Btech ICT Section-A

CAPSTONE ASSIGNMENT P6 :

"DAIICT CLUB MANAGER"

"BY noCapCoderz"

Members	Student ID
Ramoliya Shivam	202301049
Pranav Mandani	202301101
Ved Mungra	202301026
Pratham Lakhani	202301009

Faculty Mentor : Sourish Dasgupta

GitHub Repository Link :

"https://github.com/Shivam-Ramoliya/noCapCoderz"

Task :

➔ You need to build a manager for all the DA-IICT clubs. The manager ensures that a club member can be looked up in minimum time. A member can either be a faculty or a student. One should be able to search by name, ID, specific club name, or club category (i.e., arts, science & technology, sports, culture). Note that the user of this manager may not be a DA-IICT-ian and, therefore, may not know the clubs' names.

Our Approach :

➔ We have to Make the Club Management System for Particular Institute say DAIICT. In our System We Added Functionalities Like :

- Add New Member
- Delete Member
- View all Member List
- Search Member
    - ➢ Search By ID
    - ➢ Search By Name
    - ➢ Search By Club
- Add New Club
- Delete Club
- View Club List

➔ Here Main Part of Programme is that use of appropriate Data Structure. We have many options like link list with Enum, map, hash table etc. but we feel that the use of Hash Table is Appropriate to store data and also for search. It is More accurate than the map.

➔ Map sorts its elements implicitly in lexicographical order which increases the time complexity of basic operations such as Addition, Deletion of Members and Searching etc. also lexicographical sorting was not needed in our project.

➔ But we also Face some Difficulties like use of HashMap make search function by ID in O(1) and Search By Name and Club in O(n) and try keeping Space Complexity Low.

Algorithm :

➔ **Initialization:**

- #Include necessary header files (iostream, fstream, sstream, string, iomanip, vector, unordered_map, algorithm, ctime, conio.h).
- Define a Member structure to hold member information (ID, name, post, club, phone number).
- Declare function prototypes for various functionalities.
- Define global constants for file names (FILENAME, CLUBS_FILE).

➔ **Main Menu Display (main_menu):**

- Display the main menu options (add member, delete member, search member, view members, add club, delete club, view clubs, close application).
- Based on user input, call corresponding functions.
- Functionality Implementations:

➔ **Add New Member (add_new_member):**

- Prompt for password.
- Validate password.
- Prompt user to enter member details (ID, name, post, club, phone number).
- Check if the member already exists using the ID.
- Write member details to the record file (record.csv).
- Prompt to add more members or return to the main menu.

➔ **Delete Member (delete_member):**

- Prompt for password.
- Validate password.
- Prompt user to enter the ID of the member to be deleted.
- Read member records from the file (record.csv).
- Write all members except the one to be deleted to a temporary file.
- Replace the original file with the temporary file.
- Prompt success/failure message and return to the main menu.

➔ **Search Member (search_member):**

- Prompt user to choose search criteria (ID, name, club).
- Based on the choice, search for the member in the record file (record.csv).
- Display member details if found.
- Prompt to return to the search member section or return to the main menu.

➔ **View Members (view_members):**

- Read member records from the file (record.csv).
- Sort members by club name.
- Display member details sorted by club.
- Prompt to return to the main menu.

➔ **Add New Club (add_new_club):**

- Prompt for password.
- Validate password.
- Prompt user to enter club details (name, category).
- Write club details to the clubs file (clubs.csv).
- Prompt success/failure message and return to the main menu.

➔ **View Clubs (view_clubs):**

- Read club records from the file (clubs.csv).
- Display club details.
- Prompt to return to the main menu.

➔ **Delete Club (delete_club):**

- Prompt for password.
- Validate password.
- Prompt user to select the club to delete.
- Remove the selected club from the clubs file (clubs.csv).
- Prompt success/failure message and return to the main menu.

➔ **Password Verification (password):**

- Prompt user to enter a password.
- Compare the entered password with the predefined password.
- Return 1 for correct password, 0 for incorrect password.

➔ **Loading Page (Loading):**

- Display a welcome message with the names of the members.
- Display a loading animation.

- Clear the screen and display the main menu.

➔ **Main Function (main):**

- Call the Loading function to start the program.

Pseudocode :

1. Include necessary libraries:

- iostream
- fstream
- sstream
- string.h
- iomanip
- vector
- unordered_map
- algorithm
- ctime
- conio.h
- limits

2. Define a structure Member to hold member information:

- ID (integer)
- name (string)
- post (string)

- club (string)
- Phone_no (long long integer)

3. Declare function prototypes:

- main_menu()
- add_new_member()
- delete_member()
- search_member()
- view_members()
- return_to_main_menu()
- Loading()
- add_new_club()
- view_clubs()
- delete_club()
- password()
- is_member_exists()
- sort_clubs()
- select_category()
- select_club()

4. Define global constants:

- FILENAME = "record.csv"
- CLUBS_FILE = "clubs.csv"

5. Implement the main_menu() function:

- Display menu options using cout statements

- Take user input for choice using getch()

- Based on the choice, call corresponding functions or display appropriate messages

6. Implement the add_new_member() function:

- Prompt user for password using password() function

- If password matches:

  - Create a new Member object

  - Open FILENAME in append mode

  - Prompt user for member details (ID, name, post, club, Phone_no)

  - Check if member with given ID already exists using is_member_exists() function

  - If not, write member details to FILENAME

  - Close the file

  - Ask user if they want to add more members

  - If yes, recursively call add_new_member()

  - If no, return to main menu using return_to_main_menu()

- If password does not match, display error message and retry password entry

7. Implement the delete_member() function:

- Prompt user for password using password() function

- If password matches:

  - Open FILENAME in read mode

  - Create a temporary file

  - Prompt user for member ID to delete

  - Read each line from FILENAME

  - If member ID matches, do not write to temporary file

- Otherwise, write to temporary file
- Close both files
- Delete FILENAME
- Rename temporary file to FILENAME
- Display appropriate message
- Return to main menu using return_to_main_menu()

- If password does not match, display error message and retry password entry

8. Implement the search_member() function:

- Prompt user for search criteria (ID, name, club) using getch()
- Based on the choice:
  - Search by ID:
    - Prompt user for member ID
    - If found, display member details
    - Otherwise, display appropriate message
  - Search by name:
    - Prompt user for member name
    - If found, display member details
    - Otherwise, display appropriate message
  - Search by club:
    - Prompt user for club name
    - If found, display members belonging to that club
    - Otherwise, display appropriate message
  - Return to main menu using return_to_main_menu()

9. Implement the view_members() function:

- Open FILENAME in read mode
- Read member details from FILENAME
- Store members in a vector
- Sort members by club using sort_clubs() function
- Display members sorted by club
- Return to main menu using return_to_main_menu()

10. Implement the return_to_main_menu() function:

- Prompt user to press ENTER to return to main menu
- Wait for user input using getch()
- Call main_menu() function

11. Implement the Loading() function:

- Display loading screen and welcome message
- Call main_menu() function

12. Implement the add_new_club() function:

- Prompt user for password using password() function
- If password matches:
  - Prompt user for club category using select_category() function
  - Prompt user for club name
  - Open CLUBS_FILE in append mode
  - Write club name and category to CLUBS_FILE
  - Close the file
  - Display success message
  - Return to main menu using return_to_main_menu()

- If password does not match, display error message and retry password entry

13. Implement the view_clubs() function:

- Open CLUBS_FILE in read mode
- Read club details from CLUBS_FILE
- Display club details
- Return to main menu using return_to_main_menu()

14. Implement the delete_club() function:

- Prompt user for password using password() function
- If password matches:
  - Open CLUBS_FILE in read mode
  - Create a temporary file
  - Prompt user for club ID to delete
  - Read each line from CLUBS_FILE
  - If club ID matches, do not write to temporary file
  - Otherwise, write to temporary file
  - Close both files
  - Delete CLUBS_FILE
  - Rename temporary file to CLUBS_FILE
  - Display appropriate message
  - Return to main menu using return_to_main_menu()
- If password does not match, display error message and retry password entry

15. Implement the password() function:

- Prompt user for password

- Read characters without echoing to the screen
- Compare entered password with predefined password
- Return 1 if passwords match, 0 otherwise

16. Implement the is_member_exists() function:

- Open FILENAME in read mode
- Read each line from FILENAME
- Extract member ID from each line
- If ID matches given ID, return true
- Otherwise, return false

17. Implement the sort_clubs() function:

- Comparison function to sort members by club name

18. Implement the select_category() function:

- Display club categories
- Prompt user to select a category
- Return selected category

19. Implement the select_club() function:

- Open CLUBS_FILE in read mode
- Read club names from CLUBS_FILE
- Display club names and prompt user to select a club
- Return selected club

20. Implement the main function:

- Call the Loading() function to start the program

21. End of program.

Time Complexity :

➔ The time complexity of searching for a member using each of the three methods is as Follow :

➔ Searching by Member ID (Using unordered_map::find):

- Time Complexity: O(1) on average, O(n) worst-case scenario.
- Explanation: The unordered_map or say hash table. When you use memberMap.find(s) to search for a member by ID, it calculates the hash of s and directly looks up the corresponding bucket in constant time (O(1) on average). However, in the worst-case scenario where all keys have the same hash, and they all collide, the time complexity can become O(n), where n is the number of elements in the map. This is unlikely to occur with a good hash function and a reasonable load factor.
- We can avoid it By Handling the collision either by

➔ Searching by Member Name (Iterating Over the unordered_map):

- Time Complexity: O(n).
- Explanation: In this method, you iterate over all elements in the memberMap and compare the name of each member with the input name (s). Since it involves iterating through all elements, the time complexity is O(n), where n is the number of elements in the map.

➔ Searching by Club Name (Iterating Over the unordered_map):

- Time Complexity: O(n).
- Explanation: Similar to searching by member name, this method involves iterating over all elements in the memberMap and comparing the club name of each member with the input club name (club). Hence, the time complexity is O(n), where n is the number of elements in the map.

➔ In summary:

- Searching by Member ID using unordered_map::find is the most efficient with an average time complexity of O(1) due to the direct lookup in the hash table.
- Searching by Member Name or Club Name by iterating over the unordered_map has a time complexity of O(n), making it less efficient compared to searching by ID, especially when the number of elements in the map is large.

Space Complexity :

➔ Space complexity refers to the amount of memory space required by the program to execute relative to the input size. Here's a breakdown of the space complexity for each component of the program:

➔ **Global Constants**: T

- he global constants FILENAME and CLUBS_FILE are strings, and they occupy constant space regardless of the input size. So, the space complexity for global constants is O(1).

➔ **Structures and Variables:**

- The Member structure consists of integer and string variables. Each instance of the Member structure requires a constant amount of memory for its variables. So, the space complexity for structures and variables is O(1) per member.
- Other variables and function prototypes also consume a constant amount of memory. So, their space complexity is also O(1).

➔ **File Input/Output:**

- The space complexity for file input/output operations depends on the size of the files being read or written. In this program, the file size is not directly related to the input size but rather to the number of records stored in the files.
- For the record.csv file, each record consists of an integer ID, two strings, and a long long integer. The space complexity for storing each record is O(1) per record.
- Similarly, for the clubs.csv file, each record consists of a club name and a category. The space complexity for storing each club record is also O(1) per record.

➔ **Vectors and Data Structures:**

- The program uses a vector to store member records when viewing members. The space complexity for the vector is O(n), where n is the number of members.

- The unordered_map data structure is used to store member information temporarily when searching for a member. The space complexity for the unordered_map is O(n), where n is the number of members being stored temporarily.

➔ **Function Calls and Recursion:**

- The program uses function calls and recursion, but they do not consume additional memory proportional to the input size. So, the space complexity for function calls and recursion is O(1).

➔ **Considering these factors, the overall space complexity of the program can be summarized as follows:**

- Space Complexity: O(n) for storing member records in the vector and the unordered_map, where n is the number of members.
- Otherwise, the program mainly consumes constant space (O(1)).

➔ Keep in mind that the space complexity analysis provided here is a high-level estimation and may vary depending on specific implementation details and compiler optimizations.

GitHub Repository Link :

"https://github.com/Shivam-Ramoliya/noCapCoderz"

Video of Sample Code Run :

"https://drive.google.com/file/d/1HecqoQc9bYr55zptQZJzdwqstmw1xkp5/view?usp=drive_link"

Reference :

"https://cplusplus.com/reference/"

THANK YOU ☺

From noCapCoderz…

==========