National University of Computer and Emerging Sciences, Lahore Campus

Object Oriented Programming

Assignment 2

Total Marks: To be decided by TA

Section: BCS-9C **Due Date:**24th July 2023

Instructions:

Please read the following instructions carefully before submitting the assignment. 1. Follow instructions of TA for Submission
2. Submit your assignment in Google Classroom
3. It should be clear that your assignment will not get any credit if:
    - The assignment is submitted after the due date.
    - The assignment is copied (partial or full) from any source (websites, forums, students, etc.)

Task 1:

A bus is a public transport road vehicle designed to carry significantly more passengers than the average car or van. Write a class called Bus that keeps a trail of its mileage (number of miles it's driven), the number of gas left in its tank in gallons, and Bus status: broken or not. You are required to create a class Bus and give initial values to gas in the tank(decimal), and mileage(decimal):

```
Bus *m_bus = new Bus(20.0, 1200.0);
```

Bus class has a method StartAndDrive that has a parameter miles the bus should drive. The functionalities of this method are:

I. Whenever a bus starts for a ride, there is always a probability that it's going to break down. Precisely, for 0 to 10,000 miles, the probability of a failure is 0.1 (10%) and from 10,001 to 20,000 probability is 0.2 (20%) etc. If the bus goes out of order, it will stop driving.

II. On the other hand, if the bus starts successfully, it needs to cover the mileage which was given in the parameter. Mileage and the remaining amount of gas should be updated accordingly to specify the bus has driven additional miles, and has used up some gas.

A constant should be added to calculate MILES_PER_GALLON.
III. If the bus does not have enough fuel that it can cover the given distance then the bus should run out of gas. If the bus was able to cover the given distance, then the function should return true, and false otherwise (if it goes out of order or runs out of gas).

IV. Write a method called testBus that makes a new Bus and returns the mileage of that bus once it breaks down for the first time. The bus should start out with 10 gallons of gas, and 0 miles driven, and you should drive the bus in increments of 10 miles. If the bus runs out of gas, you should refill it to 10 gallons of gas. Once the bus breaks down, you should return how many miles the bus had driven.

Example 1:

• m_bus->StartAndDrive (25).

• Answer will be true and bus has 9 gallons of gas and driven miles are 1025

• MILES_PER_GALLON= 25

• Suppose, bus did not break down.

Example 2:

• m_bus->StartAndDrive (250).

• return false because of shortage of gas to cover this distance

• Bus have got 0 gallons of gas with 1,250 miles covered distance as to cover 225 miles which were left uncovered needs 9 gallons of gas.

Class methods:

bool StartAndDrive (float miles);

int getMileage();

bool statusofBus(); /* broken down or not

public void repair(); //Optional

void addFuel(double no_of_gallons);

Task 2:

Implement the polynomial class:

```
class myPolynomial

{

private:

float *coeffArray;

float *degreeArray;

int terms;

}
```

For example, a polynomial: $4x^6-2x^3+6x^2+1$ would be stored as:

coeffArray = [4 -2 6 1]

degreeArray = [6 3 2 0]

terms = 4

Make sure that the degree array is sorted in descending order so that the highest power term appears first. Also, there should be no duplicates in the degree array and it should only be allocated according to the total terms present in the polynomial.

Implement the following in MyPolynomial class

a) constructor, destructor and copy constructor

b) + operator for the addition of two polynomials

c) - operator for subtracting one polynomial from another

d) * operator for multiplying two polynomials

e) = assignment operator

f) == comparison operator

g) Print the polynomials
h) Input the polynomial from file

Example usage

Following code should work if called from a caller function: •

myPolynomial a,b,c; //constructors called, everything initialized

- input a,
- input b;
- c=a+b;
- a=c-b;
- b=a*b;
- a.print();
- b.print();
- if (a==b) cout << “same polynomials”;

Menu

Test the polynomial class via a menu based system.

Examples of menu options are:

0. Exit

1. Input polynomial a from file (for this option enter polynomial a from user) 2. Input polynomial b from file (for this option enter polynomial b from user) 3. Add a and b to give c

4. Multiply a and b to give c

5. Subtract a from b to give c

6. Print a

7. Print b
8. Print c

9. Check a == b

...

Note

The arrays should be allocated according to the total terms of the polynomial. There should be no memory leaks. You have to find out how addition, multiplication and subtraction of polynomials take place. Also, you have to make sure that the resulting polynomial's degrees are sorted in descending order without any duplicates. You can implement any required helper private function

Task 3:

Create two classes **DM** and **DB** that store the value of distances. DM stores distance

in *Meters* and *centimeters* and DB in *feet* and *inches*.

1 foot= 12 inches

1 inch=2.54 cm

For example

Distance1= 4 m 15 cm

Distance2= 5 ft 10 inch

Result_cm= (4*100+15)+ ((5*12)+10)*2.54

M= Result_cm /100

Cm= Result_cm %100

Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a *friend* function to carry out the addition operation.

The object that stores the results would be a DM object

- Create parameterized constructors of DM and DB such that DM(int m, int cm) and DB(int ft, int inches)
- Create a friend function *friend DB add(DB, DM)* which returns an object of DB after performing the addition on DB and DM objects.

Task 4:

At the start of this task, your class should have the following member functions:

```
class MyString
{
  private:
```

```
        char *str; // Pointer to the char array that holds the string
        int length; // Variable to store the length of the string (excluding NULL)
public:
// Default constructor to initialize the string to empty string
        MyString();

        // Overloaded constructor
        MyString(const char *);

        // Copy constructor
        MyString(const MyString&);

        // Returns the length (# of characters excluding the null terminator) of the string int
        getLength (); // Destructor
        ~MyString();

};
```

Now, complete the following tasks in the given order:

Task 4.1

Overload the **pre-increment operator ++** to convert the **MyString** to uppercase i.e. all lowercase letters in the given MyString will be converted into uppercase.

Task 4.2

Overload the **stream insertion operator <<** to display a **MyString** on screen.

Task 4.3

Overload the **stream extraction operator >>** to take input of a **MyString** from the user.

Task 4.4

Overload the **assignment operator(=)** to copy the values of a **MyString** object to another existing MyString object**.**

Task 4.5

Overload the **subscript operator []** to access or modify the character stored at a particular location in the given **MyString**. Also perform bound checking i.e. if the specified index is invalid, display an error message and terminate the program.

Task 4.6

Overload the **operator ==** that returns true if the two **MyStrin**g objects are equal else return false.

Good Luck ☺