

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Complex Computing Problem | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Hassaan Hameed (CT-25097)Haider (CT-25086)Samar Hasan (CT-25085) |  | | | |
|  | | | | —Programming Fundamentals—Sir Abdullah & Sir Furqan |  | | | |
|  | | |  | | |  | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  | | |  |
|  | INTRODUCTION | | | | | | |  |
|  |  | | |  |  | | |  |
|  |  | |  | | |  | |  |
|  |  |  | Our project is an Automated Teller Machine (ATM) which is used daily by many people to perform money related transactions. We chose this particular topic because it tests wide range of skills of the programmers as it uses: arrays, different library functions and also involves complex logic to make it work together. | | |  |  |  |
|  | | | | |
|  |  |  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | |  |  | |
|  | | THE PROCESS | | | | |  | |
|  | |  |  | | |  |  | |
|  | Salient Features of the ATM: **User Authentication:**  • Users will log in using their Account ID and PIN.  **Account Management:**  • Users can check their balance.  • Users can view transaction history (deposits and withdrawals).  **Withdrawal and Deposit:**  • Users can withdraw funds from their account if the balance is sufficient.  • Users can deposit money into their account, updating the balance immediately.  **Transaction Logging:**  • Every transaction (deposit or withdrawal) will be recorded in a log file using file  handling.  **Security:**  • PIN authentication ensures that only authorized users can access the system.  **Exit and Logoff:**  • Users will be able to log out after completing their transactions. | | | | | | |  |
|  | Technical Requirements | | |  |  | | |  |
|  | • **Language:** C  **• Platform:** Windows/Linux/Mac OS (console-based)  • **File Handling:** For storing transaction logs.  • **Header File:** stdio.h for standard input and output operations and other library functions for different purposes. | | |  |  | | |  |

|  |
| --- |
|  |

##### 1. Headers and macros:

 stdio.h — input/output (printf, scanf, FILE I/O).

 stdlib.h — general utilities (not strictly needed in your code as written, but often included for exit, malloc, etc.).

 string.h — string operations (not used heavily here but commonly included).

 FILE\_NAME — filename used for reading/writing account data.

 MAX\_ACCOUNTS — maximum accounts the program will load/store (array sizing and safety).

# Explanation of different sections of the code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define FILE\_NAME "text.txt"

#define MAX\_ACCOUNTS 10

**2. Data structures:**

typedef struct {

int pin;

* Account stores a single account: integer pin, name (up to 49 chars + null), and balance.
* AccountList is a container for up to MAX\_ACCOUNTS accounts and tracks how many are currently loaded (count).

char name[50];

float balance;

} Account;

typedef struct {

Account accounts[MAX\_ACCOUNTS];

int count;

} AccountList;

**3. load\_all\_account\_data():**

int load\_all\_account\_data(AccountList \*list) {

FILE \*file = fopen(FILE\_NAME, "r");

if (file == NULL) {

printf("Account file not found (%s). Please create a 'text.txt' file with account data.\n", FILE\_NAME);

return 0;

}list->count = 0;

while (list->count < MAX\_ACCOUNTS) {

int result = fscanf(file, "%d,%49[^,],%f\n",

&list->accounts[list->count].pin,

list->accounts[list->count].name,

&list->accounts[list->count].balance

);

if (result == 3) {

list->count++;

} else if (result == EOF) {

break;

} else {

printf("Error reading account data near line %d. Please check the format.\n", list->count + 1);

fclose(file);

return 0;

}

}

fclose(file);

printf("Successfully loaded %d account(s).\n", list->count);

return 1;}

Opens text.txt for reading. If missing, prints an error and returns failure (0).

Initializes list->count to 0.

Repeatedly uses fscanf to parse lines of the form: PIN,Name,Balance (comma-separated). Format explanation:

* %d reads integer PIN.
* %49[^,] reads up to 49 chars for Name stopping at the next comma (prevents overflow).
* %f reads the float Balance.
* The \n in format helps absorb the newline but is not strictly required; fscanf ignores whitespace for numeric conversions.

 If fscanf reads 3 items, it increments count.

If fscanf returns EOF, we reached end of file and stop.

If any parsing error occurs (unexpected format), the function prints an error and returns failure.

On success, it closes the file and returns 1.

**4. save\_all\_account\_data():**

Opens text.txt for writing (mode "w"). This overwrites the entire file with current account data.

Iterates all loaded accounts and writes each as PIN, Name, Balance with balance formatted to two decimals.

Closes the file.

Note: Because it overwrites the whole file, any earlier data not present in list will be lost — but this is correct if AccountList is the single source of truth.

void save\_all\_account\_data(const AccountList \*list) {

FILE \*file = fopen(FILE\_NAME, "w");

if (file == NULL) {

printf("Critical Error: Cannot save account data.\n");

return;

}

for (int i = 0; i < list->count; i++) {

fprintf(file, "%d,%s,%.2f\n",

list->accounts[i].pin,

list->accounts[i].name,

list->accounts[i].balance

);

}

fclose(file);

}

# 5. atm\_machine()

This is the main interactive ATM logic. I’ll break it into subparts.

* Declares AccountList and other local variables.
* Loads accounts from disk. If loading fails, function returns early.

##### 5.1 Setup and loading accounts:

AccountList account\_list;

int entered\_pin;

int logged\_in\_index = -1;

if (!load\_all\_account\_data(&account\_list)) {

return;

}

**5.2 Authentication**

 Prompts user for PIN.

 There is a while (getchar() != '\n'); line intended to clear the input buffer. (If the buffer is already empty this will wait for a newline; in typical usage after earlier input it might be OK, but it’s a small fragility .

 Reads PIN via scanf("%d", &entered\_pin). If input isn’t an integer, prints error and exits.

 Searches loaded accounts for a matching pin. If found, sets logged\_in\_index.

 If no match, prints error and exits.

 If found, creates an Account \*active\_account reference and welcomes user by name.

printf("\n--- sharcodec ATM Login ---\n");

printf("Enter your PIN: ");

while (getchar() != '\n');

if (scanf("%d", &entered\_pin) != 1) {

printf("Invalid PIN format. Exiting...\n");

while (getchar() != '\n');

return;

}

for (int i = 0; i < account\_list.count; i++) {

if (entered\_pin == account\_list.accounts[i].pin) {

logged\_in\_index = i;

break;

}

}

if (logged\_in\_index == -1) {

printf("Incorrect PIN or account not found. Exiting...\n");

return;

}

Account \*active\_account = &account\_list.accounts[logged\_in\_index];

printf("\nAccess Granted. Welcome %s!\n", active\_account->name);

**5.3 ATM menu loop:**

int choice;

float amount;

while (1) {

printf("\n--- Sharcodec ATM Menu for %s ---\n", active\_account->name);

printf("1. Withdraw\n2. Deposit\n3. Check Balance\n4. Exit\n");

printf("Please choose an option (1-4): ");

if (scanf("%d", &choice) != 1) {

printf("Invalid input. Please enter a number.\n");

while (getchar() != '\n');

continue;

}

if (choice == 1) { // Withdraw

save\_all\_account\_data(&account\_list);

} else if (choice == 2) { // Deposit

save\_all\_account\_data(&account\_list);

} else if (choice == 3) { // Check Balance

printf("Your current balance is: $%.2f\n", active\_account->balance);

} else if (choice == 4) { // Exit

printf("Thank you, %s. Goodbye!\n", active\_account->name);

break;

} else {

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

}

}

* Displays menu and reads user option.
* If scanf for the option fails, it clears input and repeats the menu.
* For Withdraw:
* Reads amount (float). Validates amount > 0.
* Checks sufficient balance. If OK, subtracts amount from active\_account->balance, prints success, and calls save\_all\_account\_data() to persist change.
* For Deposit:
* Reads amount, validates > 0, adds to balance, prints success, and saves file.
* For Check Balance:
* Prints current balance (from active\_account).
* Option 4 breaks the loop and exits ATM.
* Menu loop repeats until user chooses Exit.

**6.Main():**

* Simple entry point: calls the ATM function and returns.

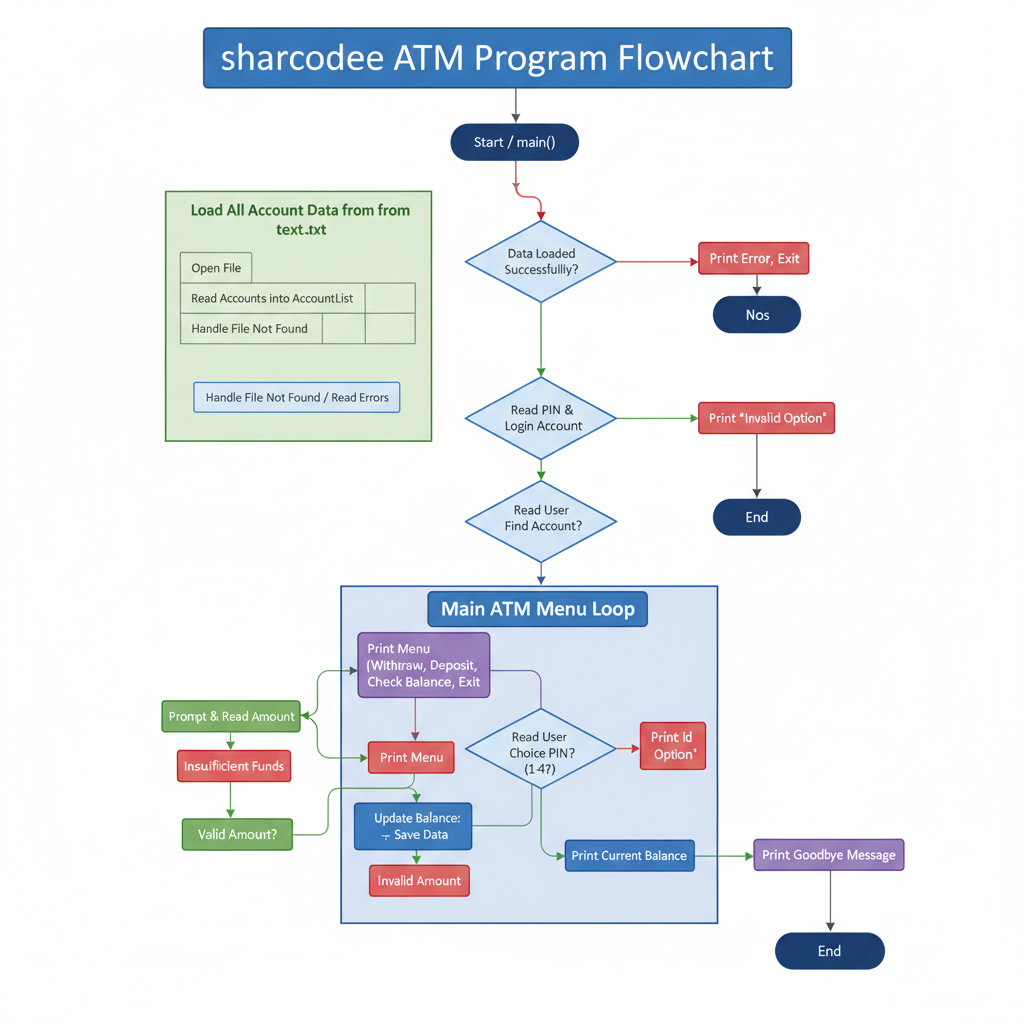
int main() {

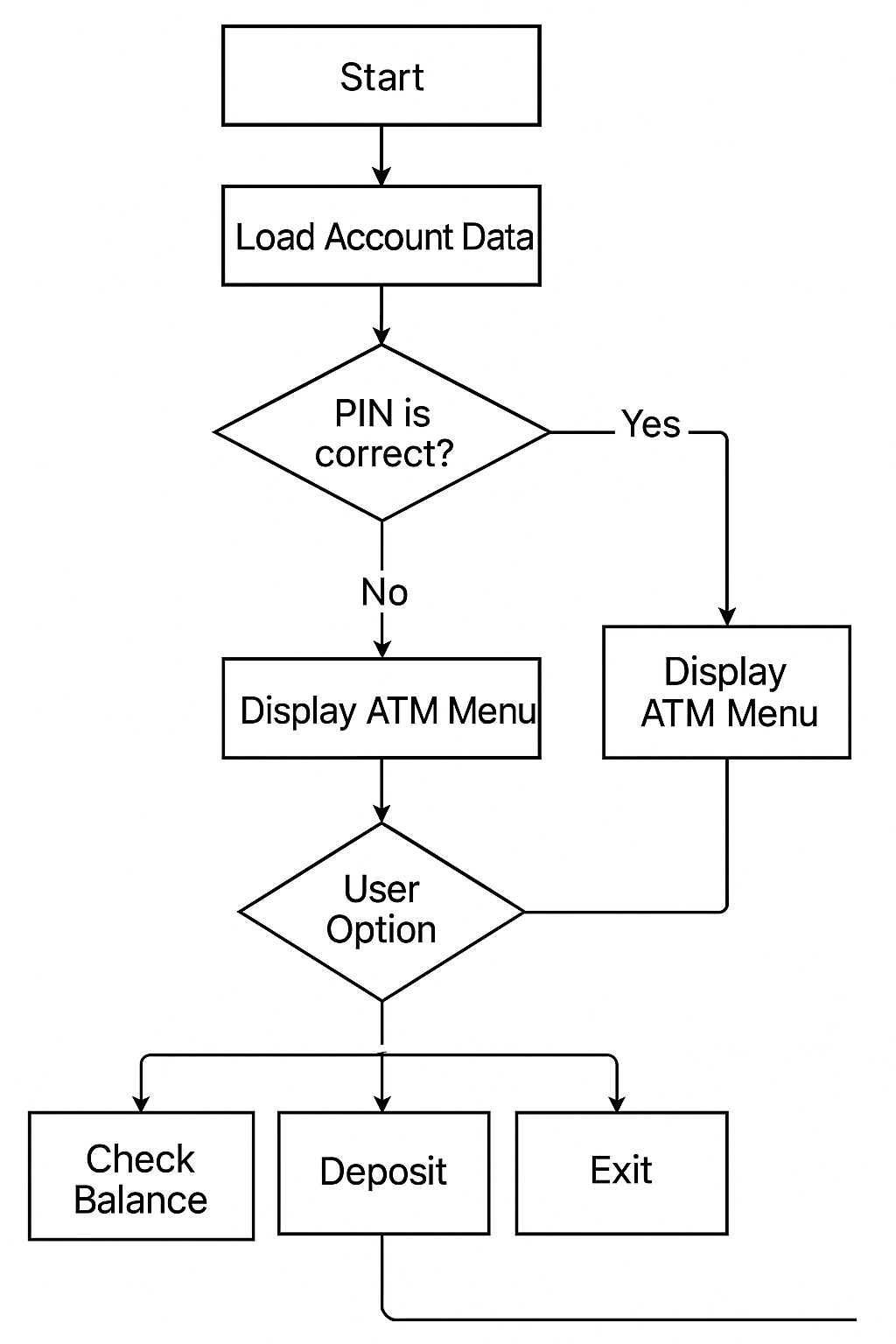
atm\_machine();

return 0;

}

# A Visual Flowchart To Show How The Program Works



A Simple Flowchart For Better Understanding