National University of Computer and Emerging Sciences, Lahore Campus

Course:	Object Oriented Programming Lab	Course Code:	CL-217
Program:	BS (Computer Science)	Semester:	Spring 2020
Duration:	220 Minutes + 20 mins for submission	Total Marks:	(40+60)
Paper Date:	16 July 2020	Weight	50%
Section:	ALL	Page(s):	4
Exam:	Final Term	Reg. No	

Instruction/Notes:

1. Understanding the question paper is also part of the exam, so do not ask any clarification.

Question # 1:

Question: **What is Sparse Matrix?**

In computer programming, a matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents a **n X m** matrix. There may be a situation in which a matrix contains a greater number of ZERO values than NON-ZERO values. Such matrix is known as **sparse matrix.**

We can store these sparse matrices using less memory, by storing only non-zero entries rather than storing all entries.

Example: n x m = 7 x 9

3	0	0	1	0	0	0	0	2
1	0	0	0	0	4	0	8	1
0	0	2	0	0	0	0	5	0
0	6	0	0	0	0	0	0	0
0	0	0	0	0	0	8	0	0
2	2	0	0	0	0	0	0	0
0	0	0	4	8	0	0	0	0

Consider two sparse matrices of Size:3x7.

Sparse Matrix # 1

3	0	0	1	0	0	0
0	0	9	0	0	0	0
7	12	9	0	8	2	8

Sparse Matrix # 2

0	0	0	1	2	0	8
0	0	0	0	0	0	0
7	3	0	0	2	0	0

The above sparse matrices will be stored in compact matrices, shown as below.

Compact Matrix # 1

Row	No. of elements	Column index	Value	Column index	Value
0	2	0	3	3	1
1	1	2	9		
2	6	0	7	1	12

2	9	4	8	5	2	6	8

Compact Matrix # 2

Row	No. of elements	Column index	Value	Column index	Value	Column index	value
0	3	3	1	4	2	6	8
1	0						
2	3	0	7	1	3	4	2

Note: You are required to only allocate exact size of memory.

To implement the idea of storing the sparse matrix into a compact matrix, you need to implement the class of **CompactMatrix. CompactMatrix** will have two data members, **rows** (int), **mat** (int **). You need to **implement** the required **functionalities:**

1. **Constructors:** Which receives a sparse matrix and stores in mat (int **) in compressed/compact form. **(20)**
2. **<< operator (Display):** Which displays the matrix in compact form. **(10)**
3. **Destructor.** **(10)**

A **sample main program** is given to you.
NOTE: While evaluation, the values of these matrices and rows/cols can be changed.

```
void main() {
        const int rows = 3, cols = 7;
        int m1[rows][cols] = { {3,0,0,1,0,0,0},{0,0,9,0,0,0,0},{7,12,9,0,8,2,8} };
        int m2[rows][cols] = { {0,0,0,1,2,0,8},{ 0,0,0,0,0,0,0},{ 7,0,3,0,2,0,0} };
        int *matrix1[rows]; int *matrix2[rows];  //array of pointers same as int**
        for (int i = 0; i < rows; i++) {
                matrix1[i] = m1[i];
                matrix2[i] = m2[i];
        }
        CompactMatrix compactMatrix1(matrix1, rows, cols);
        CompactMatrix compactMatrix2(matrix2, rows, cols);
        cout<< compactMatrix1<<endl;
        cout << "\n\n + \n\n";
        cout<< compactMatrix2<<endl;
        cout << "\n\n = \n\n";
}
```

Question # 2:

There is a Dessert Shop which takes orders and sells cookies by the dozen, ice cream, and sundaes (ice cream with a topping). Your code will be used for the taking the orders only. To do this, you will implement an inheritance hierarchy of classes derived from a **DessertItem** superclass.

The DessertItem Class

The **DessertItem** class is a *superclass* from which specific types of **DessertItems** can be derived. It contains only one data member, a name (char *). It also defines a number of methods. All of the **DessertItem** class methods except the **getCost()** method are defined in a generic way.

The **getCost()** method in **DessertItem** class should only return 0 because the method of determining the costs varies based on the type of item.

Tax amounts should be rounded to the nearest Rupees. For example, calculating the tax on a food item with a cost of 199 rupees with a tax rate of 2.0% should be 4 rupees.

The Derived Classes

All of the classes which are derived from the **DessertItem** class must define a constructor. The **Cookie** class should be derived from the **DessertItem** class. A **Cookie** item has a *number* and a *price per dozen* which are used to determine its *cost*. For example, 4 cookies @ 399 rupees /dz. = 133 rupees. The cost should be rounded to the nearest rupees.

The **IceCream** class should be derived from the **DessertItem** class. An **IceCream** item simply has a *cost and flavor name*.
The **Sundae** class should be derived from the **IceCream** class. The *cost* of a Sundae is the *cost of the IceCream* plus the *cost of the topping*.
Make the two functions getCost() and getTax() virtual in base class.
Add attributes/functions where needed to implement this system.

Order class
This class should have list of dessert Items in polymorphic order and the count of items.
E.g DessertItems** itemsList; int noOfItems.
Write appropriate constructors, destructor and printOrder function.
In the constructor, of order class only count of items will be passed. Now, you have to implement a function placeOrder() as a member function of order class. Its partial implementation is given below:

```
void placeOrder()
{
        int userChoice;
        int i=0, itemCount=0;
        while(i<this->noOfItems)
        {
                cout<<"Choose Item you want to add\n";
                cin>>userChoice;
                if(userChoice==1)
                {
                        //take parameters from user needed for a Cookie
                                itemList[itemCount++]= new Cookie(...)
                        //making base class pointer //point cookie object
                }
                else if(userChoice==2)
                {
                        //take parameters from user needed for an IceCream
                        itemList[itemCount++]= new IceCream(...)
                        //making base class pointer //point IceCream object
                }
                else if(userChoice==3)
                {
                        //take parameters from user needed for a Sundae
                        itemList[itemCount++]= new Sundae(...)
                        //making base class pointer //point Sundae object
                }
                i++;
}
```

As one person can order the system for multiple times, so in the main function you need to ask the user, how many orders you want to place. Let's say, user wants to place two orders, two instances of order class will be created dynamically and the placeorder function will be called.
Finally, all of the orders will be displayed on the system screen.

```
void main(){
	int totalOrders;
	cout<<"How many orders you want to place: ?";
	cin>>totalOrders;
	Order ** ordersList = new Order[totalOrders];
	for(int i=0; i<totalOrders; i++)
	{
		int itemsCount;
		cout<<"How many items you want: ?";
		cin>>itemsCount;
		ordersList[i] = new Order(itemsCount);
		ordersList[i]->placeOrder();
	}
	//diplay all the orders here
	int TotalCost;
	//Find totalCost (sum of costs of all items in list without tax) and print it.
	int TotalTax;
	//Find totalCost (sum of taxes of all items in list) and print it.
	//delete the dynamially allocated memory.
}
```

----------GOOD 😊 LUCK----------