#3-3, Object Oriented Programming Lab. 2023 First Semester

Due date: 2023/05/26, 23:59:59

1. (Class Inheritance with virtual) Suppose you are designing a price compare program. The program compares the discount price with the operator '<' in Sale class. Create class called Sale that has price (type double, private) and public functions are shown below. At first, the input data are price of the item. Function Bill is declared to be a virtual function. If a function is virtual, a new definition of the function is given in a derived class, then for any object of the derived class, that object will always use the definition of the virtual function that was given in the derived class. You should create the DiscountSale class which inherits from Sale class. It has a private value named DiscountPercent (type double) and public funcitons. The class also has a Bill function which calculates the discount price using the GetPrice function in Sale class. DiscountPercent is member variable in DiscountSale class. Do not add any variable or function in Sale. Prototype of Sale class and result as follows:

```
class Sale
{
   private:
      double Price;

   public:
      Sale();
      Sale(double ThePrice);
      ~Sale();

      double GetPrice();
      virtual double Bill();
      double Savings(Sale& Other);
      bool operator < (Sale& Other);
};
```

```
< Example >

====================================
Price Compare Program
====================================
Insert item1 price: $16
Insert item2 price: $12

------------------------------------------------------------
Insert discount percent: 25%
------------------------------------------------------------
Result:
Discount price of item2 is cheaper.
Saving discount price is $3.0
```

2. (Class inheritance & Function overriding) Implement two classes, Professor and Student inheriting Person class. Person class is inherited by Professor and Student class. Professor class has extra member variables of a professor number and a major, and Student class has additional member variables of a student number, a major and a school year. Override Say function for inherited classes to display all member variables in the class object. Write a program that creates Professor and Student class objects and test your code properly. (Do not change the prototype of Say function)

```
Class Person
{
      protected:
            int age;
            char name[32];

      public:
            Person();
            ~Person();
            virtual void Say()=0;
};
```

3. (Operator overloading) Create a class called Matrix for performing arithmetic with 3x3 matrices. Write a program to test your class. Each element in the matrix should be represented using double precision. Provide a constructor that enables an object to be initialized when it is declared. The constructor should contain default values in case that no initializers are provided. Provide public member functions for each of the following:
a) Printing a matrix in the form of a 3x3 array, where each element is separated by a space and each row is separated by a newline.
b) Overloading the operators +=, -=, *, to simulate the corresponding matrix operations.

4. (Doubly Linked-list) Implement a student score management system with a doubly linked list. The system stores and manages student's average score of Math, English and Science. Each score of course are read from a user and calculates the average score in a main function. All the average scores are stored in each Score class and handled by a StudentScore-List class. Score class contains one double type variable for storing the average score and two pointers indicating next and previous Score node, respectively. Note that when implementing the "Insert()" function in a StudentScoreList class, the Score nodes should be linked into ascending order by average score. And the "PrintList()" function prints all the average scores in "Score" nodes as ascending or descending order according to the parameters, "IsAscending". (Ascending order if "is Ascending" is true, otherwise, descending order is used.)

```
class StudentScoreList
{
   private:
      Node* m_pHead;
      Node* m_pTail;

   public:
      StudnetScoreList()
      ~StudentScoreList();

      void Insert(Score* pScore);
      void PrintList(bool isAscending);
};
```

```
class Node
{
   private:
      Node* m_pNext;
      Node* m_pPrev;
      double m_Avg;

   public:
      Score();
      ~Score();

      void SetAvg(double avg);
      void SetNext(Score* pNext);
      void SetPrev(Score* pPrev);
      double GetAvr();
      Score* GetNext();
      Score* GetPrev();
};
```