Programming Practice Questions

Exercise 1: Write a program that takes two matrices as input and returns the product of the two matrices. Use dynamic memory allocation to create the matrices and pointers to access the matrix elements

Exercise 2: Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected.

Model this tollbooth with a class called **tollBooth**.

1. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected.
2. A constructor initializes both of these to 0. A member function called payingCar() increments the car total and adds 0.50 to the cash total.
3. Another function, called nopayCar(), increments the car total but adds nothing to the cash total.
4. Finally, a member function called display() displays the two totals. Make appropriate member functions const.

Exercise 3: Create a class called **time** that has separate int member data for hours, minutes, and seconds.

1. One constructor should initialize this data to 0, and another should initialize it to fixed values.
2. Another member function should display it, in 11:59:59 format.
3. The final member function should add two objects of type time passed as arguments.

A main() program should create two initialized time objects (should they be const?) and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable.
Make appropriate member functions const.

Exercise 4: In ocean navigation, locations are measured in degrees and minutes of latitude and longitude. Thus if you're lying off the mouth of Papeete Harbor in Tahiti, your location is 149 degrees 34.8 minutes west longitude, and 17 degrees 31.5 minutes south latitude. This is written as 149°34.8' W, 17°31.5' S. There are 60 minutes in a degree. Longitude is measured from 0 to 180 degrees, east or west from Greenwich, England, to the international dateline in the Pacific. Latitude is measured from 0 to 90 degrees, north or south from the equator to the poles.

Create a class **angle** that includes three member variables: an int for degrees, a float for minutes, and a char for the direction letter (N, S, E, or W).
This class can hold either a latitude variable or a longitude variable.

Write one member function to obtain an angle value (in degrees and minutes) and a direction from the user, and a second to display the angle value in 179°59.9' E format.
Also write a three-argument constructor.
Write a main() program that displays an angle initialized with the constructor, and then, within a loop, allows the user to input any angle value, and then displays the value. You can use the hex character constant '\xF8', which usually prints a degree (°) symbol.

Exercise 5: Design a dateType class as follows:
private member variables:
dMonth: int
dDay: int
dYear: int

private Member Function:
setDate(int, int, int): void
getDay() const: int
getMonth() const: int
getYear() const: int
printDate() const: void
dateType(int = 1, int = 1, int = 1900)

Exercise 6:
The above dateType class in Exercise 5 was designed to implement the date in a program, but the member function setDate and the constructor do not check whether the date is valid before storing the date in the data members. Rewrite the definitions of the function setDate and the constructor so that the values for the month, day, and year are checked before storing the date into the data members. Add a function member, isLeapYear, to check whether a year is a leap year. Moreover, write a test program to test your class.

Exercise 7:
In Programming Exercise 5, the class dateType was designed and implemented to keep track of a date, but it has very limited operations. Redefine the class dateType so that it can perform the following operations on a date in addition to the operations already defined:

1. Set the month.
2. Set the day.
3. Set the year.
4. Return the month.
5. Return the day.
6. Return the year.
7. Test whether the year is a leap year.
8. Return the number of days in the month. For example, if the date is 3-12-2011, the number of days to be returned is 31 because there are 31 days in March.

9. Return the number of days passed in the year. For example, if the date is 3-18-2011, the number of days passed in the year is 77. Note that the number of days returned also includes the current day
10. Return the number of days remaining in the year. For example, if the date is 3-18-2011, the number of days remaining in the year is 288.
11. Calculate the new date by adding a fixed number of days to the date. For example, if the date is 3-18-2011 and the days to be added are 25, the new date is 4-12-2011

Exercise 8:
Write a class called Line that represents a line segment between two Points. Your Line objects should have the following methods:

- Line(Point &p1, Point &p2)

Constructs a new Line that contains the given two Points.

- Line(int x1, int y1, int x2, int y2)

Constructs a new Line that contains the given two Points.

- Line(Line &copy) // it's a copy constructor

Constructs a new Line form given line.

- Point getP1()

Returns this Line's first endpoint.

- Point getP2()

Returns this Line's second endpoint.

- double getSlope()

Returns the slope of this Line. Remember The slope of a line between points (x1, y1) and (x2, y2) is equal to (y2-y1)/(x2-x1).

- String toString()

Returns a String representation of this Line, such as "[(22, 3), (4,7)]"

Exercise 9:
Design a class String having a data member as char *s, int size, and define the appropriate function members, and your task is to overload binary operator '+' to concatenate two strings. For example:
If you pass the string "hello" and "world" for s1 and s2 respectively, it will concatenate both into s3, and display the output as "helloworld".

Exercise 10:
The C++ string class (in header <string>) overloads these operators to work on string objects:

- **String comparison** (==, !=, >, <, >=, <=): For example, you can use str1 == str2 to compare the contents of two string objects.
- **Stream insertion and extraction** (<<, >>): For example, you can use cout << str1 and cin >> str2 to output/input string objects.
- **Strings concatenation** (+, +=): For example, str1 + str2 concatenates two string objects to produce a new string object; str1 += str2 appends str2 into str1.
- **Character indexing or subscripting []**: For example, you can use str[n] to get the char at index n; or str[n] = c to modify the char at index n. Take note that [] operator does not perform index- bound check, i.e., you have to ensure that the index is within the bounds.
- **Assignment** (=): For example, str1 = str2 assigns str2 into str1.

You are required to implement all mentioned functionalities using operator overloading in a header file and finally include that header file in your implementation/cpp file and use it.

Exercise 11:
In this exercise, we are going to create a small scale Event Management System.

1. Create a class Event with following variables: char* event_name, char* event_venue, char event_date[11] and char event_time[9].
    - Input format for event_date: dd-mm-yyyy
    - Input format for event_time: hh:mm am/pm
    - Event_name and even_venue will be dynamic arrays, size should not be more than number of characters +1 for null character.

2. Implement default constructor and overloaded constructor. Print “Default Constructor Called” and “Overloaded Constructor Called” in the respective constructors. The declaration for overloaded constructor is as follows:
    - Event(char event_name[20], char event_venue[50], char event_date[11],char event_time[9]);
    - This constructor will deep copy the input cstrings to member variables.

3. Implement all setters and getters for class Event. You can create a helper function userInput() to input event details.
    - All setter functions should take input from user to change the value.
    - When every new name is set for dynamic cstring member variables, old space should be deleted and new space should be allocated.

4. Implement the destructor ~Event() for class Event. Print “Destructor Called” in the destructor. Deallocate all the dynamically allocated memory of class data members.

In main, create event with following details.
event name: MID EXAM

event venue: LAB 01
event date: 28-03-2023
event time: 02:20 PM
Call setEventName function to change the name to PRACTICE LAB

Note:

- Deallocate all dynamically allocated memory.
- Do not use strcpy() function. Copy the character array manually where needed based on ending '\0' character.
- Follow all the code indentation, naming conventions and code commenting guidelines.

Exercise 12:
A point in the x-y plane is represented by its x-coordinate and y-coordinate. Design a class, **pointType**, that can store and process a point in the x-y plane. You should then perform operations on the point, such as showing the point, setting the coordinates of the point, printing the coordinates of the point, returning the x-coordinate, and returning the y-coordinate. Also, write a test program to test the various operations on the point