CS-1004: Object-Oriented Programming

Assignment 3

(Deadline: 25th July, Summer 2023 11:59 PM)

Submission: Header (.h) files are provided where required. Create a .cpp file for each question, name the .cpp file the same as the header. E.g Car.h and Car.cpp. Also write your main class in order to execute your required functions.

You need to submit both header file and cpp file for each question.

All cpp files must contain your name, student-id, and assignment # on the top of the file in the comments. Place all your .cpp files (only) in a folder named your ROLLNUM_SECTION (e.g. 21i-0001_A). Compress the folder as a zip file and upload on Google classroom.

Deadline: The deadline to submit the assignment is `25th July, 2023` at 11:59 PM. Correct and timely submission of the assignment is the responsibility of every student.

Plagiarism: This is an individual assignment; any kind of plagiarism will result in a zero.

Restrictions: All functions will be out-of-line.

Evaluation: The assignment is of 160 marks. All submissions will be evaluated on the basis of objects created in your main class.

Question1 :-(36 Marks)

Dollar

The US dollar is composed of many different coins, which include nickels, cents, and quarters, which can be combined to make up any amount of money. In this question, you are required to create a Money Class in which a specific amount of money will be represented in terms of dollars, quarters, nickels, and cents in that precedence. This money class will consist further of the dollar, quarter, nickel, and cent classes.
For example, if your amount of money is 4.56, your class wills first store the maximum possible amount in dollars, then it will move on to quarters, nickels, and cents. You are required to implement the following functions:

```
class Money {
// think about the private data members
public:
Money();// default constructor
Money(double);// parameterized constructor
//Implement getters and setter functions
Moneyoperator+(Money m); //
Moneyoperator-(Money m); //
Moneyoperator+(Dollar d): //
Moneyoperator-(Dollar d); //
Moneyoperator+(Nickel d): //
Moneyoperator-(Nickel d); //
Moneyoperator+(Quarter d): //
Moneyoperator-(Quarter d); //
Moneyoperator+(Cent d): //
Moneyoperator-(Cent d); //

//These are all addition and subtraction on overloads to add specific coins to the total amount of money
Moneyoperator ++(int) //Round up the current amount of money to the nearest quarter
Moneyoperator --(int) //Round down the current amount of money to the nearest quarter

Booloperator>=()
Booloperator<=()
//Comparison operators
Quarters operator!()
//Returns the maximum number of quarters that can be obtained from Money
Nickelsoperator~()
//Returns the maximum number of nickels that can be obtained from Money
Moneyoperator/(int n)
//Returns the money object created if the current money amount were divided into n
parts
Moneyoperator*(int n)
//Returns the money object created by multiplying the current total amount by n

};
```

Question2 :-(34 Marks)

Date

Write a class called Date that represents a date consisting of a year, month, and day(you can think about private data members of the class). A Date object should have the methods and operators which are shown in Table 1:

Table 1

Date(int year. int month, int day)	Constructs a new Date object to represent the given date.
operator =	Overload = operator to assign values
d2=d1+1	Overload + operator which takes integer as argument It moves this Date object forward in time by the given number of days.
d2=di-4	Overload - operator which takes integer as argument It moves this Date object backward in time by the given number of days.
d3=d1+d2	Overload + operator which takes Date object as argument.
d3=d1-d2	Overload - operator which takes Date object as argument.
bool a=d1>d2:	Overload > operator which returns true or false.
bool a=d1>=d2:	Overload >= operator which returns true or false.
bool a=d1	Overload < operator which returns true or false.
bool a=dl<=d2:	Overload <= operator which returns true or false.
bool d1!=d2:	Overload != operator which returns true or false.
bool d1==d2:	Overload == operator which returns true or false.
int getDay()	Returns the day value of this date: for example, for the date 2006/07/22 returns 22.
int getMonth()	Returns the month value of this date; for example, for the date 2006/07/22 returns 7.
int getYear()	Returns the year value of this date: for example, for the date 2006/07/22 returns 2006.
bool isLeapYear()	Returns true if the year of this date is a leap year. A leap year occurs every four years, except for multiples of 100 that are not multiples of 400. For example, 1956. 1844, 1600, and 2000 are leap years, but 1983. 2002. 1700. and 1900 are not.
String toString()	Returns a String representation of this date in year/month/day order. such as "2006/07/22"

Question3 :-(48 Marks)

Big Integer

BigInt class is used for the mathematical operations that involve very big integer calculations that are outside the limit of all available primitive data types. For example, factorial of 100 contains 158 digits in it so we can't store it in any primitive data type available. We can store as large Integer as we want in it.
Your goal is to overload the operators for a generic "BigInt" class. You will need to write two files (BigInt.h and BigInt.cpp). Your implemented class must fully provide the definitions of following class (interface) functions .

```
classBigInt
{
//think about the private data members
public:
BigInt(int val = 0);
BigInt(conststring& text);
BigInt(constBigInt& copy); // copy constructor
// Binary Operators and Arithmetic Operators
BigIntoperator+(constBigInt& val) const;
BigIntoperator+(int val) const;
BigIntoperator-(constBigInt& val) const;
BigIntoperator-(int val) const;
BigIntoperator*(constBigInt& val) const;
// Compound Assignment Operators
BigIntoperator+=(constBigInt& rhs);
BigIntoperator-=(constBigInt& rhs);
BigIntoperator*=(constBigInt& rhs);
// Logical Operators
booloperator==(constBigInt& val) const;
booloperator!=(constBigInt& val) const;
booloperator<(constBigInt& val) const;
booloperator<=(constBigInt& val) const;
booloperator>(constBigInt& val) const;
booloperator>=(constBigInt& val) const;
// Unary Operators
BigInt&operator++();  // Pre-increment Operator
BigIntoperator++(int);  // Post-increment Operator
BigInt&operator--();  // Pre-decrement Operator
BigIntoperator--( int ); // Post-decrement Operator
//Conversion Operator
operator string();    // return value of the BigInt as string
~BigInt(); // destructor
};
ostream&operator<<(ostream& output, constBigInt& val); // outputs the BigInt
istream&operator>>(istream& input, BigInt& val); // inputs the BigInt
```

Question4 :-(18 Marks)

Binary Store Calculator

A BinaryStore calculator will store bytes "stored in strings" with their addresses, i.e., at each address in the BinaryStore there is a stored Byte. Each address will be 4 characters string and each byte will be 8 characters strings.

Create a class **BinaryStore** and class **Byte** that offer following overloaded operators.

class offers the following methods:

- an overloaded +=operator that will add the address in the list of BinaryStore
- an overloaded + that will add two string bytes for Byte Class
- an overloaded - that will subtract two string bytesByte Class
- an overloaded || operator that will give the string which is bit by bit logical or of act two string bytes for Byte Class
- an overloaded && operator that will give the string which is bit by bit logical and of act two string bytesfor Byte Class
- an overloaded == operator that will give bool value if they are equal or notfor Byte Class

furthermore, implement all the constructors and member functions required against following sample code.

```
int main()
{
// address will be only 4 bit long for this problem. And bytes will be only 8 bit long
// create a 10 locations binary store for storing addresses and values
        BinaryStore b1(10);
        b1+="0011"; // add a new address to the store
        b1.Add("0011",Byte("00000010")); // add the byte at newly added address
        cout<<b1;
        b1+="0110";
        b1.Add("0110",Byte("00000110"));
        b1+="1010";
        Byte nb=b1.Get("0011")+b1.Get("0110"); // add two bytes, bit by bit
        b1.Add("1010",nb);
        Byte nb2=b1.Get("1010")+b1.Get("0110");
        bool r=b1.Get("0011")==b1.Get("0110");
        cout<<endl;
        cout<< "Equal = " << r << endl;
        cout<<b1;
        Byte nb3=b1.Get("1010");
        bool r1=nb3==b1.Get("1010");
        cout<<endl;
        cout<< "Equal = " << r1 << endl;
```

```
        Byte nb4=b1.Get("0011") || b1.Get("1010"); // byte1 OR byte2 , bit by bit
        Byte nb5("00001100");
        Byte  nb6= nb4 && nb5;// byte1 AND byte2 , bit by bit
        b1+="1011";
        b1.Add("1011",nb6);
        cout<<b1;

}
```

Question # 5

Question5 :-(24 Marks)

Bouquet of **Flowers**

Your goal here is to write classes for creating a bouquet of flowers. To create the bouquet of flower you will need to write following two classes.

Design a class Flower. A "Flower" is characterized by the 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.

The class has the following behaviors:

- 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
- **operator string() const**to return value of the Flower as a string as:
  <Name><Color><Perfumed>, Price: <Price> Rs.
- Overloaded stream insertion operator. The characteristics have to be displayed in strict accordance with the following format:
  <Name><Color><Perfumed>, 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, 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;
- **operator string() const**to return value of the Bouquet as a string as:
  If the bouquet does not contain any flower,

  Still no flower in the bouquet

Else:
<Flower1>
..
<FlowerN>
Total Price: <Price_of_bouquet> Rs.

- 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
  Else:
  Perfumed Bouquet composed of:

  <Flower1>
  ..
  <FlowerN>
  Total Price: <Price_of_bouquet> Rs.

Here < FlowerX > means display of the X^{th}flower 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 perfumed rose
        Flower r2("Rose", "Yellow", 3.0, true);
        // example of perfumed 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 perfumed flowers from the bouquet.
        b2 -= r2;
        b2 -= r3;
        cout << b2;
        return 0;
}
```