A Project Report on

SIMPLE BANKING SYSTEM

Submitted in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology

In

Electronics and Communication Engineering

By

K. NITHEESH - 24KB1A04G7 D. VINAY - 24KB1A0478

Y. ABHISHEK KUMAR REDDY - 24KB1A04BY

P. SRINADH - 24KB1A04R0 K. NIRAJAN -24KB1A04G5

SK. FAIKUDDHIN - 24KB1A04X4

Under the Esteemed Supervision of

Mr. A.V.V. SATHISH (byte xl trainer)

Department of Electronics and Communication Engineering

NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY

(An Autonomous Institution)

(Approved by AICTE, Tirupati, JNTUA Accredited by NBA with 'A' Grade)

Vidyanagar, Naidupeta road, Kota

2024-2028

AKNOWLEDGEMENT

It is with immense pleasure that we would like to express our indebted gratitude to our project guide **Mr. A.V.V. Sathish (byte xl trainer),** who has guided us a lot and encouraged us in every step of the project work, her valuable moral support and guidance throughout the project helped us to a greater extent.

Our sincere thanks to my team and faculty who are supported us to complete this project and **NBKRIST**

DECLARATION

We hereby to declare that the project entitled “**SIMPLE BANKING SYSTEM”** is a genuine project. The work has been submitted to the **NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY,** Vidyanagar, permanently affiliated to **JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, ANANTHAPURAMU,** in a partial fulfillment of the **B.TECH** degree, We further declare that this project work has not been submitted in full or part for the award of any degree of this or any other educational institutions.

By

KONDA NITHEESH(24KB1A04G7)

DASARI VINAY(24KB1A0478)

PANABAKA SRINADH(24KB1A04R0)

KOLAMGARI NIRANJAN(24KB1A04G5)

SHAIK FAIKUDDHIN(24KB1A04X4)

YETURU ABHISHEK KUMAR REDDY(24KB1A04BY)

INTRODUCTION OF SIMPLE BANKING SYSTEM :

The Bank Management System is a console-based application developed using the C programming language. It simulates the basic functionalities of a banking environment, allowing users to create and manage customer accounts. The system uses structures and arrays to store and manipulate customer data such as account number, account holder name, and account balance.

This project aims to demonstrate the practical use of user-defined data types, functions, and menu-driven programming. By providing a simple interface, users can perform operations like creating new accounts, depositing and withdrawing money, viewing all accounts, searching for specific accounts, and deleting accounts.

The application operates entirely in memory without the use of files or databases, making it ideal for educational purposes and understanding the fundamentals of data storage and manipulation in C.

🔑 Key Features:

- **Use of structures for real-world modeling.**
- **Implementation of CRUD operations (Create, Read, Update, Delete).**
- **Interactive menu system for user input.**
- **Linear search and array shifting techniques for data handling.**

AIM :

To develop a simple console-based Bank Management System using the C programming language and structures that allows users to perform basic banking operations such as account creation, deposit, withdrawal, account deletion, search, and display of account details.

📌 Objectives:

- **To understand and apply structures in C for managing real-world entities (like bank accounts).**
- **To implement menu-driven programming for interactive user operations.**
- **To perform basic data management using static arrays and procedural programming.**
- **To develop hands-on skills in fileless data handling, user input validation, and control structures.**

🔄 Processes

1. Create Account – Takes user input, stores account info, initializes balance to 0.
2. Deposit – Adds amount to the matched account balance.
3. Withdraw – Deducts amount if sufficient balance exists.
4. Delete Account – Removes account by shifting array elements.
5. Display Accounts – Shows all account details.
6. Search Account – Finds and displays details of a specific account.
7. Exit – Ends the program.

📦 Contents

- Headers: stdio.h, string.h
- Macro: #define MAX_ACCOUNTS 100
- Structures:
  - Account – stores number, name, balance
  - Bank – array of accounts, number of active accounts
- Functions:
  - createAccount(), deposit(), withdraw(), deleteAccount(), displayAccounts(), searchAccount()
- Main Function: Handles menu and user choices.

Program :

```
#include <stdio.h>
#include <string.h>

#define MAX_ACCOUNTS 100

struct Account {
    int accountNumber;
    char accountHolderName[100];
    float balance;
};

struct Bank {
    struct Account accounts[MAX_ACCOUNTS];
    int numAccounts;
};

void createAccount(struct Bank *bank) {
    if (bank->numAccounts < MAX_ACCOUNTS) {
        printf("Enter account number: ");
        scanf("%d", &bank->accounts[bank->numAccounts].accountNumber);

        // Clear the input buffer
        while (getchar() != '\n');

        printf("Enter account holder name: ");
        fgets(bank->accounts[bank->numAccounts].accountHolderName, sizeof(bank->accounts[bank->numAccounts].accountHolderName), stdin);
        // Remove trailing newline from the input
        bank->accounts[bank->numAccounts].accountHolderName[strcspn(bank->accounts[bank->numAccounts].accountHolderName, "\n")] = '\0';

        bank->accounts[bank->numAccounts].balance = 0.0;
        bank->numAccounts++;
        printf("Account created successfully!\n");
```

```
    } else {
        printf("Maximum number of accounts 
reached!\n");
    }
}

void deposit(struct Bank *bank) {
    int accountNumber;
    float amount;
    int found = 0;

    printf("Enter account number: ");
    scanf("%d", &accountNumber);

    for (int i = 0; i < bank->numAccounts; i++) {
        if (bank->accounts[i].accountNumber == 
accountNumber) {
            printf("Enter amount to deposit: ");
            scanf("%f", &amount);
            bank->accounts[i].balance += amount;
            printf("Deposit successful! New balance: 
%.2f\n", bank->accounts[i].balance);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Account not found!\n");
    }
}

void withdraw(struct Bank *bank) {
    int accountNumber;
    float amount;
    int found = 0;

    printf("Enter account number: ");
    scanf("%d", &accountNumber);
    for (int i = 0; i < bank->numAccounts; i++) {
        if (bank->accounts[i].accountNumber ==
accountNumber) {
            printf("Enter amount to withdraw: ");
            scanf("%f", &amount);
            if (amount > bank->accounts[i].balance) {
                printf("Insufficient balance!\n");
            } else {
                bank->accounts[i].balance -= amount;
                printf("Withdrawal successful! New balance:
%.2f\n", bank->accounts[i].balance);
            }
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Account not found!\n");
    }
}
```

```
void deleteAccount(struct Bank *bank) {
    int accountNumber;
    int found = 0;

    printf("Enter account number to delete: ");
    scanf("%d", &accountNumber);

    for (int i = 0; i < bank->numAccounts; i++) {
        if (bank->accounts[i].accountNumber ==
accountNumber) {
            // Shift the accounts to the left to fill the gap
            for (int j = i; j < bank->numAccounts - 1; j++) {
                bank->accounts[j] = bank->accounts[j + 1];
            }
            bank->numAccounts--;
            printf("Account deleted successfully!\n");
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Account not found!\n");
    }
}

void displayAccounts(struct Bank *bank) {
    if (bank->numAccounts == 0) {
        printf("No accounts available!\n");
    } else {
        printf("Available Accounts:\n");
        for (int i = 0; i < bank->numAccounts; i++) {
            printf("Account Number: %d\n", bank-
>accounts[i].accountNumber);
            printf("Account Holder Name: %s\n", bank-
>accounts[i].accountHolderName);
            printf("Balance: %.2f\n\n", bank-
>accounts[i].balance);
        }
    }
```

```
}

void searchAccount(struct Bank *bank) {
    int accountNumber;
    int found = 0;

    printf("Enter account number to search: ");
    scanf("%d", &accountNumber);

    for (int i = 0; i < bank->numAccounts; i++) {
        if (bank->accounts[i].accountNumber ==
accountNumber) {
            printf("Account found!\n");
            printf("Account Number: %d\n", bank-
>accounts[i].accountNumber);
            printf("Account Holder Name: %s\n", bank-
>accounts[i].accountHolderName);
            printf("Balance: %.2f\n", bank-
>accounts[i].balance);
            found = 1;
            break;
```

```c
        }
    }

    if (!found) {
        printf("Account not found!\n");
    }
}

int main() {
    struct Bank bank;
    bank.numAccounts = 0;
    int choice;

    while (1) {
        printf("\nBanking System Menu:\n");
        printf("1. Create Account\n");
        printf("2. Deposit\n");
        printf("3. Withdraw\n");
        printf("4. Delete Account\n");
        printf("5. Display Accounts\n");
        printf("6. Search Account\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createAccount(&bank);
                break;
            case 2:
                deposit(&bank);
                break;
            case 3:
                withdraw(&bank);
                break;
            case 4:
                deleteAccount(&bank);
                break;
            case 5:
                displayAccounts(&bank);
```

```
            break;
          case 6:
            searchAccount(&bank);
            break;
          case 7:
            printf("Exiting...\n");
            return 0;
          default:
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Work flow :

📌

1. Start the Program

- The program starts executing from the main() function.
- It initializes a Bank structure to store account details.
- Enters an infinite loop to show the menu repeatedly until the user exits.

📋 2. Display Menu

The user is presented with a list of options:

1. Create Account
2. Deposit
3. Withdraw
4. Delete Account
5. Display Accounts
6. Search Account
7. Exit

3. User Input (Choice)

- The user selects an option by entering the corresponding number.
- The program uses a switch statement to handle the selected operation.

4. Operation Execution

Depending on the user’s choice:

✅ **Create Account**

- Prompts for account number and account holder name.
- Initializes balance to 0 and adds the account to the array.

💰 **Deposit**

- Prompts for account number and amount.
- Searches for the account and adds the amount to the balance.

💸 **Withdraw**

- Prompts for account number and withdrawal amount.
- Validates sufficient balance before deducting the amount.

❌ Delete Account

- Prompts for account number.
- If found, shifts the accounts in the array to remove the record.

🧾 Display Accounts

- Loops through the array and prints details of all existing accounts.

🔍 Search Account

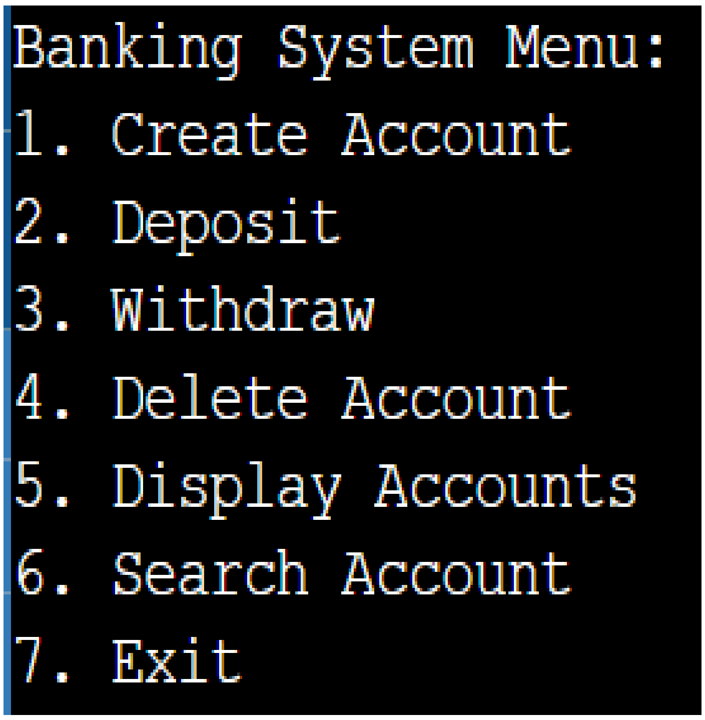
- Prompts for account number.
- If found, displays the account holder’s details and balance.

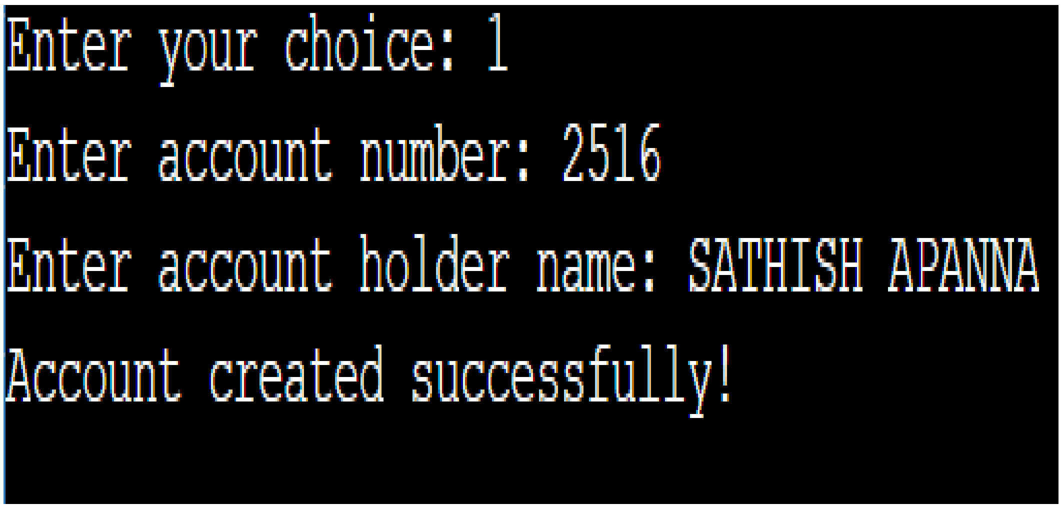
🚪 Exit

Flow chart :

```
Start
  ↓
Initialize Bank Structure
  ↓
Display Menu → Get User Choice
  ↓
┌──────────────────────────────┐
|     Perform Operation     |
|  (Based on user choice)    |
└──────────────────────────────┘
  ↓
Repeat Until Choice = 7
  ↓
Exit Program
```

OUTPUT IMAGES :


```
Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Delete Account
5. Display Accounts
6. Search Account
7. Exit
```


```
Enter your choice: 1
Enter account number: 2516
Enter account holder name: SATHISH APANNA
Account created successfully!
```

ADVANTAGES OF SIMPLE BANKING SYSTEM :

1.Fast & Efficient Processing

$\boxed{\rightarrow}$ C language executes operations quickly with low memory usage.

2. Structured Data Management

$\boxed{\rightarrow}$ struct helps group account details logically (name, balance,

number, etc.)

3. Quick Search & Retrieval

$\boxed{\rightarrow}$ Trees and hash tables allow fast access to customer records.

4. Dynamic Account Handling

$\boxed{\rightarrow}$ Linked lists let us add or delete accounts without fixed size limitations

✅ Real-Time Uses:

1. **Basic Account Management System**
   - Used in small financial institutions or local banking setups to manage account holders, balances, and transactions**.**
2. **Educational Simulations and Practicals**
   - Helps students understand core banking operations, data handling, and C programming through real-world scenarios.
3. **ATM Logic Demonstration**
   - Simulates how deposits, withdrawals, and balance checks are performed in ATM systems.
4. **Prototype for Banking Applications**
   - Acts as a starting point for developing larger, real-world banking or fintech software.
5. **Local Money Lending System**
   - Can be adapted for village co-operative societies, self-help groups, or private money lenders to manage client transactions.

6. **POS System Integration (Conceptual)**
    - The code logic could be adapted into a point-of-sale (POS) terminal for storing and modifying customer financial data.
7. **CLI-based Finance Management Tool**
    - Useful for developers or finance teams needing a command-line tool to track dummy financial operations during testing.

Conclusion :

The Bank Management System developed using the C programming language efficiently simulates basic banking operations such as:

- Account creation
- Deposits and withdrawals
- Deleting accounts
- Displaying and searching account details

By utilizing structures, arrays, and user-defined functions, this project provides a clear and modular approach to handling real-time banking data. It enhances understanding of file-less data storage, user input handling, and logical control flow in C.

This system is best suited for:

- Educational purposes
- Learning basic data structures
- Simulating a prototype for future banking applications