**DLITHE PROJECT REPORT**

**PROJECT ID: CP014**

**PROJECT TITLE:** BANK MANAGEMENT SYSTEM

**TEAM MEMBERS:** AVANI K D (4MT21IS009)

DEVI JANANI (4MT21IS015)

SHREYA VINAYAK NAIK (4MT21IS043)

SPANDANA CHATRA (4MT21IS049)

U R NAGAVENI (4MT21IS057)

**REPORT**

1. **ABSTRACT:**

The Bank Management System is a web-based application designed to generate and maintain daily payment transactions with customer accounts. The system provides features such as CRUD operations, cash withdrawal and deposit, and checking customer details. It aims to streamline banking operations and improve customer service.

1. **INTRODUCTION:**
   1. **BACKGROUND:**

The development of the Bank Management System is driven by the need for a robust and efficient system to manage daily banking operations. Traditional manual processes are prone to errors and inefficiencies, and this system aims to address these issues.

* 1. **OBJECTIVES:**

The objectives of the Bank Management System are:

* To provide a secure login system for users
* To allow CRUD operations for managing customer accounts
* To enable cash withdrawal and deposit functionality
* To provide options for checking customer account details
* To maintain transaction records and update account balances

1. **TECHNOLOGIES USED:**

The Library Management System is implemented using the following technologies:

• C programming language

• File handling for data storage

1. **SYSTEM ARCHITECTURE:**
   1. Front-End :

The front-end of the system is implemented using the console or command-line interface, where users interact with the system by entering commands and providing input.

4.2 Back-End:

The back-end of the system consists of C programming logic that performs various operations such as adding, displaying, modifying, and deleting records. It also handles file handling to store and retrieve data.

4.3 Database :

The system uses files to act as databases for storing information about Admin and Users. Each entity has a separate file for data storage. It handles CRUD operations, transaction processing, and account balance updates.

1. **PROJECT MODULES:**

The project consists of the following modules:

* MODULE 1: ADMIN
* Create Account
* Update Account
* Find Account
* MODULE 2: USER
* Update Account
* Find Account
* Deposit
* Withdraw
* Check Balance

1. **Design and Implementation :**

6.1 Front-End Design :

• The front-end design is based on a command-line interface, where users can select

options by entering numbers corresponding to their desired actions.

6.2 Back-End Design :

• The back-end logic is implemented in the C programming language.

• Each module has its functions for adding, displaying, modifying, and deleting records.

• File handling functions are used to perform operations on data files.

6.3 Database Design :

• Data for user is stored in a text file.

• Each file contains records in a structured format.

## 7. Features and Functionality:

* User login system with secure authentication
* CRUD operations for managing customer accounts
* Cash withdrawal and deposit functionality
* Options for checking customer account details
* Transaction record maintenance and balance updates

**8.Testing**

The Bank Management System undergoes rigorous testing to ensure its functionality, security, and usability. It includes unit testing, integration testing, and user acceptance testing to identify and resolve any issues or bugs.

**9.Challenges Faced**

During the development of the Bank Management System, some challenges were faced, including:

* Data validation and handling user inputs
* Maintaining data integrity and security
* Ensuring smooth integration between front-end and back-end components
* Optimizing system performance and scalability

**10.Future Enhancements**

In the future, the Bank Management System can be enhanced with additional features such as:

* Account statement generation
* Transaction history tracking
* Notifications and alerts for account activities
* Integration with external payment gateways

**11.Conclusion**

The Bank Management System is a comprehensive solution for managing customer accounts and processing transactions. It improves efficiency, accuracy, and security in banking operations. With its user-friendly interface and robust functionality, it enhances the overall banking experience for both customers and bank staff.

**12. Appendices:**

A. System Overview

The Bank Management System is designed to manage daily payment transactions with customer accounts. This section provides an overview of the system's functionality and features.

B. Login System

Before accessing the main system, users are required to pass through a login system to ensure security and authentication.

C. Customer Record Information

1. Customer Data Entry

Customers are required to provide the following information when creating an account:

IFSC number

Account number

Name

Address

Phone number

Initial deposit amount

Account type (e.g., Saving, Current, Fixed)

2. Updating Customer Information

When updating existing customer information, users can modify the following details:

Name

Address

Phone number

3. Account Search

Users can search for customer accounts in two ways:

By Account number

By Account name

4. Customer Record Listing

The system provides a listing of customer records displaying:

Customer's name

Account number

IFSC number

Address

Contact details

D. Transaction Management

Transactions within the bank involve two main operations:

Depositing amounts

Withdrawing amounts

1. Deposit and Withdrawal

To perform deposit or withdrawal transactions, users must provide the transaction amount.

The system automatically maintains transaction records, including total bank balance for each customer.

E. System Features

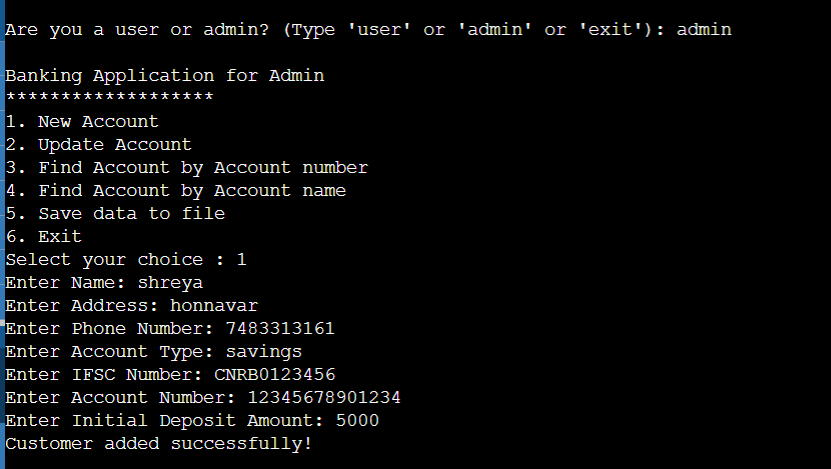
Key features of the Bank Management System include:

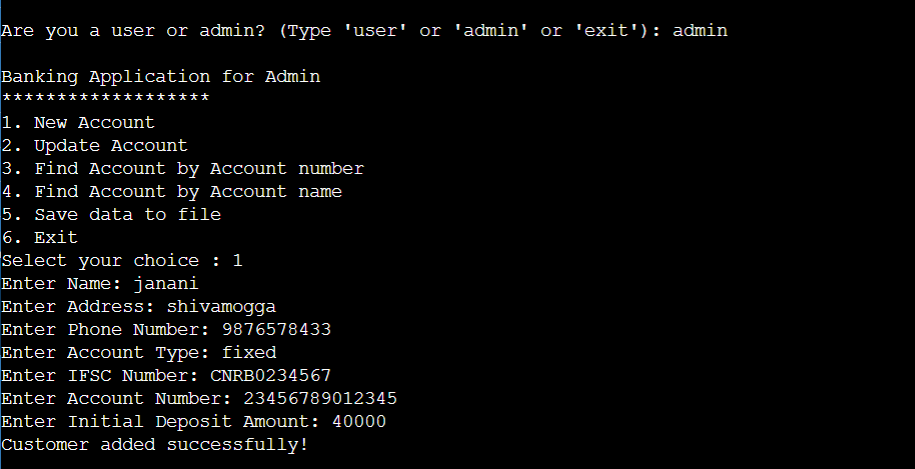
CRUD (Create, Read, Update, Delete) operations for customer records

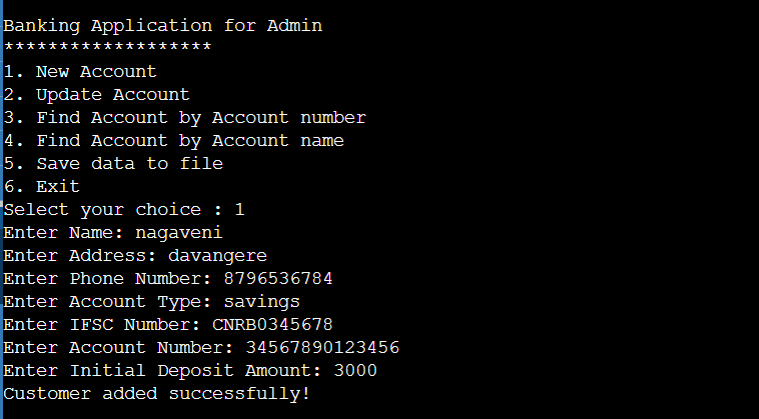
Transaction recording (Debit/Credit)

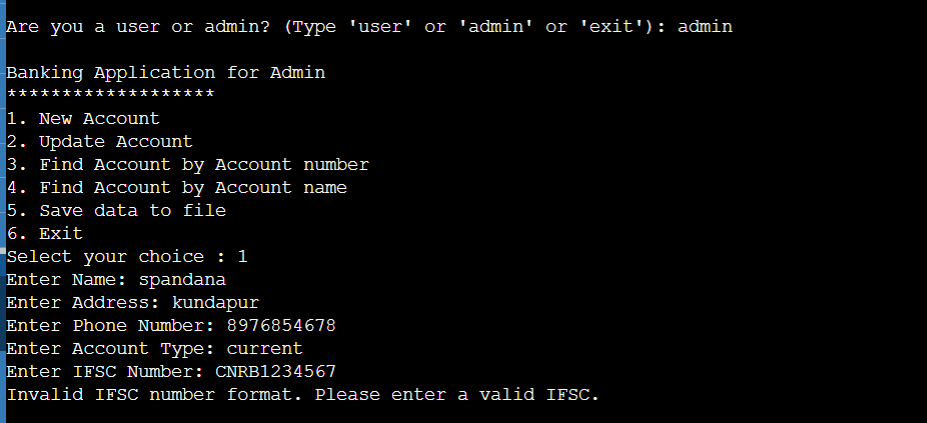
Balance checking for customer accounts

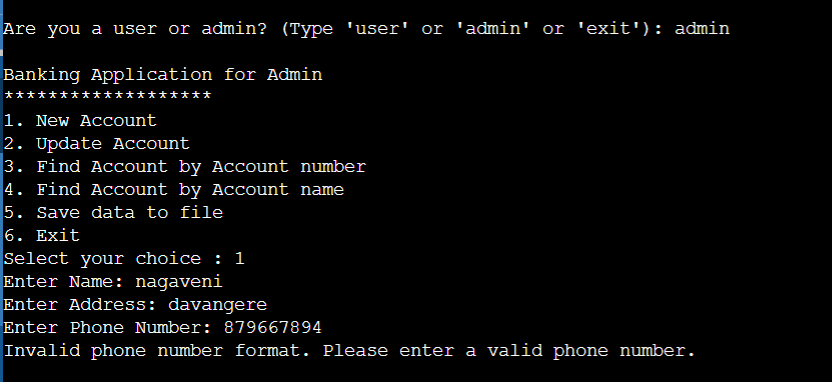
**13.1 Screenshots:**

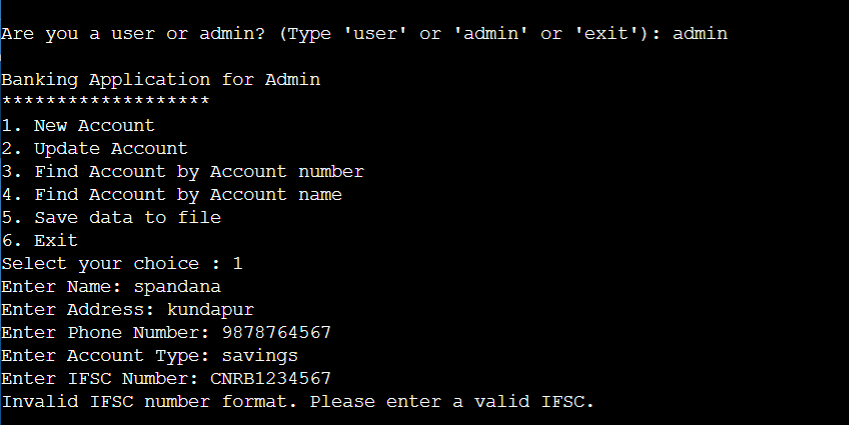
****

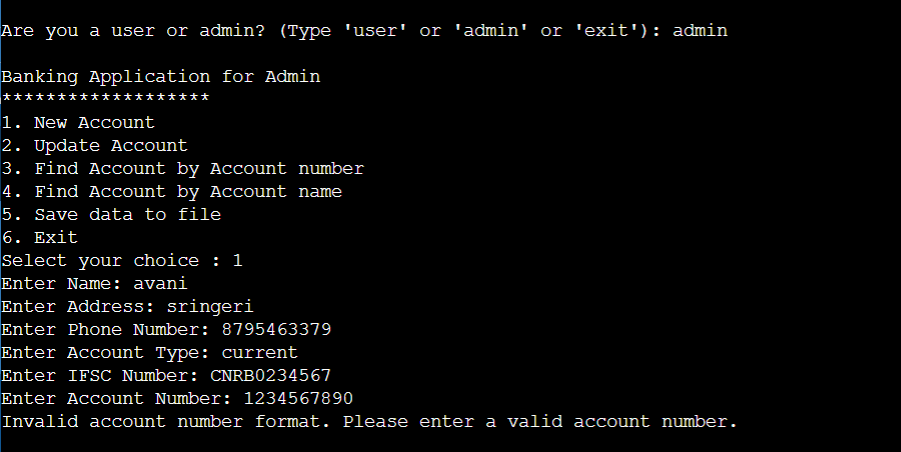
****

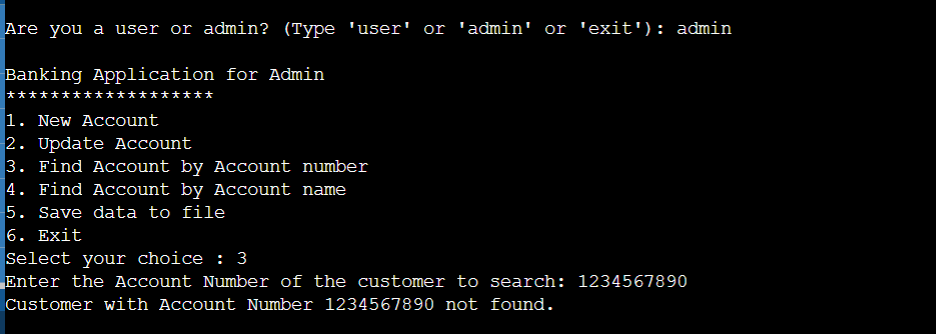
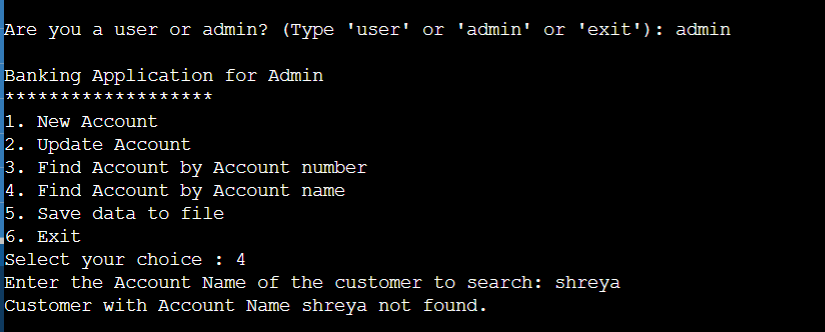
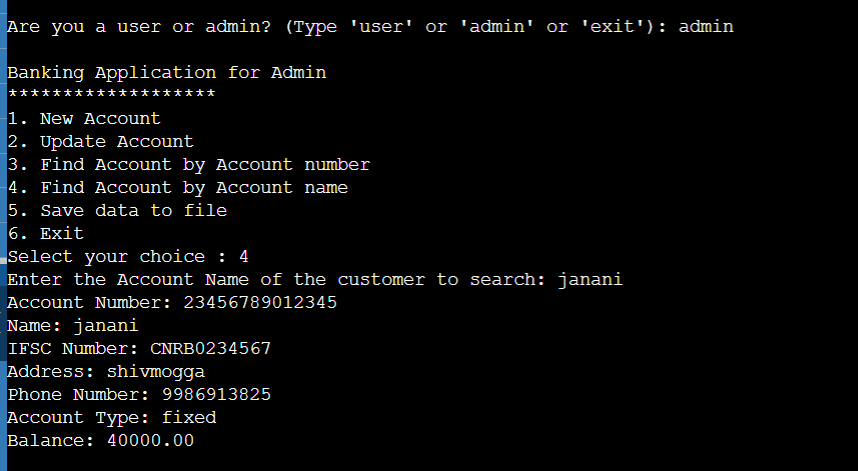
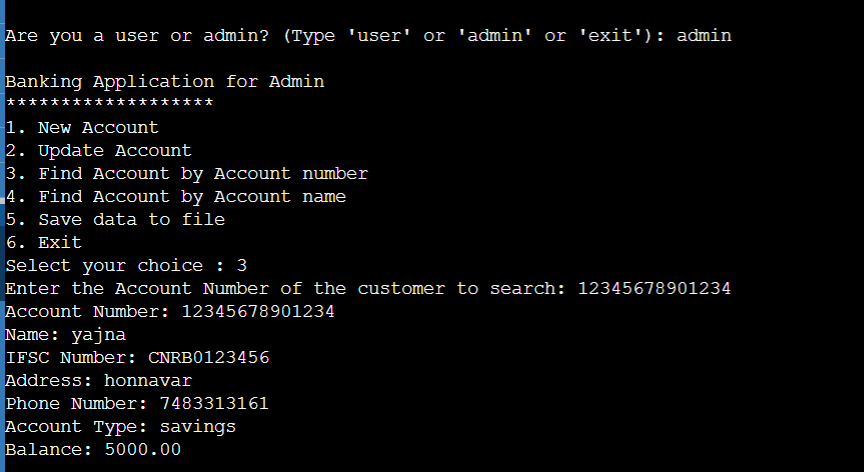
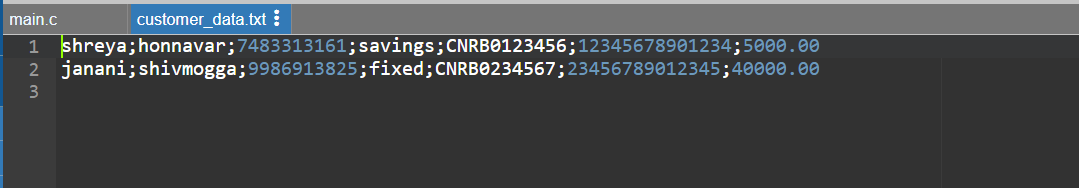
****

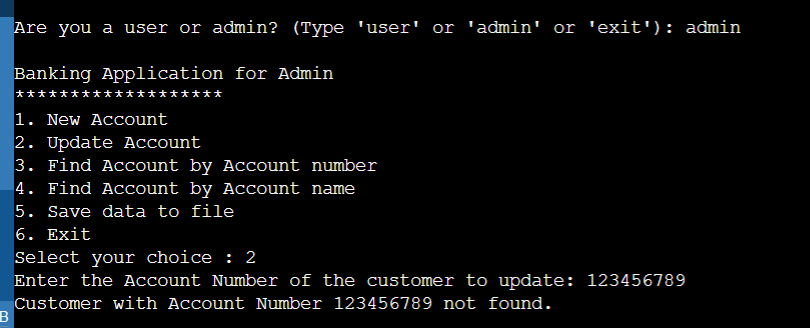
****

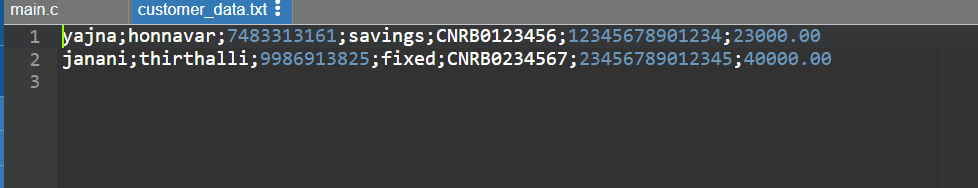
****

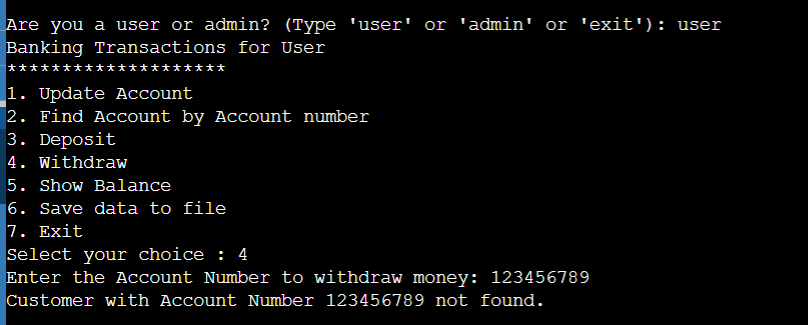
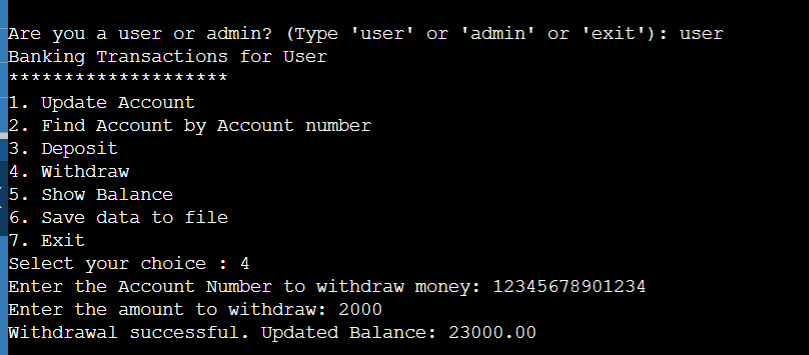
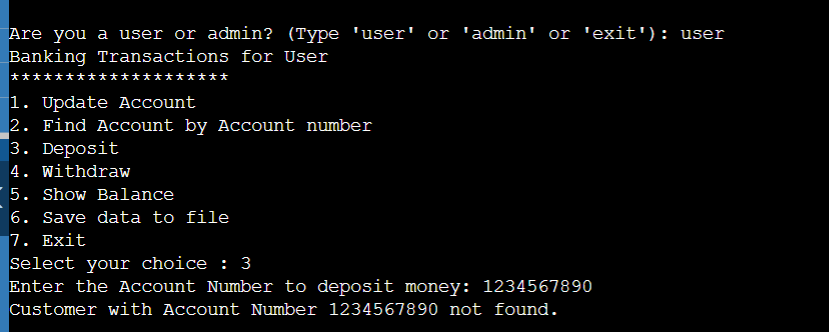
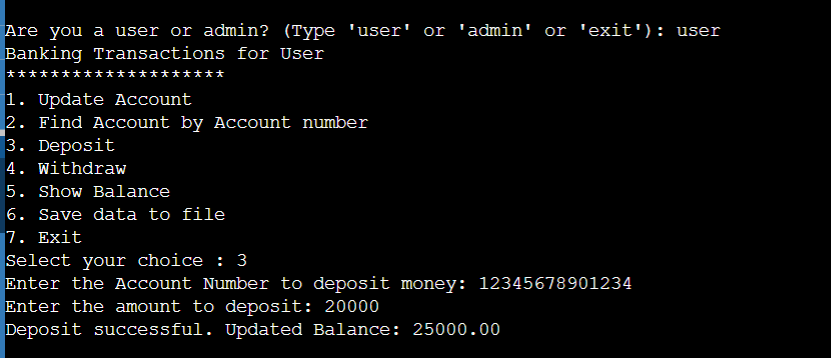
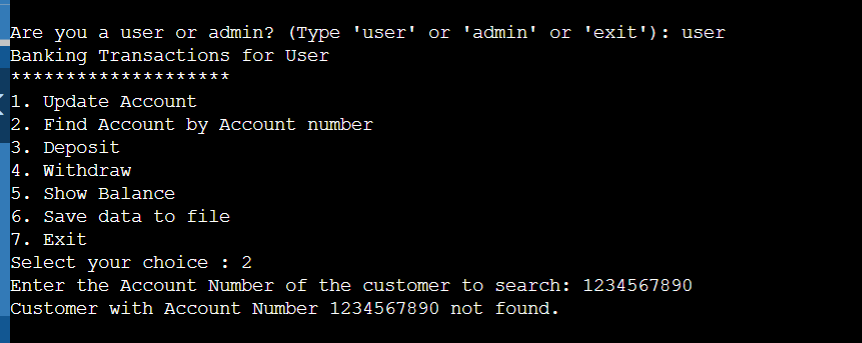
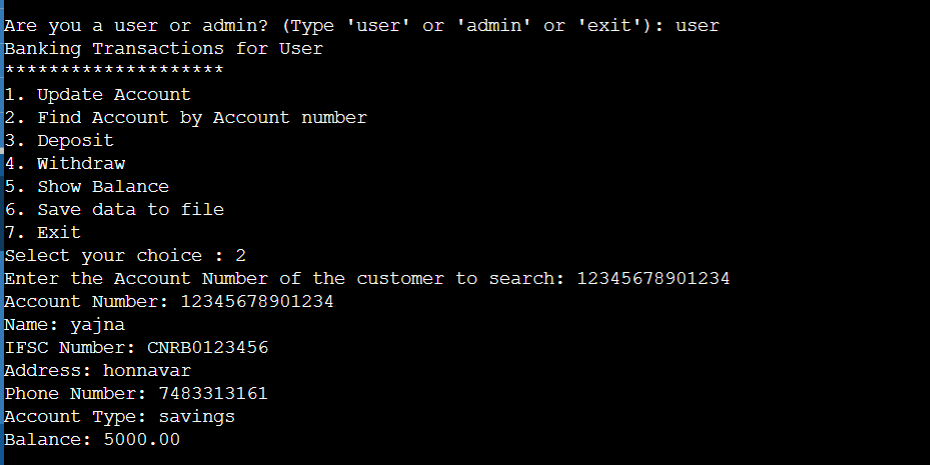
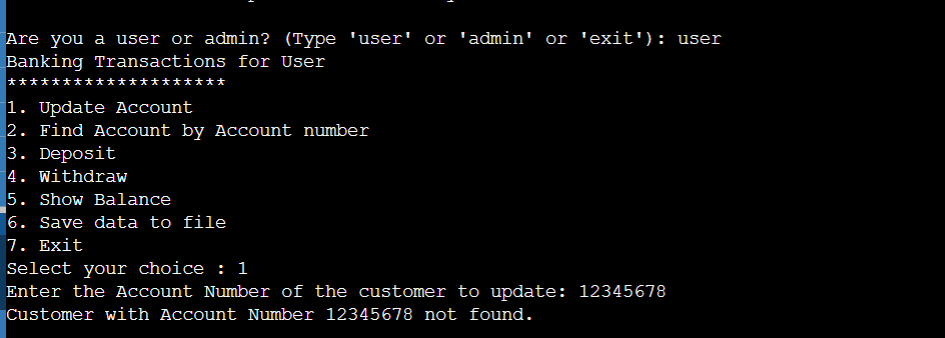
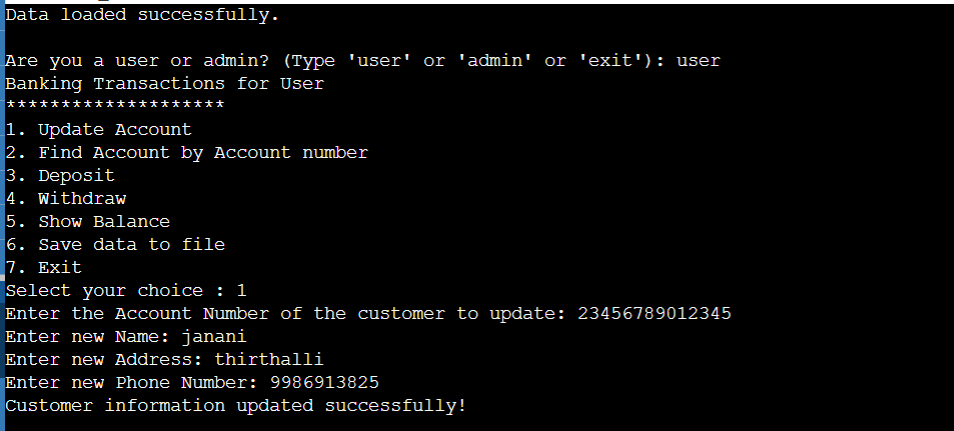
****

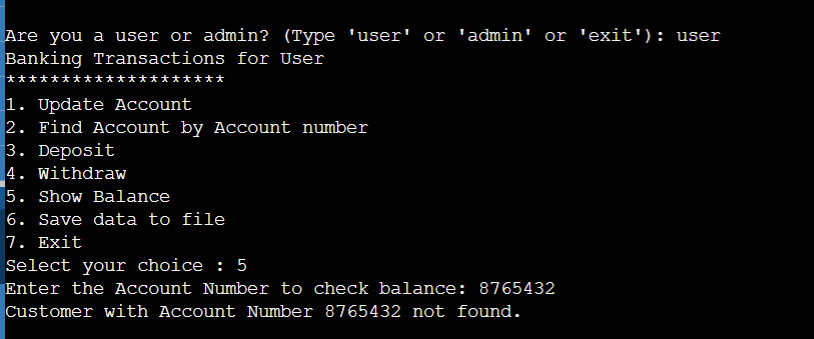
****

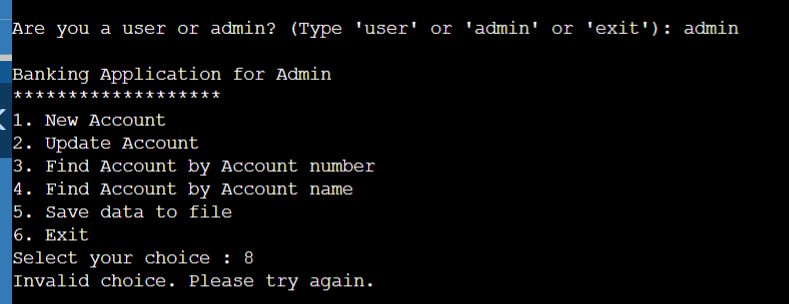
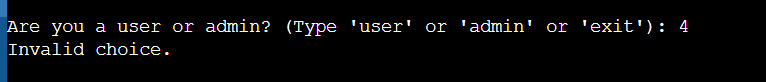
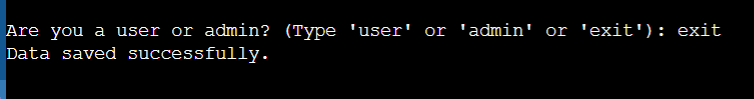
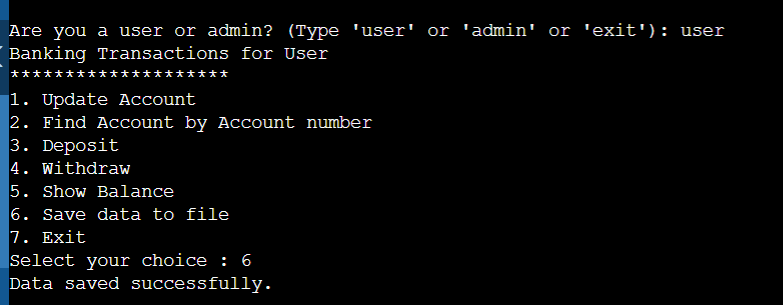
****

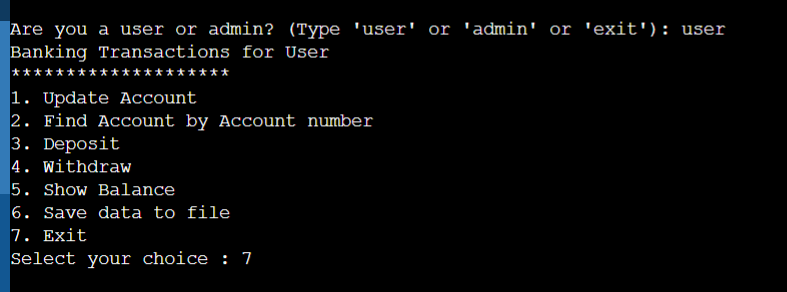
****

****

****

****

****

****

**13.2 Code Snippets:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_CUSTOMERS 100

struct Customer {

char name[50];

char address[100];

char phone[15];

char accountType[20];

char ifsc[12]; // Changed the size of IFSC to 12 to include null terminator

char accountNumber[21]; // Changed the size of Account Number to 21 to include null terminator

float balance;

char dob[11]; // Date of birth (format: dd/mm/yyyy)

int age;

};

struct Customer \*customers[MAX\_CUSTOMERS];

int numCustomers = 0;

// Function prototypes

void adminMenu();

void userMenu();

void addCustomer();

void updateCustomer();

struct Customer \*findCustomerByAccountNumber(const char \*accountNumber);

void searchByAccountNumber();

void searchByAccountName();

void deposit();

void withdraw();

void checkBalance();

void saveDataToFile();

void loadDataFromFile();

int isValidIFSC(const char \*ifsc);

int isValidPhoneNumber(const char \*phoneNumber);

int isValidAccountNumber(const char \*accountNumber);

int main() {

loadDataFromFile();

char person[10];

while (1) {

printf("\nAre you a user or admin? (Type 'user' or 'admin' or 'exit'): ");

scanf("%s", person);

if (strcmp(person, "admin") == 0) {

adminMenu();

} else if (strcmp(person, "user") == 0) {

userMenu();

} else if (strcmp(person, "exit") == 0) {

saveDataToFile(); // Save data before exiting

exit(0);

} else {

printf("Invalid choice.\n");

}

}

return 0;

}

void adminMenu() {

int choice1;

printf("\nBanking Application for Admin\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("1. New Account\n");

printf("2. Update Account\n");

printf("3. Find Account by Account number\n");

printf("4. Find Account by Account name\n");

printf("5. Save data to file\n");

printf("6. Exit\n");

printf("Select your choice : ");

scanf("%d", &choice1);

switch (choice1) {

case 1:

addCustomer();

break;

case 2:

updateCustomer();

break;

case 3:

searchByAccountNumber();

break;

case 4:

searchByAccountName();

break;

case 5:

saveDataToFile();

break;

case 6:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

void userMenu() {

int choice2;

printf("Banking Transactions for User\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("1. Update Account\n");

printf("2. Find Account by Account number\n");

printf("3. Deposit\n");

printf("4. Withdraw\n");

printf("5. Show Balance\n");

printf("6. Save data to file\n");

printf("7. Exit\n");

printf("Select your choice : ");

scanf("%d", &choice2);

switch (choice2) {

case 1:

updateCustomer();

break;

case 2:

searchByAccountNumber();

break;

case 3:

deposit();

break;

case 4:

withdraw();

break;

case 5:

checkBalance();

break;

case 6:

saveDataToFile();

break;

case 7:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

void addCustomer() {

if (numCustomers >= MAX\_CUSTOMERS) {

printf("Cannot add more customers. The system is full.\n");

return;

}

struct Customer \*newCustomer = (struct Customer \*)malloc(sizeof(struct Customer));

printf("Enter Name: ");

scanf(" %[^\n]", newCustomer->name);

printf("Enter Address: ");

scanf(" %[^\n]", newCustomer->address);

printf("Enter Phone Number: ");

scanf(" %s", newCustomer->phone);

if (!isValidPhoneNumber(newCustomer->phone)) {

printf("Invalid phone number format. Please enter a valid phone number.\n");

free(newCustomer);

return;

}

printf("Enter Account Type: ");

scanf(" %[^\n]", newCustomer->accountType);

printf("Enter IFSC Number: ");

scanf(" %s", newCustomer->ifsc);

// Validate IFSC number

if (!isValidIFSC(newCustomer->ifsc)) {

printf("Invalid IFSC number format. Please enter a valid IFSC.\n");

free(newCustomer); // Free the allocated memory

return;

}

printf("Enter Account Number: ");

scanf(" %s", newCustomer->accountNumber);

// Validate account number

if (!isValidAccountNumber(newCustomer->accountNumber)) {

printf("Invalid account number format. Please enter a valid account number.\n");

free(newCustomer); // Free the allocated memory

return;

}

printf("Enter Initial Deposit Amount: ");

scanf("%f", &newCustomer->balance);

customers[numCustomers] = newCustomer;

numCustomers++;

printf("Customer added successfully!\n");

}

void updateCustomer() {

char accountNumber[20];

printf("Enter the Account Number of the customer to update: ");

scanf(" %s", accountNumber);

struct Customer \*customerToUpdate = findCustomerByAccountNumber(accountNumber);

if (customerToUpdate == NULL) {

printf("Customer with Account Number %s not found.\n", accountNumber);

return;

}

printf("Enter new Name: ");

scanf(" %[^\n]", customerToUpdate->name);

printf("Enter new Address: ");

scanf(" %[^\n]", customerToUpdate->address);

printf("Enter new Phone Number: ");

scanf(" %s", customerToUpdate->phone);

// Validate phone number

if (!isValidPhoneNumber(customerToUpdate->phone)) {

printf("Invalid phone number format. Please enter a valid phone number.\n");

return;

}

printf("Customer information updated successfully!\n");

}

struct Customer \*findCustomerByAccountNumber(const char \*accountNumber) {

int i;

for (i = 0; i < numCustomers; i++) {

if (strcmp(accountNumber, customers[i]->accountNumber) == 0) {

return customers[i];

}

}

return NULL;

}

void searchByAccountNumber() {

char accountNumber[20];

printf("Enter the Account Number of the customer to search: ");

scanf(" %s", accountNumber);

struct Customer \*customer = findCustomerByAccountNumber(accountNumber);

if (customer == NULL) {

printf("Customer with Account Number %s not found.\n", accountNumber);

return;

}

printf("Account Number: %s\n", customer->accountNumber);

printf("Name: %s\n", customer->name);

printf("IFSC Number: %s\n", customer->ifsc);

printf("Address: %s\n", customer->address);

printf("Phone Number: %s\n", customer->phone);

printf("Account Type: %s\n", customer->accountType);

printf("Balance: %.2f\n", customer->balance);

}

void searchByAccountName() {

char accountName[50];

printf("Enter the Account Name of the customer to search: ");

scanf(" %[^\n]", accountName);

int i;

int found = 0;

for (i = 0; i < numCustomers; i++) {

if (strcmp(accountName, customers[i]->name) == 0) {

printf("Account Number: %s\n", customers[i]->accountNumber);

printf("Name: %s\n", customers[i]->name);

printf("IFSC Number: %s\n", customers[i]->ifsc);

printf("Address: %s\n", customers[i]->address);

printf("Phone Number: %s\n", customers[i]->phone);

printf("Account Type: %s\n", customers[i]->accountType);

printf("Balance: %.2f\n", customers[i]->balance);

found = 1;

}

}

if (!found) {

printf("Customer with Account Name %s not found.\n", accountName);

}

}

void deposit() {

char accountNumber[20];

printf("Enter the Account Number to deposit money: ");

scanf(" %s", accountNumber);

struct Customer \*customer = findCustomerByAccountNumber(accountNumber);

if (customer == NULL) {

printf("Customer with Account Number %s not found.\n", accountNumber);

return;

}

float amount;

printf("Enter the amount to deposit: ");

scanf("%f", &amount);

customer->balance += amount;

printf("Deposit successful. Updated Balance: %.2f\n", customer->balance);

}

void withdraw() {

char accountNumber[20];

printf("Enter the Account Number to withdraw money: ");

scanf(" %s", accountNumber);

struct Customer \*customer = findCustomerByAccountNumber(accountNumber);

if (customer == NULL) {

printf("Customer with Account Number %s not found.\n", accountNumber);

return;

}

float amount;

printf("Enter the amount to withdraw: ");

scanf("%f", &amount);

if (amount > customer->balance) {

printf("Insufficient balance. Withdrawal not allowed.\n");

return;

}

customer->balance -= amount;

printf("Withdrawal successful. Updated Balance: %.2f\n", customer->balance);

}

void checkBalance() {

char accountNumber[20];

printf("Enter the Account Number to check balance: ");

scanf(" %s", accountNumber);

struct Customer \*customer = findCustomerByAccountNumber(accountNumber);

if (customer == NULL) {

printf("Customer with Account Number %s not found.\n", accountNumber);

return;

}

printf("Account Number: %s\n", customer->accountNumber);

printf("Name: %s\n", customer->name);

printf("Balance: %.2f\n", customer->balance);

}

void saveDataToFile() {

FILE \*file = fopen("customer\_data.txt", "w");

if (file == NULL) {

printf("Error opening file. Data not saved.\n");

return;

}

for (int i = 0; i < numCustomers; i++) {

fprintf(file, "%s;%s;%s;%s;%s;%s;%.2f\n", customers[i]->name, customers[i]->address,

customers[i]->phone, customers[i]->accountType, customers[i]->ifsc,

customers[i]->accountNumber, customers[i]->balance);

}

fclose(file);

printf("Data saved successfully.\n");

}

void loadDataFromFile() {

FILE \*file = fopen("customer\_data.txt", "r");

if (file == NULL) {

printf("Error opening file. Starting with an empty database.\n");

return;

}

char buffer[256];

while (fgets(buffer, sizeof(buffer), file)) {

struct Customer \*newCustomer = (struct Customer \*)malloc(sizeof(struct Customer));

sscanf(buffer, "%[^;];%[^;];%[^;];%[^;];%[^;];%[^;];%f", newCustomer->name, newCustomer->address,

newCustomer->phone, newCustomer->accountType, newCustomer->ifsc,

newCustomer->accountNumber, &newCustomer->balance);

customers[numCustomers] = newCustomer;

numCustomers++;

}

fclose(file);

printf("Data loaded successfully.\n");

}

int isValidIFSC(const char \*ifsc) {

// IFSC code must be exactly 11 characters long

if (strlen(ifsc) != 11) {

return 0; // Invalid length

}

// First four characters must be uppercase alphabets

for (int i = 0; i < 4; i++) {

if (!isupper(ifsc[i])) {

return 0; // Invalid character

}

}

// Fifth character must be 0 (zero)

if (ifsc[4] != '0') {

return 0; // Invalid character

}

// The remaining six characters can be alphanumeric

for (int i = 5; i < 11; i++) {

if (!isalnum(ifsc[i])) {

return 0; // Invalid character

}

}

return 1; // Valid IFSC code

}

int isValidPhoneNumber(const char \*phoneNumber) {

// Phone number must be exactly 10 digits long

if (strlen(phoneNumber) != 10) {

return 0; // Invalid length

}

// Check if all characters are digits

for (int i = 0; i < 10; i++) {

if (!isdigit(phoneNumber[i])) {

return 0; // Contains non-digit characters

}

}

return 1; // Valid phone number

}

int isValidAccountNumber(const char \*accountNumber) {

// Account number should not be empty

if (strlen(accountNumber) == 0) {

return 0; // Invalid length

}

// Check if all characters are digits

for (int i = 0; i < 14; i++) {

if (!isdigit(accountNumber[i])) {

return 0; // Contains non-digit characters

}

}

// You can add additional rules here based on the specific format required

// For example, check for a minimum and maximum length, or other format-specific rules

return 1; // Valid account number

}