St. Mary University

Programming Fundamentals I

Bank management system

Section:B

Group 5

Group Members	ID No
1.Aliya Shemsu	RCD/0173/2017
2.Dawit Ftwi	RCD/1819/2017
3.Kalkidan Belete	RCD/0195/2017
4.Kalkidan Solomon	RCD/0195/2017
5.Natnael Agibu	RCD/0213/2017
6.Sirgut Serke	RCD/0925/2017
7.Temesgen Tilahun	RCD/0219/2017
8.Yohana Teklu	RCD/0195/2017

Submitted to:Mr.Dawit

Submission date: July 9,2025

A Bank Management System in C++ is a console-based application that simulates basic banking operations using structured programming and object-oriented principles. It’s a great project for mastering modularization, file handling, and class design.

Overview

Bank Management System typically includes:

• User Interface: Menu-driven console interface for interaction.

• Account Management: Create, view, update, and delete bank accounts.

• Transaction Handling: Deposit, withdraw, and transfer funds.

• Data Persistence: Use of file handling or databases to store user and transaction data. •

Security Features: Basic login authentication using usernames and passwords or PINs.

•Open Account-Collects user details and initializes account with a starting balance.

•Deposit Money-Adds a specified amount to the account balance.

•Withdraw Money-Deducts funds from the account if sufficient balance exists.

•Check Balance-Displays current account balance with formatted output.

•Close Account-Deletes user data and deactivates the account.

•Login System-Authenticates users before allowing access to their account

•BankAccount-Manages individual account operations like deposit, withdraw, and summary.

Functionality

•createAccount()-Initializes a new account by collecting the user’s name and initial balance.

•loadAccountFromFile()-Reads previously saved account details from a file, allowing the user to resume use.

•saveAccountToFile()-Writes current account data (like name and balance) to a file for persistence.

•showBankingMenu()-Displays the main banking options: balance check, deposit, withdrawal, etc.

•showBalance()-Outputs the current account balance in a readable format.

•deposit()-Adds funds to the account and updates the balance.

•withdraw()-Deducts funds from the account if sufficient balance is available.

•initialChoice-Stores the user’s choice from the main menu: Login, Register, or Exit.

•cin.fail()-Checks if the user entered invalid input (like a letter instead of a number), then resets input state.

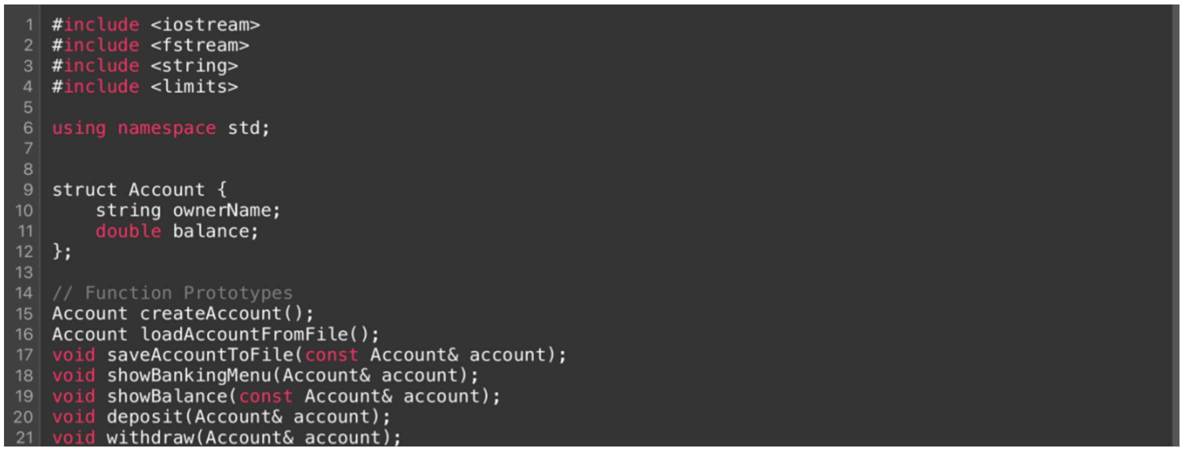
•Account myAccount;-Declares a temporary object to hold account info during the session.

•loadAccountFromFile()-Attempts to read account data from a file and return it. Used for login.

•createAccount()-Collects user input to generate a new account. Used during registration.

•saveAccountToFile()-Persists newly created account details to a file so it can be retrieved later.

•showBankingMenu()-Launches the banking operations menu (e.g., check balance, deposit, withdraw) for the logged-in user.



```
#include <iostream>
#include <fstream>
#include <string>
#include <limits>

using namespace std;


struct Account {
    string ownerName;
    double balance;
};

// Function Prototypes
Account createAccount();
Account loadAccountFromFile();
void saveAccountToFile(const Account& account);
void showBankingMenu(Account& account);
void showBalance(const Account& account);
void deposit(Account& account);
void withdraw(Account& account);
```

•myAccount.ownerName.empty()-Checks if login failed due to missing or empty account data.

•1.Login - Attempts to load and use an existing account.

•2.Register - Creates and saves a new account, then opens the banking menu.

•3.Exit -Ends the program and displays a farewell message.

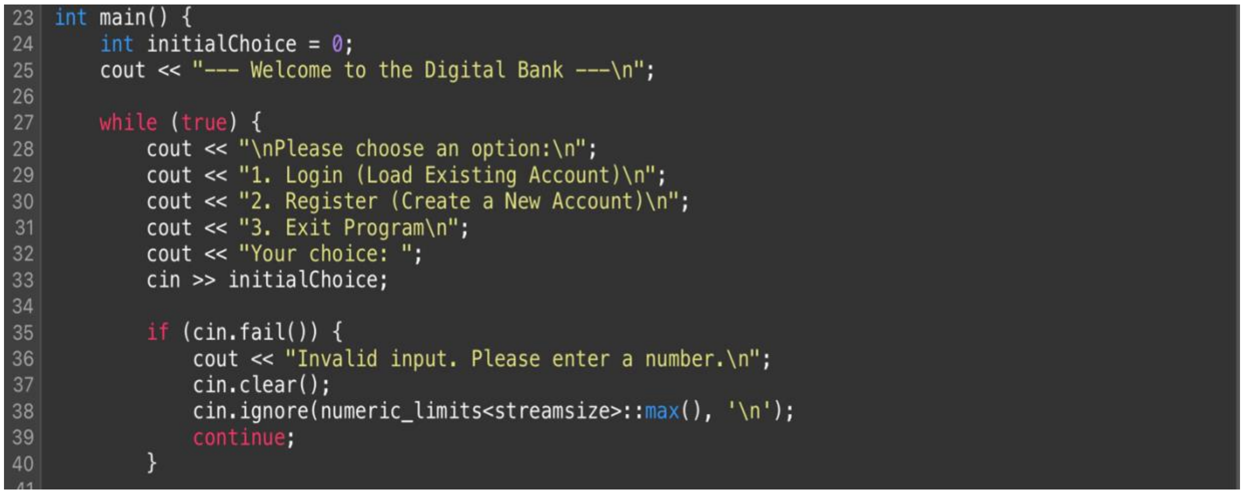
•Invalid input-Triggers error message and loops again.

•do { ... } while (choice != 4); Keeps showing the banking menu until the user selects Logout (option 4).

•cout statements-Displays a friendly menu with options to show balance, deposit, withdraw, or logout.

•cin >> choice-Takes user input to determine which operation to execute.

•cin.fail() check-Detects invalid input (e.g., letters instead of numbers) and resets the input stream.



```
int main() {
    int initialChoice = 0;
    cout << "--- Welcome to the Digital Bank ---\n";

    while (true) {
        cout << "\nPlease choose an option:\n";
        cout << "1. Login (Load Existing Account)\n";
        cout << "2. Register (Create a New Account)\n";
        cout << "3. Exit Program\n";
        cout << "Your choice: ";
        cin >> initialChoice;

        if (cin.fail()) {
            cout << "Invalid input. Please enter a number.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
```

•switch(choice)-Directs the flow based on user's input—running the selected banking operation.

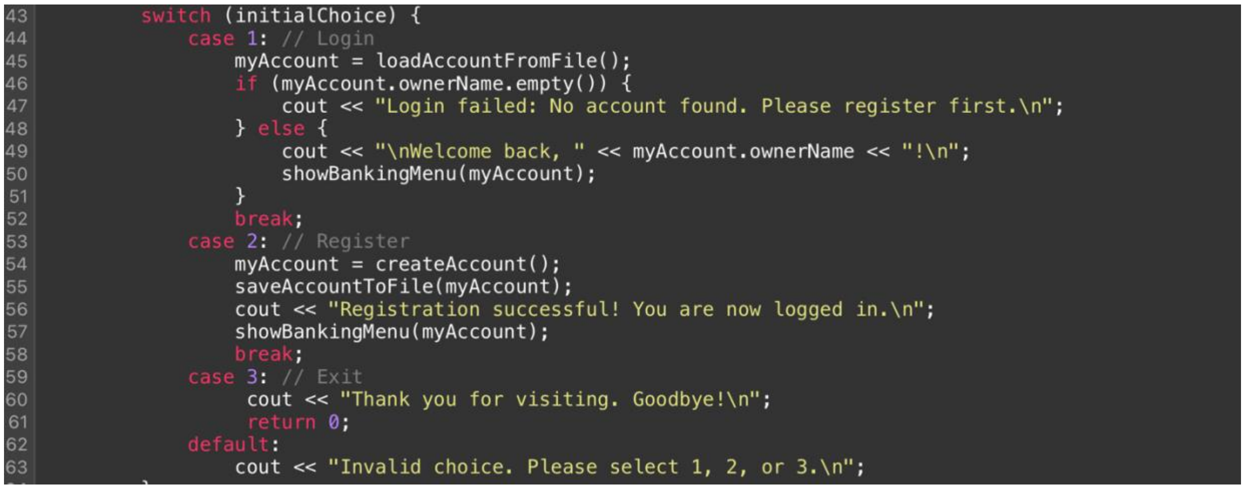
•showBalance(account)-Displays the account's current balance with proper formatting.

•deposit(account)-Adds money to the user's account.

•withdraw(account)-Subtracts money from the account (after validating sufficient balance).

•saveAccountToFile(account)-Saves the account data upon logout to ensure persistence for future sessions.

•default case -Handles invalid menu choices and prompts the user to try again.



```
        switch (initialChoice) {
            case 1: // Login
                myAccount = loadAccountFromFile();
                if (myAccount.ownerName.empty()) {
                    cout << "Login failed: No account found. Please register first.\n";
                } else {
                    cout << "\nWelcome back, " << myAccount.ownerName << "!\n";
                    showBankingMenu(myAccount);
                }
                break;
            case 2: // Register
                myAccount = createAccount();
                saveAccountToFile(myAccount);
                cout << "Registration successful! You are now logged in.\n";
                showBankingMenu(myAccount);
                break;
            case 3: // Exit
                 cout << "Thank you for visiting. Goodbye!\n";
                 return 0;
            default:
                cout << "Invalid choice. Please select 1, 2, or 3.\n";
```

•Account createAccount()` Function

Creates an `Account` object named `newAccount`.

Prompts the user for their full name, using `getline` to allow multi-word names.

Asks for an initial deposit, validating that the input is a positive number.

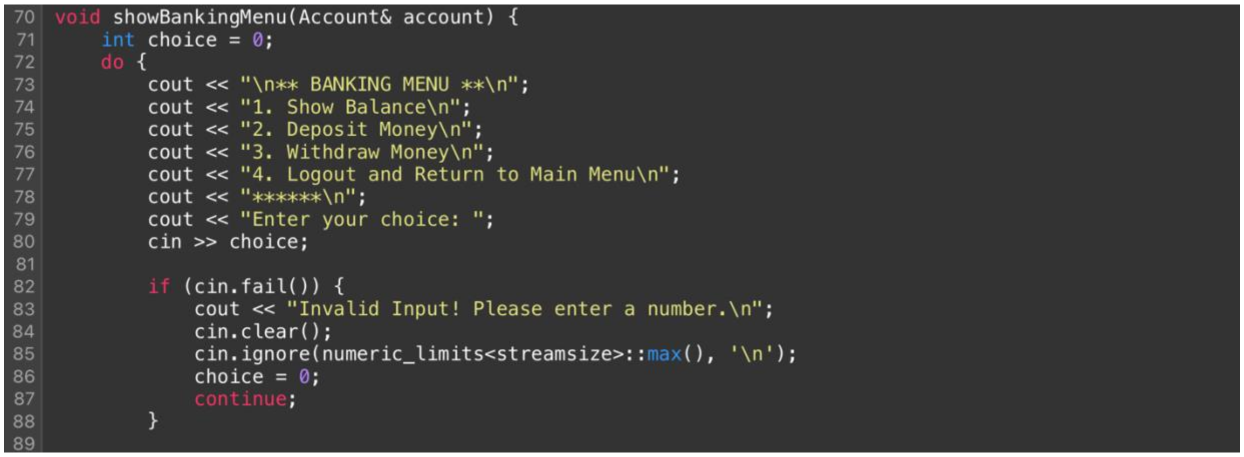
Uses `cin.ignore` and `cin.clear` to handle bad input gracefully.

Returns the fully initialized account.

•showBalance(const Account& account) Function:

`fixed` and `precision(2)` for consistent formatting (e.g., showing `1234.50`).

`account.ownerName` and `account.balance` fields to personalize the output.



```
void showBankingMenu(Account& account) {
    int choice = 0;
    do {
        cout << "\n** BANKING MENU **\n";
        cout << "1. Show Balance\n";
        cout << "2. Deposit Money\n";
        cout << "3. Withdraw Money\n";
        cout << "4. Logout and Return to Main Menu\n";
        cout << "*****\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (cin.fail()) {
            cout << "Invalid Input! Please enter a number.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            choice = 0;
            continue;
        }

```

•deposit(Account& account)

Purpose: Adds funds to the user's account.

Logic:Prompts for an amount.

Checks if the amount is positive.

Adds the valid amount to `account.balance`.

Displays a warning if the entered amount is invalid.

•withdraw(Account& account)`

Purpose: Removes funds from the account, ensuring safe withdrawal.

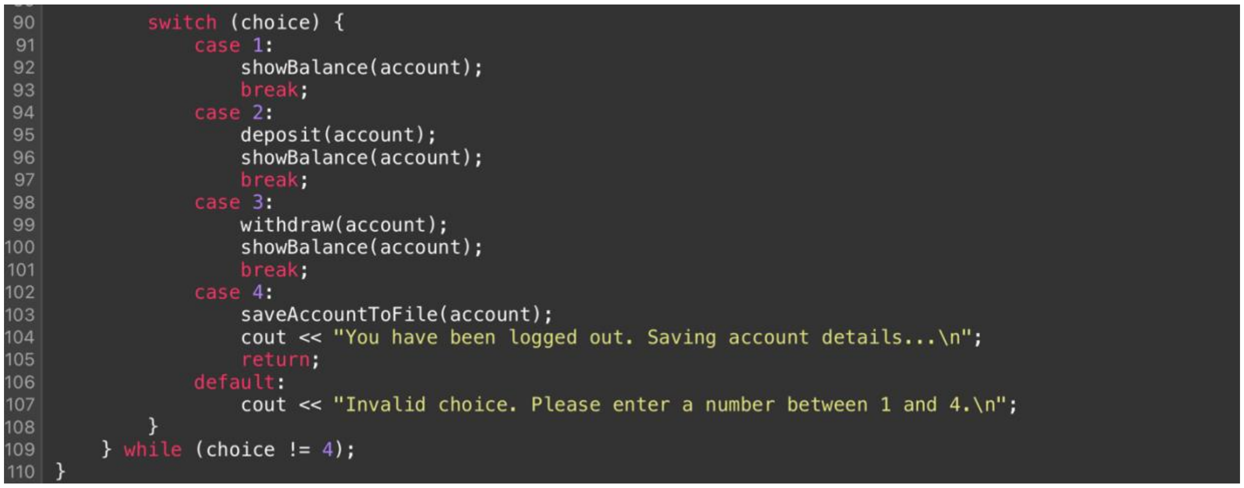
Logic: Prompts for withdrawal amount.

Validates that the amount is Positive.

Less than or equal to `account.balance`.

Subtracts the valid amount.

If invalid, displays appropriate messages (insufficient funds or negative amount)



```
        switch (choice) {
            case 1:
                showBalance(account);
                break;
            case 2:
                deposit(account);
                showBalance(account);
                break;
            case 3:
                withdraw(account);
                showBalance(account);
                break;
            case 4:
                saveAccountToFile(account);
                cout << "You have been logged out. Saving account details...\n";
                return;
            default:
                cout << "Invalid choice. Please enter a number between 1 and 4.\n";
        }
    } while (choice != 4);
}
```

•saveAccountToFile(const Account& account)` Purpose:

Saves the account's state for future sessions.

Logic: Opens a file named `"account.txt"` using `ofstream`.

Writes the `ownerName` and `balance` to the file (each on its own line).

Closes the file if successful, or shows an error message if file opening fails.

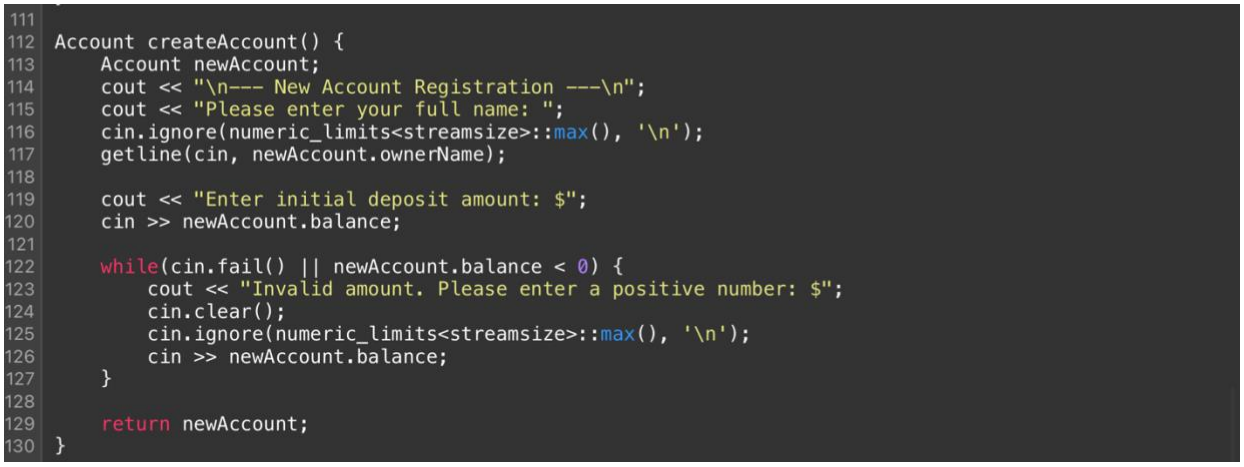
Displays error if file opening fails.

•Account loadAccountFromFile():

Initializes a blank `Account` with zero balance.

Uses `ifstream` to read `ownerName` and `balance`.

Returns the populated `Account` object.



```
Account createAccount() {
    Account newAccount;
    cout << "\n--- New Account Registration ---\n";
    cout << "Please enter your full name: ";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    getline(cin, newAccount.ownerName);

    cout << "Enter initial deposit amount: $";
    cin >> newAccount.balance;

    while(cin.fail() || newAccount.balance < 0) {
        cout << "Invalid amount. Please enter a positive number: $";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cin >> newAccount.balance;
    }

    return newAccount;
}
```