TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Proposal

On

Bank Management System

Submitted by:

Bidhan Shrestha THA081BEI009

Kabindra Panta THA081BEI012

Nirajan Chaudhary THA081BEI022

Shuvam Neupane THA081BEI043

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

February, 2025

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to the Institute of Engineering, Thapathali Campus, for providing us with the opportunity to enhance our knowledge and understanding of C programming. We are also extremely thankful to our teachers and supervisors, Prajwol Pakka and Anup Shrestha and also the Department of Electronics and Computer Engineering at Thapathali Campus for their invaluable guidance, ideas, and support that helped us successfully carry out this project.

Bidhan Shrestha (THA081BEI009)
Kabindra Panta (THA081BEI012)
Nirajan Chaudhary (THA081BEI022**)**
Shuvam Neupane (THA081BEI043)

ABSTRACT

This project aims to develop a Bank Management System using the C programming language. In analyzing current banking systems, it is clear that they often require complex operations such as account management, fund transfers, and transaction tracking. However, many of these processes are either not fully automated or can be inefficient, especially when done manually or with outdated technology. This project seeks to address these issues by creating a system that simplifies and automates common banking tasks, such as user registration, account creation, fund deposits, withdrawals, and money transfers, while maintaining a user-friendly and secure environment. The system will also allow users to view their transaction history and display account details, ensuring a smooth banking experience. With this project, we aim to enhance the overall functionality and efficiency of the banking system, creating a modern platform for managing personal finances.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

CBS	Core Banking Systems
GUI	Graphical User Interface
BMS	Bank Management System

1. INTRODUCTION

1.1 Background

Banking is an essential part of everyday life, allowing individuals and businesses to manage their finances efficiently. Traditional banking systems require a lot of manual work, which can be time-consuming and prone to errors. With the advancement of technology, computerized banking systems have made financial transactions faster, more secure, and more convenient.

This project is about a Bank Management System developed in C programming language, designed to perform essential banking operations such as creating accounts, depositing funds, withdrawing money, transferring funds, and viewing transaction history. The system uses structured programming and data handling techniques to ensure smooth and error-free transactions. It stores account details in arrays and maintains transaction records for better financial tracking.

The main goal of this project is to provide a simple and efficient banking solution that can be used in small financial institutions or as a foundation for more advanced banking applications. This system can be further enhanced by integrating file storage for permanent data saving, security features for data protection, and a graphical user interface for better user experience. By implementing such a system, we can reduce human effort, minimize errors, and improve the overall efficiency of banking operations.

Features	Traditional Banking System	Modern Banking System
Accessibility	Customers must visit the bank for transactions	Accessible anytime, anywhere through online banking and mobile apps

Transaction Speed	Slow, manual processing of transactions	Fast and automated transactions with minimal human effort
Security	Relies on physical records, prone to fraud and errors.	Uses encryption, biometrics, and multi-factor authentication for security
Record Keeping	Paper-based documentation, difficult to manage	Digital record-keeping, making data storage and retrieval easy
Customer Service	Face-to-face interactions, limited support hours	24/7 customer support through chatbots, emails, and call centers
Cost Efficiency	High operational costs due to manual processes	Lower costs as automation reduces the need for manual labor.
Transaction Methods	Cash and cheque-based transactions	Digital payments, online transfers, UPI, credit/debit cards, and mobile wallets
Loan and Credit Processing	Lengthy approval process with extensive paperwork	Quick approval through AI-based credit analysis and digital verification
Fraud Detection	Hard to detect fraud quickly	AI and machine learning help in real-time fraud detection and prevention
Integration with Technology	Limited use of technology, mostly manual operations	Highly integrated with technology, using AI blockchain, and cloud computing

Table 1: Difference between Traditional and Modern Banking Management System

1.2. Motivation

This Bank Management System (BMS) is our first project, and it marks the beginning of our journey into software development. After learning the basics of the C programming language, we wanted to apply our skills to build a practical and useful application. The idea of developing a BMS came from our desire to create a system that could automate and simplify everyday banking tasks such as account creation, deposits, withdrawals, and fund transfers. We realized that with the growing reliance on digital banking, automating these processes is crucial to ensuring efficiency, security, and accuracy.

We chose C for this project because it allows us to have control over system resources and ensures that the program runs efficiently. It also gives us the flexibility to work on essential banking features, while offering ease of learning and readability, which is important for a first project. As this is our first attempt at creating a real-world application, we aim to build a secure and user-friendly system that will lay a strong foundation for future, more complex projects. The process of developing this BMS will help us learn the practical aspects of programming, including designing a system, handling data, and ensuring smooth interactions with users. This project serves as a great starting point for us to gain experience in software development while creating a useful tool for managing bank accounts and transactions.

1.3 Project Objectives

- ➢ To enhance our understanding of C programming by applying it to real-world banking tasks, improving our skills in areas like file handling, memory management, and data structure implementation.

- ➢ To implement secure banking features such as account creation, fund transfers, deposits, and withdrawals, ensuring safe and reliable transactions within the system.

1.4 Project Scope and Applications

The Bank Management System (BMS) project is designed to automate key banking operations, ensuring a more efficient and secure way to manage bank accounts and transactions. The system aims to make banking tasks like account creation, deposits, withdrawals, and fund transfers accessible to both customers and bank staff through a simple, user-friendly interface.

For data storage and transaction management, the system utilizes file handling capabilities in C, ensuring secure and efficient record-keeping. The key factors leading to the development of this system in C are:

i. Faster Execution and Memory Management
ii. Efficient Handling of Banking Operations
iii. Security and Data Integrity
iv. Scalability and Future Enhancements

2. LITERATURE REVIEW

2.1 Bank management system

Banking systems have changed a lot, especially with new technologies. In the past, banks used manual systems to manage accounts, but today, core banking systems (CBS) allow banks to centralize account management. This means that customers can access their accounts from different branches or even online. With the rise of mobile and internet banking, customers now have 24/7 access to perform transactions, check their balances, and manage their finances. Technologies like cloud computing, real-time transaction processing, and distributed databases help make these systems efficient. However, the security of online banking is still a big concern. Banks use encryption, two-factor authentication, and biometric verification to protect customers' data. While these systems make banking easier and more accessible, there are still challenges like cyber-attacks, system failures, and the lack of internet access in rural areas. These problems highlight the need for better and safer banking systems that can be used by more people and provide better security. This motivates further development of banking systems that are more secure, reliable, and accessible.

2.2 C language

In parallel, the implementation of banking systems using C programming has been a common approach, especially for small to medium-sized applications that prioritize performance. C's efficiency in handling memory and system resources makes it suitable for applications like banking systems that require fast, real-time transaction processing. In C, data structures such as arrays and linked lists are used to manage banking data, with structures like Account and Transaction storing critical information about users and their financial activities. C-based banking systems often incorporate file handling techniques to persist data and maintain transaction logs, providing an audit trail for financial activities. However, C programming has limitations when it comes to scalability and security. For instance, while C provides fine control over memory management, this also leads to potential issues like memory leaks and buffer overflows, which can cause system crashes or data corruption. Furthermore, C is not well-suited for developing graphical user

interfaces (GUIs), making these systems less user-friendly compared to modern banking applications that feature web and mobile interfaces. Additionally, the lack of built-in support for advanced features such as cloud integration or artificial intelligence in C programming limits the scope of banking systems built solely with this language. Despite these drawbacks, C's efficiency and control over system resources have made it a useful tool for developing the core functionalities of banking systems. However, the scalability and security issues inherent in C-based systems motivate the exploration of more robust technologies and platforms for building large-scale, secure, and user-friendly banking systems. These criticisms underscore the need for ongoing development in banking systems that can handle a larger volume of transactions and offer enhanced security and user experience.

3.PROPOSED SYSTEM ARCHITECTURE

3.1 Login Interface

This ER diagram outlines the process of entering a bank system, showing different paths for admins and customers. It starts with the user entering the bank, then proceeds to login. For customers, it checks if they have an account. If they do, they log in; if not, they register by providing their name, address, phone, and password. The login process involves entering account details and password, which are then verified against the database. If the information matches, the user is taken to the home page; if not, they receive an "invalid info" message. In essence, it guides the user through either logging in or registering to access the bank system.

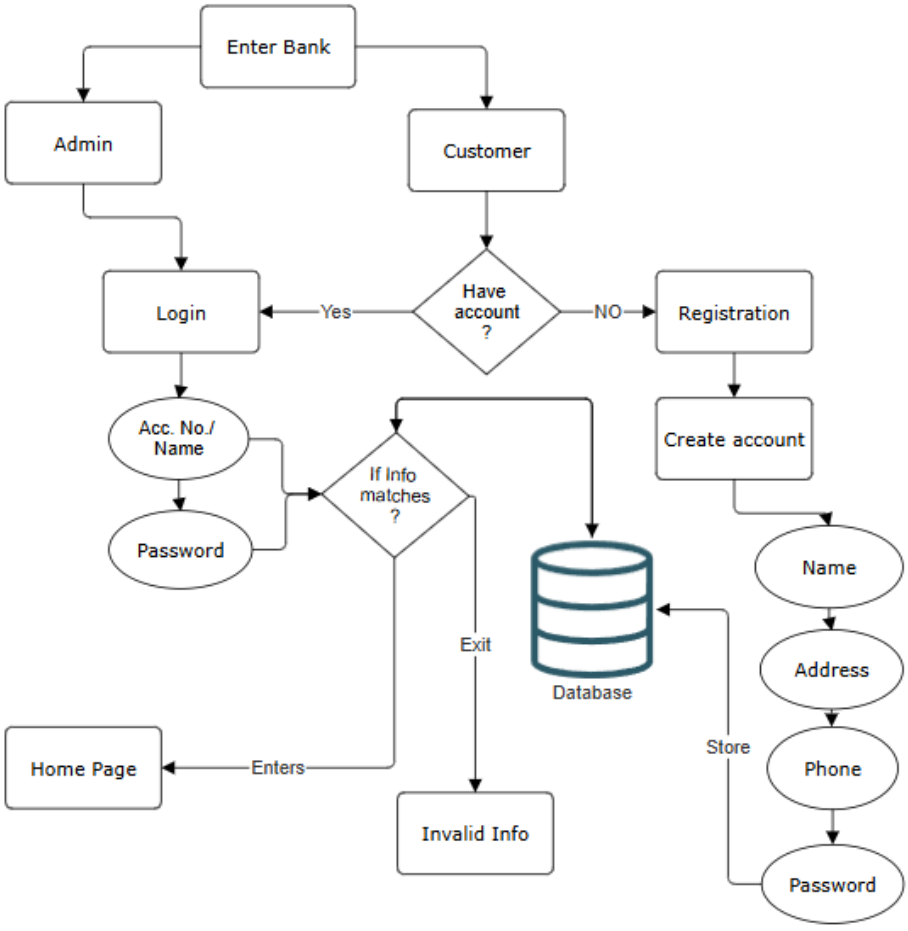

Figure 1: Login Interface of Bank Management System

3.2 Customer Options

The ER diagram presented below shows how users navigate a banking application. It starts at the "Home Page," leading to the "Menu." From there, users can choose to "Deposit Funds," "Withdraw Funds," "Transfer Funds," or check "Balance Inquiry." Each action interacts with a "Database" to process transactions, illustrating the connection between user actions and backend database functions.

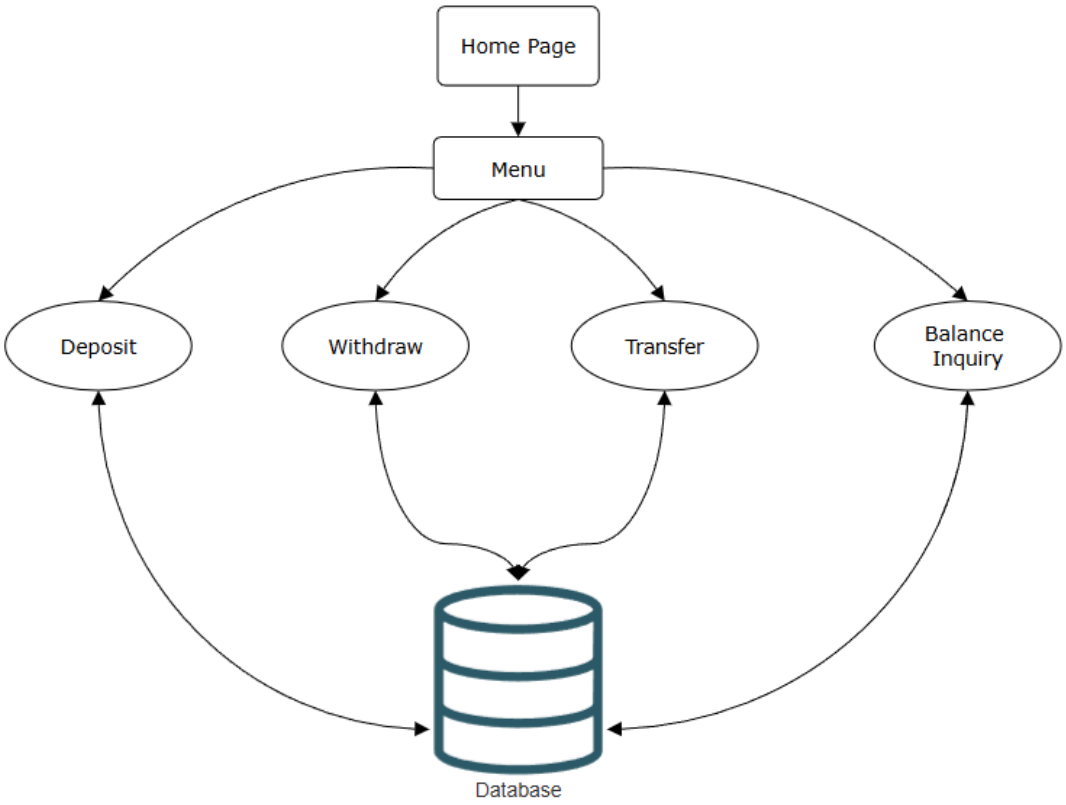

Figure 2: Customer Menu Session

4. METHODOLOGY

The Bank Management System developed in C will simulate a simple bank's core functionality for account management, transactions, and user authentication. This methodology will describe the processes involved in implementing the system, including the system's design, key components, data structures, and testing procedures.

4.1. System Overview

The Bank Management System in C will include the following core functions:

- Login: To authenticate the user with credentials.
- RegisterUser: To register new users into the system.
- CreateAccount: To create a new bank account for users.
- DeleteAccount: To delete an existing bank account.
- UpdateAccount: To modify the details of a bank account.
- DisplayAccount: To display account information.
- DepositFunds: To deposit money into an account.
- WithdrawFunds: To withdraw money from an account.
- TransferFunds: To transfer money from one account to another.

4.2. Algorithm

4.2.1. Login Algorithm

- Step 1: Prompt the user to enter their Acc.No and password.
- Step 2: Verify the entered credentials by checking against the stored data.
- Step 3: If the credentials are correct, display "Login successful" and proceed to the main menu.
- Step 4: If the credentials are incorrect, display "login failed" and prompt the user to retry login.

4.2.2. Register User Algorithm

- Step 1: Prompt the user to enter a new username, password, phone, and Password.
- Step 2: Check if the entered username is unique by comparing it with existing usernames.
- Step 3: If the username is unique, store the user's details and display "User registered successfully."
- Step 4: If the username is already taken, display "You've already registered with us" and ask the user to choose a different one.

4.2.3. Create Account Algorithm

- Step 1: Prompt the user to enter their name, address, phone number.
- Step 2: Verify if the entered account type is valid (i.e., it should match one of the predefined types).
- Step 3: If the account no. is valid, create a new account using the provided details and store the information in the database
- Step 4: If the account no. is invalid, display the message "Invalid account no." and prompt the user to try again.

4.2.4. Deposit Funds Algorithm

- Step 1: Prompt the user to enter the account number and the deposit amount.
- Step 2: Verify that the deposit amount is greater than 0.
- Step 3: If the deposit amount is valid, update the account balance and record the transaction as a deposit.
- Step 4: Display "Deposit successful."
- Step 5: If the deposit amount is invalid (e.g., zero or negative), display "Invalid deposit amount" and prompt the user to enter a valid amount.

4.3. Preliminary Process Modeling

The process of managing user and account data will follow these steps:

1. User Authentication: Users must log in with a valid username and password.
2. Account Management: Users can create, update, and delete their bank accounts.
3. Transaction Management: Users can perform deposits, withdrawals, and fund transfers.
4. Transaction History: The system records all transactions, which can be viewed by the user.

The database model will store:

- User Table: userID, username, password, Address, phone
- Account Table: accountNumber, userID, accountType, balance
- Transaction Table: transactionID, accountNumber, date, amount

4.4. Data Modeling

The Bank Management System will use simple data structures to model user and account information:

User Data Structure:

```
struct User
{
   char username[50];
   char password[50];
   char fullName[100];
   char email[100];
};
```

Account Data Structure:

```
struct Account
{
   int accountNumber;
   char accountType[20];
   float balance;
};
```

Transaction Data Structure:

```
struct Transaction
{
   int transactionID;
   int accountNumber;
   float amount;
   char transactionType[20];
   char date[20];
};
```

4.5. Database Design

The system will use file handling to simulate a database, saving and reading user, account, and transaction data in separate files.

- users.dat: Stores user information (username, password, etc.).
- accounts.dat: Stores account details (account number, account type, balance).
- transactions.dat: Stores transaction records (transaction ID, amount, type).

4.6. Interface Design

The system interface will be text-based, providing the following screens:

- Login Screen: Prompt the user for username and password.
- Main Menu: Options for creating an account, viewing accounts, and performing transactions.
- Transaction Screen: Options to deposit, withdraw, transfer funds.
- Error Handling: Display appropriate error messages for invalid actions.

4.7. Performance Parameters

The following performance parameters will be measured during the testing phase:

- Response Time: Time taken by the system to process a user's request.
- System Reliability: System's ability to handle unexpected input without crashing.
- Data Integrity: Ensuring transactions and account details are accurately updated,

4.8. Testing

Testing will be carried out in stages:

1. Unit Testing: Test each function (e.g., login, create account) individually to ensure it works as expected.
2. Integration Testing: Test how functions interact with each other (e.g., checking if account creation updates the account list).
3. System Testing: Test the entire system for performance, load handling, and bug identification.
4. Acceptance Testing: Ensure the system meets the specified requirements.

Each test will be executed under controlled conditions and the results will be documented for further analysis.

5.TIME ESTIMATION

Table 2: Gantt Chart with Project Activities and Timeline

6.FEASIBILITY ANALYSIS

6.1 Technical Feasibility

The project is technically feasible as it is developed using the C programming language, which is efficient for handling structured data and real-time processing. The system utilizes basic data structures such as arrays and structures to manage accounts and transactions. The required hardware and software, such as a computer with a C compiler, are easily available. However, the system lacks a graphical user interface (GUI) and advanced security features, making it suitable for small-scale banking operations rather than large-scale enterprise systems.

6.2 Economic Feasibility

The cost of developing this project is minimal since it only requires basic computing resources and text editors like VS- Code, Code Blocks etc. Unlike commercial banking software, there are no licensing fees, making it cost-effective. However, for large-scale deployment, additional investment may be needed for security improvements, cloud integration, and database management systems.

6.3 Operational Feasibility

The system is easy to use and allows users to do basic banking tasks like creating accounts, depositing money, withdrawing money, and transferring funds. Since it runs on a command-line interface/terminal, users do not need to know about complex commands. While the system works well, it may not be easy to use for people who are used to modern banking apps with buttons and menus or basically GUIs.

References

[1] A. S. Tanenbaum and H. Bos, "Modern Operating Systems," 4th ed., Pearson, 2014.

[2] J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach," 8th ed., Pearson, 2021.

[3] R. S. Pressman and B. R. Maxim, "Software Engineering: A Practitioner's Approach," 9th ed., McGraw-Hill, 2020.

[4] W. Stallings, "Cryptography and Network Security: Principles and Practice," 7th ed., Pearson, 2017.

[5] GNU Compiler Collection (GCC), "Tools and Libraries for C Project," Available at: https://gcc.gnu.org/