtual void input() {
      cout << "Enter Laboratory name: ";
      getline(cin, name);
      cout << "Enter Laboratory location: ";
      getline(cin, location);
   }

   // Member function to show data
```

```
    virtual void show() {
        cout << "Laboratory Name: " << name << endl;
        cout << "Laboratory Location: " << location << endl;
    }

    // Virtual destructor
    virtual ~Laboratory() {}
};

class WetLab : public Laboratory {
private:
    int no_of_microscopes;
    string scientist_name;

public:
    // Overriding input function
     void input() {
        cout << "Enter Laboratory name: ";
        getline(cin, name);
        cout << "Enter Laboratory location: ";
        getline(cin, location);

        cout << "Enter number of microscopes: ";
        cin >> no_of_microscopes;
        cin.ignore(); // To ignore the newline character left in the buffer
        cout << "Enter Scientist's name: ";
        getline(cin, scientist_name);
    }
```

```
    // Overriding show function
    void show()  {
      cout << "Laboratory Name: " << name << endl;
      cout << "Laboratory Location: " << location << endl;
      cout << "Number of Microscopes: " << no_of_microscopes << endl;
      cout << "Scientist's Name: " << scientist_name << endl;
    }

    // Setter and Getter
    void setNoOfMicroscopes(int microscopes)
            {
                    no_of_microscopes = microscopes;
            }
    int getNoOfMicroscopes()
            {
                     return no_of_microscopes;
            }



    void setScientistName(const string& name)
            {
                    scientist_name = name;
            }
    string getScientistName()
            {
             return scientist_name;
            }
};
```

```
class DryLab : public Laboratory {
private:
   int no_of_computers;
   int capacity;

public:
   // Overriding input function
   void input() {
      cout << "Enter Laboratory name: ";
      getline(cin, name);
      cout << "Enter Laboratory location: ";
      getline(cin, location);

      cout << "Enter number of computers: ";
      cin >> no_of_computers;
      cout << "Enter capacity: ";
      cin >> capacity;
      cin.ignore(); // To ignore the newline character left in the buffer
   }

   // Overriding show function
   void show() {
      cout << "Laboratory Name: " << name << endl;
      cout << "Laboratory Location: " << location << endl;
      cout << "Number of Computers: " << no_of_computers << endl;
      cout << "Capacity: " << capacity << endl;
   }

   // Setter and Getter for no_of_computers
```

```
    void setNoOfComputers(int computers)
            {
              no_of_computers = computers;
            }

    int getNoOfComputers()
            {
              return no_of_computers;
            }

    // Setter and Getter for capacity
    void setCapacity(int cap)
            {
             capacity = cap;
            }
    int getCapacity()
            {
             return capacity;
            }
};

int main() {
    // instance of Laboratory
    Laboratory lab;
    lab.input();
    lab.show();
    cout << endl;
```

```
//instance of WetLab
        Laboratory *p;
        WetLab w;
        p =&w;
        p->input();
        p->show();
        cout<<endl;


 // instance of DryLab
 DryLab dryLab;
 dryLab.input();
 dryLab.show();
 cout << endl;

 return 0;
```

}

C:\Users\salma\Desktop\Ass4

```
Enter Laboratory name: riaz
Enter Laboratory location: rwd
Laboratory Name: riaz
Laboratory Location: rwd

Enter Laboratory name: riaz
Enter Laboratory location: rwd
Enter number of microscopes: 5
Enter Scientist's name: john
Laboratory Name: riaz
Laboratory Location: rwd
Number of Microscopes: 5
Scientist's Name: john

Enter Laboratory name: heelo
Enter Laboratory location: ew
Enter number of computers: 37
Enter capacity: 23
Laboratory Name: heelo
Laboratory Location: ew
Number of Computers: 37
Capacity: 23


--------------------------------
Process exited after 35.78 seconds with return value 0
Press any key to continue . . .
```

Q6.

```
#include <iostream>
#include <string>

using namespace std;

// Base class Record
class Record {
protected:
   int rollNo;
   string course1Name;
   string course2Name;

public:
   // Parameterized constructor
   Record(int roll, const string& course1, const string& course2)
      : rollNo(roll), course1Name(course1), course2Name(course2) {}

   // Getter functions
   int getRollNo() const
            { return rollNo; }

   string getCourse1Name()
           { return course1Name; }
```

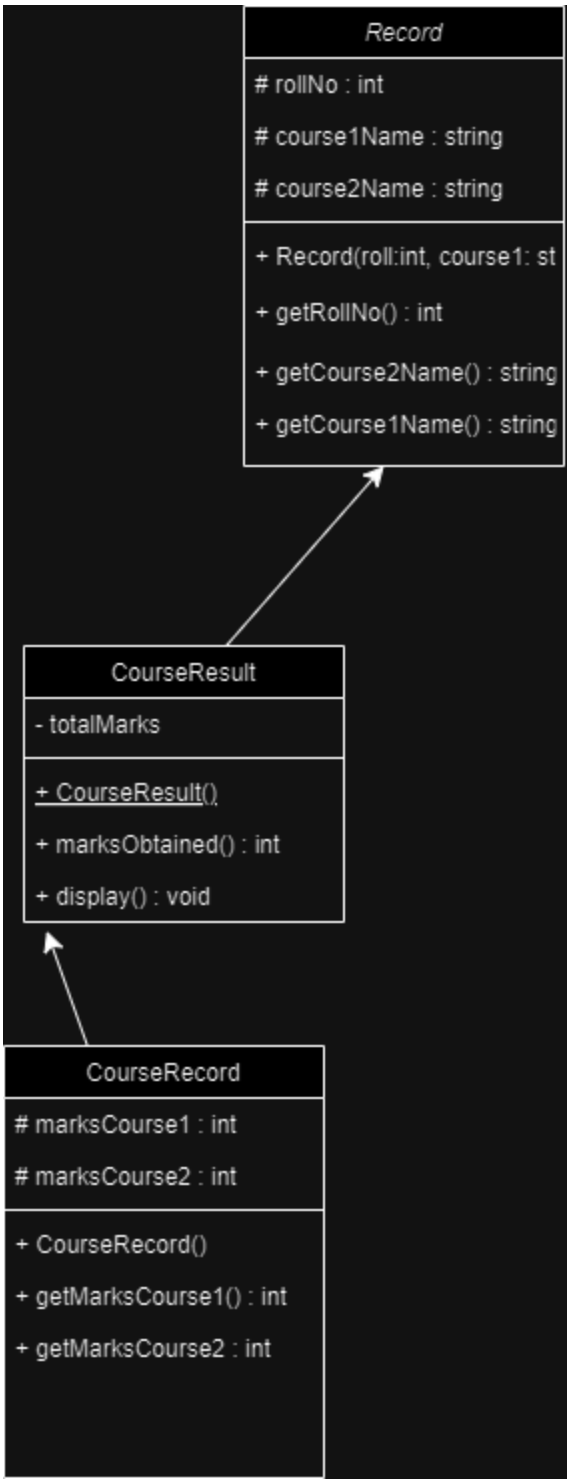
```
    string getCourse2Name()
            { return course2Name; }
};

// Derived class CourseRecord
class CourseRecord : public Record {
protected:
    int marksCourse1;
    int marksCourse2;

public:
    // Parameterized constructor
    CourseRecord(int roll, const string& course1, const string& course2, int marks1, int marks2)
        : Record(roll, course1, course2), marksCourse1(marks1), marksCourse2(marks2) {}

    // Getter functions
    int getMarksCourse1()
            { return marksCourse1; }

    int getMarksCourse2()
            { return marksCourse2; }
};

// Derived class CourseResult
class CourseResult : public CourseRecord {
private:
    int totalMarks;

public:
```

```
    // Parameterized constructor
    CourseResult(int roll, const string& course1, const string& course2, int marks1, int marks2)
        : CourseRecord(roll, course1, course2, marks1, marks2) {
        totalMarks = marksObtained();
    }

    // Function to calculate and return total marks
    int marksObtained() {
        totalMarks = marksCourse1 + marksCourse2;
        return totalMarks;
    }

    // Function to display all information
    void display()  {
        cout << "Roll No: " << getRollNo() << endl;
        cout << "Course 1 Name: " << getCourse1Name() << endl;
        cout << "Course 2 Name: " << getCourse2Name() << endl;
        cout << "Marks in Course 1: " << getMarksCourse1() << endl;
        cout << "Marks in Course 2: " << getMarksCourse2() << endl;
        cout << "Total Marks: " << totalMarks << endl;
    }
};

int main() {
    // instance of CourseResult
    CourseResult result(1, "Mathematics", "Physics", 85, 90);
    result.display();

    return 0;
```

}

```
C:\Users\salma\Desktop\Ass4

Roll No: 1
Course 1 Name: Mathematics
Course 2 Name: Physics
Marks in Course 1: 85
Marks in Course 2: 90
Total Marks: 175

--------------------------------
Process exited after 0.0122 seconds with return value 0
Press any key to continue . . .
```

UML Diagram of Q6 Down Here:


Record
rollNo : int
course1Name : string
course2Name : string
+ Record(roll:int, course1: st
+ getRollNo() : int
+ getCourse2Name() : string
+ getCourse1Name() : string
CourseResult
- totalMarks
+ CourseResult()
+ marksObtained() : int
+ display() : void
CourseRecord
marksCourse1 : int
marksCourse2 : int
+ CourseRecord()
+ getMarksCourse1() : int
+ getMarksCourse2 : int

The End.