CCP – Report

PF (CT-175)

Team Name:

CodeCubed

Group Members:

Arbish Tehseen (CT-25053)

Aatiqa Batool (CT-25056)

Saita Ahuja (CT-25065)

Discipline : BCIT

Teacher :

Sir Muhammad Abdullah

Project Title

Bank Management System.

Abstract

The **Bank Management System** is a C-based project that brings essential banking operations to a simple console interface. It empowers admins to manage accounts and lets users perform transactions like deposits, withdrawals, transfers, and balance checks securely. With features like mini statements, interest calculation, and account analysis, the system combines practicality with simplicity—showcasing how real-world banking can be efficiently simulated through programming logic.

1. Introduction :

The **Bank Management System** is a console-based C project that simulates essential banking operations in a simple yet efficient way. It allows users to manage multiple accounts with secure options for deposits, withdrawals, transfers, and balance inquiries. With password protection, safeguards against invalid transactions, and organized account tracking through arrays, the system ensures both security and usability. Additional features like interest calculation, highest balance display, and account summaries enhance its practicality. This project effectively demonstrates core programming concepts—arrays, loops, conditionals, and string handling—through a realistic banking simulation.

2. Technical Terms and Concepts :

- **Arrays for Data Management :** Multiple **parallel arrays** are used to store customer details such as names, account numbers, passwords, balances, and

transaction history. This approach efficiently handles multiple records within memory and simplifies account tracking.

- **String Handling and Validation :** String functions like **`strcmp()`** and **`strcpy()`** are used for password comparison, account verification, and data copying—ensuring secure and accurate string manipulation throughout the program.

- **Conditional and Looping Structures :** The program heavily relies on **`if-else`, `for`**, and **`while` loops** for decision-making, input validation, and repeated menu navigation, enabling dynamic and interactive program control.

- **Menu-Driven Program Using Switch Case :** The system uses **nested `switch`** statements to organize operations into modules such as Admin Panel, ATM Services, Help Desk, and Additional Features—making the interface structured and user-friendly.

- **Transaction Recording with 2D Arrays :** A **two-dimensional array** efficiently maintains up to ten recent transactions per account using formatted strings and counters, providing a mini statement feature similar to real-world banking systems.

- **Arithmetic and Logical Operations :** The system performs **arithmetic operations** for deposits, withdrawals, transfers, and interest calculations, along with logical checks for balance limits, account existence, and secure transactions.

- **Structured and Modular Design :** The code follows a logical structure with distinct functional sections, improving readability, maintainability, and demonstrating the concept of modular programming even within a single **`main()`** function.

- **User Interaction through Console I/O :** Using **`printf()`** and **`scanf()`**, the program implements clear, interactive input–output communication, allowing smooth user experience while performing banking tasks in a console environment.

3. System Design and Implementation :

3.1 System Modules

- **3.1.1 Admin Panel:**
  - Add, delete, and manage customer accounts.
  - Reset account passwords.
  - View all accounts and summaries.
- **3.1.2 ATM Services:**
  - Deposit and withdraw money.
  - Check balance.
  - Transfer funds to other accounts.
  - View mini statements (last 10 transactions).
- **3.1.3 Help Desk:**
  - Guide users on account operations.
  - Provide step-by-step instructions for banking services.
- **3.1.4 Additional Features:**
  - Interest calculation for all accounts.
  - Display account with highest balance.
  - List accounts below a specific balance threshold.
  - Total account summary.

3.2 Data Management

- **Arrays for storage:**
  - `char names[100][50]` → Stores customer names.
  - `int accNo[100]` → Stores account numbers.
  - `char passwords[100][20]` → Stores account passwords.
  - `float balance[100]` → Stores account balances.
  - `char transactions[100][10][100]` → Stores last 10 transactions per account.
- **Counters:**

  - `int transCount[100]` → Tracks the number of transactions per account.
- **Total Accounts:**
  - **`int totalAccounts`** → Keeps count of total accounts in the system.

3.3 Security and Validation

- Password-protected admin and user logins.
- Checks for sufficient balance before withdrawal or transfer.
- Ensures valid account numbers for all operations.

3.4 Implementation Flow

- **Menu-driven interface** using **`switch-case`.**
- **Loops** for repeated menu navigation and transaction handling.
- **Conditional statements** for decision-making (login verification, balance checks, etc.).
- **String functions** (`strcmp`, `strcpy`) for password and name handling.
- **Transaction logs** recorded using **`sprintf()and 2D Array`.**

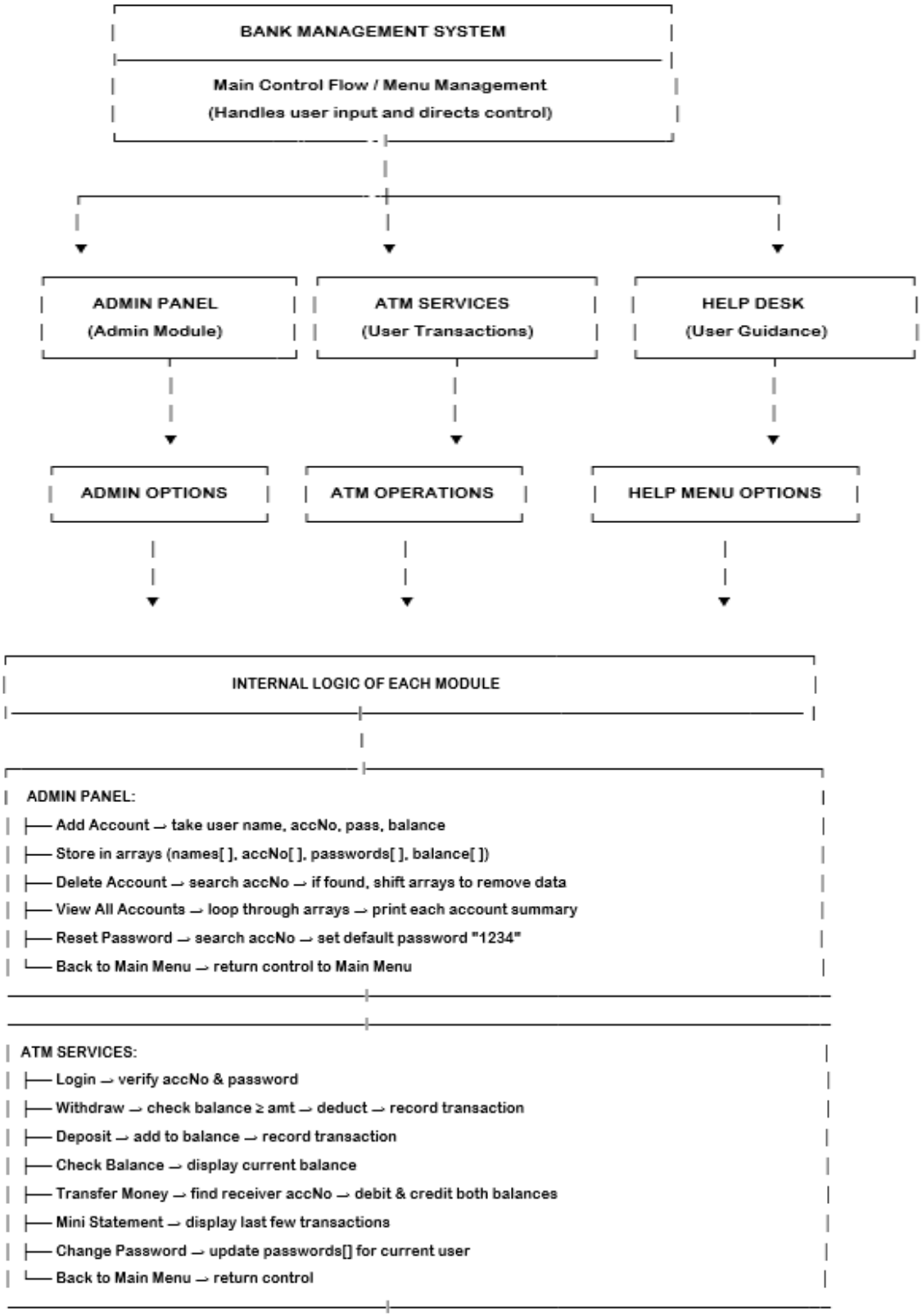
3.5 Modular Functions

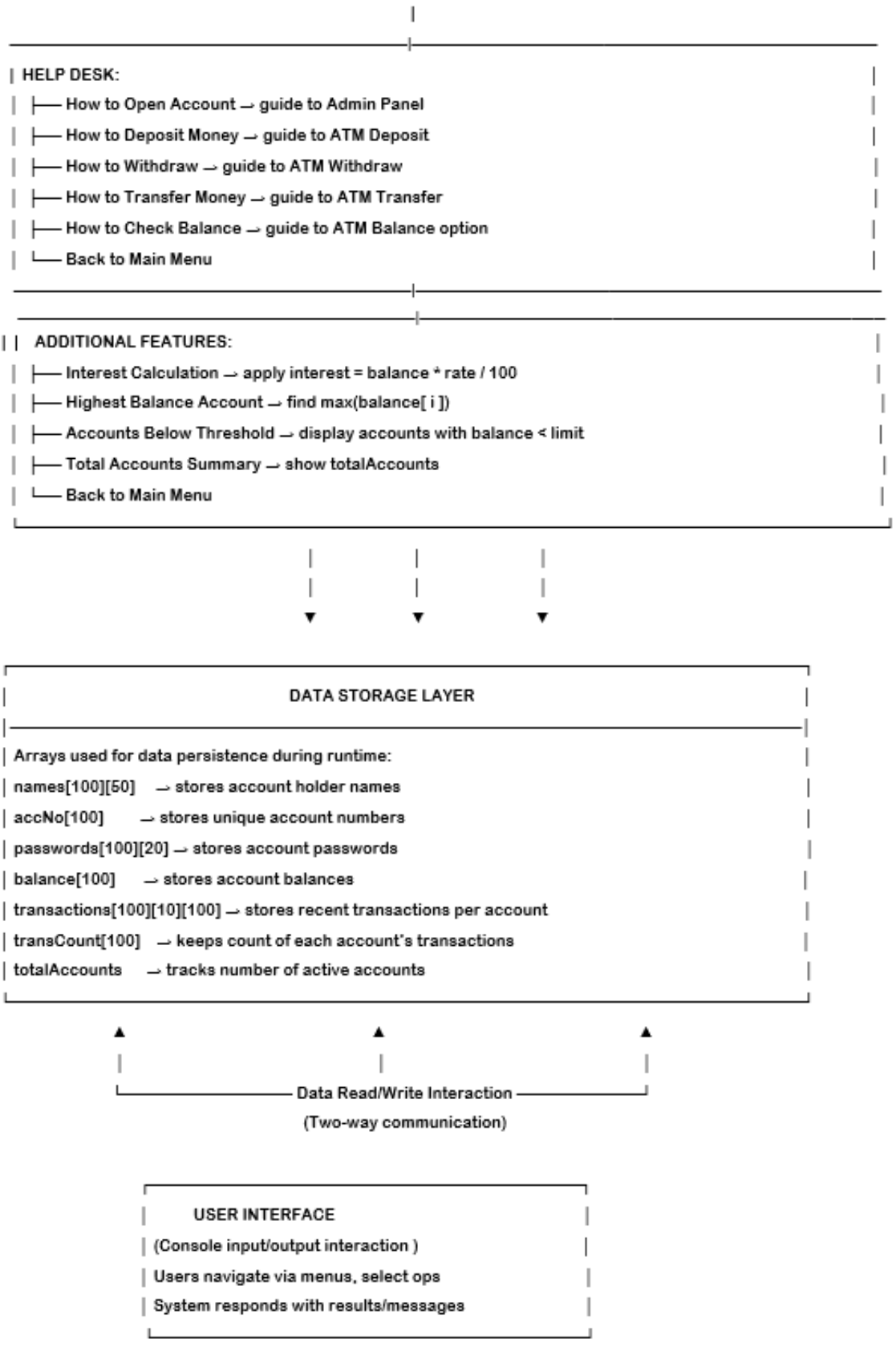
Although implemented in `main()`, the program logically follows modular concepts:

- **`add_account()`** → Adds new account with initial balance.
- **`delete_account()`** → Removes account and shifts remaining data.
- **`reset_password()`** → Resets a user's password.
- **`atm_withdraw()`** → Withdraws funds with balance check.
- **`atm_deposit()`** → Deposits funds into account.
- **`atm_transfer()`** → Transfers money between accounts with validation.

- **`view_mini_statement()`** → Prints recent transactions for a user.
- **`calculate_interest()`** → Adds interest to all account balances.
- **`view_highest_balance()`** → Displays account with maximum balance.
- **`view_accounts_below_threshold()`** → Lists accounts below a set balance.

4 . Architechture Diagram:




HELP DESK:
How to Open Account → guide to Admin Panel
How to Deposit Money → guide to ATM Deposit
How to Withdraw → guide to ATM Withdraw
How to Transfer Money → guide to ATM Transfer
How to Check Balance → guide to ATM Balance option
Back to Main Menu
ADDITIONAL FEATURES:
Interest Calculation → apply interest = balance * rate / 100
Highest Balance Account → find max(balance[i])
Accounts Below Threshold → display accounts with balance < limit
Total Accounts Summary → show totalAccounts
Back to Main Menu
DATA STORAGE LAYER
Arrays used for data persistence during runtime:
names[100][50] → stores account holder names
accNo[100] → stores unique account numbers
passwords[100][20] → stores account passwords
balance[100] → stores account balances
transactions[100][10][100] → stores recent transactions per account
transCount[100] → keeps count of each account's transactions
totalAccounts → tracks number of active accounts
Data Read/Write Interaction
(Two-way communication)
USER INTERFACE
(Console input/output interaction)
Users navigate via menus, select ops
System responds with results/messages

4. Code :

```
#include <stdio.h>
#include <string.h>
int main() {
   // ------------------------- DATA STORAGE -------------------------
   // Arrays to store account information
   char names[100][50];         // Names of up to 100 users
   int accNo[100];              // Account numbers
   char passwords[100][20];     // Account passwords
   float balance[100];          // Account balances
   char transactions[100][10][100]; // Stores last 10 transactions per account
   int transCount[100] = {0};   // Keeps count of transactions per account

   int totalAccounts = 0;       // Total number of accounts in the system
   char adminPass[20] = "admin123"; // Hardcoded admin password
   int choice, i, j;            // General loop and menu variables
   char tempPass[20];           // Temporary variable to take admin password
input
   int accIndex = -1;           // Used to store index of logged-in account

   // ------------------------- MAIN LOOP -------------------------
   while (1) {
      // Main menu
      printf("\n==============================\n");
      printf("     BANK MANAGEMENT SYSTEM\n");
      printf("==============================\n");
      printf("1. Admin Login\n");
      printf("2. ATM Services\n");
      printf("3. Help Desk\n");
      printf("4. Additional Features\n");
      printf("5. Exit\n");
      printf("Enter choice: ");
      scanf("%d", &choice);
```

```
        switch (choice) {

            // ------------------------- ADMIN LOGIN -------------------------
            case 1: {
                printf("\nEnter Admin Password: ");
                scanf("%s", tempPass);

                // Verify admin password
                if (strcmp(tempPass, adminPass) == 0) {
                    int adminChoice;
                    while (1) {
                        // Admin panel options
                        printf("\n--- ADMIN PANEL ---\n");
                        printf("1. Add New Account\n");
                        printf("2. Delete Account\n");
                        printf("3. View All Accounts\n");
                        printf("4. Reset Account Password\n");
                        printf("5. Back to Main Menu\n");
                        printf("Enter choice: ");
                        scanf("%d", &adminChoice);

                        switch (adminChoice) {
                            case 1:  // Add new account
                                                    printf("\nEnter Name: ");
                                scanf("%s", names[totalAccounts]);
                                fflush(stdin);
                                                printf("Enter Account
Number: ");
                                scanf("%d", &accNo[totalAccounts]);
                                printf("Set Password: ");
                                scanf("%s", passwords[totalAccounts]);
                                printf("Enter Initial Balance: ");
                                scanf("%f", &balance[totalAccounts]);
                                printf("Account Created Successfully!\n");
                                totalAccounts++;
```

```
        break;

    case 2:  // Delete existing account
        printf("\nEnter Account Number to Delete: ");
        int delAcc;
        scanf("%d", &delAcc);
        int found = 0;

        // Search account to delete
        for (i = 0; i < totalAccounts; i++) {
            if (accNo[i] == delAcc) {
                // Shift elements to remove account
                for (j = i; j < totalAccounts - 1; j++) {
                    strcpy(names[j], names[j + 1]);
                    accNo[j] = accNo[j + 1];
                    strcpy(passwords[j], passwords[j + 1]);
                    balance[j] = balance[j + 1];
                }
                totalAccounts--;
                printf("Account Deleted!\n");
                found = 1;
                break;
            }
        }
        if (!found) printf("Account Not Found!\n");
        break;

    case 3:  // View list of all accounts
        printf("\n--- ALL ACCOUNTS ---\n");
        for (i = 0; i < totalAccounts; i++) {
            printf("Name: %s | AccNo: %d | Balance: %.2f\n",
                   names[i], accNo[i], balance[i]);
        }
        break;
```

```
                    case 4:  // Reset password to new password entered by
admin
                      printf("Enter Account Number to Reset Password: ");
                      int resAcc;
                      scanf("%d", &resAcc);
                      found = 0;

                     // Search and reset
                     for (i = 0; i < totalAccounts; i++) {
                      if (accNo[i] == resAcc) {
                      char newPass[20];
                     printf("Enter new password for account %d: ", resAcc);
                    scanf("%s", newPass);        // Take new password input
                   strcpy(passwords[i], newPass);  // Set the new password
                       printf("Password reset successfully!\n");
                    found = 1;
                   break;
                     }
                  }
                     if (!found) printf("Account Not Found!\n");
                    break;

                    case 5:
                       // Exit admin panel
                       goto endAdmin;
                 }
              }
            endAdmin:
              ;
            } else {
              printf("Incorrect Admin Password!\n");
            }
            break;
        }
```

```
// ------------------------- ATM SERVICES -------------------------
case 2: {
   int userAcc;
   char userPass[20];

   // Login section
   printf("\nEnter Account Number: ");
   scanf("%d", &userAcc);
   printf("Enter Password: ");
   scanf("%s", userPass);

   int loggedIn = 0;

   // Validate user credentials
   for (i = 0; i < totalAccounts; i++) {
      if (accNo[i] == userAcc && strcmp(passwords[i], userPass) ==
0) {
         accIndex = i;
         loggedIn = 1;
         break;
      }
   }

   // If login is successful
   if (loggedIn) {
      int atmChoice;
      while (1) {
         printf("\n--- ATM SERVICES ---\n");
         printf("1. Withdraw Money\n");
         printf("2. Deposit Money\n");
         printf("3. Check Balance\n");
         printf("4. Transfer Money\n");
         printf("5. Mini Statement\n");
         printf("6. Back to Main Menu\n");
```

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

                // Withdraw money
                if (atmChoice == 1) {
                    float amt;
                    printf("Enter amount to withdraw: ");
                    scanf("%f", &amt);
                    if (balance[accIndex] >= amt) {
                        balance[accIndex] -= amt;
                        printf("Withdraw successful! New balance: %.2f\n",
balance[accIndex]);
                        sprintf(transactions[accIndex][transCount[accIndex]++],
                                "Withdraw: -%.2f", amt);
                    } else {
                        printf("Insufficient Balance!\n");
                    }
                }

                // Deposit money
                else if (atmChoice == 2) {
                    float amt;
                    printf("Enter amount to deposit: ");
                    scanf("%f", &amt);
                    balance[accIndex] += amt;
                    printf("Deposit successful! New balance: %.2f\n",
balance[accIndex]);
                    sprintf(transactions[accIndex][transCount[accIndex]++],
                            "Deposit: +%.2f", amt);
                }

                // Check balance
                else if (atmChoice == 3) {
                    printf("Current Balance: %.2f\n", balance[accIndex]);
                }
```

```
            // Transfer money to another account
            else if (atmChoice == 4) {
                int recvAcc;
                float amt;
                printf("Enter receiver account number: ");
                scanf("%d", &recvAcc);
                printf("Enter amount to transfer: ");
                scanf("%f", &amt);

                int recvIndex = -1;
                for (i = 0; i < totalAccounts; i++) {
                    if (accNo[i] == recvAcc) {
                        recvIndex = i;
                        break;
                    }
                }

                if (recvIndex != -1 && balance[accIndex] >= amt) {
                    balance[accIndex] -= amt;
                    balance[recvIndex] += amt;
                    printf("Transfer successful!\n");
                    sprintf(transactions[accIndex][transCount[accIndex]++],
                            "Transfer Out: -%.2f", amt);

sprintf(transactions[recvIndex][transCount[recvIndex]++],
                            "Transfer In: +%.2f", amt);
                } else {
                    printf("Transfer Failed!\n");
                }
            }

            // View mini statement (recent transactions)
            else if (atmChoice == 5) {
                printf("\n--- MINI STATEMENT ---\n");
```

```
                for (j = 0; j < transCount[accIndex]; j++) {
                    printf("%s\n", transactions[accIndex][j]);
                }
            }

            // Exit ATM menu
            else if (atmChoice == 6)
                goto endATM;
        }
    endATM:
        ;
    } else {
        printf("Invalid Login!\n");
    }
    break;
}

// ------------------------- HELP DESK -------------------------
case 3: {
    int helpChoice;
    while (1) {
        // Help menu options
        printf("\n--- HELP DESK ---\n");
        printf("1. How to open account?\n");
        printf("2. How to deposit money?\n");
        printf("3. How to withdraw money?\n");
        printf("4. How to transfer money?\n");
        printf("5. How to check balance?\n");
        printf("6. Back to Main Menu\n");
        printf("Enter choice: ");
        scanf("%d", &helpChoice);

        // Simple informational responses
        if (helpChoice == 1)
```

```
                printf("Visit bank or admin section to create a new
account.\n");
            else if (helpChoice == 2)
                printf("Login to ATM Services and select Deposit.\n");
            else if (helpChoice == 3)
                printf("Login to ATM Services and select Withdraw.\n");
            else if (helpChoice == 4)
                printf("Login to ATM Services and select Transfer.\n");
            else if (helpChoice == 5)
                printf("Login to ATM Services and select Check
Balance.\n");
            else if (helpChoice == 6)
                break;
            }
            break;
        }

        // ------------------------- ADDITIONAL FEATURES ----------------
----------
        case 4: {
            int addChoice;
            while (1) {
                printf("\n--- ADDITIONAL FEATURES ---\n");
                printf("1. Interest Calculation\n");
                printf("2. View Highest Balance Account\n");
                printf("3. View Accounts Below Balance Threshold\n");
                printf("4. Total Accounts Summary\n");
                printf("5. Back to Main Menu\n");
                printf("Enter choice: ");
                scanf("%d", &addChoice);

                // Add interest to all accounts
                if (addChoice == 1) {
                    float rate;
                    printf("Enter interest rate (%%): ");
```

```
            scanf("%f", &rate);
            for (i = 0; i < totalAccounts; i++) {
                float interest = balance[i] * rate / 100.0;
                balance[i] += interest;
            }
            printf("Interest added successfully to all accounts!\n");
        }

        // Display account with highest balance
        else if (addChoice == 2) {
            if(totalAccounts == 0){
                printf("No accounts yet!\n");
                continue;
            }
            int maxIndex = 0;
            for(i = 1; i < totalAccounts; i++)
                if(balance[i] > balance[maxIndex])
                    maxIndex = i;
            printf("Highest Balance Account: %s | AccNo: %d | Balance:
%.2f\n",
                    names[maxIndex], accNo[maxIndex],
balance[maxIndex]);
        }

        // Display accounts with balance below a limit
        else if (addChoice == 3) {
            float limit;
            printf("Enter balance threshold: ");
            scanf("%f", &limit);
            printf("Accounts below %.2f:\n", limit);
            for(i = 0; i < totalAccounts; i++)
                if(balance[i] < limit)
                    printf("Name: %s | AccNo: %d | Balance: %.2f\n",
                        names[i], accNo[i], balance[i]);
        }
```

```
                // Display total number of accounts
                else if (addChoice == 4) {
                    printf("Total Accounts: %d\n", totalAccounts);
                }

                // Exit additional features
                else if (addChoice == 5)
                    break;
            }
            break;
        }

        // ------------------------- EXIT -------------------------
        case 5: {
            char ch;
            printf("Are you sure you want to exit? (y/n): ");
            scanf(" %c", &ch);
            if (ch == 'y' || ch == 'Y') {
                printf("Thank you for using our Bank!\n");
                return 0;
            }
            break;
        }

        // Invalid menu input
        default:
            printf("Invalid choice! Try again.\n");
    }
  }

  return 0;
}
```

5. Results

All core functionalities of the system were tested, including admin operations, ATM services, and additional features. Every module performed as expected: accounts were created, deleted, and updated correctly; deposits, withdrawals, and transfers were processed accurately; transaction histories and balance checks worked flawlessly. Edge cases like incorrect passwords, insufficient funds, and non-existent accounts were handled properly.

6. Function Reliability

The system is **highly reliable** for small-scale usage (up to 100 accounts). Functions execute consistently without crashing, data is stored and retrieved correctly, and error handling ensures stability during invalid inputs or operations.

7. User Experience

The program offers a **simple and intuitive interface**. Menu-driven navigation allows users and admins to access functions easily. Clear prompts and feedback messages enhance usability, while mini statements and additional features like interest calculation and balance thresholds improve the overall banking experience.

8. Assumptions

- The system assumes a **maximum of 100 accounts** for testing purposes.
- Account numbers are **unique**, and users provide valid numeric inputs.
- Transactions are **limited to 10 recent entries per account** for the mini statement.
- Users are expected to **remember passwords**; no password recovery exists outside admin reset.
-

9. Future Work

- **Database Integration:** Move from arrays to a proper database for unlimited accounts and persistent storage.
- **Enhanced Security:** Implement hashed passwords and multi-factor authentication.
- **Extended Transactions:** Keep a full transaction history with timestamps and detailed logs.
- **Graphical User Interface (GUI):** Replace console-based menus with an intuitive GUI.
- **Additional Features:** Include loan management, interest calculators, account statements in PDF, and notifications.

10. Conclusion

The **Bank Management System** successfully demonstrates a fully functional mini-banking solution with **secure admin controls, ATM services, and helpful additional features**. It is **robust, reliable, and user-friendly**, providing a realistic simulation of banking operations. With future enhancements like GUI and database support, this system could evolve into a **comprehensive digital banking platform**.