NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES
ISLAMABAD CAMPUS
COMPUTER PROGRAMMING (CS103) - FALL 2018
ASSIGNMENT-4

Due Date: October 29, 2018 (05:00 pm)
Instructions:

1. *Make sure that you read and understand each and every instruction.*
2. *Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded straight zero in this assignment or all assignments.*
3. *Submit a single '.zip' file for your assignment and each problem solution must be provided in a folder and three separate CPP file. For instance, you must name the folder containing solution of first problem as 'q1' and second as 'q2' and so on.*
4. *You have to write each class in header and .cpp files Name the files as per your question. For instance, folder named as q1 will have three files. q1.h, q1.cpp and q1main.cpp.No code will be accepted without these files.*
5. *Note: You have to follow the submission instructions to the letter. Failing to do so can get a zero in assignment.*

Q1) Number Array Class: Design a class that has an array of floating-point numbers. The constructor should accept an integer argument and dynamically allocate the array to hold that many numbers. The destructor should free the memory held by the array. In addition, overload the following operators to extend the functionality of the class:

- Array(); // a default constructor
- Array(**int** size); // a parametrized constructor initializing an Array of predefined size
- Array(**float** *arr, **int** size); // initializes the Array with an existing Array
- Array(**const** Array &); // copy constructor
- **float**& **operator**[](**int** i); // returns the integer at index [i] after checking the out of range error
- **const** Array & **operator**=(**const** Array&); //copy the array
- Array **operator**+(**const** Array&); //adds two Array
- Array **operator**-(**const** Array&); //subtracts two Array
- Array **operator**++(); //adds one to each element of Array
- Array **operator**++(**int**); //adds one to each element of Array
- Array& **operator**--(**int**); //subtracts one from each element of array
- **bool operator**==(**const** Array&)**const**; //returns true if two arrays are same
- **bool operator**!(); // returns true if the Array is empty
- **void operator**+=(**const** Array&); //adds two Array
- **void operator**-=(**const** Array&); //subtracts two Array
- **int operator** ()(**int** idx, **int** val); // erases the value val at idx. Returns 1 for a successful deletion and -1 if idx does not exists or is invalid. Shift the elements after idx to the left.
- ~Array(); // destructor...
- ostream& **operator**<<(ostream& input, **const** Array&); //Inputs the Array
- istream& **operator**>>(istream& ouput, Array&); //Outputs the Array

Q2) Rational Class: Create a class called Rational for performing arithmetic with fractions. Write a program to test your class.
Use integer variables to represent the private data of the class

- Numerator
- Denominator.

Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should contain default values in case no initializers are provided and should store the fraction in reduced form. For example, the fraction 2/4 would be stored in the object as 2 in the numerator and 4 in the denominator. Overload the following operators:

- +, - , /, * that adds, subtract, multiply and divide two rational numbers
- = to copy a rational number into another number.
- <, >, <=, >=,! =, = = that compares two rational numbers.
- ++, -- (pre/post increment and decrement) to add a constant value in a rational number.
- <<, >> to input and output a rational number

Q3) Day of the Year: Assuming that a year has 365 days, write a class named **DayOfYear** that takes an integer representing a day of the year and translates it to a string consisting of the month followed by day of the month. For example,

Day 2 would be *January 2*.
Day 32 would be *February 1*.
Day 365 would be *December 31*.

The constructor for the class should take as parameter an integer representing the day of the year, and the class should have a member function print() that prints the day in the month day format. The class should have an integer member variable to represent the day. You are going to need global array of string objects that can be used to assist in the translation from the integer format to the month-day format. Overload the following operators for **DayOfYear** class:

- ***operator+ (int val)*** // Adds val into the days and convert into appropriate day of the year
- ***operator+ (DayOfYear &right)*** // Adds right into the current object and return result.
- ***operator- (DayOfYear &right)*** // Subtract right into the current object and return result.
- ***operator=*** // Copy one object to the other
- ***operator***== //Checks either the two objects are equal or not
- ***operator++*** //post and pre increment operator. These operators should modify the DayOfYear object so that it represents the next day. If the day is already the end of the year, the new value of the object will represent the first day of the year
- ***operator- -*** //post and pre decrement operator These operators should modify the DayOfYear object so that it represents the previous day. If the day is already the first day of the year, the new value of the object will represent the last day of the year.
- ***operator<<*** prints the day in the month day format.
- ***operator>>*** inputs the member function of **DayOfYear** class.

Q4) Implementation of Bouquet of Flowers: Your goal here is to write classes for creating a bouquet of flowers.
To create the bouquet of flower your will need to write following two classes.
Design a class Flower. A flower is characterized by following attributes:

- a name
- a color
- a basic price per unit
- an indication whether the flower is perfumed or not
- and an indication to know whether the flower is on sale.

and with following behavior:

- a constructor initializing the attributes using parameters given in the order shown by the provided main(); a default constructor will not be necessary but the last two parameters will have false as default value;
- a price method returning the flower's price : the price will be the base price if the flower is not on sale; otherwise, the price will be half the base price;
- a bool perfume() method indicating whether the flower is perfumed or not;
- Overloaded stream insertion operator. The characteristics have to be displayed in strict accordance with the following format :

  <Name> <Color> <earfumed>, Price: <Price> Rs.
- an overloading of the == operator returning true if two flowers are identical, false otherwise. Two flowers are considered identical if they have the same name, the same color, and the two flowers are both either perfumed or not (neither the price nor the fact that the flower is on sale or not is involved in the comparison).

Next write a "Bouquet" class which will be modeled using a dynamic array of Flowers. The Bouquet class offers the following methods:

- a method bool perfume() returning true if the bouquet is perfumed and false otherwise; a bouquet is perfumed if at least one of its flowers is perfumed;
- a method price without parameters returning the price of the bouquet of flowers; This is the sum of the prices of all its flowers; this sum is multiplied by two if the bouquet is perfumed;
- a stream insertion method, should display all information of bouquet with the total price. This method will display the characteristics of the bouquet of flowers respecting rigorously the following format :

If the bouquet does not contain any flower,
Still no flower in the bouquet
or :
Perfumed Bouquet composed of:
<Flower1>
..
<FlowerN>
Total Price: <Price_of_bouquet> Rs.

Here < FlowerX > means display of the Xth ower of the bouquet in the format specified by the overload of the << operator. There is a newline after displaying each flower and after displaying the price of the bouquet.

- an overload of the += operator which allows adding a flower to the bouquet, the flower will always be added at the end.
- an overload of the - = operator taking as a parameter a flower and removing from the bouquet all the flowers identical to the latter (according to the definition of the == operator);
- an overloaded + operator according its usage in the provided main
- an overloaded - operator according to its usage in the provided main

```
int main() {
    // example of Yellow oderless rose.
    Flower r1("Rose", "Yellow", 1.5);
    cout << r1 << endl;
    // example of Yellow parfumed rose
    Flower r2("Rose", "Yellow", 3.0, true);
    // example of parfumed Red rose on sale
    Flower r3("Rose", "Red", 2.0, true, true);
    Bouquet b1;
    b1 += r1; // add one Flower of r1 type
    b1 += r1; // add another Flower of r1
    b1 += r2;
    b1 += r3;
    cout << b1 << endl;

   b1 = b1 - r1; // Delete all the Flowers of type r1
   cout << b1 << endl;
   Bouquet b2;
   b2 = b1 + r1; // Add one Flower of type r1
   cout << b2 << endl;

   // Delete all the parfumed flowers from the bouquet.
   b2 -= r2;
   b2 -= r3;
   cout << b2;
   return 0;
}
```