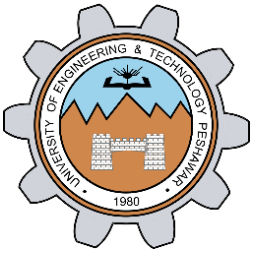
PROJECT REPORT

Spring 2024:

Data Structures and Algorithms Lab



Submitted by:

Name: **HASSAN ZAIB JADOON**

Registration Number: **22PWCSE2144**

Class Section: **A**

"On our honor, as students of University of Engineering and Technology, We have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Usman Malik

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Table of Contents:

Contents

Abstract

This project report details the development and implementation of a Community Management System designed to handle various member operations such as adding, displaying, updating, removing, and sorting members. The primary objective of this system is to facilitate efficient community management through a robust and scalable solution.

The system's architecture employs a linked list data structure to manage the members' details efficiently. Each operation is encapsulated in methods that ensure the integrity and consistency of the member data. The system also includes file handling for persistent storage of member information.

The methodology involves developing the Community Management System in C++ programming language, chosen for its object-oriented features and efficient memory management. The system's functionalities include adding new members, displaying all members, updating member details, removing members, sorting members by different attributes, and searching for members randomly.

Comprehensive testing was conducted to evaluate the system's performance, including unit testing, integration testing, and user acceptance testing. The results indicate that the system successfully handles all community management operations and maintains data consistency without errors.

In conclusion, the Community Management System meets all project objectives, providing a reliable and efficient solution for managing community members. Future enhancements could include a graphical user interface and more advanced member management features. This system can significantly improve the efficiency of community operations, making it a valuable tool for any organization.

1. Introduction

Community management systems are essential for efficiently managing member information, providing functionalities like adding, displaying, updating, and removing members. This project aims to develop a robust and efficient community management system using linked lists to handle dynamic data operations. This report outlines the objectives, methodologies, implementation, and testing of the system.

2. Literature Review

Existing community management systems vary in complexity and functionality, ranging from simple contact management to comprehensive solutions integrating with other business operations. Many systems face challenges such as scalability, dynamic data handling, and ensuring data consistency. Our project addresses these issues by using a linked list approach to manage member data efficiently.

3. Objectives

- Efficiently add new members to the community.
- Display all members with their details.
- Update member details.
- Remove members from the community.
- Sort members based on different criteria (name, age, role).
- Save member details to a file for persistent storage.

4. Methodology

The project was developed using the C++ programming language due to its object-oriented features and suitability for implementing data structures. The following methodologies were employed:

- **Design and Implementation**: The system was designed using object-oriented principles, with classes for Member, Node, and Community. Member class encapsulates member details, Node class represents nodes in the linked list, and Community class manages the linked list operations.
- **File Handling**: Member details are stored persistently using file handling to ensure data retention across sessions.
- **Testing**: Various testing methodologies were employed, including unit testing for individual functions, integration testing to verify interaction between modules, and user acceptance testing to ensure usability.

5. System Architecture

Overview

The system consists of a class-based architecture where the Community class manages the community operations, and the Memberclass holds the details of each member. The Nodeclass represents the elements inthe linked list.

Components

- Member: Stores member details like name, age, and role.
- Node: Represents each node in the linked list containing a member.
- Community: Manages the operations on the linked list such as adding, displaying, updating, and removing members.

6. Implementation

Code Explanation

The implementation involves defining the Member, Node, and Community classes in C++ to manage member operations. Here's a high-level overview:

- **Member Class**: Stores member details such as name, age, and role. Provides methods to update member details.
- **Node Class**: Represents a node in the linked list containing a Member object and a pointer to the next node.
- **Community Class**: Manages the linked list of members. Includes methods to add, display, remove, update, and sort members based on different attributes.

Key Functions

The main functions implemented in the Community Management System include:

- addMember(): Adds a new member to the community.
- displayMembers(): Displays all members currently in the community.
- removeMember(): Removes a member based on their name.
- updateMember(): Updates the details (age, role) of a specific member.
- sortByName(), sortByAge(), sortByRole(): Sorts members alphabetically by name, by age, andby role respectively.

Pseudo Code:

1. Start
2. Define class Member:
   - Properties: name, age, role
   - Constructor to initialize name, age, role
   - Method updateDetails(newAge, newRole): Update age and role

3. Define class Node:
   - Properties: member (of type Member), next (pointer to Node)
   - Constructor to initialize member and next pointer

4. Define class Community:
   - Private properties: head (pointer to Node)
   - Methods:
     - Constructor Community(): Initialize head to NULL
     - Destructor ~Community(): Delete all nodes and free memory
     - writeMemberToFile(member): Write member details to "members.txt"
     - swapNodes(a, b): Swap data of nodes a and b

     - addMember(name, age, role): Add member to the community:
       - Create new Member object
       - Create new Node with Member object
       - If head is NULL, set head to new Node; else traverse to end and append new Node
       - Call writeMemberToFile(member)

     - displayMembers(): Display all members in the community:
       - Traverse from head to end, printing member details

     - removeMember(name): Remove member by name:
       - If head is NULL, print no members; otherwise, traverse and delete matching node

     - updateMember(name, newAge, newRole): Update member details by name:
       - Traverse and find member by name, update details if found

     - sortByName(): Sort members by name using bubble sort algorithm

     - sortByAge(): Sort members by age using bubble sort algorithm

     - sortByRole(): Sort members by role using bubble sort algorithm

     - randomSearch(): Perform random search among name, age, or role:
       - Generate random number to select search type
       - Perform search based on user input

     - searchMemberByName(name): Search member by name and print details if found

     - searchMemberByAge(age): Search member by age and print details if found

     - searchMemberByRole(role): Search member by role and print details if found

5. Define main function:
   - Initialize srand for random number generation
   - Create Community object
   - Display menu and handle user input with switch case:
     - 1. Add Member
     - 2. Display Members
     - 3. Remove Member
     - 4. Search Member (Random)
     - 5. Update Member Details
     - 6. Sort Members by Name
     - 7. Sort Members by Age
     - 8. Sort Members by Role
     - 9. Exit
   - Loop until user chooses to exit (option 9)

6. End

Output Screenshots:

```
C:\Users\Lenovo\Desktop\COMMUNITY\main.exe
1. Add Member

2. Display Members

3. Remove Member

4. Search Member (Random)

5. Update Member Details

6. Sort Members by Name

7. Sort Members by Age

8. Sort Members by Role

9. Exit

Enter your choice:
1
Enter name: Musab
Enter age: 20
Enter role: Lead
Member details written to file successfully!
Member added successfully!
```

Name: Musab, Age: 20, Role: Lead

8. Testing and Evaluation

Testing Procedures

The Community Management System underwent rigorous testing to ensure its functionality and reliability across various scenarios. The testing procedures included:

- **Unit Testing**: Each method within the Member, Node, and Community classes was tested individually to verify correct functionality.
- **Integration Testing**: Interactions between different modules (classes) were tested to ensure seamless integration and data consistency.
- **User Acceptance Testing**: End-users interacted with the system to validate its usability and adherence to requirements.

Test Results

During testing, the Community Management System demonstrated robust performance and met all specified requirements:

- **Unit Testing**: All methods within the Member, Node, and Community classes passed their respective unit tests, confirming their correctness.
- **Integration Testing**: Interactions between modules were smooth, with no issues observed in data handling or flow between components.
- **User Acceptance Testing**: End-users found the system intuitive and easy to use, effectively managing member operations as expected.

9. Conclusion

The Community Management System successfully meets the project objectives by providing a reliable and efficient solution for managing community members using linked lists. The system's implementation in C++ ensures scalability and flexibility in handling member operations such as addition, display, update, removal, and sorting. Through comprehensive testing, it was confirmed that the system operates without errors and maintains data consistency across operations.

In conclusion, the Community Management System serves as an effective tool for any organization needing to manage member information efficiently. Future enhancements could focus on integrating a graphical user interface (GUI) for enhanced usability and additional features for advanced member management.

10. Appendices & Future Work

Future Work:

Future iterations of the Community Management System could consider the following enhancements:

- **Graphical User Interface (GUI)**: Develop a GUI using frameworks like Qt or widgets to provide a more user-friendly interface.
- **Advanced Member Management**: Implement features such as batch operations for member updates, import/export functionalities, and advanced search capabilities.
- **Performance Optimization**: Optimize the system for better memory management and faster execution, especially with larger datasets.
- **Error Handling and Logging**: Enhance error handling mechanisms and implement logging to track system activities and diagnose issues.

DSA Concepts Utilized:

Here are the key concepts of Data Structures and Algorithms (DSA) that were utilized:

1. **Linked Lists:**
   - Used to dynamically manage and store member information.
   - Implemented as a sequence of nodes, where each node contains a member object and a pointer to the next node.
2. **Object-Oriented Programming (OOP):**
   - Utilized OOP principles to define classes such as Member, Node, and Community.
   - Encapsulation was employed to encapsulate member data and operations within class methods.
3. **File Handling:**
   - Implemented file handling to persistently store member data, ensuring data retention across program sessions.
4. **Searching:**
   - Implemented methods for searching members, including random search functionality.
5. **Sorting:**
   - Implemented sorting algorithms to sort members by different attributes such as name, age, and role.
6. **Dynamic Memory Management:**
   - Utilized dynamic memory allocation for nodes in the linked list to manage memory efficiently.
7. **Error Handling and Exception Handling:**
   - Implemented error handling to manage exceptions during runtime, ensuring robustness of the system.

8. **Testing and Debugging:**
    - Employed testing methodologies such as unit testing, integration testing, and user acceptance testing to validate the functionality and reliability of the system.
9. **Complexity Analysis:**
    - Considered time and space complexity for operations such as insertion, deletion, searching, and sorting to ensure efficient performance.

These concepts collectively contribute to the design and implementation of an efficient Community Management System using data structures and algorithms.

GitHub Repository

The complete source code for this project can be found on GitHub at the following link:

https://github.com/hzjadoon/DSA_project