GYM MEMBERSHIP SYSTEM

TABLE OF CONTENTS

1.0 INTRODUCTION

The Gym Membership Management System is a C++ program designed to facilitate the administration and management of a gym's member database. Developed as a console application, the system offers functionalities for gym administrators. It features the creation and management of member accounts, and supports various operations such as updating member information, deleting members, searching for members, and sorting members based on different criteria.

The system incorporates key data structures, including linked lists and hash tables, to efficiently manage gym member information. Additionally, this system incorporates merge sort, a sorting algorithm to enhance the program's sorting capabilities.

This report aims to provide an overview of the Gym Membership Management System, emphasizing the integration of the data structures and algorithm, delving into its functionalities, chosen data structures and algorithm, showcasing its features through program screenshots.

Through this report, readers will gain insights into the design, implementation and functionality of the Gym Membership Management System, which can also serve as a resource for helping to understand data structures and algorithms concepts of the implemented C++ program.

2.0 PROBLEM STATEMENTS

In a world where people are becoming more aware of the importance of living a healthy lifestyle, the number of potential customers to the gym will only keep increasing. In this fast-paced world of fitness, managing gym memberships efficiently is crucial for both members and gym administrators. Thus, we aim to address that situation by creating an organized and systematic gym membership management system by implementing robust data structures and algorithms using the C++ language.

3.0 Final ADT SPECIFICATIONS

void displayMainMenu()	**Result:** Displays the first menu of the program where the admin logs in
void AdminActionsMenu()	**Result:** Displays the admin menu. The menu once the admin has logged into the system.
void adminLogin ()	**Result:** Ensures admin login details are correct to log the respective admin into the system
void createAdminAccount()	**Result:** Creates an account for the admin with a respective username and password
void createMemberAccount()	**Result:** Creates a new member in the gym system with the members name, ID, and membership type

bool isValidAdmin (const std::string& username, const std::string& password)	**Requirements:** - Username and password cannot be empty. - Username should be entirely in lower case. **Result:** Checks whether the username and password entered by the admin at the login page is valid with the one created. If it is, the admin is logged into the system. Else, the system Displays that the details are incorrect and will not be logged into the system.
bool isValidAdmin (const std::string& enteredUsername, const std::string& enteredPassword)	**Requirements:** - Entered username and passwords cannot be empty. - Username should be entirely in lower case. **Result:** checks the validity of administrator credentials

bool isValidMember (int enteredMemberID)	**Requirements:** - Entered member ID must be a positive integer **Result:** checks whether a provided member ID exists in the Gym Membership Management System.
void updateMemberInfo ()	**Requirements:** - The input memberNode must not be nullptr. - The memberNode should represent a valid and existing member in the system. **Result:** Updates the member info by either updating their name, membership type or active status.

void updateMemberFile (memberNode* member)	**Requirements:** - The input memberNode pointer must not be nullptr. - The memberNode should represent a valid and existing member in the system. **Result**: takes a pointer to a MemberNodeas a parameter. This function is responsible for updating the information of a specific member in the file "member_credentials.txt."
void loadMembersFromFile ()	**Result:** reads member data from the "member_credentials.txt" file, populates the linked list (memberList), and updates the unordered_map (memberMap).
void deleteMember()	**Result**: Deletes a member's details from the system

void searchMember()	**Result:** Searches a member's details in the system based on a search criteria (in this program, the criterias are name and membership type)
bool compareMembersByMembershipType(const MemberNode* a, const MemberNode* b)	**Requirements:** - Both input pointers (a and b) must not be nullptr. - The MemberNode pointers should represent valid and existing members in the system. - The comparison should adhere to a specific order. **Result**: determines the order of two member nodes based on their membership types. It returns true if the membership type of the first node is less than the second node, facilitating the sorting of members.

bool compareMembersByName(const MemberNode* a, const MemberNode* b)	**Requirements:** - Both input pointers (a and b) must not be nullptr. - The MemberNode pointers should represent valid and existing members in the system. - The comparison should consider the full name and be case-sensitive. **Result**: evaluates the alphabetical order of member names for two member nodes. It returns true if the name of the first node is less than the name of the second node, facilitating sorting based on member names.
void SortMembers()	**Result:** sorts members by user-defined criteria,
int generateUniqueMemberID()	**Result**: produces a unique identifier for each new member

void displayMembersInfo()	**Result:** provides a concise summary of member details, including ID, name, membership type, and status
void mergeSort(std::vector<MemberNode*>& membersVector, int left, int right)	**Requirements:** - The membersVector should be a valid vector containing MemberNode pointers. - 'left' and 'right' parameters define the range of elements to be sorted within the vector. - 'left' should be a valid index within the vector (0 <= left < membersVector.size()). - 'right' should be a valid index within the vector (0 <= right < membersVector.size()).. - Proper boundary checks and handling should be implemented to ensure the function works within the specified range. **Result:** implements the merge sort algorithm to efficiently sort a vector of MemberNode pointers based on specified indices.

void merge(std::vector<MemberNode*>& membersVector, int left,int mid, int right)	**Requirements:** - The membersVector should be a valid vector containing MemberNode pointers. - 'left', 'mid', and 'right' parameters define the subarrays to be merged within the vector. - 'left' should be a valid index within the vector (0 <= left < membersVector.size()). - 'mid' should be a valid index within the vector (left <= mid < right). - 'right' should be a valid index within the vector (mid < right < membersVector.size()).. - The MemberNode pointers within the vector should be valid and represent existing members in the system. - Proper boundary checks and handling should be implemented to ensure the function works within the specified range. **Result:** combines two sorted halves of a vector (membersVector) to create a fully sorted portion.

4.0 IMPLEMENTATION DETAILS

Linked List

One of the data structure concepts that was applied to this code is the linked list. For our program to run smoothly, the ability to efficiently handle varying member numbers is crucial. Linked lists emerge as an invaluable solution to navigate the complexities of fluctuating memberships.

Linked lists, with their dynamic memory allocation, prove invaluable in gym management systems where member numbers vary. Their dynamic size property adeptly accommodates the fluctuating list of members.

Efficient insertion and deletion set linked lists apart from arrays in dynamic environments like gyms. Adding or removing members is seamless, as new nodes are created or removed without disrupting other elements. This contrasts sharply with arrays, where shifting is necessary.

The absence of pre-allocation in linked lists optimizes memory usage, a significant advantage in gym systems where member counts are unpredictable. This eliminates the need to estimate maximum members, avoiding potential memory wastage or shortage.

Updating member information in a linked list is straightforward, involving direct modification of specific nodes without the computational expense of shifting elements. This stands in stark contrast to arrays, where updating necessitates shifting, proving less efficient.

Sorting members based on criteria is efficiently handled by linked lists. A vector is created for sorting, showcasing the flexibility of linked lists. Post-sorting, the vector can be converted back to a linked list, allowing for effective data manipulation.

Linked lists, known for their simplicity, are particularly suited for beginners. Their easy implementation and management free developers to concentrate on the gym management system's functionality, such as membership handling and billing, without being bogged down by complex data manipulation.

Linked lists stand out as a robust choice, seamlessly adapting to the dynamic nature of member numbers. Their efficiency in insertion, deletion, and sorting, coupled with simplicity, positions linked lists as a key asset for this program.

Hash Table

One of the reasons our group decided to implement hash tables into our project was for the swift and accurate retrieval of member information. Hash tables, with their constant-time average complexity, become a cornerstone in ensuring quick access, efficiency, and reliability based on unique member IDs.

Hash tables offer constant-time average complexity for search and retrieval, crucial in a gym management system for quick access based on unique member IDs. This ensures seamless member login, efficient information updates, and easy membership validity checks.

Enforcing key uniqueness in hash tables prevents duplicate member IDs, maintaining a one-to-one correspondence between IDs and member data. This enhances efficiency and reliability, but it's crucial to consider the specific system requirements when choosing data structures.

Similar to linked lists, hash tables provide efficient insertion and deletion operations with constant average time complexity. This is vital for accommodating changes in the gym's member count over time.

Unlike arrays, hash tables dynamically resize to handle varying elements, eliminating the need for manual intervention. This flexibility is especially advantageous in a gym system where member numbers fluctuate.

Hash tables offer flexibility in storing additional member-related information, beyond the member ID, enhancing retrieval efficiency. This versatility simplifies operations, allowing the system to access all member information using a single ID lookup.

Member validation is streamlined with hash tables, as the system quickly verifies existence and retrieves associated information based on member IDs. This simplifies tasks like checking member status or updating details.

In summary, the gym management system benefits from the efficiency and flexibility of hash tables, ensuring quick access, preventing duplicates, accommodating dynamic changes, and simplifying member-related operations. While linked lists offer simplicity, hash tables provide a robust solution for managing diverse member information seamlessly in a dynamic environment.

Merge Sort

In the intricate operations of a gym management system, the stability of sorting algorithms holds significant importance, particularly when arranging members based on diverse criteria. Which is one of the reasons our group chose to implement merge sort, as it is renowned for its stability.

Merge sort is a stable sorting algorithm, meaning that it maintains the relative order of equal elements. In the gym management system, stability is essential when sorting members based on multiple criteria (e.g., name or membership type). This ensures that members with the same key value will retain their original order, providing a consistent and predictable sorting outcome.

Merge sort is well-suited for linked lists. Unlike other sorting algorithms like quicksort or heapsort, merge sort doesn't rely on random access to elements, making it efficient for linked lists where sequential access is more natural. Merge sort's divide-and-conquer approach aligns with the structure of linked lists, facilitating the creation of sorted sublists that are later merged.

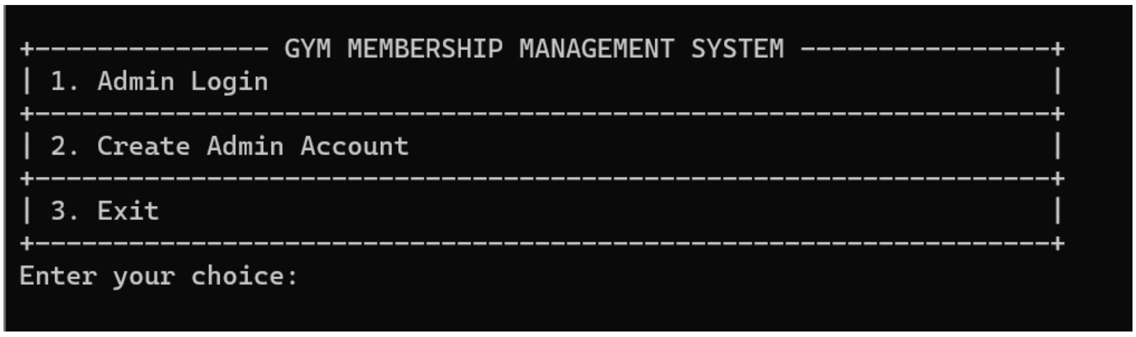
Merge sort guarantees a consistent O(n log n) time complexity for the worst, average, and best cases. This predictability makes it a reliable choice for sorting operations in a gym management system, where maintaining a consistent level of performance is crucial for user experience.

Merge sort's efficiency becomes more apparent when dealing with large datasets. Its logarithmic time complexity ensures reasonable performance even as the number of gym members grows. This scalability is vital for a gym management system that may need to handle a substantial number of members over time.

While in-place sorting algorithms like quicksort can be efficient, they often involve complex pointer manipulations in the case of linked lists. Merge sort, on the other hand, doesn't perform in-place sorting, making it simpler to implement and manage, especially when working with linked data structures.

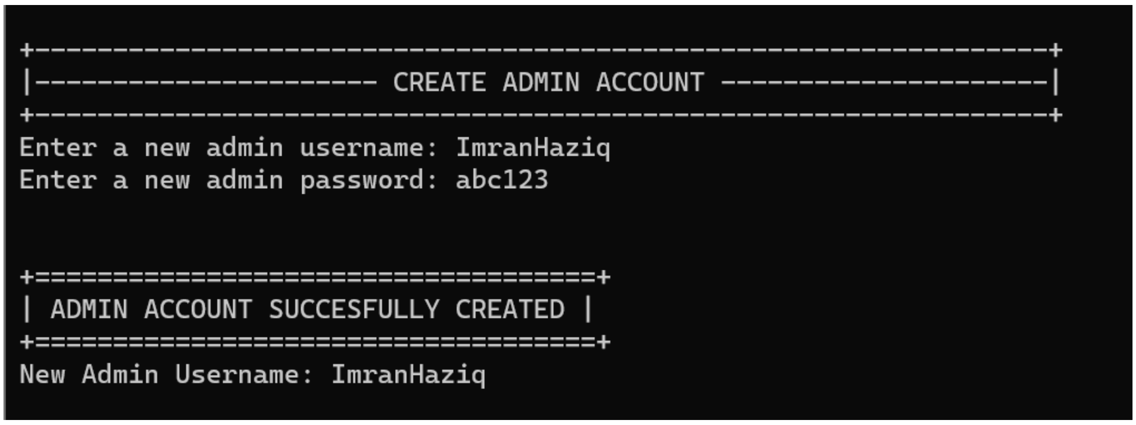
Merge sort's divide-and-conquer strategy results in consistent memory usage, which is essential in resource management. In a gym management system, where the system's responsiveness is critical, avoiding excessive memory fluctuations contributes to a smoother user experience.

5.0 PROGRAM SCREENSHOT



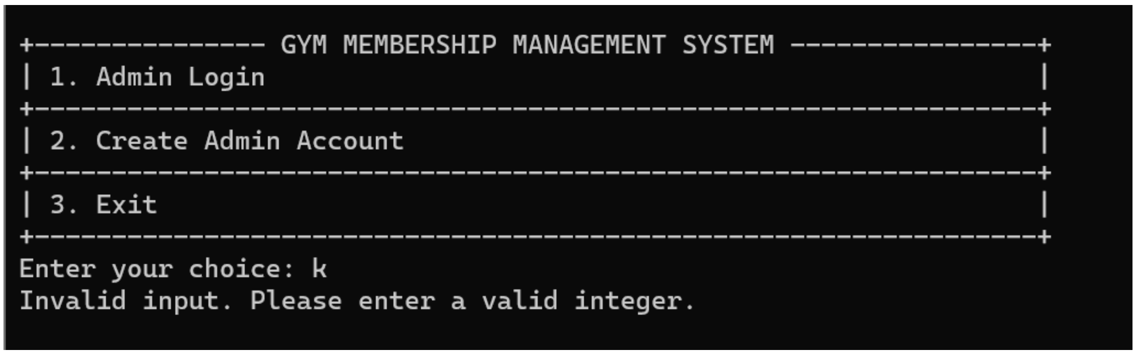
```
+--------------- GYM MEMBERSHIP MANAGEMENT SYSTEM ----------------+
| 1. Admin Login                                                  |
+-----------------------------------------------------------------+
| 2. Create Admin Account                                         |
+-----------------------------------------------------------------+
| 3. Exit                                                         |
+-----------------------------------------------------------------+
Enter your choice:
```

Home Page



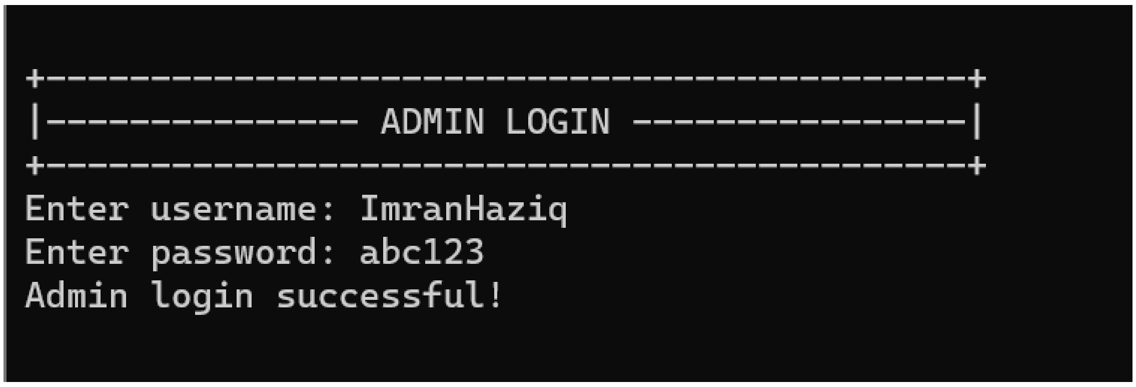
```
+-----------------------------------------------------------------+
|--------------------- CREATE ADMIN ACCOUNT ---------------------|
+-----------------------------------------------------------------+
Enter a new admin username: ImranHaziq
Enter a new admin password: abc123


+===================================+
| ADMIN ACCOUNT SUCCESFULLY CREATED |
+===================================+
New Admin Username: ImranHaziq
```

Create Admin Account



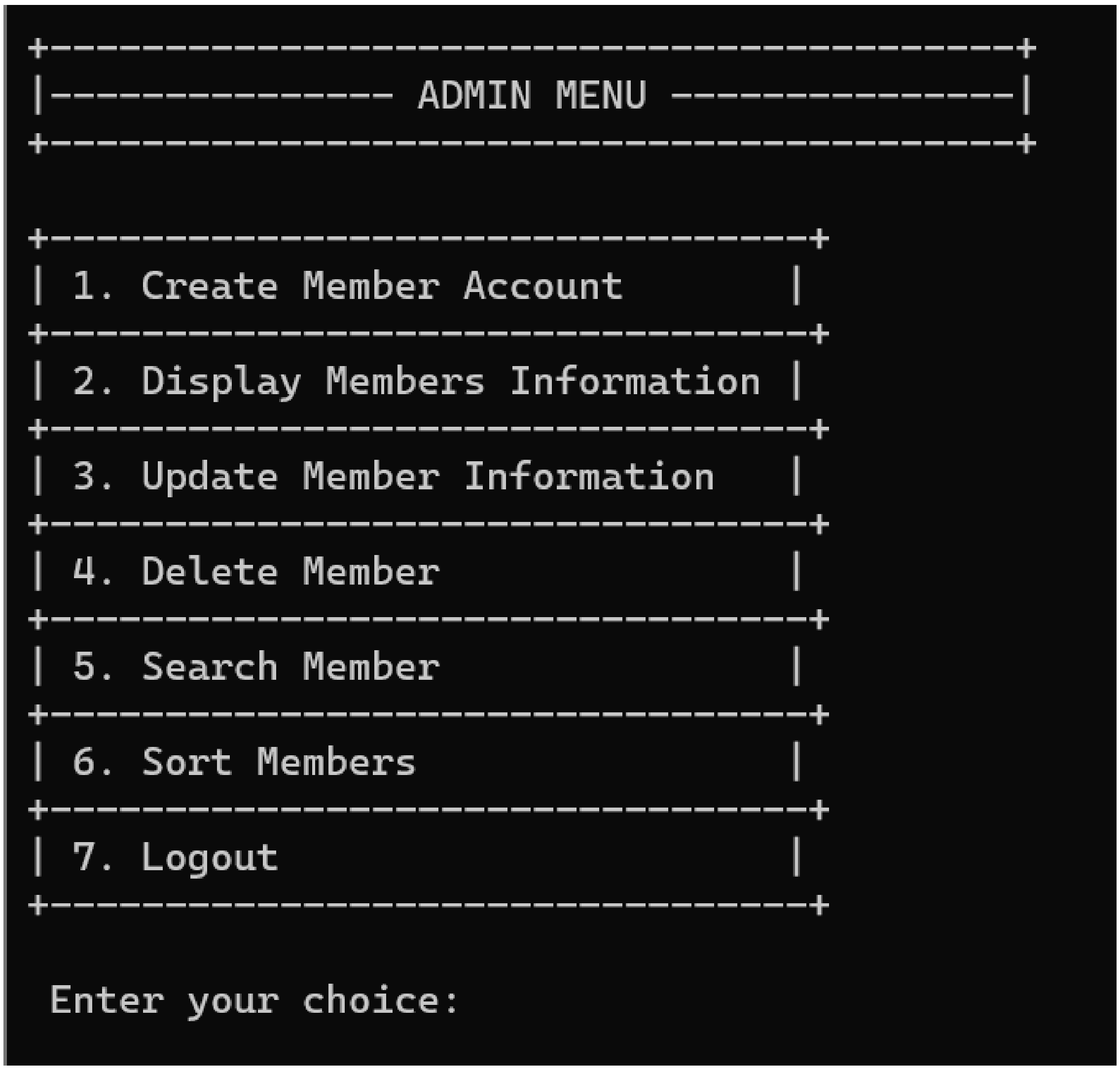
```
+--------------- GYM MEMBERSHIP MANAGEMENT SYSTEM ----------------+
| 1. Admin Login                                                  |
+-----------------------------------------------------------------+
| 2. Create Admin Account                                         |
+-----------------------------------------------------------------+
| 3. Exit                                                         |
+-----------------------------------------------------------------+
Enter your choice: k
Invalid input. Please enter a valid integer.
```

Exception Handling



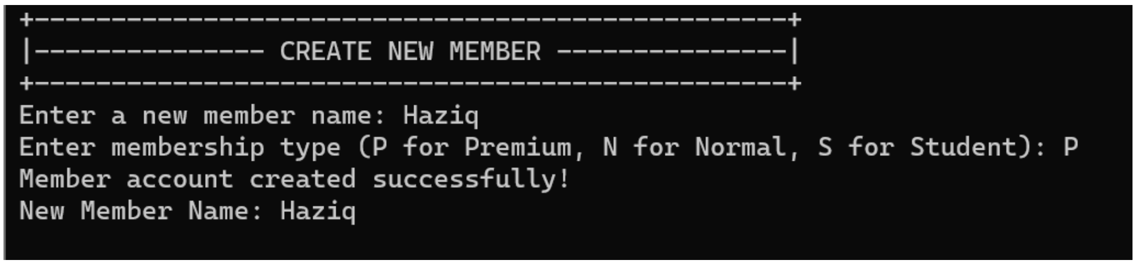
```
+-----------------------------------------+
|-------------- ADMIN LOGIN ---------------|
+-----------------------------------------+
Enter username: ImranHaziq
Enter password: abc123
Admin login successful!
```

Admin Login



```
+-----------------------------------------+
|-------------- ADMIN MENU ---------------|
+-----------------------------------------+


+----------------------------------+
| 1. Create Member Account         |
+----------------------------------+
| 2. Display Members Information |
+----------------------------------+
| 3. Update Member Information   |
+----------------------------------+
| 4. Delete Member                 |
+----------------------------------+
| 5. Search Member                 |
+----------------------------------+
| 6. Sort Members                  |
+----------------------------------+
| 7. Logout                        |
+----------------------------------+

 Enter your choice:
```

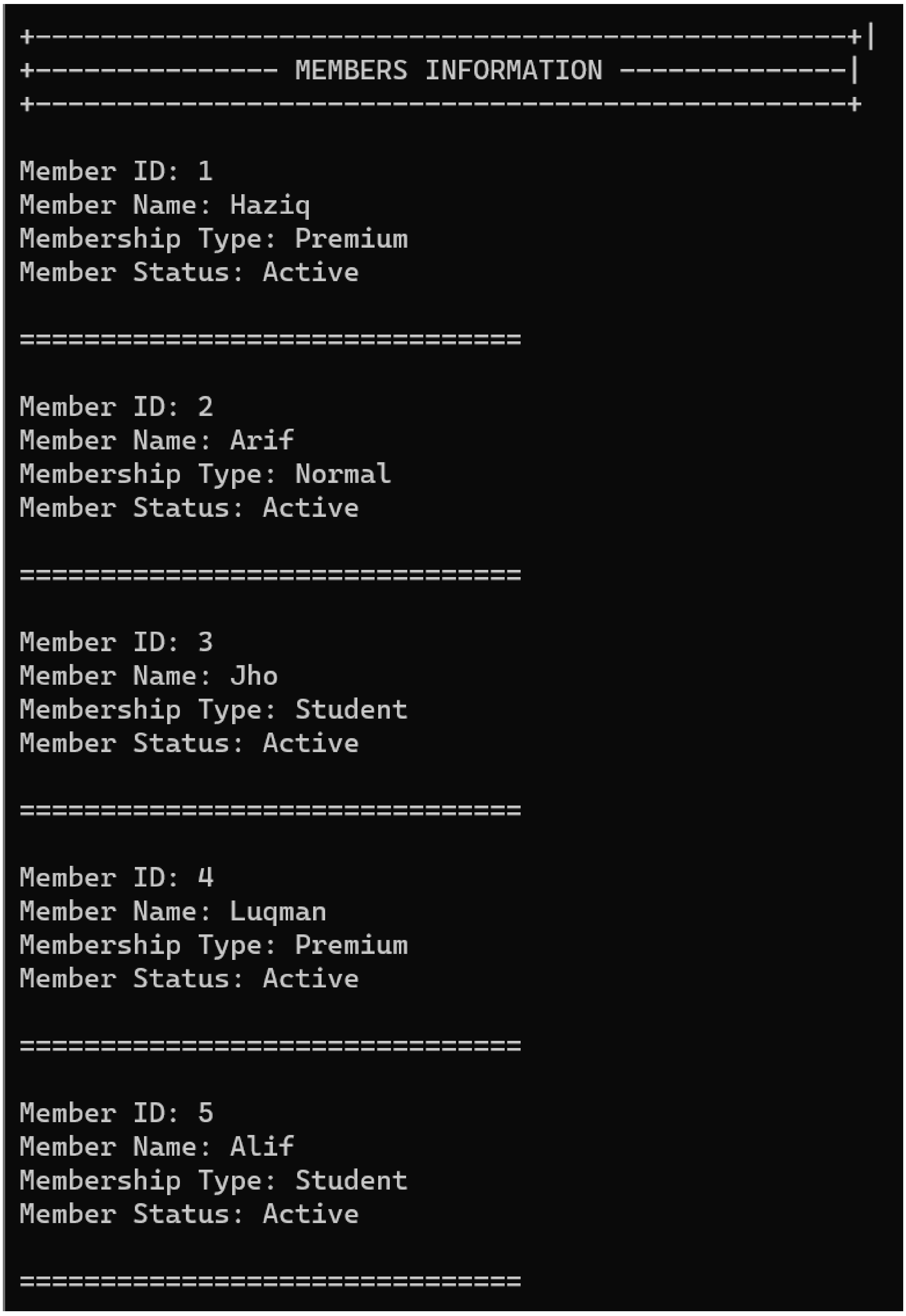
Admin Menu



```
+-----------------------------------------------+
|--------------- CREATE NEW MEMBER ---------------|
+-----------------------------------------------+
Enter a new member name: Haziq
Enter membership type (P for Premium, N for Normal, S for Student): P
Member account created successfully!
New Member Name: Haziq
```

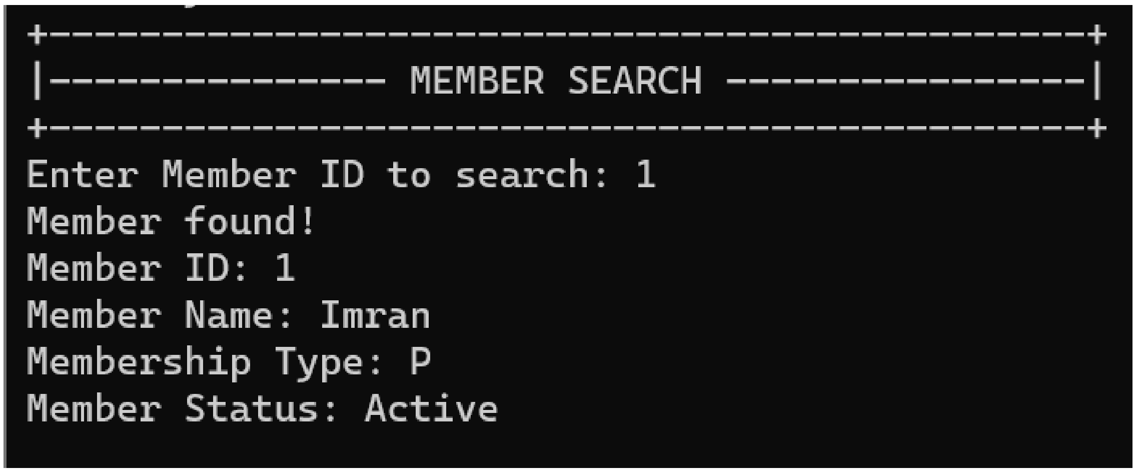
Create New Member

Display Member



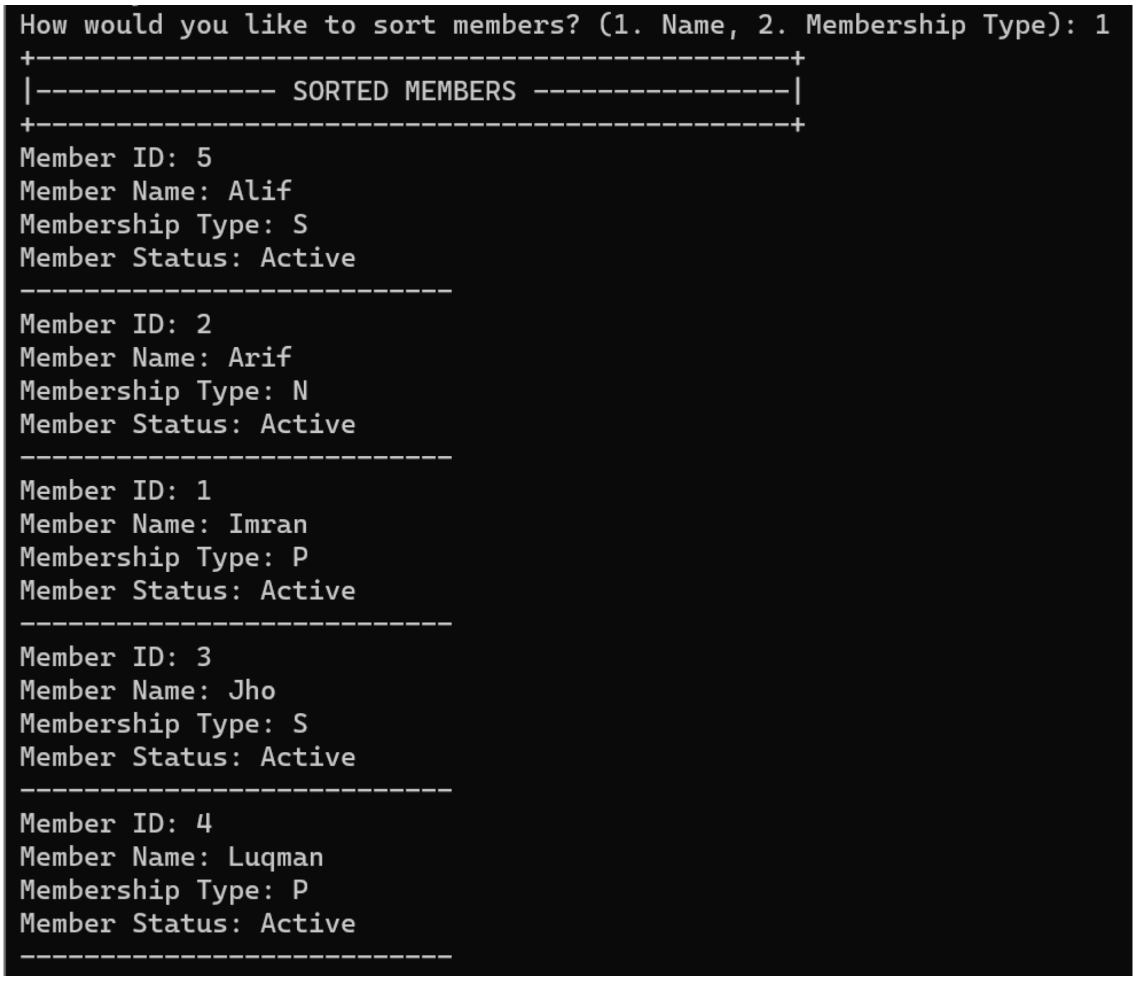
```
+-----------------------------------------------+|
+--------------- MEMBERS INFORMATION --------------|
+-----------------------------------------------+

Member ID: 1
Member Name: Haziq
Membership Type: Premium
Member Status: Active

===============================

Member ID: 2
Member Name: Arif
Membership Type: Normal
Member Status: Active

===============================

Member ID: 3
Member Name: Jho
Membership Type: Student
Member Status: Active

===============================

Member ID: 4
Member Name: Luqman
Membership Type: Premium
Member Status: Active

===============================

Member ID: 5
Member Name: Alif
Membership Type: Student
Member Status: Active

===============================
```

Display Member



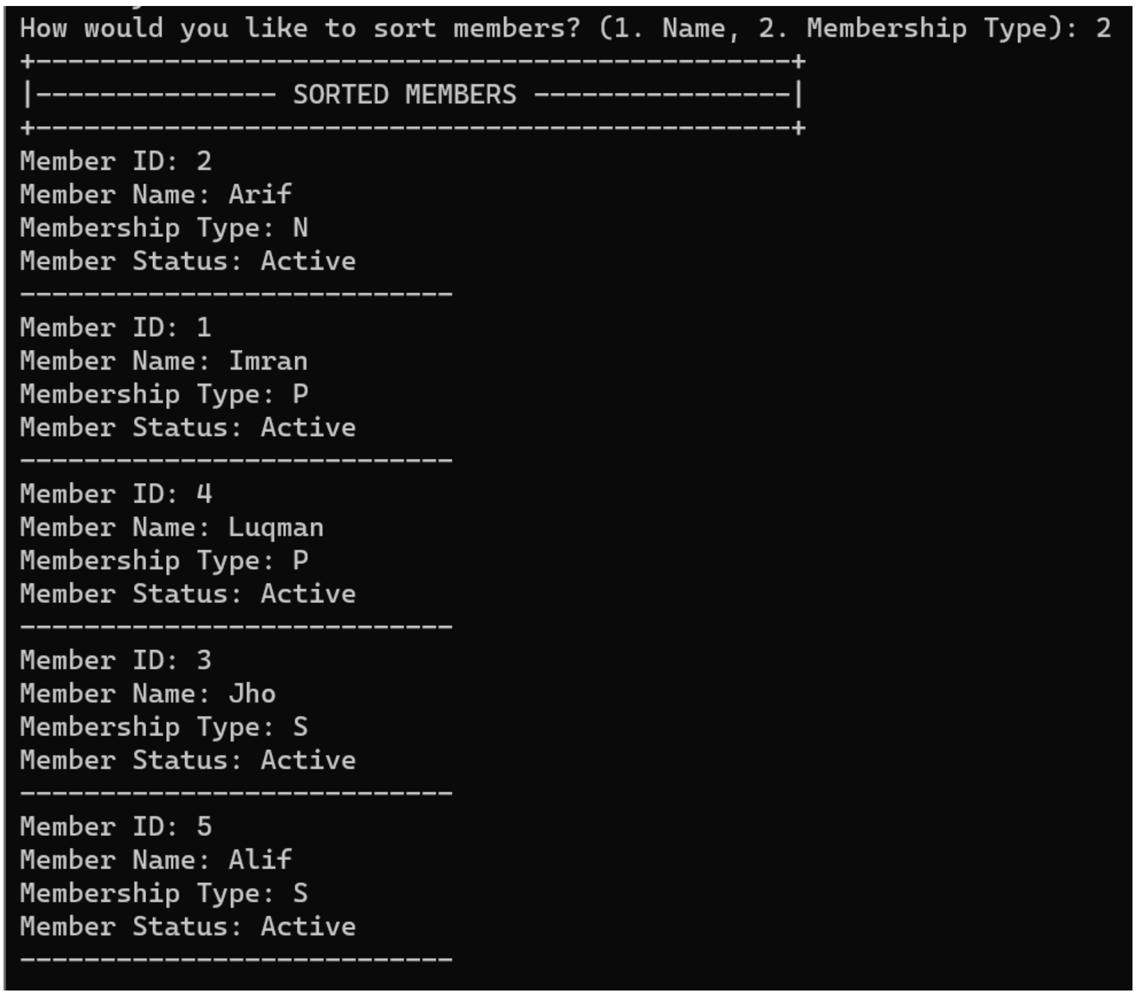
```
+-----------------------------------------------+
|--------------- MEMBER SEARCH ----------------|
+-----------------------------------------------+
Enter Member ID to search: 1
Member found!
Member ID: 1
Member Name: Imran
Membership Type: P
Member Status: Active
```

Member Search



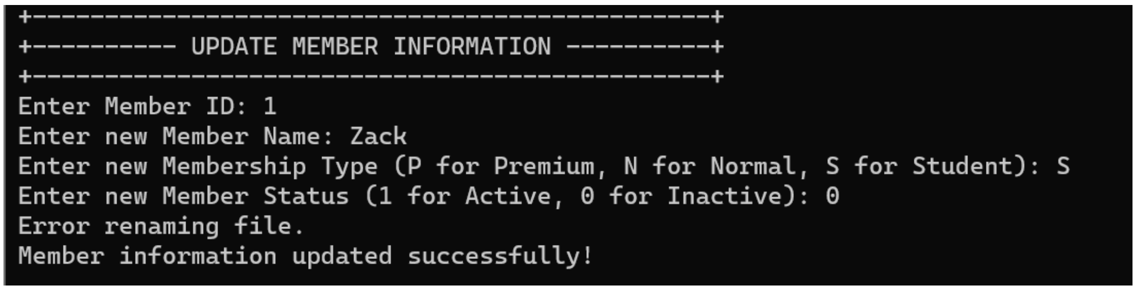
```
How would you like to sort members? (1. Name, 2. Membership Type): 1
+-----------------------------------------------+
|--------------- SORTED MEMBERS ----------------|
+-----------------------------------------------+
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 1
Member Name: Imran
Membership Type: P
Member Status: Active
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
```

Sorted Members by Name



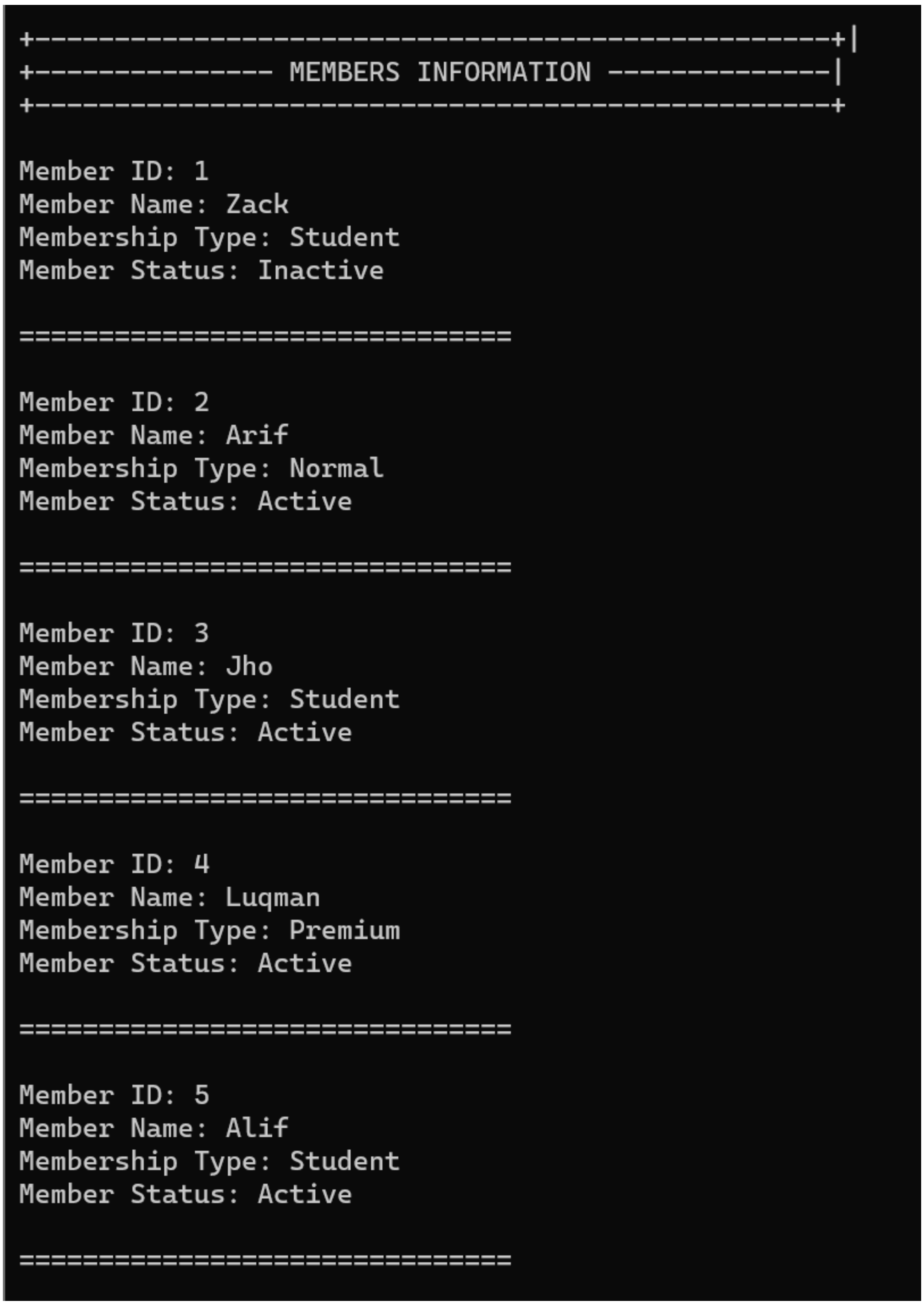
```
How would you like to sort members? (1. Name, 2. Membership Type): 2
+-----------------------------------------------+
|--------------- SORTED MEMBERS ----------------|
+-----------------------------------------------+
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 1
Member Name: Imran
Membership Type: P
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
```

Sorted Member by Membership Type



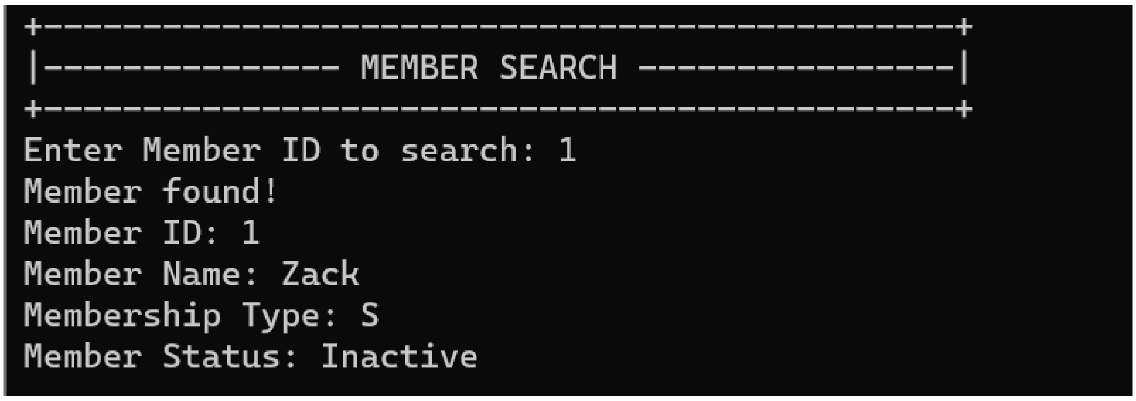
```
+-----------------------------------------------+
+---------- UPDATE MEMBER INFORMATION ----------+
+-----------------------------------------------+
Enter Member ID: 1
Enter new Member Name: Zack
Enter new Membership Type (P for Premium, N for Normal, S for Student): S
Enter new Member Status (1 for Active, 0 for Inactive): 0
Error renaming file.
Member information updated successfully!
```

Update Member Information (Previously Haziq)



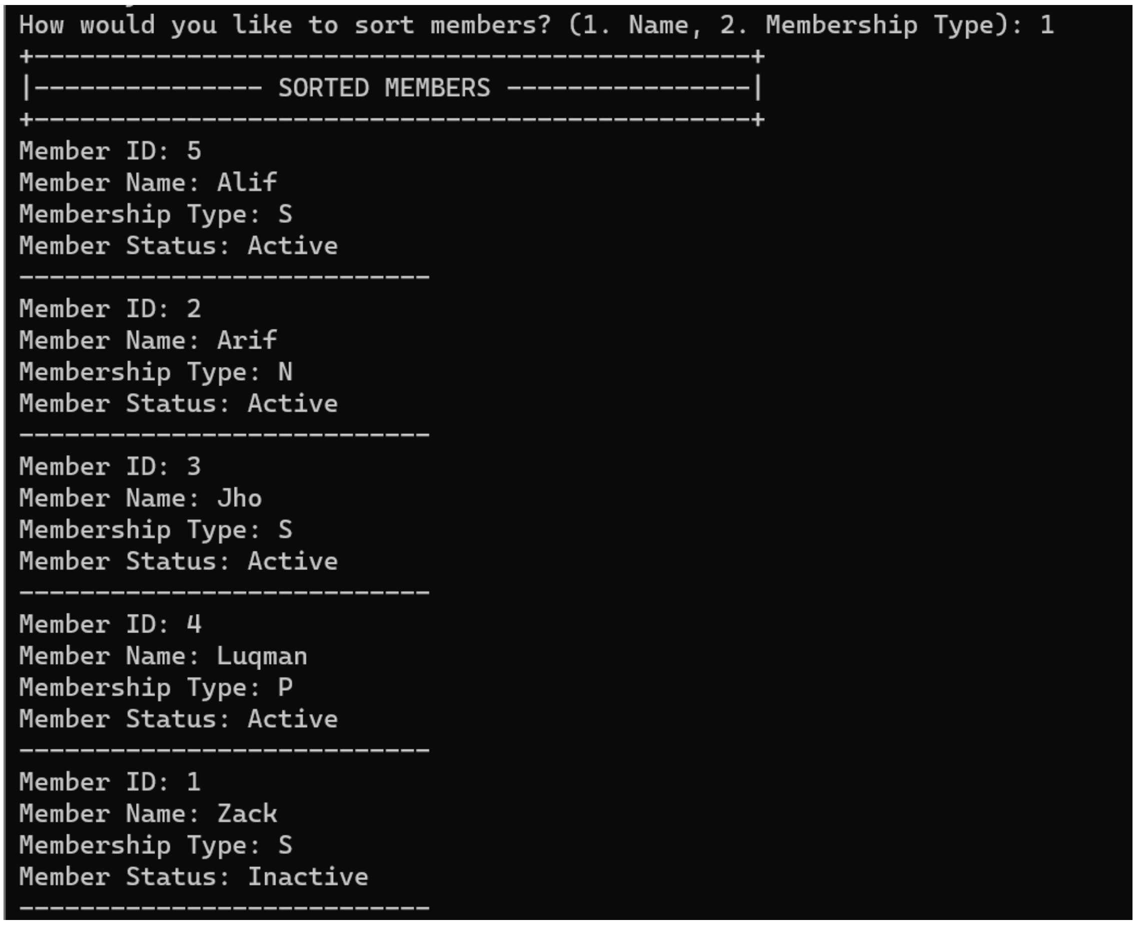
```
+--------------------------------------------+|
+-------------- MEMBERS INFORMATION -------------|
+--------------------------------------------+

Member ID: 1
Member Name: Zack
Membership Type: Student
Member Status: Inactive

===============================

Member ID: 2
Member Name: Arif
Membership Type: Normal
Member Status: Active

===============================

Member ID: 3
Member Name: Jho
Membership Type: Student
Member Status: Active

===============================

Member ID: 4
Member Name: Luqman
Membership Type: Premium
Member Status: Active

===============================

Member ID: 5
Member Name: Alif
Membership Type: Student
Member Status: Active

===============================
```

Display of Updated Member Information



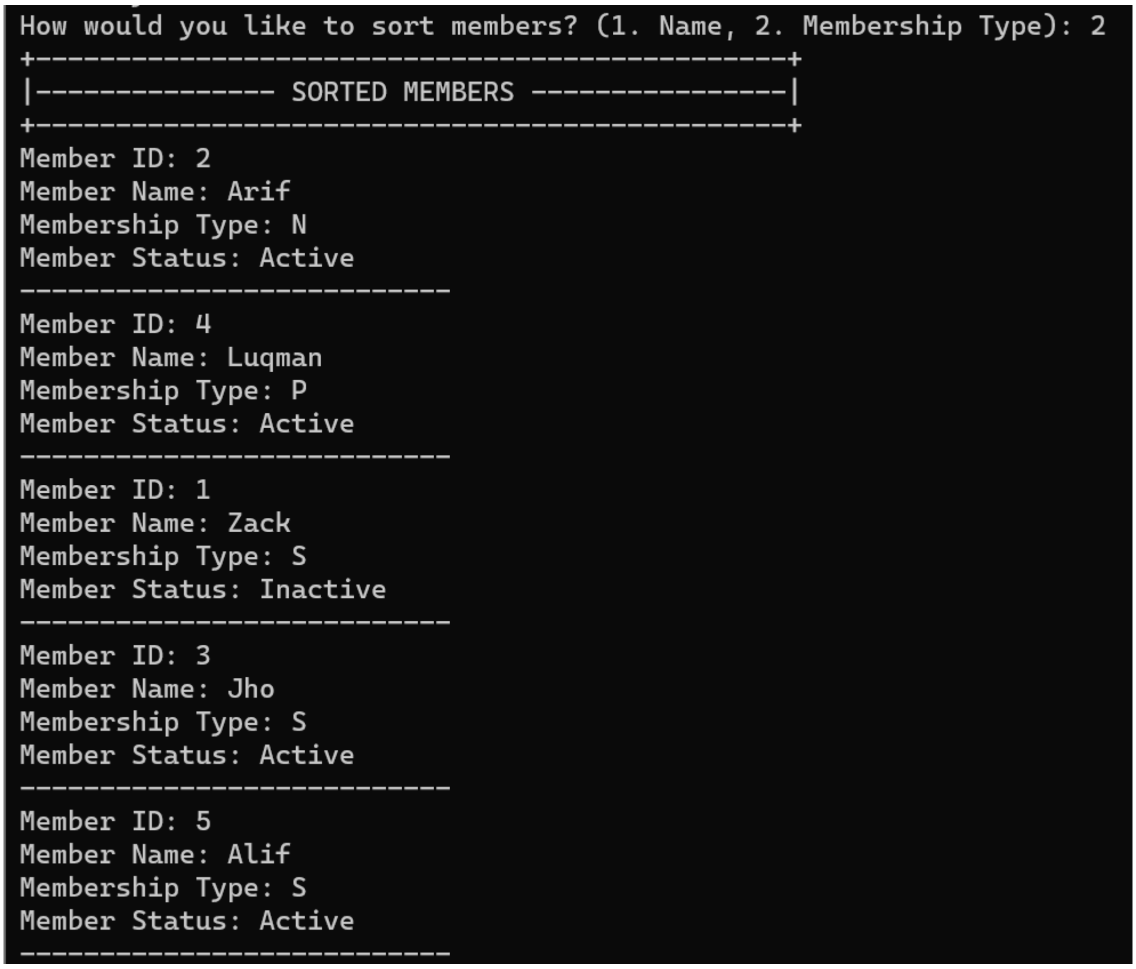
```
+----------------------------------------------+
|-------------- MEMBER SEARCH ---------------|
+----------------------------------------------+
Enter Member ID to search: 1
Member found!
Member ID: 1
Member Name: Zack
Membership Type: S
Member Status: Inactive
```

Search for Updated Member Information



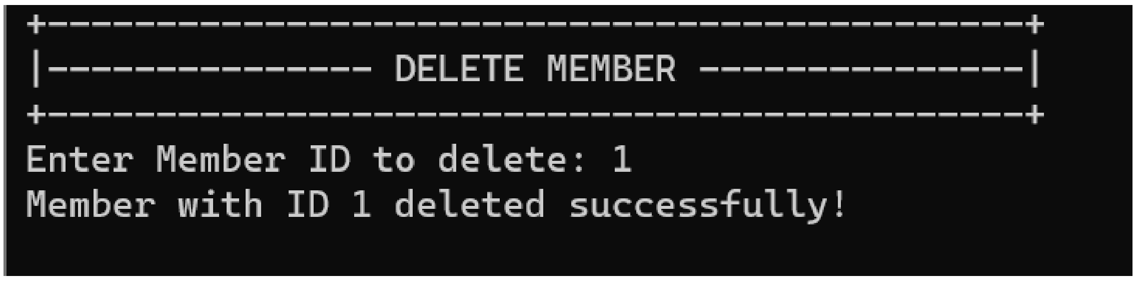
```
How would you like to sort members? (1. Name, 2. Membership Type): 1
+----------------------------------------------+
|-------------- SORTED MEMBERS ---------------|
+----------------------------------------------+
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
Member ID: 1
Member Name: Zack
Membership Type: S
Member Status: Inactive
---------------------------
```

Sorted Member by Name with Updated Information



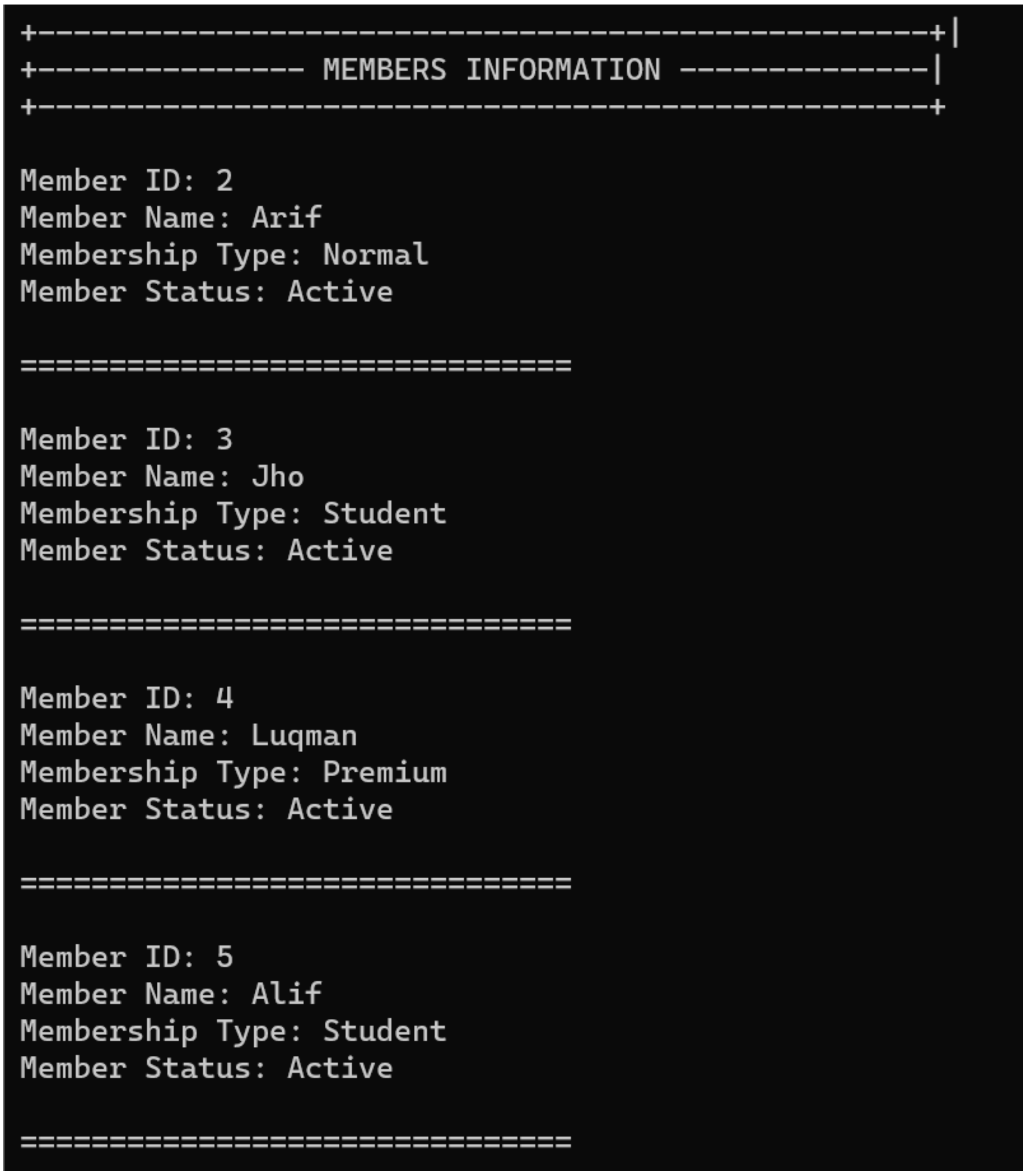
```
How would you like to sort members? (1. Name, 2. Membership Type): 2
+------------------------------------------------+
|--------------- SORTED MEMBERS ----------------|
+------------------------------------------------+
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
Member ID: 1
Member Name: Zack
Membership Type: S
Member Status: Inactive
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
```

Sorted Members by Membership Type with Updated Information



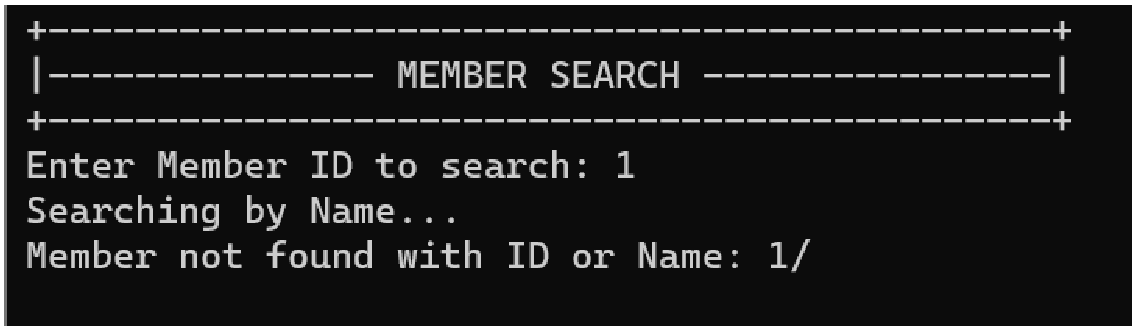
```
+----------------------------------------------+
|--------------- DELETE MEMBER ---------------|
+----------------------------------------------+
Enter Member ID to delete: 1
Member with ID 1 deleted successfully!
```

Delete Member (Zack with ID:1)



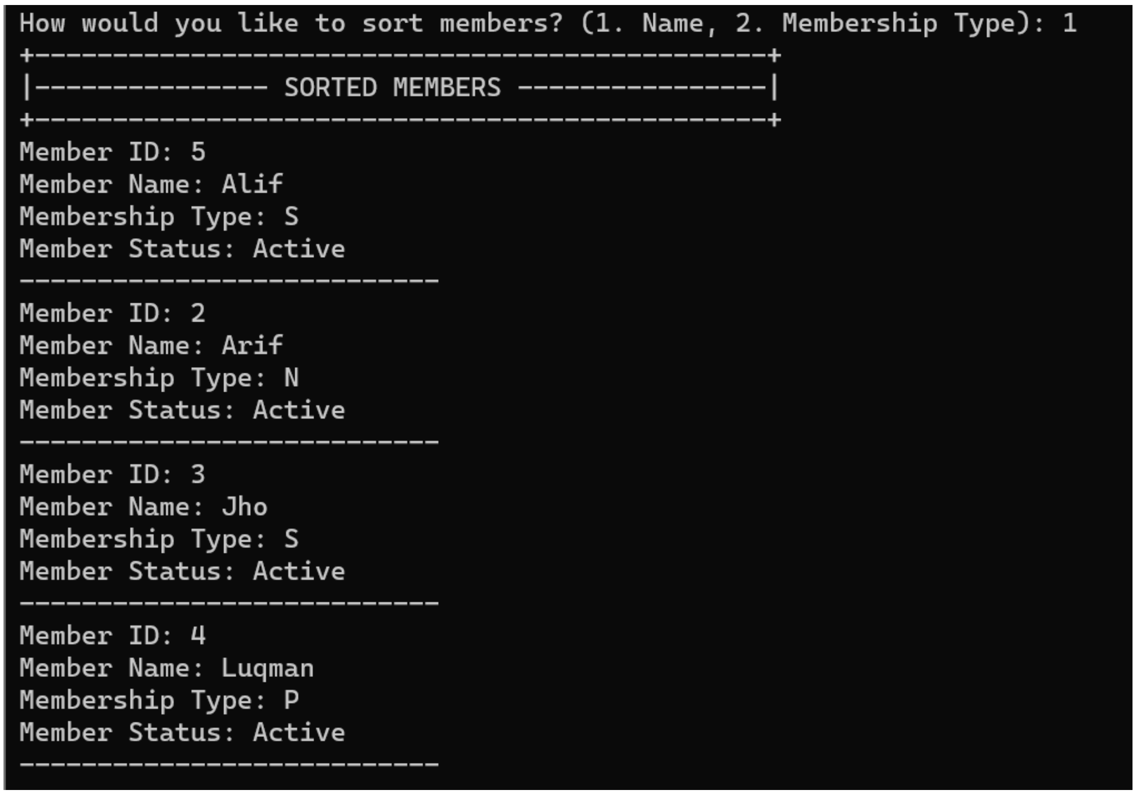
```
+-----------------------------------------------+|
+--------------- MEMBERS INFORMATION --------------|
+-----------------------------------------------+

Member ID: 2
Member Name: Arif
Membership Type: Normal
Member Status: Active

================================

Member ID: 3
Member Name: Jho
Membership Type: Student
Member Status: Active

================================

Member ID: 4
Member Name: Luqman
Membership Type: Premium
Member Status: Active

================================

Member ID: 5
Member Name: Alif
Membership Type: Student
Member Status: Active

================================
```

Display Member after Delete Member



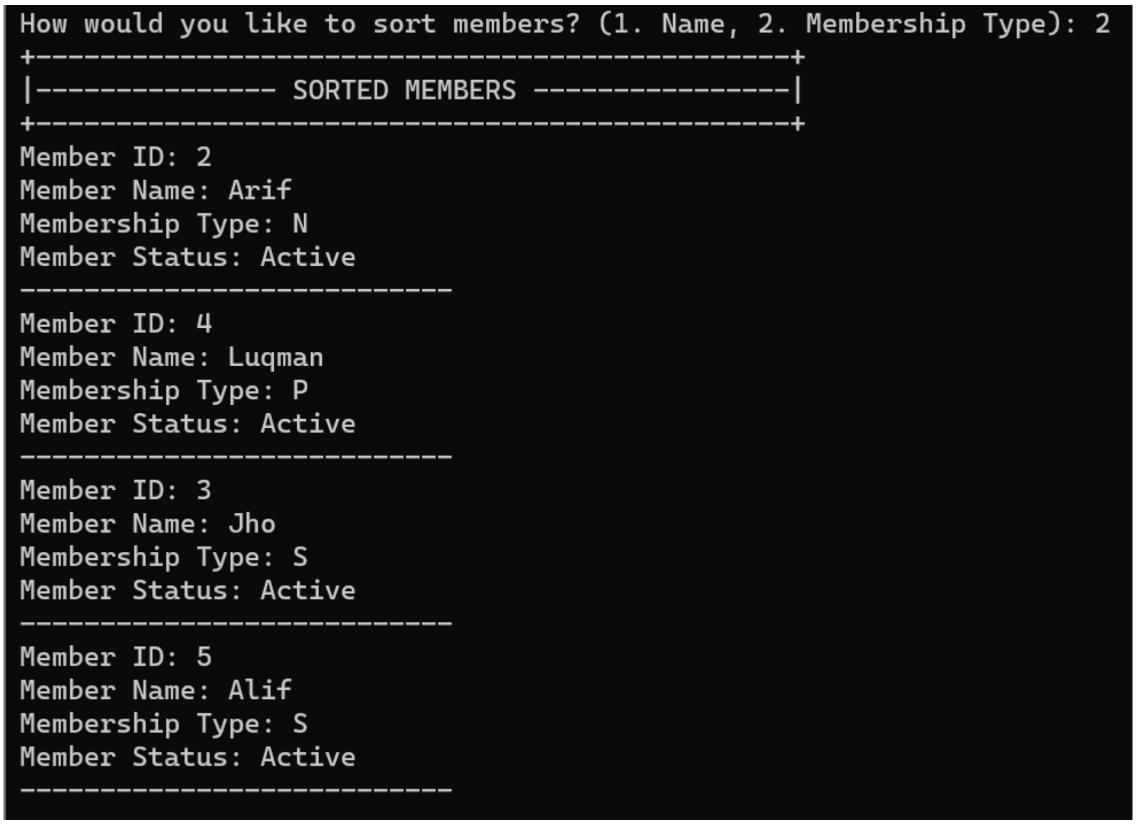
```
+-----------------------------------------------+
|--------------- MEMBER SEARCH ----------------|
+-----------------------------------------------+
Enter Member ID to search: 1
Searching by Name...
Member not found with ID or Name: 1/
```

Search Member after Delete Member



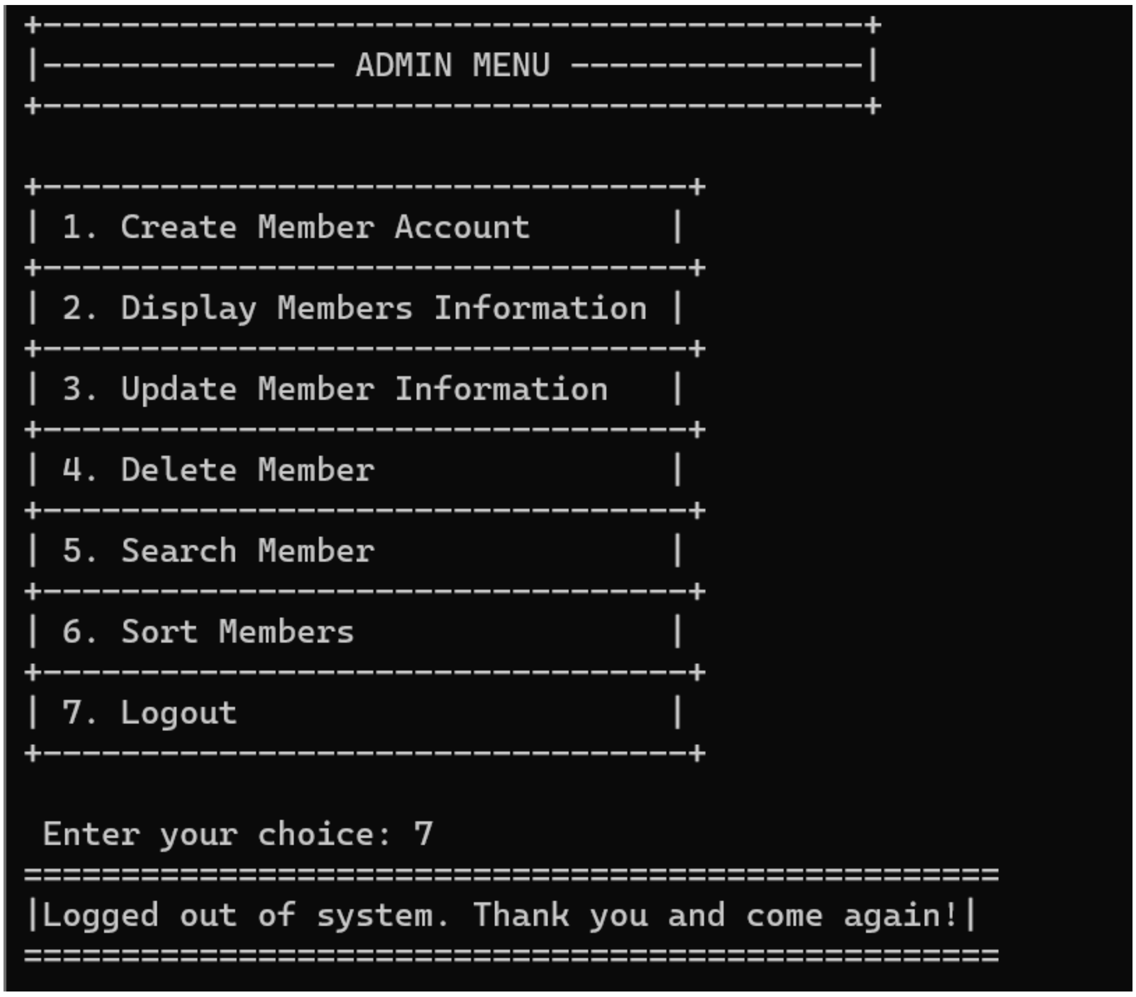
```
How would you like to sort members? (1. Name, 2. Membership Type): 1
+-----------------------------------------------+
|-------------- SORTED MEMBERS ----------------|
+-----------------------------------------------+
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
```

Sorted Member by Name after Delete Member



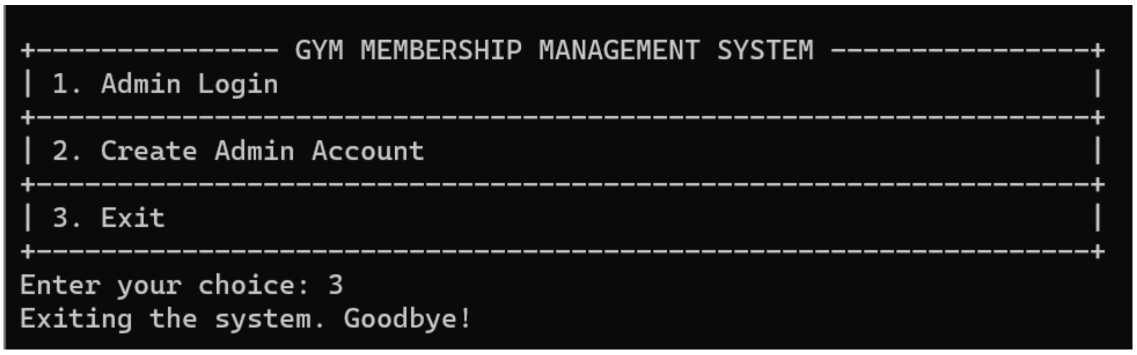
```
How would you like to sort members? (1. Name, 2. Membership Type): 2
+-----------------------------------------------+
|-------------- SORTED MEMBERS ----------------|
+-----------------------------------------------+
Member ID: 2
Member Name: Arif
Membership Type: N
Member Status: Active
---------------------------
Member ID: 4
Member Name: Luqman
Membership Type: P
Member Status: Active
---------------------------
Member ID: 3
Member Name: Jho
Membership Type: S
Member Status: Active
---------------------------
Member ID: 5
Member Name: Alif
Membership Type: S
Member Status: Active
---------------------------
```

Sorted Member by Membership Type after Delete Member



```
+-------------------------------------------+
|-------------- ADMIN MENU ---------------|
+-------------------------------------------+

+----------------------------------+
| 1. Create Member Account        |
+----------------------------------+
| 2. Display Members Information |
+----------------------------------+
| 3. Update Member Information   |
+----------------------------------+
| 4. Delete Member               |
+----------------------------------+
| 5. Search Member               |
+----------------------------------+
| 6. Sort Members                |
+----------------------------------+
| 7. Logout                      |
+----------------------------------+

 Enter your choice: 7
==================================================
|Logged out of system. Thank you and come again!|
==================================================
```

Logging out from the Admin Menu



```
+--------------- GYM MEMBERSHIP MANAGEMENT SYSTEM ----------------+
| 1. Admin Login                                                  |
+-----------------------------------------------------------------+
| 2. Create Admin Account                                         |
+-----------------------------------------------------------------+
| 3. Exit                                                         |
+-----------------------------------------------------------------+
Enter your choice: 3
Exiting the system. Goodbye!
```

Exit From the system

6.0 CONCLUSION

The Gym Membership Management System represents a robust and efficient solution for administering gym memberships, offering a suite of features for administrators and members alike. Through the implementation of core functionalities such as account creation, updating member information, and searching for members, the system ensures a seamless user experience.

The Gym Membership Management System exhibits a robust architecture, integrating advanced algorithms and data structures to ensure optimal performance and user satisfaction. In addition to the merge sort algorithm, the system leverages hash tables and linked lists to enhance efficiency and manage member data seamlessly.

Hash tables play a crucial role in facilitating rapid member data retrieval. Utilizing a hash function, the system efficiently maps member IDs to corresponding information, enabling constant-time access. This data structure excels in scenarios where quick and direct access to specific members is paramount, contributing to the overall responsiveness of the system.

Linked lists, on the other hand, provide a flexible and dynamic structure for organizing member information. The use of a linked list in the Gym Membership Management System allows for efficient insertion and deletion of members. The linked list's adaptability contributes to the system's ability to handle real-time updates with ease.

Additionally, the merge sort, known for its efficiency and stability, contributes significantly to optimizing the sorting process within the system. By leveraging merge sort, the Gym Membership Management System can efficiently organize and present member data, providing administrators with a streamlined view of the membership base.

Looking ahead, the system holds potential for further improvements and expansions. Future iterations could explore additional sorting algorithms or introduce more sophisticated data structures to accommodate evolving requirements. Additionally, considerations for enhanced security features, reporting functionalities, and scalability will be instrumental in adapting the system to the dynamic needs of a growing gym.