PROJECT REPORT
Bank Management System

Course: Complex Computing Problem (CCP

Subject: Programming Fundamental

Supervised by: Sir Muhammad Abadlah

Department of Computer Science and Information Technolity (CSIT)

Prepared by: Muhammad Waddah

Roll Number: CT-25091

✓ **Prepare it in C language**

Mohammed Wadhah

BANK MANAGEMENT SYSTEM – Project Report

1. Introduction

This project presents a **Simple Bank Management System** developed using the **C programming language**. The system provides users with essential banking operations such as creating new accounts, depositing funds, withdrawing money, and checking account balances. It is designed to be user-friendly, efficient, and structured in a way that makes it suitable for both learning and practical simulation of banking operations.

The program demonstrates several fundamental programming concepts, including the use of **structures** to store account information, **arrays** to manage multiple accounts, and **functions** to implement modular, reusable code. A **menu-driven interface** guides the user through all available operations, ensuring an interactive and intuitive experience.

Furthermore, the project highlights important programming principles such as **input validation**, **error handling**, and **loop control**, ensuring smooth and reliable execution. By separating functionalities across multiple files, the system also exemplifies **modular design**, which enhances code readability, maintainability, and scalability for future enhancements.

Overall, this Bank Management System not only simulates real-world banking operations but also provides an excellent demonstration of applying core C programming skills to a practical, functional project.

2. Problem Statement & Objectives

Problem Statement:
Design and implement a console-based bank management system that supports basic operations such as account creation, deposit, withdrawal, and balance inquiry using structured programming.

Objectives:

- Apply procedural programming concepts such as loops, switch-case, and functions
- Use structures to store account information
- Provide a clean and user-friendly menu interface
- Ensure input validation
- Demonstrate modular programming through multiple files (.c and .h)

3. System Requirements

Hardware Requirements:
• Any standard PC or laptop
• Minimum 2GB RAM
• Minimal storage

Software Requirements:
• C Compiler (GCC / Dev-C++ / CodeBlocks)
• Any text editor (VS Code, Notepad++, etc.)

4. Algorithms and Flowcharts

4.1 Algorithm

High-level algorithm for the bank management program:

⋆ Algorithm 1: Add New Account

Step 1: Display the message **“Enter Account Number”**.
Step 2: Read the account number from the user.
Step 3: Display the message **“Enter Account Holder Name”**.
Step 4: Read the account holder name.
Step 5: Display the message **“Enter Initial Balance”**.
Step 6: Read the initial balance.
Step 7: Store the entered information in the next available index of the accounts list.
Step 8: Increment the total number of accounts.
Step 9: Display a confirmation message **“Account Added Successfully”**.

⋆ Algorithm 2: Deposit Money

Step 1: Display the message **“Enter Account Number”**.
Step 2: Read the account number.
Step 3: Search for the account inside the accounts list.
    • If the account is **not found**, display **“Account Not Found”** and stop.
Step 4: If the account exists, display the message **“Enter Deposit Amount”**.
Step 5: Read the deposit amount.
Step 6: Add the deposit amount to the account’s balance.
Step 7: Display the message **“Deposit Successful”** showing the updated balance.

⋆ Algorithm 3: Withdraw Money

Step 1: Display the message **“Enter Account Number”**.
Step 2: Read the account number.
Step 3: Search for the account in the accounts list.
  • If the account is **not found**, display **“Account Not Found”** and stop.
Step 4: Display the message **“Enter Amount to Withdraw”**.
Step 5: Read the withdrawal amount.
Step 6: Compare the amount with the account’s current balance:
  • If the amount is **greater**, display **“Insufficient Balance”** and stop.
  • Otherwise, continue.
Step 7: Deduct the withdrawal amount from the balance.
Step 8: Display **“Withdrawal Successful”** with the updated balance.

⋆ Algorithm 4: Check Balance

Step 1: Display the message **“Enter Account Number”**.
Step 2: Read the account number.
Step 3: Search for the account in the accounts list.
  • If not found → Display **“Account Not Found”**.
Step 4: If found → Display:
  • Account Number
  • Account Holder Name
  • Current Balance

⋆ Algorithm 5: View All Accounts

Step 1: Display the title **“All Accounts”**.
Step 2: Loop through each account in the accounts list.
  • Print the account number
  • Print the account holder name
  • Print the balance
Step 3: End the loop.

⋆ Algorithm 6: Main Program Workflow

Step 1: Initialize an empty array for storing accounts.
Step 2: Set the account counter = 0.
Step 3: Repeat the following steps until the user chooses Exit:

  • Display the main menu.
  • Read the user’s choice.

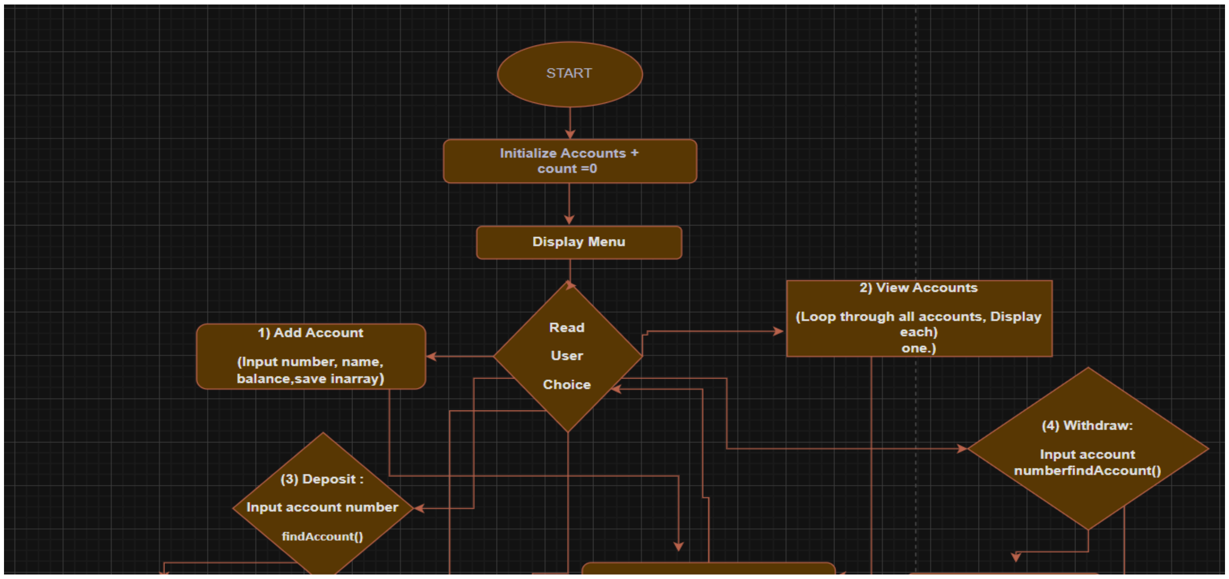
Based on the choice:

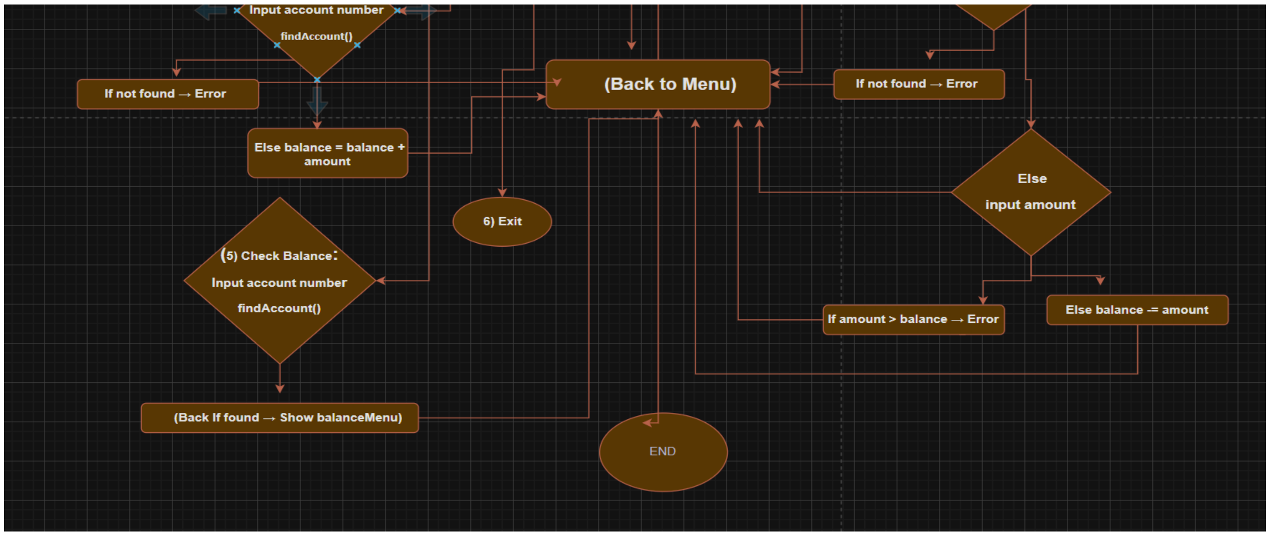
- 1 → Call **Add New Account**
- 2 → Call **View Accounts**
- 3 → Call **Deposit Money**
- 4 → Call **Withdraw Money**
- 5 → Call **Check Balance**
- 6 → Terminate the program

- If none of the above → Display **“Invalid Choice”**

Step 4: End the program with a goodbye message.

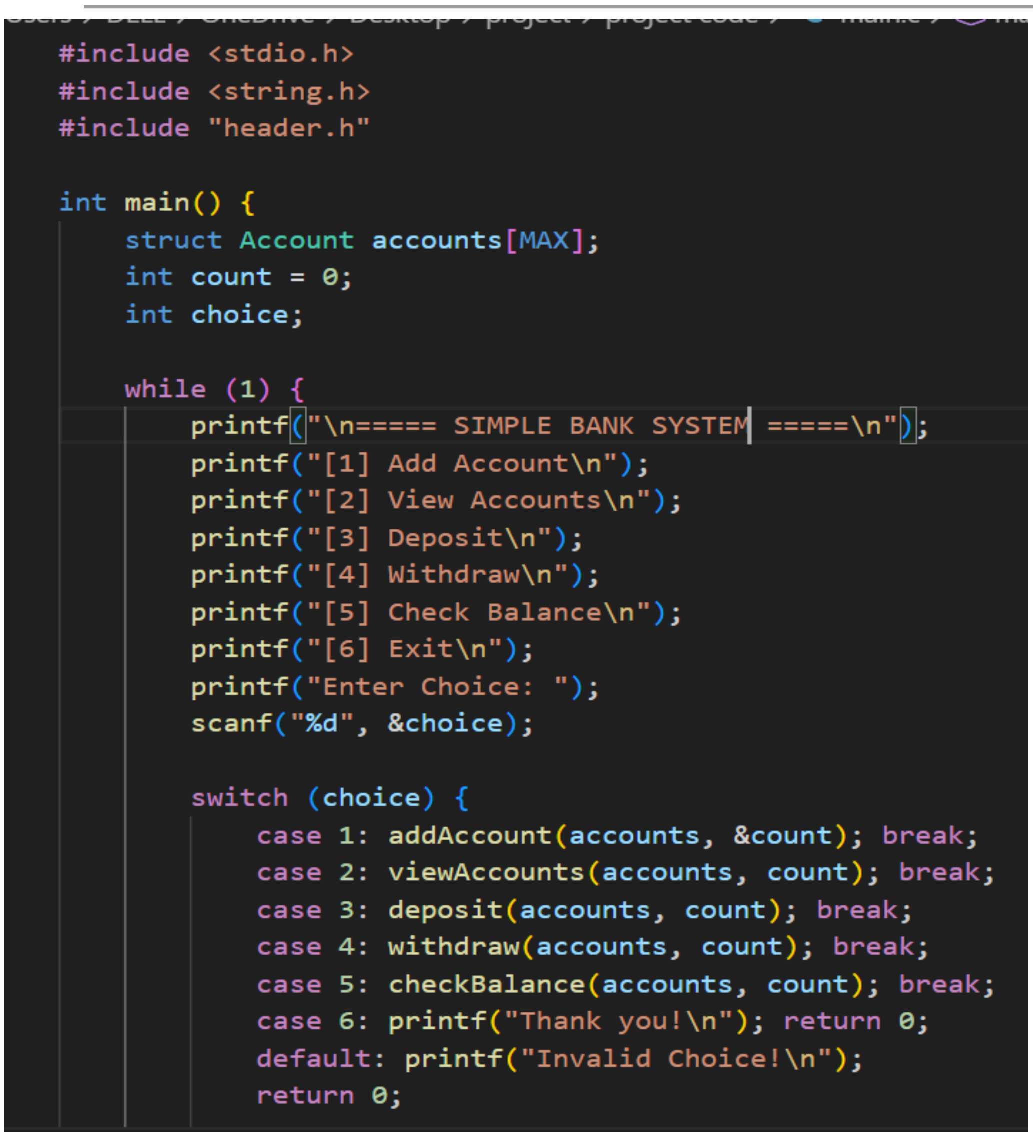
4.2 Flowchart


START
Initialize Accounts + count =0
Display Menu
Read User Choice
1) Add Account (Input number, name, balance,save inarray)
2) View Accounts (Loop through all accounts, Display each) one.)
(3) Deposit : Input account number findAccount()
(4) Withdraw: Input account numberfindAccount()



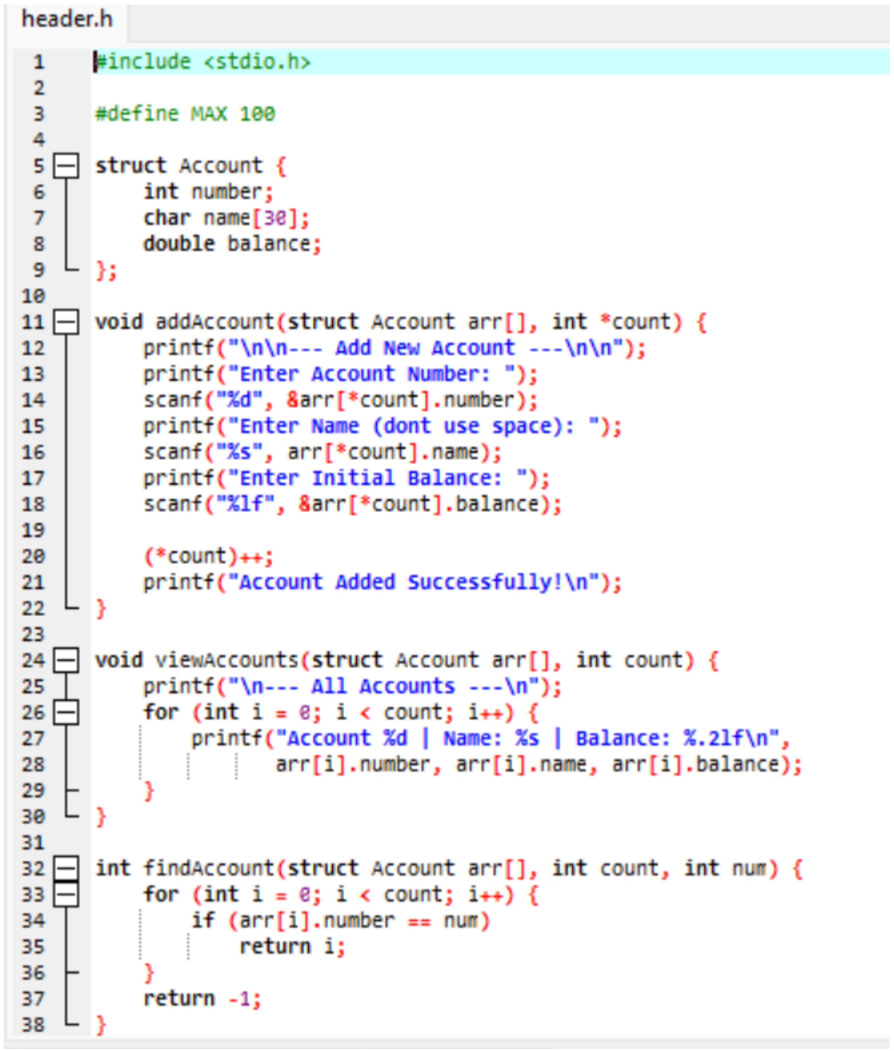
5. Source Code

The system is divided into modular components to improve organization, with each part responsible for a specific function.



```c
#include <stdio.h>
#include <string.h>
#include "header.h"

int main() {
    struct Account accounts[MAX];
    int count = 0;
    int choice;

    while (1) {
        printf("\n===== SIMPLE BANK SYSTEM =====\n");
        printf("[1] Add Account\n");
        printf("[2] View Accounts\n");
        printf("[3] Deposit\n");
        printf("[4] Withdraw\n");
        printf("[5] Check Balance\n");
        printf("[6] Exit\n");
        printf("Enter Choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addAccount(accounts, &count); break;
            case 2: viewAccounts(accounts, count); break;
            case 3: deposit(accounts, count); break;
            case 4: withdraw(accounts, count); break;
            case 5: checkBalance(accounts, count); break;
            case 6: printf("Thank you!\n"); return 0;
            default: printf("Invalid Choice!\n");
            return 0;
```

5.1 Source Code Explanation



header.h

```c
#include <stdio.h>

#define MAX 100

struct Account {
    int number;
    char name[30];
    double balance;
};

void addAccount(struct Account arr[], int *count) {
    printf("\n\n--- Add New Account ---\n\n");
    printf("Enter Account Number: ");
    scanf("%d", &arr[*count].number);
    printf("Enter Name (dont use space): ");
    scanf("%s", arr[*count].name);
    printf("Enter Initial Balance: ");
    scanf("%lf", &arr[*count].balance);

    (*count)++;
    printf("Account Added Successfully!\n");
}

void viewAccounts(struct Account arr[], int count) {
    printf("\n--- All Accounts ---\n");
    for (int i = 0; i < count; i++) {
        printf("Account %d | Name: %s | Balance: %.2lf\n",
               arr[i].number, arr[i].name, arr[i].balance);
    }
}

int findAccount(struct Account arr[], int count, int num) {
    for (int i = 0; i < count; i++) {
        if (arr[i].number == num)
            return i;
    }
    return -1;
}
```

Explanation:
The `header.h` file contains all function declarations and the definition of the `Account` structure.
The structure stores each account's number, name, and balance, allowing the program to manage multiple accounts efficiently.
Function declarations include:

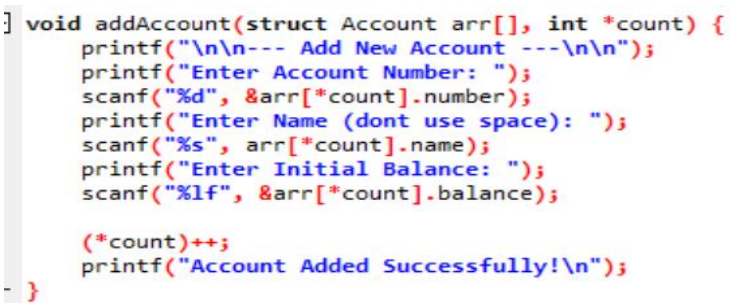
- `addAccount()` → Adds a new account
- `viewAccounts()` → Displays all accounts
- `deposit()` → Adds money to an account
- `withdraw()` → Withdraws money from an account
- `checkBalance()` → Shows the balance of a selected account

functions.c

functions.c

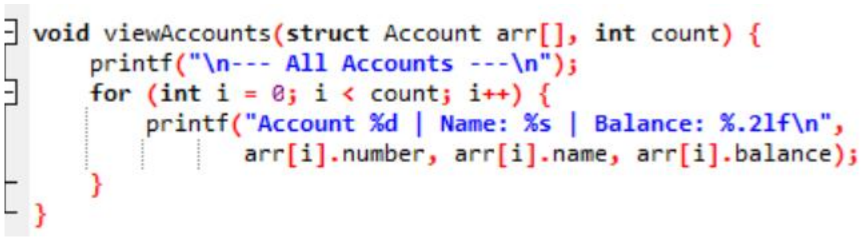
Explanation:
The **`functions.c`** file implements the functions declared in the header. Each function handles a specific task:

1. **`addAccount`**`()`: Reads user input, stores data in the accounts array, updates the account count.



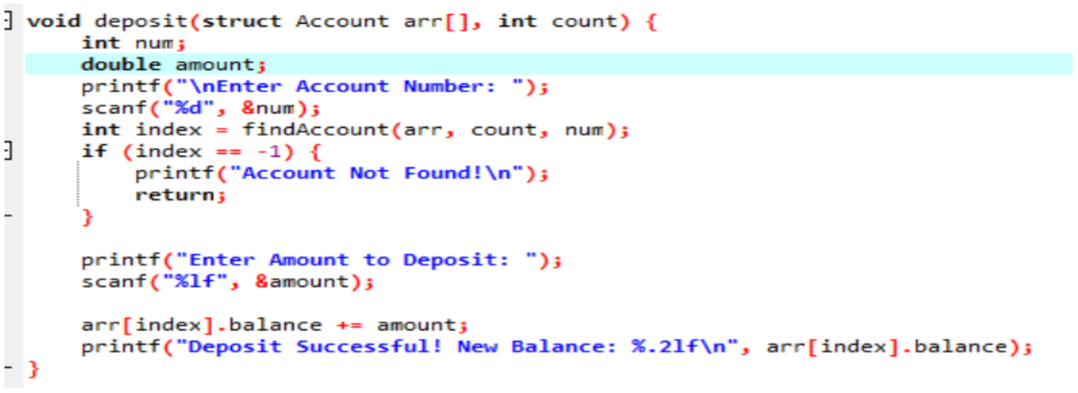
```c
void addAccount(struct Account arr[], int *count) {
    printf("\n\n--- Add New Account ---\n\n");
    printf("Enter Account Number: ");
    scanf("%d", &arr[*count].number);
    printf("Enter Name (dont use space): ");
    scanf("%s", arr[*count].name);
    printf("Enter Initial Balance: ");
    scanf("%lf", &arr[*count].balance);

    (*count)++;
    printf("Account Added Successfully!\n");
}
```

2. **`viewAccounts`**`()`: Displays all accounts with account number, name, and balance.



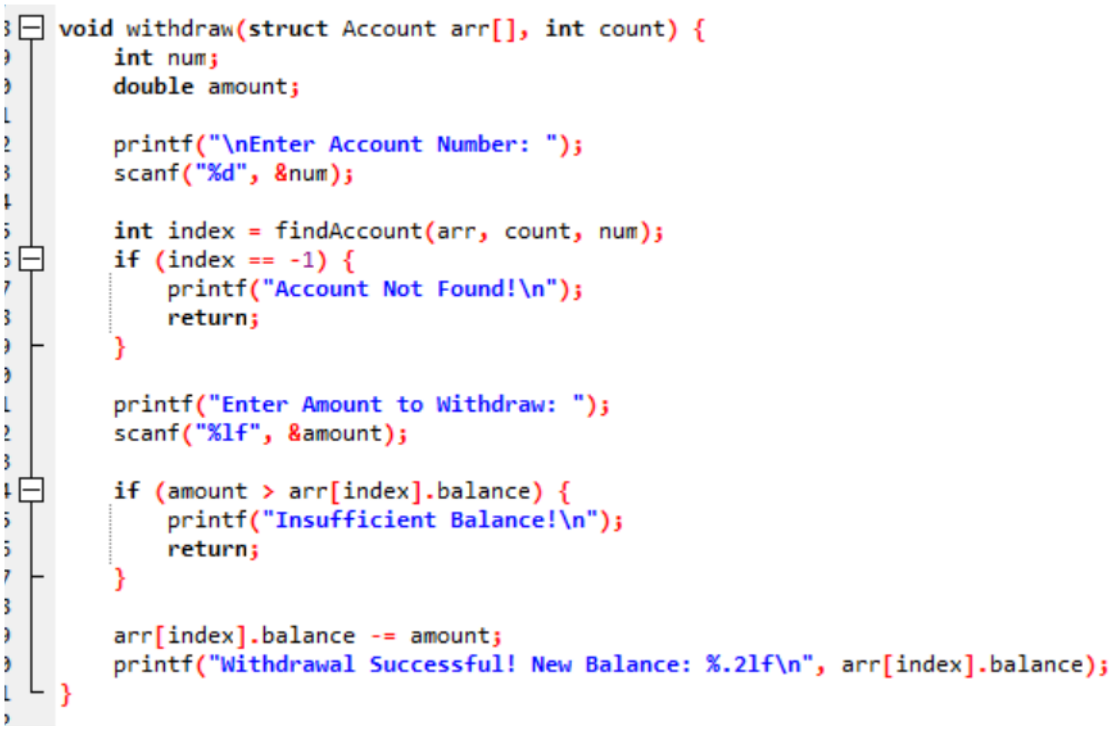
```c
void viewAccounts(struct Account arr[], int count) {
    printf("\n--- All Accounts ---\n");
    for (int i = 0; i < count; i++) {
        printf("Account %d | Name: %s | Balance: %.2lf\n",
               arr[i].number, arr[i].name, arr[i].balance);
    }
}
```

3. **`deposit`**`()`: Finds the account by number and adds the specified amount.



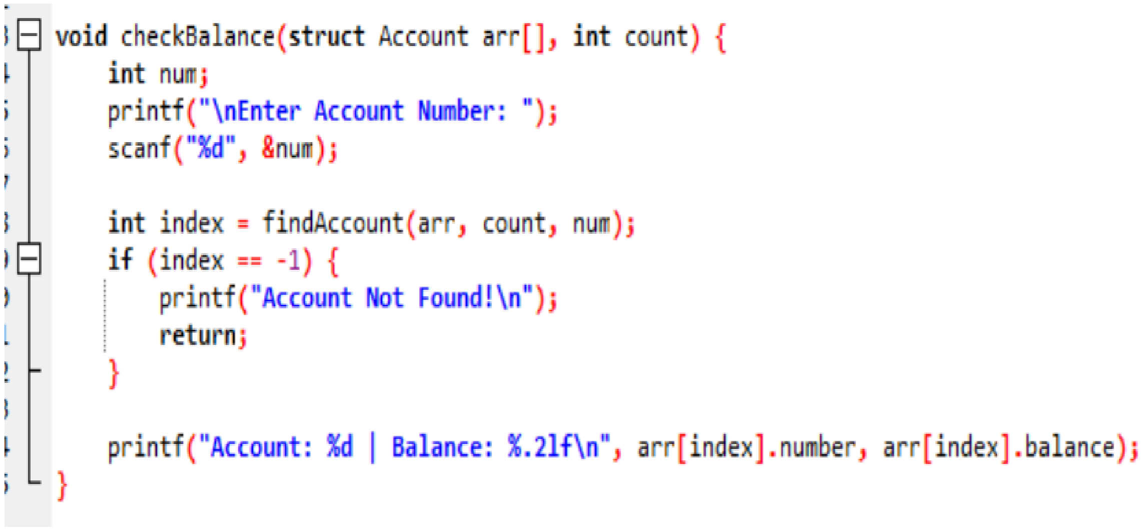
```c
void deposit(struct Account arr[], int count) {
    int num;
    double amount;
    printf("\nEnter Account Number: ");
    scanf("%d", &num);
    int index = findAccount(arr, count, num);
    if (index == -1) {
        printf("Account Not Found!\n");
        return;
    }

    printf("Enter Amount to Deposit: ");
    scanf("%lf", &amount);

    arr[index].balance += amount;
    printf("Deposit Successful! New Balance: %.2lf\n", arr[index].balance);
}
```

`4.withdraw()`: Checks for sufficient balance, deducts the amount, updates balance.



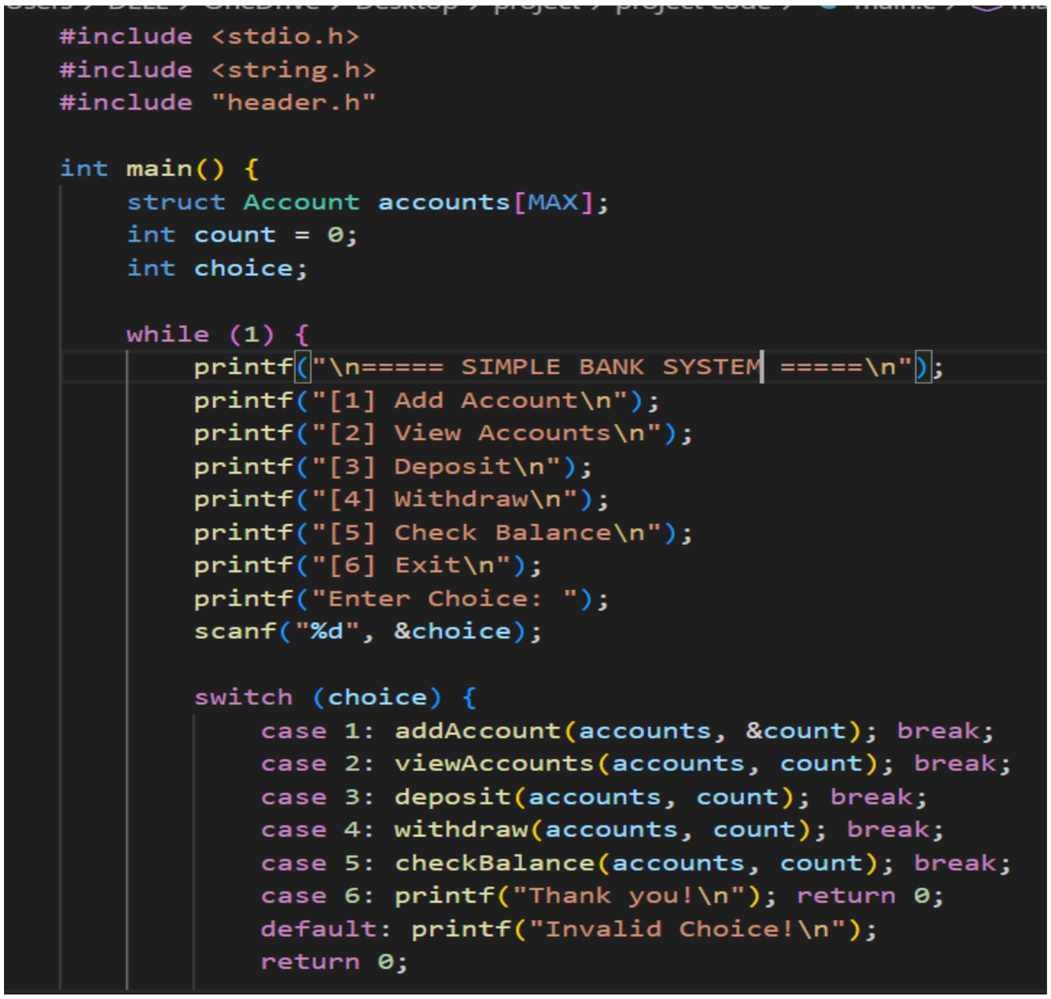
```c
void withdraw(struct Account arr[], int count) {
    int num;
    double amount;

    printf("\nEnter Account Number: ");
    scanf("%d", &num);

    int index = findAccount(arr, count, num);
    if (index == -1) {
        printf("Account Not Found!\n");
        return;
    }

    printf("Enter Amount to Withdraw: ");
    scanf("%lf", &amount);

    if (amount > arr[index].balance) {
        printf("Insufficient Balance!\n");
        return;
    }

    arr[index].balance -= amount;
    printf("Withdrawal Successful! New Balance: %.2lf\n", arr[index].balance);
}
```

`5.checkBalance()`: Displays account details.



```c
void checkBalance(struct Account arr[], int count) {
    int num;
    printf("\nEnter Account Number: ");
    scanf("%d", &num);

    int index = findAccount(arr, count, num);
    if (index == -1) {
        printf("Account Not Found!\n");
        return;
    }

    printf("Account: %d | Balance: %.2lf\n", arr[index].number, arr[index].balance);
}
```

main.c

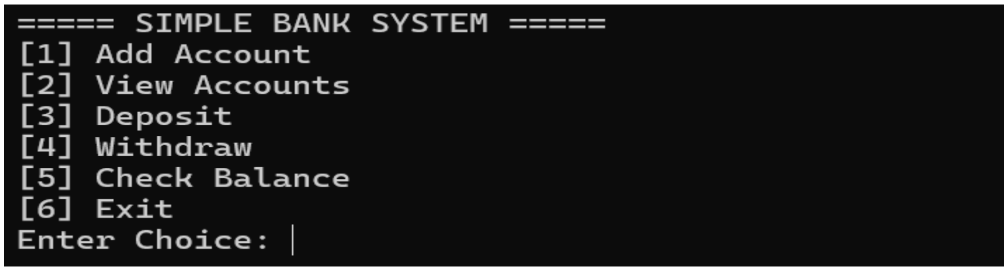


```c
#include <stdio.h>
#include <string.h>
#include "header.h"

int main() {
    struct Account accounts[MAX];
    int count = 0;
    int choice;

    while (1) {
        printf("\n===== SIMPLE BANK SYSTEM =====\n");
        printf("[1] Add Account\n");
        printf("[2] View Accounts\n");
        printf("[3] Deposit\n");
        printf("[4] Withdraw\n");
        printf("[5] Check Balance\n");
        printf("[6] Exit\n");
        printf("Enter Choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addAccount(accounts, &count); break;
            case 2: viewAccounts(accounts, count); break;
            case 3: deposit(accounts, count); break;
            case 4: withdraw(accounts, count); break;
            case 5: checkBalance(accounts, count); break;
            case 6: printf("Thank you!\n"); return 0;
            default: printf("Invalid Choice!\n");
            return 0;
```

Explanation:
The `main.c` file contains the **menu-driven interface** and main program loop:

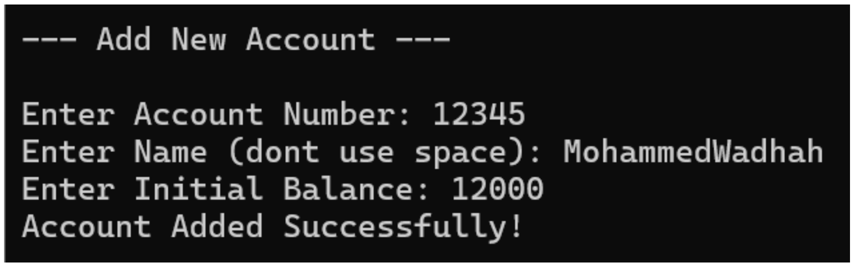
- A `while` loop runs until the user chooses to exit.
- Menu options: Add Account, View Accounts, Deposit, Withdraw, Check Balance, Exit.
- User input is read and validated.
- Corresponding functions from `functions.c` are called based on the choice.
- This modular structure makes the program **organized, readable, and maintainable**.

5.2 Description of Functionalities & Sample Output



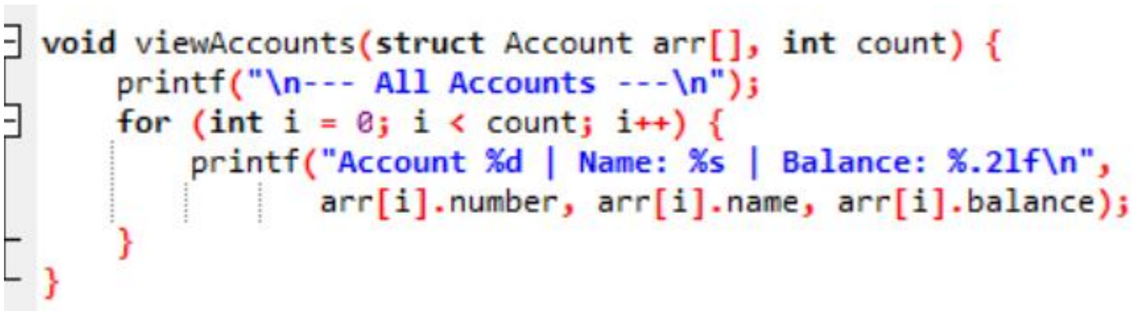
```
===== SIMPLE BANK SYSTEM =====
[1] Add Account
[2] View Accounts
[3] Deposit
[4] Withdraw
[5] Check Balance
[6] Exit
Enter Choice:
```

Explanation:
The menu provides numbered options for easy navigation. It guides the user to perform all banking operations interactively.



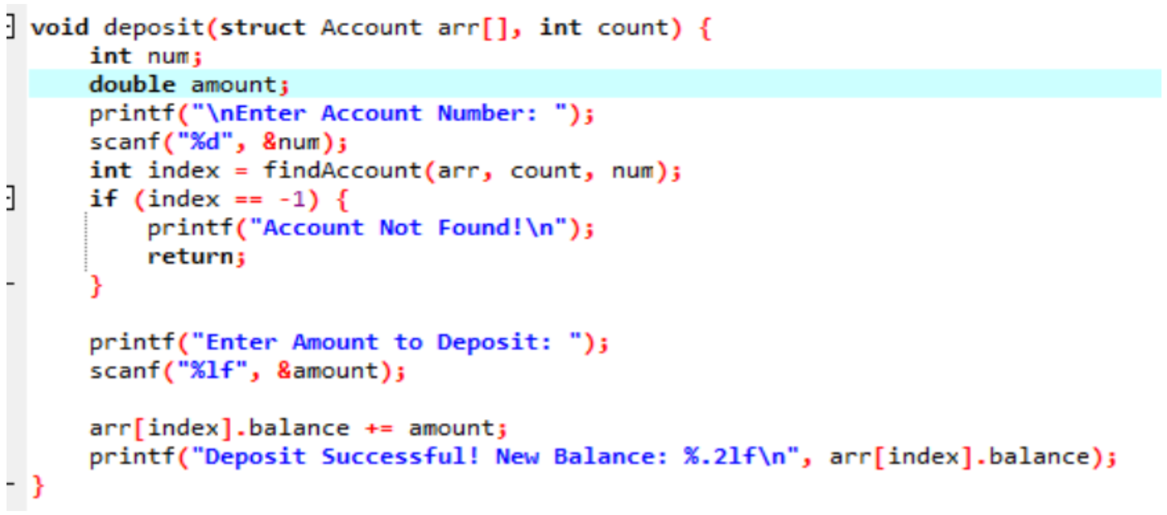
```
--- Add New Account ---

Enter Account Number: 12345
Enter Name (dont use space): MohammedWadhah
Enter Initial Balance: 12000
Account Added Successfully!
```

Explanation:
The program prompts for account number, name, and initial balance. After successful input, it confirms the account creation.



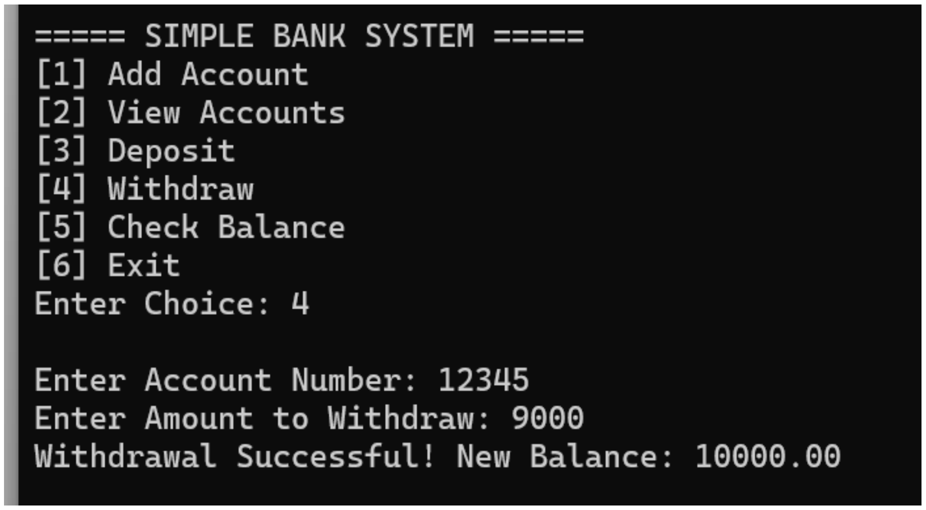
```c
void viewAccounts(struct Account arr[], int count) {
    printf("\n--- All Accounts ---\n");
    for (int i = 0; i < count; i++) {
        printf("Account %d | Name: %s | Balance: %.2lf\n",
               arr[i].number, arr[i].name, arr[i].balance);
    }
}
```

Explanation:
All accounts are displayed in tabular form, showing account number, name, and current balance.



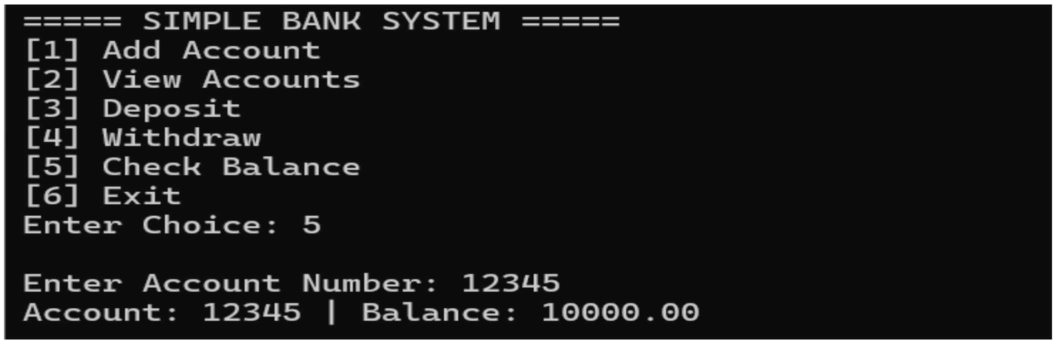
```c
void deposit(struct Account arr[], int count) {
    int num;
    double amount;
    printf("\nEnter Account Number: ");
    scanf("%d", &num);
    int index = findAccount(arr, count, num);
    if (index == -1) {
        printf("Account Not Found!\n");
        return;
    }

    printf("Enter Amount to Deposit: ");
    scanf("%lf", &amount);

    arr[index].balance += amount;
    printf("Deposit Successful! New Balance: %.2lf\n", arr[index].balance);
}
```

Explanation:
After selecting an account and entering the deposit amount, the balance is updated and a confirmation message is displayed.



```
===== SIMPLE BANK SYSTEM =====
[1] Add Account
[2] View Accounts
[3] Deposit
[4] Withdraw
[5] Check Balance
[6] Exit
Enter Choice: 4

Enter Account Number: 12345
Enter Amount to Withdraw: 9000
Withdrawal Successful! New Balance: 10000.00
```

Explanation:
Withdrawals are checked against the account balance. If sufficient, the amount is deducted and the new balance is shown.



```
===== SIMPLE BANK SYSTEM =====
[1] Add Account
[2] View Accounts
[3] Deposit
[4] Withdraw
[5] Check Balance
[6] Exit
Enter Choice: 5

Enter Account Number: 12345
Account: 12345 | Balance: 10000.00
```

Explanation:
Users can enter an account number to view account details including account holder name and current balance.

5.3 Concepts Covered and Justification

	Concept	Usage in the Project	Reason / Justification
1	**Structures (`struct`)**	Creating the `record` structure to store customer information (name, account number, phone, balance, etc.).	Organizes multiple related fields into one object, making data handling cleaner and more efficient.
2	**Arrays**	Used for storing strings such as customer name, email, address, and UserID.	Arrays allow storing multiple values in a single variable, which is essential for handling text input.
3	**Pointers**	Passing balances and structures by reference (e.g., updating balance using `&balance`).	Enables modifying original values inside functions and improves performance by avoiding copying large data.
4	**Functions**	Project divided into functions: **`addAccount(),`** **`viewAccounts(),`** **`editAccount(),`** **`searchAccount(), deposit().`**	Improves modularity, readability, troubleshooting, and code reusability.
5	**File Handling**	Storing and retrieving data from **`record.bin`** **using** **`fopen(),`** **`fwrite(), fread().`**	Required for permanent data storage so that account information is saved after program exit.
6	**Conditional Statements**	Checking login validity, verifying account existence, ensuring enough balance before withdrawal.	Controls program logic and decision-making at every stage.
7	**Loops (for / while)**	Iterating through account records while searching, viewing, or editing data.	Automates repetitive tasks and allows processing multiple accounts efficiently.
8	**Switch Case**	Handling menu choices for Admin Menu, User Menu, and Transaction Menu.	Simplifies menu navigation and organizes branching options cleanly.
9	**User Input Handling**	Receiving account details and login credentials using **`scanf()`** and **`fgets()`**.	Essential for interacting with users and collecting necessary information.
10	**Menu-Driven Programming**	Admin/User menus with options for add, view, edit, delete, deposit, withdraw, etc.	Provides a clear and beginner-friendly user interface inside the console.

Explanation:
The project demonstrates key programming concepts:

1. **Structures:** Store all account information in a single entity.
2. **Functions:** Modular design separates tasks for clarity and reusability.
3. **Arrays:** Enable management of multiple accounts in memory.
4. **Menu-driven programming:** Provides user-friendly interaction.
5. **Input validation:** Prevents crashes and ensures smooth execution.

Justification:

Using these concepts makes the program **organized, modular, maintainable, and scalable**. Future enhancements could include transaction history, account deletion, or file storage for persistent data.

6. Limitations and Future Enhancements

Limitations:
• No file storage (data lost when program ends)
• No authentication system
• No search filters or sorting

Future Enhancements:
• Add file handling to save accounts
• Add password system for security
• Add transaction history
• Add GUI or SQL database connection

7. Conclusion

The Bank Management System demonstrates core C programming concepts, structured design, and a functional menu-based interface. It is simple, efficient, and can be extended into a full banking application.

8. Report Prepared :

Report Prepared by: Mohammed Wadhah