Metro Rail Ticketing System

By,

Team Pixelites OPS

CSE103-Fall 2023  
Section: 12

Faculty: Dr. Hasan Mahmood Aminul Islam

Wednesday, 27 December 2023

Table of contents:

Introduction \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Description\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Methodology \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Results and Discussion \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conclusion \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### Introduction:

*Project Title:* Metro Rail Ticketing System.

*Group Members:*

Shahriar Mahabub Oishik(**Team Lead** **&** **Backend Developer**)

ID: 2023-3-60-073

Pulok Akibuzzaman(**Frontend Developer**)

ID: 2023-3-60-051

Md. Ashik-Uz-Zaman(**Tester & Bug Hunter**)

ID: 2023-3-60-074

**Abstract:**

The Metro Rail Ticketing System ensures a dynamic and very user-friendly experience. You can buy one-time tickets, cancel your tickets, can join our membership program and buy pre-tickets, can buy future traveling tickets, check routes and timetables.

#### Problem Statement:

The Metro Rail Ticketing System is a comprehensive C language project designed to automate and streamline the ticketing process for metro rail services. The system aims to provide a user-friendly interface for both commuters and administrators, ensuring efficient ticket purchase, validation, and management.

**Motivation:**   
 The importance of this project lies in enhancing the overall efficiency and user experience of metro rail services. Manual ticketing processes are prone to errors, delays, and inefficiencies, leading to inconvenience for commuters and challenges for administrators. By automating these processes through a user-friendly interface, the Metro Rail Ticketing System aims to improve the accuracy and speed of ticket transactions. This, in turn, contributes to a more convenient and reliable public transportation system, benefiting both commuters and administrators alike. The motivation behind this project is to create a technologically advanced solution that addresses the shortcomings of traditional ticketing systems, ultimately making metro rail travel more accessible and user-friendly.

**Features:**

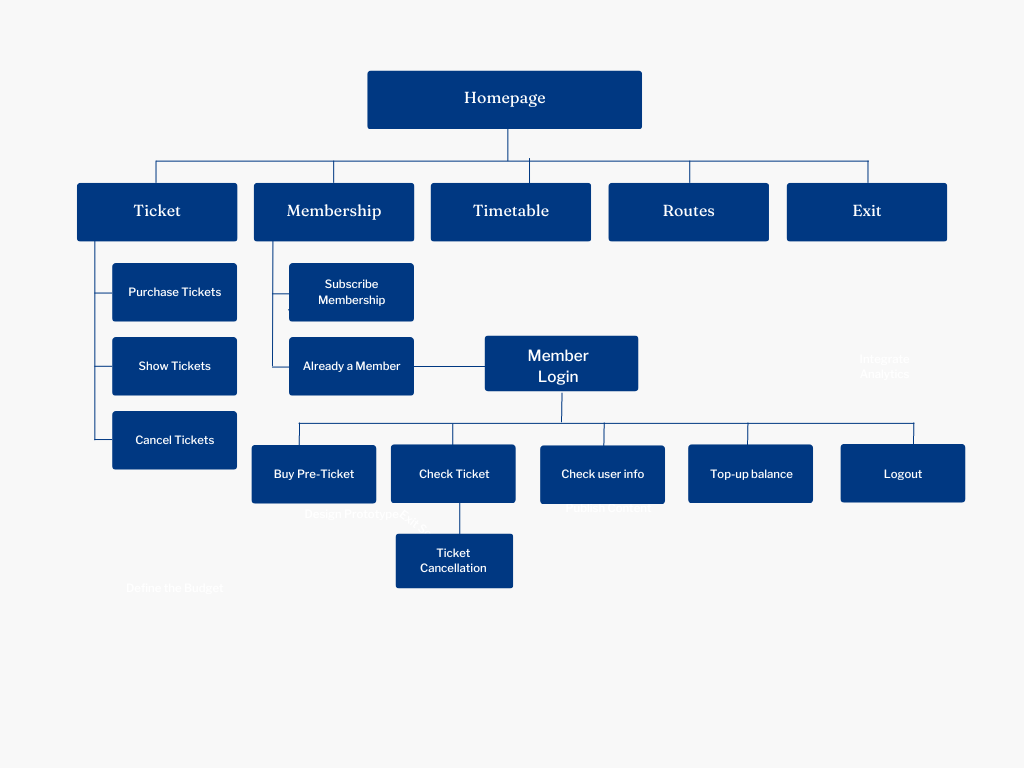
* **Welcome Splash Screen:**
  + Greet users with a friendly welcome as soon as they start the system.
* **Purchase Tickets:**
  + Can easily buy metro tickets with a straightforward purchase process.
* **Show Tickets:**
  + View your purchased tickets conveniently within the system.
* **Cancel Tickets:**
  + Cancel tickets hassle-free if plans change.
* **Subscribe Membership:**
  + Easily subscribe to a membership plan for exclusive perks.
* **Login for Members:**
  + Members can log in to access personalized features and information.
* **Timetable:**
  + Check the metro timetable for convenient planning.
* **Routes:**
  + Explore the available metro routes for easy navigation.
* **Exit Function:**
  + Close the system effortlessly with a dedicated exit function.
* **Thank You Splash Screen:**
  + End the user experience on a positive note with a thank-you message.

**Methodology:**

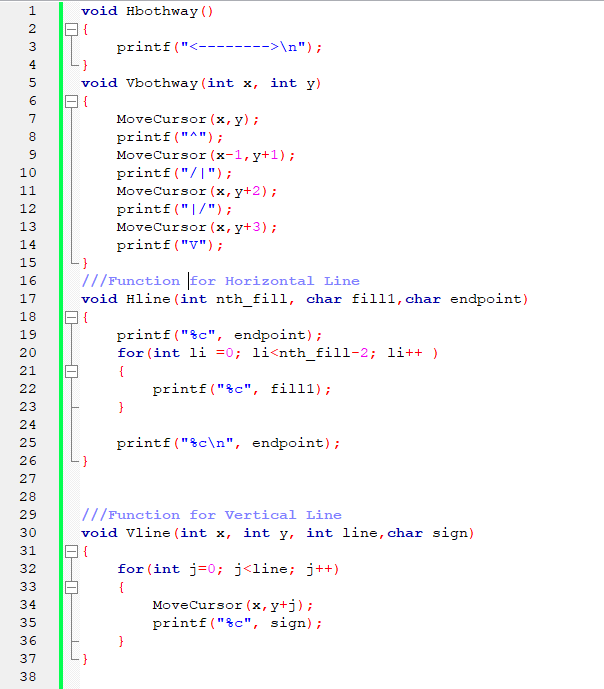
We are using the C17 version of C.  
We are using the following headers:

* stdio.h
* stdlib.h
* windows.h
* time.h
* And 5 user-defined header files.

**Flow Chart:**



**Block Header File Explanation:**

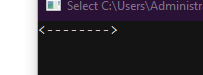


**From 1-3 Line:**

**Hbothway()**

***Functionality:*** The function of **Hbothway()** prints an arrow directing both ways horizontally and moves to the next line.

***for instance:*** It will print this arrow from wherever the cursor is-



**From 4-10 line:**

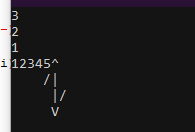
**Vbothway(int x, int y)**

***Input:*** The function of **Vbothway(int x, int y)** takes two integers **‘x’** and **‘y’** as arguments.

***Functionality:*** Uses the integer **‘x’** as starting point from (x+1)-th character for printing in specific line and the integer **‘y’** as (y+1)-th line of command prompt and prints an arrow directing both ways vertically.

**for instance**: This code will print that arrow.(**numbers in CMD is used for demonstration not part of the function**)



****

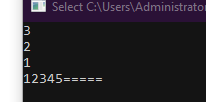
**From 17-26 Line:**

**Hline(int nth\_fill,char fill1, char endpoint)**

***Input:*** The function of **Hline(int nth\_fill,char fill1, char endpoint)** takes integer **‘nth\_fill’**,**’ fill1**’ and a character **’endpoint’** as argument .

***Functionality:*** Prints (**‘nth\_file’-2)** times of **‘char fill1’** character contiguously from wherever the cursor is.

**for instance**: This code will print that horizontal line of “=”.(**numbers in CMD is used for demonstration not part of the function**) 

****

**From 30-37 Line:**

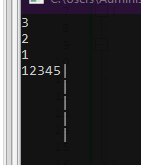
**Vline(int x,int y, int line,char sign)**

***Input:*** The function of **Vline(int x,int y, int line,char sign)** takes integers **‘x’,’y’,’line’** and a character ’ **sign**’ as arguments .

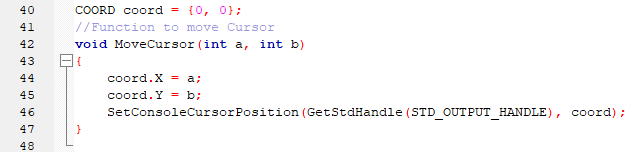
***Functionality:*** Prints ‘**sign’** character for ‘**line’** times vertically starting from (y+1)th line and (x+1)th character of that line contiguously.

**for instance**: This code will print that vertical line.(**numbers in CMD is used for demonstration not part of the function**)

****

****

**MoveCursor(int a,int b)**

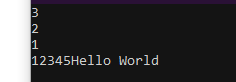
****

***Input:*** The MoveCursor function takes two integers, a and b, as input arguments.

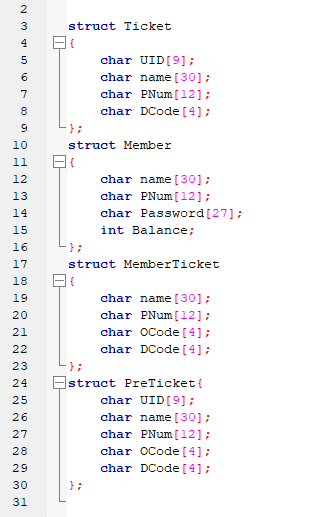
***Functionality:*** ‘**coord.X = a’** Updates the X coordinate of the coord structure with the value of a. ‘**coord.Y = b’** Updates the Y coordinate of the coord structure with the value of b. ‘**SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord)**’ Uses the Windows Console API functions to set the console cursor position to the coordinates specified in the coord structure and ‘**SetConsoleCursorPosition’** function takes two parameters. ‘**GetStdHandle(STD\_OUTPUT\_HANDLE)’** Retrieves the standard output handle for the console. ‘**coord’** The new coordinates for the console cursor.

**for instance**: This code will print “Hello World” from 4th line and 6th character of 4th line (**numbers in CMD is used for demonstration not part of the function**)

****

****

**DataTypesHeader File Explanation:**

****

In this header file there are 4 User-defined DataTypes.

**For brief explanation:  
Ticket:**

* + **UID[9]**: Represents a unique identifier with a length of 9 characters.
  + **name[30]**: Represents the name with a maximum length of 30 characters.
  + **PNum[12]**: Represents a phone number with a maximum length of 12 characters.
  + **DCode[4]**: Represents a destination code with a length of 4 characters.

**Member:**

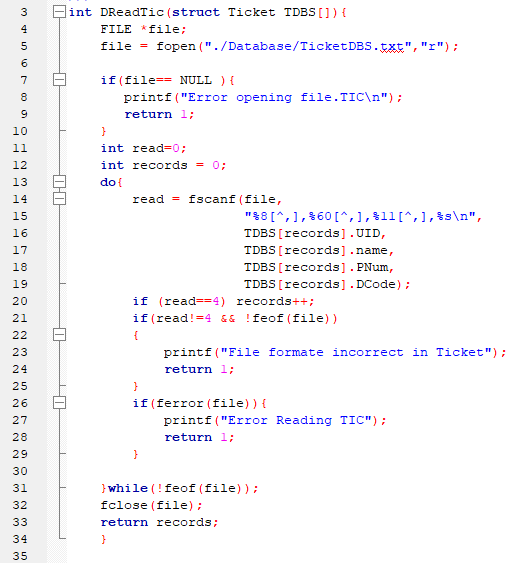
* + **name[30]**: Represents the name of a member with a maximum length of 30 characters.
  + **PNum[12]**: Represents the phone number of a member with a maximum length of 12 characters.
  + **Password[27]**: Represents the password of a member with a length of 27 characters.
  + **Balance**: Represents the balance of a member, assuming it's an integer.

**MemberTicket:**

* + **name[30]**: Represents the name associated with a member's ticket with a maximum length of 30 characters.
  + **PNum[12]**: Represents the phone number associated with a member's ticket with a maximum length of 12 characters.
  + **OCode[4]**: Represents an origin code with a length of 4 characters.
  + **DCode[4]**: Represents a destination code with a length of 4 characters.

**PreTicket:**

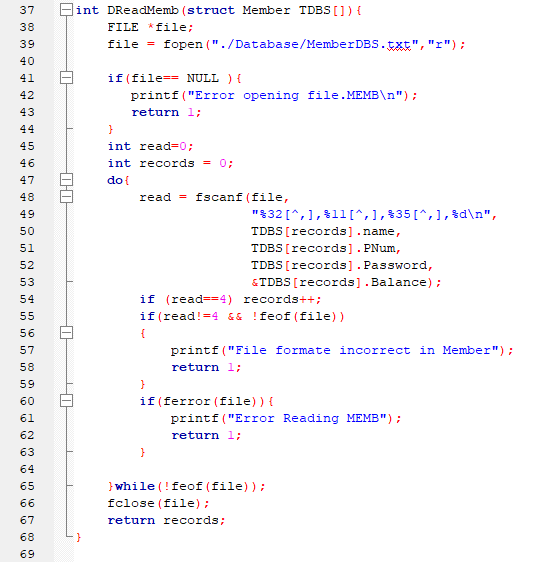
* + **UID[9]**: Represents a unique identifier with a length of 9 characters.
  + **name[30]**: Represents the name associated with a pre-ticket with a maximum length of 30 characters.
  + **PNum[12]**: Represents the phone number associated with a pre-ticket with a maximum length of 12 characters.
  + **OCode[4]:** Represents an origin code with a length of 4 characters.
  + **DCode[4]:** Represents a destination code with a length of 4 characters.

**RnW Header File Explanation:** 

**DReadTic(struct Ticket TDBS[]);**

***Input:*** The `DReadTic` function takes struct ticket datatype array as an argument

***Functionality:*** this function opens a file named "TicketDBS.txt," reads comma-separated records with four fields, populates a `Ticket` structure array, and returns the number of records read. It performs error checks for file opening, data reading, and file format issues.

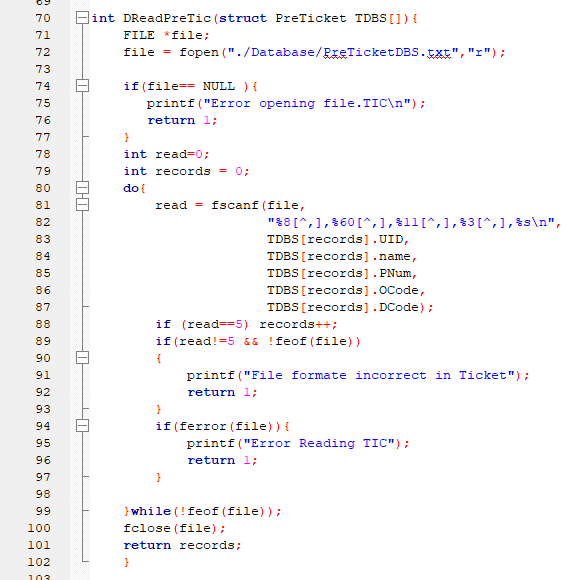


**DReadMemb(struct Member TDBS[]);**

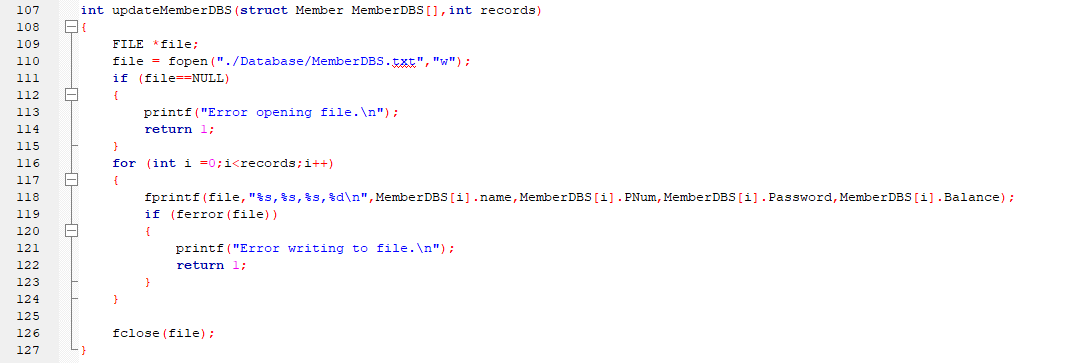
***Input:*** DReadMemb function takes an array of struct Member Datatype as an argument

***Functionality:*** The DReadMemb function opens and reads data from a file named "MemberDBS.txt" into an array of structures representing members. It checks for errors in file opening, data reading, and file format.

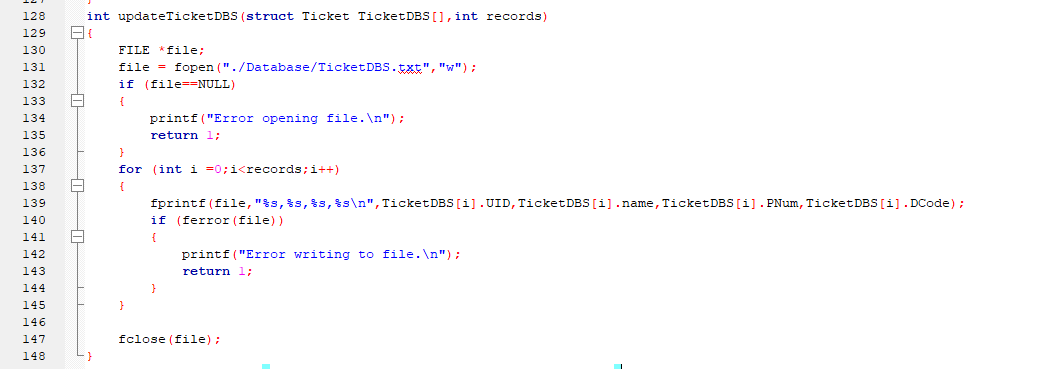
***Return:*** The function returns the number of records read.

****

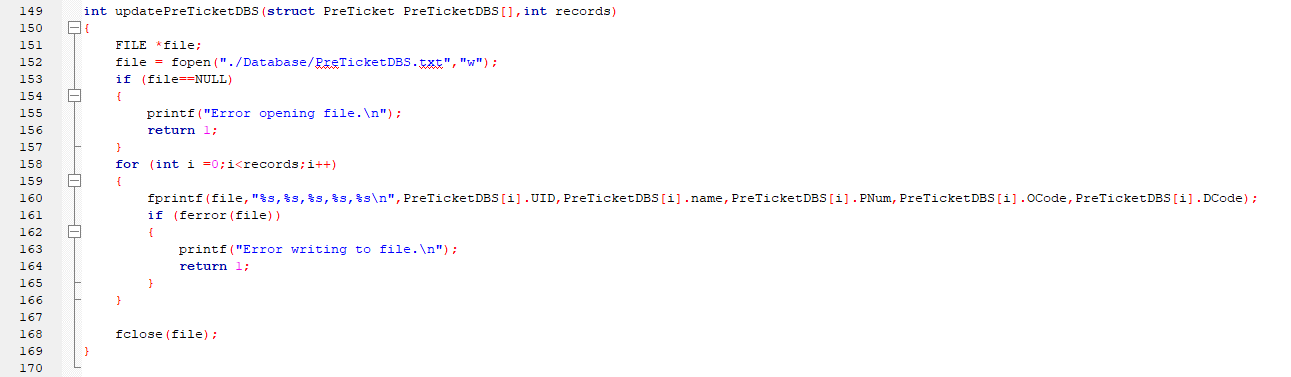
**The `DReadPreTic` function opens and reads data from a file named "PreTicketDBS.txt" into an array of structures representing pre-tickets. It checks for errors in file opening, data reading, and file format, then returns the number of records read.**

****

**Reads member data from struct array, writes it to file line-by-line with commas separating values, checks for errors and returns 1 if there’s any.**

****

**Saves ticket data (UID, name, phone, code) from struct array to "TicketDBS.txt" comma-separated, exits on errors.**

****

**This code loops through ticket data in TicketDBS, writes each ticket's UID, name, phone, and code to "TicketDBS.txt" comma-separated, and checks for errors, stopping on any and returning 1.**

**Main Function (main.c)**

**Overall Structure:**

* **The code manages a train ticketing system.**
* **It handles tasks like purchasing, viewing, and canceling tickets.**
* **It uses text-based menus for user interaction.**
* **It stores ticket data in text files.**

**Key Functionalities:**

**Splash Screen:** Displays an initial loading screen.

**Home Page: Offers options for:**

* Purchasing Tickets.
* Viewing Tickets.
* Canceling Tickets.
* Managing Members.
* Exit.

**Ticket Page: Handles ticket-related actions:**

* Purchasing Tickets.
* Viewing Tickets.

**Canceling Tickets.**

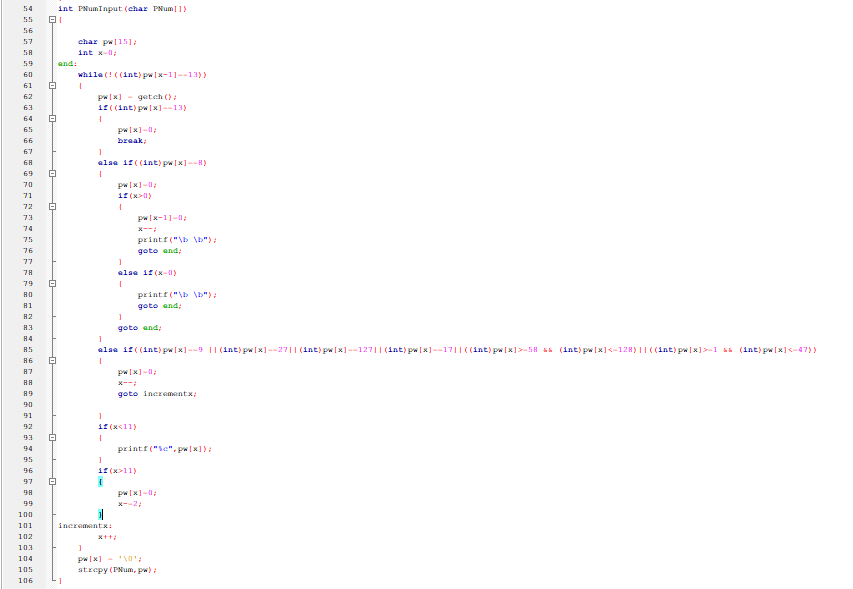
**Key Points:**

* User Input: Relies on keyboard input for user choices and data entry.
* Database Files: Uses plain text files to store ticket and member data.
* Data Structures: Employs arrays of structures to manage ticket and member information.
* Error Handling: Includes some basic error handling, such as input validation.
* Modular Design: Divides code into functions for better organization and readability.

**Special Function:**

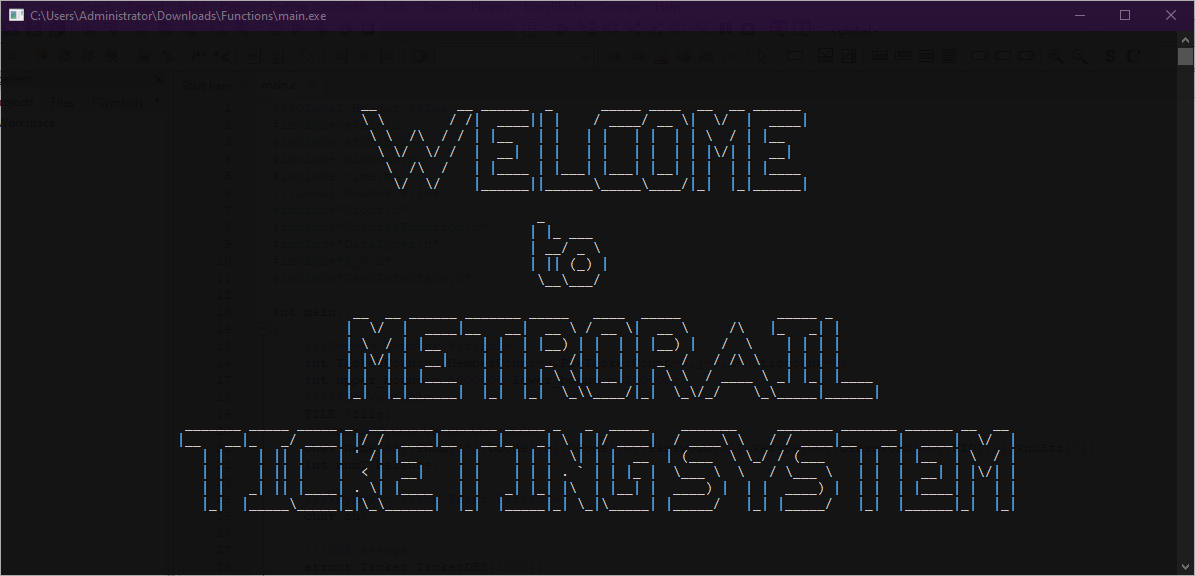
****

**Code reads password secretly (no input shown). Uses asterisks, backspace, ignores special keys, limits length to 29. Stores password in another array.**

****

**Read the phone number. ESC works. Only digits allowed. Max length 11. Stores number in another array.**

**User Interface:**

****

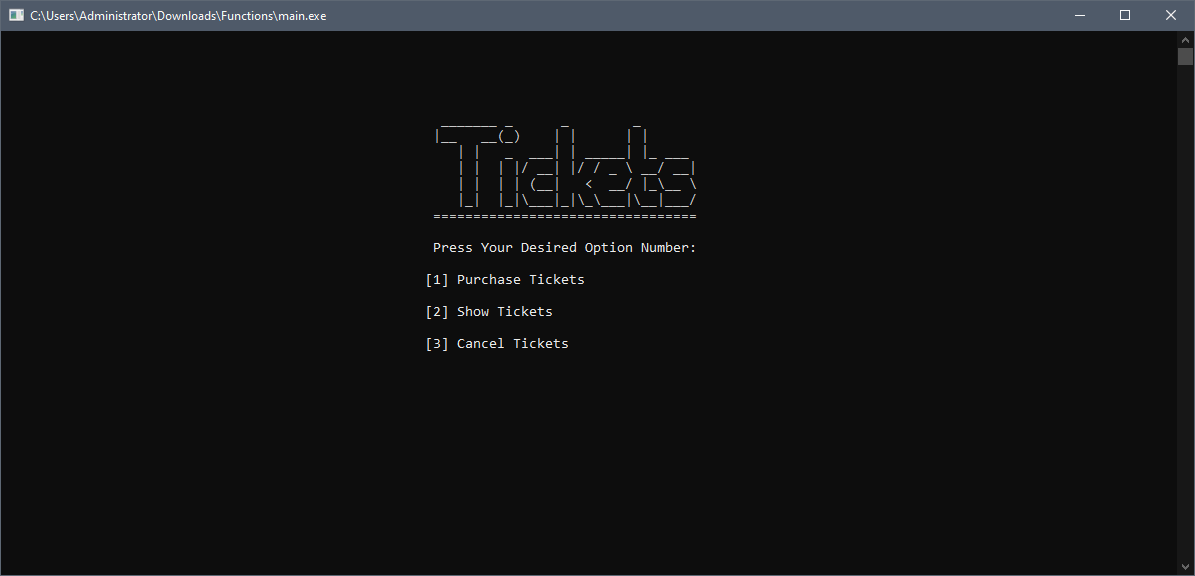
****

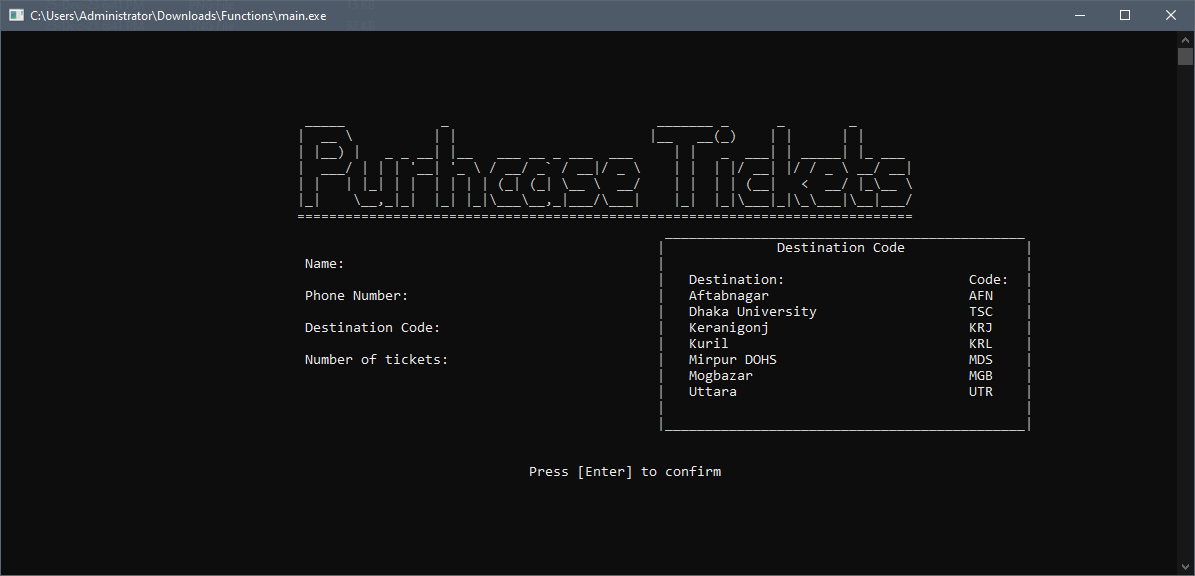
This code defines a function called HomePage that displays the main menu of a bus reservation system.The function first clears the screen and then prints a bus logo made of ASCII characters. It then prints a horizontal line and a prompt for the user to enter their desired option number.The function then prints the following menu options:

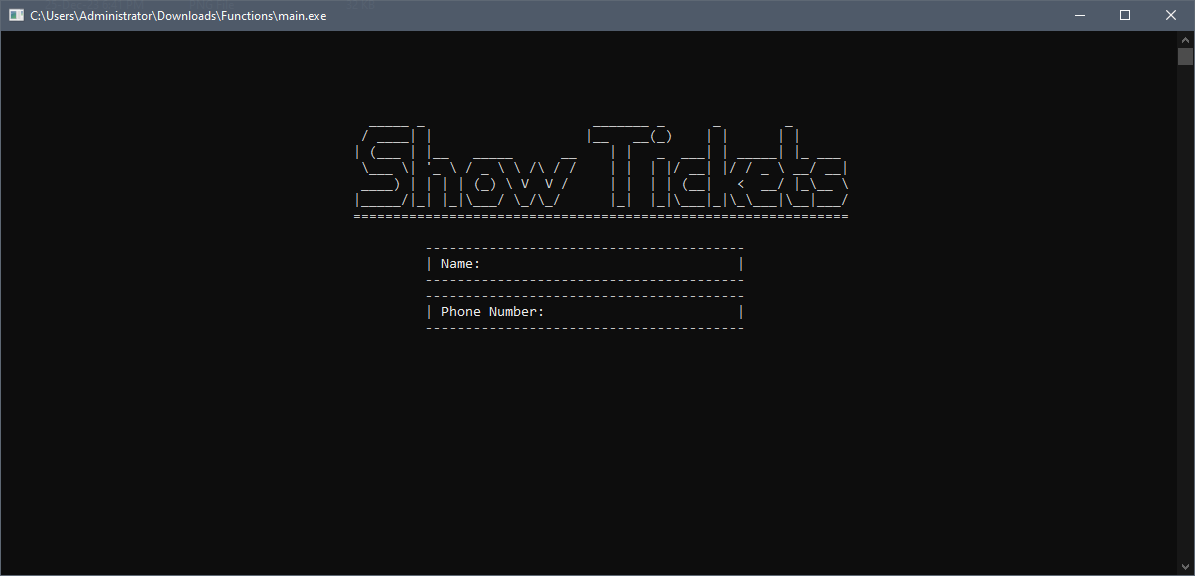
[1] Tickets [2]Membership [3] Timetable [4] Routes

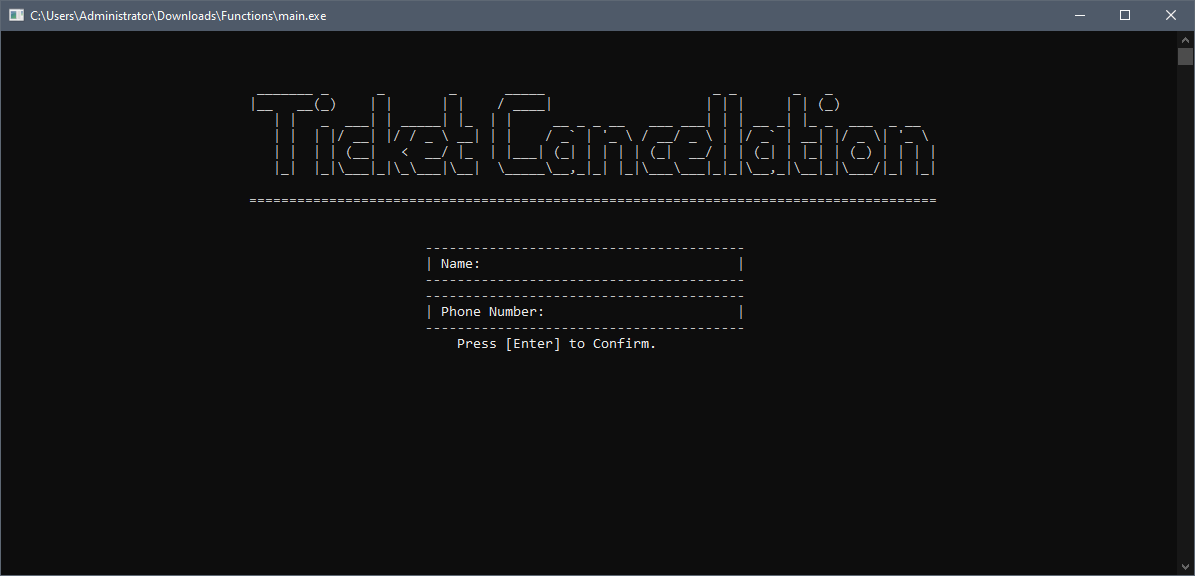
Finally, the function prints an exit option and moves the cursor to the beginning of the next line.

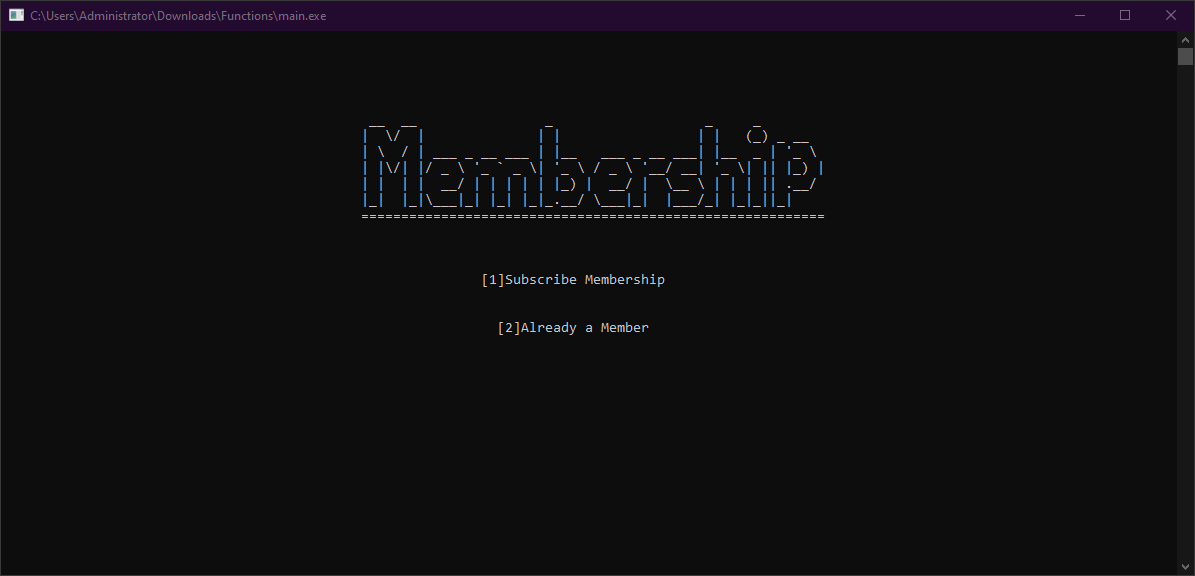
**Here’s some snapshots of our user-interface:**

****

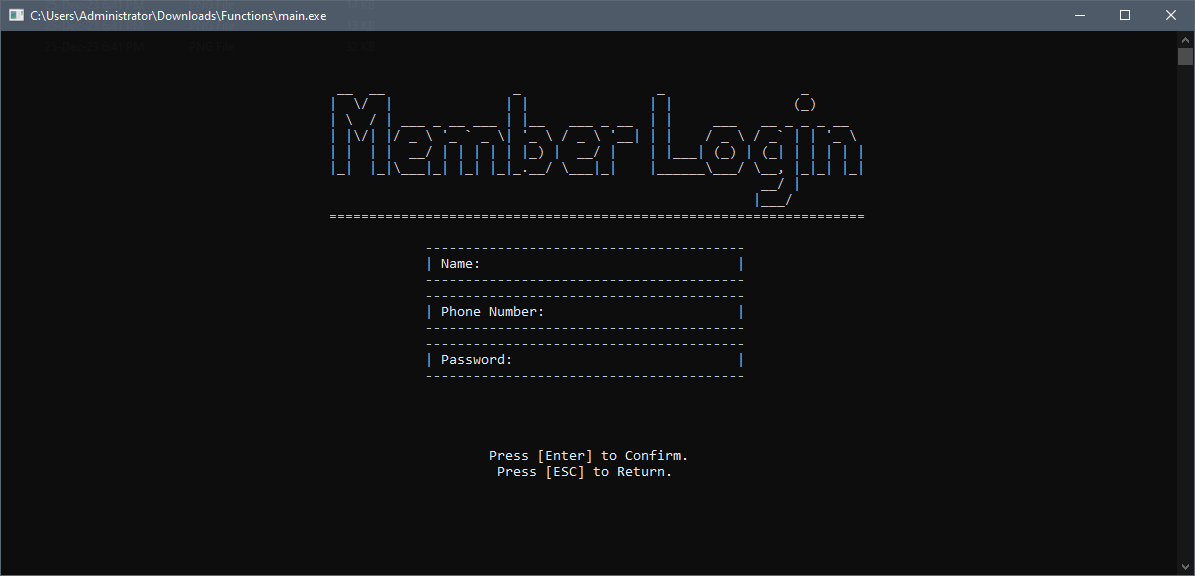
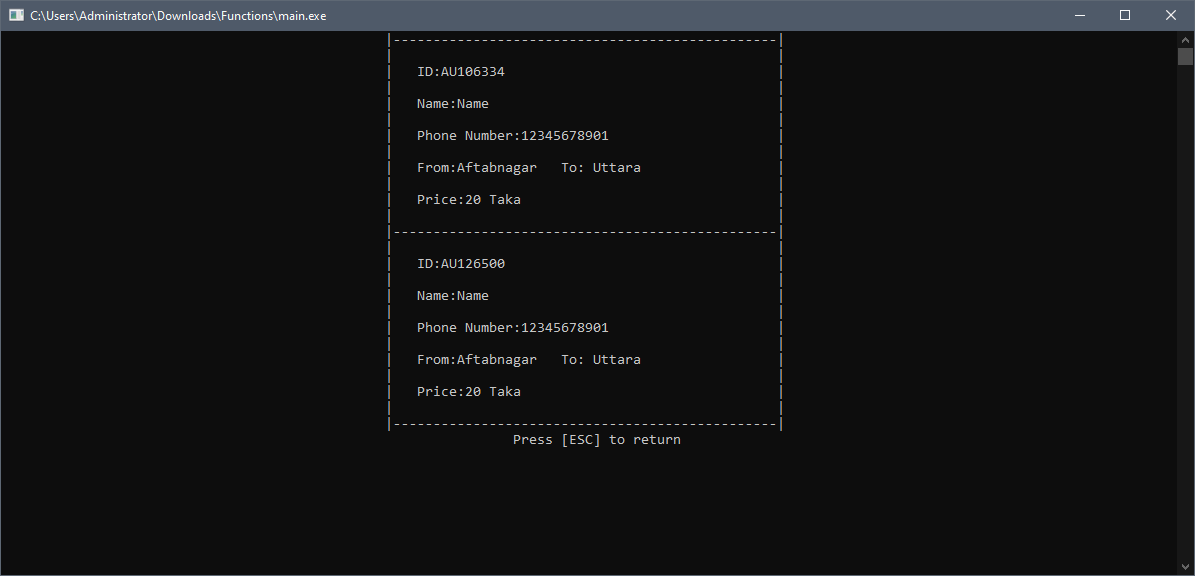
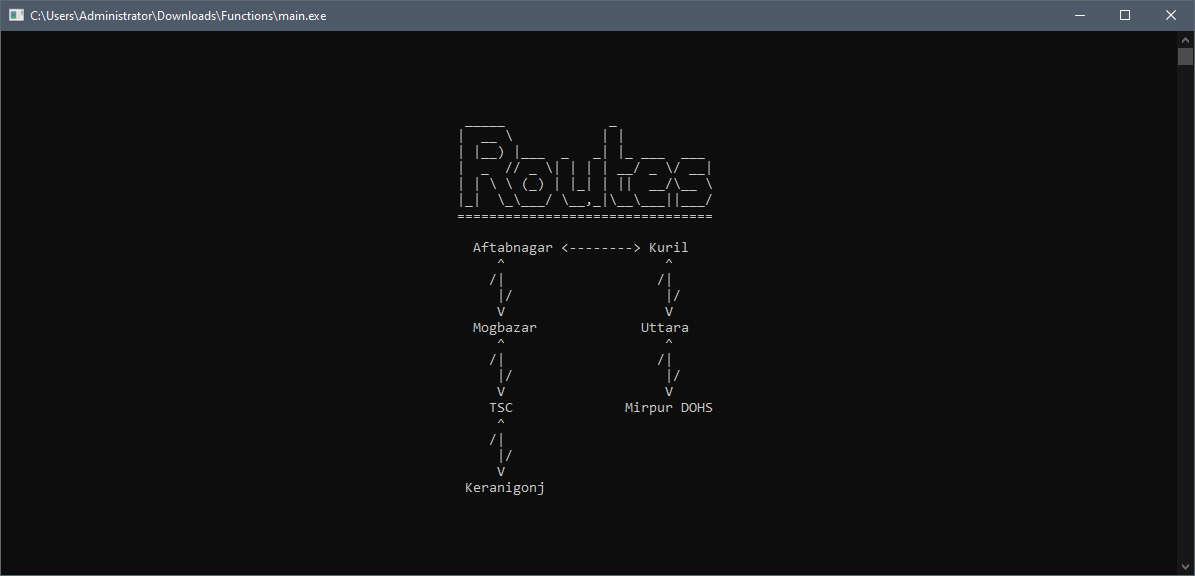
****

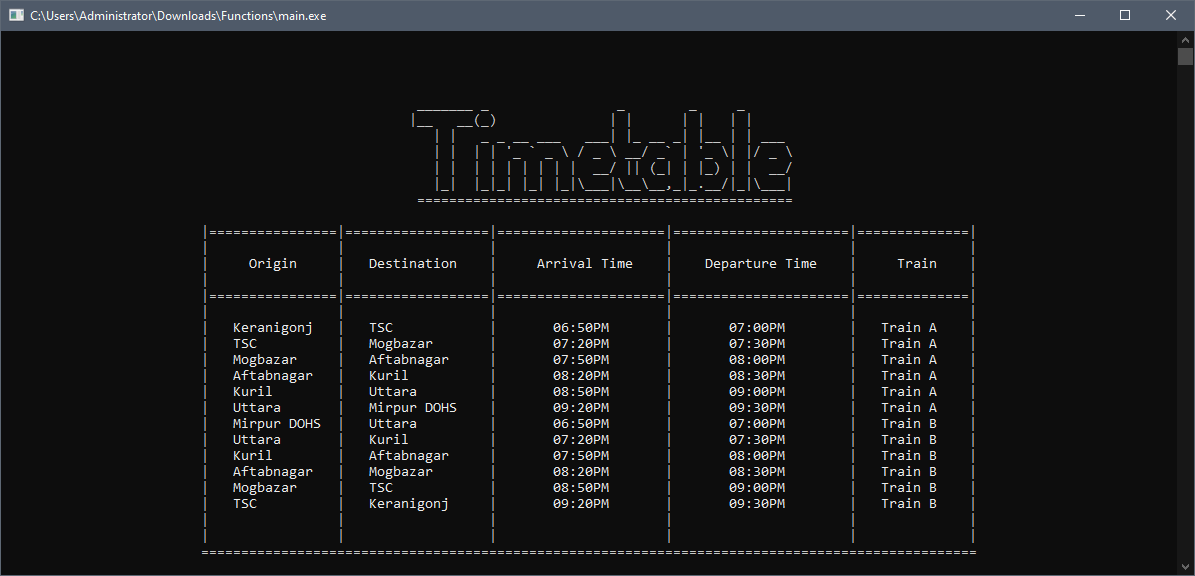
****

****

****

****

****

****

**Here's a breakdown of the TimeTable function:**

**Functionality:**It displays a train timetable in a visually formatted table, with information about origin, destination, arrival time, departure time, and train name. It dynamically adjusts the displayed timetable based on the current time of day.

**Key Steps:**

**Gets current time:**Uses time(NULL) to get the current time as a time\_t value.

Uses localtime to convert it to a tm struct for easier time component access.

**Clears the screen:**

Uses system("cls") to clear the console screen for a fresh display.

**Prints heading and table structure:**

Prints a large, centered header "Train Schedule" using multiple printf calls and cursor positioning.

Draws horizontal and vertical lines using Hline and Vline functions (not shown) to create a table structure.

Labels the columns as "Origin," "Destination," "Arrival Time," "Departure Time," and "Train."

**Fills in table content:**

Prints station names under "Origin" and "Destination" columns.

Uses conditional statements based on the current time to print appropriate arrival and departure times for different time slots (e.g., 5 AM-9 AM, 9 AM-1 PM, etc.).

Prints "Train A" or "Train B" under the "Train" column.

**Moves cursor to the end:**

Positions the cursor at the bottom of the table for a cleaner display.

**Important Notes:**

The code assumes the existence of MoveCursor, Hline, and Vline functions for screen formatting but doesn't provide their definitions. The timetable data (station names, times, train names) is hardcoded within the function. It doesn't handle user input or any actions beyond displaying the timetable.

**Result and Discussion:**

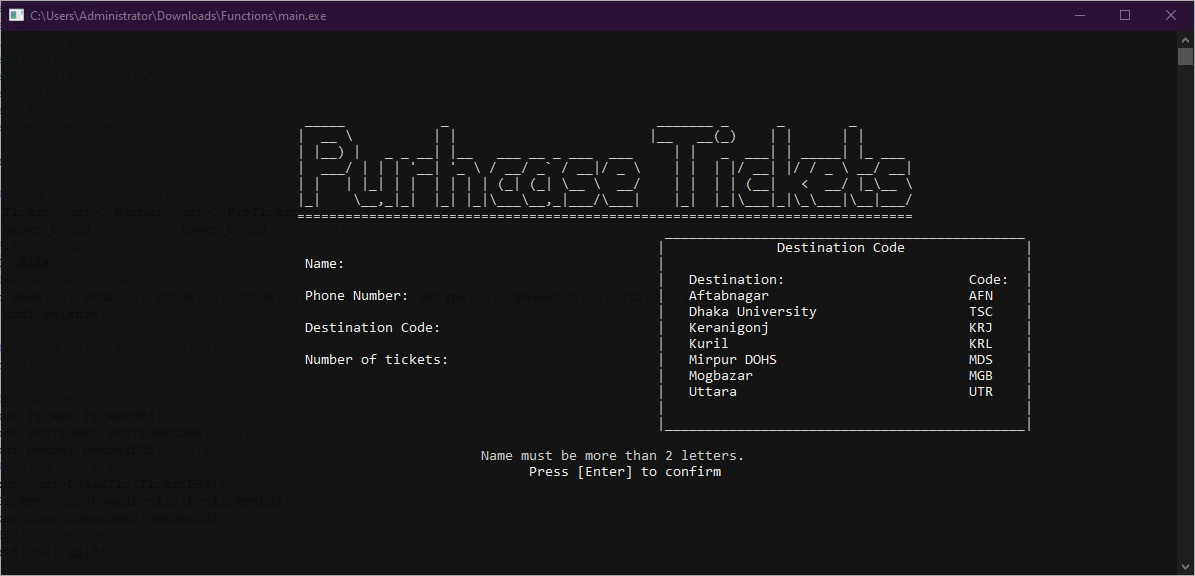
Testing and Validation: We tested possible interactions/Input from users and restricted the input for users, so that program doesn’t take any input that crashes the program.

***Fixed the problems through testing:***

**Purchase Tickets:**

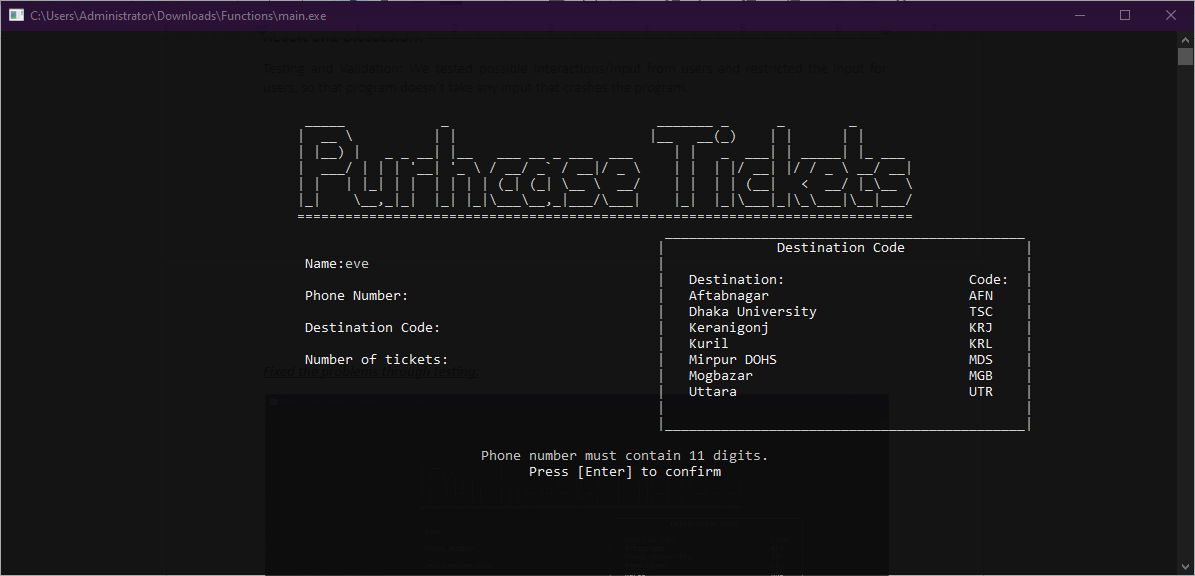
**Name Limit:**

To avoid suspicious name input from users, we limited the user from using anything other than alphabets by using **NameInput();** function. and we restricted the user to use name with minimum 3 characters or else it will show an error message and will take

******

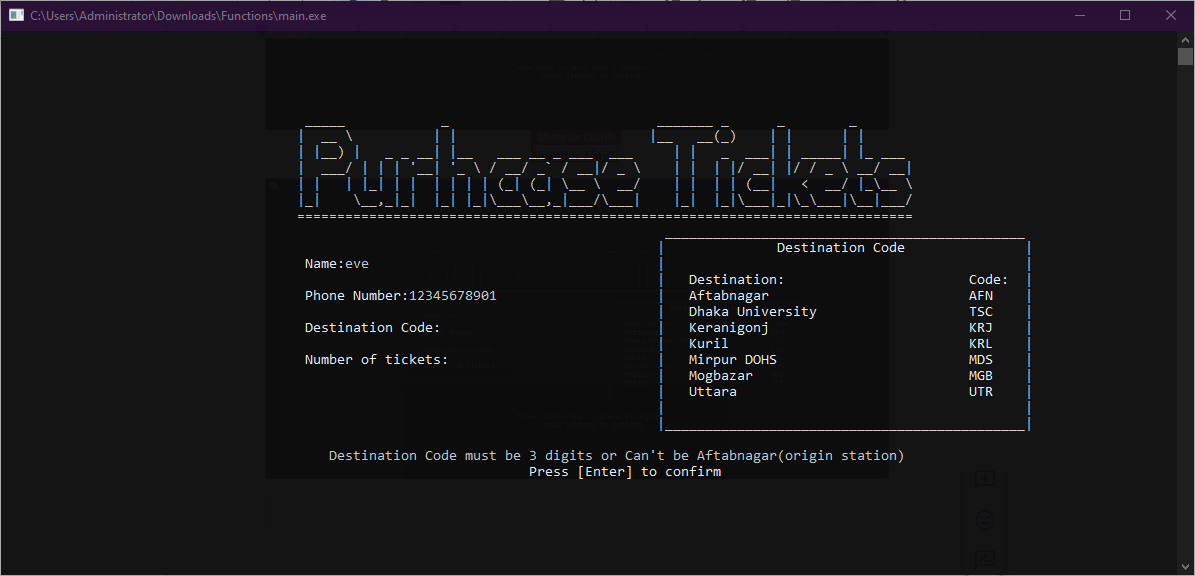
**Number Limit:**

To avoid extra or less than 11 digit number input from user,we limited the user to use only 11 digits and number by using **PNumInput();**

****

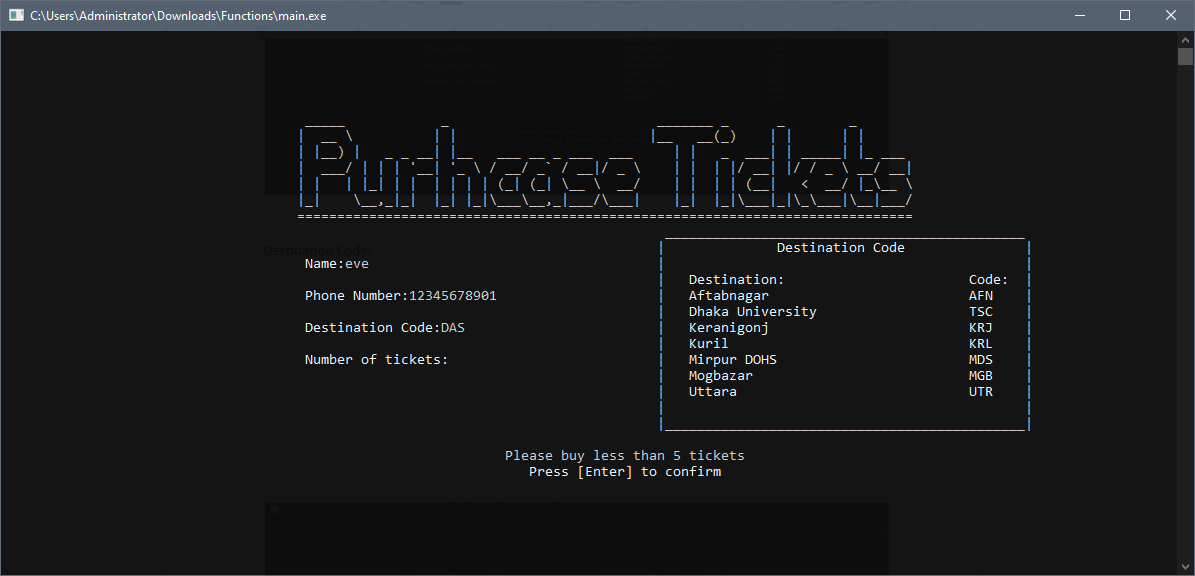
**Destination Code:**

To avoid extra or less than 3 digit Code input from user,we limited the user to use only 3 digits and Alphabets by using **CodeInput();**

****

**Ticket Limitation:**

We restricted user to buy ticket not more than 4 by using a basic while loop.

****

**System Specifications:**

* Operating System: Windows 10 (For Best Optimized Result)
* Code::Blocks Version 20.03-r11983
* CMD prompt launch size: 147×34 in character.

**Future Scope:**

* Color Integration.
* Sound Integration.
* Online ticketing system.
* Type only access.
* No refund policy.

**Conclusion:**

The Metro Rail Ticketing System is designed to revolutionize how we navigate urban transit. With dynamic features, vibrant visuals, and user-friendly options, we aim to simplify ticketing for everyone. The system's adaptability, accessibility, and continuous improvement underscore our commitment to making metro travel efficient, enjoyable, and tailored to user needs. As we move forward, user feedback will be crucial in refining and expanding the system, ensuring it remains a seamless and connected part of modern commuting. The Metro Rail Ticketing System isn't just a tool; it's a step towards hassle-free, user-centric metro journeys.