"Bank Management System"

Dept. of Computer Science and Engineering

A Project on Banking Base using Object Oriented Programming.

Submitted 27th November, 2025

for BSc. Engg. in C.S.E.

Submitted to: Humayra Ferdous(Lecturer)

Dept. of C.S.E. BUBT

Submitted By:

Luckman Rafique (20255103042)

Md. Taimur Rahman (20255103046)

Md. Yousuf Ali (20255103059)

Md. Adnan Sami (20255103068)

Md. Siam Khan (20255103069)

Kazi Yeaj Morshed Rizve (20255103074)

Abstract:

This project implements a console-based banking system in C++ that allows users to securely create and manage bank accounts through a menu-driven interface. The system supports core banking operations such as account creation with input validation, deposits, withdrawals protected by a numeric password, balance inquiries, viewing detailed account information, and displaying transaction histories with timestamps. Data persistence is achieved using text files to store account records, a global account counter, and an appended transaction log, ensuring that information is retained across program executions. The use of object-oriented design with separate BankAccount and BankingSystem classes, along with file handling and basic input masking for passwords, demonstrates the practical application of C++ concepts to simulate a simplified banking environment suitable for academic and learning purposes.

Contents:

1. Introduction

1.1. Objective

- To design and implement a console-based banking system in C++ that performs core banking operations such as account creation, deposit, withdrawal, and balance inquiry.
- To apply object-oriented programming concepts using classes and objects for clear separation of account and system logic.
- To implement persistent storage of account data and transaction history using file handling so that data is retained across multiple runs of the program.
- To incorporate basic security and validation through password-protected withdrawals and input checks for user details and transaction amounts.

1.2. Project Scope

- The system supports creating Savings and Current accounts with defined minimum balance requirements and an initial deposit at account opening.
- Users can perform deposits, withdrawals, balance checks, account detail viewing, and transaction history viewing through a menu-driven console interface.
- All account information, transaction logs, and the account number counter are stored in external text files to maintain records over time.
- An admin-only option is provided to display all existing accounts with key details such as account number, holder name, type, and balance.

1.3. Our Contributions

- Designed the overall system architecture and implemented the BankAccount and BankingSystem classes to organize functionalities logically.
- Implemented secure withdrawal using a 4-digit password with masked input, along with validation for name, phone number, email, and transaction amounts.
- Added automatic account number generation and persistent storage of accounts, transaction history, and account counter using file handling.
- Developed user-friendly console menus and messages to guide users through banking operations and error handling.

1.4. Conclusions

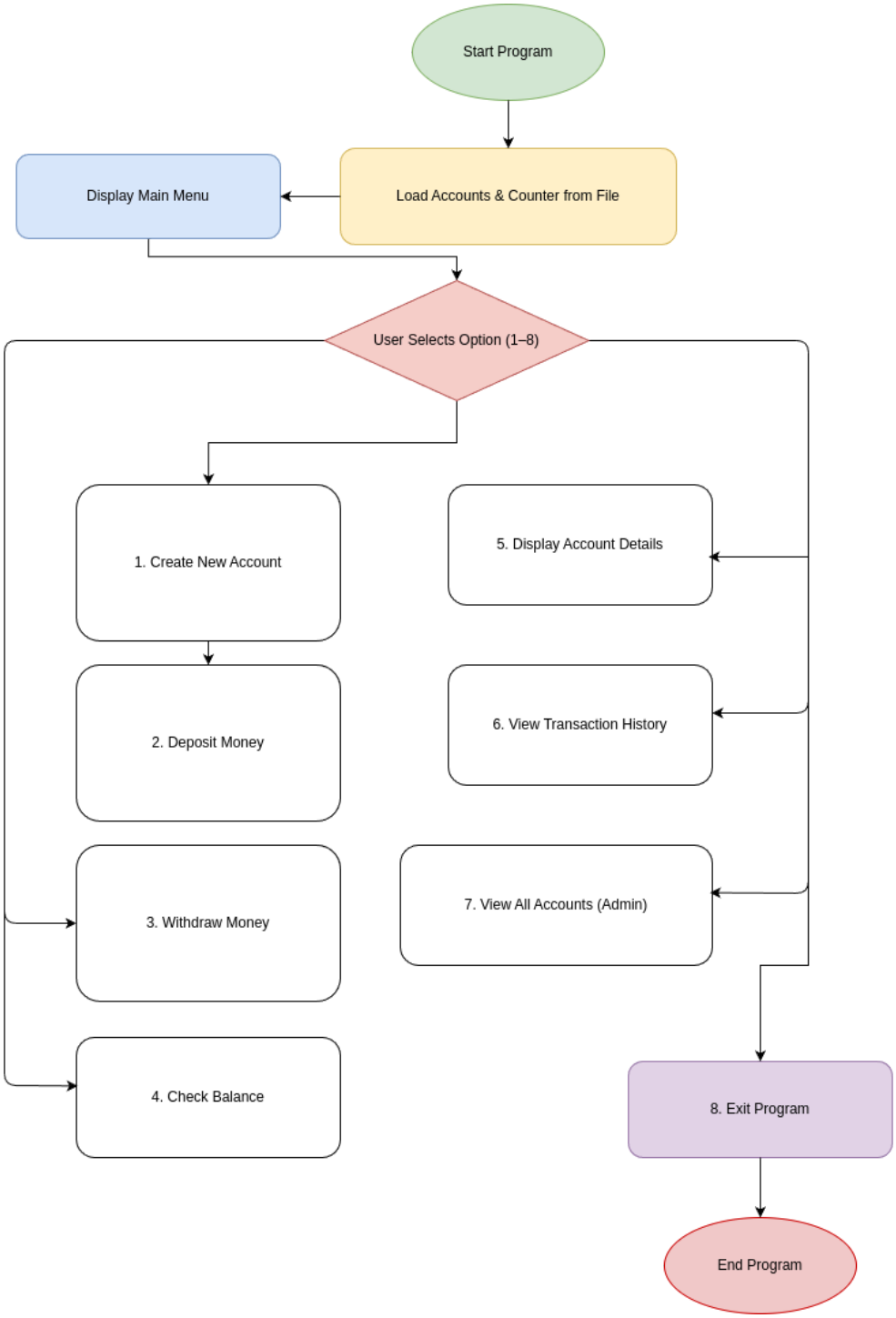
This C++ banking system project successfully achieves its aim of simulating core banking operations in a simple, console-based environment while reinforcing key programming concepts. Through features such as account creation, deposits, withdrawals with password protection, balance inquiries, and transaction history tracking, the system demonstrates practical use of object-oriented design, file handling, and input validation. It provides a clear structure with separate classes for accounts and overall management, ensuring modularity and easier future extension. Overall, the project meets its objectives and offers a solid foundation that can be enhanced later with additional services, graphical interfaces, database integration, and stronger security mechanisms.

2.Proposed Model

2.1. Software Requirement

- A C++ compiler that supports at least the C++11 standard (such as GCC, MinGW, Clang, or MSVC).
- An Integrated Development Environment (IDE) or code editor like Code::Blocks, Dev-C++, Visual Studio, or VS Code with C++ extensions for writing, compiling, and debugging the code.
- An operating system such as Windows, Linux, or macOS that can execute console-based C++ applications and supports standard file I/O operations.
- A standard C++ runtime library and basic command-line terminal or
- console to interact with the menu-driven interface and handle file-based data storage.

2.2. Diagram of the full project



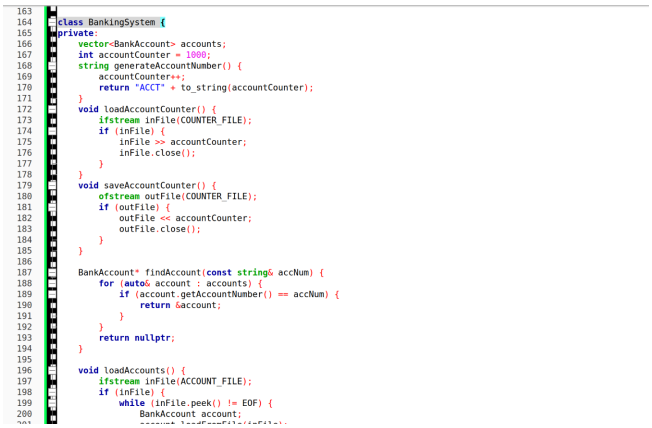
3. Implementation of Our System

3.1. Introduction

The implementation of our system is a C++ console-based banking application built using object-oriented programming and file handling. It uses two main classes, BankAccount and BankingSystem, to manage account data, perform core operations (create, deposit, withdraw, check balance, view history), and interact with users through a menu-driven interface.

3.2. Overview

```
const double CURRENT_MIN_BALANCE = 500.0;

class BankAccount {
private:
    string accountNumber;
    string accountHolderName;
    string address;
    string phoneNumber;
    string email;
    double balance;
    string accountType;
    string password;
    vector<string> transactionHistory;

public:
    BankAccount(string accNum = "", string name = "", string addr = "",
                string phone = "", string mail = "", double initialDeposit = 0.0,
                string type = "Savings", string pwd = "1234")
        : accountNumber(accNum), accountHolderName(name), address(addr),
          phoneNumber(phone), email(mail), balance(initialDeposit),
          accountType(type), password(pwd) {
        if (initialDeposit > 0) {
            addTransaction("Account opened with initial deposit: " + to_string(initialDeposit) + " BDT");
        }
    }


    string getAccountNumber() const { return accountNumber; }
    string getAccountHolderName() const { return accountHolderName; }
    double getBalance() const { return balance; }
    string getAccountType() const { return accountType; }
    string getPassword() const { return password; }
    const vector<string>& getTransactionHistory() const { return transactionHistory; }

    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            addTransaction("Deposit: +" + to_string(amount) + " BDT");
            cout << "Deposit successful. New balance: " << fixed << setprecision(2) << balance << " BDT" << endl;
```

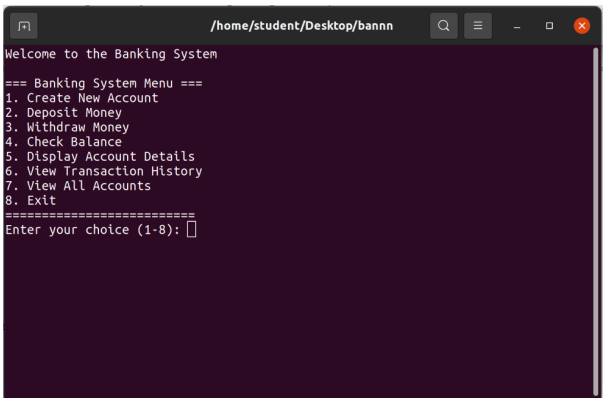


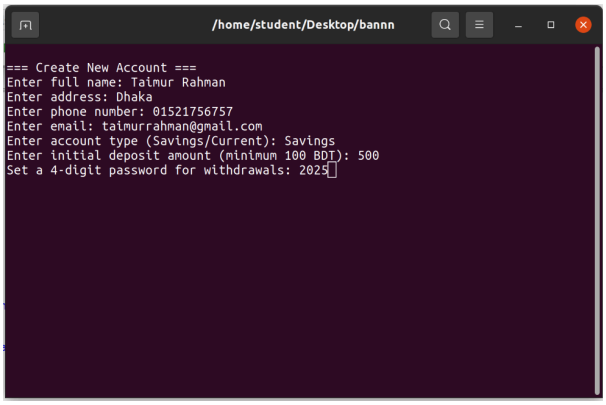
```
class BankingSystem {
private:
    vector<BankAccount> accounts;
    int accountCounter = 1000;
    string generateAccountNumber() {
        accountCounter++;
        return "ACCT" + to_string(accountCounter);
    }
    void loadAccountCounter() {
        ifstream inFile(COUNTER_FILE);
        if (inFile) {
            inFile >> accountCounter;
            inFile.close();
        }
    }
    void saveAccountCounter() {
        ofstream outFile(COUNTER_FILE);
        if (outFile) {
            outFile << accountCounter;
            outFile.close();
        }
    }

    BankAccount* findAccount(const string& accNum) {
        for (auto& account : accounts) {
            if (account.getAccountNumber() == accNum) {
                return &account;
            }
        }
        return nullptr;
    }

    void loadAccounts() {
        ifstream inFile(ACCOUNT_FILE);
        if (inFile) {
            while (inFile.peek() != EOF) {
                BankAccount account;
```

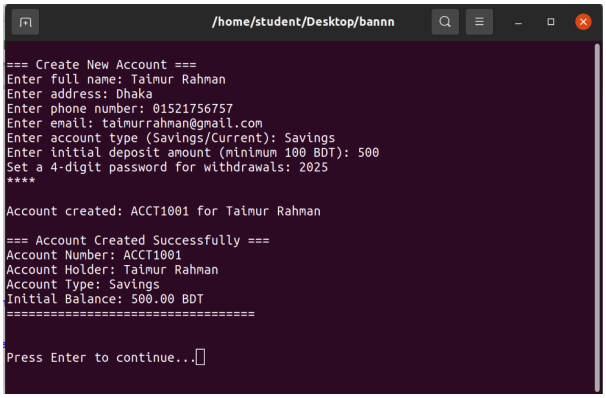
4. Experimental Results and Evaluation

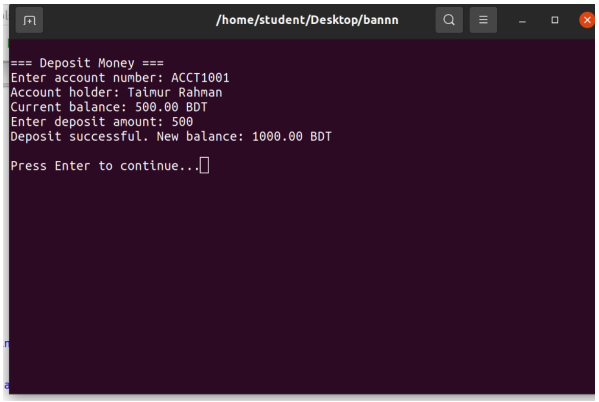
4.1. Result Analysis

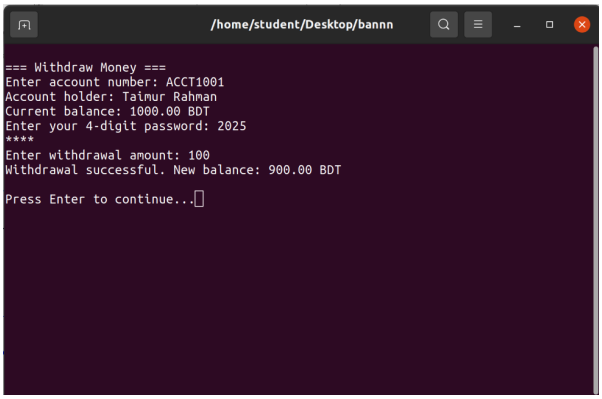
Some screenshots of our program output are shown below in order.

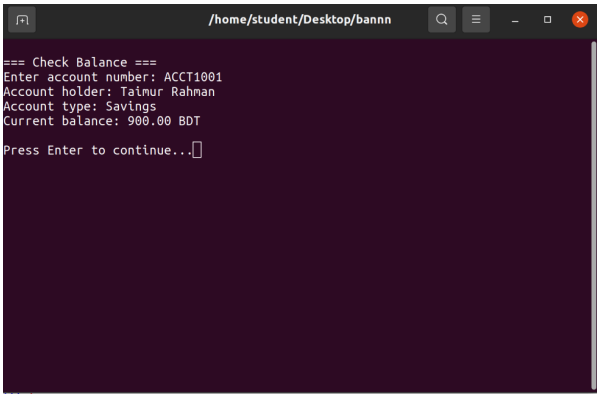


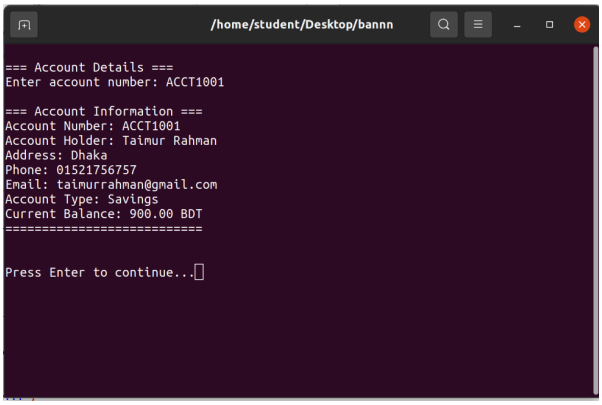



/home/student/Desktop/bannn
=== Create New Account ===
Enter full name: Taimur Rahman
Enter address: Dhaka
Enter phone number: 01521756757
Enter email: taimurrahman@gmail.com
Enter account type (Savings/Current): Savings
Enter initial deposit amount (minimum 100 BDT): 500
Set a 4-digit password for withdrawals: 2025
****
Account created: ACCT1001 for Taimur Rahman
=== Account Created Successfully ===
Account Number: ACCT1001
Account Holder: Taimur Rahman
Account Type: Savings
Initial Balance: 500.00 BDT
==========
Press Enter to continue...


/home/student/Desktop/bannn
=== Deposit Money ===
Enter account number: ACCT1001
Account holder: Taimur Rahman
Current balance: 500.00 BDT
Enter deposit amount: 500
Deposit successful. New balance: 1000.00 BDT
Press Enter to continue...


/home/student/Desktop/bannn
=== Withdraw Money ===
Enter account number: ACCT1001
Account holder: Taimur Rahman
Current balance: 1000.00 BDT
Enter your 4-digit password: 2025
****
Enter withdrawal amount: 100
Withdrawal successful. New balance: 900.00 BDT
Press Enter to continue...





4.2. Evaluation

The developed banking system successfully performs core operations such as account creation, deposit, withdrawal, balance inquiry, and transaction history viewing through a simple menu-driven interface. It uses file handling for data persistence and object-oriented design for better structure, but still has limitations like a text-based UI and basic security, leaving room for future improvements such as a GUI, stronger authentication, and database integration.

5.Future Scope and Limitations

5.1. Future Scope of Our Project

- Add a graphical user interface (GUI) to make the system easier and more attractive to use.
- Use a database instead of text files to improve security, speed, and scalability.
- Introduce extra features such as fund transfer, interest calculation, and account closing.
- Implement stronger security with encrypted passwords, roles (admin/user), and multi-user support.

5.2. Limitation

- Console-based interface only; no GUI.
- Uses plain text files, which are less secure and less efficient than databases.
- Single-user, offline system with no real-time or network/online access.
- Supports only basic banking features and simple password protection.

5.3. Conclusion

- The project successfully demonstrates a basic banking system using C++ and object-oriented programming.
- It meets its main goals: creating accounts, handling deposits and withdrawals, checking balances, and storing transaction history.
- Although limited in UI, security, and scalability, it forms a strong base for future upgrades and learning.