CASE STUDY: PROGRAMMING 2

**Project Manager:**

John Mathew C. Parocha

**Front-End Developers:**

Corish Anne V. Arce

Summer Love Parativo

**Back-End Developers:**

Mary Anne B. Purawan

John Mathew C. Parocha

**QA System Tester**:

Patrick N. Panlilio

**Documentation:**

Aliah S. Del Rosario

**GROUP 4**

**BSIT 1-1**

**TABLE OF CONTENTS**

Contents

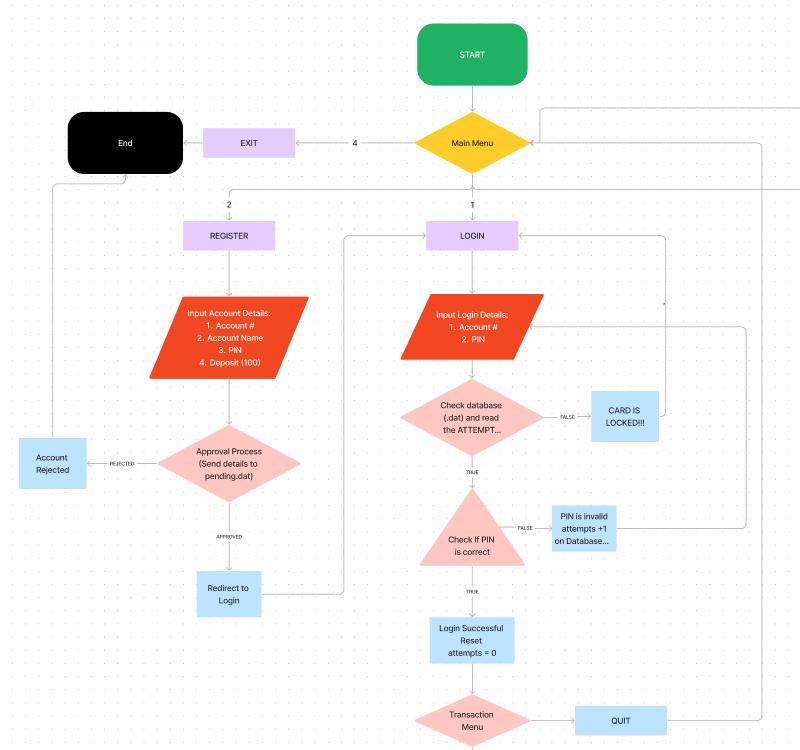
[FLOWCHART (FILE-HANDLING) 3](#_Toc170072092)

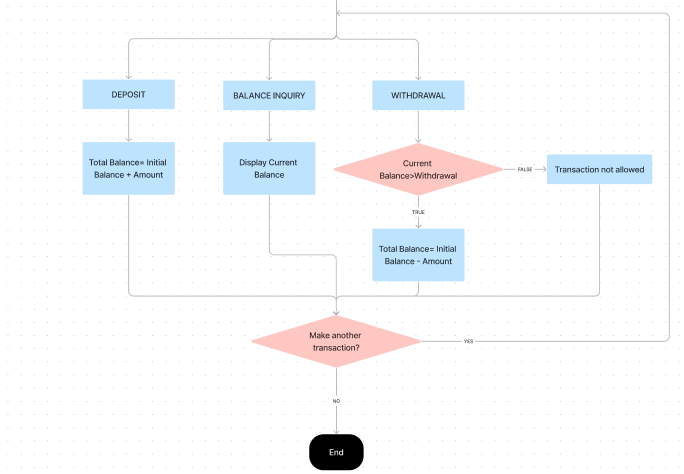
[TASK ALLOCATION AND TRACKING 4](#_Toc170072093)

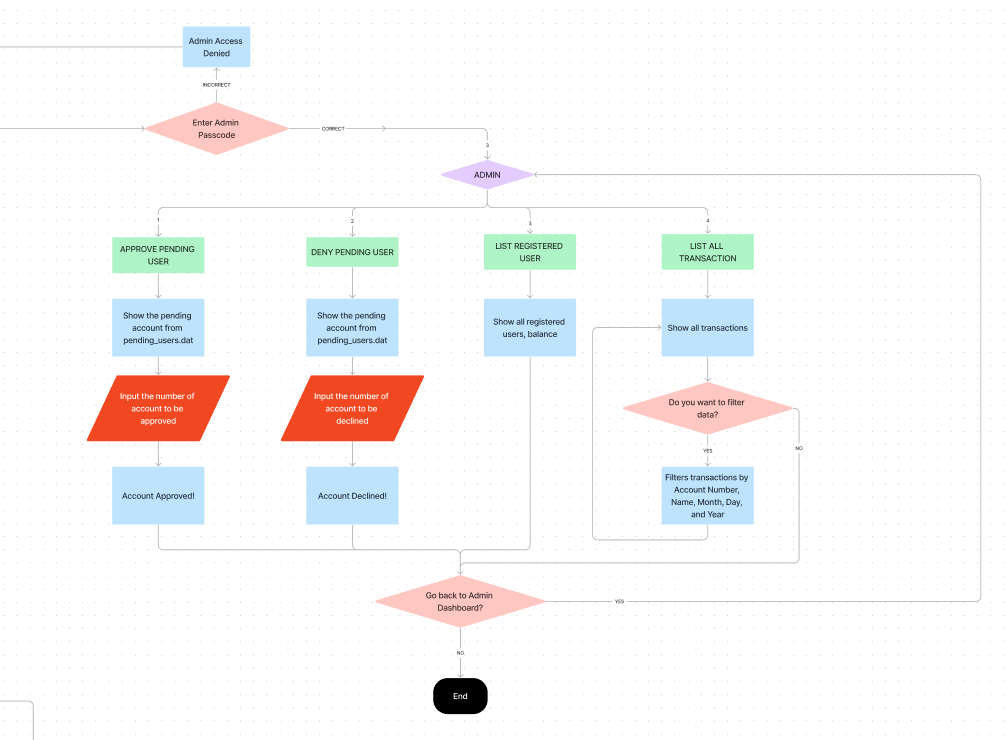
[Program (File-Handling) 5](#_Toc170072094)

[Program (Linked List) 33](#_Toc170072095)

FLOWCHART (FILE-HANDLING)







TASK ALLOCATION AND TRACKING

In this case study, we used Microsoft Excel in tracking the system’s progress-making. We were able to crawl-in the member’s process with their allocated tasks.

**A screenshot of a computer

Description automatically generated** **A screenshot of a computer

Description automatically generated**

Program (File-Handling)

1. Create a file handling menu driven program that will do the following:

1. Create ACCOUNTS.DAT  with the following fields

Account number 10 characters

Account name 25 characters

PIN 6 digits

Deposit 10 digits, inclusive of 2 decimal places

Transaction code 1 character

Amount 10 digits, inclusive of 2 decimal places

Account Balance 10 digits, inclusive of 2 decimal places

\*\*\*  Enter as much records as you want.

1. TRANSACTION PROCESSING

1. Determine if the account number entered matches the PIN.
2. If true, then display the message “VALID PIN!  The account belongs to <Account name>”.  Otherwise, display the message “INVALID PIN!” Only 3 tries is allowed to enter the correct PIN. Otherwise, display the message, “Card is captured! Please visit your bank branch to retrieve your card.” Then display the Main Menu.
3. If a valid pin was entered, the program must be able to simulate a deposit or withdrawal transaction.  Valid codes are B, D and W, Balance Inquiry, Deposit and Withdrawal respectively.
4. New balance = initial deposit (+/-) amount
5. If Deposit is less than the withdrawal, a remark “Not enough funds” will be displayed. Transaction will not be allowed.
6. PRINT RECORDS – displays all records in ACCOUNTS.DAT

Output layout:

Account Number Account Name Account  Balance

  xxxxxxxxxx AAAAAAAAAAAAAAAA     10000.00

  xxxxxxxxxx             BBBBBBBBBBBBBBBBB     22560.00

       and so on …

1. EXIT – close the program

Sample interface

1. File creation

Enter Account Number:

Enter Account Name:

Enter PIN:

Enter Deposit:

1. Transaction Processing

Enter Account Number:

Enter PIN:

When a valid PIN was entered

Enter Transaction Code: <B>, <D>, <W>

                  Create a menu for  Balance Inquiry, Deposit, Withdrawal

TRANSACTION MENU

1. BALANCE INQUIRY
2. DEPOSIT
3. WITHDRAWAL
4. QUIT

\*\*\* When option D is chosen, display the Main Menu

When an invalid PIN was entered, display the message “ INVALID PIN! You have \_\_ tries left”.  Up to three tries only.  If PIN entry  reached three tries, display the message “ CARD CAPTURED! Please get your card from your Bank Branch.” Then, Main Menu is displayed.

**PROGRAM:**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include <time.h>

// Structure for a pending user

struct PendingUser{

int AccountNumber;

char FirstName[50];

char LastName[50];

char AccountPassword[50];

int AccountPin;

double AccountBalance;

};

// Structure for a denied user

struct DeniedUser{

int AccountNumber;

char FirstName[50];

char LastName[50];

char AccountPassword[50];

int AccountPin;

double AccountBalance;

};

// Structure for a registered user

struct RegisteredUser{

int AccountNumber;

char FirstName[50];

char LastName[50];

char AccountPassword[50];

int AccountPin;

int Attempts;

double AccountBalance;

};

// Structure for a transactions

struct TransactionData{

int AccountNumber;

char FirstName[50];

char LastName[50];

char TransactionCode[2];

char DateSTR[11];

char TimeSTR[9];

double Amount;

};

// Functions to be used (Prototyping)

void ClearScreen();

void AdminMenu();

void RegisterUser();

void LoginUser();

void ApprovePendingUser();

void DenyPendingUser();

void ListRegisteredUser();

void ListTransact();

void FilterData();

void SearchData(int AN, char \*FN, char \*LN, char \*Month, char \*Date, char \*Year);

int Transaction(int AccountNumber, int EnteredPin);

int ExitFunction(char \*str);

// ANSI escape codes for colors and formatting

#define RESET "\033[0m"

#define BOLD "\033[1m"

#define ITALIC "\033[3m"

#define UNDERLINE "\033[4m"

#define BLACK "\033[30m"

#define RED "\033[31m"

#define GREEN "\033[32m"

#define YELLOW "\033[33m"

#define BLUE "\033[34m"

#define MAGENTA "\033[35m"

#define CYAN "\033[36m"

#define WHITE "\033[37m"

#define BG\_BLACK "\033[40m"

#define BG\_RED "\033[41m"

#define BG\_GREEN "\033[42m"

#define BG\_YELLOW "\033[43m"

#define BG\_BLUE "\033[44m"

#define BG\_MAGENTA "\033[45m"

#define BG\_CYAN "\033[46m"

#define BG\_WHITE "\033[47m"

int main() {

while (1) {

ClearScreen();

printf(GREEN "==============================================\n" RESET);

printf(WHITE BOLD BG\_GREEN "\t Welcome to the Bank \n" RESET);

printf(GREEN "==============================================\n" RESET);

printf(CYAN "\n\t\t1. Login\n" RESET);

printf(YELLOW "\t\t2. Register\n" RESET);

printf(MAGENTA "\t\t3. Admin\n" RESET);

printf(RED "\t\t4. Exit\n\n" RESET);

printf(GREEN "==============================================\n" RESET);

printf(WHITE BOLD "\nSelect a Choice (1-4): " RESET);

char input[10];

int choice;

scanf("%s", input);

// Check if the input is a valid digit between 1 and 4

if (isdigit(input[0]) && strlen(input) == 1 && input[0] >= '1' && input[0] <= '4') {

choice = atoi(input);

} else {

printf(RED "\nInvalid option! Please enter a number between 1 and 4. (Press any key to continue)\n" RESET);

getch();

continue;

}

switch (choice) {

case 1:

LoginUser();

break;

case 2:

RegisterUser();

break;

case 3:

AdminMenu();

break;

case 4:

printf(WHITE BG\_GREEN BOLD "\nThank you for using the Bank system. Goodbye!\n" RESET);

return 0;

default:

printf(RED "\nChoice Error: Please input a valid number! (Press any key to continue)\n" RESET);

getch(); // Wait for user input before clearing the screen

break;

}

}

return 0;

}

void AdminMenu() {

char input\_pass[50];

int login = 0;

while (1) { // Infinite loop until correct password is entered or 'x' is entered

ClearScreen();

printf(MAGENTA BOLD "=================================\n" RESET);

printf(MAGENTA BOLD " Admin Login Menu \n" RESET);

printf(MAGENTA BOLD "=================================\n" RESET);

printf(WHITE BOLD "Press X to Cancel\n" RESET);

printf(MAGENTA BOLD "Enter Admin Passcode : " RESET);

scanf("%s", input\_pass);

if (strcmp(input\_pass, "pass12345") == 0) {

printf(GREEN "\nSuccessfully Logged in!" RESET);

printf(YELLOW "\n(Press any Key to Continue)" RESET);

getch();

login = 1;

break; // Exit the loop after successful login

} else if (strcmp(input\_pass, "x") == 0) {

printf(GREEN"\nGoing Back to the Main Menu...\n");

printf(YELLOW"(Press any Key to Continue)");

getch();

break;

} else {

printf(RED "\nWrong password. Try again.\n" RESET);

printf(YELLOW"(Press any Key to Continue)");

getch();

}

}

if (login == 1) {

while (1) {

ClearScreen();

printf(MAGENTA BOLD "=================================\n" RESET);

printf(MAGENTA BOLD " Admin Menu \n" RESET);

printf(MAGENTA BOLD "=================================\n" RESET);

printf(GREEN "\n1. Approve Pending User\n" RESET);

printf(YELLOW "2. Deny Pending User\n" RESET);

printf(CYAN "3. List Registered User\n" RESET);

printf(WHITE "4. List of all Transactions\n" RESET);

printf(RED "5. Exit\n\n" RESET);

printf(MAGENTA BOLD "Select a Choice (1-5): " RESET);

char input[10];

int choice;

scanf("%s", input);

// Check if the input is a valid digit between 1 and 5

if (isdigit(input[0]) && strlen(input) == 1 && input[0] >= '1' && input[0] <= '5') {

choice = atoi(input);

} else {

printf(RED "\nInvalid option! Please enter a number between 1 and 5. (Press Any Key to Continue)\n" RESET);

getch();

continue;

}

switch (choice) {

case 1:

ApprovePendingUser();

break;

case 2:

DenyPendingUser();

break;

case 3:

ListRegisteredUser();

break;

case 4:

ListTransact();

break;

case 5:

printf(GREEN BOLD "\nExiting Admin Menu...\n" RESET);

getch();

return;

default:

printf(RED "\nChoice Error: Please input a valid number! (Press Any Key to Continue)\n" RESET);

getch(); // Wait for user input before clearing the screen

break;

}

}

}

}

void RegisterUser() {

struct RegisteredUser existingUser;

struct PendingUser newUser;

char enteredFirstName[50];

char enteredLastName[50];

char strpin[6]; // 6 digits

int NewAccNum;

FILE \*p\_userFile;

FILE \*userFile;

bool accountExists = false; // Flag to indicate if account number exists

ClearScreen();

// Open or create the users file in read/write mode to manage existing accounts

userFile = fopen("users.dat", "a+b");

if (userFile == NULL) {

printf(RED "Error opening users file!\n" RESET);

getch();

return;

}

// Open or create the pending users file in append mode to handle new registrations

p\_userFile = fopen("pending\_users.dat", "a+b");

if (p\_userFile == NULL) {

printf(RED "Error opening pending users file!\n" RESET);

fclose(userFile); // Close previously opened users file

getch();

return;

}

printf(YELLOW BOLD "================================\n" RESET);

printf(YELLOW BOLD " Registration Menu \n" RESET);

printf(YELLOW BOLD "================================\n" RESET);

printf(WHITE "Press X to Exit\n" RESET);

char EnteredNumber[10];

while (1) {

printf(YELLOW "\nEnter Account Number: " RESET);

scanf("%s", EnteredNumber);

int length = strlen(EnteredNumber);

int valid = 1;

// User Defined Function for Exiting Midway

if (ExitFunction(EnteredNumber) == 1){

return;

}

// Check if the entered number has 10 digits and is numeric

else if (length != 10) {

valid = 0;

} else {

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

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

valid = 0;

break;

}

}

}

if (valid == 1) {

NewAccNum = abs(atoi(EnteredNumber));

break;

} else {

printf(RED "\nPlease Enter a 10-digit Account Number!\n" RESET);

}

}

// Check if the account number already exists in pending\_users.dat

rewind(p\_userFile); // Reset file position indicator to the beginning

while (fread(&newUser, sizeof(struct PendingUser), 1, p\_userFile)) {

if (newUser.AccountNumber == NewAccNum) {

accountExists = true;

break;

}

}

// Check if the account number already exists in users.dat

rewind(userFile); // Reset file position indicator to the beginning

while (fread(&existingUser, sizeof(struct RegisteredUser), 1, userFile)) {

if (existingUser.AccountNumber == NewAccNum) {

accountExists = true;

break;

}

}

if (accountExists == true) {

printf(RED "\nAccount number already exists. Registration failed.\n" RESET);

printf(YELLOW "(Press Any Key to Continue)" RESET);

getch();

} else {

newUser.AccountNumber = NewAccNum;

// Proceed with registration if the account number is unique and write the new user to pending\_users.dat

printf(YELLOW "Enter First Name: " RESET);

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

if (ExitFunction(enteredFirstName) == 1){

return;

}

strcpy(newUser.FirstName, enteredFirstName);

printf(YELLOW "Enter Last Name: " RESET);

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

if (ExitFunction(enteredLastName) == 1){

return;

}

strcpy(newUser.LastName, enteredLastName);

printf(YELLOW "Enter Account Password: " RESET);

scanf("%s", newUser.AccountPassword);

if (ExitFunction(newUser.AccountPassword) == 1){

return;

}

while (1) {

printf(YELLOW "Enter Account Pin (6 digits): " RESET);

scanf("%s", strpin);

if (ExitFunction(strpin) == 1){

break;

}

if (strlen(strpin) == 6 &&

isdigit(strpin[0]) != 0 &&

isdigit(strpin[1]) != 0 &&

isdigit(strpin[2]) != 0 &&

isdigit(strpin[3]) != 0 &&

isdigit(strpin[4]) != 0 &&

isdigit(strpin[5]) != 0)

{

newUser.AccountPin = atoi(strpin);

break;

}

else {

printf(RED "\nPlease enter a 6-digit Pin!\n" RESET);

}

}

newUser.AccountBalance = 0.0;

char enteredInitialDeposit[10];

while (newUser.AccountBalance != 100.0) {

printf(YELLOW "Enter Initial Deposit (Exactly 100php): " RESET);

scanf("%s", enteredInitialDeposit);

if (ExitFunction(enteredInitialDeposit) == 1) {

break; // Exit function if required

}

// Convert enteredInitialDeposit to float using strtof

double deposit = strtod(enteredInitialDeposit, NULL);

// Check if the conversion was successful and the deposit is exactly 100 PHP

if (deposit == 100.0) {

newUser.AccountBalance = deposit;

break; // Exit the loop since the correct deposit is entered

} else {

printf(RED "\nPlease deposit Exactly 100php!\n" RESET);

}

}

system("cls");

printf(YELLOW BOLD "================================\n" RESET);

printf(YELLOW BOLD " Account Details\n(Please Save This Information!) \n" RESET);

printf(YELLOW BOLD "================================\n" RESET);

printf(RED "Might Have Changed Than Your Original to Prevent Scams\n");

printf(WHITE "Account Number : %d\n", newUser.AccountNumber);

printf("First Name : %s\n", newUser.FirstName);

printf("Last Name : %s\n", newUser.LastName);

printf("Password : %s\n", newUser.AccountPassword);

printf("Pin : %d\n", newUser.AccountPin);

printf("Initial Deposit : %.2f Php\n", newUser.AccountBalance);

printf(YELLOW BOLD "================================\n" RESET);

printf(GREEN "\nUser registration is now pending for Approval!\n");

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

getch();

// Write the new user to pending\_users.dat

fwrite(&newUser, sizeof(struct PendingUser), 1, p\_userFile);

} // end of else

// Close the files

fclose(p\_userFile);

fclose(userFile);

}

void LoginUser() {

struct RegisteredUser user;

FILE \*userFile, \*pendingFile;

int accountNumber, enteredPin;

char enteredPassword[50];

char input[7]; // Increased to 7 to accommodate 6 digits plus null terminator

int checkpass = 0;

int checkpin = 0;

int foundUser = 0;

bool accExists = false;

bool isPending = false;

// Open the file in read/write mode

userFile = fopen("users.dat", "r+b");

if (userFile == NULL) {

printf(RED "Error opening file!\n" RESET);

return;

}

// Open the pending users file in read-only mode

pendingFile = fopen("pending\_users.dat", "r+b");

if (pendingFile == NULL) {

printf(RED "Error opening pending users file!\n" RESET);

fclose(userFile);

return;

}

// Control Structure to ensure that the number entered is 10 DIGIT ONLY. Does not allow alphabet and Does not allow Special Characters

while (1) {

char strAN[11]; // Increased to 11 to accommodate 10 digits plus null terminator

ClearScreen();

printf(BLUE BOLD "=========================\n" RESET);

printf(BLUE BOLD " Login Menu \n" RESET);

printf(BLUE BOLD "=========================\n" RESET);

printf(WHITE BOLD "(Press X key to Exit)\n" RESET);

printf(CYAN "\nEnter Account Number: " RESET);

scanf("%s", strAN);

if (strcmp(strAN, "x") == 0 || strcmp(strAN, "X") == 0) {

printf(RED "\nGoing Back to the Menu...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

getch();

fclose(pendingFile);

fclose(userFile);

return;

}

else if (strlen(strAN) == 10 &&

isdigit(strAN[0]) && isdigit(strAN[1]) && isdigit(strAN[2]) &&

isdigit(strAN[3]) && isdigit(strAN[4]) && isdigit(strAN[5]) &&

isdigit(strAN[6]) && isdigit(strAN[7]) && isdigit(strAN[8]) &&

isdigit(strAN[9]))

{

accountNumber = atoi(strAN);

// Check if the user is in users.dat

rewind(userFile); // Reset file position indicator to the beginning

while (fread(&user, sizeof(struct RegisteredUser), 1, userFile)) {

if (user.AccountNumber == accountNumber) {

accExists = true;

break;

}

else

accExists = false;

}

break;

}

else {

printf(RED "\nPlease enter a 10-digit Account Number!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)" RESET);

getch();

}

} // End of control structure for account number

// Close pending file as it's no longer needed

fclose(pendingFile);

if (!accExists) {

printf(RED "\nThe Account Number that you have entered is not Registered Yet!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

getch();

fclose(userFile);

return;

}

while (!checkpass) {

printf(CYAN "Enter Password: " RESET);

scanf("%s", enteredPassword);

if (ExitFunction(enteredPassword) == 1) {

fclose(userFile);

return;

}

// Read the file to verify the password

rewind(userFile); // Reset file position indicator to the beginning

while (fread(&user, sizeof(struct RegisteredUser), 1, userFile)) {

if (user.AccountNumber == accountNumber) {

foundUser = 1;

if (strcmp(user.AccountPassword, enteredPassword) == 0) {

checkpass = 1;

}

break;

}

}

if (!checkpass) {

if (foundUser) {

printf(RED "Invalid Password!\n" RESET);

} else {

printf(RED "Account not found!\n" RESET);

fclose(userFile);

return;

}

}

}

while (!checkpin) {

if (user.Attempts >= 3) {

printf(RED "\nCard is captured! Please visit your bank branch to retrieve your card.\n" RESET);

printf(YELLOW "(Press Any Key to Continue)" RESET);

getch();

fclose(userFile);

return;

}

while (1) {

printf(CYAN "Enter Pin: " RESET);

scanf("%s", input);

if (ExitFunction(input) == 1) {

fclose(userFile);

return;

}

if (strlen(input) == 6 &&

isdigit(input[0]) && isdigit(input[1]) &&

isdigit(input[2]) && isdigit(input[3]) &&

isdigit(input[4]) && isdigit(input[5])) {

enteredPin = atoi(input);

if (user.AccountPin == enteredPin) {

checkpin = 1;

user.Attempts = 0;

// Move the file pointer back to the current record position

fseek(userFile, -sizeof(struct RegisteredUser), SEEK\_CUR);

// Update the record in the file

fwrite(&user, sizeof(struct RegisteredUser), 1, userFile);

fflush(userFile); // Ensure the write operation is completed

printf(GREEN BOLD "\nLogin Successful!\n" RESET);

printf(CYAN BOLD "Welcome, %s\n" RESET, user.FirstName);

getch();

// Call Transaction function and handle exit

int transactionResult = Transaction(accountNumber, enteredPin);

if (transactionResult == 0) {

// Transaction ended, return to stop further execution

fclose(userFile);

return;

}

} else {

user.Attempts += 1;

printf(RED "\nInvalid Pin! " RESET);

printf(YELLOW "Attempt %d/3\n" RESET, user.Attempts);

// Move the file pointer back to the current record position

fseek(userFile, -sizeof(struct RegisteredUser), SEEK\_CUR);

// Update the record in the file

fwrite(&user, sizeof(struct RegisteredUser), 1, userFile);

fflush(userFile); // Ensure the write operation is completed

}

} else {

printf(RED "\nPlease Enter a 6 Digit Pin!\n" RESET);

}

}

}

}

void ApprovePendingUser() {

FILE \*p\_userFile, \*userFile, \*tempFile;

int accountNumber;

char choice;

char TempName[100];

char TempLastName[50];

char input[10];

do {

struct PendingUser pendingUser;

struct RegisteredUser newUser;

int found = 0;

ClearScreen();

// Open the files and create if they don't exist

p\_userFile = fopen("pending\_users.dat", "rb");

userFile = fopen("users.dat", "ab");

if (p\_userFile == NULL || userFile == NULL) {

printf(RED "Error opening file!\n" RESET);

return;

}

// Display pending users

printf(GREEN BOLD "========================================\n" RESET);

printf(GREEN BOLD " Pending Users for Approval \n" RESET);

printf(GREEN BOLD "========================================\n" RESET);

while (fread(&pendingUser, sizeof(struct PendingUser), 1, p\_userFile)) {

printf(GREEN "Account Number: " WHITE "%d\n" RESET, pendingUser.AccountNumber);

strcpy(TempName, pendingUser.FirstName);

strcat(TempName, " ");

strcat(TempName, pendingUser.LastName);

printf(GREEN "Name: " WHITE "%s\n\n" RESET, TempName);

}

printf(GREEN BOLD "========================================\n" RESET);

printf(WHITE "\nPress X to Cancel\n" RESET);

printf(YELLOW "Enter Account Number to Approve: " RESET);

scanf("%s", input);

if (ExitFunction(input) == 1){

break;

}

accountNumber = atoi(input);

rewind(p\_userFile); // Reset file pointer position to the beginning

// Read pending users and transfer the approved one to registered users

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

while (fread(&pendingUser, sizeof(struct PendingUser), 1, p\_userFile)) {

if (pendingUser.AccountNumber == accountNumber) {

found = 1;

newUser.AccountNumber = pendingUser.AccountNumber;

strcpy(newUser.FirstName, pendingUser.FirstName);

strcpy(newUser.LastName, pendingUser.LastName);

strcpy(newUser.AccountPassword, pendingUser.AccountPassword);

newUser.AccountPin = pendingUser.AccountPin;

newUser.AccountBalance = pendingUser.AccountBalance;

newUser.Attempts = 0;

fwrite(&newUser, sizeof(struct RegisteredUser), 1, userFile);

} else {

fwrite(&pendingUser, sizeof(struct PendingUser), 1, tempFile);

}

}

fclose(p\_userFile);

fclose(userFile);

fclose(tempFile);

// Delete the original pending\_users.dat file and rename the temp file to pending\_users.dat

remove("pending\_users.dat");

rename("temp.dat", "pending\_users.dat");

if (found == 1)

printf(GREEN "User approved successfully!\n" RESET);

else

printf(RED "Account number not found!\n" RESET);

printf(WHITE BOLD "\nDo you want to approve another user? ("GREEN"Y"WHITE BOLD"/"RED"N"WHITE BOLD"): " RESET);

scanf(" %c", &choice); // Note the space before %c to consume the newline character

} while (choice == 'y' || choice == 'Y');

if (choice == 'n' || choice == 'N'){

printf(GREEN "\nGoing Back to the Admin Menu...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)..." RESET);

getch();

}

}

void DenyPendingUser() {

FILE \*p\_userFile, \*deniedFile, \*tempFile;

int accountNumber;

char choice;

char TempName[100];

char TempLastName[50];

char input[10];

do {

struct PendingUser pendingUser;

struct DeniedUser deniedUser;

int found = 0;

ClearScreen();

// Open the files

p\_userFile = fopen("pending\_users.dat", "rb");

deniedFile = fopen("denied\_users.dat", "ab");

if (p\_userFile == NULL || deniedFile == NULL) {

printf(RED "Error opening file!\n" RESET);

return;

}

// Display pending users

printf(RED BOLD "========================================\n" RESET);

printf(RED BOLD " Pending Users for Denial \n" RESET);

printf(RED BOLD "========================================\n" RESET);

while (fread(&pendingUser, sizeof(struct PendingUser), 1, p\_userFile)) {

printf(RED "Account Number: " WHITE "%d\n" RESET, pendingUser.AccountNumber);

strcpy(TempName, pendingUser.FirstName);

strcat(TempName, " ");

strcat(TempName, pendingUser.LastName);

printf(RED "Name: " WHITE "%s\n\n" RESET, TempName);

}

printf(RED BOLD "========================================\n" RESET);

printf(WHITE "\nPress X to Cancel\n" RESET);

printf(YELLOW "Enter Account Number to Approve: " RESET);

scanf("%s", input);

if (ExitFunction(input) == 1){

break;

}

accountNumber = atoi(input);

rewind(p\_userFile); // Reset file pointer position to the beginning

// Read pending users and transfer the denied one to denied users

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

while (fread(&pendingUser, sizeof(struct PendingUser), 1, p\_userFile)) {

if (pendingUser.AccountNumber == accountNumber) {

found = 1;

deniedUser.AccountNumber = pendingUser.AccountNumber;

strcpy(deniedUser.FirstName, pendingUser.FirstName);

strcpy(deniedUser.LastName, pendingUser.LastName);

strcpy(deniedUser.AccountPassword, pendingUser.AccountPassword);

deniedUser.AccountPin = pendingUser.AccountPin;

deniedUser.AccountBalance = pendingUser.AccountBalance;

fwrite(&deniedUser, sizeof(struct DeniedUser), 1, deniedFile);

} else {

fwrite(&pendingUser, sizeof(struct PendingUser), 1, tempFile);

}

}

fclose(p\_userFile);

fclose(deniedFile);

fclose(tempFile);

// Delete the original pending\_users.dat file and rename the temp file to pending\_users.dat

remove("pending\_users.dat");

rename("temp.dat", "pending\_users.dat");

if (found == 1)

printf(RED "User denied successfully!\n" RESET);

else

printf(RED "Account number not found!\n" RESET);

printf(WHITE BOLD "\nDo you want to deny another user? ("GREEN"Y"WHITE BOLD"/"RED"N"WHITE BOLD"): " RESET);

scanf(" %c", &choice); // Note the space before %c to consume the newline character

} while (choice == 'y' || choice == 'Y');

if (choice == 'n' || choice == 'N'){

printf(GREEN "\nGoing Back to the Admin Menu...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)..." RESET);

getch();

}

}

void ListRegisteredUser() {

FILE \*userFile;

struct RegisteredUser user;

// Open the users file in read mode

userFile = fopen("users.dat", "rb");

if (userFile == NULL) {

printf(CYAN "Error opening file!\n" RESET);

return;

}

// Print header

printf(CYAN BOLD "========================================================================================================================\n" RESET);

printf(GREEN BOLD "| %-20s "CYAN"|"BLUE BOLD" %-25s "CYAN"|"MAGENTA BOLD" %-15s "CYAN"|"WHITE" %-10s "CYAN"|"YELLOW BOLD" %-15s "CYAN"|"RED BOLD" %-5s "CYAN"|\n" RESET, "Account Number", "Name", "Password", "PIN", "Balance", "Attempt");

printf(CYAN BOLD "========================================================================================================================\n" RESET);

// Read and print each registered user

while (fread(&user, sizeof(struct RegisteredUser), 1, userFile)) {

printf(WHITE "| %20d | %-25s | %-15s | %10d | %15.2f | %5d |\n" RESET, user.AccountNumber, user.FirstName, user.AccountPassword, user.AccountPin, user.AccountBalance, user.Attempts);

}

printf(CYAN BOLD "========================================================================================================================\n" RESET);

fclose(userFile);

printf(YELLOW "(Press Any Key to Continue)..." RESET);

getch();

}

int Transaction(int AccountNumber, int EnteredPin) {

struct RegisteredUser user;

struct TransactionData transaction;

time\_t rawtime;

struct tm \*timeinfo;

char date[11]; // For storing date in the format YYYY-MM-DD

char time\_str[9]; // For storing time in the format HH:MM:SS

FILE \*userFile;

FILE \*transFile;

int quit = 0;

double amount;

char str[10];

bool foundUser = false;

// Open the users file in read/write mode

userFile = fopen("users.dat", "r+b");

if (userFile == NULL) {

printf(CYAN "Error opening users file!\n" RESET);

return -1;

}

// Open the transactions file in append mode, and create it if it doesn't exist

transFile = fopen("transactions.dat", "a+b");

if (transFile == NULL) {

printf(CYAN "Error opening transactions file!\n" RESET);

fclose(userFile);

return -1;

}

// Find the user in the file

while (fread(&user, sizeof(struct RegisteredUser), 1, userFile)) {

if (user.AccountNumber == AccountNumber && user.AccountPin == EnteredPin) {

foundUser = true;

break;

}

}

if (!foundUser) {

printf(CYAN "Account not found!\n" RESET);

fclose(userFile);

fclose(transFile);

return -1;

}

while (quit == 0) {

system("cls");

printf(CYAN BOLD "========================================\n" RESET);

printf(CYAN BOLD " Welcome %s! \n" RESET, user.FirstName);

printf(CYAN BOLD "========================================\n" RESET);

printf(GREEN "1. Deposit\n");

printf(YELLOW "2. Withdraw\n");

printf(CYAN "3. Balance Inquiry\n");

printf(MAGENTA "4. Transaction History\n");

printf(RED "5. Exit\n");

printf(CYAN BOLD "========================================\n" RESET);

printf("Select an option: ");

int choice = 0;

char input[10];

scanf("%s", input);

// Check if the input is a valid digit

if (isdigit(input[0]) && strlen(input) == 1) {

choice = atoi(input);

} else {

printf(CYAN "Invalid option! Please enter a number between 1 and 5.\n" RESET);

getch();

continue;

}

// Reset TotalDeposit and TotalWithdrawal for each iteration

float TotalDeposit = 0, TotalWithdrawal = 0;

// Rewind the transaction file pointer to the beginning

rewind(transFile);

while (fread(&transaction, sizeof(struct TransactionData), 1, transFile)) {

if (user.AccountNumber == AccountNumber && (strcmp(transaction.TransactionCode, "D") == 0))

TotalDeposit += transaction.Amount;

else if (user.AccountNumber == AccountNumber && (strcmp(transaction.TransactionCode, "W") == 0))

TotalWithdrawal += transaction.Amount;

}

switch (choice) {

case 1: // Deposit

printf(CYAN BOLD "========================================\n" RESET);

printf(WHITE BOLD "\n(Enter X key to Cancel Deposit)" RESET);

printf(GREEN "\nEnter Amount to Deposit: " RESET);

scanf("%s", str);

if (ExitFunction(str) == 1){

break;

}

else if (strlen(str) > 10) {

printf(RED "\nInput exceeds maximum length of 10 digit!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

break;

}

//int isValid = 1;

//for (int i = 0; i < strlen(str); ++i) {

//if (!isdigit(str[i])) {

//isValid = 0;

//break;

//}

//}

//if (isdigit(str[0]) == 0 && isdigit(str[1]) == 0 &&

//isdigit(str[2]) == 0 && isdigit(str[3]) == 0 &&

//isdigit(str[4]) == 0 && isdigit(str[5]) == 0 &&

//isdigit(str[6]) == 0 && isdigit(str[7]) == 0 &&

//isdigit(str[8]) == 0 && isdigit(str[9]) == 0 ){

// printf(RED "\nPlease Only Enter a Numerical Character!\n" RESET);

// break;

//}

else{

amount = strtod(str, NULL);

if (amount <= 0){

printf(RED "\nPlease Enter a Positive Amount!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

}

else {

user.AccountBalance += amount;

printf(GREEN "Deposit successful! \n"YELLOW"New balance: %.2f\n" RESET, user.AccountBalance);

// Record the transaction

transaction.AccountNumber = user.AccountNumber;

strcpy(transaction.FirstName, user.FirstName);

strcpy(transaction.LastName, user.LastName);

strcpy(transaction.TransactionCode, "D"); // 'D' for Deposit

transaction.Amount = amount;

// Get the current time

time(&rawtime);

// Convert it to local time representation

timeinfo = localtime(&rawtime);

// Format the date and store it in the date array

strftime(date, sizeof(date), "%Y-%m-%d", timeinfo);

// Format the time and store it in the time array

strftime(time\_str, sizeof(time\_str), "%H:%M:%S", timeinfo);

// Copy date and time to Data Structure

strcpy(transaction.DateSTR, date);

strcpy(transaction.TimeSTR, time\_str);

fwrite(&transaction, sizeof(struct TransactionData), 1, transFile);

// Update the user's balance in the file

fseek(userFile, -sizeof(struct RegisteredUser), SEEK\_CUR);

fwrite(&user, sizeof(struct RegisteredUser), 1, userFile);

fflush(userFile); // Ensure the write operation is completed

}

}

break; // end of case 1

case 2: // Withdraw

printf(CYAN BOLD "========================================\n" RESET);

printf(WHITE BOLD "\n(Enter X key to Cancel Deposit)" RESET);

printf(YELLOW "\nEnter Amount to Withdraw: " RESET);

scanf("%s", str);

if (ExitFunction(str) == 1){

break;

}

else if (strlen(str) > 10) {

printf(RED "\nInput exceeds maximum length of 10 digit!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

break;

}

amount = strtod(str, NULL);

TotalWithdrawal += amount;

if (amount <= 0){

printf(RED "\nPlease Enter a Positive Amount!\n" RESET);

break;

}

else if (TotalDeposit < TotalWithdrawal) {

printf(RED "Insufficient funds!\n" RESET);

break;

} else {

user.AccountBalance -= amount;

printf(CYAN "Withdrawal successful! "YELLOW"New balance: %.2f\n" RESET, user.AccountBalance);

// Record the transaction

transaction.AccountNumber = user.AccountNumber;

strcpy(transaction.FirstName, user.FirstName);

strcpy(transaction.LastName, user.LastName);

strcpy(transaction.TransactionCode, "W"); // 'W' for Withdraw

transaction.Amount = amount;

// Get the current time

time(&rawtime);

// Convert it to local time representation

timeinfo = localtime(&rawtime);

// Format the date and store it in the date array

strftime(date, sizeof(date), "%Y-%m-%d", timeinfo);

// Format the time and store it in the time array

strftime(time\_str, sizeof(time\_str), "%H:%M:%S", timeinfo);

// Copy date and time to Data Structure

strcpy(transaction.DateSTR, date);

strcpy(transaction.TimeSTR, time\_str);

fwrite(&transaction, sizeof(struct TransactionData), 1, transFile);

// Update the user's balance in the file

fseek(userFile, -sizeof(struct RegisteredUser), SEEK\_CUR);

fwrite(&user, sizeof(struct RegisteredUser), 1, userFile);

fflush(userFile); // Ensure the write operation is completed

}

break;

case 3: // View Balance

printf(CYAN BOLD "========================================\n" RESET);

printf(GREEN "\nYour total deposit: "WHITE"%.2lf\n", TotalDeposit);

printf(YELLOW"Your total withdrawal: "WHITE"%.2lf\n", TotalWithdrawal);

printf(CYAN"Your current balance: "WHITE"%.2lf\n" RESET, user.AccountBalance);

break;

case 4: // See Transaction History

// Print header

printf(CYAN BOLD "========================================\n" RESET);

printf(CYAN BOLD "\nTransaction History:\n" RESET);

printf(CYAN BOLD "--------------------------------------------------------------\n" RESET);

printf(CYAN BOLD "|"WHITE" %15s "CYAN BOLD"|"WHITE" %15s "CYAN BOLD"|"WHITE" %10s "CYAN BOLD" |"WHITE" %8s "CYAN BOLD"|\n" RESET, "T-Code", "Amount", "Date", "Time");

printf(CYAN BOLD "--------------------------------------------------------------\n" RESET);

// Rewind the transaction file pointer to the beginning

rewind(transFile);

// Read and print each transaction

while (fread(&transaction, sizeof(struct TransactionData), 1, transFile)) {

if (transaction.AccountNumber == AccountNumber) {

if (strcmp(transaction.TransactionCode, "D") == 0) {

printf(CYAN BOLD "|"GREEN" %15s "CYAN BOLD"|"YELLOW" %15.2lf+ "CYAN BOLD"|"WHITE" %10s "CYAN BOLD"|"WHITE" %8s "CYAN BOLD"|\n" RESET, "Deposit", transaction.Amount, transaction.DateSTR, transaction.TimeSTR);

} else if (strcmp(transaction.TransactionCode, "W") == 0) {

printf(CYAN BOLD "|"RED" %15s "CYAN BOLD"|"YELLOW" %15.2lf- "CYAN BOLD"|"WHITE" %10s "CYAN BOLD"|"WHITE" %8s "CYAN BOLD"|\n" RESET, "Withdrawal", transaction.Amount, transaction.DateSTR, transaction.TimeSTR);

}

}

}

printf(CYAN BOLD "--------------------------------------------------------------\n" RESET);

break;

case 5: // Exit

printf(RED "\nExiting...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

quit = 1; // Set quit to 1 to exit the loop

break;

default:

printf(RED BOLD"\nInvalid option!\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

quit = 0;

break;

}

getch();

}

fclose(userFile);

fclose(transFile);

return 0;

}

void ListTransact() {

struct TransactionData transaction;

FILE \*transFile;

// Open the transactions file in read mode

transFile = fopen("transactions.dat", "rb");

if (transFile == NULL) {

printf(MAGENTA "Error opening transactions file!\n" RESET);

return;

}

// Print header

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf(MAGENTA BOLD " All Transactions in the Bank \n" RESET);

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf(YELLOW " %-20s "MAGENTA"|"CYAN BOLD" %-25s "MAGENTA"| "GREEN"%-6s "MAGENTA"|"WHITE" %-15s "MAGENTA"|"WHITE" %-12s "MAGENTA"|"YELLOW BOLD" %-8s "MAGENTA"|\n", "Account Number", "Name", "T-Code", "Date", "Time", "Amount");

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

// Read and print each transaction

while (fread(&transaction, sizeof(struct TransactionData), 1, transFile)) {

char fullName[100]; // Assuming the combined length of first name and last name won't exceed 100 characters

strcpy(fullName, transaction.FirstName); // Copy first name to fullName

strcat(fullName, " "); // Add a space

strcat(fullName, transaction.LastName); // Concatenate last name to fullName

// Format and print each transaction row

if (strcmp(transaction.TransactionCode, "D") == 0) {

printf(WHITE " %-20d "MAGENTA"|"WHITE" %-25s "MAGENTA"| " GREEN "%-6s" MAGENTA " |"WHITE" %-15s "MAGENTA"|"WHITE" %-12s "MAGENTA"| " WHITE "%-8.2lf "MAGENTA"|\n" RESET, transaction.AccountNumber, fullName, "Deposit", transaction.DateSTR, transaction.TimeSTR, transaction.Amount);

} else if (strcmp(transaction.TransactionCode, "W") == 0) {

printf(WHITE " %-20d "MAGENTA"|"WHITE" %-25s "MAGENTA"| " RED "%-6s" MAGENTA " |"WHITE" %-15s "MAGENTA"|"WHITE" %-12s "MAGENTA"| " WHITE "%-8.2lf "MAGENTA"|\n" RESET, transaction.AccountNumber, fullName, "Withdraw", transaction.DateSTR, transaction.TimeSTR, transaction.Amount);

} else {

printf(WHITE " %-20d "MAGENTA"|"WHITE" %-25s "MAGENTA"| " YELLOW "%-6s" MAGENTA " |"WHITE" %-15s "MAGENTA"|"WHITE" %-12s "MAGENTA"| " WHITE "%-8.2lf "MAGENTA"|\n" RESET, transaction.AccountNumber, fullName, "Unknown", transaction.DateSTR, transaction.TimeSTR, transaction.Amount);

}

}

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

// Prompt for filtering data

while (1) {

char decision;

printf(YELLOW BOLD "\nWould you like to Filter Data? ("GREEN"Y"WHITE"/"RED"N"YELLOW BOLD"): " RESET);

scanf(" %c", &decision); // Notice the space before %c to consume any leading whitespace

if (isalpha(decision)) {

if (decision == 'Y' || decision == 'y') {

printf(GREEN"\nFiltering data... "YELLOW"\n(Press Any Key to Continue)...\n");

getch();

FilterData();

break; // Exit the loop after filtering

} else if (decision == 'N' || decision == 'n') {

break; // Exit the loop if no filtering is required

} else {

printf(YELLOW"\nInvalid input. Please enter "GREEN"Y "YELLOW"or "RED"N"YELLOW".\n");

}

} else {

printf(YELLOW"\nInvalid input. Please enter "GREEN"Y "YELLOW"or "RED"N"YELLOW".\n");

}

}

printf(GREEN "\nReturning to Admin Menu! "YELLOW"(Press Any Key to Proceed)..." RESET);

getch();

fclose(transFile);

}

void FilterData() {

int enteredAN = 0000000000; //1231231231

char enteredFN[50] = "NULL"; //

char enteredLN[50] = "NULL";

char enteredDATE[5] = "NULL";

char enteredMONTH[5] = "NULL";

char enteredYEAR[5] = "NULL";

while (1) {

system("cls");

SearchData(enteredAN, enteredFN, enteredLN, enteredMONTH, enteredDATE, enteredYEAR);

printf(MAGENTA BOLD "\n===========================================================================================================\n" RESET);

printf(MAGENTA BOLD " Search for the Following Data \n" RESET);

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf(GREEN"(Press Any Number to Edit Input)\n");

printf(CYAN "1. Account Number : %d\n", enteredAN);

printf(CYAN BOLD "2. First Name : %s\n", enteredFN);

printf(CYAN BOLD "3. Last Name : %s\n", enteredLN);

printf(YELLOW "4. Month : %s\n", enteredMONTH);

printf(YELLOW "5. Date : %s\n", enteredDATE);

printf(YELLOW "6. Year : %s\n", enteredYEAR);

printf(RED "7. Exit\n" RESET);

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf("\nSelect a Choice: ");

int choice = 0;

char input[10];

scanf("%s", input);

// Check if the input is a valid digit

if (isdigit(input[0]) && strlen(input) == 1) {

choice = atoi(input);

} else {

printf(RED "Invalid option! Please enter a number between 1 and 7. "YELLOW"(Press Any Key to Continue)\n" RESET);

getch();

continue;

}

switch (choice) {

case 1:

printf(CYAN "Enter the Account Number that you would like to search: ");

scanf("%d", &enteredAN);

break;

case 2:

printf(CYAN BOLD "Enter the First Name that you would like to search: ");

scanf("%s", enteredFN);

break;

case 3:

printf(CYAN BOLD "Enter the Last Name that you would like to search: ");

scanf("%s", enteredLN);

break;

case 4:

printf(YELLOW "Enter the Month that you would like to search: ");

scanf("%s", enteredMONTH);

break;

case 5:

printf(YELLOW "Enter the Date that you would like to search: ");

scanf("%s", enteredDATE);

break;

case 6:

printf(YELLOW "Enter the Year that you would like to search: ");

scanf("%s", enteredYEAR);

break;

case 7:

return;

default:

printf(RED "Invalid Choice! "YELLOW"(Press Any Key to Continue)...\n" RESET);

getch();

break;

}

}

}

void SearchData(int AN, char \*FN, char \*LN, char \*Month, char \*Date, char \*Year) {

struct TransactionData transaction;

FILE \*transFile;

// Open the transactions file in read mode

transFile = fopen("transactions.dat", "rb");

if (transFile == NULL) {

printf(RED "Error opening transactions file!\n" RESET);

return;

}

// Print header

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf(MAGENTA BOLD " All Transactions in the Bank \n" RESET);

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

printf(YELLOW " %-20s " MAGENTA "|" CYAN BOLD " %-25s " MAGENTA "| %-6s " MAGENTA "| %-15s " MAGENTA "| %-12s " MAGENTA "| %-10s |\n" RESET, "Account Number", "Name", "T-Code", "Date", "Time", "Amount");

printf(MAGENTA BOLD "===========================================================================================================\n" RESET);

// Read and print each transaction

while (fread(&transaction, sizeof(struct TransactionData), 1, transFile)) {

int match = 1; // Flag to check if the transaction matches the search criteria

// Check each search condition

if (AN != 0000000000 && transaction.AccountNumber != AN) {

match = 0;

}

if (strcmp(FN, "NULL") != 0 && strcmp(transaction.FirstName, FN) != 0) {

match = 0;

}

if (strcmp(LN, "NULL") != 0 && strcmp(transaction.LastName, LN) != 0) {

match = 0;

}

if (strcmp(Date, "NULL") != 0 && strncmp(transaction.DateSTR + 8, Date + 0, 2) != 0) {

match = 0;

}

if (strcmp(Month, "NULL") != 0 && strncmp(transaction.DateSTR + 5, Month + 0, 2) != 0){

match = 0;

}

if (strcmp(Year, "NULL") != 0 && strncmp(transaction.DateSTR, Year, 4) != 0) {

match = 0;

}

if (match == 1) {

char fullName[100]; // Assuming the combined length of first name and last name won't exceed 100 characters

strcpy(fullName, transaction.FirstName); // Copy first name to fullName

strcat(fullName, " "); // Add a space

strcat(fullName, transaction.LastName); // Concatenate last name to fullName

// Print formatted transaction data

printf(WHITE " %-20d " MAGENTA "|" WHITE " %-25s " MAGENTA "| " WHITE "%-6s" MAGENTA " |" WHITE " %-15s " MAGENTA "|"WHITE" %-12s " MAGENTA "| " WHITE "%-10.2lf " MAGENTA "|\n" RESET,

transaction.AccountNumber, fullName, transaction.TransactionCode, transaction.DateSTR, transaction.TimeSTR, transaction.Amount);

}

}

fclose(transFile);

}

void ClearScreen() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

int ExitFunction(char \*str){

if (strcmp(str, "x") == 0){

printf(RED "\nGoing Back to the Menu...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

getch();

return 1; // Indicate exit

}

else if (strcmp(str, "X") == 0){

printf(RED "\nGoing Back to the Menu...\n" RESET);

printf(YELLOW "(Press Any Key to Continue)\n" RESET);

getch();

return 1; // Indicate exit

}

else

return 0; // Indicate continue

}

Program (Linked List)

Create a menu driven program which will have the following choices:

                       MAIN MENU

1. Create a Linear Linked List
2. Display a Linear Linked List
3. Insert a number in the Linear Linked List
4. Delete a number from the Linear Linked List
5. Search a number
6. Exit

Enter choice:

Requirements:

* The Main Menu should be displayed after the execution of a choice
* Create a Linear Linked List first to enable the other options.
* In the Insert a number option, the number to be inserted should be placed in its proper place in the sorted linear linked list.  Ask for another number to be inserted until ‘N’ or ‘n’ is entered.

**Note:  A number should be inserted in a sorted list, therefore, a number cannot be inserted at the end of the linked list and then be sorted.**

* In the Delete a number option, user is prompted to enter a number to be deleted.  If the number is not in the linear linked list, then, a message will be displayed “ Number to be deleted is not in the list”.  Otherwise, the number and all numbers with the same value should be deleted from the list. Ask for another number to be deleted until ‘N’ or ‘n’ is entered.
* In the Search a number option, user is prompted to enter a number to be searched until user enters ‘N’ or ‘n’ in the prompt “Do you want to search another number? “
* If the number searched is in the list, then display the message “ <number> is in the list”. Otherwise, display the message “ <number> is not in the list”.
* In the Search a number option, user is prompted to enter a number to be searched until user enters ‘N’ or ‘n’ in the prompt “Do you want to search another number? “
* If the number searched is in the list, then display the message “ <number> is in the list”. Otherwise, display the message “ <number> is not in the list”.

**PROGRAM:**

#include <conio.h>

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define the structure of a node in the linked list

struct node {

int x; // Value of the node

struct node \*next; // Pointer to the next node in the list

};

// Declare head, curr, and tail pointers for the linked list

struct node \*head, \*curr, \*tail;

// Function declarations

void CreateList();

void SortList();

void DisplayList();

void InsertNode();

void DeleteNode();

void SearchNode();

int main() {

int choice; // Variable to store the user's menu choice

char input[10]; // Buffer to store the user's input

while (1) { // Infinite loop to keep the menu running until the user decides to exit

system("cls"); // Clear the screen

printf("Welcome to the Linked List Program\n\n");

printf("1. Create a Linear Linked List\n");

printf("2. Display a Linear Linked List\n");

printf("3. Insert a number in the Linear Linked List\n");

printf("4. Delete a number in the Linear Linked List\n");

printf("5. Search a number\n");

printf("6. Exit\n");

printf("\nSelect a Choice : ");

scanf("%s", input); // Read the user's input

// Check if the input is a valid single digit

if (isdigit(input[0]) && strlen(input) == 1) {

choice = atoi(input); // Convert input to an integer

} else {

printf("Invalid option! Please enter a number between 1 and 6. (Press Any Key to Continue)\n");

getch(); // Wait for user input before continuing

continue; // Continue the loop to prompt the user again

}

// Switch case to handle the user's menu choice

switch (choice) {

case 1:

CreateList(); // Create a linked list

break;

case 2:

DisplayList(); // Display the linked list

break;

case 3:

InsertNode(); // Insert a node into the linked list

break;

case 4:

DeleteNode(); // Delete a node from the linked list

break;

case 5:

SearchNode(); // Search for a node in the linked list

break;

case 6:

printf("Exiting the program...");

getch(); // Wait for user input before exiting

return 0; // Exit the program

default:

printf("\nChoice Error: Please input a valid number! (Press Any Key to Continue)\n");

getch(); // Wait for user input before continuing

}

}

}

// Function to create a linked list

void CreateList() {

printf("\nEnter 'X' to Exit\n");

head = curr = tail = NULL; // Initialize the head, curr, and tail pointers to NULL

while (1) { // Infinite loop to continuously accept user input

int isValid = 0; // Flag to check if the input is valid

while (1) { // Inner loop for input validation

char input[10]; // Buffer to store the user's input

printf("\nEnter a value for X: ");

scanf("%s", input); // Read the user's input

// Check if input is 'X' or 'x' to exit

if (strcasecmp(input, "X") == 0) {

goto exit\_loop; // Exit both loops if the user enters 'X'

}

// Check if input is a valid integer (positive or negative)

isValid = 1; // Assume the input is valid

int startIdx = 0;

if (input[0] == '-') {

startIdx = 1; // Start checking digits from index 1 if there's a negative sign

}

for (int i = startIdx; i < strlen(input); i++) {

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

isValid = 0; // Set isValid to 0 if any character is not a digit

break;

}

}

if (isValid) {

// Allocate memory for the new node only after validation

curr = (struct node \*)malloc(sizeof(struct node));

if (curr == NULL) {

printf("Memory allocation failed!\n");

return;

}

curr->x = atoi(input); // Convert input to an integer and assign to the node

break; // Exit the inner loop if a valid integer is entered

} else {

printf("\nPlease enter a valid integer!\n");

}

}

// Insert the new node into the linked list

if (head == NULL) { // If the list is empty, set the new node as the head

head = curr;

head->next = NULL;

tail = curr;

} else { // Otherwise, append the new node to the end of the list

tail->next = curr;

curr->next = NULL;

tail = curr;

}

}

exit\_loop:

// If head is NULL, then no data was entered

if (head == NULL) {

printf("No record/s entered!");

goto tapos;

}

// Print the data in the linked list

printf("\n\nLinked list data:\n");

curr = head; // Start from the head node

while (curr != NULL) {

printf("%3d ", curr->x);

curr = curr->next; // Move to the next node

}

// Sort the list after creation

SortList();

// Print the sorted data in the linked list

printf("\n\nSorted list data:\n");

curr = head; // Start from the head node

while (curr != NULL) {

printf("%3d ", curr->x);

curr = curr->next; // Move to the next node

}

printf("\n\n(Press Any Key to go back to the Menu).");

tapos:

getch(); // Wait for user input before continuing

}

// Function to sort the linked list in ascending order

void SortList() {

int temp; // Temporary variable for swapping values

if (head == NULL) { // If the list is empty, print a message and return

printf("List is empty.\n");

return;

}

curr = head; // Start from the head node

while (curr->next != NULL) {

tail = curr->next; // Set tail to the next node

while (tail != NULL) {

if (curr->x > tail->x) { // If the current node's value is greater than the next node's value

// Swap the values of the nodes

temp = curr->x;

curr->x = tail->x;

tail->x = temp;

}

tail = tail->next; // Move to the next node

}

curr = curr->next; // Move to the next node

}

}

// Function to display the linked list

void DisplayList() {

printf("\n\nLinked list data:\n");

curr = head; // Start from the head node

while (curr != NULL) {

printf("%3d ", curr->x);

curr = curr->next; // Move to the next node

}

getch(); // Wait for user input before continuing

}

// Function to insert a node into the linked list in sorted order

void InsertNode() {

char choice = 'Y';

while (choice == 'Y' || choice == 'y') {

int num; // Variable to store the number to be inserted

int isValid = 0; // Flag to check if the input is valid

while (1) { // Inner loop for input validation

char input[10]; // Buffer to store the user's input

printf("\nEnter a number to insert: ");

scanf("%s", input); // Read the user's input

// Check if input is a valid integer (positive or negative)

isValid = 1; // Assume the input is valid

int startIdx = 0;

if (input[0] == '-') {

startIdx = 1; // Start checking digits from index 1 if there's a negative sign

}

for (int i = startIdx; i < strlen(input); i++) {

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

isValid = 0; // Set isValid to 0 if any character is not a digit

break;

}

}

if (isValid) {

num = atoi(input); // Convert input to an integer

break; // Exit the inner loop if a valid integer is entered

} else {

printf("\nPlease enter a valid integer!\n");

}

}

// Create a new node

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

if (new\_node == NULL) {

printf("Memory allocation failed!\n");

return;

}

new\_node->x = num; // Assign the number to the new node

new\_node->next = NULL;

// Insert the new node into the linked list in sorted order

if (head == NULL || head->x >= num) {

// Insert at the beginning of the list

new\_node->next = head;

head = new\_node;

} else {

// Traverse the list to find the insertion position

struct node \*prev = head;

struct node \*curr = head->next;

while (curr != NULL && curr->x < num) {

prev = curr;

curr = curr->next;

}

// Insert the new node between prev and curr

prev->next = new\_node;

new\_node->next = curr;

}

printf("Number %d inserted into the list.\n", num);

printf("\nDo you want to insert another number? (Y/N) : ");

scanf(" %c", &choice); // Read the user's choice

}

printf("\n(Press Any Key to Continue)\n");

printf("Exiting the Insertion Program...");

getch(); // Wait for user input before continuing

}

// Function to delete a node from the linked list

void DeleteNode() {

char choice = 'Y';

while (choice == 'Y' || choice == 'y') {

int num; // Variable to store the number to be deleted

int isValid = 0; // Flag to check if the input is valid

while (!isValid) { // Inner loop for input validation

char input[10]; // Buffer to store the user's input

printf("\nEnter a number to delete: ");

scanf("%s", input); // Read the user's input

// Check if input is a valid integer (positive or negative)

isValid = 1; // Assume the input is valid

int startIdx = 0;

if (input[0] == '-') {

startIdx = 1; // Start checking digits from index 1 if there's a negative sign

}

for (int i = startIdx; i < strlen(input); i++) {

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

isValid = 0; // Set isValid to 0 if any character is not a digit

break;

}

}

if (isValid) {

num = atoi(input); // Convert input to an integer

} else {

printf("\nPlease enter a valid integer!\n");

}

}

struct node \*temp = head; // Start from the head node

struct node \*prev = NULL; // Pointer to keep track of the previous node

int found = 0; // Flag to indicate whether the number is found in the list

// Traverse the list to find the node to be deleted

while (temp != NULL) {

if (temp->x == num) {

found = 1; // Set found to 1 if the number is found

// If the node to be deleted is the head node

if (prev == NULL) {

head = temp->next; // Update head to the next node

free(temp); // Free the memory of the deleted node

temp = head; // Update temp to the new head

} else {

prev->next = temp->next; // Update the previous node's next pointer

free(temp); // Free the memory of the deleted node

temp = prev->next; // Update temp to the next node after deletion

}

} else {

prev = temp; // Update prev to the current node

temp = temp->next; // Move to the next node

}

}

if (!found) {

printf("Number %d is not in the list. Cannot delete.\n", num);

} else {

printf("Number %d deleted from the list.\n", num);

}

printf("\nDo you want to delete another number? (Y/N) : ");

scanf(" %c", &choice); // Read the user's choice

}

printf("\n(Press Any Key to Continue)\n");

printf("Exiting the Deletion Program...");

getch(); // Wait for user input before continuing

}

// Function to search for a node in the linked list and count duplicates

void SearchNode() {

char choice = 'Y';

while (choice == 'Y' || choice == 'y') {

int num; // Variable to store the number to be searched

int isValid = 0; // Flag to check if the input is valid

while (!isValid) { // Inner loop for input validation

char input[10]; // Buffer to store the user's input

printf("\nEnter a number to search: ");

scanf("%s", input); // Read the user's input

// Check if input is a valid integer (positive or negative)

isValid = 1; // Assume the input is valid

int startIdx = 0;

if (input[0] == '-') {

startIdx = 1; // Start checking digits from index 1 if there's a negative sign

}

for (int i = startIdx; i < strlen(input); i++) {

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

isValid = 0; // Set isValid to 0 if any character is not a digit

break;

}

}

if (isValid) {

num = atoi(input); // Convert input to an integer

} else {

printf("\nPlease enter a valid integer!\n");

}

}

struct node \*current\_node = head; // Start from the head node

int count = 0; // Counter to count the instances of the number

// Traverse the list to find the node and count instances

while (current\_node != NULL) {

if (current\_node->x == num) {

count++; // Increment the count if the number is found

}

current\_node = current\_node->next; // Move to the next node

}

if (count > 0) {

printf("%d is in the list %d time(s).\n", num, count);

} else {

printf("%d is not in the list.\n", num);

}

printf("\nDo you want to search another number? (Y/N) : ");

scanf(" %c", &choice); // Read the user's choice

}

printf("\n(Press Any Key to Continue)\n");

printf("Exiting the Search Program...");

getch(); // Wait for user input before continuing

}