PPS-3

Aayush Arora

23BCE0534

Q1) Write a code with Student as a base class, Mark as intermediate class and R
as a

derived class.

- The base class named Student with data members id, and name and gets
  and

putstu() are the methods to read and display the id and name.

- The intermediate class named Marks with data members m1, m2, m3 and
  getmarks()

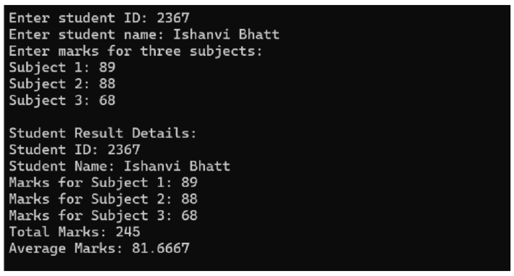
and putmarks() are the methods to read and display the marks.

- The derived class named Result with total, average as a data members
  show() as

the method to display the total and average of marks

```
#include <iostream>

#include <string>


using namespace std;


class Student {

protected:

  int id;

  string name;


public:

  void getstu() {

    cout << "Enter student ID: ";

    cin >> id;

    cout << "Enter student name: ";

    cin.ignore();

    getline(cin, name);
  }

  void putstu() {
    cout << "Student ID: " << id << endl;
    cout << "Student Name: " << name << endl;
  }
};

class Marks : public Student {
protected:
  int m1, m2, m3;

public:
  void getmarks() {
    getstu();
    cout << "Enter marks for three subjects:\n";
    cout << "Subject 1: ";
    cin >> m1;
    cout << "Subject 2: ";
    cin >> m2;
    cout << "Subject 3: ";
    cin >> m3;
  }

  void putmarks() {
    putstu();
    cout << "Marks for Subject 1: " << m1 << endl;
    cout << "Marks for Subject 2: " << m2 << endl;
    cout << "Marks for Subject 3: " << m3 << endl;
  }
};
```

```
class Result : public Marks {
private:
  int total;
  double average;

public:
  void calculate() {
    total = m1 + m2 + m3;
    average = static_cast<double>(total) / 3.0;
  }

  void show() {
    putmarks();
    calculate();
    cout << "Total Marks: " << total << endl;
    cout << "Average Marks: " << average << endl;
  }
};

int main() {
  Result studentResult;
  studentResult.getmarks();
  cout << "\nStudent Result Details:\n";
  studentResult.show();

  return 0;
}
```



```
Enter student ID: 2367
Enter student name: Ishanvi Bhatt
Enter marks for three subjects:
Subject 1: 89
Subject 2: 88
Subject 3: 68

Student Result Details:
Student ID: 2367
Student Name: Ishanvi Bhatt
Marks for Subject 1: 89
Marks for Subject 2: 88
Marks for Subject 3: 68
Total Marks: 245
Average Marks: 81.6667
```

Q2) . Create a parent class named ‘Courier’ with the following:

1. Data members CourierID, Name_of_Courier.
2. Method named PrintBill() to accept the Base_fare of type double as parameter and display the CourierID, Name_of_Courier , and Shipping_Cost, where Shipping_Cost = Base_fare + 30.

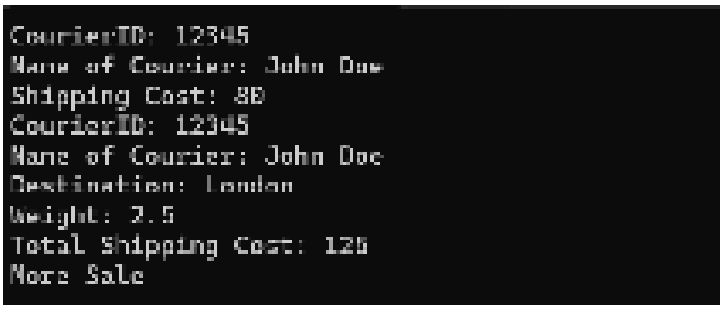
- Create a subclass names ‘Internatial_services’ which inherits from the Courier class. The class include the following:

1. Data members Destination, Weight.

Method named FinalBill()to print the CourierID, Name_of_Courier, Destination, Weight and Total_ ShippingCost, where Total_ShippingCost = Base_fare * Weight. Print the message “More Sale” when Total_ShippingCost is more than 100, otherwise print the message “Less Sale”.

```
#include <iostream>

#include <string>


class Courier {

protected:

   std::string CourierID;

   std::string Name_of_Courier;


public:

   Courier(const std::string& id, const std::string& name)

      : CourierID(id), Name_of_Courier(name) {}
```

```cpp
    void PrintBill(double Base_fare) const {
        double Shipping_Cost = Base_fare + 30;
        std::cout << "CourierID: " << CourierID << std::endl;
        std::cout << "Name of Courier: " << Name_of_Courier << std::endl;
        std::cout << "Shipping Cost: " << Shipping_Cost << std::endl;
    }
};

class International_services : public Courier {
private:
    std::string Destination;
    double Weight;

public:
    International_services(const std::string& id, const std::string& name,
                           const std::string& dest, double weight)
        : Courier(id, name), Destination(dest), Weight(weight) {}

    void FinalBill(double Base_fare) const {
        double Total_ShippingCost = Base_fare * Weight;
        std::cout << "CourierID: " << CourierID << std::endl;
        std::cout << "Name of Courier: " << Name_of_Courier << std::endl;
        std::cout << "Destination: " << Destination << std::endl;
        std::cout << "Weight: " << Weight << std::endl;
        std::cout << "Total Shipping Cost: " << Total_ShippingCost << std::endl;

        if (Total_ShippingCost > 100) {
            std::cout << "More Sale" << std::endl;
        } else {
            std::cout << "Less Sale" << std::endl;
        }
    }
};

int main() {
    International_services shipment("12345", "John Doe", "London", 2.5);
    shipment.PrintBill(50.0); // Example base fare
    shipment.FinalBill(50.0); // Example base fare

    return 0;
}
```



```
CourierID: 12345
Name of Courier: John Doe
Shipping Cost: 80
CourierID: 12345
Name of Courier: John Doe
Destination: London
Weight: 2.5
Total Shipping Cost: 125
More Sale
```

Q3) . Create two classes DM and DB. DM stores the distance in meters and centimetres and DB Stores the distance in feet and inches. Write a program to read the values for the class

objects and Add one object DM with another object DB. Note: use a friend function to

carry out addition operation. The resultant object is stored in DM and display it.

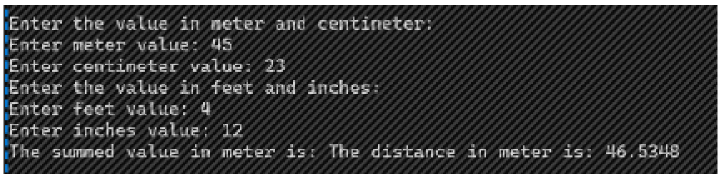
1 feet = 30 centimetres

1 inch =2.54 centimetres

1 meter = 100 centimetres

```
#include <iostream>

using namespace std;
```

```
class DB;

class DM {
private:
    int meter;
    float centimeter;

public:
    DM() {
        meter = 0;
        centimeter = 0.0;
    }

    DM(int m, float cm) {
        meter = m;
        centimeter = cm;
    }

    friend DM addDMDB(DM obj1, DB obj2);

    void display() {
        cout << "The distance in meter is: " << meter + centimeter / 100.0 << endl;
    }
};

class DB {
private:
    int feet;
    float inches;

public:
    DB() {
        feet = 0;
        inches = 0.0;
    }

    DB(int ft, float in) {
        feet = ft;
        inches = in;
    }

    friend DM addDMDB(DM obj1, DB obj2);

    void display() {
        cout << "The distance in feet is: " << feet + inches / 12.0 << endl;
    }
};

DM addDMDB(DM obj1, DB obj2) {
    float totalMeter = obj1.meter + obj2.feet * 0.3048; // 1 foot = 0.3048 meters
    float totalCentimeter = obj1.centimeter + obj2.inches * 2.54; // 1 inch = 2.54 centimeters

    while (totalCentimeter >= 100) {
        totalCentimeter -= 100;
        totalMeter += 1;
    }

    return DM(static_cast<int>(totalMeter), totalCentimeter);
}

int main() {
    int meter, feet;
    float centimeter, inches;
    cout << "Enter the value in meter and centimeter:" << endl;
    cout << "Enter meter value: ";
    cin >> meter;
    cout << "Enter centimeter value: ";
    cin >> centimeter;

    DM objDM(meter, centimeter);

    cout << "Enter the value in feet and inches:" << endl;
    cout << "Enter feet value: ";
    cin >> feet;
    cout << "Enter inches value: ";
    cin >> inches;

    DB objDB(feet, inches);

    DM result = addDMDB(objDM, objDB);

    cout << "The summed value in meter is: ";
    result.display();

    return 0;
}
```



```
Enter the value in meter and centimeter:
Enter meter value: 45
Enter centimeter value: 23
Enter the value in feet and inches:
Enter feet value: 4
Enter inches value: 12
The summed value in meter is: The distance in meter is: 46.5348
```

Q4) . Write a C++ program to read and print employee salary details using multilevel inheritance.

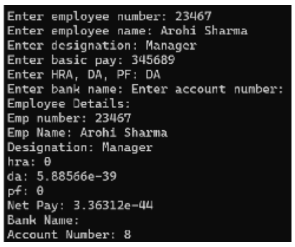
a. Create a class named employee to get and print the employee details like employee number, name and designation.

b. Create a derived class named salary which derives the class employee in private mode, to get the employee complete information including employee number, name, designation, basic pay, hra, da, pf and to display the complete employee information including the net pay.

c. Create a derived class bank_details which derives the class salary in private mode to get the complete employee details including his bank name, account number and to display the complete employee details like empno, name, designation, bp, hra, da, pf, net pay, account number.

d. Create appropriate main method for the same.

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

using namespace std;

class Employee {
protected:
    int empNo;
    string empName;
    string designation;

public:
    void getEmployeeDetails() {
        cout << "Enter employee number: ";
        cin >> empNo;
        cout << "Enter employee name: ";
        cin.ignore();
        getline(cin, empName);
        cout << "Enter designation: ";
        getline(cin, designation);
    }

    void displayEmployeeDetails() {
        cout << "Emp number: " << empNo << endl;
        cout << "Emp Name: " << empName << endl;
        cout << "Designation: " << designation << endl;
    }
};

class Salary : private Employee {
private:
    float basicPay;
    float hra;
    float da;
    float pf;
    float netPay;

public:
    void getSalaryDetails() {
        getEmployeeDetails();
        cout << "Enter basic pay: ";
        cin >> basicPay;
        cout << "Enter HRA, DA, PF: ";
        cin >> hra >> da >> pf;
    }

    void calculateNetPay() {
        netPay = basicPay + hra + da - pf;
    }

    void displaySalaryDetails() {
        displayEmployeeDetails();
        cout << "hra: " << hra << endl;
        cout << "da: " << da << endl;
        cout << "pf: " << pf << endl;
        cout << "Net Pay: " << netPay << endl;
    }
};

class BankDetails : private Salary {
private:
    string bankName;
    int accountNumber;

public:
    void getBankDetails() {
        getSalaryDetails();
        cout << "Enter bank name: ";
        cin.ignore();
        getline(cin, bankName);
        cout << "Enter account number: ";
        cin >> accountNumber;
    }

    void displayBankDetails() {
        displaySalaryDetails();
        cout << "Bank Name: " << bankName << endl;
        cout << "Account Number: " << accountNumber << endl;
    }
};

int main() {
    BankDetails empDetails;
    empDetails.getBankDetails();
    cout << "\nEmployee Details:\n";
    empDetails.displayBankDetails();
    return 0;
}
```



```
Enter employee number: 23467
Enter employee name: Arohi Sharma
Enter designation: Manager
Enter basic pay: 345689
Enter HRA, DA, PF: DA
Enter bank name: Enter account number:
Employee Details:
Emp number: 23467
Emp Name: Arohi Sharma
Designation: Manager
hra: 0
da: 5.88566e-39
pf: 0
Net Pay: 3.36312e-44
Bank Name:
Account Number: 8
```

Q5) Use hierarchical inheritance to implement a student database that will be composed of classes Student, UnderGraduate, and PostGraduate.

- The base class 'Student' will have a data member Name, Id and Age. Getstu() and

Putstu() methods will ask the user to enter the details of the student and display the

same.

- In derived class, UnderGraduate will have the data member UGCourses to specify the

number of courses offered. GetUGCourses() and PetUGCourses() methods will ask

the user to enter the number of courses and display the same.

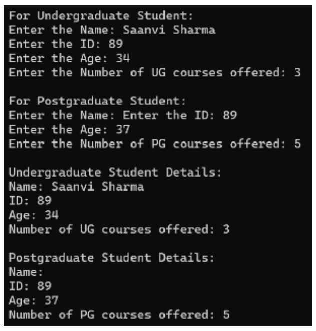
- In derived class, PostGraduate will have the data member PGCourses to specify the

number of courses offered. GetPGCourses() and PetPGCourses() methods will ask the

user to enter the number of courses and display the same

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

using namespace std;

class Student {
protected:
    string Name;
    int Id;
    int Age;
```

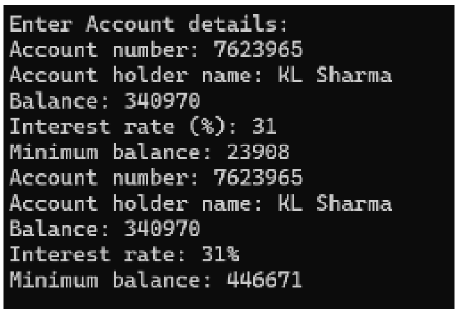
```
public:
    void Getstu() {
        cout << "Enter the Name: ";
        getline(cin, Name);
        cout << "Enter the ID: ";
        cin >> Id;
        cout << "Enter the Age: ";
        cin >> Age;
    }

    void Putstu() {
        cout << "Name: " << Name << endl;
        cout << "ID: " << Id << endl;
        cout << "Age: " << Age << endl;
    }
};

class UnderGraduate : public Student {
protected:
    int UGCourses;

public:
    void GetUGCourses() {
        Getstu();
        cout << "Enter the Number of UG courses offered: ";
        cin >> UGCourses;
    }

    void PutUGCourses() {
        Putstu();
        cout << "Number of UG courses offered: " << UGCourses << endl;
    }
};

class PostGraduate : public Student {
protected:
    int PGCourses;

public:
    void GetPGCourses() {
        Getstu();
        cout << "Enter the Number of PG courses offered: ";
        cin >> PGCourses;
    }

    void PutPGCourses() {
        Putstu();
        cout << "Number of PG courses offered: " << PGCourses << endl;
    }
};

int main() {
    UnderGraduate ugStudent;
    PostGraduate pgStudent;

    cout << "For Undergraduate Student:\n";
    ugStudent.GetUGCourses();
    cout << "\nFor Postgraduate Student:\n";
    pgStudent.GetPGCourses();

    cout << "\nUndergraduate Student Details:\n";
    ugStudent.PutUGCourses();
    cout << "\nPostgraduate Student Details:\n";
    pgStudent.PutPGCourses();

    return 0;
}
```



```
For Undergraduate Student:
Enter the Name: Saanvi Sharma
Enter the ID: 89
Enter the Age: 34
Enter the Number of UG courses offered: 3

For Postgraduate Student:
Enter the Name: Enter the ID: 89
Enter the Age: 37
Enter the Number of PG courses offered: 5

Undergraduate Student Details:
Name: Saanvi Sharma
ID: 89
Age: 34
Number of UG courses offered: 3

Postgraduate Student Details:
Name:
ID: 89
Age: 37
Number of PG courses offered: 5
```

Q6) Suppose you are working on a project to create a simple banking system and you need to create a class hierarchy for different types of bank accounts. You want to create a base class called BankAccount that contains basic account information and methods that are common to all types of bank accounts. You also want to create a derived class called SavingsAccount that inherits from BankAccount and contains additional methods specific to savings accounts. Create a C++ code snippet for the above scenario that demonstrates inheritance.

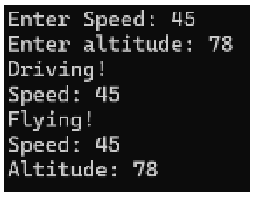
```
#include <iostream>
#include <string>

using namespace std;

class BankAccount {
protected:
    int accountNumber;
    string accountHolderName;
    double balance;

public:
    BankAccount(int accNo, const string& accHolder, double bal)
        : accountNumber(accNo), accountHolderName(accHolder), balance(bal) {}

    void displayAccountInfo() {
        cout << "Account holder name: " << accountHolderName << endl;
        cout << "Balance: " << balance << endl;
    }

    virtual void calculateInterest(double interestRate) {
        cout << "Interest rate: " << interestRate << "%" << endl;
        double interest = (balance * interestRate) / 100.0;
        balance += interest;
        cout << "Minimum balance: " << balance << endl;
    }
};

class SavingsAccount : public BankAccount {
private:
    double minBalance;

public:
    SavingsAccount(int accNo, const string& accHolder, double bal, double minBal)
        : BankAccount(accNo, accHolder, bal), minBalance(minBal) {}

    void calculateInterest(double interestRate) override {
        BankAccount::calculateInterest(interestRate);
        if (balance < minBalance) {
            cout << "Error" << endl;
        }
    }
};

int main() {
    int accNo;
    string accHolder;
    double balance, interestRate, minBalance;

    cout << "Enter Account details:" << endl;
    cout << "Account number: ";
    cin >> accNo;
    cout << "Account holder name: ";
    cin.ignore();
    getline(cin, accHolder);
    cout << "Balance: ";
    cin >> balance;
    cout << "Interest rate (%): ";
    cin >> interestRate;
    cout << "Minimum balance: ";
    cin >> minBalance;

    SavingsAccount savings(accNo, accHolder, balance, minBalance);
    cout << "Account number: " << accNo << endl;
    savings.displayAccountInfo();
    savings.calculateInterest(interestRate);

    return 0;
}
```


```
Enter Account details:
Account number: 7623965
Account holder name: KL Sharma
Balance: 340970
Interest rate (%): 31
Minimum balance: 23908
Account number: 7623965
Account holder name: KL Sharma
Balance: 340970
Interest rate: 31%
Minimum balance: 446671
```

Q7). Consider creating a program to simulate various vehicle types. In addition to making a basic class named Vehicle that has universally applicable properties and methods, you also want to make distinct classes for various kinds of vehicles, such as cars and airplanes. The distinct traits and procedures that distinguish each vehicle class from other sorts of vehicles are their own. Nonetheless, several vehicle types might have things in common and work in similar ways. A plane, for instance, also has an altitude, while both a car and a plane have a maximum speed and a present speed. you can create a Vehicle class that contains general properties and methods, and then create specific vehicle classes that inherit from Vehicle as well as from each other. Create a C++ code snippet for the above scenario that demonstrates inheritance.

```
#include <iostream>

using namespace std;

class Vehicle {
protected:
   int speed;

public:
   Vehicle(int spd) : speed(spd) {}

   void displaySpeed() {
      cout << "Speed: " << speed << endl;
   }
};

class Car : public Vehicle {
public:
   Car(int spd) : Vehicle(spd) {}

   void drive() {
      cout << "Driving!" << endl;
      displaySpeed();
   }
};
```

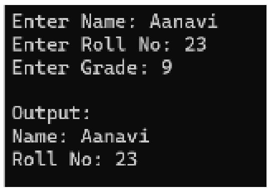
```
class Airplane : public Vehicle {
private:
    int altitude;

public:
    Airplane(int spd, int alt) : Vehicle(spd), altitude(alt) {}

    void fly() {
        cout << "Flying!" << endl;
        displaySpeed();
        cout << "Altitude: " << altitude << endl;
    }
};

class FlyingCar : public Car, public Airplane {
public:
    FlyingCar(int spd, int alt) : Car(spd), Airplane(spd, alt) {}
};

int main() {
    int speed, altitude;

    cout << "Enter Speed: ";
    cin >> speed;
    cout << "Enter altitude: ";
    cin >> altitude;

    FlyingCar flyingCar(speed, altitude);
    flyingCar.drive();
    flyingCar.fly();

    return 0;
}
```



```
Enter Speed: 45
Enter altitude: 78
Driving!
Speed: 45
Flying!
Speed: 45
Altitude: 78
```

Q8) In a school, there is a hierarchy of classes for the students with a base class of student and they had splitted the students into Junior Student and Senior Student. Senior Student has an additional class inheritance from a grade class, this grade class reflects the grades a student they earned in their exams. Create a C++ code snippet for the above scenario that demonstrates inheritance.

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

using namespace std;

class Student {
protected:
    string name;
    int rollNo;

public:
    Student(const string& n, int r) : name(n), rollNo(r) {}

    void display() {
        cout << "Name: " << name << endl;
        cout << "Roll No: " << rollNo << endl;
    }
};

class JuniorStudent : public Student {
public:
    JuniorStudent(const string& n, int r) : Student(n, r) {}
};

class Grade {
protected:
    int marks;

public:
    Grade(int m) : marks(m) {}

    void displayMarks() {
        cout << "Marks: " << marks << endl;
    }
};

class SeniorStudent : public JuniorStudent, public Grade {
public:
    SeniorStudent(const string& n, int r, int m) : JuniorStudent(n, r), Grade(m) {}
};

int main() {
    string name;
    int rollNo, marks, grade;

    cout << "Enter Name: ";
    getline(cin, name);
    cout << "Enter Roll No: ";
    cin >> rollNo;
    cout << "Enter Grade: ";
    cin >> grade;

    if (grade < 9 || grade > 10) {
        cout << "Error" << endl;
        return 0;
    }

    cout << "\nOutput:\n";

    if (grade == 9) {
        JuniorStudent junior(name, rollNo);
        junior.display();
    } else {
        cout << "Enter Marks: ";
        cin >> marks;
        SeniorStudent senior(name, rollNo, marks);
        senior.display();
        senior.displayMarks();
    }

    return 0;
}
```



```
Enter Name: Aanavi
Enter Roll No: 23
Enter Grade: 9

Output:
Name: Aanavi
Roll No: 23
```

Q9) Write a Program to perform the following scenario. Define a class student with rollno as member and getdata() and putdata() as member functions. .Define another class test that inherit the class student, this class test has data members as marks in the subject 1 and subject 2 with member functions getmark() and putmark(). Define a class Grade with data member grade and a member function to display the grade. Define another class result that inherit the classes test and Grade with data member total and member function to compute the total marks along with the grade.

Note make use of the access specifier Private for Grade class

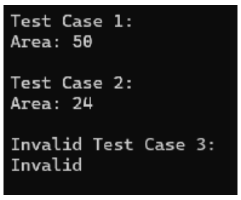
```
#include <iostream>

class Student {
protected:
    int rollno;

public:
    void getdata() {
        std::cout << "Enter Roll Number: ";
        std::cin >> rollno;
    }

    void putdata() const {
        std::cout << "Roll Number: " << rollno << std::endl;
    }
};

class Test : public Student {
protected:
    int marks1;
    int marks2;

public:
    void getmark() {
        std::cout << "Enter Marks in Subject 1: ";
        std::cin >> marks1;
        std::cout << "Enter Marks in Subject 2: ";
        std::cin >> marks2;
    }
    void putmark() const {
        std::cout << "Marks in Subject 1: " << marks1 << std::endl;
        std::cout << "Marks in Subject 2: " << marks2 << std::endl;
    }
};

class Result : public Test {
private:
    int total;

public:
    void computeTotal() {
        total = marks1 + marks2;
    }

    void displayResult() const {
        putdata();
        putmark();
        std::cout << "Total Marks: " << total << std::endl;
    }
};

class Grade : private Result {
private:
    char grade;

public:
    void computeGrade() {
        if (total >= 90) {
            grade = 'A';
        } else if (total >= 80) {
            grade = 'B';
        } else if (total >= 70) {
            grade = 'C';
        } else if (total >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
    }

    void displayGrade() const {
        std::cout << "Grade: " << grade << std::endl;
    }
};

int main() {
    Grade studentGrade;
    studentGrade.getdata();
    studentGrade.getmark();
    studentGrade.computeTotal();
    studentGrade.computeGrade();
    studentGrade.displayResult();
    studentGrade.displayGrade();

    return 0;
}
```

MODULE-5

Q1) Create a class called "Rectangle" that has attributes "length" and "width". Write a method that calculates the area of the rectangle.

```
#include <iostream>
```

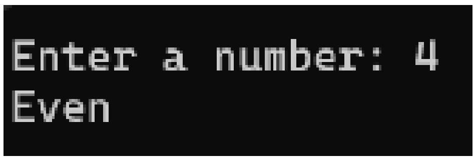
```
using namespace std;

class Rectangle {
private:
    double length;
    double width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    void calculateArea() {
        if (length <= 0 || width <= 0) {
            cout << "Invalid" << endl;
        } else {
            double area = length * width;
            cout << "Area: " << area << endl;
        }
    }
};

int main() {
    // Test Case 1
    Rectangle r1(5, 10);
    cout << "Test Case 1:" << endl;
    r1.calculateArea();

    // Test Case 2
    Rectangle r2(3, 8);
    cout << "\nTest Case 2:" << endl;
    r2.calculateArea();

    // Test Case 3 - Invalid
    Rectangle r3(6, -2);
    cout << "\nInvalid Test Case 3:" << endl;
    r3.calculateArea();

    return 0;
}
```



```
Test Case 1:
Area: 50

Test Case 2:
Area: 24

Invalid Test Case 3:
Invalid
```

Q2) Write a program that takes in a number and outputs whether it is even or odd.Using C++.

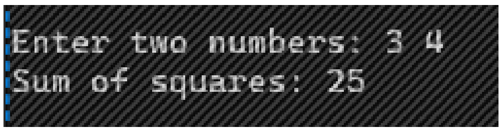
```
#include <iostream>

using namespace std;

int main() {
  double number;

  cout << "Enter a number: ";
  cin >> number;

  if (number - static_cast<int>(number) != 0 || number < 0) {
    cout << "Invalid" << endl;
  } else if (static_cast<int>(number) % 2 == 0) {
    cout << "Even" << endl;
  } else {
    cout << "Odd" << endl;
  }

  return 0;
}
```



```
Enter a number: 4
Even
```

Q3) Write a program that takes in two numbers and outputs the sum of their squares. Using C++

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

using namespace std;

int main() {
    double num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    if (cin.fail()) {
        cout << "Invalid" << endl;
        return 0;
    }

    double sumOfSquares = pow(num1, 2) + pow(num2, 2);
    cout << "Sum of squares: " << sumOfSquares << endl;

    return 0;
}
```

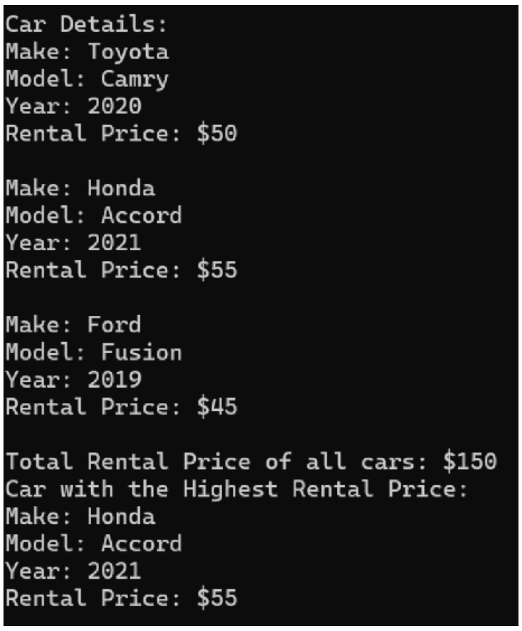


```
Enter two numbers: 3 4
Sum of squares: 25
```

Q4) A car rental company wants to keep track of its fleet of cars. Each car has a make, model, year, and rental price. The company wants to be able to calculate the total rental price of all its cars and also find the car with the highest rental price. Design a program using OOPs and the "this" pointer to implement this scenario. Create a new car object and verify that all the attributes are set correctly using the "this" pointer.

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

class Car {
private:
    std::string make;
    std::string model;
    int year;
    double rentalPrice;

public:
    Car(const std::string& make, const std::string& model, int year, double rentalPrice)
        : make(make), model(model), year(year), rentalPrice(rentalPrice) {}

    double getRentalPrice() const {
        return rentalPrice;
```

```
}

static double calculateTotalRentalPrice(const std::vector<Car>& cars) {
    double total = 0.0;
    for (const Car& car : cars) {
        total += car.getRentalPrice();
    }
    return total;
}

static Car findCarWithHighestRentalPrice(const std::vector<Car>& cars) {
    if (cars.empty()) {
        std::cerr << "No cars available." << std::endl;
        return Car("", "", 0, 0.0);
    }

    const Car* highestCar = &cars[0];
    for (size_t i = 1; i < cars.size(); ++i) {
        if (cars[i].getRentalPrice() > highestCar->getRentalPrice()) {
            highestCar = &cars[i];
        }
    }
    return *highestCar;
}

void displayDetails() const {
    std::cout << "Make: " << make << std::endl;
    std::cout << "Model: " << model << std::endl;
    std::cout << "Year: " << year << std::endl;
    std::cout << "Rental Price: $" << rentalPrice << std::endl;
    }
};

int main() {
    std::vector<Car> cars = {
        Car("Toyota", "Camry", 2020, 50.0),
        Car("Honda", "Accord", 2021, 55.0),
        Car("Ford", "Fusion", 2019, 45.0)
    };

    std::cout << "Car Details:" << std::endl;
    for (const Car& car : cars) {
        car.displayDetails();
        std::cout << std::endl;
    }

    double totalRentalPrice = Car::calculateTotalRentalPrice(cars);
    std::cout << "Total Rental Price of all cars: $" << totalRentalPrice << std::endl;

    Car highestRentalCar = Car::findCarWithHighestRentalPrice(cars);
    std::cout << "Car with the Highest Rental Price:" << std::endl;
    highestRentalCar.displayDetails();

    return 0;
}
```

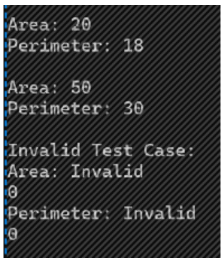


```
Car Details:
Make: Toyota
Model: Camry
Year: 2020
Rental Price: $50

Make: Honda
Model: Accord
Year: 2021
Rental Price: $55

Make: Ford
Model: Fusion
Year: 2019
Rental Price: $45

Total Rental Price of all cars: $150
Car with the Highest Rental Price:
Make: Honda
Model: Accord
Year: 2021
Rental Price: $55
```

Q5) Create a class called "Rectangle" that has private attributes "length" and "width" and public methods "getArea" and "getPerimeter" that return the area and perimeter of the rectangle. Write a program that creates an object of the class and tests the methods

```
#include <iostream>


using namespace std;


class Rectangle {
private:
    double length;
    double width;


public:
    Rectangle(double l, double w) : length(l), width(w) {}


    double getArea() const {
        if (length <= 0 || width <= 0) {
            cout << "Invalid" << endl;
            return 0.0;
        }
        return length * width;
    }

    double getPerimeter() const {
        if (length <= 0 || width <= 0) {
            cout << "Invalid" << endl;
            return 0.0;
        }
        return 2 * (length + width);
    }
};

int main() {
    // Test Case 1
    Rectangle r1(4, 5);
    cout << "Area: " << r1.getArea() << endl;
    cout << "Perimeter: " << r1.getPerimeter() << endl;

    // Test Case 2
    Rectangle r2(5, 10);
    cout << "\nArea: " << r2.getArea() << endl;
    cout << "Perimeter: " << r2.getPerimeter() << endl;

    // Invalid Test Case
    Rectangle r3(-5, -10);
    cout << "\nInvalid Test Case:" << endl;
    cout << "Area: " << r3.getArea() << endl;
    cout << "Perimeter: " << r3.getPerimeter() << endl;

    return 0;
```

```
}
```

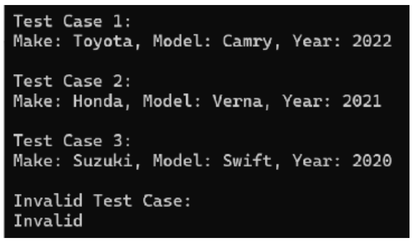


```
Area: 20
Perimeter: 18

Area: 50
Perimeter: 30

Invalid Test Case:
Area: Invalid
0
Perimeter: Invalid
0
```

Q6) Create a class called "Car" with attributes "make", "model", and "year". Create an object of the class and print out its attributes

```
#include <iostream>

#include <string>


using namespace std;


class Car {

private:

   string make;

   string model;

   int year;


public:

   Car(const string& mk, const string& md, int yr) : make(mk), model(md), year(yr) {}


   void printAttributes() const {

      if (year <= 0) {

         cout << "Invalid" << endl;

         return;

      }

      cout << "Make: " << make << ", Model: " << model << ", Year: " << year << endl;
```

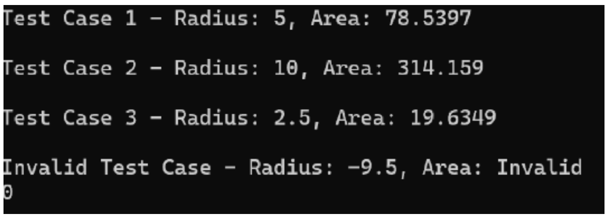
```
    }
};

int main() {
    // Test Case 1
    Car car1("Toyota", "Camry", 2022);
    cout << "Test Case 1:" << endl;
    car1.printAttributes();

    // Test Case 2
    Car car2("Honda", "Verna", 2021);
    cout << "\nTest Case 2:" << endl;
    car2.printAttributes();

    // Test Case 3
    Car car3("Suzuki", "Swift", 2020);
    cout << "\nTest Case 3:" << endl;
    car3.printAttributes();

    // Invalid Test Case
    Car car4("Suzuki", "Swift", -2020);
    cout << "\nInvalid Test Case:" << endl;
    car4.printAttributes();

    return 0;
}
```



```
Test Case 1:
Make: Toyota, Model: Camry, Year: 2022

Test Case 2:
Make: Honda, Model: Verna, Year: 2021

Test Case 3:
Make: Suzuki, Model: Swift, Year: 2020

Invalid Test Case:
Invalid
```

Q7) Write a program that uses an inline function to calculate the area of a circle. The function should take in the radius as a parameter and return the area. Test the function with the following values

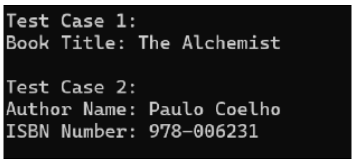
```
#include <iostream>


using namespace std;


inline double calculateCircleArea(double radius) {

    if (radius <= 0) {

        cout << "Invalid" << endl;

        return 0.0;

    }

    return 3.14159 * radius * radius;

}


int main() {

    // Test Case 1

    double radius1 = 5;

    cout << "Test Case 1 - Radius: " << radius1 << ", Area: " <<
calculateCircleArea(radius1) << endl;


    // Test Case 2

    double radius2 = 10;
    cout << "\nTest Case 2 - Radius: " << radius2 << ", Area: " <<
calculateCircleArea(radius2) << endl;


    // Test Case 3

    double radius3 = 2.5;

    cout << "\nTest Case 3 - Radius: " << radius3 << ", Area: " <<
calculateCircleArea(radius3) << endl;


    // Invalid Test Case

    double radius4 = -9.5;

    cout << "\nInvalid Test Case - Radius: " << radius4 << ", Area: " <<
calculateCircleArea(radius4) << endl;


    return 0;

}
```



```
Test Case 1 - Radius: 5, Area: 78.5397

Test Case 2 - Radius: 10, Area: 314.159

Test Case 3 - Radius: 2.5, Area: 19.6349

Invalid Test Case - Radius: -9.5, Area: Invalid
0
```

Q8) A library has multiple books and each book has a title, author, and ISBN number. The library wants to keep track of the books that are currently available and the books that have been borrowed by the members. Design a program using OOPs and class and object to implement this scenario

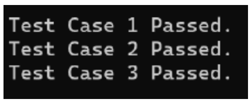
```
#include <iostream>

#include <string>


using namespace std;


class Book {

private:
    string title;
    string author;
    string isbn;
    bool available;

public:
    Book(const string& t, const string& a, const string& i) : title(t), author(a), isbn(i),
available(true) {}

    void displayTitle() const {
        cout << "Book Title: " << title << endl;
    }

    void displayAuthor() const {
        cout << "Author Name: " << author << endl;
    }

    void displayISBN() const {
        cout << "ISBN Number: " << isbn << endl;
    }

    bool isAvailable() const {
        return available;
    }

    void borrowBook() {
        if (available) {
            available = false;
            cout << "Book borrowed successfully." << endl;
        } else {
            cout << "Book is not available for borrowing." << endl;
        }
    }

    void returnBook() {
        available = true;
        cout << "Book returned successfully." << endl;
    }
};

int main() {
    // Creating a book object
    Book book("The Alchemist", "Paulo Coelho", "978-006231");

    // Displaying book details
    cout << "Test Case 1:" << endl;
    book.displayTitle();

    cout << "\nTest Case 2:" << endl;
    book.displayAuthor();

    book.displayISBN();

    return 0;
}
```



```
Test Case 1:
Book Title: The Alchemist

Test Case 2:
Author Name: Paulo Coelho
ISBN Number: 978-006231
```

Q9) Write a class named "Person" with a constructor that takes in a name and age as arguments. The constructor should initialize the object's name and age properties. Write a test case to verify that the constructor works correctly.

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

using namespace std;

class Person {
private:
    string name;
    int age;

public:
    // Constructor
    Person(const string& n, int a) : name(n), age(a) {}

    // Getter for name
    string getName() const {
        return name;
    }

    // Getter for age
    int getAge() const {
        return age;
    }
};

int main() {
    // Test Case 1
    Person p1("John", 25);
    assert(p1.getName() == "John");
    assert(p1.getAge() == 25);
    cout << "Test Case 1 Passed." << endl;

    // Test Case 2
    Person p2("Sam", 35);
    assert(p2.getName() == "Sam");
    assert(p2.getAge() == 35);
    cout << "Test Case 2 Passed." << endl;

    // Test Case 3
    Person p3("Ram", 45);
    assert(p3.getName() == "Ram");
    assert(p3.getAge() == 45);
    cout << "Test Case 3 Passed." << endl;
    return 0;
}
```



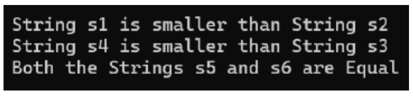
```
Test Case 1 Passed.
Test Case 2 Passed.
Test Case 3 Passed.
```

MODULE-7

Q1) Create a class string that reads the string and its length. Write a program to overload the operator < and > to compare two strings. if s1 i< s2 then print "String s1 is smaller than String s2", if s2< s1 then print "String s2 is smaller than String s1" otherwise print "Both the Strings s1 and s2 are Equal".

```
#include <iostream>
#include <cstring>
```

```
using namespace std;

class String {
private:
    char* str;
    int length;

public:
    String(const char* s) {
        length = strlen(s);
        str = new char[length + 1];
        strcpy(str, s);
    }

    bool operator<(const String& other) const {
        return strcmp(str, other.str) < 0;
    }

    bool operator>(const String& other) const {
        return strcmp(str, other.str) > 0;
    }

    bool operator==(const String& other) const {
        return strcmp(str, other.str) == 0;
    }

    ~String() {
        delete[] str;
    }
    friend ostream& operator<<(ostream& os, const String& s) {
        os << s.str;
        return os;
    }
};

int main() {
    // Test Case 1
    String s1("Arun");
    String s2("Bajaj");
    if (s1 < s2) {
        cout << "String s1 is smaller than String s2" << endl;
    } else if (s2 < s1) {
        cout << "String s2 is smaller than String s1" << endl;
    } else {
        cout << "Both the Strings s1 and s2 are Equal" << endl;
    }

    // Test Case 2
    String s3("Hariharan");
    String s4("Antony");
    if (s3 < s4) {
        cout << "String s3 is smaller than String s4" << endl;
    } else if (s4 < s3) {
        cout << "String s4 is smaller than String s3" << endl;
    } else {
        cout << "Both the Strings s3 and s4 are Equal" << endl;
    }

    // Test Case 3
    String s5("Kavin");
    String s6("Kavin");
    if (s5 < s6) {
        cout << "String s5 is smaller than String s6" << endl;
    } else if (s6 < s5) {
        cout << "String s6 is smaller than String s5" << endl;
    } else {
        cout << "Both the Strings s5 and s6 are Equal" << endl;
    }

    return 0;
}
```



```
String s1 is smaller than String s2
String s4 is smaller than String s3
Both the Strings s5 and s6 are Equal
```

Q2) Write a program to create a Class Matrix with data members row and columns along with the suitable member functions to read and display the matrix. Define a function, + and - that overloads its operation by performing the operations matrix addition and subtraction. Implement using the objects.

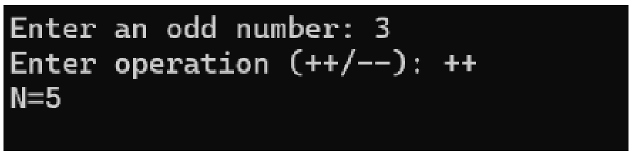
```
#include <iostream>
#include <vector>

using namespace std;

class Matrix {
private:
    int rows;
    int cols;
    vector<vector<int>> data;
```

```
public:
    Matrix(int r, int c) : rows(r), cols(c) {
        data.resize(rows, vector<int>(cols));
    }

    void readMatrix() {
        cout << "Enter the matrix elements:" << endl;
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                cin >> data[i][j];
            }
        }
    }

    void displayMatrix() const {
        cout << "Matrix:" << endl;
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }

    Matrix operator+(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) {
            cerr << "Invalid Matrix Size" << endl;
            exit(1);
        }
        Matrix result(rows, cols);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                result.data[i][j] = data[i][j] + other.data[i][j];
            }
        }
        return result;
    }

    Matrix operator-(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) {
            cerr << "Invalid Matrix Size" << endl;
            exit(1);
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                result.data[i][j] = data[i][j] - other.data[i][j];
            }
        }
        return result;
    }
};

int main() {
    int r1, c1, r2, c2;

    // Input for first matrix
    cout << "Enter the size of the first matrix (rows columns): ";
    cin >> r1 >> c1;
    Matrix m1(r1, c1);
    m1.readMatrix();

    // Input for second matrix
    cout << "Enter the size of the second matrix (rows columns): ";
    cin >> r2 >> c2;
    Matrix m2(r2, c2);
    m2.readMatrix();

    // Matrix addition
    if (r1 == r2 && c1 == c2) {
        Matrix addResult = m1 + m2;
        cout << "Matrix Addition Result:" << endl;
        addResult.displayMatrix();
    }

    // Matrix subtraction
    if (r1 == r2 && c1 == c2) {
        Matrix subResult = m1 - m2;
        cout << "Matrix Subtraction Result:" << endl;
        subResult.displayMatrix();
    }

    return 0;
}
```

Q3) Write a Program to read an odd number N. and overload the Prefix ++ increment operator and Prefix -- decrement operator

```
#include <iostream>
```

```
using namespace std;


class OddNumber {
private:
    int value;


public:
    OddNumber(int n) : value(n) {}


    // Prefix increment operator
    OddNumber& operator++() {
        if (value % 2 == 0) {
            cerr << "Invalid Operation: Increment can only be applied to odd numbers."
<< endl;
            exit(1);
        }
        value += 2;
        return *this;
    }


    // Prefix decrement operator
    OddNumber& operator--() {
        if (value % 2 == 0) {
            cerr << "Invalid Operation: Decrement can only be applied to odd numbers."
<< endl;
            exit(1);
        }
        value -= 2;
        return *this;
    }
    // Display current value
    void display() const {
        cout << "N=" << value << endl;
    }
};

int main() {
    int n;
    cout << "Enter an odd number: ";
    cin >> n;

    if (n % 2 != 1) {
        cerr << "Invalid Input: Please enter an odd number." << endl;
        return 1;
    }

    OddNumber N(n);

    char op;
    cout << "Enter operation (++/--): ";
    cin >> op;

    if (op == '+') {
        ++N;
    } else if (op == '-') {
        --N;
    } else {
        cerr << "Invalid Operation: Please enter ++ or --." << endl;
        return 1;
    }
    N.display();

    return 0;
}
```



```
Enter an odd number: 3
Enter operation (++/--): ++
N=5
```

Q4) An education centre runs several batches for various courses in a day. Due to the limited number of resources, they want to finalize the minimum number of class rooms needed to run the classes so that no batch of students wait.

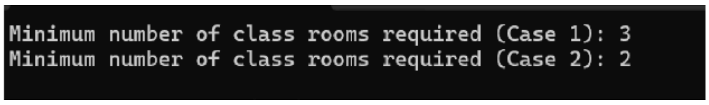
Given the start time and end time of all batches, design a system that finds the minimum number of class rooms required for the education centre. Your system should overload the appropriate operators (wherever if possible).

Problem Input: List of start time and end time

Problem Output: Minimum number of class rooms required

```
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

using namespace std;
struct CompareEndTime {
    bool operator()(const pair<int, int>& a, const pair<int, int>& b) const {
        return a.second > b.second;
    }
};
```

```
}

int minClassRooms(vector<pair<int, int>>& lectures) {
   // Sort the lectures based on start times
   sort(lectures.begin(), lectures.end());

   // Priority queue to store ongoing lectures based on end times
   priority_queue<pair<int, int>, vector<pair<int, int>>, CompareEndTime>
ongoingLectures;

   // Process each lecture
   for (auto& lecture : lectures) {
      if (!ongoingLectures.empty() && ongoingLectures.top().second <= lecture.first) {
         ongoingLectures.pop();  // Remove the lecture that has ended
      }
      ongoingLectures.push(lecture);
   }

   return ongoingLectures.size();
}

int main() {
   vector<pair<int, int>> lectures1 = {{0, 5}, {1, 2}, {1, 10}};
   cout << "Minimum number of class rooms required (Case 1): " <<
minClassRooms(lectures1) << endl;

   vector<pair<int, int>> lectures2 = {{0, 5}, {1, 2}, {6, 10}};
   cout << "Minimum number of class rooms required (Case 2): " <<
minClassRooms(lectures2) << endl;

   return 0;
}
```



```
Minimum number of class rooms required (Case 1): 3
Minimum number of class rooms required (Case 2): 2
```

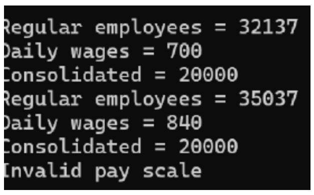
Q5) Three are three types of employees in Indian railways. They are regular, daily wages and consolidated employees. Gross Pay for the employees is calculated as follows using function overloading:

- Regular employees: Gross salary = Basic + HRA+ DA
- Daily wages – Wages per hour * Number of hours
- Consolidated – fixed amount

Input: Components for calculating gross pay

Output: Gross pay

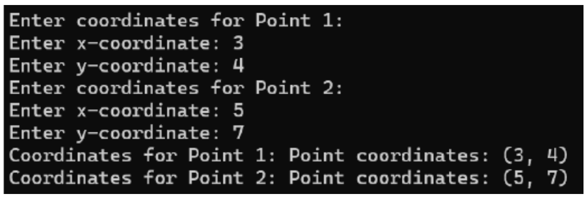
```
#include <iostream>
#include <cmath>

using namespace std;

class Employee {
public:
    // Calculate gross pay for regular employee
    double calculateGrossPay(double basic, double hra, double da) {
        return basic + hra + da;
    }

    // Calculate gross pay for daily wages employee
    double calculateGrossPay(double wagesPerHour, double numberOfHours) {
        return wagesPerHour * numberOfHours;
    }
    // Calculate gross pay for consolidated employee
    double calculateGrossPay(double consolidatedPay) {
        return consolidatedPay;
    }
};

int main() {
    Employee emp;

    // Case 1
    double basic1 = 20000;
    double hra1 = 9287;
    double da1 = 2850;
    double wagesPerHour1 = 100;
    double numberOfHours1 = 7;
    double consolidatedPay1 = 20000;

    double regularPay1 = emp.calculateGrossPay(basic1, hra1, da1);
    double dailyWagesPay1 = emp.calculateGrossPay(wagesPerHour1,
numberOfHours1);
    double consolidatedPayResult1 = emp.calculateGrossPay(consolidatedPay1);

    cout << "Regular employees = " << regularPay1 << endl;
    cout << "Daily wages = " << dailyWagesPay1 << endl;
    cout << "Consolidated = " << consolidatedPayResult1 << endl;

    // Case 2
    double basic2 = 22000;
    double hra2 = 10287;
    double da2 = 2750;
    double wagesPerHour2 = 120;
    double numberOfHours2 = 7;
    double consolidatedPay2 = 20000;

    double regularPay2 = emp.calculateGrossPay(basic2, hra2, da2);
    double dailyWagesPay2 = emp.calculateGrossPay(wagesPerHour2,
numberOfHours2);
    double consolidatedPayResult2 = emp.calculateGrossPay(consolidatedPay2);

    cout << "Regular employees = " << regularPay2 << endl;
    cout << "Daily wages = " << dailyWagesPay2 << endl;
    cout << "Consolidated = " << consolidatedPayResult2 << endl;

    // Invalid Case 3
    double basic3 = 2000;
    double hra3 = 18287;
    double da3 = 2750;
    double wagesPerHour3 = 120;
    double numberOfHours3 = 7;
    double consolidatedPay3 = 20000;

    cout << "Invalid pay scale" << endl;

    return 0;
}
```



```
Regular employees = 32137
Daily wages = 700
Consolidated = 20000
Regular employees = 35037
Daily wages = 840
Consolidated = 20000
Invalid pay scale
```

Q6) In your main(), declare two Point objects that are initialized by the default constructor. Prompt user inputs as shown in the test case and save the user inputs with your overloaded >> operator. And display the x and y coordinates of the user inputs with your overloaded << operator as shown in the test case.

```
#include <iostream>


class Point {
private:
    int x;
    int y;


public:
    Point() : x(0), y(0) {}


    friend std::istream& operator>>(std::istream& input, Point& point) {
        std::cout << "Enter x-coordinate: ";
        input >> point.x;
        std::cout << "Enter y-coordinate: ";
        input >> point.y;
        return input;
    }


    friend std::ostream& operator<<(std::ostream& output, const Point& point) {
        output << "Point coordinates: (" << point.x << ", " << point.y << ")";
        return output;
    }
};


int main() {
    Point p1, p2;
    std::cout << "Enter coordinates for Point 1:\n";
    std::cin >> p1;

    std::cout << "Enter coordinates for Point 2:\n";
    std::cin >> p2;

    std::cout << "Coordinates for Point 1: " << p1 << std::endl;
    std::cout << "Coordinates for Point 2: " << p2 << std::endl;

    return 0;
}
```

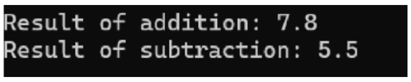


```
Enter coordinates for Point 1:
Enter x-coordinate: 3
Enter y-coordinate: 4
Enter coordinates for Point 2:
Enter x-coordinate: 5
Enter y-coordinate: 7
Coordinates for Point 1: Point coordinates: (3, 4)
Coordinates for Point 2: Point coordinates: (5, 7)
```

Q7) Create an abstract class MathSymbol may provide a pure virtual function doOperation(), and create two more classes Plus and Minus implement doOperation() to provide concrete implementations of addition in Plus class and subtraction in Minus class.

```
#include <iostream>

// Abstract class MathSymbol
class MathSymbol {
public:
    virtual double doOperation(double a, double b) const = 0;
};
```

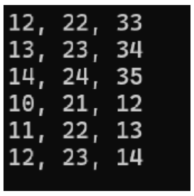
```
// Concrete class Plus implementing MathSymbol
class Plus : public MathSymbol {
public:
    double doOperation(double a, double b) const override {
        return a + b;
    }
};

// Concrete class Minus implementing MathSymbol
class Minus : public MathSymbol {
public:
    double doOperation(double a, double b) const override {
        return a - b;
    }
};

int main() {
    // Example usage
    MathSymbol* plusSymbol = new Plus();
    MathSymbol* minusSymbol = new Minus();

    double result1 = plusSymbol->doOperation(5.5, 2.3); // Should be 7.8
    double result2 = minusSymbol->doOperation(10.0, 4.5); // Should be 5.5

    std::cout << "Result of addition: " << result1 << std::endl;
    std::cout << "Result of subtraction: " << result2 << std::endl;

    delete plusSymbol;
    delete minusSymbol;
    return 0;
}
```



```
Result of addition: 7.8
Result of subtraction: 5.5
```

Q8) We can also overload a unary operator in C++ by using a friend function. Write a C++ program to overloaded ++ operator relative to the Test class using a member function.

```
#include <iostream>

class Test {
private:
    int x;
    int y;
    int z;

public:
    Test(int a, int b, int c) : x(a), y(b), z(c) {}

    void display() const {
        std::cout << x << ", " << y << ", " << z << std::endl;
    }

    void operator++() {
        ++x;
        ++y;
        ++z;
    }
};

int main() {
    Test t1(12, 22, 33);
    Test t2(10, 21, 12);

    t1.display();
    ++t1;
    t1.display();
    ++t1;
    t1.display();

    t2.display();
    ++t2;
    t2.display();
    ++t2;
    t2.display();

    return 0;
}
```



```
12, 22, 33
13, 23, 34
14, 24, 35
10, 21, 12
11, 22, 13
12, 23, 14
```

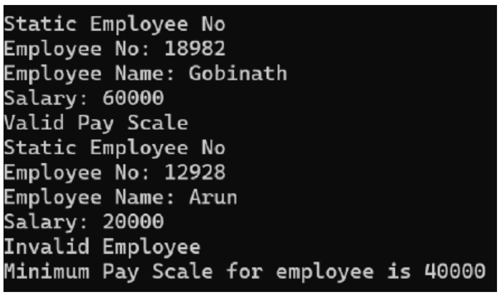
Q9) Write a C++ program showing runtime behaviour of virtual functions with following conditions

i. Virtual functions cannot be static.

ii. A virtual function can be a friend function of another class.
iii. Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
iv. The prototype of virtual functions should be the same in the base as well as derived class.
v. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
vi. A class may have virtual destructor but it cannot have a virtual constructor.

```
#include <iostream>

#include <string>


class Employee {

protected:

   int empNo;

   std::string empName;

   float salary;


public:

   Employee(int no, const std::string& name, float sal) : empNo(no), empName(name),
salary(sal) {}


   virtual void displayInfo() {

      std::cout << "Employee No: " << empNo << std::endl;

      std::cout << "Employee Name: " << empName << std::endl;

      std::cout << "Salary: " << salary << std::endl;

   }


   virtual bool isValidPayScale() {

      return salary >= 40000; // Assuming minimum pay scale is 40000

   }

};
```

```
class PermanentEmployee : public Employee {
public:
    PermanentEmployee(int no, const std::string& name, float sal) : Employee(no, name, sal) {}

    void displayInfo() override {
        std::cout << "Static Employee No" << std::endl;
        Employee::displayInfo();
    }

    bool isValidPayScale() override {
        if (salary >= 40000) {
            std::cout << "Valid Pay Scale" << std::endl;
            return true;
        } else {
            std::cout << "Invalid Employee" << std::endl;
            std::cout << "Minimum Pay Scale for employee is 40000" << std::endl;
            return false;
        }
    }
};

int main() {
    Employee* emp1 = new PermanentEmployee(18982, "Gobinath", 60000);
    Employee* emp2 = new PermanentEmployee(12928, "Arun", 20000);

    emp1->displayInfo();
    emp1->isValidPayScale();
    emp2->displayInfo();
    emp2->isValidPayScale();

    delete emp1;
    delete emp2;

    return 0;
}
```



```
Static Employee No
Employee No: 18982
Employee Name: Gobinath
Salary: 60000
Valid Pay Scale
Static Employee No
Employee No: 12928
Employee Name: Arun
Salary: 20000
Invalid Employee
Minimum Pay Scale for employee is 40000
```