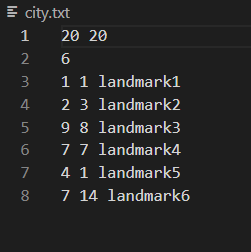
**Requirements:**

1. city.txt -> This text file contains the information about our 2D grid and the coordinates of all the landmarks which are present in the grid. The format of input should be as follows:

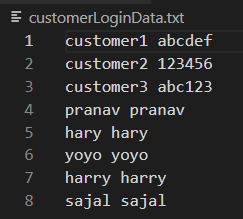


Here, the first line are the rows and columns of the grid respectively.

The second line contains the number of landmarks in the city (say m).

All the following ‘m’ lines contain 2 integers and a string, the first and second integers being the abscissa and the ordinates of the landmark respectively. The string is the name of the landmark.

1. customerLoginData.txt -> This text file contains the login details of the customers which have been registered in the database. The input format should be as follows:

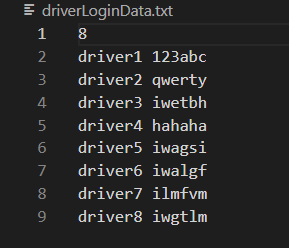


If there are ‘n’ registered users, this file will contain ‘n’ lines, each line containing exactly 2 space separated strings. The first string will be the name of a user and the second string will be their password.

(A drawback of this program is that the username **must not** contain a space.)

If we add a new user to the application, the login details of the user will be appended in this file.

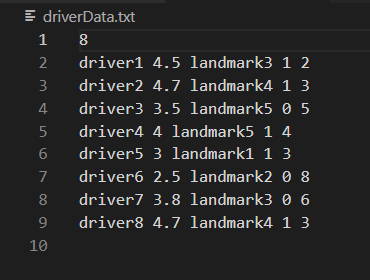
1. driverLoginData.txt -> This file contains the login details of the drivers registered to the app. The only difference from ‘customerLoginData.txt’ is that no new drivers can be added in this file. The input format for the file should be as follows :-



The first line contains the number of drivers registered with the app (say m).

The next ‘m’ lines will contain exactly 2 space separated strings, the first of them being the username and second being the respective password for a driver. Note that we **must not** have usernames which contain spaces for proper functioning of the program.

1. driverData.txt -> This input file will contain all the necessary details for the driver object (like name, speed, current location, etc.). The input format is explained below:-

The first line will contain the number of registered drivers in the app (say m).

The ‘m’ lines that follow will contain the following things, all separated by spaces :-

1. A string - the name of the driver
2. A floating point value - the performance rating of the driver
3. A string - name of the current location (a landmark) of the driver
4. An integer value - This is set to 0 if the driver is not available and is 1 if the driver is available
5. An integer value - The speed at which the driver drives the cab.

**Description Of Classes**

1. **City :-**

This class contains the following data members :-

Rows - The number of rows in the imaginary 2D grid. (m)

Columns - The number of columns in the grid. (n).

Grid - An m\*n matrix.

It also contains the following methods :-

City() - a default constructor. This constructor does not specify the number of rows or columns of the city and hence if an object is created using this, the instance variables of that object will contain default values.

City(int rows, int columns) - This is a parameterized constructor which initializes the instance variables of the object. This constructor also initializes the grid.

These setter methods ensure that the data remains encapsulated.

setRows() - helps us set the number of rows in the city.

setColumns - helps us set the columns of the city.

makeGrid() - this method initializes the grid matrix of the object by allotting it memory appropriate to the number of rows and columns that have been taken as input.

checkLandmark(List<Landmark> l) :- This method makes sure that the given input is correct. If there exists a landmark which does not lie on our 2D grid, this method will return false and the program will terminate.

1. **Landmark :-**

This class has the following data members:

Name - The name of a landmark in the city.

Abscissa - The x coordinate of the landmark.

Ordinate - The y coordinate of the landmark.

This class also has the following methods :-

Landmark() - This is the default constructor. Whenever an object is created using this constructor, the origin is set as a landmark.

Landmark(Landmark obj) - This is the constructor which can be used to create a new Landmark object using a reference from a previously existing landmark.

setName(String s) - This method is used to set the name of the landmark.

setAbscissa(int x) - This method is used to set the x coordinate of the landmark.

setOrdinate(int y) - This method is used to set the y coordinate of the landmark.

getAbscissa() - This method returns the x coordinate of the current landmark.

getOrdinate() - This method return the y coordinate of the current landmark.

1. **Driver :-**

This class contains the following data members:

name: The name of the driver (taken as input).

rating: Performance rating of the driver (taken as input).

currentLocation: This is the landmark where driver is currently at (name of the landmark is taken as input).

available: decides the availability status of the driver. (taken as input)

speed: The speed at which driver drives the cab. (taken as input).

eta: The estimated time in which driver will pick up the customer.

notify: This helps the driver know if there are any available customers.

This class contains the following methods:

calculateEta(int pickupX, int pickupY) : This method takes the pickup location of the customer as an input and returns the ETA using the current location of our driver.

startRide() : Driver calls this method after the ride has started.

endRide() : Driver calls this method when the ride has ended.

getName() : returns the name of the driver.

setRating(float rating) : allows us to set the performance rating of the driver.

setSpeed(int speed) : allows us to set the speed of the driver.

setCurrentLocation (Landmark l) : initializes the current location of the driver.

getRating() : returns the performance rating of the driver.

setName() : allows us to set the name of the newly registered driver.

changeAvailableStatus() : Driver can call this method whenever he wants to change his availability status.

Driver() : default constructor for the driver class.

Driver(String name, float rating, Landmark currentLocation, boolean available, int speed) : Parameterized constructor of the driver class which initializes the instance variables of the driver object.

1. **Customer:**

This class contains the following data members:

name: The name of the customer.

pickupLocation : The pickup location of the customer.

destination: The destination of the customer.

This class contains the following methods:

cancelBooking(Driver d) :- When the driver has accepted the ride, this method gives a choice to the customer and allows it to cancel the booking. The program will terminate if the user decides to do so.

setDestination(String name, int x, int y) :- this method sets the destination location of the customer by creating a new Landmark object using the input parameters.

setPickupLocation(String name, int x, int y) :- this method works in a similar way to the setDestination method and initializes the pickup Location of the customer by creating a new Landmark object.

**The working of our code is explained in the Journey Class.**

**Journey:**

Function countLineFast -> This method is used to count the number of lines in the given text file.

**This class contains the following data members:**

Scanner sc = Scanner sc is used to open city.txt and driverData.txt .

Scanner lg = Scanner lg is used to open customerLoginData.txt file and DriverLoginData.txt file .

Scanner in= To input from the console.

**Files used :-**

City.txt = Information about the dimensions of the city and various landmarks available in the city.

Driver.txt= Contains the name of drivers, their ratings, their current locations, availability status and speed for each driver respectively.

CustomerLoginData.txt = Contains name of the customer and their passwords.

DriverLoginData.txt= Name of the drivers and their passwords.

**Working of code**

**Login:-**

On running the code a choice is given to the user to either login or create a new account. If the user chooses to create a new account then the user is asked about their username. The username entered should be unique i.e. it shouldn’t be in the database already. If it is in the database then the program will keep on asking to enter any other username or enter ‘quit’ to terminate the program. After entering a unique username the user is asked for a password and after that both these details are stored in the database and the user is moved to the login page where he should re-enter these details for login.

On login screen the users have to enter the correct username(i.e. it should be present in the database) and password to login successfully. If the username enters incorrect username then code will keep on asking for a valid username (or quit to terminate). After entering correct username there are 3 chances to enter the correct password. If not done in 3 chances the program will terminate.

**List<landmarks>** **landmarks** stores all the available landmarks from the the city.txt file.

After this there is a check if all the landmarks entered in the city.txt are within the coordinates of the city or not. If they are not then the program terminates with a message stating that landmarks can not be out of city, and asks the users to check their input (the text files).

**Customer journey details:-**

After logging in successfully the customer is asked about the pickup location. The entered pickup location must be an available landmark, if not the program will keep on asking about the user to enter valid location. Similarly destination is asked and it should be valid as well as separate from the pickup location for obvious reasons.

**Available Driver details :-**

After entering the pickup location and destination a list of available drivers along with their ETA’s and ratings is displayed. Also the fare of the journey is displayed and the user has to enter the index of the driver that they want to chose for the drive. Note that the fare is same for all drivers since it depends on the distance between the pickup location and destination and not between the driver and pickup location. The user has to enter the index of the driver chosen by them and if entered incorrectly it will keep on asking to enter correct index. After this the control is shifted to the chosen driver.

**Login for driver :-**

The chosen driver is asked to enter their password. Again 3 tries are given to enter correct password. If not entered correctly in 3 chances, a message showing that driver is unable to login is printed and the control is given back to the customer. The same list of drivers is shown again, displaying the available drivers and allowing the user to choose another driver.

If the password is entered correctly then the driver is logged into the app and the journey details are shown to him with the pickup location, destination and fare provided. The driver has option to either accept the ride or reject it. If the ride is rejected the control is again given back to customer and the message is shown about rejection of ride and the user is asked to chose another driver.

If the driver accepts the ride then continuous ETA is displayed on the screen according to condition that one second in reality means 30 seconds in the app. After the ETA is 0 a message is shown that the ride has reached and after 3 seconds(just for reference) a message is shown that the ride has ended.