PROJECT TITLE: BANKING MANAGEMENT SYSTEM IN C

NAME- PIYUSH RALHAN & DEV SAWHNEY

SUBJECT- PROGRAMMING IN C

SAP ID- 590022305 & 590022263

SUBMITTED TO: Dr. Neeraj Chugh

DATE – 29 NOVEMBER 2025

BATCH-23

COURSE- BTECH CSE

GITHUB REPO FOR CODE- Piyush-ralhan/BANKMANAGEMENTSYSTEM: A console-based Banking Management System in C that manages accounts, transactions, and data storage using file handling and modular programming. Built as part of my UPES C Programming course, showcasing skills in problem-solving, structures, and system design.

Bank Management System Report

A Project in C Programming

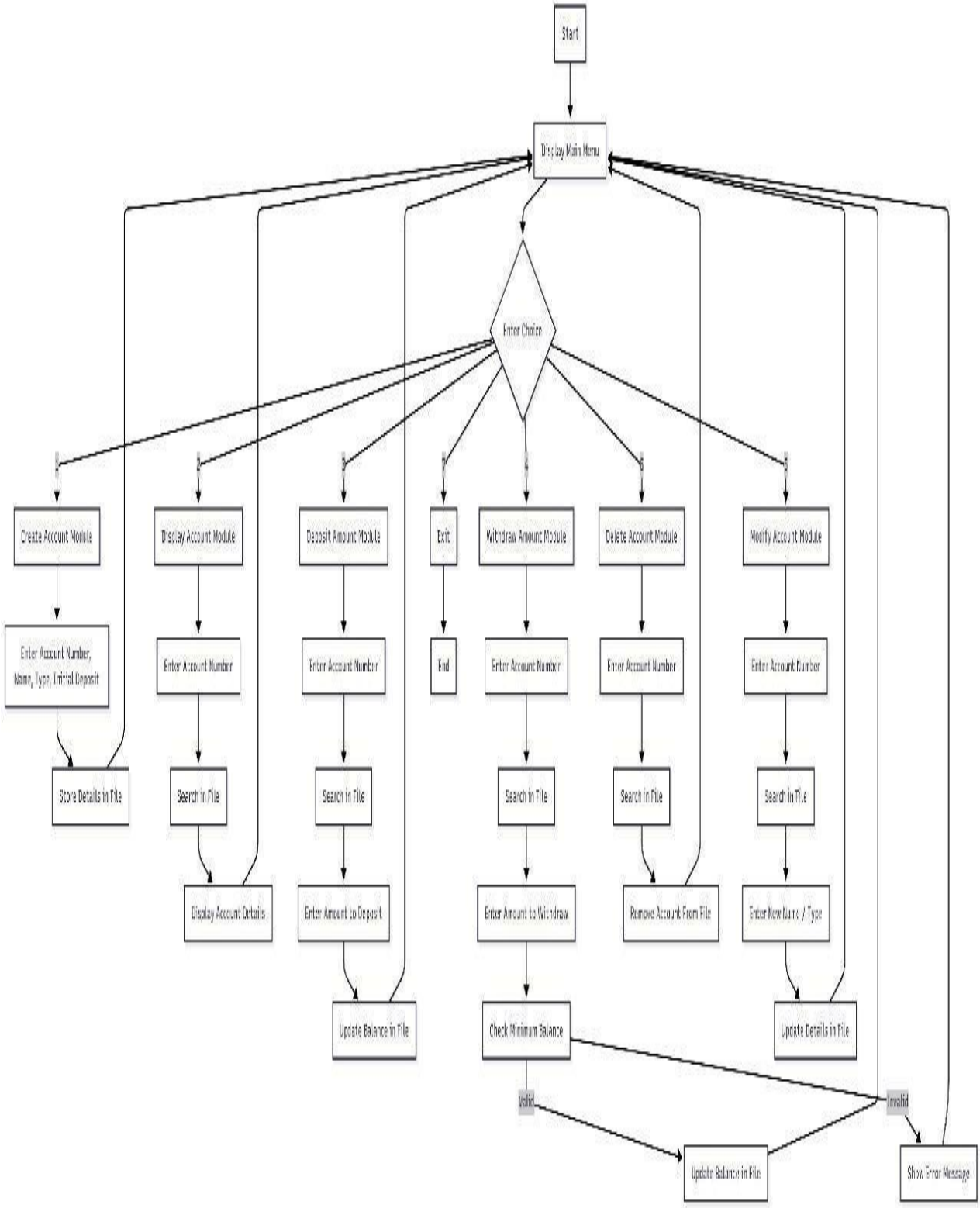
ABSTARCT

This project presents a Bank Management System developed in the C programming language. The system allows operations such as creating accounts, displaying account details, depositing and withdrawing money, modifying account information, and deleting accounts. The objective of this project is to demonstrate file handling, structures, modular programming, and basic data management techniques in C.

PROBLEM DEFINITION

The purpose of this project is to implement a simple banking system capable of performing essential financial operations. The program maintains records using binary files and provides users with a menu-driven interface to manage account information securely and efficiently.

FLOWCHART


Start
Display Main Menu
Enter Choice
1
2
3
4
5
6
Create Account Module
Display Account Module
Deposit Amount Module
Exit
Withdraw Amount Module
Delete Account Module
Modify Account Module
Enter Account Number, Name, Type, Initial Deposit
Enter Account Number
Enter Account Number
End
Enter Account Number
Enter Account Number
Enter Account Number
Store Details in File
Search in File
Search in File
Search in File
Search in File
Search in File
Display Account Details
Enter Amount to Deposit
Enter Amount to Withdraw
Remove Account From File
Enter New Name / Type
Update Balance in File
Check Minimum Balance
Update Details in File
Valid
Invalid
Update Balance in File
Show Error Message

ALGORITHM

1. Start of Program

1. Begin program execution.
2. Load header files and structure definitions.
3. Set FILENAME = "account.dat" for storing all account records.

2. Main Menu Loop

. Display the main menu:

1. Create Account
2. Display Account 3. Deposit Amount
4. Withdraw Amount
5. Modify Account
6. Delete Account
7. Exit

2. Accept user's choice in variable choice.
3. Execute task using a switch(choice):

3. Create Account Module

Function: create_account()

1. Open the file account.dat in append binary mode.
2. Input account number from user.

3. Call account_exists(acc_no):

-
-

Open file

Search for matching account number

- If found, return 1; else return 0

4. If account already exists → display message and stop.

5. Otherwise:

- Read name (string)
- Read account type (S or C) • Read initial deposit amount

6. Validate:

- Type must be S or C
- Balance must not be negative

7. Write the Account structure into the account.dat file.

8. Close file.

9. Display "Account created successfully".

4. Display Account Module

Function: display_account()

1. Ask for account number.

-
-

2. Open account.dat in read mode.
3. Loop through file: • Read each account structure
- If account number matches:
- Display:
- Account number
- Name

  Type

  Balance
- Set found = 1

4. If not found after loop → display “Account not found”.
5. Close the file.

5. Deposit / Withdraw Module

Function: deposit_withdraw(int is_deposit)

1. Open account.dat in read+write mode.
2. Ask user for account number.
3. Loop through file:

-
-

- Read each account
- If account matches:
- Display name and balance
- Ask for amount
- Validate amount must be positive
- If is_deposit == 1:
- Add amount to balance
- Else (withdraw):
- Check if amount ≤ balance
- If not, show “Insufficient Balance”
- Else subtract amount
- Move file pointer back by size of structure
  Overwrite updated structure

Show “Deposit/Withdrawal successful” 4.

If account not found → display message.

5. Close the file.

6. Modify Account Module

-
-

Function: modify_account()

1. Ask for account number.

2. Open account.dat in read+write mode.
3. Loop through file until matching account is found.
4. For the found account:

- Display existing Name, Type & Balance
- Ask new name (skip if blank)
- Ask new type (skip if invalid)
- Ask new balance (skip if -1)

5. Move file pointer back.
6. Rewrite modified account.
7. Display “Account updated successfully”.
8. If not found → print “Account not found”.
9. Close file.

7. Delete Account Module

Function: delete_account()

1. Ask for account number.
2. Open account.dat in read mode.
3. Open temp.txt in write mode.
4. Loop through all accounts:

- If account number matches → skip writing (delete)
- Else → write to temp file

5. Close both files.
6. If found:

- Delete account.dat
- Rename temp.txt → account.dat
- Display “Account deleted”

7. Else:

- Delete temp file
- Display “Account not found”

8. Exit Program

1. If user chooses option 7:

- Display “Thankyou for visiting.”
- Terminate program using exit(0).

9. End

IMPLEMENTATION OF CODE SNIPPET

The Bank Management System is implemented using a modular structure, where each operation such as creating an account, depositing money, withdrawing, modifying, displaying, or deleting an account is written in a separate C file with its own header. A binary file (account.dat) is used to store all account records, ensuring secure and permanent data storage. The program reads and writes account structures using fread() and fwrite(), and updates records by seeking the correct file position with fseek(). Account validation is performed through the account_exists() function, while clear_stdin() ensures clean input handling.

MAIN MENU

The main menu displays the available operations and calls the corresponding functions using switch-case:

```c
int main()
{
    int choice;
    while (1)
    {
        printf("\n=========================\n");
        printf("    BANK MANAGEMENT\n");
        printf("=========================\n");
        printf("1. Create new account\n");
        printf("2. Display account\n");
        printf("3. Deposit amount\n");
        printf("4. Withdraw amount\n");
        printf("5. Modify account\n");
        printf("6. Delete account\n");
        printf("7. Exit\n");
        printf("-------------------------\n");
        printf("Enter your choice: ");

        scanf("%d", &choice);
        getchar();
        switch (choice)
        {
        case 1:
            create_account();
            break;
        case 2:
            display_account();
            break;
        case 3:
            deposit_withdraw(1);
            break;
        case 4:
            deposit_withdraw(0);
            break;
        case 5:
            modify_account();
            break;
        case 6:
            delete_account();
            break;
        case 7:
            printf("Thankyou for visiting.\n");
            exit(0);
        default:
            printf("Invalid choice. Try 1-7.\n");
        }
    }
    return 0;
}
```

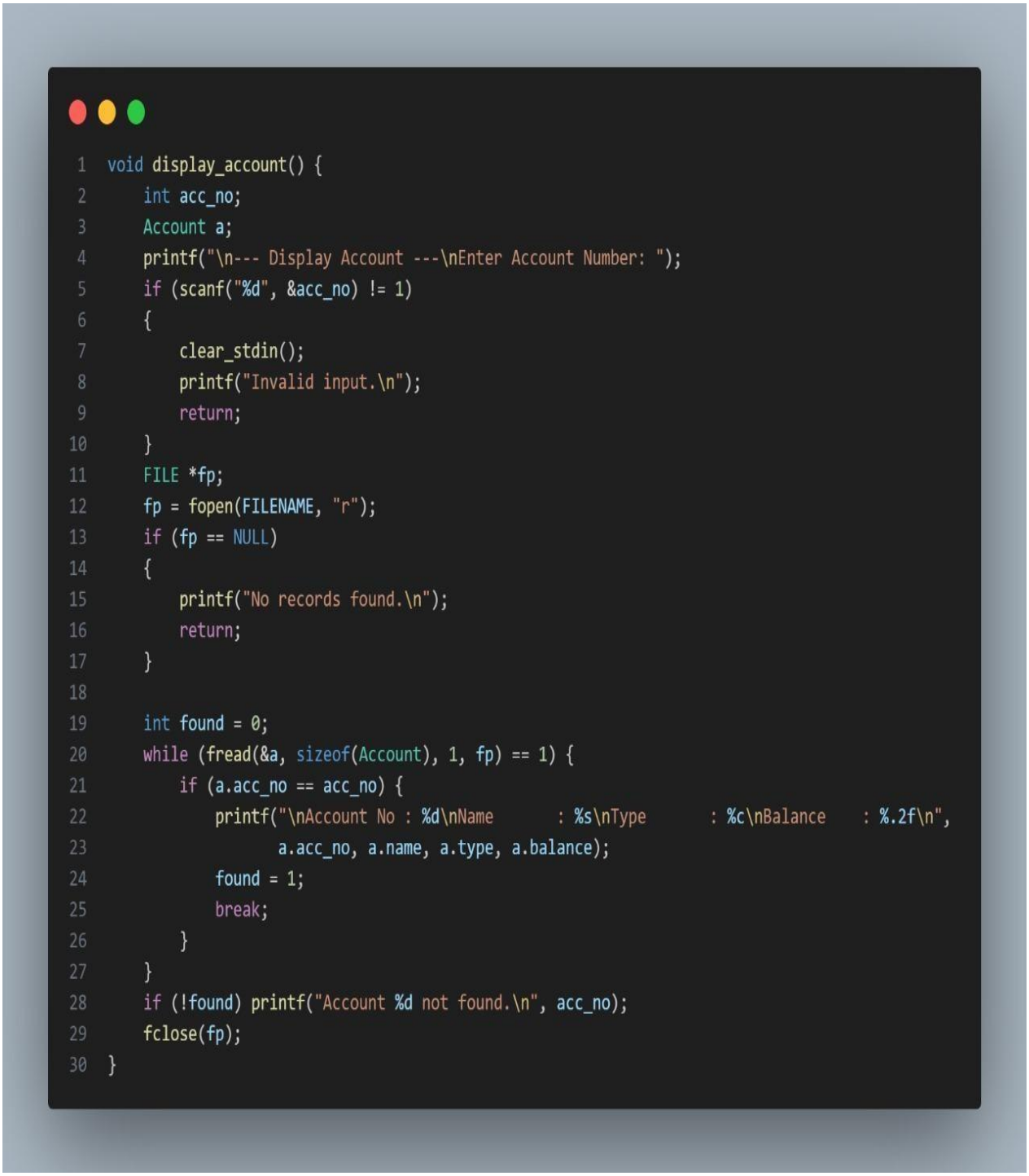
CREATE ACCOUNT

Creates a new folder at the specified path :

```c
void create_account(void) {
    FILE *fp = fopen(FILENAME, "ab");
    if (fp == NULL)
    {
        printf("File open error");
        return;
    }
    Account a;
    printf("\n--- Create New Account ---\n");
    printf("Enter Account Number: ");
    if (scanf("%d", &a.acc_no) != 1)
    {
        printf("Invalid input.\n");
        return;
    }
    if (account_exists(a.acc_no))
    {
        printf("Account %d already exists.\n", a.acc_no);
        return;
    }
    fflush(stdin);
    printf("Enter the Name: ");
    fgets(a.name,50 , stdin);
    printf("Account Type (S for Savings / C for Current): ");
    a.type = getchar();
    clear_stdin();
    if (a.type != 'S' && a.type != 'C' && a.type != 's' && a.type != 'c') {
        printf("Invalid account type. Use S or C.\n");
        return;
    }
    printf("Initial Deposit Amount: ");
    if (scanf("%lf", &a.balance) != 1)
    {
        printf("Invalid amount.\n");
        return;
    }
    if (a.balance < 0)
    {
        printf("Balance cannot be negative.\n");
        return;
    }


    fwrite(&a, sizeof(Account), 1, fp);
    fclose(fp);

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

```

DISPLAY ACCOUNT

By choosing this option we can display account details:



```c
void display_account() {
    int acc_no;
    Account a;
    printf("\n--- Display Account ---\nEnter Account Number: ");
    if (scanf("%d", &acc_no) != 1)
    {
        clear_stdin();
        printf("Invalid input.\n");
        return;
    }
    FILE *fp;
    fp = fopen(FILENAME, "r");
    if (fp == NULL)
    {
        printf("No records found.\n");
        return;
    }

    int found = 0;
    while (fread(&a, sizeof(Account), 1, fp) == 1) {
        if (a.acc_no == acc_no) {
            printf("\nAccount No : %d\nName       : %s\nType       : %c\nBalance    : %.2f\n",
                   a.acc_no, a.name, a.type, a.balance);
            found = 1;
            break;
        }
    }
    if (!found) printf("Account %d not found.\n", acc_no);
    fclose(fp);
}
```

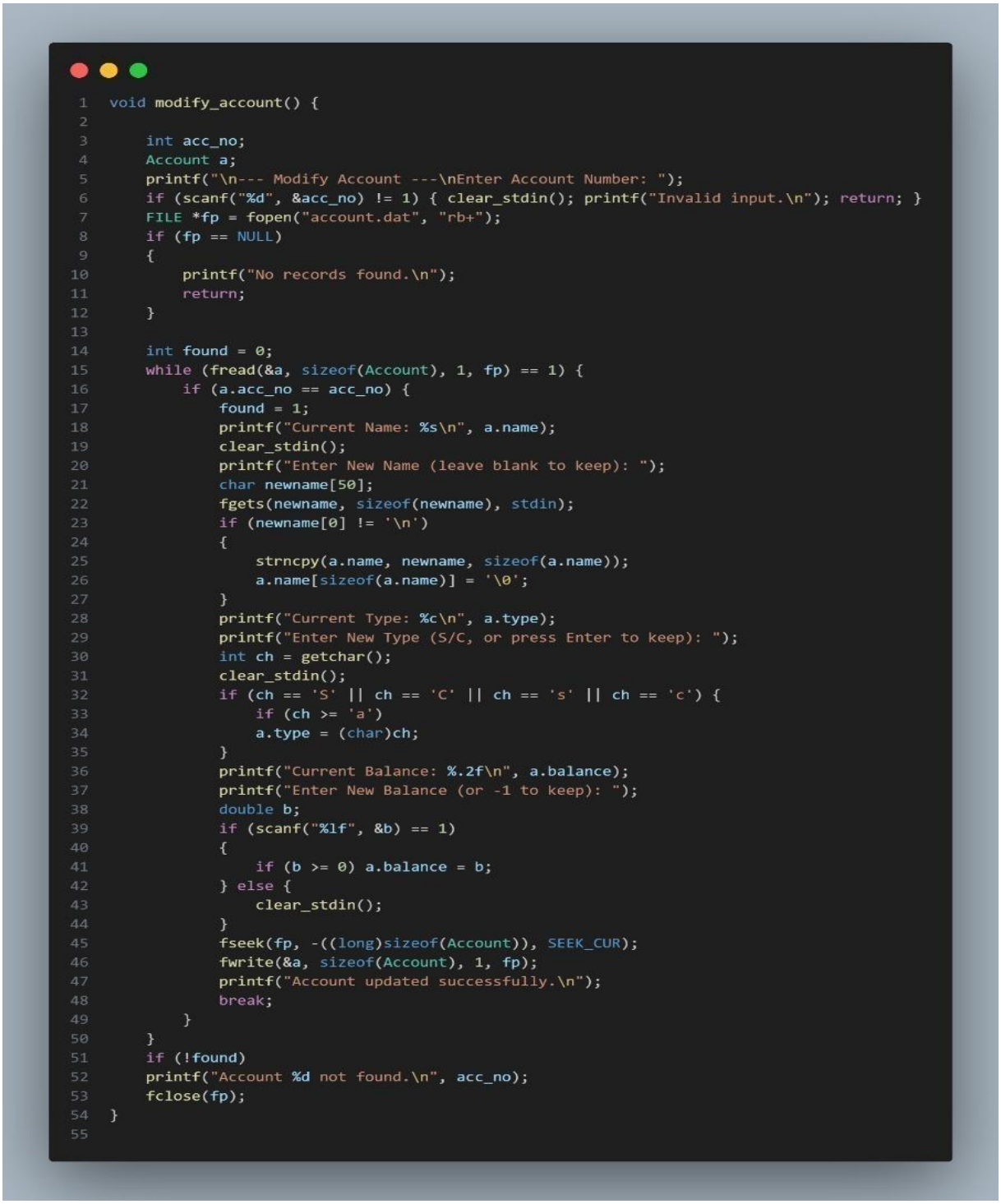
DEPOSIT WITHDRAWL

By choosing withdrawl option we can withdraw money from the certain account:

```c
void deposit_withdraw(int is_deposit)
{
    double amt;
    FILE *fp = fopen(FILENAME, "rb+");
     if (fp== NULL)
    {
        printf("No records found.\n");
        return;
    }
    int acc_no;
    printf("\n--- %s ---\n", is_deposit ? "Deposit" : "Withdraw");
    printf("Enter Account Number: ");
    if (scanf("%d", &acc_no) != 1)
    {
        printf("Invalid input.\n");
        return;
    }
    Account a;
    int found = 0;
    while (fread(&a, sizeof(Account), 1, fp) == 1) {
        if (a.acc_no == acc_no) {
            found = 1;
            printf("Account found: %s | Balance: %.2f\n", a.name, a.balance);
            printf("Enter amount: ");

            if (scanf("%lf", &amt) != 1) { clear_stdin(); printf("Invalid amount.\n"); break; }
            if (amt <= 0)
            {
                printf("Amount must be positive.\n");
                break;
            }
            if (is_deposit)
            {
                a.balance += amt;
            } else
            {
                if (amt > a.balance) {
                    printf("Insufficient balance.\n");
                    break;
                }
                a.balance -= amt;
            }
            // write back
            fseek(fp, -((long)sizeof(Account)), SEEK_CUR);
            fwrite(&a, sizeof(Account), 1, fp);
            printf("%s successful. New balance: %.2f\n", is_deposit ? "Deposit" : "Withdrawal", a.balance);
            break;
        }
    }
    if (!found) printf("Account %d not found.\n", acc_no);
    fclose(fp);
}
```

MODIFY ACCOUNT

By choosing modify account we can modify account details:



```c
void modify_account() {

    int acc_no;
    Account a;
    printf("\n--- Modify Account ---\nEnter Account Number: ");
    if (scanf("%d", &acc_no) != 1) { clear_stdin(); printf("Invalid input.\n"); return; }
    FILE *fp = fopen("account.dat", "rb+");
    if (fp == NULL)
    {
        printf("No records found.\n");
        return;
    }

    int found = 0;
    while (fread(&a, sizeof(Account), 1, fp) == 1) {
        if (a.acc_no == acc_no) {
            found = 1;
            printf("Current Name: %s\n", a.name);
            clear_stdin();
            printf("Enter New Name (leave blank to keep): ");
            char newname[50];
            fgets(newname, sizeof(newname), stdin);
            if (newname[0] != '\n')
            {
                strncpy(a.name, newname, sizeof(a.name));
                a.name[sizeof(a.name)] = '\0';
            }
            printf("Current Type: %c\n", a.type);
            printf("Enter New Type (S/C, or press Enter to keep): ");
            int ch = getchar();
            clear_stdin();
            if (ch == 'S' || ch == 'C' || ch == 's' || ch == 'c') {
                if (ch >= 'a')
                a.type = (char)ch;
            }
            printf("Current Balance: %.2f\n", a.balance);
            printf("Enter New Balance (or -1 to keep): ");
            double b;
            if (scanf("%lf", &b) == 1)
            {
                if (b >= 0) a.balance = b;
            } else {
                clear_stdin();
            }
            fseek(fp, -((long)sizeof(Account)), SEEK_CUR);
            fwrite(&a, sizeof(Account), 1, fp);
            printf("Account updated successfully.\n");
            break;
        }
    }
    if (!found)
    printf("Account %d not found.\n", acc_no);
    fclose(fp);
}

```

DELETE ACCOUNT

By choosing delete account we can delete account details:

```c
int delete_account() {
    int acc_no;
    printf("\n--- Delete Account ---\nEnter Account Number: ");
    if (scanf("%d", &acc_no) != 1) {
        clear_stdin(); printf("Invalid input.\n");
        return -1;
    }
    FILE *fp;
    fp =fopen("account.dat", "rb");
    if (fp == NULL)
    {
        printf("No records found.\n");
        return 0;
    }
    FILE *tmp = fopen("temp.txt", "wb");
    if (tmp == NULL) {
        printf("Temp file error");
        fclose(fp);
        return -1;
    }
    Account a;
    int found = 0;
    while (fread(&a, sizeof(Account), 1, fp) == 1) {
        if (a.acc_no == acc_no) {
            found = 1;
            continue;
        }
        fwrite(&a, sizeof(Account), 1, tmp);
    }
    fclose(fp);
    fclose(tmp);
    if (found) {
        remove("account.dat");
        rename("temp.txt", "account.dat");
        printf("Account %d deleted.\n", acc_no);
    } else {
        remove("temp.txt");
        printf("Account %d not found.\n", acc_no);
    }
    return 0;
}

```

Testing and Results

Test Case	Input Provided	Expected Output	Actual Result	Status
1. Create Account	Account No: 101 Name: Rahul Type: S Balance: 5000	Account created successfully	Account created successfully	✓ Passed
2. Display Account	Account No: 101 Acc No: 101	Show full account details	Displayed correct details	✓ Passed
3. Deposit Amount	Acc No: 101 Amount: 1500	Balance should increase	Balance updated correctly	✓ Passed
4. Withdraw Amount	Acc No: 101 Amount: 1000	Deduction should happen	Withdrawal successful	✓ Passed
5. Modify Account	Change name/ type/balance	Updated fields saved	Updated successfully	✓ Passed

CONCLUSION

The Bank Management System successfully provides an efficient and reliable way to manage customer accounts through simple menu-driven operations. By using file handling in C, the system ensures safe storage and easy retrieval of account data while supporting essential banking features like deposits, withdrawals, and updates. The project demonstrates strong implementation of structured programming concepts and serves as a practical foundation for building more advanced banking applications in the future.

.

FUTURE WORK

Possible improvements for future development include:

- Adding password authentication for account security.
- Implementing transaction history.
- Adding interest calculation.
- Creating a graphical user interface (GUI).