

**OOP LAB (V-8)**

**PROJECT REPORT**

**RAILWAY MANGEMENT SYSTEM**

**NAME:** ATIQ UR REHMAN

**STUDENT ID:** F2023266530

**1. Introduction**

This project implements a railway reservation system using C++ Object Oriented Programming (OOP) concepts. It offers functionalities for managing train reservations, food orders, and employee details, employee management.

**2. System Design**

The system utilizes a menu-driven interface for user interaction. Users can perform various actions based on their roles (user or administrator).

**3. Class Description**

**1. train Class**

**-Attributes:**

**- train\_name:** Name of the train.

**- train\_no:** Train number.

**- departure\_time:** Departure time of the train.

**- arrival\_time:** Arrival time of the train.

**- departure\_location:** Departure location of the train.

**- destination\_location:** Destination location of the train.

**- fare:** Fare of the train.

**- Methods:**

**- addintrain():** Adds new train schedules.

**- modifyschdule():** Modifies existing train schedules.

**- display():** Displays train schedules.

**- searchtrain():** Searches for a train based on criteria.

**2. ticket Class (Inherits from train)**

**- Attributes:**

**- total:** Total fare for the tickets.

**- no:** Number of passengers.

**- tarinnn\_no:** Train number for booking.

**- Methods:**

**- orderticket():** Books tickets for passengers.

**- payticketbill():** Processes payment for tickets.

**- dis():** Displays user menu after booking.

**3. food Class**

**- Attributes:**

**- itemname:** Name of the food item.

**- price:** Price of the food item.

**- Methods:**

**- modifyfood():** Modifies existing food items.

**- addinfood():** Adds new food items.

**- displayfood():** Displays the food menu.

**- orderfood():** Orders food items.

**- payfood():** Processes payment for food orders.

**4. details Class (Abstract Class)**

**-Methods:**

**- employeedetails():** Pure virtual function to display employee details.

**5. employee Class (Inherits from details)**

**- Methods:**

**- employeedetails():** Displays employee details.

**- searchbydepartment():** Searches employees by department.

**- addinemployee():** Adds new employee details.

**- changeinemployee():** Modifies existing employee details.

**- searchspecificeemployee():** Searches for a specific employee.

**6. user Class**

**- Attributes:**

**- choice:** User's choice for FAQs.

**- Methods:**

**- faqs():** Displays FAQs and handles user input.

**7. management Class (Inherits from train)**

**- Attributes:**

**- username:** Admin username.

**- password:** Admin password.

**- admin\_username:** Admin username for login.

**- admin\_password:** Admin password for login.

**- adminname:** New admin username.

**- adminpassword:** New admin password.

**- adminpin:** Admin pin for account creation.

**- Methods:**

**- setConsoleColor():** Sets the console text color.

**- displayy():** Displays the main menu with arrow key navigation.

**- handleMenu():** Handles menu navigation using arrow keys.

**- captcha():** Displays a captcha for verification.

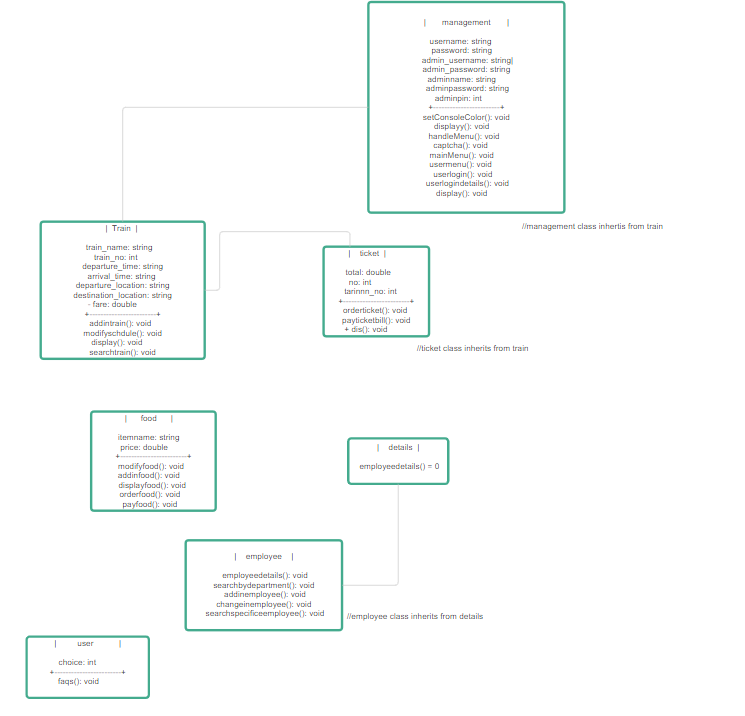
**- mainMenu():** Displays the main admin menu.

**- usermenu():** Displays the user menu.

**- userlogin():** Handles user login.

**- userlogindetails():** Displays all user login details.

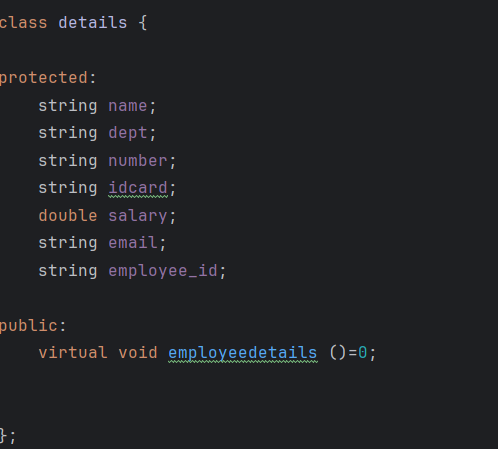
**- display():** Displays the main menu.

**4.UML DIAGRAM OF CLASSES**

**5.OOP Concepts Used**

**a.Abstraction**

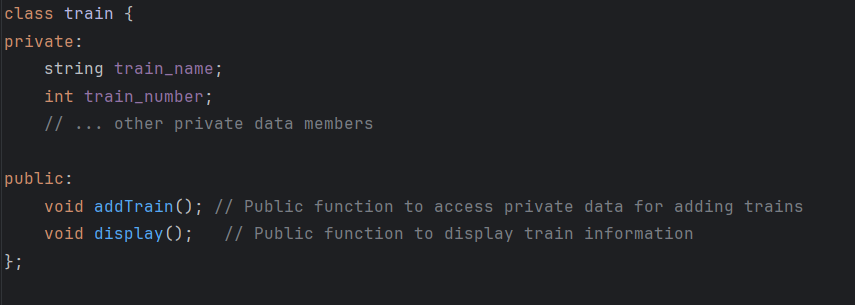
The abstract class **details** defines the template for employee information. This hides the specific implementation details of the employee class and provides a common interface for accessing employee data.

****

**b.Encapsulation**

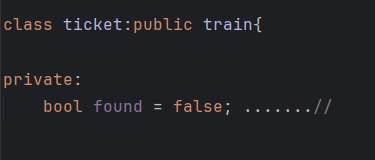
Each class encapsulates its data members and member functions. Data members are typically declared private, while functions provide controlled access to manipulate the data. This promotes data integrity and security.

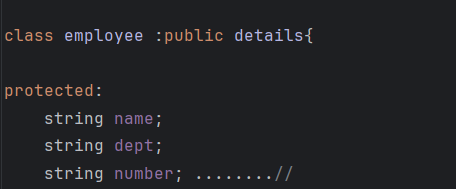
**For example (train class)**

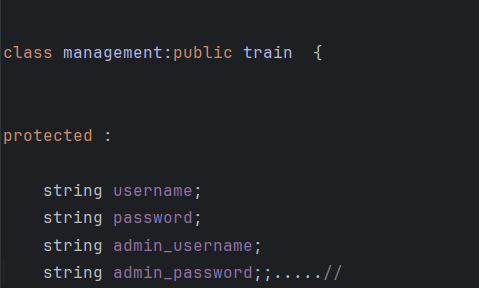
****

**c. Inheritance**

The **ticket** class inherits from the **train** class, and the **employee** class inherits from the **details** class , **management** class inherits from **train** class This allows the derived classes to reuse and extend the functionality of the base classes.







**d. Polymorphism**

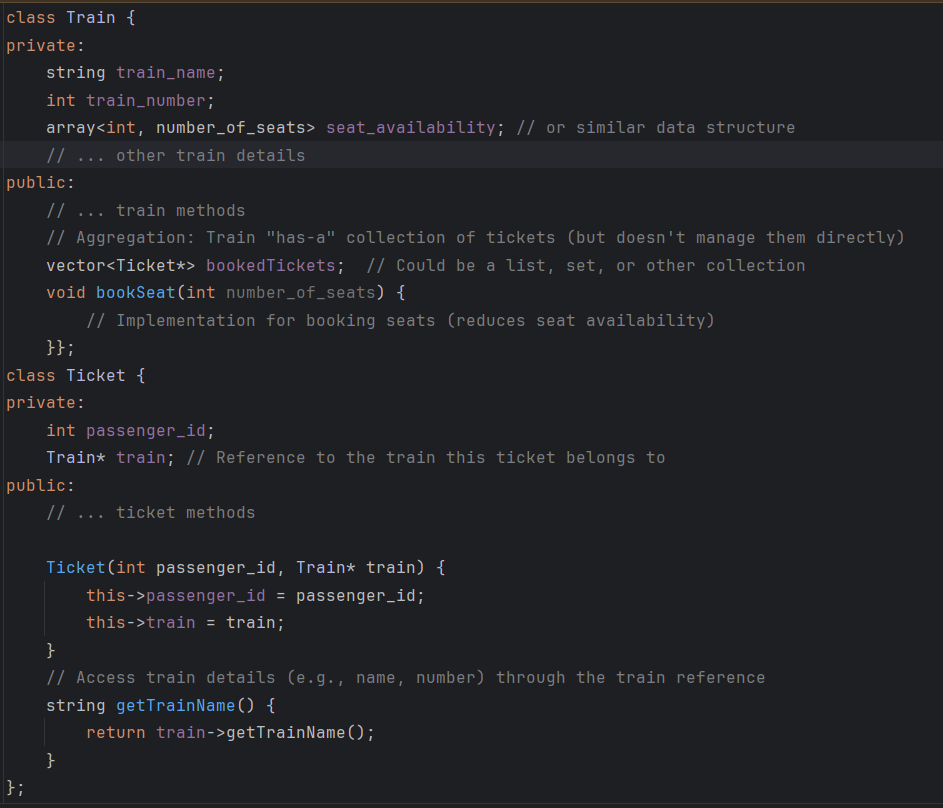
Concept of polymorphismis also covered in this project .

**FOR EXAMPLE:**

The train class is used as a base class for the ticket class. The **display()** method in the train class is overridden in the ticket class to provide specific functionality.

**Aggregation**

The project likely exhibits aggregation in the relationship between the **train** class and the **ticket** class (assuming **ticket** doesn't manage seat availability directly).



**Explanation:**

* The **Train** class maintains a collection of **Ticket** objects **(bookedTickets**) using a vector (or another collection type).
* However, the **Train** class doesn't directly manage the **Ticket** objects' lifecycle. Tickets can potentially exist outside the context of a specific train (e.g., canceled tickets).
* This demonstrates a weak "has-a" association between **Train** and **Ticket**, suggesting aggregation.

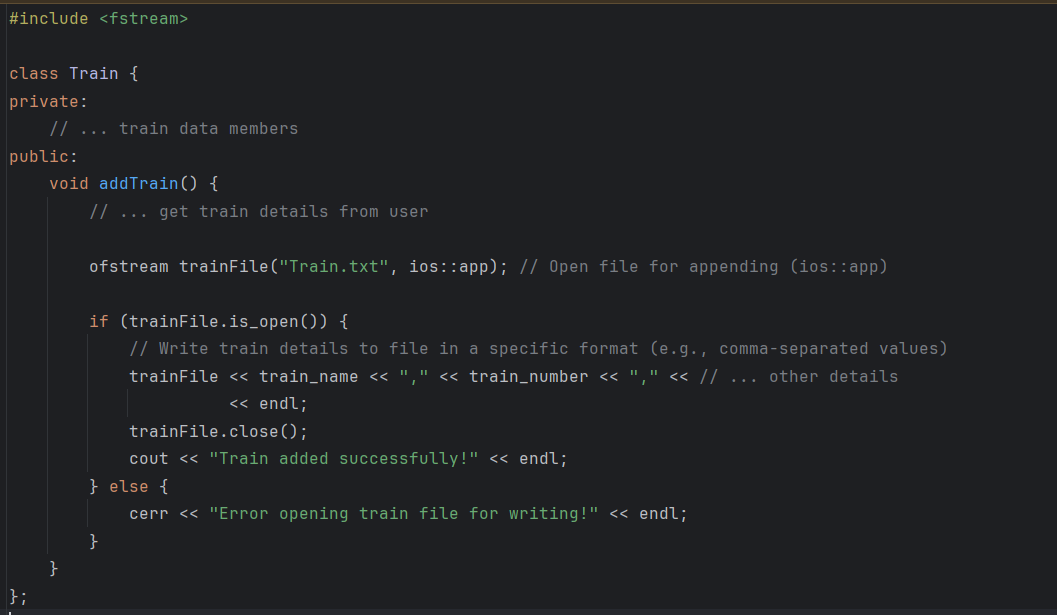
**File Handling**

Based on the class descriptions and functionalities mentioned, file handling is probably used in the following areas:

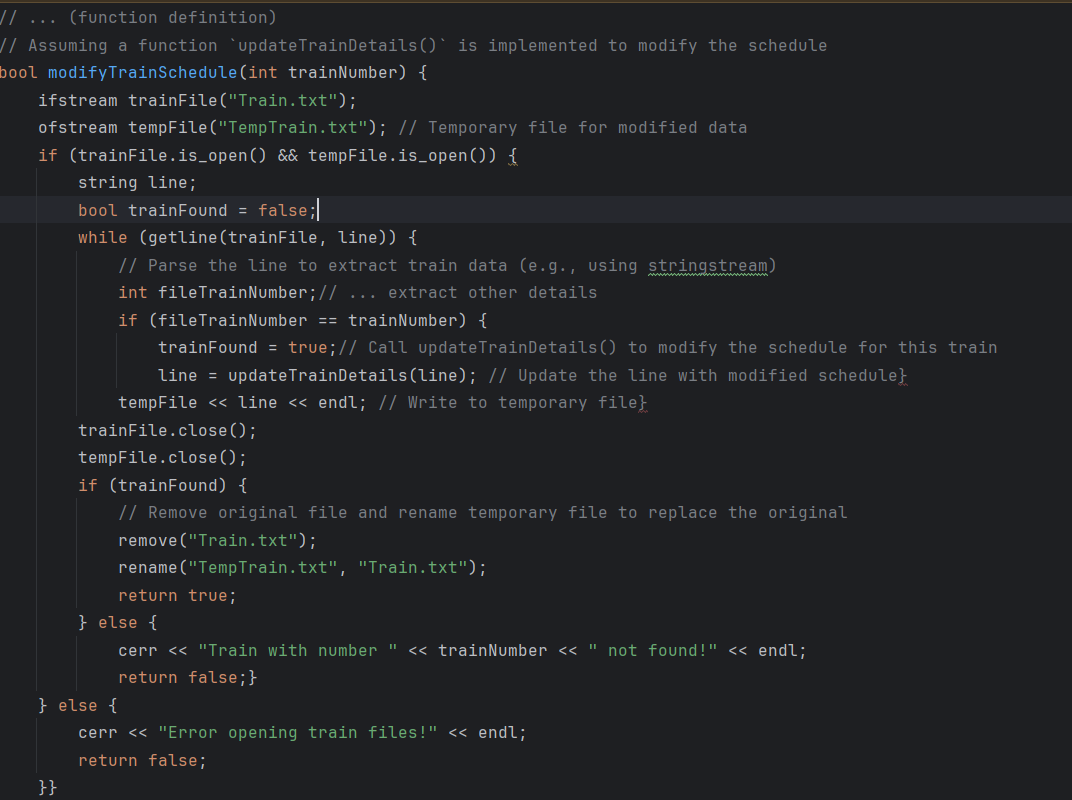
**1.Train Data:**

* **addTrain():** Likely involves writing train details (name, number, schedule, fare) to a file (e.g., **Train.txt**).
* **modifySchedule():** Might involve reading train data from the file, modifying the schedule, and then rewriting the entire data structure back to the file.
* **searchTrain():** Might involve reading train data from the file to search for trains based on specific criteria.

**addTrain()** **(Writing Train Data to a File):**

****

**modifySchedule() (Reading and Updating Train Data):**



**2.Employee Data Management (Similar Approach):**

* You can create similar functions for **addInEmployee()** to write new employee data and **changeInEmployee()** to read, modify, and rewrite employee data to a file named **employee.txt**.

**3.Ticket Data (Potential):**

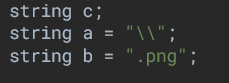
* **orderTicket():** Could potentially involve creating a new ticket record with passenger information and train details, then writing it to a file **(e.g., Tickets.txt).**
* **cancelTicket():** Might involve reading ticket data from a file, finding the canceled ticket, and potentially removing it or marking it as canceled.

**4.Food Data (Potential):**

* **addInfood():** Likely involves writing new food item details (name, price, quantity) to a file **(e.g., food.txt).**
* **modifyFood():** Might involve reading food data from the file, finding the item to modify, and then rewriting the updated data back to the file.
* **displayFood():** Probably involves reading food data from the file to display the available items and their prices.

**Captcha code**

The captcha function is defined to perform the CAPTCHA verification process. It does not take any parameters and does not return any value.



**`c`:** A string variable to store the user's input.

`**a`:** A string containing a backslash, used to construct the file path.

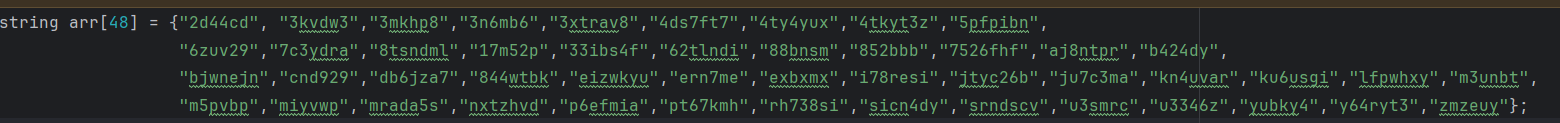
**`b`:** A string containing the file extension `.png`, used to construct the file path.

**Random Seed Initialization:**

****

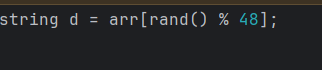
`**srand(time(0))`:** Initializes the random number generator with the current time. This ensures that the random numbers generated are different each time the program runs.

**CAPTCHA Code Array:**

****

`**arr`:** An array of strings containing forty eight CAPTCHA codes. These codes are used to simulate the **CAPTCHA images.**

**Random CAPTCHA Selection:**

****

`**rand() % 48**`: Generates a random index **(0 or 47)** to select one of the CAPTCHA codes from the `arr` array.

**`d`:** Stores the randomly selected CAPTCHA code.

**Constructing the File Path:**



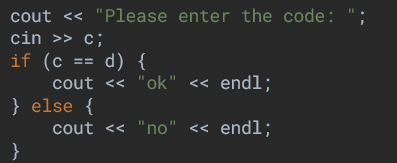
**`fullPath`:** Constructs the full file path to the CAPTCHA image by concatenating the base path **(`path`),** the backslash **(`a`),** the randomly selected CAPTCHA code **(`d`),** and the file extension **(`b`).**

**Displaying the CAPTCHA Image:**

****

`**system(fullPath.c\_str())`:** Executes a system command to open the CAPTCHA image file. The **`c\_str()`** function converts the **`fullPath`** string to a C-style string (a null-terminated character array), which is required by the **`system`** function.

**User Input and Verification:**

****

`cout << "Please enter the code: "`: Prompts the user to enter the CAPTCHA code.

`cin >> c`: Reads the user's input and stores it in the variable `c`.

`if (c == d)`: Compares the user's input (`c`) with the randomly selected CAPTCHA code (`d`).

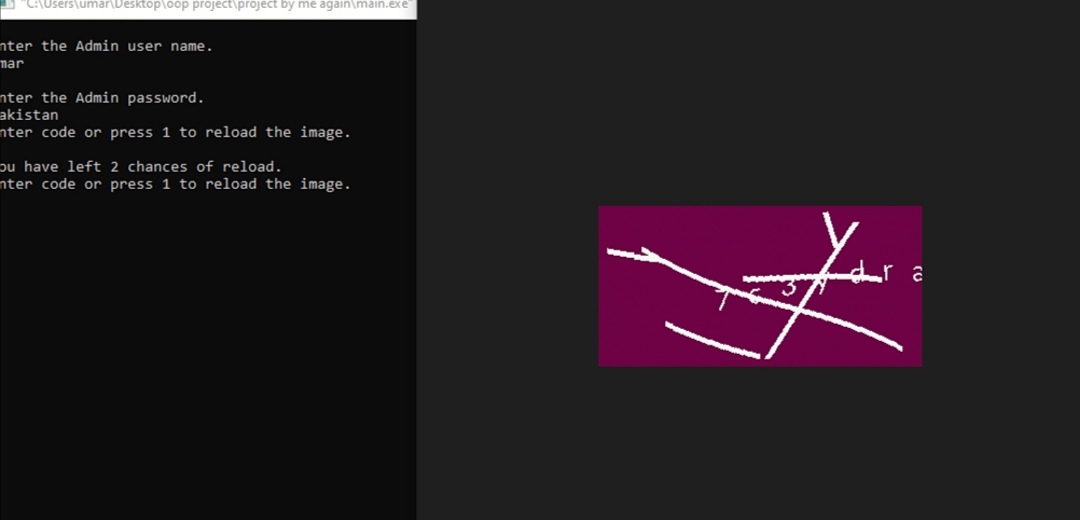
**Captcha images saved in a folder:**

****

**Summary:**

* The `captcha` function simulates a CAPTCHA verification process by:
* Randomly selecting a CAPTCHA code from a predefined array.
* Constructing the file path to the corresponding CAPTCHA image.
* Displaying the CAPTCHA image using a system command.
* Prompting the user to enter the CAPTCHA code.
* Verifying the user's input and providing feedback.

**Output:**

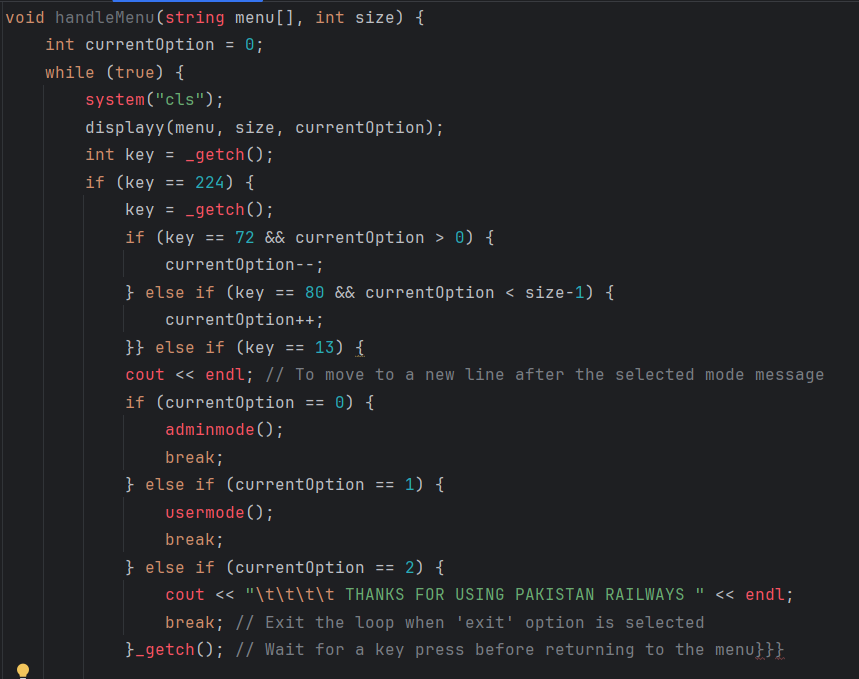


**Additional Functions:**

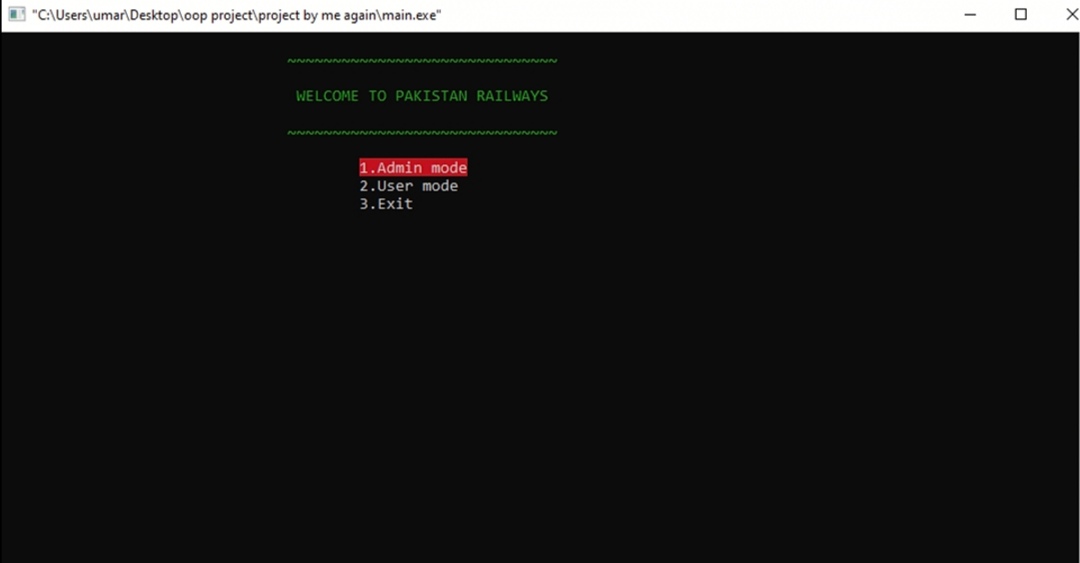
**Arrow Key Navigation:**

* **Function:** handleMenu()
* **Description:** This function allows the user to navigate the menu using arrow keys. It highlights the selected option and executes the corresponding action when the Enter key is pressed.

**Code:**

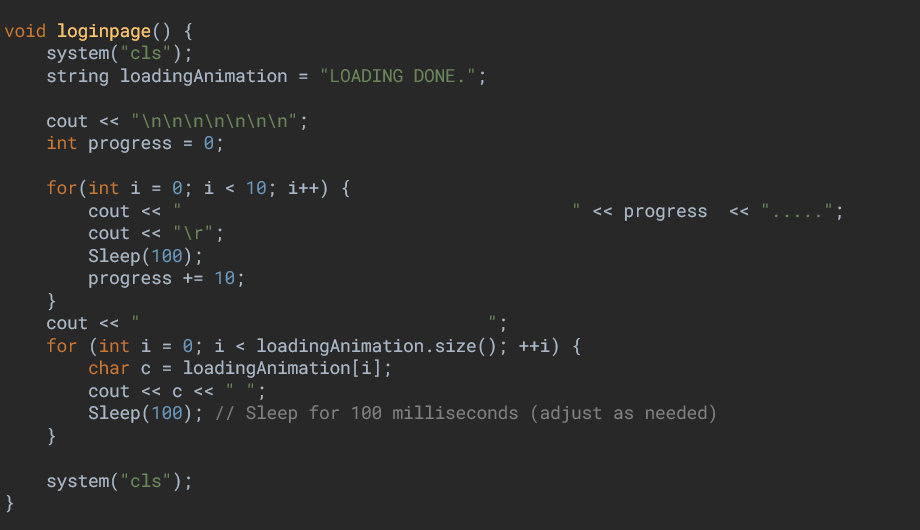
****

**OUTPUT:**



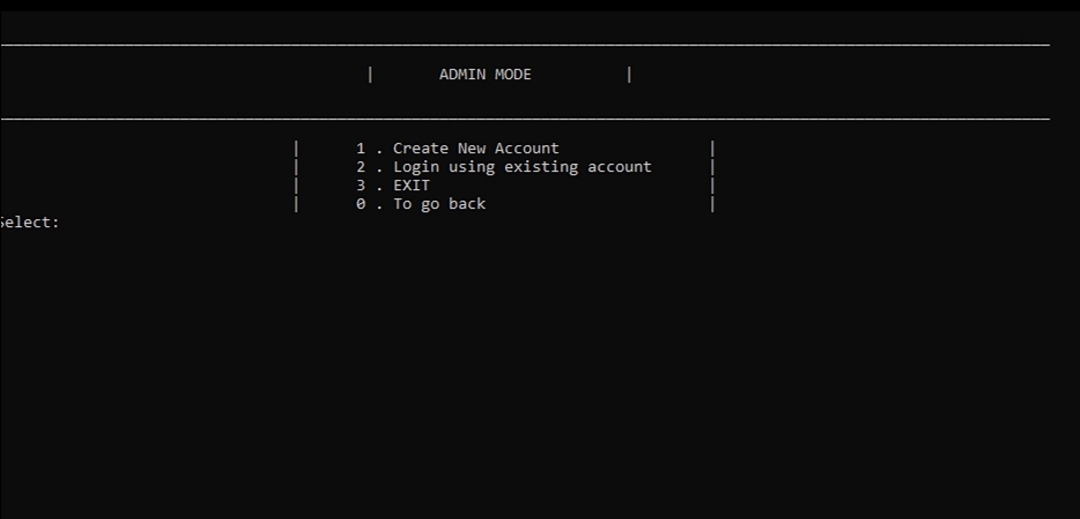
**Loading Panel:**

* **Function:** loginpage()
* **Description:** This function displays a loading animation when the application starts. It simulates a loading process by printing dots and a loading message with a delay.

**Code:** ****

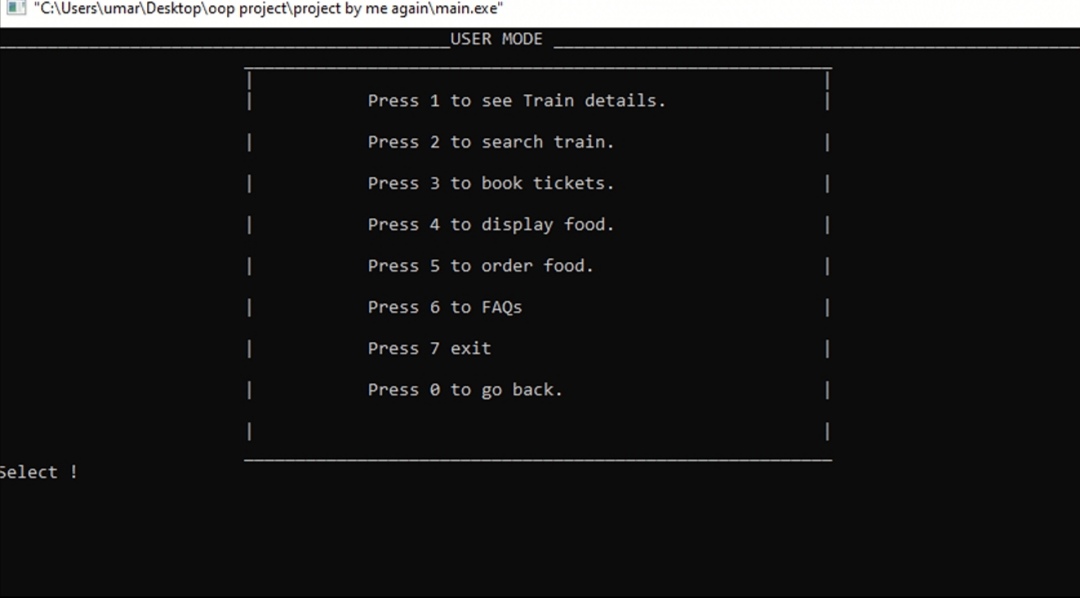
**OUTPUTS**

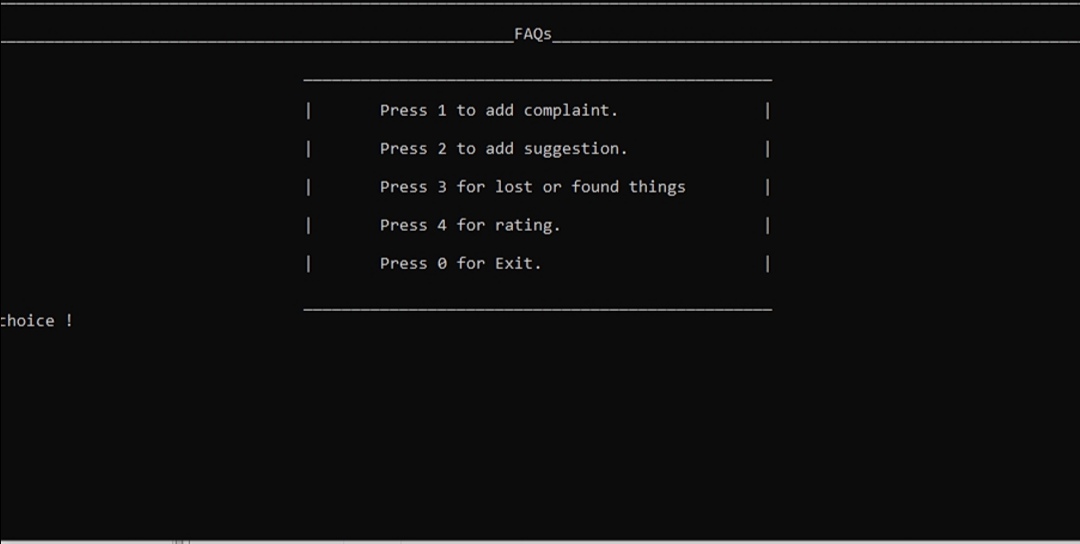
**Admin mode :**



**User mode :**



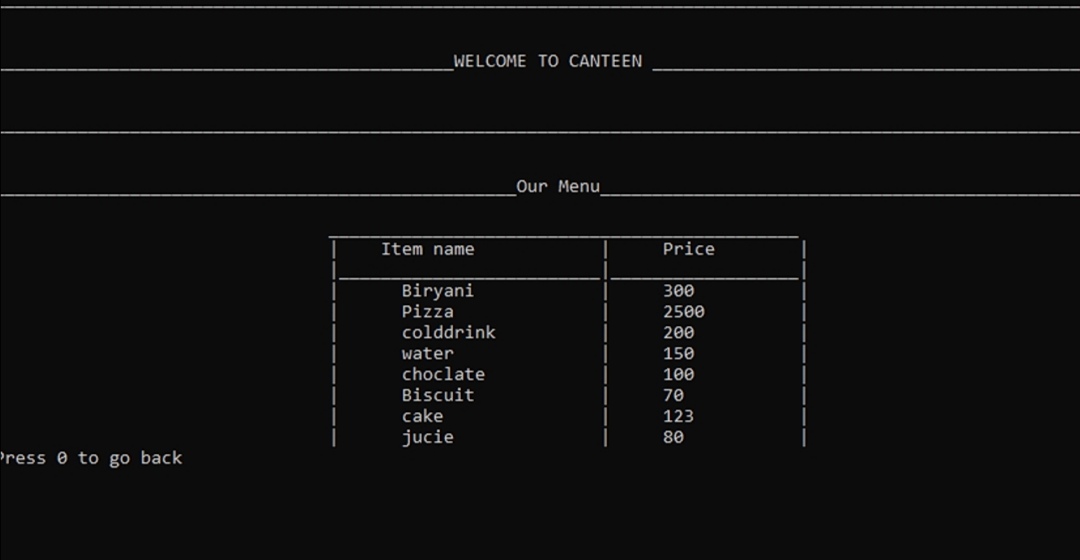




**Train details:**



**Canteen menu:**



**Conclusion:**

This railway reservation system demonstrates the use of core OOP concepts in C++. It provides a user-friendly interface for managing train reservations, food orders, and employee details. By incorporating error handling and utilizing standard C++ libraries, the project can be further enhanced for robustness and wider compatibility.