TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PASHCHIMANCHAL CAMPUS

A Project Report

On

Student Data Management System

Submitted By:

Aabiskar Regmi (PAS079BCT001)

Submitted To:

Department of Electronics and Computer Engineering

Pashchimanchal Campus

Pokhara, Nepal

August 10, 2024

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported and guided me throughout the process of conducting this report on **Student Data Management System**. This endeavor would not have been possible without their valuable contributions and assistance.

I extend my heartfelt thanks to my supervisor **Er. Hari Prasad Baral**, whose expertise and guidance have been instrumental in shaping the direction of this project. His insightful feedback and continuous encouragement have been invaluable in refining the scope and focus of this project.

I would also like to thank my friends and colleagues for their meaningful discussions, brainstorming sessions, and moral support. This work would not have been possible without the collective efforts of these individuals and organizations. While any shortcomings in this report are solely my responsibility, their contributions have significantly enriched its content.

Thank you.

Aabiskar Regi(PAS079BCT001)

ABSTRACT

This project report presents the design and implementation of a Student Data Management System developed using C++. The primary objective of the system is to streamline the management of student information within an educational institution. The system allows for the efficient handling of student records, including functionalities such as adding new students, displaying existing records, sorting data based on various criteria, searching for specific students, and deleting records when necessary.

The project incorporates essential programming concepts such as object-oriented programming, file handling, and basic encryption techniques to ensure data security. The system features a user-friendly interface that guides the user through various operations, ensuring ease of use and reliability. Additionally, the system includes a secure login mechanism with the option to change credentials, thereby safeguarding the integrity of the data.

The project demonstrates a practical application of C++ in creating a robust and efficient management tool that can significantly improve data management processes in educational settings. The system's ability to handle up to 100 student records, along with its various sorting and searching capabilities, highlights its effectiveness in organizing and retrieving student information. This report details the development process, the challenges faced, and the solutions implemented to create a functional and reliable system.

Table of Contents

1. INTRODUCTION

In today's educational institutions, efficient management of student data is essential for the smooth functioning of administrative processes. From recording personal details to tracking academic performance, a well-organized system is crucial for ensuring that student information is accessible, secure, and up-to-date. The Student Data Management System developed in this project addresses these needs by providing a comprehensive solution for managing student records. Built using C++, the system offers various functionalities, including adding, displaying, sorting, searching, and deleting student data. The system is designed with a focus on usability, security, and efficiency, making it a valuable tool for educational institutions of varying sizes.

1.1 Background Introduction

In educational institutions, managing student records through traditional manual methods is not only time-consuming but also susceptible to errors and inconsistencies. As institutions grow, the volume of data increases, making it challenging to maintain accuracy and efficiency in administrative tasks. To address these challenges, digital solutions have become crucial in streamlining the management of student information.

This project, developed using object-oriented programming (OOP) principles, presents a Student Data Management System designed to simplify the process of storing, updating, and retrieving student records. By utilizing file handling for persistent data storage and basic encryption techniques to secure login credentials, the system provides a secure and efficient method to manage student data. It aims to reduce the administrative burden, minimize human error, and ensure the accuracy and reliability of student information within the institution.

1.2 Motivation

The Student Data Management System was developed to address the inefficiencies and errors associated with manual student record management in educational institutions. As student numbers grow, traditional methods like paper-based systems or simple spreadsheets become increasingly inadequate. The motivation for this project lies in creating a secure, efficient, and scalable solution using object-oriented programming. By streamlining data management tasks such as sorting, searching, and secure access, this system aims to reduce administrative burden and improve the accuracy and reliability of student information.

1.3 Objectives

The main objectives of the program are listed below:

- **Efficient Data Management**: To develop a system that allows educational institutions to efficiently store, manage, and retrieve student information, reducing the reliance on manual record-keeping and minimizing the risk of data errors.
- **Secure Access and User Authentication**: To implement secure access controls and user authentication mechanisms, ensuring that only authorized personnel can view or modify student records, thereby protecting sensitive information from unauthorized access.

1.4 Scope

The Student Data Management System will efficiently manage and organize student records, enabling secure storage, retrieval, and modification of data. Key features include sorting, searching, and deleting records, along with user authentication for data security. The system is designed for small to medium-sized institutions and is built using C++ with a focus on modularity and ease of maintenance.

2. LITERATURE REVIEW

The development of student data management systems has been extensively explored through various frameworks and methodologies. The GitHub repository OOP-Project provides a practical example of a student management system using object-oriented programming (OOP) principles. This project demonstrates effective data handling through modular code, showcasing best practices in encapsulation and code reuse, which are essential for managing complex data structures efficiently.

Insights from Stack Overflow discussions emphasize the challenges and solutions related to data handling and security. Key issues include managing large datasets and ensuring robust encryption techniques to protect sensitive information. These discussions highlight the importance of practical solutions for file handling and user authentication, aligning with established practices in data management.

The Scaler article on Student Record Management Systems in C++ complements this by offering specific guidance on implementing such systems using C++. It covers essential aspects like file streams for data persistence, encryption for security, and practical considerations for error handling and validation. Combined with advice from AI tools like ChatGPT and Claude AI, this review underscores the importance of creating secure, scalable, and user-friendly systems tailored to diverse institutional needs.

3. METHODOLOGY

3.1 System Development Approach

To create the Student Data Management System, we utilized an iterative and incremental development approach. This methodology allowed us to develop the system in stages, enabling ongoing refinement and adjustments based on feedback and testing. By choosing C++, we leveraged its object-oriented programming capabilities to build a modular and maintainable system, which facilitated easier updates and improvements throughout the development process.

3.2 Requirements Gathering

The initial phase involved comprehensive requirements gathering. We conducted research into existing student management systems and reviewed relevant literature to identify essential features and functionalities. Our primary objectives included:

- **Data Handling:** Implementing features to add, display, sort, search, and delete student records.
- **Security:** Incorporating secure login functionality to manage user credentials and protect sensitive information.

By analyzing these requirements, we established a clear roadmap for the system's development.

3.3 System Design

Our design focused on simplicity and efficiency, encompassing the following aspects:

- **Class Design:** The system comprises two core classes: `Student` and StudentManager. The `Student` class manages individual student data, while the StudentManager class oversees operations related to student records.
- **File Management:** We used two primary files for data storage:
  - `students.txt` for storing student information.
  - `password.txt` for managing login credentials, with added encryption for security.
- **User Interface:** A menu-driven interface was designed to ensure a user-friendly experience, allowing users to easily navigate and perform various functions within the system.

3.4 Algorithm

The following algorithm outlines the steps involved in the Student Data Management System:

1. **Start Program**
   - Begin the execution of the program.
   - Display a loading or welcome screen if needed.
2. **Initialization**
   - Define constants for file names, encryption key, and the maximum number of students.
   - Define the ‘Student‘ class with attributes: ‘name‘, ‘address‘, ‘roll number‘, ‘contact number‘, ‘faculty‘, ‘semester‘, and ‘year‘.
   - Define the ‘StudentManager‘ class to handle loading, saving, adding, and managing student data.
3. **File Initialization**
   - Call the ‘initializeFiles()‘ function:
     - Check if the password file exists. If it does not, create the password file with default credentials (‘admin‘, ‘admin‘), encrypted using XOR encryption.
     - Check if the student file exists. If it does not, create an empty student file.
4. **Display Home Page**
   - Call the ‘displayHomePage()‘ function to show a welcome message with system and developer information.
5. **Login Process**
   - Call the ‘login()‘ function:
     - Load stored username and password by decrypting them from the password file.
     - Prompt the user to enter the username and password.
     - If the credentials match, grant access to the system. If not, display an error message and exit the program.
6. **Main Menu**
   - Display the main menu with the following options:
     - Option 1: Display existing data.
     - Option 2: Add new data.
     - Option 3: Sort data.
     - Option 4: Search data.
     - Option 5: Delete data.

- Option 6: Change login credentials.
- Option 7: Exit the program.

7. **Main Program Loop**
   - Repeat the following steps until the user chooses to exit:
     - **Option 1: Display existing data**
       - Call ‘loadStudents()‘ to load the student records from the file.
       - Call ‘disstd()‘ to display the student records in a tabular format.
     - **Option 2: Add new data**
       - Call ‘addstd()‘.
       - Prompt the user to enter student details: ‘name‘, ‘address‘, ‘roll number‘, ‘contact number‘, ‘faculty‘, ‘semester‘, and ‘year‘.
       - Store the new student data and append it to the student file.
     - **Option 3: Sort data**
       - Call ‘sortstd()‘.
       - Prompt the user to choose a sorting criterion (Name, Roll Number, or Year).
       - Sort the student data based on the selected criterion.
       - Save the sorted data back to the student file.
     - **Option 4: Search data**
       - Call ‘searchstd()‘.
       - Prompt the user to enter search criteria (Name or Roll Number).
       - Display the matching student record(s).
     - **Option 5: Delete data**
       - Call ‘deletestd()‘.
       - Prompt the user to enter the roll number of the student to be deleted.
       - Remove the corresponding student record from the file.
     - **Option 6: Change login credentials**
       - Call ‘changeCredentials()‘.
       - Prompt the user to enter the new username and password.
       - Encrypt and save the new credentials to the password file.
     - **Option 7: Exit the program**
       - Display a message indicating the program is exiting.
       - Terminate the program.

3.5 Implementation

The implementation phase involved translating the design into functional code. Key tasks included:

- **Data Management:** Developing functions to load, save, and add student records, utilizing file operations to ensure data persistence.
- **Encryption:** Implementing a basic XOR encryption method to secure login credentials.

3.6 Testing

Testing was conducted in multiple stages to ensure system functionality:

- **Unit Testing:** Each component and function was tested individually to verify correctness.
- **Integration Testing:** We tested the interactions between different components to ensure seamless integration.
- **User Testing:** End-users tested the system to provide feedback, which guided final adjustments and improvements.

3.7 Documentation and Reporting

Detailed documentation was maintained throughout the project:

- **Code Documentation:** Comprehensive comments and explanations were included in the code to enhance readability and facilitate future maintenance.
- **User Documentation:** Guides and instructions were prepared to assist users in navigating and operating the system effectively.

4. SYSTEM DESCRIPTION

4.1 System Overview

The Student Data Management System is designed to efficiently handle student records within an educational institution. The system provides functionalities to add, display, sort, search, and delete student information. It also includes features for managing user login credentials, ensuring secure access to the system.

4.2 System Components

The system comprises the following key components:

- **User Interface:** Provides an interactive interface for users to perform operations like viewing, adding, and managing student records. It includes a home page, login page, and a main menu for various functionalities.
- **Student Data Storage:** Uses file-based storage to save and retrieve student records. This includes a student data file and a password file for authentication.
- **Student Class:** Defines the structure for storing student information, including attributes such as name, address, roll number, contact , faculty, semester, and year.
- **StudentManager Class:** Handles the core functionalities of the system, including loading, saving, adding, and managing student data.
- **Encryption and Decryption:** Implements XOR encryption for secure storage of login credentials.

4.3 Functional Description

The system operates based on the following core functionalities:

1. **Initialization:** The system initializes by checking and setting up necessary files for storing student data and user credentials.
2. **Home Page Display:** Displays a welcome message and system information to the user.
3. **Login Process:** Ensures secure access by validating user credentials against stored encrypted data.
4. **Main Menu Operations:** Provides options for users to display, add, sort, search, delete student records, change login credentials, or exit the program.
5. **File Operations:** Includes functions for loading, saving, and updating student data files as well as managing user credentials.

4.4 Flowchart

The flowchart below illustrates the high-level workflow of the Student Data Management System, capturing the main processes from program start to user interactions.

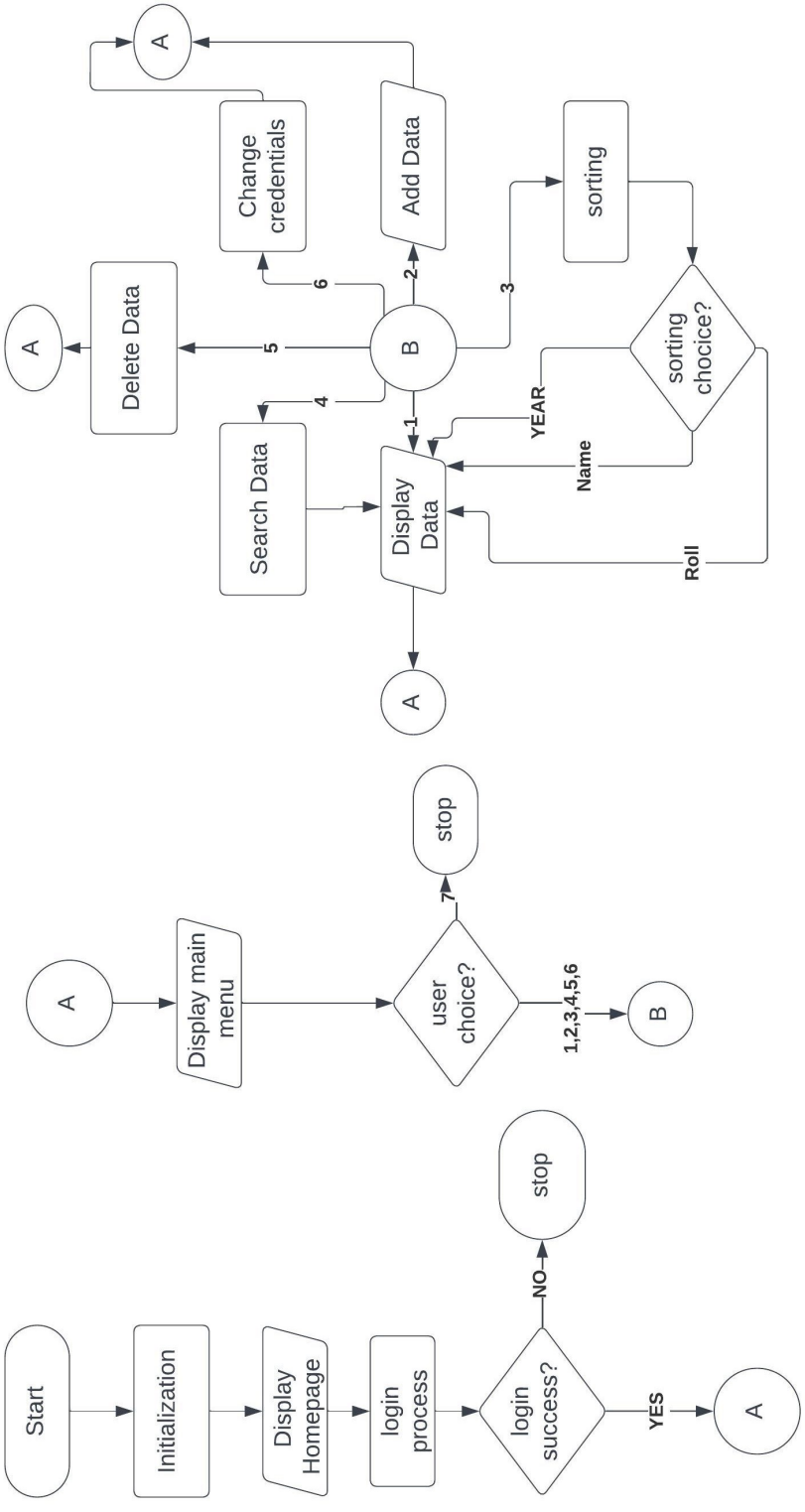

Figure 4.1: Flowchart of the Student Data Management System.

5. RESULT AND ANALYSIS

In this project, we successfully developed a Student Data Management System using C++. The system includes functionalities for adding, displaying, sorting, searching, and deleting student records, as well as managing user credentials for secure access. Throughout the development process, we focused on ensuring data integrity, ease of use, and security through simple encryption techniques for user credentials.

The system was tested with sample data to evaluate its performance. The following observations were made:

- **Data Entry and Storage**: The system efficiently stores student records in a structured format. Data entry is straightforward, and records are stored securely in a file, ensuring that data is not lost between sessions.
- **Search and Sort Functionality**: The implemented search and sort algorithms performed effectively, enabling quick retrieval and organization of student data. Sorting by roll number or name helps in easy navigation through the records.
- **User Interface**: The console-based user interface, while basic, is functional and user-friendly. The prompts are clear, and the flow of operations is logical, making the system easy to use even for those with minimal technical expertise.
- **Security Measures**: The system includes a simple yet effective password protection mechanism. Although the encryption is basic, it adds a layer of security to the system, ensuring that only authorized users can access or modify student records.

Overall, the project met its objectives, providing a reliable and secure solution for managing student data. However, there are areas for improvement, particularly in enhancing the user interface and security features, which could be explored in future iterations.

6. IMAGES AND SCREENSHOTS

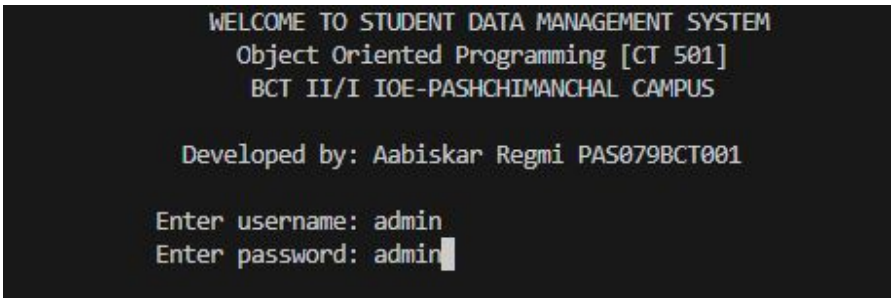

Figure 6.1: Homepage

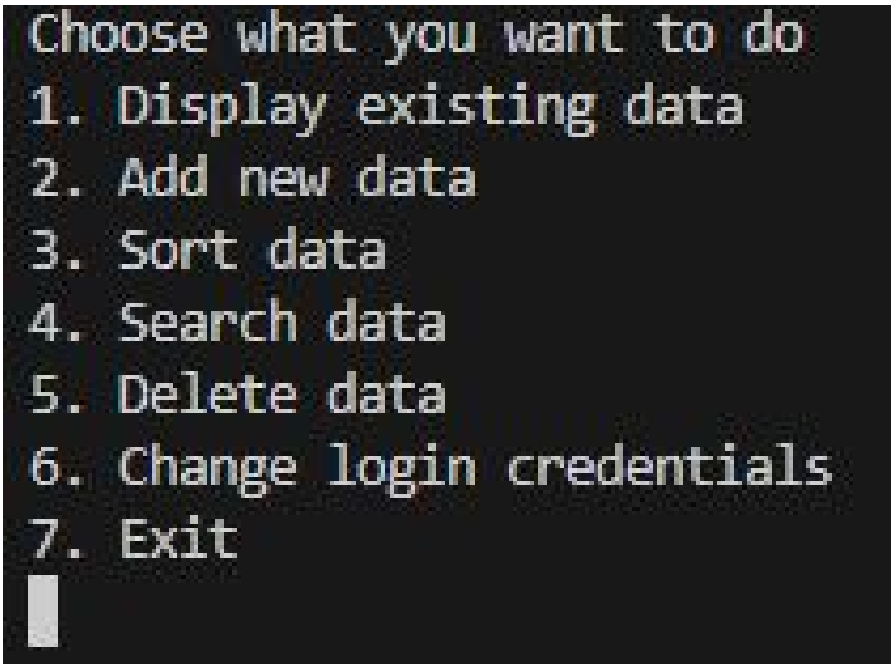

Figure 6.2: Menu Screen

```
Enter student data (enter 'q' to stop):
Enter name (or 'q' to quit): Aabiskar Regmi
Enter address: Machhapuchchhre
Enter roll no: 1
Enter contact no: 9849942073
Enter faculty: BCT
Enter semester: 3
Enter year: 2
Enter name (or 'q' to quit): Abhishek Regmi
Enter address: kaski
Enter roll no: 2
Enter contact no: 9880184085
Enter faculty: BCE
Enter semester: 6
Enter year: 3
Enter name (or 'q' to quit): Ganesh Regmi
Enter address: Dang
Enter roll no: 4
Enter contact no: 9856089633
Enter faculty: BEI
Enter semester: 2
Enter year: 1
Enter name (or 'q' to quit): q
Data added successfully!

Press Enter to continue...
```

Figure 6.3: Add Data Screen

```
S.N.    Name           Address         Roll No.        Contact         Faculty   Semester  Year
-------------------------------------------------------------------------------------------------
1       Aabiskar Regmi  Machhapuchchhre 1               9849942073      BCT       3         2
2       Abhishek Regmi  kaski           2               9880184085      BCE       6         3
3       Ganesh Regmi    Dang            4               9856089633      BEI       2         1

Press Enter to continue...
```

Figure 6.4: Display All Records

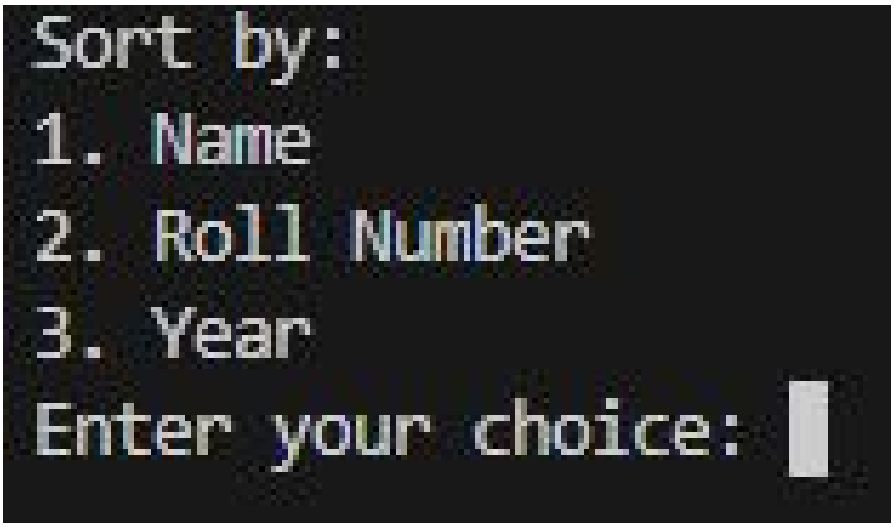

Figure 6.5: Sorting Menu

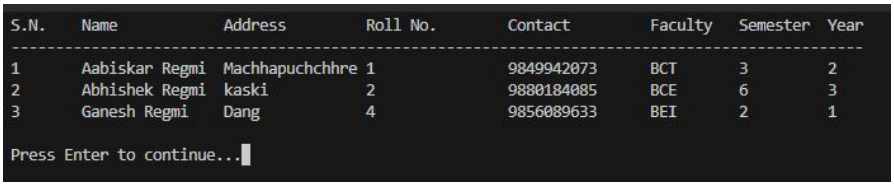

Figure 6.6: Sort by Name

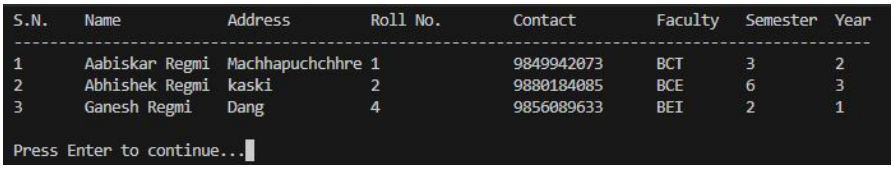

Figure 6.7: Sort by Roll Number

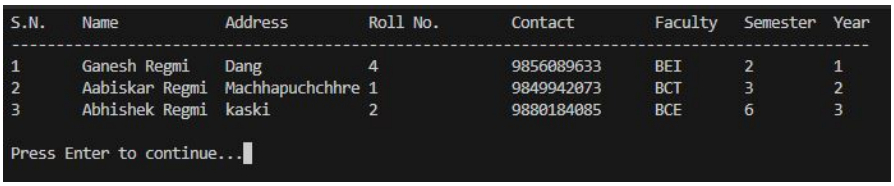

Figure 6.8: Sort by Year

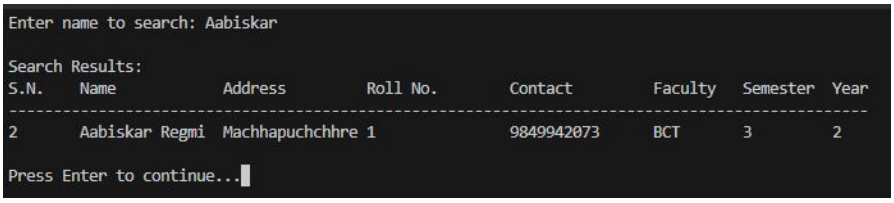

Figure 6.9: Search Screen

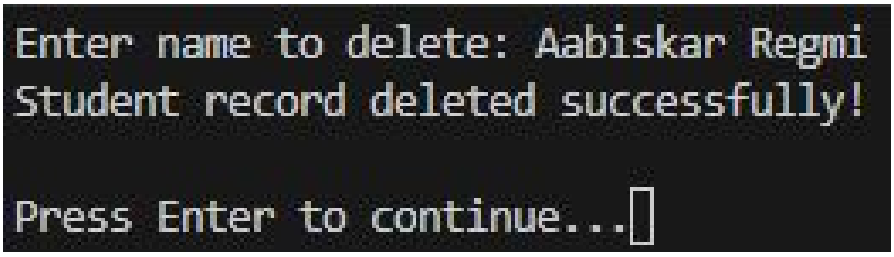

Figure 6.10: Delete Screen

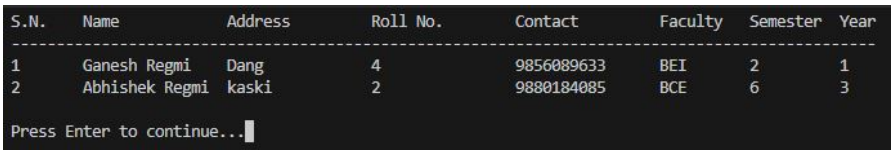

Figure 6.11: Data after deletion

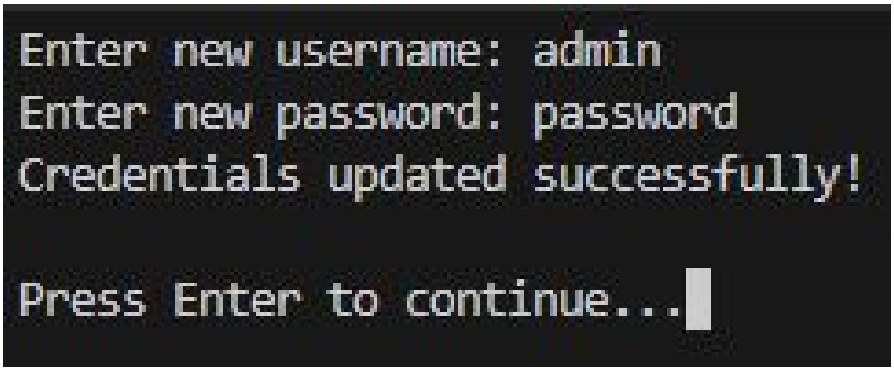

Figure 6.12: Change login details

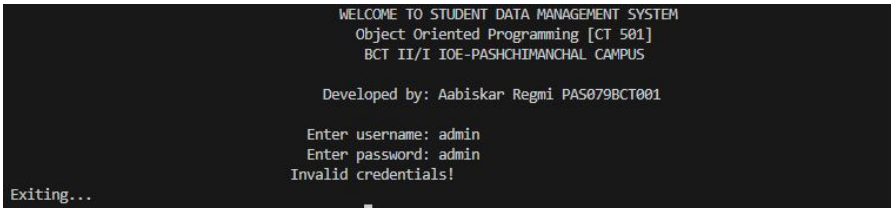

Figure 6.13: Uauthorized attempt

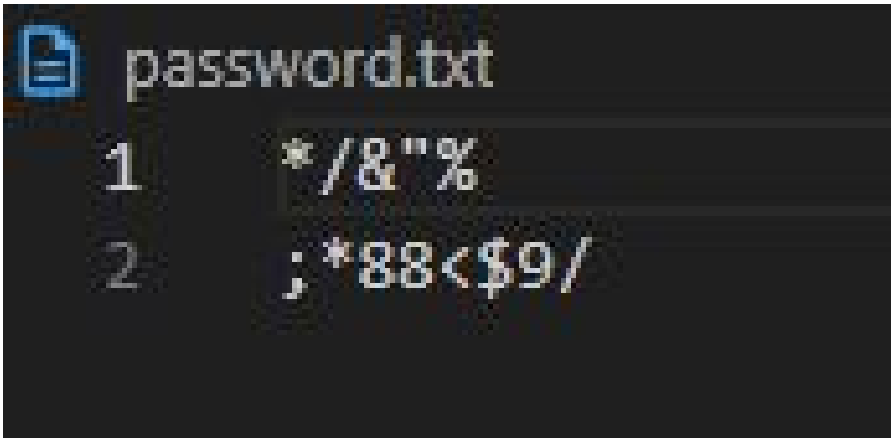

Figure 6.14: Password fiel screen

7. RECOMMENDATION AND FUTURE ENHANCEMENT

Recommendation

The Student Data Management System lays a solid foundation but can be improved with the following:

1. **Data Validation:** Ensure correct format for all inputs, including contact numbers and roll numbers.
2. **Error Handling:** Enhance error messages for file operations to aid in troubleshooting.
3. **Password Security:** Upgrade to secure hashing algorithms like bcrypt for login credentials.
4. **User Interface Enhancements:** Consider a GUI or menu-driven system to improve user experience.
5. **Search Functionality:** Expand search to include criteria such as roll number, semester, and faculty.
6. **Data Backup and Recovery:** Implement a system to protect against data loss and facilitate recovery.

Future Advancements

Future improvements could include:

1. **Cloud Storage:** Migrate data to the cloud for better accessibility and backup.
2. **System Integration:** Link with other educational tools for a comprehensive view of student data.
3. **Data Analytics:** Add features for generating performance reports and analyzing trends.
4. **Mobile Access:** Develop a mobile app for on-the-go data management.
5. **Security Enhancements:** Introduce role-based access control and two-factor authentication.
6. **AI Integration:** Use AI for tasks like automated data entry and personalized recommendations.

Implementing these suggestions will enhance the system's robustness, security, and usability, better serving educational institutions.

8. CONCLUSION

The Student Data Management System, developed using C++, effectively addresses the challenges of manual student record management. By implementing core Object-Oriented Programming (OOP) concepts like classes, objects, inheritance, and polymorphism, the system provides a robust and efficient solution for storing, managing, and retrieving student information.

Through this project, we gained a deeper understanding of C++ programming, particularly its application in real-world problem-solving. The experience of designing, coding, and testing the system solidified our grasp of OOP principles and their practical implementation.

While the current system fulfills its basic functionalities, there's significant potential for enhancement. Incorporating advanced features like data analytics, cloud integration, and improved security can elevate the system to a more comprehensive and sophisticated solution for educational institutions.

Overall, this project serves as a valuable learning experience and a foundation for future development in the field of software engineering.

9. BIBLIOGRAPHY

1. Lafore, R. (2002). *Object-oriented programming in C++* (4th ed.). Sams Publishing.
2. `https://github.com/UAws/OOP-Project`.
   GitHub - OOP Project

3. `https://www.scaler.com/topics/student-record-management-system-in-cpp`.
   Scaler - Student Record Management System in C++

4. `https://stackoverflow.com/`.
   Stack Overflow