BANK ACCOUNT MANAGEMENT SYSTEM

21CSS101J – PROGRAMMING FOR PROBLEM-SOLVING

Mini Project Report

Submitted by

Kaushtubh Kumar [Reg. No.: RA2311003010402]
B.Tech. CSE - CORE

SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act, 1956)
S.R.M. NAGAR, KATTANKULATHUR – 603 203
CHENGALPATTU DISTRICT

November 2023

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act, 1956)
S.R.M. NAGAR, KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that Mini project report titled Bank Account Management System is the bonafide work of Reg.No RA2311003010402 Name Kaushtubh Kumar who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr.R.Deepa

Assistant Professor

Dept of Computing Technologies

SIGNATURE

Dr.Pushpalatha M

Professor

Head of Department

Dept of Computing Technologies

TABLE OF CONTENTS

S No.	Title	Page No.
1	Problem Statement	4
2	Methodology / Procedure/ Algorithm	5
3	Flowchart	6
4	Coding (C/Python)	7
5	Front-end code (HTML, CSS, Javascript) [Optional]	15
6	Modules of the proposed work	16
7	Results/Screenshots	17
8	Conclusion	21
9	References	22

1. Problem Statement

Bank Account Management System in C: Mini Project

Create a Bank Account Management System in C, focusing on efficiency and user-friendly functionalities. The system should allow users to open new accounts, deposit and withdraw money, check balances, view detailed account information, update account details, and close accounts. Utilize file handling for persistent storage, ensuring data integrity. Implement modular functions to organize code logically, and employ error handling for input validation and transaction integrity. The system should feature a menu-driven interface for user interaction. Ensure each account has a unique identifier and use appropriate data structures for efficient data management. Provide an option to display a list of all accounts with relevant details. The successful implementation of this mini project should showcase a balance between simplicity, functionality, and adherence to programming best practices. Deliverables include well-commented source code, a concise project report, and a set of test cases demonstrating the system's reliability and correctness.

2. Algorithm

Here's a basic algorithm for a bank account management system:

1. Start.

2. Display Menu:
   - Display options for the user, such as:
     - 1. Display Balance
     - 2. Deposit
     - 3. Withdraw
     - 4. Exit

3. User Input:

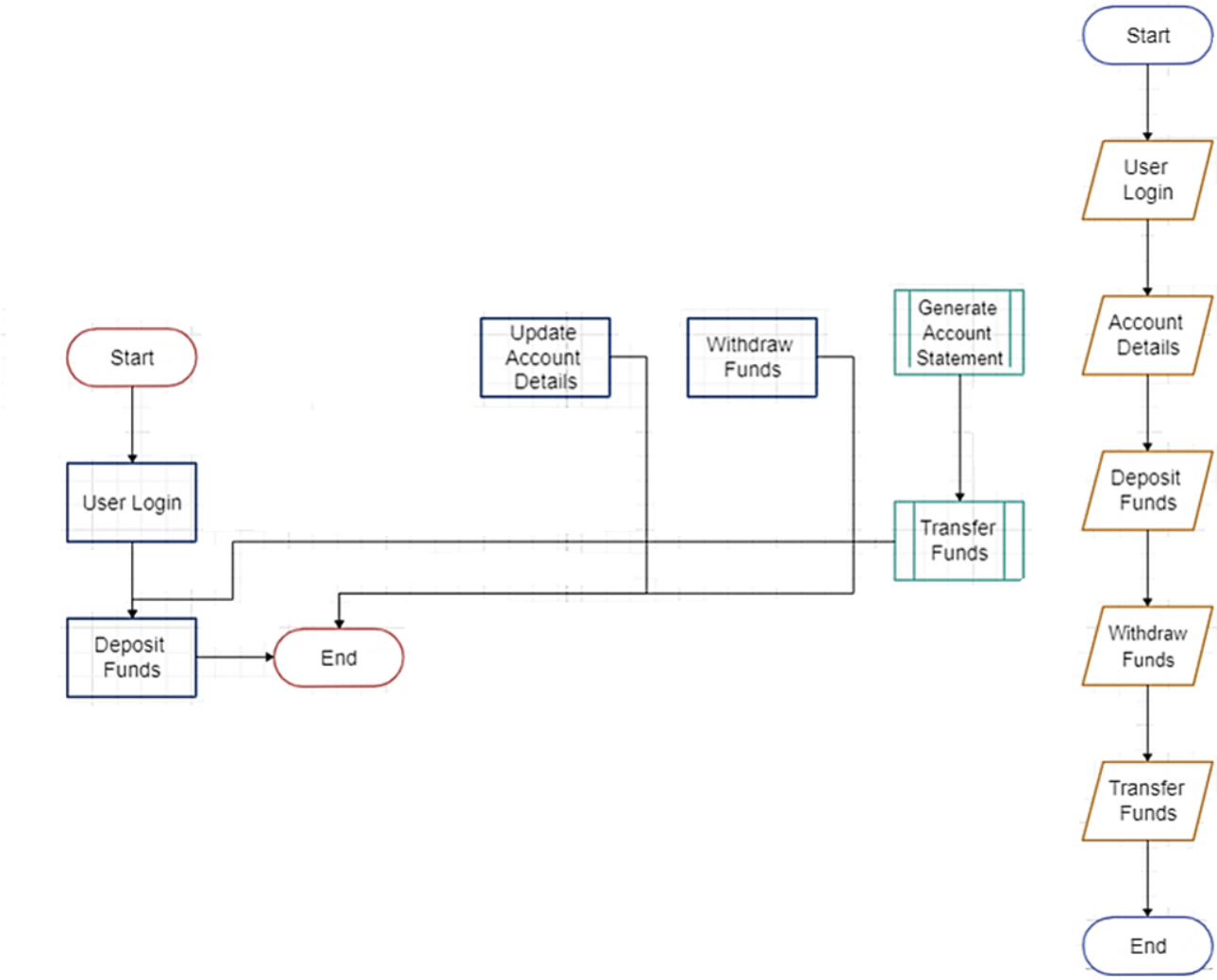
4. Process User Choice:
   - If choice =1:
     - Display account balance.
   - If choice =2:
     - Prompt the user to enter the deposit amount.
     - Update the account balance.
   - If choice=3:
     - Prompt the user to enter the withdrawal amount.
     - Check if the withdrawal amount is within the available balance.
     - Update the account balance if the withdrawal is valid.
   - If choice =4:
     - Exit the program.

5. Display Result:
   - Display the updated account balance or appropriate messages.

6. Repeat or Exit:
   - Ask the user if they want to perform another transaction.
   - If yes, go back to the menu; if no, exit the program.

7. End:
   - Terminate the program.

3. Flow chart



Bank Account Management System

4. Coding (C/Python)

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

struct Account {
    int accNumber;
    char holderName[50];
    float balance;
};

// Function prototypes
void createAccount();
void depositMoney();
void withdrawMoney();
void checkBalance();
void displayAccountDetails();
void updateAccount();
void deleteAccount();
void displayAllAccounts();

int main() {
    int choice;

    do {
        printf("\nBank Account Management System\n");
        printf("1. Create Account\n");
        printf("2. Deposit Money\n");
        printf("3. Withdraw Money\n");
        printf("4. Check Balance\n");
        printf("5. Display Account Details\n");
        printf("6. Update Account\n");
        printf("7. Delete Account\n");
        printf("8. Display All Accounts\n");
```

```
        printf("0. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createAccount();
                break;
            case 2:
                depositMoney();
                break;
            case 3:
                withdrawMoney();
                break;
            case 4:
                checkBalance();
                break;
            case 5:
                displayAccountDetails();
                break;
            case 6:
                updateAccount();
                break;
            case 7:
                deleteAccount();
                break;
            case 8:
                displayAllAccounts();
                break;
            case 0:
                printf("Exiting program. Goodbye!\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 0);

    return 0;
}

void createAccount() {
    struct Account newAccount;
    FILE *file;

    file = fopen("accounts.dat", "ab");
```

```
    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &newAccount.accNumber);
    printf("Enter Holder Name: ");
    scanf("%s", newAccount.holderName);
    printf("Enter Initial Balance: ");
    scanf("%f", &newAccount.balance);

    fwrite(&newAccount, sizeof(struct Account), 1, file);

    printf("Account created successfully.\n");

    fclose(file);
}

void depositMoney() {
    int accNumber;
    float amount;
    FILE *file;

    file = fopen("accounts.dat", "r+b");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Enter Deposit Amount: ");
        scanf("%f", &amount);
```

```
        currentAccount.balance += amount;

        fseek(file, -sizeof(struct Account), SEEK_CUR);
        fwrite(&currentAccount, sizeof(struct Account), 1, file);

        printf("Deposit successful. New balance: %.2f\n", currentAccount.balance);
    } else {
        printf("Account not found.\n");
    }

    fclose(file);
}

void withdrawMoney() {
    int accNumber;
    float amount;
    FILE *file;

    file = fopen("accounts.dat", "r+b");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Enter Withdrawal Amount: ");
        scanf("%f", &amount);

        if (currentAccount.balance >= amount) {
            currentAccount.balance -= amount;

            fseek(file, -sizeof(struct Account), SEEK_CUR);
```

```
            fwrite(&currentAccount, sizeof(struct Account), 1, file);

            printf("Withdrawal successful. New balance: %.2f\n", currentAccount.balance);
        } else {
            printf("Insufficient balance.\n");
        }
    } else {
        printf("Account not found.\n");
    }

    fclose(file);
}

void checkBalance() {
    int accNumber;
    FILE *file;

    file = fopen("accounts.dat", "rb");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Account Balance: %.2f\n", currentAccount.balance);
    } else {
        printf("Account not found.\n");
    }

    fclose(file);
}

void displayAccountDetails() {
```

```
    int accNumber;
    FILE *file;

    file = fopen("accounts.dat", "rb");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Account Details\n");
        printf("Account Number: %d\n", currentAccount.accNumber);
        printf("Holder Name: %s\n", currentAccount.holderName);
        printf("Balance: %.2f\n", currentAccount.balance);
    } else {
        printf("Account not found.\n");
    }

    fclose(file);
}

void updateAccount() {
    int accNumber;
    FILE *file;

    file = fopen("accounts.dat", "r+b");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
```

```
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Enter new Holder Name: ");
        scanf("%s", currentAccount.holderName);

        fseek(file, -sizeof(struct Account), SEEK_CUR);
        fwrite(&currentAccount, sizeof(struct Account), 1, file);

        printf("Account updated successfully.\n");
    } else {
        printf("Account not found.\n");
    }

    fclose(file);
}

void deleteAccount() {
    int accNumber;
    FILE *file, *tempFile;

    file = fopen("accounts.dat", "rb");
    tempFile = fopen("temp.dat", "wb");

    if (file == NULL || tempFile == NULL) {
        printf("Error opening file.\n");
        return;
    }

    printf("Enter Account Number: ");
    scanf("%d", &accNumber);

    struct Account currentAccount;
    int found = 0;

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        if (currentAccount.accNumber == accNumber) {
```

```
            found = 1;
        } else {
            fwrite(&currentAccount, sizeof(struct Account), 1, tempFile);
        }
    }

    fclose(file);
    fclose(tempFile);

    remove("accounts.dat");
    rename("temp.dat", "accounts.dat");

    if (found) {
        printf("Account deleted successfully.\n");
    } else {
        printf("Account not found.\n");
    }
}
void displayAllAccounts() {
    FILE *file;

    file = fopen("accounts.dat", "rb");

    if (file == NULL) {
        printf("Error opening file.\n");
        return;
    }
    struct Account currentAccount;
    printf("\nAll Accounts\n");
    printf("-----------------------------------\n");
    printf("Account Number\tHolder Name\tBalance\n");
    printf("-----------------------------------\n");

    while (fread(&currentAccount, sizeof(struct Account), 1, file) == 1) {
        printf("%d\t\t%s\t\t%.2f\n", currentAccount.accNumber, currentAccount.holderName,
currentAccount.balance);
    }

    printf("-----------------------------------\n");

    fclose(file);}
```

5. Front-end code (HTML)

```
<!DOCTYPE html>
<html>
<head>
  <title>Bank Account Management</title>
</head>
<body>

<h1>Welcome to Bank Account Management</h1>

<h2>Create Account</h2>
<form action="account_management" method="post">
  Account Number: <input type="number" name="accountNumber"><br>
  Name: <input type="text" name="name"><br>
  Initial Balance: <input type="number" step="0.01" name="balance"><br>
  <input type="submit" value="Create Account">
</form>

<h2>Deposit</h2>
<form action="account_management" method="post">
  Account Number: <input type="number" name="accountNumber"><br>
  Amount to Deposit: <input type="number" step="0.01" name="amount"><br>
  <input type="hidden" name="action" value="deposit">
  <input type="submit" value="Deposit">
</form>

<h2>Withdraw</h2>
<form action="account_management" method="post">
  Account Number: <input type="number" name="accountNumber"><br>
  Amount to Withdraw: <input type="number" step="0.01" name="amount"><br>
  <input type="hidden" name="action" value="withdraw">
  <input type="submit" value="Withdraw">
</form>

<h2>Check Balance</h2>
<form action="account_management" method="post">
  Account Number: <input type="number" name="accountNumber"><br>
  <input type="hidden" name="action" value="checkBalance">
  <input type="submit" value="Check Balance">
</form>
```

```
    </body>
    </html>
_____
```

6. Modules of the proposed work

In a C program for a bank account management system, you can organize the functionality into several modules to promote modularity, readability, and maintainability. Here's a suggestion for potential modules:

1. createAccount():
   - Function to create a new bank account.
   - Collects necessary information from the user, such as name, address, and initial deposit.
   - Generates a unique account number for the new account.

2. depositMoney():
   - Allows the user to deposit money into their existing account.
   - Takes the account number and the amount to deposit as input.
   - Updates the account balance accordingly.

3. withdrawMoney():
   - Enables the user to withdraw money from their account.
   - Requires the account number and the withdrawal amount as input.
   - Verifies if the withdrawal is possible based on the account balance.

4. checkBalance():
   - Displays the current balance of a specific account.
   - Takes the account number as input and retrieves the corresponding balance.

5. displayAccountDetails():
   - Shows detailed information about a specific account.
   - Requires the account number as input and displays details like account holder name, address, and balance.

6. updateAccount():
   - Allows the user to update account information.
   - Takes the account number as input and provides options to modify details like name, address, or contact information.

7. deleteAccount():
   - Permanently removes an existing account.
   - Takes the account number as input and deletes the associated account record.

8. displayAllAccounts():
   - Lists details of all accounts in the system.
   - Displays information such as account number, account holder name, and balance for each existing account.```

7. Results/Screenshots

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 1
Enter Account Number: 1003
Enter Holder Name: Arjun
Enter Initial Balance: 15500
Account created successfully.
```

Create an account.

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 2
Enter Account Number: 1003
Enter Deposit Amount: 500
Deposit successful. New balance: 16000.00
```

Deposit money in Account.

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 3
Enter Account Number: 1003
Enter Withdrawal Amount: 1000
Withdrawal successful. New balance: 15000.00
```

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 4
Enter Account Number: 1003
Account Balance: 15000.00
```

Withdraw money from account and Check Balance from Account.

To Display Account Details.

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 5
Enter Account Number: 1003
Account Details
Account Number: 1003
Holder Name: Arjun
Balance: 15000.00
```

To Update an Account Information.

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 6
Enter Account Number: 1003
Enter new Holder Name: Arjuna
Account updated successfully.
```

To Delete an Account .

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 7
Enter Account Number: 1003
Account deleted successfully.
```

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 8

All Accounts
-------------------------------------------------
Account Number  Holder Name     Balance
-------------------------------------------------
1001            Abhik           15000.00
-------------------------------------------------
```

To Display all Accounts Saved in the Binary File.

```
Bank Account Management System
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. Display Account Details
6. Update Account
7. Delete Account
8. Display All Accounts
0. Exit
Enter your choice: 0
Exiting program. Goodbye!
PS C:\Users\abhik\OneDrive\Documents\Desktop\PPS PROJECT>
```

To Exit from the program.

Frontend Using Html

Welcome to Bank Account Management

Create Account

Account Number:
Name:
Initial Balance:
Create Account

Deposit

Account Number:
Amount to Deposit:
Deposit

Withdraw

Account Number:
Amount to Withdraw:
Withdraw

Check Balance

Account Number:
Check Balance

8. Conclusion

The bank account management project in C demonstrates a successful application of programming principles, employing modular design, encapsulation, and abstraction for a robust and user-friendly system. Core features include user authentication, transaction handling, and a well-designed user interface. The project incorporates essential functionalities like balance inquiry, deposit, withdrawal, and transaction history, ensuring comprehensive account management. Design decisions, such as using appropriate data structures and error handling, enhance system reliability. Optional modules like file handling and reporting contribute to scalability. Adherence to coding standards promotes code quality, reusability, and ease of maintenance, establishing a solid foundation for potential integration into larger financial systems. Overall, the project showcases effective software engineering in C, providing a versatile solution for managing bank accounts.

9. References

[1] "C Programming Absolute Beginner's Guide", Perry, Perry, and Mille , 3rd Edition , Year of Publication:2014 , ISBN-13: 978-0-7897-5211-4

[2] "C Programming for the Absolute Beginner, Second Edition" , Vine, Vine, and Waite , 2nd Edition , Year of Publication:2007 , ISBN-13:978-1-59863-339-4

[3] "C Programming Language" , Brian W. Kernighan and Dennis M. Ritchie , 2nd Edition , Year of Publication:1988 , ISBN-13: 978-0131103627

[4] GeeksforGeeks (https://www.geeksforgeeks.org/c-programming-language/)

[5] Learn-C.org (https://www.learn-c.org/)

[6] Cprogramming.com (https://www.cprogramming.com/)