ASSIGNMENT 1 (SECJ1023)

PROGRAMMING TECHNIQUE II

SECTION 03 & 04, SEM 2, 2020/2021

INSTRUCTIONS TO THE STUDENTS

- *This assignment must be done in a group consisting of 3 members.*
- *Please refer to the group list to find out your group members.*
- *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 programs/assignments will get **ZERO** marks (both parties, students who copied, and students that share their work).*
- *Please enter the **name and matrics number of your group members**, and the completion date as comments in your program.*
- *Each group **must answer 2 out of 3** of the following questions. **Question 1** is **compulsory** and one of Question 2 or Question 3.*

SUBMISSION PROCEDURE

- *Please submit this assignment no later than **May 23, 2021, Sunday (00:00 MYT)**.*
- *Only one submission per group is required for the submission which is the source codes (the file with the extension .cpp) and the input files (the text file with the extension *.txt) if any.*
- *Submit the assignment via the UTM's e-learning system.*

QUESTION 1

You are being appointed as the developer of a new record management system in a clinic to manage the patient and doctor records. The system would be based on the UML class diagram in **Figure 1**.

You must write a complete C++ program based on the instruction below:

a. Write a class named **Name** that consists of information such as first name and last name. The **Name** class consists of a constructor with two arguments which are first name and last name. It also contains the methods that access the full name.

b. Write a class named **Patient**. The class consists of specific information such as patient ID, gender, age, the aggregation of the class of **Address** as well as the composition of **HealthInfo** and **Name** classes. The **Patient** class also provides functions such as a constructor with default arguments to initialize a patient with their patient ID, names, gender, address, height, and weight. It also contains the accessor methods to get the information of patient ID, gender, age, address, doctor, full name from the class of **Name**, and weight and height from the class of **HealthInfo**.

** Every time a patient is added, a message will be displayed such as the following:

"Patient: <full name here> has been added"

c. Write a class named **Doctor**. The class consists of a constructor that accepts three arguments which are staff ID, first name, and last name.

d. The **Doctor** class is associated with the **Patient** class where each patient is attended by one of the doctors. Information of the doctors can be associated in **Patient** class via `attendBy(Doctor)` function that saves the information of the doctor into the attribute of **Doctor** in **Patient.**

e. Write a class named **Address** which is aggregated into **Patient** class. The class consists of information such as street name, city, postal code, and state. This class has a constructor that takes up four arguments for the address info as well as an accessor method to get the complete address.

f. Write a class named **HealthInfo** which contains the specific measurements for a patient (height, and weight). The class has a two-argument constructor. The class also consists of the accessor methods for those attributes.

g. You are given the incomplete **main()** function (refer to **Figure 2**). You are required to do the following tasks:

   i. Initialize **THREE** patients and addresses based on **Table 1** below.

   ii. Then, create a dynamic list to hold the objects of **Patient**.

   iii. Save the patient in (i) into the dynamic list in (ii).

   iv. After initializing the three patients and saved into the list, the program will invoke the `displayRecord` function (refer to **Figure 2**) to display the list of patients as shown in **Figure 3.**

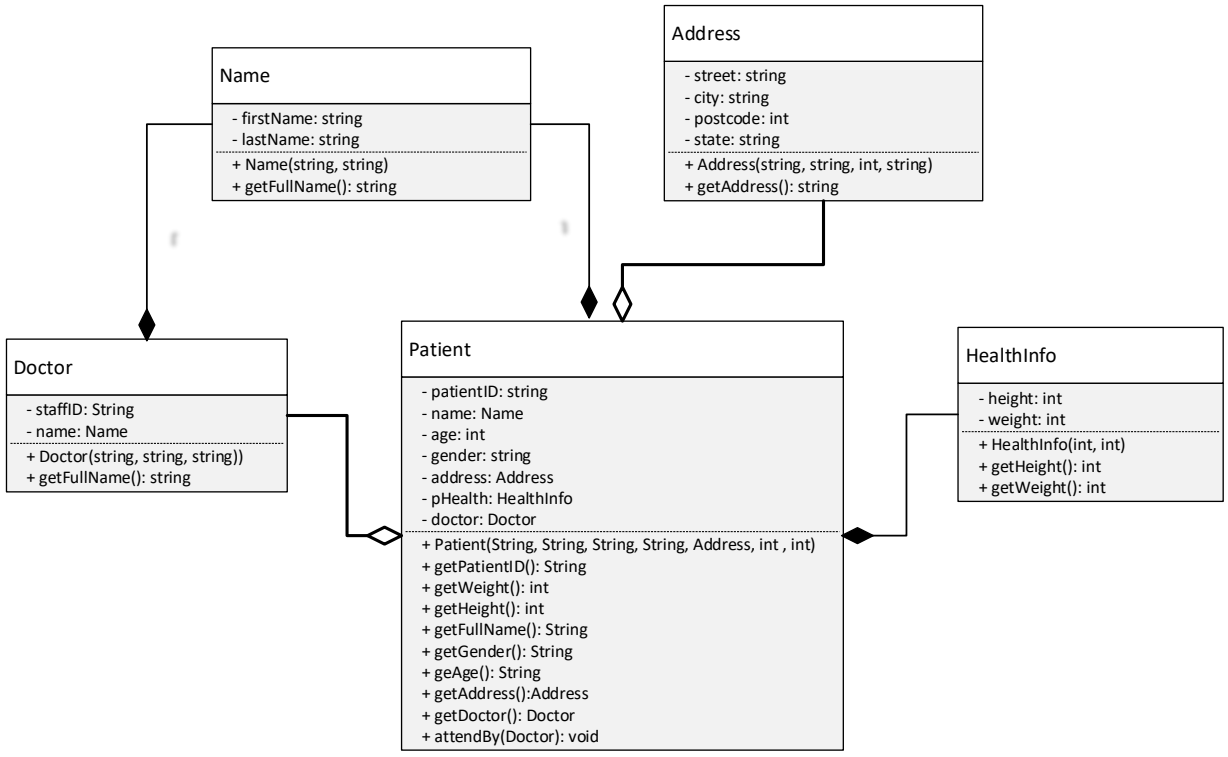

Figure 1: UML Diagram

Table 1: Patient info

Patient ID	Name	Gender	Street	City	Postal code	State	Height	Weight	Attend by
P0001	Akmal Adnan	Male	Jalan Pahlawan	Skudai	81300	Johor	180	82	Dr. Kumar Moorthy
P0002	Syafiq Yusof	Male	Jalan Flora	Skudai	81300	Johor	186	80	Dr. Kumar Moorthy
P0003	Mei Ling Koh	Female	Jalan Bakti	Skudai	81300	Johor	168	45	Dr. Stephen Koh

Table 2: Doctor info

Staff ID	Name
S0001	Kumar Moorthy
S0002	Stephen Koh

```
1   //This function display the record of the patient
2   void displayRecord(Patient pL[]) {
3      cout << "\nPatient Record Management System" << endl
4           << "================================" << endl << endl
5           << setw(4) << "No" << setw(10) << "PatientID" << setw(15) << "Name"
6           << setw(5) << "Age" << setw(8) << "Gender" << setw(37) << "Address"
7           << setw(8) << "Height" << setw(8) << "Weight" << setw(20) << "Attend By"
8           << endl << setw(4) << "--" << setw(10) << "---------" << setw(15)
9           << "----" << setw(5) << "---" << setw(8) << "------" << setw(37)
10          << "-------" << setw(8) << "------" << setw(8) << "------" << setw(20)
11          << "---------" << endl;
12
13     for(int i=0; i < Patient::numPatient; i++)
14       cout << setw(4) << (i+1) << setw(10) << pL[i].getPatientID() << setw(15)
15            << pL[i].getFullName() << setw(5) << pL[i].getAge() << setw(8)
16            << pL[i].getGender() << setw(37) << pL[i].getAddress()->getAddress()
17            << setw(8) << pL[i].getHeight() << setw(8) <<  pL[i].getWeight()
18            << "Dr. " << pL[i].getDoctor()->getFullName() << endl;
19
20     cout << "\n--------------------" << endl;
21     cout << "Total Patients: " << Patient::numPatient << endl;
22  }
23
24  int main() {
25     //(i)Initialize the Address objects based on the information given in Table 1
26
27     //(ii)Initialize the Patient objects based on the information given in Table 1
28
29     //(iii)Initialize the Doctor objects based on the information given in Table 2
30
31     //(iv)Assign Doctor to Patient based on information given in Table 1
32
33     //(v)Create a dynamic list to hold the objects of Patient
34     //v(a)Define a Patient pointer
35     //v(b)Use a pointer in v(a) to dynamically allocate an array of Patient objects
36
37     //(vi)Save the patient in (ii) into the dynamic list in v(b)
38
39     cout << left;
40     displayRecord(patientList);
41     return 0;
42  }
```

Figure 2: `displayRecord` function and incomplete `main` function

```
Patient: Akmal Adnan has been added!
Patient: Syafiq Yusof has been added!
Patient: Mei Ling Koh has been added!

Patient Record Management System
================================

No  PatientID Name           Age  Gender  Address                              Height  Weight  Attend By
--  --------- ----           ---  ------  -------                              ------  ------  ---------
1   P0001     Akmal Adnan    25   Male    Jalan Pahlawan, 81300 Johor, Skudai  180     82      Dr. Kumar Moorthy
2   P0002     Syafiq Yusof   24   Male    Jalan Flora, 81300 Johor, Skudai     186     80      Dr. Stephen Koh
3   P0003     Mei Ling Koh   30   Female  Jalan Bakti, 81300 Johor, Skudai     168     45      Dr. Kumar Moorthy

------------------
Total Patients: 3

--------------------------------
Process exited after 0.1455 seconds with return value 0
Press any key to continue . . .
```

Figure 3: Output of the program

Question 2

Consider the class diagram in **Figure 4** which illustrates the data model for supervisions of student projects. Each student can only have a project and is supervised by a lecturer.

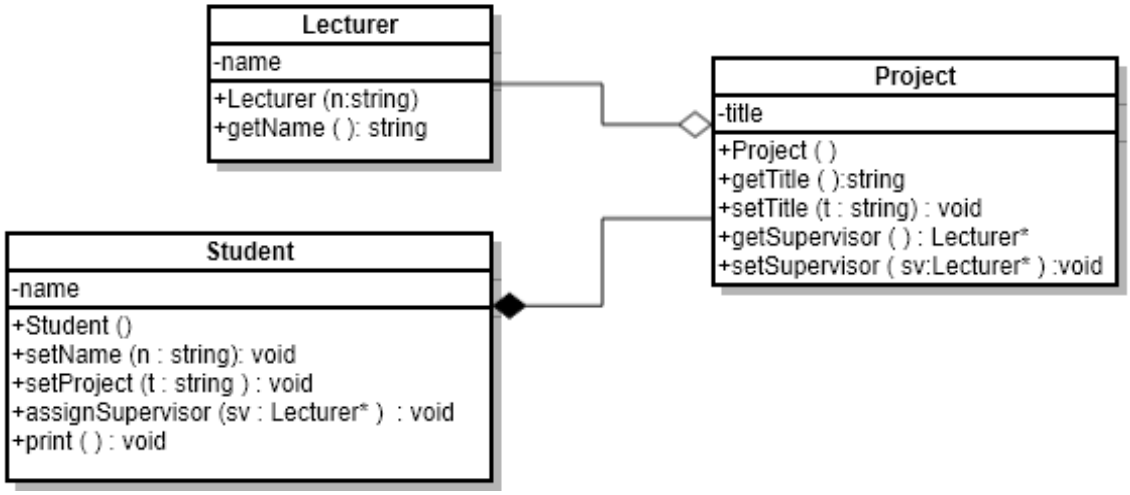

Figure 4: Class diagram for project supervisions

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

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

2. 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).
    f.
3. Create two objects of `Lecturer` for "Dr. Ali Bakar" and "Prof. Dr. Abu Samah Abdullah".

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

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

6. 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.

7. Print all the students. The screen output should look like as in **Figure 5** and **Figure 6**.

```
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 5: 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 6: 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 between the two classes is shown in **Figure 7**.

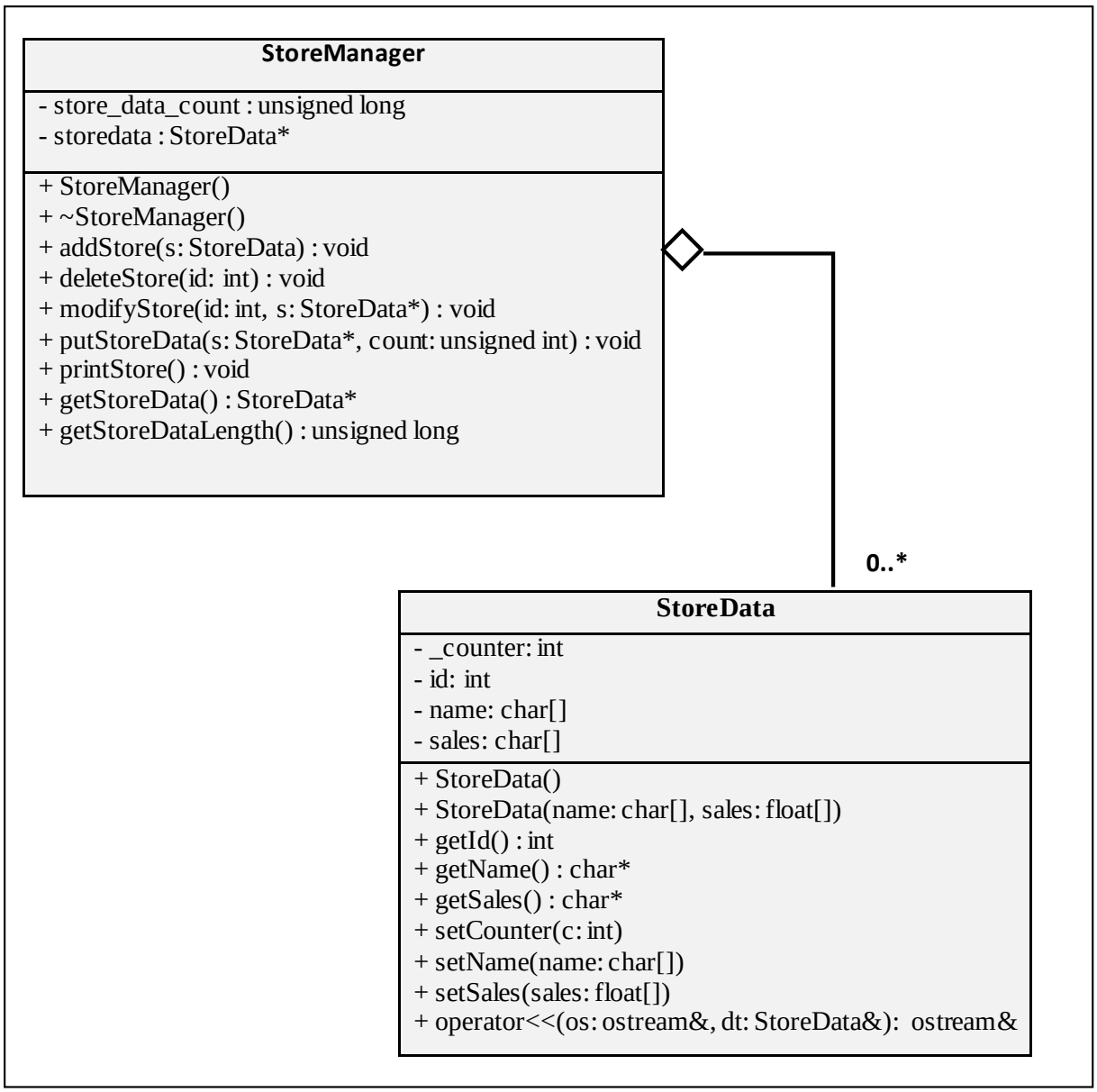

Figure 7: UML class diagram

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:*** **Do not modify existing code** 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 first parameter (**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.

The sample input/ output of the program is shown in **Figure 8**. ***Note:*** The one in **bold** is 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 8: Sample input/ output of the program