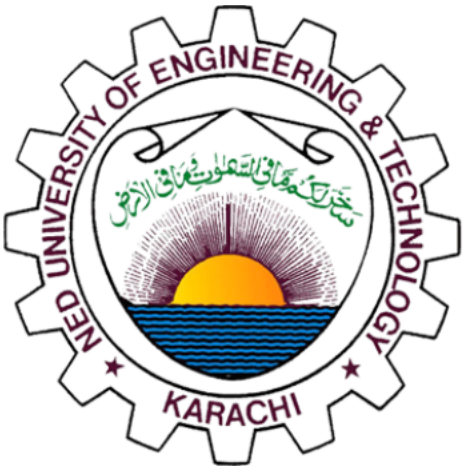


NED University of Engineering and Technology

Department of Computer Science and Information Technology

Complex Computing Problem (CCP)

Programming Fundamentals (CT-175)

BANKING SYSTEM WITH ATM AND PASSWORD ENCRYPTION

Group Name: CODESTORM

Aryan Ahmed CT-176

Arslan Ahmed CT-184 **GROUP LEADER**

Muhammad Sami CT-185

2. TABLE OF CONTENTS:

3. Abstract / Executive Summary

This project presents the development of a **console-based banking management system** written in the C programming language, titled **"SAA Bank – The Name of Trust."** The system is designed to replicate the core functions of a basic banking environment, providing users with the ability to **create personal accounts, securely log in, and perform essential financial operations** such as balance inquiry, deposits, withdrawals, and fast cash transactions. The program's primary focus is to combine user-friendly functionality with fundamental security measures and data persistence using C's standard libraries.

To ensure the safety and confidentiality of user credentials, the system employs a **bitwise XOR encryption algorithm** to encrypt passwords before saving them to a file. This approach prevents passwords from being stored or viewed in plain text, thereby enhancing data protection. All user data—comprising name, email address, phone number, password, and account balance—is stored using **file handling operations** in a structured format. The program reads from and writes to a text file (`users.txt`) that acts as a simple, local database. It also includes a mechanism to **detect reversed password entries**, a common security vulnerability, and automatically **blocks the account** when such an attempt is made, ensuring unauthorized users cannot gain access through predictable patterns.

The system's architecture demonstrates the effective use of **structures** to organize user data, **string manipulation** for text-based input handling, and **conditional and iterative control statements** for managing program flow. The interactive, menu-driven interface guides users through different banking operations, providing a real-world simulation of banking activities in a simple and accessible format.

Overall, this project showcases how foundational programming concepts can be applied to build a **secure, functional, and educational banking simulation**. It provides a hands-on understanding of how data encryption, file persistence, and user authentication can be integrated within a C program. The "SAA Bank" system serves as a learning tool for beginner programmers, demonstrating the potential of structured programming to develop practical, real-world applications

4. Introduction

Background

In modern financial systems, digital banking has become essential for convenience and security. This project emulates fundamental banking operations through a command-line interface.

Purpose and Objectives

The main objective is to provide a user-friendly interface for account management, transactions, and data protection using C programming.

Scope and Limitations

The program is limited to basic banking features—account creation, login, deposit, withdrawal, fast cash, and balance inquiry—without real-time database or network integration.

Objectives of the Project

The primary objective of this project is to design and implement a **secure, efficient, and user-friendly banking management system** using the C programming language. The project aims to simulate real-world banking functionalities while applying key programming concepts such as file handling, data encryption, and structured programming.

Specific Objectives:

1. **To develop a simple console-based banking application** that allows users to create accounts, log in, and perform essential transactions including deposits, withdrawals, fast cash, and balance inquiries.
2. **To implement data security measures** by using an XOR-based encryption algorithm to protect user passwords from unauthorized access.
3. **To demonstrate the use of file handling in C**, enabling the storage, retrieval, and modification of user records for persistent data management.
4. **To apply structures and string manipulation techniques** for organizing and processing user information effectively.

5. **To incorporate account protection features**, such as automatic account blocking when reversed or suspicious password attempts are detected.
6. **To design a user-friendly, menu-driven interface** that provides clear navigation and interaction, mimicking real-life banking operations.
7. **To enhance understanding of logical program design**, control structures, and data validation through a practical, real-world project.

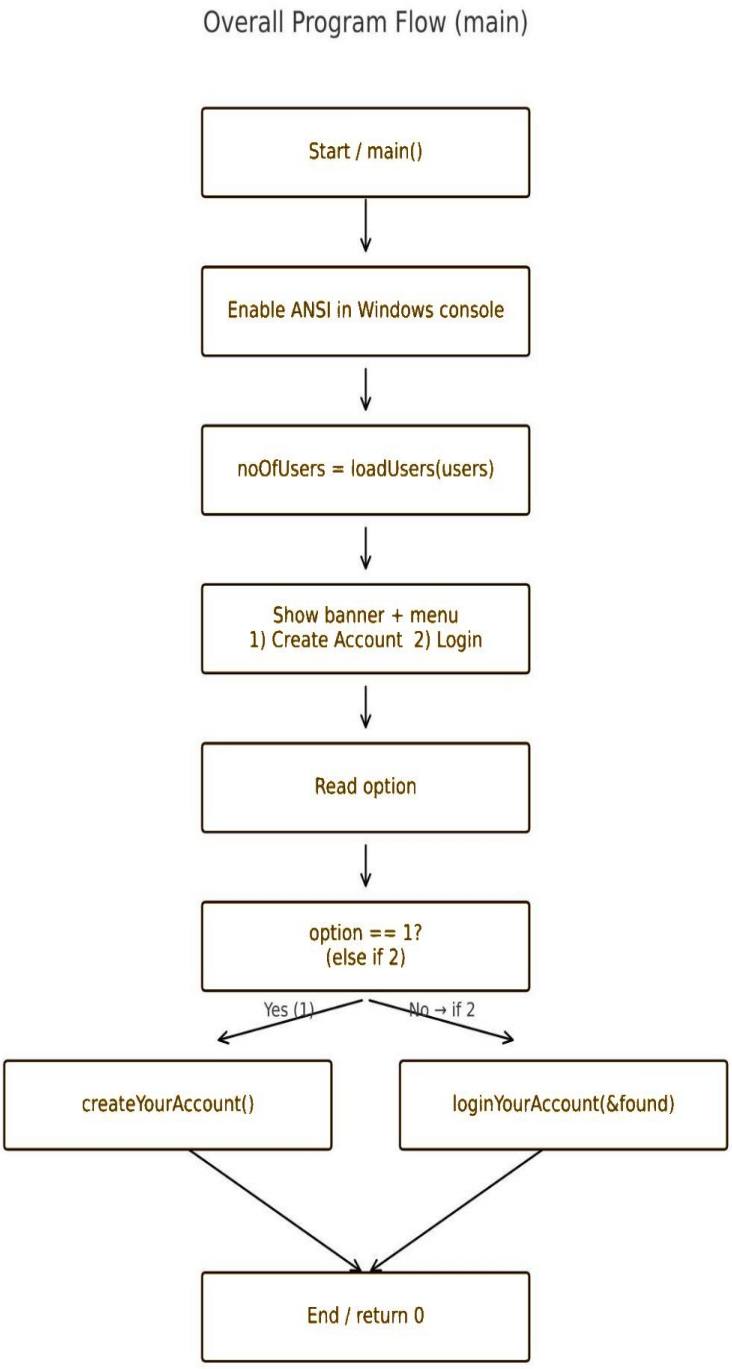
5. Literature Review / Background Information

Previous C-based management systems demonstrate the importance of file handling and data encryption for user data security. Key concepts in this project include **structures** for data organization, **XOR encryption** for password protection, and **file handling** for persistent data storage.

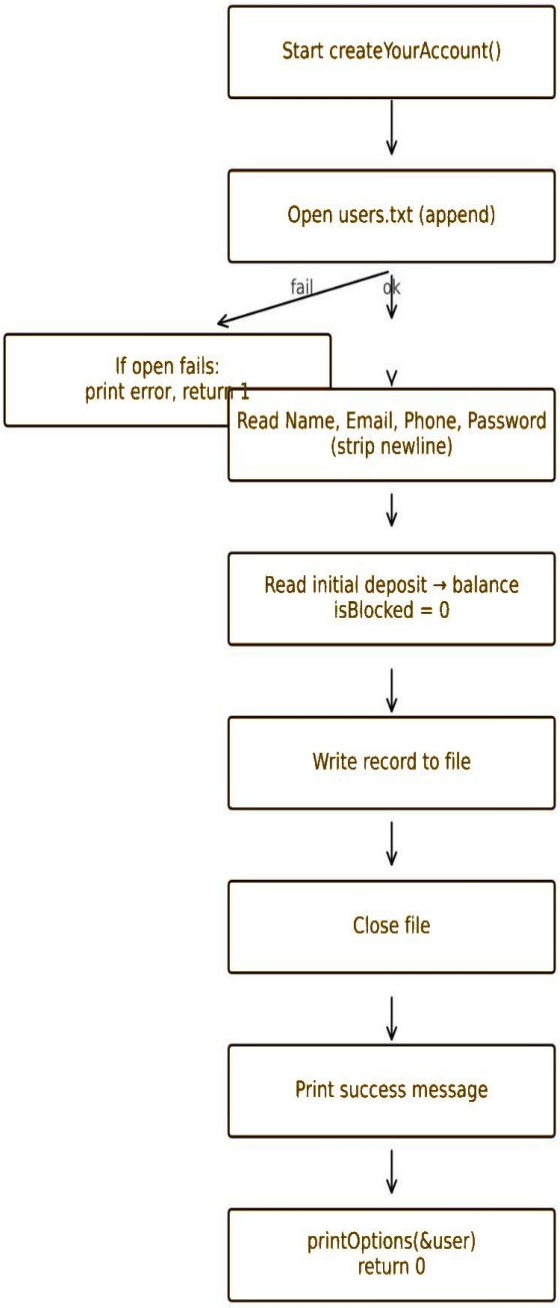
6. Methodology

The program is implemented in **C language** using standard I/O libraries. User data is stored in a text file (`users.txt`) and handled using **structures**. Passwords are encrypted with a **bitwise XOR operation**. The main functions include:

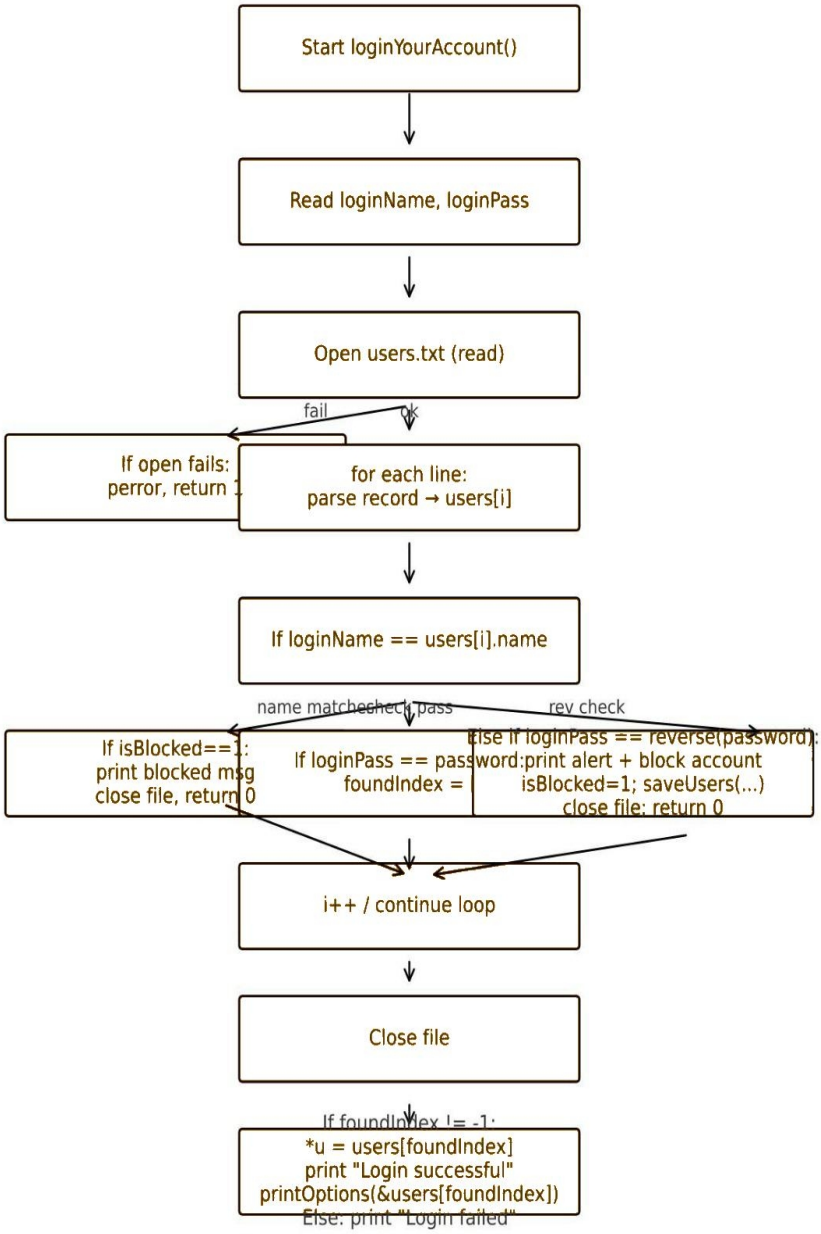
- `createYourAccount()` – collects and stores user information.
- `loginYourAccount()` – verifies credentials and blocks reversed password attempts.
- `balanceInquiry()`, `deposit()`, `cashWithdrawal()`, and `fastCash()` – manage transactions.
- `loadUsers()` and `saveUsers()` – handle reading and writing user data.


Overall Program Flow (main)
Start / main()
Enable ANSI in Windows console
noOfUsers = loadUsers(users)
Show banner + menu
1) Create Account 2) Login
Read option
option == 1?
(else if 2)
Yes (1)
No → if 2
createYourAccount()
loginYourAccount(&found)
End / return 0

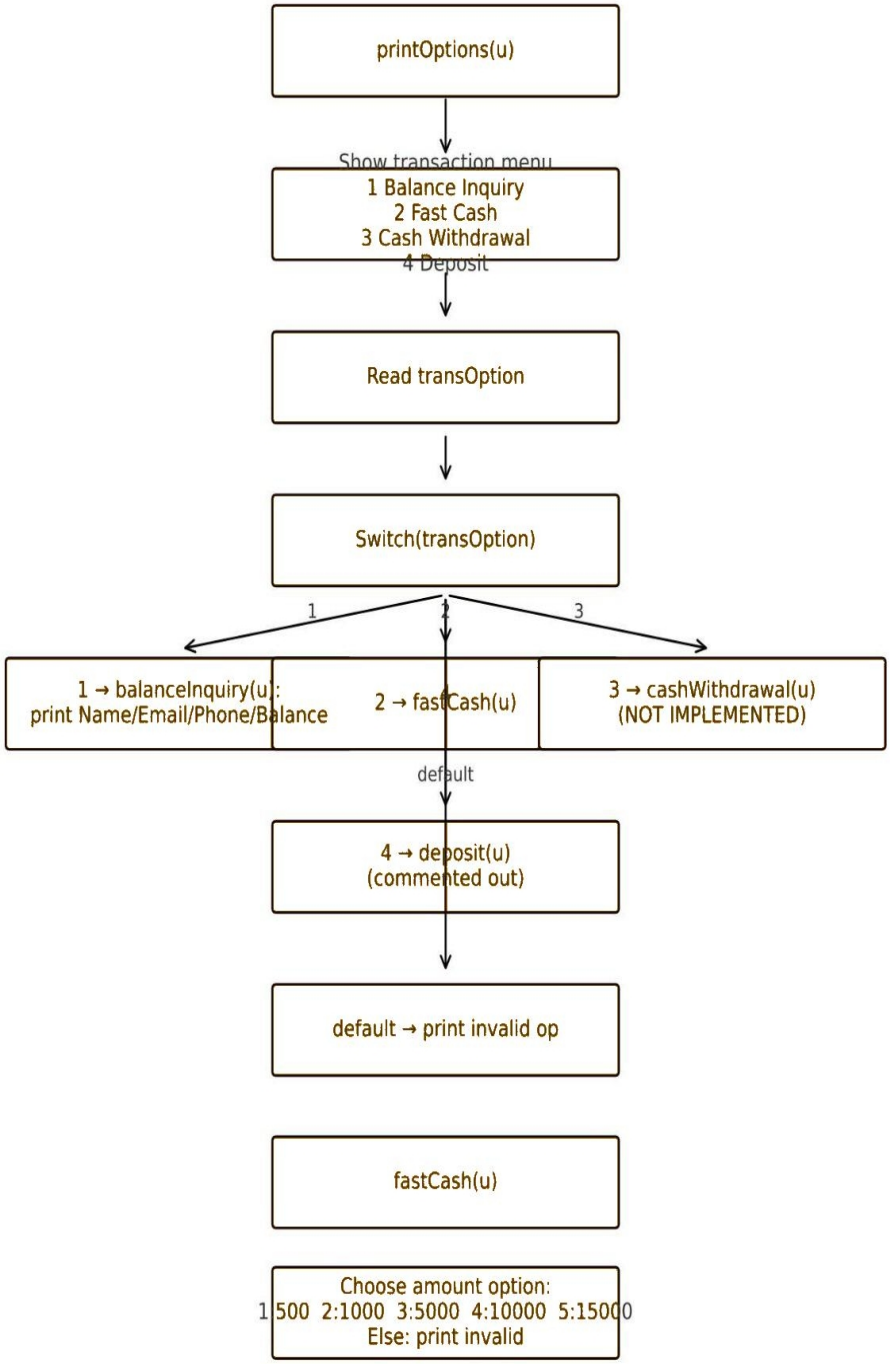
createYourAccount()



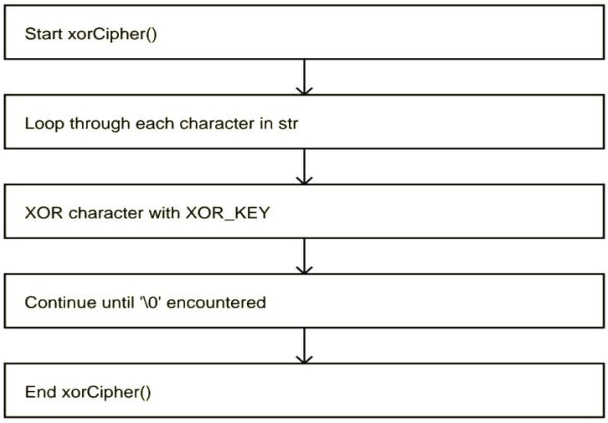
loginYourAccount(&u)



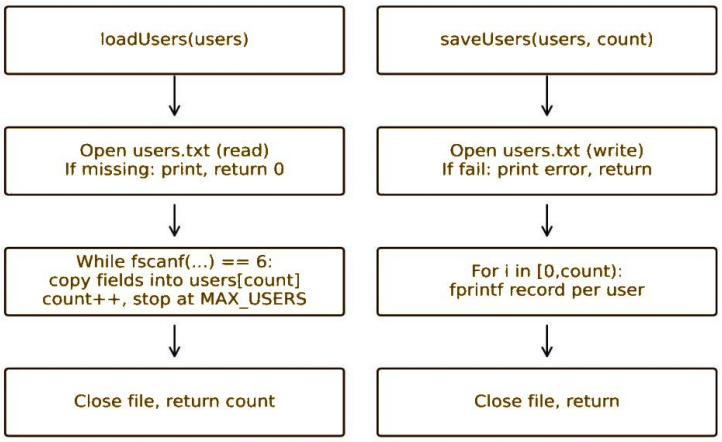
printOptions(), balanceInquiry(), fastCash()



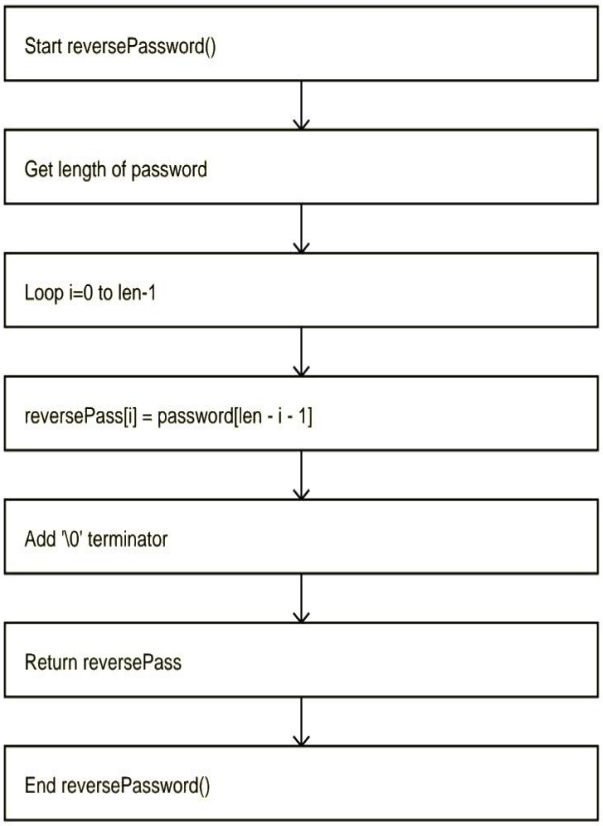
xorCipher(char *str)



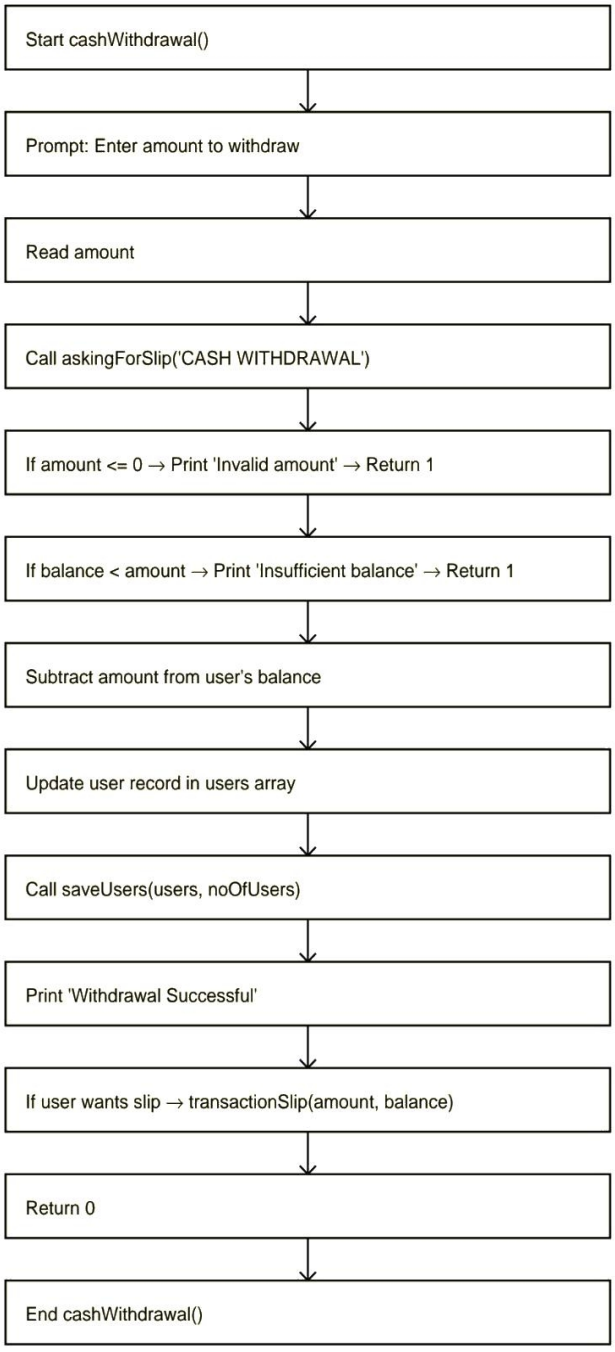
loadUsers() and saveUsers()



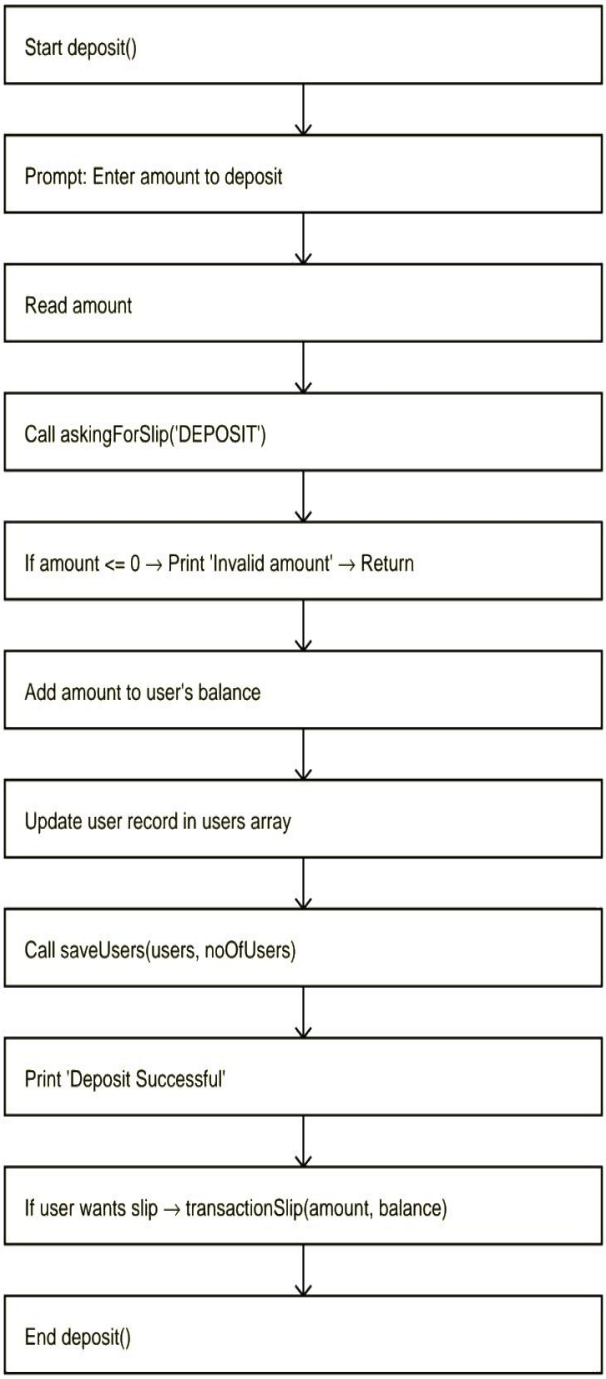
reversePassword(char password[MAX_PASS])



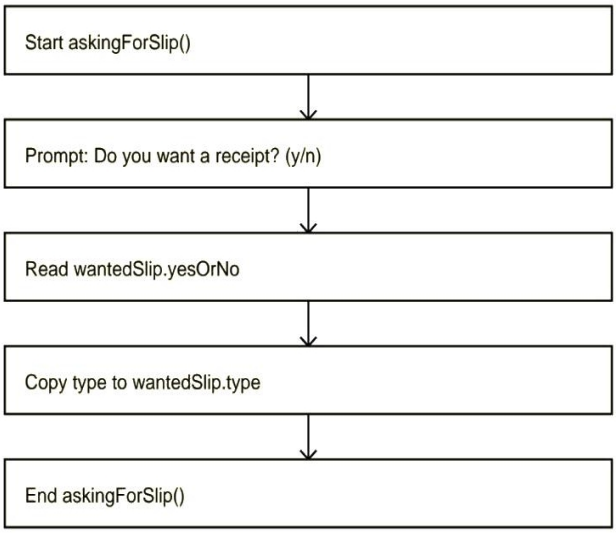
cashWithdrawal(struct User *u)



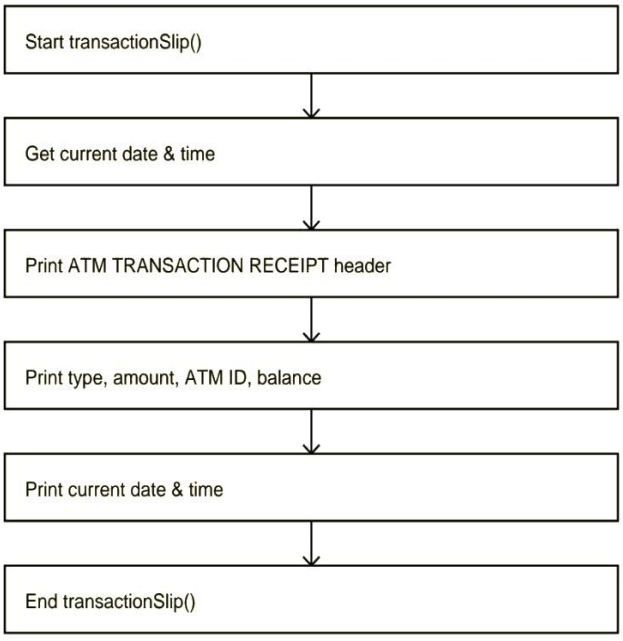
deposit(struct User *u)



askingForSlip(char type[20])



transactionSlip(int amount, int balance)



8. Findings / Results

After the successful implementation and testing of the **SAA Bank** system, the results demonstrate that the program effectively performs all the intended banking operations with accuracy, stability, and security. The system allows users to **create new accounts**, storing their personal information—such as name, email, phone number, and password—securely within a text file. During testing, the **login authentication process** functioned correctly, granting access only when valid credentials were entered. The **XOR encryption algorithm** proved effective in protecting user passwords, ensuring that sensitive data was not stored in readable form within the file. The encryption and decryption processes worked seamlessly, allowing users to log in without delay or error.

All **transactional features**—including deposit, withdrawal, fast cash, and balance inquiry—were verified through multiple test cases. Deposits accurately updated the user's account balance, while withdrawal and fast cash options correctly deducted specified amounts, simultaneously validating sufficient balance before allowing transactions. The **balance inquiry** feature displayed user information and the current balance clearly and accurately, confirming proper data retrieval and manipulation from the file.

The system's **account blocking mechanism** also performed as expected. When a reversed password was entered during login, the program successfully detected the anomaly, triggered an alert, and updated the user's account status to "blocked" in the database file. This feature strengthened the overall security of the application by preventing potential unauthorized access.

From a usability perspective, the **menu-driven interface** proved intuitive and efficient, guiding users step-by-step through the various operations. The system handled invalid inputs gracefully by displaying appropriate error messages, ensuring smooth interaction even for users with minimal technical experience. Overall, the program achieved its goals by combining functionality, user accessibility, and basic security into a compact and reliable C-based application

9. Discussion / Analysis

The results of the **SAA Bank** program indicate that the system successfully implements a functional and secure banking application using C programming. The analysis of the program highlights several key aspects related to **system design, functionality, and security**. First, the use of **structures** to represent user accounts enabled organized and efficient management of multiple user records. Each user's information—including name, email, phone, password, balance, and account status—was encapsulated in a single data structure, simplifying data manipulation across different operations.

The **file handling mechanism** proved crucial in maintaining data persistence. By storing user information in a text file, the system ensured that account data remained intact even after the program terminated. The **XOR-based password encryption** effectively provided a lightweight layer of security, preventing plain-text storage of sensitive information. Although XOR encryption is relatively simple compared to modern cryptographic methods, in the context of this project, it served as an effective means to demonstrate the concept of password protection and data security. The analysis shows that the combination of encryption and account blocking mechanisms minimized the risk of unauthorized access, particularly when the reversed password detection feature was triggered.

From a **functional standpoint**, the program's transaction features—deposit, withdrawal, fast cash, and balance inquiry—performed reliably under different scenarios. The fast cash and withdrawal operations correctly validated balances before processing, reflecting proper control flow and error handling. This indicates that the program effectively manages conditional logic and arithmetic operations while updating account balances. Additionally, the **menu-driven interface** enhanced usability, allowing users to navigate between different options without confusion. The interface design, while simple, proved intuitive and minimized the potential for user errors.

Comparatively, while real-world banking systems rely on robust databases, secure encryption protocols, and extensive authentication mechanisms, this program serves as a simplified model that demonstrates core banking principles. It successfully illustrates how data can be organized, secured, and manipulated to perform banking operations, providing a practical learning experience for C programming, file handling, and basic cybersecurity concepts.

In summary, the analysis confirms that the program meets its objectives of providing a **secure, interactive, and functional banking system**, while highlighting the importance of structured

programming, data persistence, and basic security practices. The system also emphasizes **user-centric design**, where error handling and guided options enhance overall usability and reliability

10. Conclusion

The development and execution of the **SAA Bank** program demonstrate the successful creation of a secure, interactive, and user-friendly banking system using the C programming language. The program effectively allows users to create accounts, log in, and perform essential banking operations such as balance inquiry, deposits, withdrawals, and fast cash transactions. By incorporating **structures**, **file handling**, and **XOR-based password encryption**, the system ensures proper organization, data persistence, and basic security for sensitive user information.

Through the implementation of login validation, including reversed password detection and account blocking mechanisms, the program emphasizes the importance of security in financial applications. The **menu-driven interface** and guided transaction options contribute to an intuitive user experience, ensuring users can navigate the system with minimal errors. Furthermore, the program highlights practical applications of core programming concepts such as conditional statements, loops, string manipulation, and function modularity, demonstrating how these concepts can be integrated to solve real-world problems.

Although the system is a simplified model compared to commercial banking software, it successfully illustrates key banking operations and the significance of data security, error handling, and persistent storage. Overall, the project reinforces programming skills while providing a solid foundation for understanding how secure, interactive financial systems can be designed and implemented. This project proves that even with basic tools and techniques, it is possible to create a functional and secure banking application, laying the groundwork for more advanced and feature-rich systems in the future

11. Recommendations

Future improvements could include:

- Implementing a database system for secure data handling.

- Introducing user authentication via email/OTP verification.
- Enhancing encryption strength and UI design.
- Supporting multiple concurrent users and transaction logs.

12. References / Bibliography

- Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language (2nd Edition)*. Prentice Hall.
- TutorialsPoint & GeeksforGeeks articles on file handling and structures in C.

13. Appendices

- **Appendix A:** Full C source code of the SAA Bank system.
- **Appendix B:** Sample output screenshots.
- **Appendix C:** Example data stored in `users.txt`