SIMPLE BANKING APPLICATIONS

Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

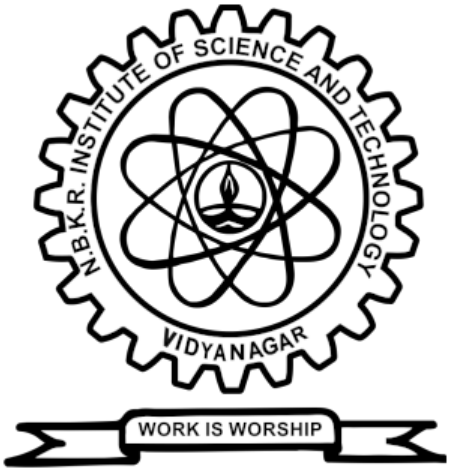
24KB1A05NB

24KB1A05JK

24KB1A05FC

Under the Guidance of

SMT.B.SRUTHI

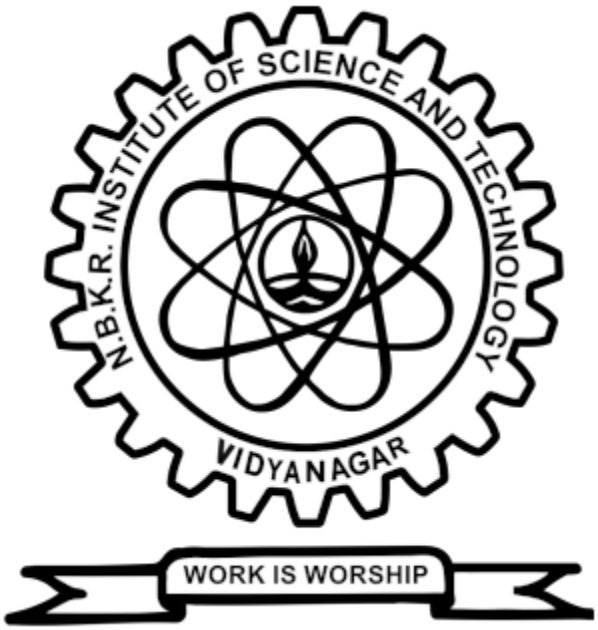


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NBKRIST

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project report entitled Simple Banking Applications being submitted by

24KB1A05NB

24KB1A05JK

24KB1A05FC

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Jawaharlal Nehru Technological University, Kakinada is a record of bonafied work carried out under my guidance and supervision.

SMT.B.SRUTHI	**Dr. HOD RAJSHEKAR REDDY**
	M.Tech, Ph.D
Designation	**Head of the Department**

DECLARATION

I hereby declare that the dissertation entitled **Simple Banking Applications** submitted for the B.Tech Degree is my original work and the dissertation has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Place:Vidyanagar

24KB1A05N

24KB1A05JK

24KB1A05FC

Date: 05-05-25

Acknowledgement

I would like to express my sincere gratitude to all those who supported me throughout the course of this project.

First and foremost, I am deeply thankful to SMT.B.SRUTHI, whose guidance, feedback, and support were invaluable in completing this work.

I also extend my appreciation to NBKRIST for providing the resources and environment necessary for the successful execution of the project.

Special thanks to my colleagues, friends, and family for their encouragement and moral support during this journey.

Lastly, I am grateful to all those who contributed, directly or indirectly, to the completion of this project.

Abstract

This project is a simple banking application that allows users to perform basic banking tasks. The main features include

creating a bank account, depositing money, withdrawing money, and viewing transaction statements. Each user can create an account and safely manage their money. The application keeps track of all deposits and withdrawals and shows a clear list of all transactions. This system is easy to use and helps understand how basic banking works in a digital form.

1.INTRODUCTION

Banking systems are crucial for managing financial transactions efficiently and securely. A simple banking application simulates essential features of a bank, such as creating an account, depositing funds, withdrawing funds, and displaying transaction statements. Developing such a system in C provides hands-on experience with structured

programming, file handling, and modular design. This project is particularly useful for students learning foundational programming concepts and their practical applications.

2.PROBLEM STATEMENT

Manual banking processes are inefficient, prone to human error, and lack real-time tracking. Even basic operations like keeping track of deposits and withdrawals can become complex without automation. There's a need for a simple, console-based banking system that performs core banking tasks programmatically using the C language, allowing users to handle transactions securely and conveniently.

3. SCOPE

The project will focus on implementing the following features using the C language:

- Account creation (with simple user information)
- Deposit of funds
- Withdrawal with balance checks
- Displaying transaction history
- File-based storage for account and transaction data

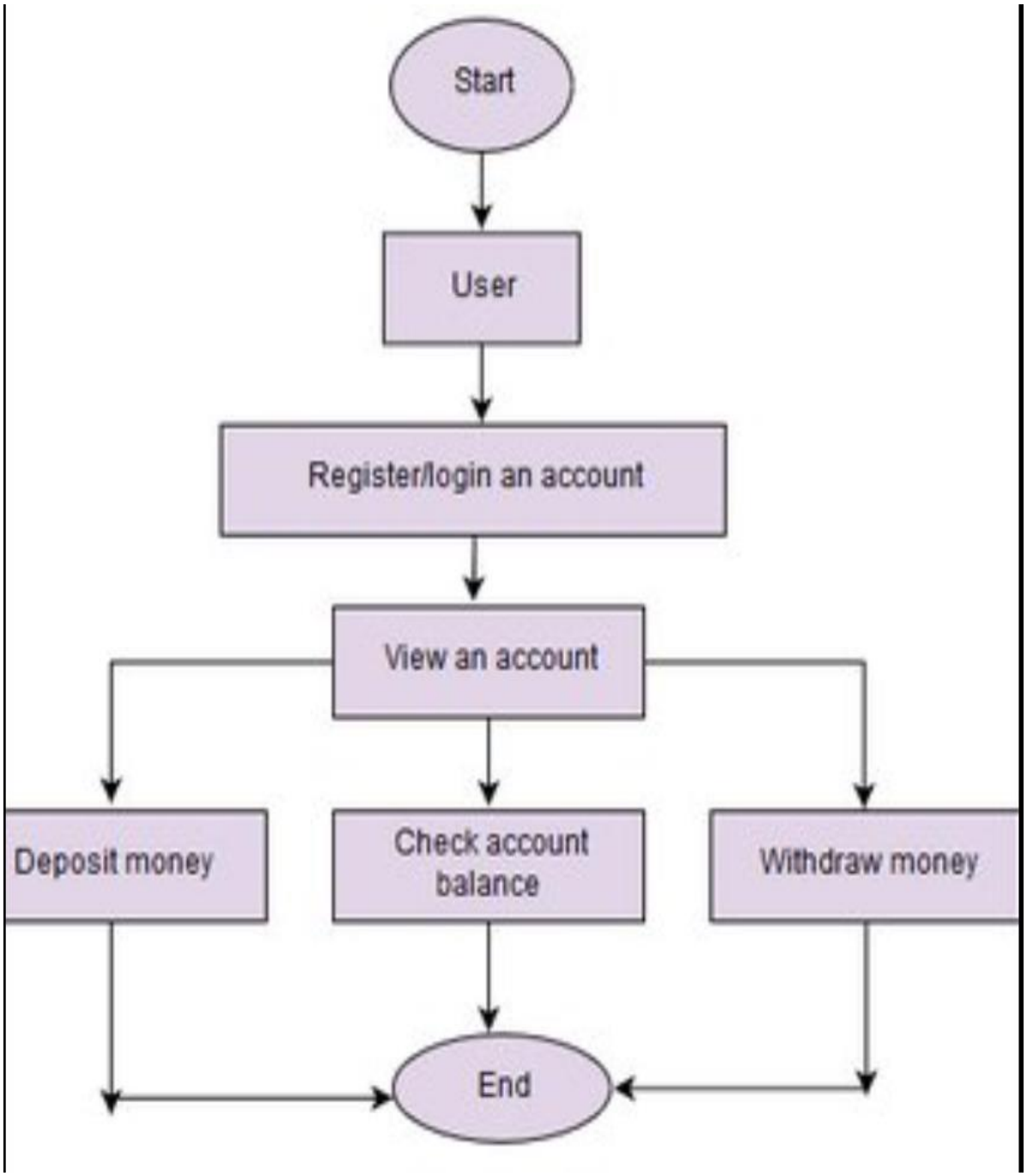
4.OBJECTIVE

- To build a simple, functional banking system in C
- To handle key banking operations: account creation, deposits, withdrawals, and statements
- To use structured programming and file handling for persistent data
- To develop a modular system with reusable and maintainable code
- To improve understanding of practical applications of C programming

5.Software Requirement Analysis

- **Operating System**: Any OS that supports a C compiler (e.g., Windows, Linux, macOS)
- **Compiler**: GCC, Clang, Turbo C, or any standard C compiler
- **IDE (Optional)**: Code::Blocks, Dev C++, VS Code, or any C-compatible edito

6.CONTROL FLOW CHART



7.Modules And Their Functionalities

This C program is a **simple banking system** that allows for account management, deposits, withdrawals, and viewing mini statements using linked lists for transaction history.

Below are the **modules (functional sections)** of the program and their **functionalities**:

🔹 Account and Transaction Structures

Functionality:

- Account stores individual user information including account number, name, balance, and a pointer to the transaction history.
- Transaction is a node in a singly linked list storing the transaction type and amount.

🔹 createAccount()

Functionality:

- Adds a new account to the system.
- Prompts for the user's name.
- Assigns a unique account number.
- Initializes balance to 0 and transaction history to NULL.
- Increases accountCount.

🔹 findAccount(int accNo)

Functionality:

- Searches for an account by account number.
- Returns a pointer to the account if found, otherwise NULL.

🔹 addTransaction(Account* acc, const char* type, float amount)

Functionality:

- Dynamically allocates and inserts a new transaction at the beginning of the linked list.
- Stores the type ("Deposit" or "Withdraw") and amount.

◈ deposit()

Functionality:

- Prompts for account number and deposit amount.
- Validates the account and amount.
- Updates the account balance.
- Records the transaction using addTransaction().

◈ withdraw()

Functionality:

- Prompts for account number and withdrawal amount.
- Validates the account and ensures sufficient balance.
- Updates the account balance.
- Records the transaction using addTransaction().

◈ showMiniStatement()

Functionality:

- Prompts for an account number.
- Displays the account holder's name, balance, and list of transactions in reverse chronological order (due to head insertion in the linked list).

◈ menu()

Functionality:

- Displays the main menu in a loop.
- Allows the user to select:
    1. Create a new account
    2. Deposit money
    3. Withdraw money
    4. View mini statement
    5. Exit the program

◈ main()

Functionality:

- Calls the menu() function to start the application.

6.Code

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

#define MAX_ACCOUNTS 100
#define NAME_LEN 50

// Transaction node (for mini statement)
typedef struct Transaction {
```

```
    char type[10];
    float amount;
    struct Transaction* next;
} Transaction;

// Account structure
typedef struct {
    int accountNumber;
    char name[NAME_LEN];
    float balance;
    Transaction* transactions; // Head of the linked list
} Account;

// Array of accounts
Account accounts[MAX_ACCOUNTS];
int accountCount = 0;

// Function to add a transaction to mini statement
void addTransaction(Account* acc, const char* type,
float amount) {
    Transaction* newTrans =
(Transaction*)malloc(sizeof(Transaction));
    strcpy(newTrans->type, type);
    newTrans->amount = amount;
    newTrans->next = acc->transactions;
    acc->transactions = newTrans;
}

// Function to create a new account
```

```
void createAccount() {
    if (accountCount >= MAX_ACCOUNTS) {
        printf("Cannot create more accounts.\n");
        return;
    }

    Account* acc = &accounts[accountCount];
    acc->accountNumber = 1000 + accountCount;
    printf("Enter name: ");
    scanf(" %[^\n]", acc->name);
    acc->balance = 0.0;
    acc->transactions = NULL;

    printf("Account created successfully! Account Number:
%d\n", acc->accountNumber);
    accountCount++;
}

// Find account by account number
Account* findAccount(int accNo) {
    for (int i = 0; i < accountCount; i++) {
        if (accounts[i].accountNumber == accNo) {
            return &accounts[i];
        }
    }
    return NULL;
}

// Deposit function
```

```
void deposit() {
    int accNo;
    float amount;
    printf("Enter account number: ");
    scanf("%d", &accNo);

    Account* acc = findAccount(accNo);
    if (acc == NULL) {
        printf("Account not found!\n");
        return;
    }

    printf("Enter amount to deposit: ");
    scanf("%f", &amount);
    if (amount <= 0) {
        printf("Invalid amount!\n");
        return;
    }

    acc->balance += amount;
    addTransaction(acc, "Deposit", amount);

    printf("Deposit successful. New balance: %.2f\n", acc-
>balance);
}

// Withdraw function
void withdraw() {
    int accNo;
```

```
    float amount;
    printf("Enter account number: ");
    scanf("%d", &accNo);

    Account* acc = findAccount(accNo);
    if (acc == NULL) {
        printf("Account not found!\n");
        return;
    }

    printf("Enter amount to withdraw: ");
    scanf("%f", &amount);
    if (amount <= 0 || amount > acc->balance) {
        printf("Invalid or insufficient amount!\n");
        return;
    }

    acc->balance -= amount;
    addTransaction(acc, "Withdraw", amount);

    printf("Withdrawal successful. New balance: %.2f\n",
acc->balance);
}

// Show mini statement
void showMiniStatement() {
    int accNo;
    printf("Enter account number: ");
    scanf("%d", &accNo);
```

```
    Account* acc = findAccount(accNo);
    if (acc == NULL) {
        printf("Account not found!\n");
        return;
    }

    printf("\nMini Statement for %s (Account No: %d)\n",
acc->name, acc->accountNumber);
    printf("Current Balance: %.2f\n", acc->balance);
    printf("Transactions:\n");

    Transaction* t = acc->transactions;
    if (t == NULL) {
        printf("No transactions yet.\n");
        return;
    }

    while (t != NULL) {
        printf("%s: %.2f\n", t->type, t->amount);
        t = t->next;
    }
}

// Main menu
void menu() {
    int choice;

    while (1) {
```

```
        printf("\n--- Banking System Menu ---\n");
        printf("1. Create Account\n");
        printf("2. Deposit\n");
        printf("3. Withdraw\n");
        printf("4. Mini Statement\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: createAccount(); break;
            case 2: deposit(); break;
            case 3: withdraw(); break;
            case 4: showMiniStatement(); break;
            case 5: printf("Thank you!\n"); exit(0);
            default: printf("Invalid choice. Try again.\n");
        }
    }
}

int main() {
    menu();
    return 0;
}
```

5. Output Screens

```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice: 1
Enter name: sivaji
Account created successfully! Account Number: 1000

--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice: 2
Enter account number: 1000
Enter amount to deposit: 5000
Deposit successful. New balance: 5000.00

--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice:
```

```
--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice: 3
Enter account number: 1000
Enter amount to withdraw: 2000
Withdrawal successful. New balance: 3000.00

--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice: 4
Enter account number: 1000

Mini Statement for sivaji (Account No: 1000)
Current Balance: 3000.00
Transactions:
Withdraw: 2000.00
Deposit: 5000.00
```

```
Mini Statement for sivaji (Account No: 1000)
Current Balance: 3000.00
Transactions:
Withdraw: 2000.00
Deposit: 5000.00

--- Banking System Menu ---
1. Create Account
2. Deposit
3. Withdraw
4. Mini Statement
5. Exit
Enter your choice: 5
Thank you!


...Program finished with exit code 0
Press ENTER to exit console.
```

8. Conclusion

This banking application performs the main functions needed to manage a bank account. It allows users to:

- ❖ Create a new account with personal details.
- ❖ Deposit money into their account.
- ❖ Withdraw money with balance checks.
- ❖ View transaction history for all deposits and withdrawals.

The system works correctly by updating balances after each transaction and saving the records for future reference. It is easy to use, well-organized, and can be improved later by adding features like login security, interest calculation, or online transfers. This makes it a strong basic model for a real banking system.