ASSIGNMENT 1
SECJ/ SCSJ1023 – Programming Technique II

- This assignment can be done individually or in pairs.
- Your programs must follow the input and output as required in the text and shown in the examples. You must test the programs with (but not limited to) all the input given in the examples.
- Any form of plagiarisms is **NOT ALLOWED**. Students who copied other student's program/ assignment will get **ZERO** mark (both parties, student who copied and student that share their work).
- Please insert your name, matrices number, section of your class and date as a comment in your program.
- Only one submission per pairs (partners) is required for the submission which is the source code (the file with the extension .cpp). Submit the assignment via the UTM's e-learning system.

Question 1

Consider the class diagram in **Figure 1** which shows the data model for a car rental company. Note that the company has set the rule that each customer can only rent one car at a time.

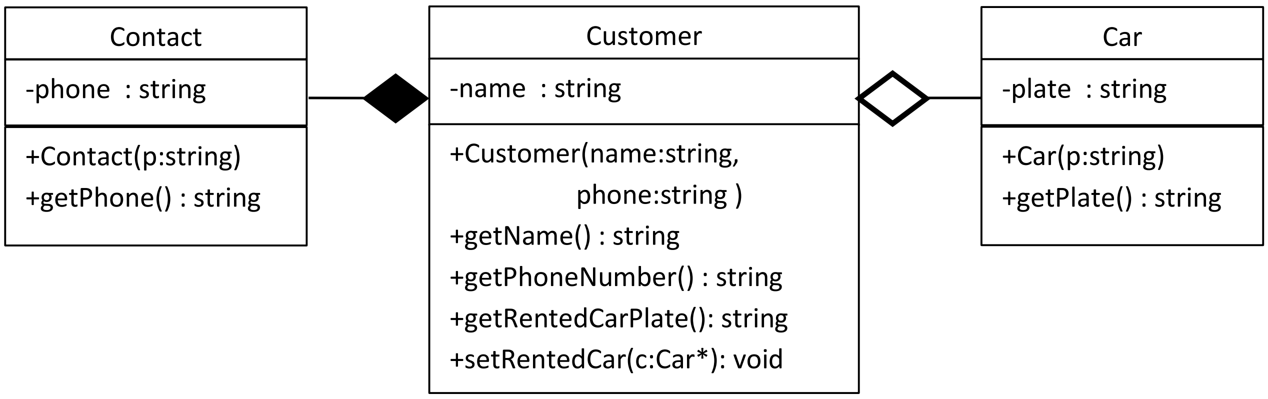

Figure 1: Class diagram for a car rental service

Based on the class diagram, write a C++ program which performs the following tasks:

1. Implement all the three classes with the given attributes and operations. Note that, the purpose of each operation is as the name implies.

2. Test the classes by creating an object of Car and an array of customers with the following data.

Customer's Name	**Phone Number**	**Rented Car Plate**
Ahmad Kamal	015-75769800	JSQ245
Siti Nurdiana Abdullah	014-8889900	

Note that, the column **"Rented Car Plate"** for the second customer is empty because she does not rent any car at the moment.

3. Print the array of customers onto the screen. The screen output should look like as in **Figure 2**.

```
Customer's Name: Ahmad Kamal
Phone Number: 015-75769800
Rented Car  : JSQ245

Customer's Name: Siti Nurdiana Abdullah
Phone Number: 014-8889900
Rented Car  :
```

Figure 2: Screen output

Question 2

Consider the class diagram in **Figure 3** which illustrates the data model for supervisions of student projects. Each student can only have a project and is supervised by a lecturer.

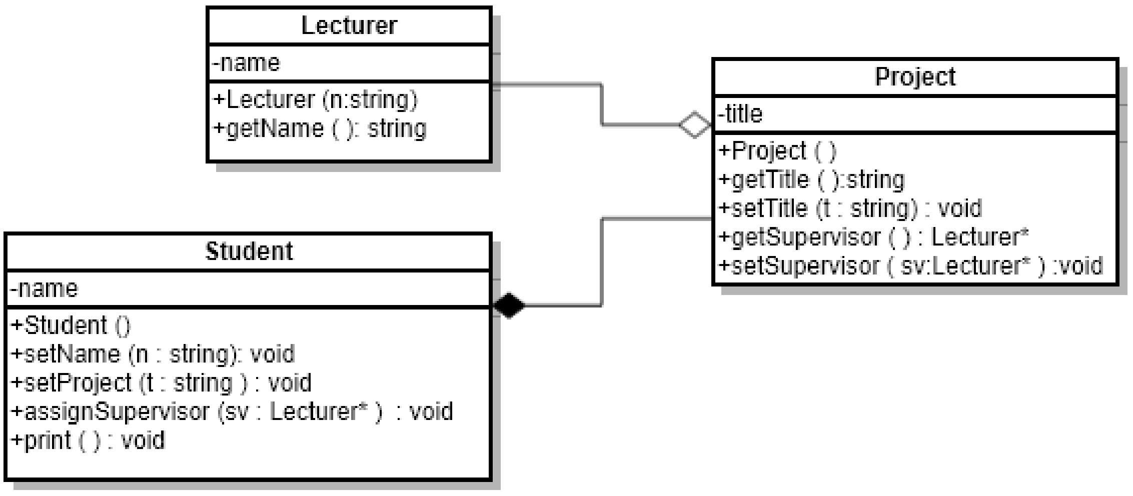
Lecturer
-name
+Lecturer (n:string)
+getName ( ): string
Project
-title
+Project ( )
+getTitle ( ):string
+setTitle (t : string) : void
+getSupervisor ( ) : Lecturer*
+setSupervisor (sv:Lecturer*) :void
Student
-name
+Student ()
+setName (n : string): void
+setProject (t : string) : void
+assignSupervisor (sv : Lecturer*) : void
+print ( ) : void

Figure 3: Class diagram for project supervisions

Write a complete C++ program based on the following tasks:

4. Class `Project` has the following methods:
   a. the constructor
   b. `getSupervisor`
   c. `setSupervisor`

5. Class `Student` has the following methods:
   a. the constructor
   b. `setName`
   c. `setProject`
   d. `assignSupervisor`
   e. `print`: to display the student's name and project's title, and also the supervisor's name (but only if the student has a supervisor).

6. Create two objects of `Lecturer` for “Dr. Ali Bakar” and “Prof. Dr. Abu Samah Abdullah”.

7. Then create an array of objects of `Student` with maximum number of students to store into the array is 15.

8. Read a list of students consisting of names and project titles, from an input file and store them into an array.

9. Assign supervisors to students as follows:
   a. The first lecturer is assigned to be the supervisor for the first and second students.
   b. The second lecturer is assigned to the last student.

10. Print all the students. The screen output should look like as in **Figure 4** and **Figure 5**.

```
Student    : Alina Atan
Project    : Anti-Intrusion System
Supervisor : Dr. Ali Bakar

Student    : Siti Nurdiana Abdullah
Project    : CCTV-Based Fire Alarm System
Supervisor : Dr. Ali Bakar

Student    : Azrul Malik
Project    : Low Energy Drone
Supervisor : Prof. Dr. Abu Samah Abdullah
```

Figure 4: Screen output with the input file, **`student_list1.txt`**

```
Student    : Sit Aminah
Project    : Cost-Effective Green Fuel
Supervisor : Dr. Ali Bakar

Student    : Kamarul Ariffin
Project    : IoT-based Flood Monitoring
Supervisor : Dr. Ali Bakar

Student    : Abdul Jabar
Project    : Energy Saving with IoT

Student    : Kamariah Jalil
Project    : Camera-based Heat Detection

Student    : Seman Abdullah
Project    : Image-based Search Engine

Student    : Rozita Abdul
Project    : Programmer Friendly Metaprogramming
Supervisor : Prof. Dr. Abu Samah Abdullah
```

Figure 5: Screen output with the input file, **`student_list2.txt`**

Question 3

XYZ Pvt. Ltd. is a company that sells various item and has several stores throughout Malaysia. Every year, the HQ will collect sales data from all of its stores in paper form. The company is experimenting on the idea of using IT to record and store data in a file and hired a programmer to write a program. However, the programmer quits due to some unknown issue and you are tasked to complete his work.

The program was coded in C++ (see **xyzstore.cpp**) and contains two classes: **StoreManager** and **StoreData**. The relationship of the two classes is shown in **Figure 6**.

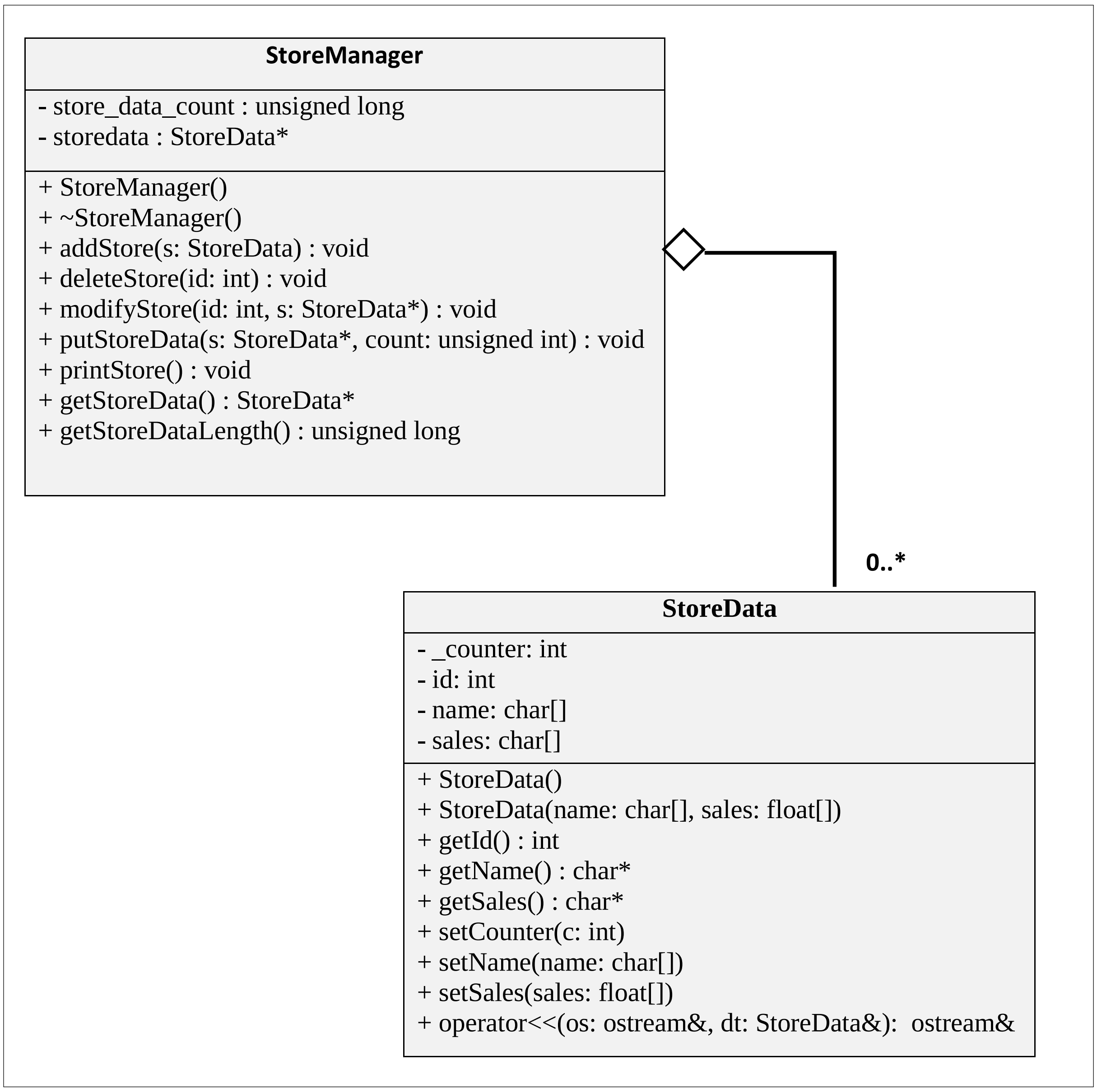
StoreManager
- store_data_count : unsigned long
- storedata : StoreData*
+ StoreManager()
+ ~StoreManager()
+ addStore(s: StoreData) : void
+ deleteStore(id: int) : void
+ modifyStore(id: int, s: StoreData*) : void
+ putStoreData(s: StoreData*, count: unsigned int) : void
+ printStore() : void
+ getStoreData() : StoreData*
+ getStoreDataLength() : unsigned long
0..*
StoreData
- _counter: int
- id: int
- name: char[]
- sales: char[]
+ StoreData()
+ StoreData(name: char[], sales: float[])
+ getId() : int
+ getName() : char*
+ getSales() : char*
+ setCounter(c: int)
+ setName(name: char[])
+ setSales(sales: float[])
+ operator<<(os: ostream&, dt: StoreData&): ostream&

Figure 6: UML class diagram

Below are details of each of the classes used in the program:

a) **StoreManager** class

   i) This class manages a dynamically allocated array of **StoreData** via '**storedata**'.

   ii) '**store_data_count**' keeps track of the number of **StoreData** pointed to by '**storedata**'.

   iii) '**addStore**' function adds **StoreData s** into the array '**storedata**'. The array is a dynamically allocated array. When a new store is added, a new memory allocation is made that could fit existing and the newly added store. Data from the previously allocated memory is copied to the newly allocated one. Once done, the previously allocated memory region is free-ed and the '**storedata**' is updated to point to the newly allocated memory.

   iv) '**deleteStore**' function deletes a **StoreData** entry pointed by '**storedata**' based on matching '**id**'. This is done by allocating a new memory region with one less space than the current one, copying all current **StoreData** to the newly allocated memory except the one with an **id** matching the one given in the parameter to the function **deleteStore**. Once done, the previously allocated memory region is free-ed and the '**storedata**' is updated to point to the newly allocated memory.

   v) '**modifyStore**' function updates '**name**' and '**sales**' of **StoreData** pointed by '**storedata**' based on the given parameter '**id**', with '**name**' and '**sales**' from **StoreData** in '**s**'.

   vi) '**putStoreData**' function updates '**storedata**' and '**store_data_count**' with '**s**' and '**count**'.

   vii) '**printStore**' function prints out data from each of the **StoreData** in the array pointed to by '**storedata**' (making use of the overloaded **operator<<** function).

   viii) '**getStoreData**' function is an accessor for '**storedata**'.

   ix) '**getStoreDataLength**' function is an accessor for '**store_data_count**'.

b) **StoreData** class

   i) This class store the '**name**' of a store and '**sales**' data (for 12 months) of a store.

   ii) There is a member called '**id**' which is set based on an internal counter '**_counter**'.

   iii) Function names starting with "**get**" are accessors while functions starting with "**set**" are mutators.

   iv) The class overloads **operator<<** to enable easy display of its internal data (via **cout**) such as '**id**', '**name**', and '**sales**'.

Based on the class diagram and description of classes given above, complete the program (**xyzstore.cpp**) that does the following tasks. ***IMPORTANT NOTE:*** <u>**Do not modify existing code**</u> in the template given.

Task 1: Write a default constructor for **StoreData** that sets or initializes **id**, **name**, and **sales** to 0.

Task 2: Write an accessor function for **StoreData**'s **id**.

Task 3: Write an accessor function for **StoreData**'s **name**.

Task 4: Write an accessor function for **StoreData**'s **sales**.

Task 5: Write a mutator function for **StoreData**'s **_counter**.

Task 6: Write a mutator function for **StoreData**'s **name**.

Task 7: Write a mutator function for **StoreData**'s **sales**.

Task 8: Write an overloaded **operator<<** function to provide access to the value of **id**, **name**, and **sales** of **StoreData**.

Task 9: Write a destructor for **StoreManager** that will deallocate any allocated memory for **storedata**, if any.

Task 10: In '**addStore**' function

(a) : Allocate memory to **temp** to contain current and new **StoreData**.

(b) : Copy existing **StoreData** to newly allocated space.

(c) : Copy new **StoreData s** to end of newly allocated space.

(d) : Update **store_data_count** to mark current size of **storedata**.

(e) : Allocate memory for **StoreData** array of one element.

Task 11: Search for **StoreData** in **storedata** with **id** matching the one given in the <u>first parameter</u> (**id**), and set it's **name** and **sales** to the one in **second parameter**. Exit the function once done.

Task 12: In '**printStore**' function

(a) : Print "***No data to print!***" message if there is no **StoreData** added, and exit the function.

(b) : Print all **StoreData** in **storedata**.

Task 13: Write an accessor function for **StoreManager**'s **storedata**.

Task 14: In standalone '**write**' function

(a) : Open **filename** for output in binary mode.

(b) : Write all **StoreData** in **s** to the file.

(c) : Close the opened file**.**

Task 15: In standalone '**load**' function.

(a) : Open **filename** for input in binary mode.

(b) : Display an error message and exit function if unable to open the file.

(c) : Get size of the file and assign it to **file_length**.

(d) : Calculate the number of **StoreData** objects in the file.

(e) : Allocate memory to contain all **StoreData** objects.

(f) : Read all data from the file into the newly allocated memory.

(g) : Close the opened file.

Sample input/ output of the program is shown in the **Figure 7**. ***Note:*** The one in **bold** are keyboard input to the program. Make sure that your code matches the output/ input sample given.

```
Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 1
Enter store name: skudai
Enter sales data : 11 11 45 78 56 25 36 25 14 85 23 65


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 1
Enter store name: lol
Enter sales data : 45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     lol      45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
```

```
[6] Load sales data from file
-
[0] Exit
 Select task : 2
Enter id of store to modify : 2
Enter new store name : mersing
Enter new sales data : 45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 5
[SAVE] Enter filename : record.dat


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 1


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
```

```
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 1
Error !!! Id 1 not found !


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 3
Enter id of store to Delete : 2


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4
 No data to print !


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
```

```
[6] Load sales data from file
-
[0] Exit
 Select task : 6
[LOAD] Enter filename : record.dat


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 4

Sales data (id, name, sales) :
[1]     skudai   11 11 45 78 56 25 36 25 14 85 23 65
[2]     mersing  45 56 58 52 36 58 47 58 69 52 41 25


Welcome to XYZ Pvt Ltd -=<[XYZ]>=-
[1] Add new sales data
[2] Modify sales data
[3] Delete sales data
[4] Displays sales data
-
[5] Write sales data to file
[6] Load sales data from file
-
[0] Exit
 Select task : 0
Thank you ! :)
```

Figure 7: Sample input/ output of program