TDS2111 Data Structures and Algorithms

**INTRODUCTION**

The Club Management System assists in the administration of club memberships, as one could surmise from the name of the software. The process of joining most clubs involves filling out some paperwork, and the new member should expect to receive some kind of receipt for the membership fee that they pay. This is standard procedure in the vast majority of clubs. There is always a possibility that you may lose your membership receipt, which could lead to even more confounding circumstances. The Club Management System is something that could be created to tackle all of these issues. Through the use of this online club management system, members may sign up for the club easily. Additionally, the system also offers advantages of organized and effective operations.

**PROBLEM STATEMENTS**

The project is a Club management system console application using C++ and data structure. The objective of the project is to implement data structure efficiently to develop applications. This application helps to store data of club members and retrieves the information successfully. Club Management System Project is essentially software designed to keep track of member information in any club. It stores data such as their members' personal information. The goal of "club Management System" is to keep track of members' data. Scheduling is a technology that makes the process of informing activities and notifications in the club where it is implemented simple. The club.exe file is the application where we can add a member and his information, subscription month and subscription amount. The functions include, we can delete the member information, we can update the member information and we can display all the member information.

**OBJECTIVES**

The Smart Club Suite is designed to achieve the following objectives:

1. The whole system should be menu driven and the main menu should be displayed automatically.
2. The software can capture all the data and store it on the computer.
3. The software has a user-friendly interface and appealing appearance to ease the work of the end users.
4. The software can tell for how many months a user needs subscription
5. The new system must be designed to allow the club to record the details of its members’ date of joining, name, address, etc.
6. The system can display and edit old and new information successfully
7. Above all, the software system should be able to eliminate all the paperwork required in implementing the task of data maintenance, which previously was tedious, time consuming and error prone.

**ADT SPECIFICATION**

**Club.cpp and Club.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data members** | **Description** | **Return type** |
| displayMembers | customers (Linkedlist object) | Displays all members in the club | void |
| check | id (int), customers (Linkedlist object) | Prompts user to input member id and checks if member exists in the club | void |
| administrator | member\_name (string), address (string), phonenumber (int), substription\_month (int), substription\_amount (int), customers (Linkedlist object), waitingList (Queue object) | Prompts user to input member information and adds member to the club or waiting list | void |
| delPro | id (int), customers (Linkedlist object), waitingList (Queue object) | Prompts user to input member id and deletes member from club, adds member from waiting list if available | void |
| modify | updateid (int), choice (int), customers (Linkedlist object) | Prompts user to input member id and allows user to update member information | void |
| displayWaitingList | waitingList (Queue object) | Displays all members in the waiting list | void |

Here is a list of the data members used in the club.cpp and club.h files:

* customers: an instance of the Linkedlist class that holds a list of Member objects.
* waitingList: an instance of the Queue class that holds a list of Member objects that are waiting to be added to the customers list.
* id: an integer representing a unique identification number for a Member.
* member\_name: a string representing the name of a Member.
* address: a string representing the address of a Member.
* phonenumber: an integer representing the phone number of a Member.
* substription\_month: an integer representing the number of months for which a Member has subscribed.
* substription\_amount: an integer representing the amount paid by a Member for their subscription

**ClubManagmentSystem.cpp**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data Members** | **Description** | **Return Type** |
| displayMembers | customers (Linkedlist object) | displays the list of customers in the club | void |
| check | customers (Linkedlist object) | checks if a member exists in the club | void |
| administrator | customers (Linkedlist object), waitingList (Queue object) | adds a member to the club or the waiting list if the club is full | void |
| delPro | customers (Linkedlist object), waitingList (Queue object) | deletes a member from the club and adds a member from the waiting list if the club has space | void |
| modify | customers (Linkedlist object) | updates the information of a member in the club | void |
| displayWaitingList | waitingList (Queue object) | displays the list of members in the waiting list | void |
| main | choice (int), person (Club object), b (int), c (int) | prompts the user to login, view the rules, or exit the system, and allows the user to perform various actions in the club management system | int |

Here is a brief explanation of the data members used in ClubManagementSystem.cpp file:

* choice: an integer variable used in the main function to store the user's choice of action
* person: an object of the Club class
* b: an integer variable used in the main function to store the user's choice of action within the management dashboard
* c: an integer variable used in the main function, but it is not used in any way in the code provided

**Customer.cpp and Customer.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data Members** | **Description** | **Return Type** |
| Member(Member\*) | id, member\_name, address, substription\_month, substription\_amount, phonenumber | Constructor that creates a copy of a given Member object | void |
| Member(int, string, string, int, int, int) | id, member\_name, address, substription\_month, substription\_amount, phonenumber | Constructor that initializes a Member object with the provided data | void |
| setId(int) | id | Sets the id of the Member object | void |
| setMember\_name(string) | member\_name | Sets the name of the Member object | void |
| setaddress(string) | address | Sets the address of the Member object | void |
| setsubstricption\_month(int) | substription\_month | Sets the subscription month of the Member object | void |
| setsubstricption\_amount(int) | substription\_amount | Sets the subscription amount of the Member object | void |
| setphonenumber(int) | phonenumber | Sets the phone number of the Member object | void |
| getId() | id | Gets the id of the Member object | int |
| getMember\_name() | member\_name | Gets the name of the Member object | string |
| getaddress() | address | Gets the address of the Member object | string |
| getsubstricption\_month() | substription\_month | Gets the subscription month of the Member object | int |
| getsubstricption\_amount() | substription\_amount | Gets the subscription amount of the Member object | int |
| getphonenumber() | phonenumber | Gets the phone number of the Member object | int |
| displayinfo() | id, member\_name, address, substription\_month, substription\_amount, phonenumber | Displays the information of the Member object | void |

Here is a brief explanation of the data members used in customer.cpp file:

* int id: This is an integer representing the ID of the member.
* string member\_name: This is a string representing the name of the member.
* string address: This is a string representing the address of the member.
* int phonenumber: This is an integer representing the phone number of the member.
* int substription\_month: This is an integer representing the subscription month of the member.
* int substription\_amount: This is an integer representing the subscription amount of the member.

**linkedlist.cpp and linkedlist.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data members** | **Description** | **Return type** |
| Linkedlist() | head, maxSize | Default constructor that initialises the head member variable to NULL and the maxSize member variable to 0 | N/A |
| size() | maxSize | Returns the number of nodes in the list | int |
| deleteNode(int id) | head, temp1, temp2, maxSize | Removes a node from the list with the specified id | N/A |
| insertNode(Member \*data) | head, newNode, maxSize | Inserts a new node with a pointer to a Member object at the end of the list | N/A |
| printList() | head, temp, maxSize | Prints the data of each Member object in the list | N/A |
| getnode(int id) | head, temp | Returns a pointer to the Member object in the list with the specified id | Member\* |

Here is a brief explanation of the data members used in the Linkedlist class:

* head: a pointer to the first node in the linked list.
* maxSize: an integer that keeps track of the number of nodes in the linked list.

In addition to these data members, the member functions of the Linkedlist class also uses some local variables:

* temp1, temp2: pointers to Node objects that are used to traverse and manipulate the linked list.
* ListLen: an integer that is not used in the code.

**Node.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data members** | **Description** | **Return Type** |
| Node(Member\* data) | data, next | Constructor that initialises the data member variable to a pointer to a Member object and the next member variable to NULL | N/A |

The Node structure does not have any member functions beyond the constructor. The data and next member variables are both pointers, with data pointing to a Member object and next pointing to the next Node in a linked list. The #pragma once directive tells the preprocessor to include this header file only once in a single compilation. This can be useful in preventing multiple definitions of the same entities.

Here is a brief explanation of the data members used in the Node struct:

* data: a pointer to a Member object.
* next: a pointer to the next Node in a linked list.

**queue.cpp and queue.h**

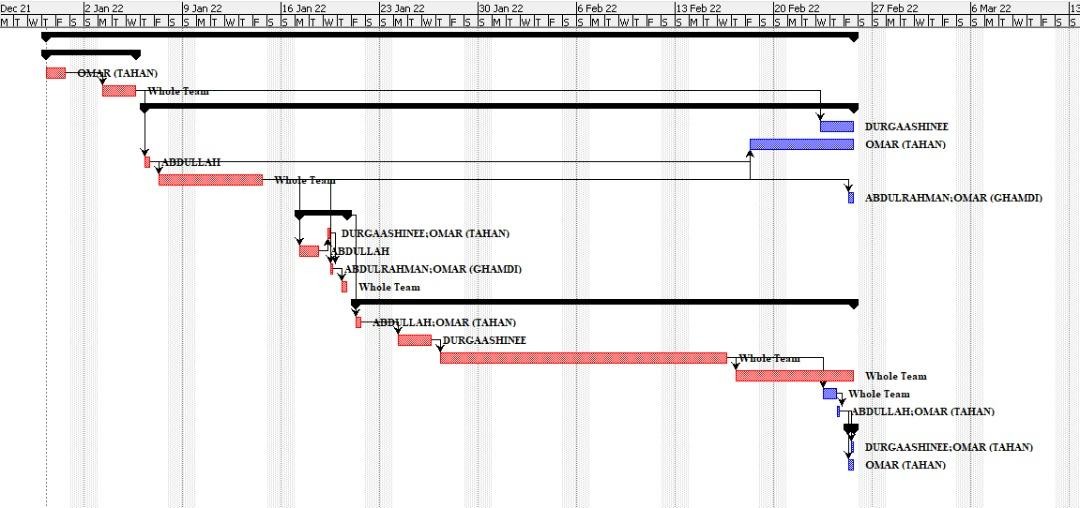
|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Data members** | **Description** | **Return type** |
| Queue() | front rear maxSize | Constructor for Queue class | N/A |
| isEmpty() | maxSize | Check if the queue is empty | bool |
| enQueue(Member \*value) | front rear maxSize | Add a Member object to the queue | N/A |
| getFront() | front | Get the Member object at the front of the queue | Member\* |
| memberDeQueue() | front maxSize | Remove the Member object at the front of the queue | N/A |
| displayTHEQueue() | front | Display the Member objects in the queue | N/A |

Here is a brief explanation of the data members used in the queue structure:

* front: a pointer to a Node object representing the front of the queue.
* rear: a pointer to a Node object representing the rear of the queue.
* maxSize: an integer representing the maximum size of the queue

**GANNT CHART AND WORK BREAKDOWN STRUCTURE**





## IPLEMENTATION DETAILS

The Data structures we have used and implemented in our project are two, Linked list, and Queue. Linked lists and queues are important data structures in C++ due to a variety of reasons as we have mentioned below.

When the extent and size of a data collection cannot be predicted in advance or when data has to be added or removed often, linked lists are frequently employed. Linked lists have the advantage of being readily enlarged and expanded since new nodes may be added or removed without having to replicate the entire list. A constant-time insertion and deletion of entries at any place in the list is also possible with linked lists which is an important feature to our system.

Queues are important for activities such as scheduling and managing requests in a system because they store data in a first-in, first-out (FIFO) fashion. They are an excellent option when adding data to the back of the queue (enqueue) and removing it from the front of the queue are the main actions on the data (dequeue). The method used to implement a queue will depend on the requirements of the application in question. Common methods include utilizing an array or a linked list. In our program we have grouped the linked list with a queue instead of an array to make the best use of our system.

Generally speaking, linked lists and queues can be an advantageous good choice when the size of a data collection is not known in advance, and when data must be constantly added to or removed from, and when the main operations on the data include adding and deleting pieces at precise points in the list or queue. Which is the exact case in our system as it is a club management system where students can register to the club every semester or leave the club, so we constantly have a flow of students coming in and out of the system, students moving around updating their addresses and phone numbers as well which is why we have implemented the edit functions.

Linked list implementation:

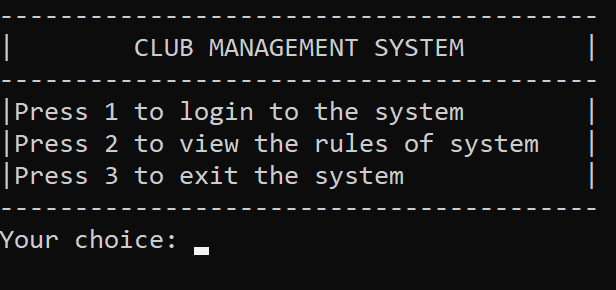
* We used the list to store each member's information.
* We set the maximum size of the list to be 10, meaning at a time only 10 members can register to the club.
* Whenever a customer registers after the maximum size of the list is reached he/she is added to the waiting list.

Queue Implementation:

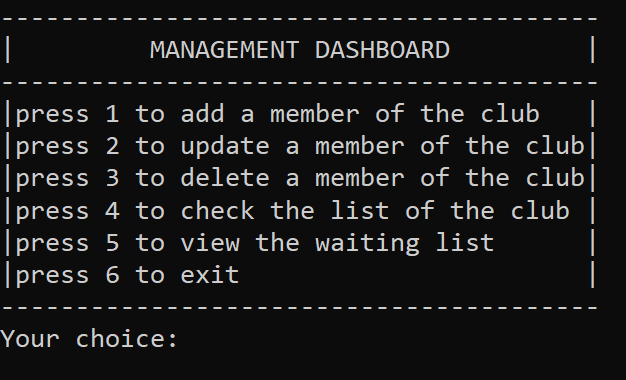
* We used the queue as the waiting list.
* When the list is full and a member is deleted a member from the queue (only if the queue is not empty) is added to the list.
* Meaning if there are 10 members in the list and 1 is deleted or removed so 9 remain.
* one member is added from the queue to the list making it 10 members again.

**PROGRAM SCREENSHOT**

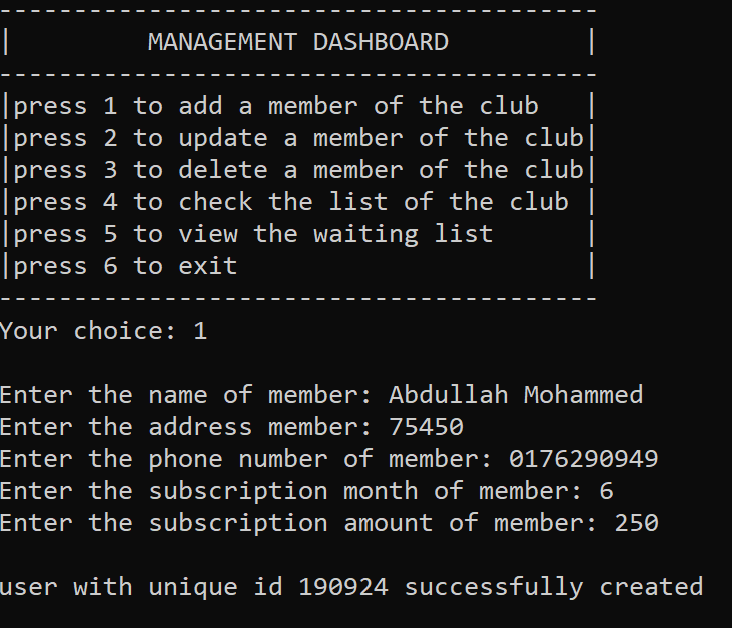
1. Default interface



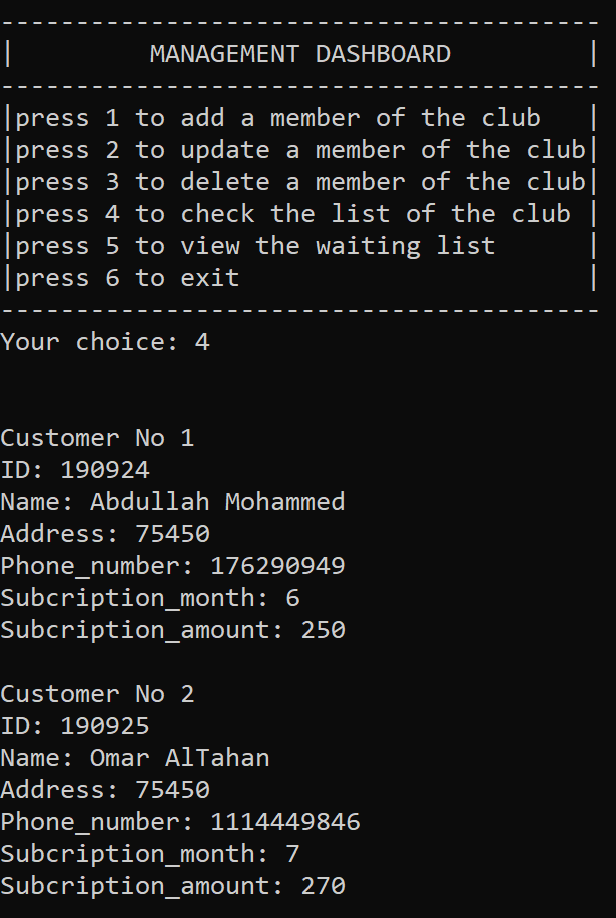
1. Management Dashboard after choosing to log in



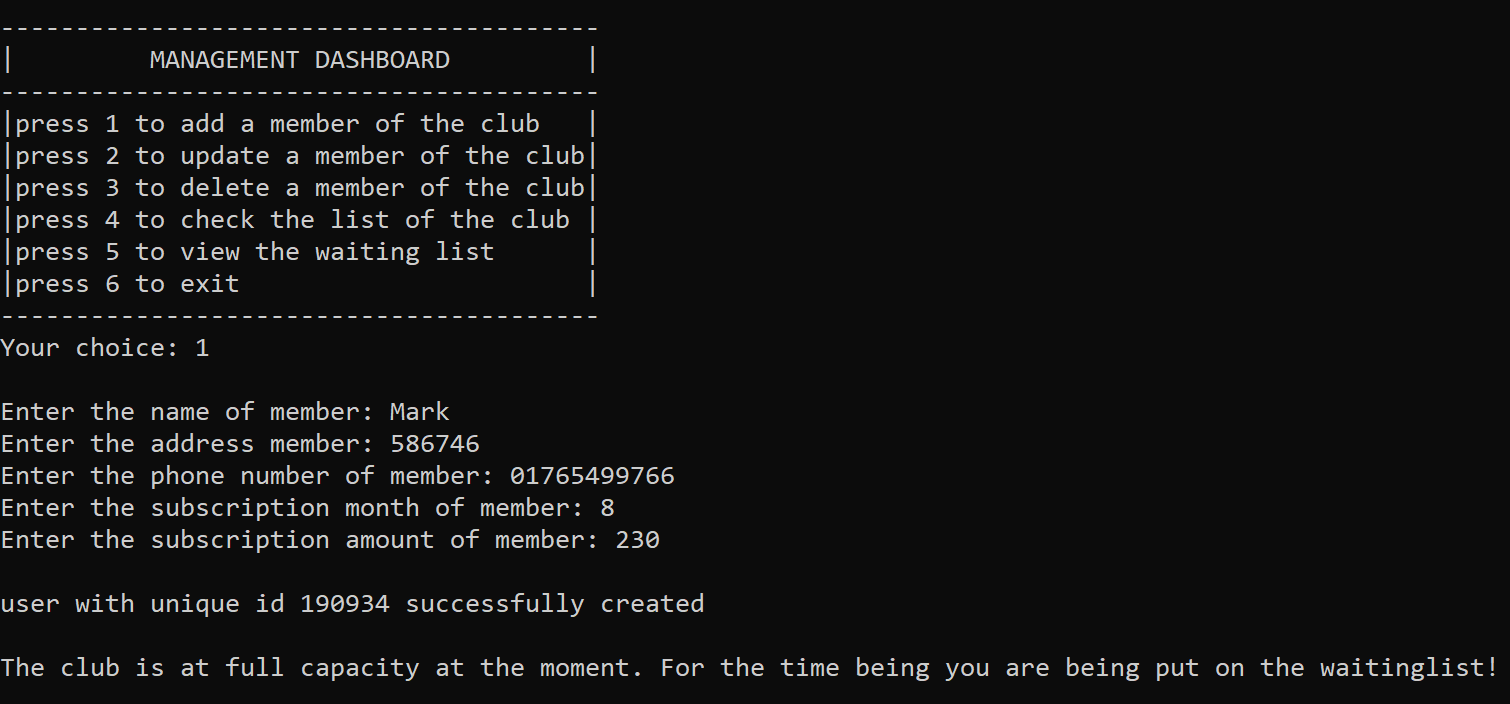
1. Add a member



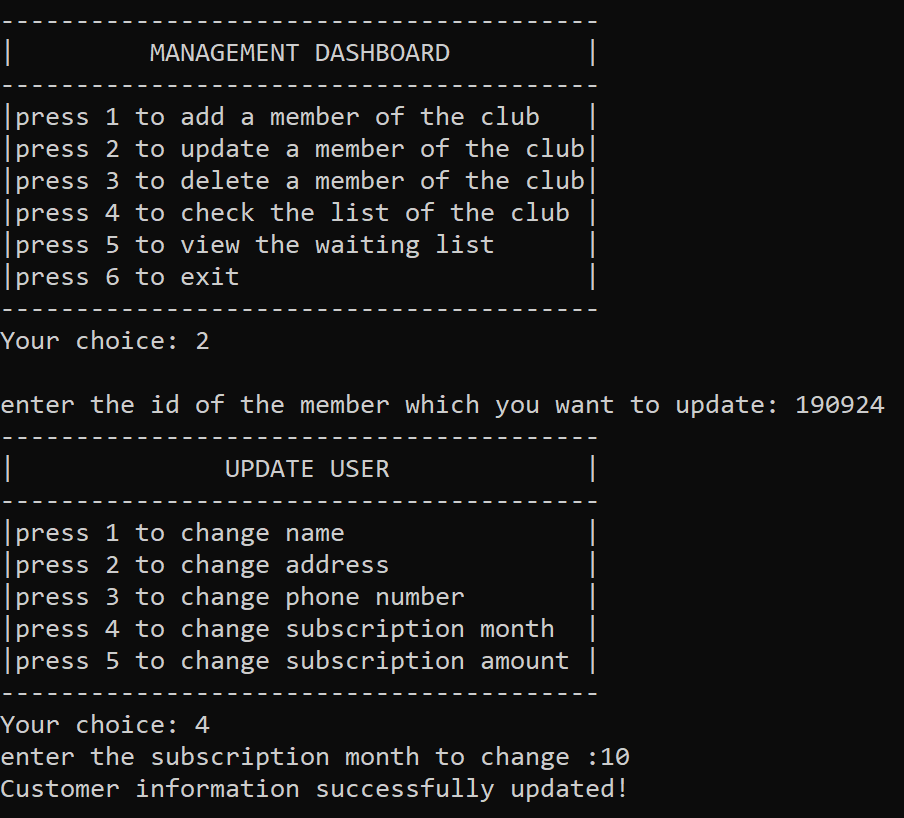
1. Check the list of registered members of the club



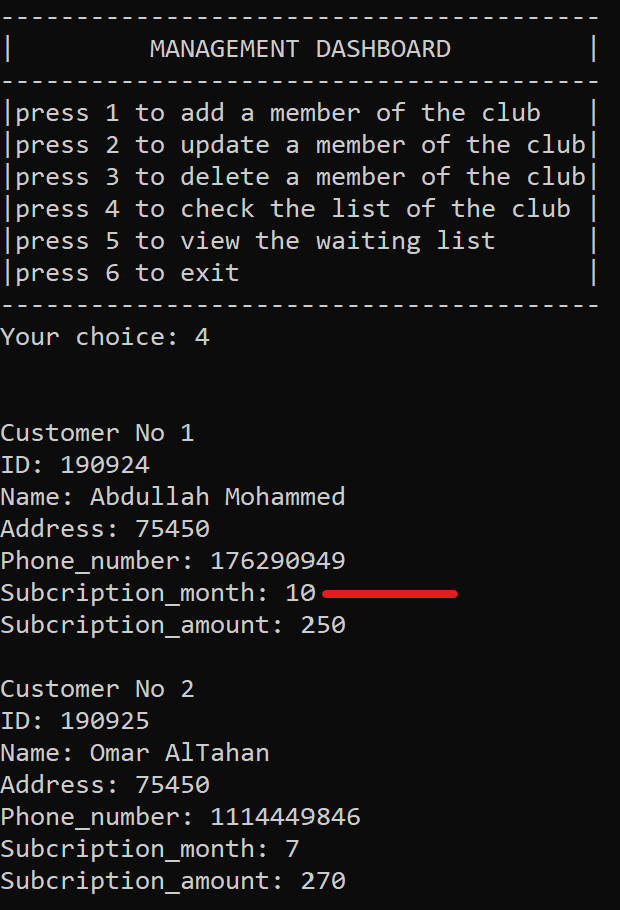
1. Once we reach the maximum number of members then we start filling the new members at the waiting list. So, Mark is in the waiting list



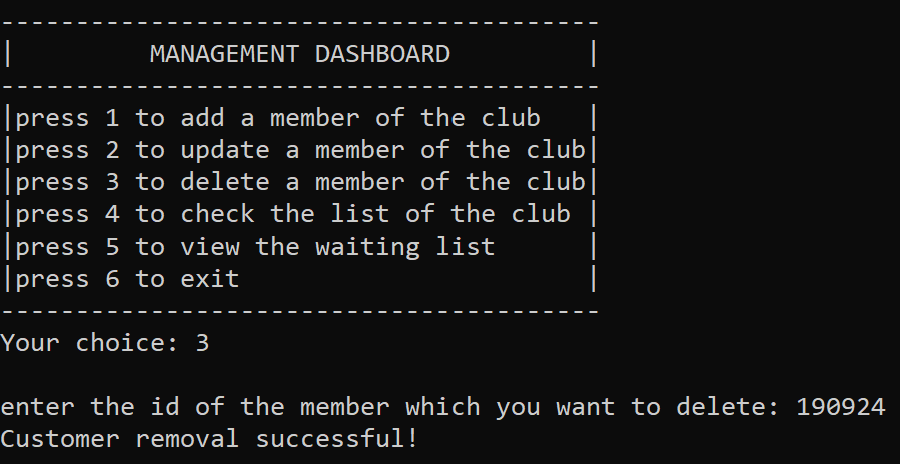
1. Update a member of the club



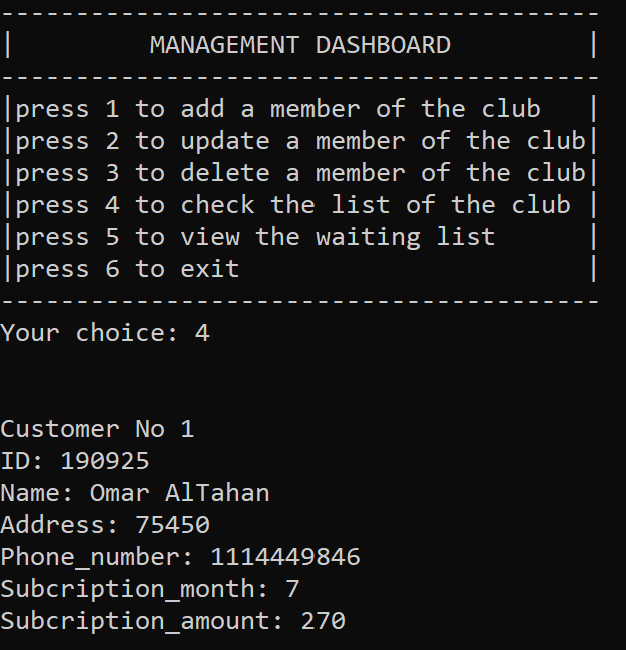
1. Now, once we check again we will find the updated version of information’s



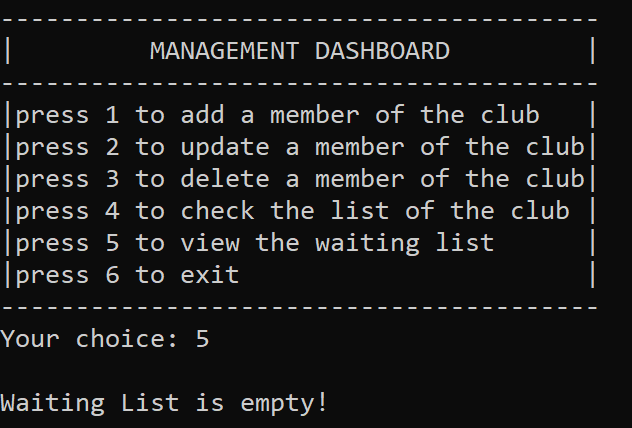
1. Deleting a member from the club



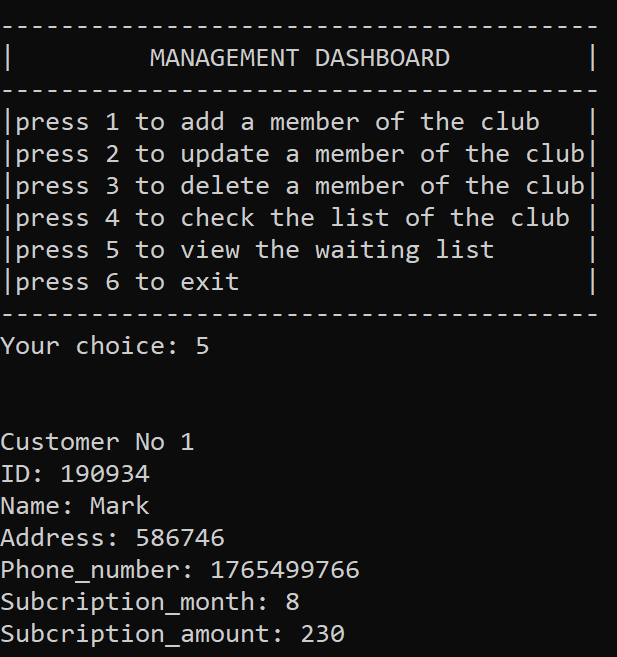
1. Once we check again, we will find the updated version of information’s



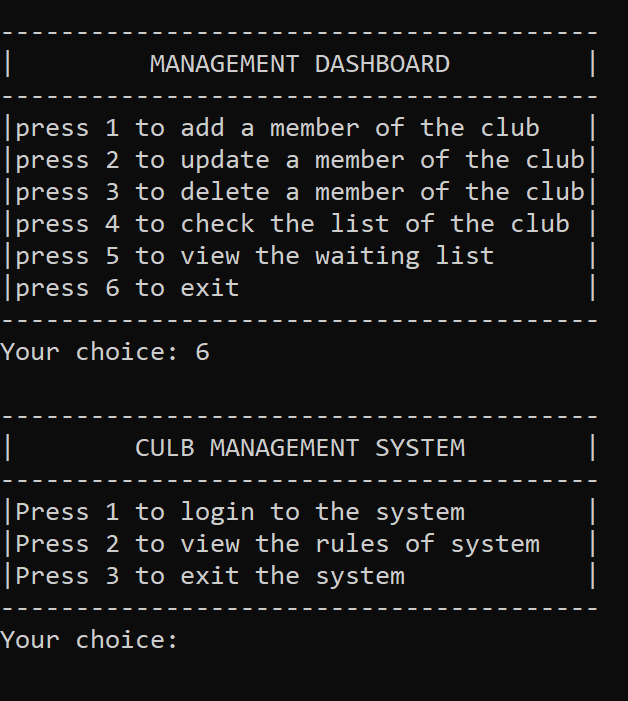
1. Viewing the waiting list in case we still have space to register new member to our club



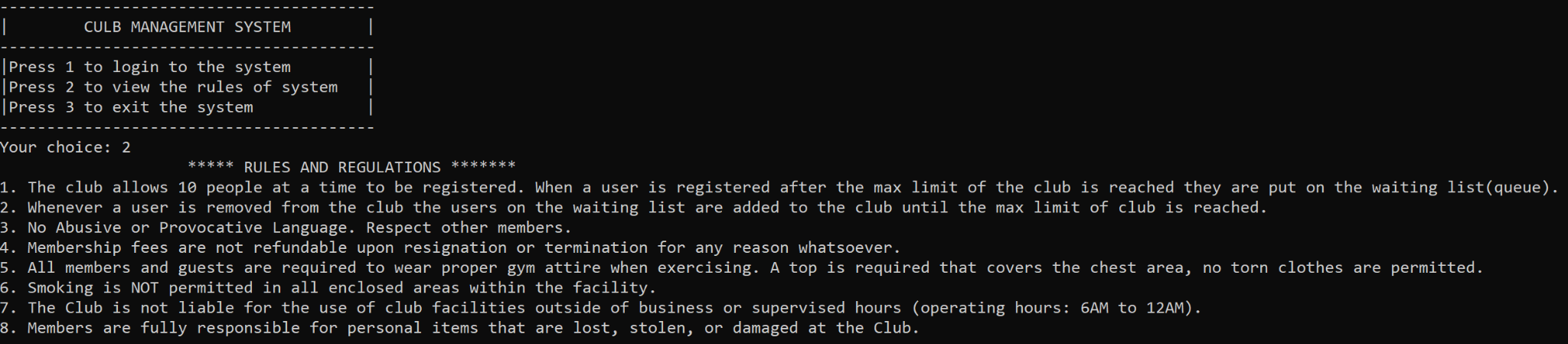
1. In case our club is full of members then we can view the waiting list



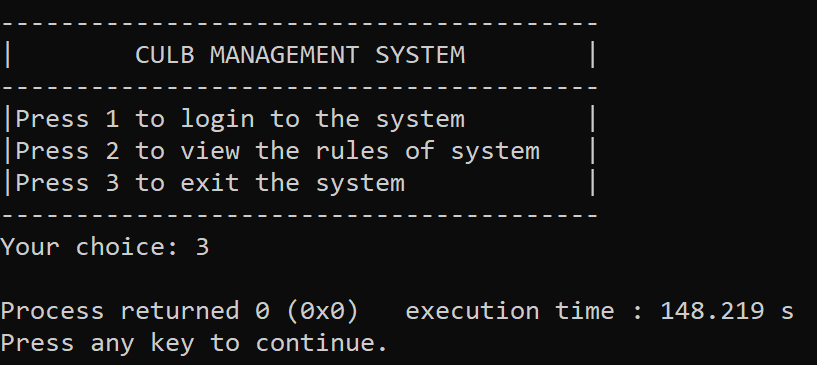
1. Once we exit from the Management Dashboard, we will return back to our default interface



1. Viewing the rules of our club



1. Exiting the system



**CONCLUSION**

Overall, our club management system program that has been designed is useful for managing a club. It provides several functions that allow administrators to easily add, delete, and modify member information, as well as display the current list of members and check if a particular customer is a member. These features make it easy to keep track of the club's members and their subscription time and expiry, it also ensures that all members are properly registered, and their information is up-to-date. In addition, the program's use of a waiting list helps to ensure that new members will be added to the club when a spot opens up in a fair and orderly manner, even when the club is at its maximum capacity. Overall, the program is a valuable resource for any club looking to streamline its membership management processes.

In conclusion we have developed a fully functional Club Management System using C++, we have implemented the use of Data Structures and Search/Sort algorithms. The code is well written and is bug-free , the program also has good error handling with proper error handling messages.