Volt

Generated by Doxygen 1.9.6

Chapter 1

Volt

Electric Mobility

1.1 Motivação

Este projeto da Unidade Curricular (UC) Estruturas de Dados Avançadas (EDA), integrada no 2º semestre do 1º ano da Licenciatura em Engenharia de Sistemas Informáticos, visa o reforço e a aplicação dos conhecimentos adquiridos ao longo do semestre.

Com este projeto de avaliação pretende-se sedimentar os conhecimentos relativos à definição e manipulação de estruturas de dados dinâmicas na linguagem de programação C.

O âmbito deste projeto reside no desenvolvimento de uma solução de software na área da micromobilidade. O crescente ecossistema de novas formas de mobilidade social, nomeadamente aquelas que ocorrem entre distâncias curtas, tem promovido a necessária integração de múltiplos meios de deslocação. Esta transformação na forma como a mobilidade é realizada, fator essencial para o desenvolvimento dos espaços, cidades e outros, irá depender de ações que permitam agilizar a utilização dos meios de transporte que suportem uma mobilidade mais fácil, rápida, limpa e económica, como por exemplo os meios de mobilidade elétrica (trotinetes, bicicletas, etc.)

1.2 Objetivo

A essência deste projeto prende-se com o desenvolvimento de uma solução de software que permita agilizar a gestão (registo, partilha, utilização) de meios de mobilidade urbana num contexto de uma smart-city.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 adjacent Struct Reference

`#include <header.h>`

Data Fields

- char id [SIZE_LOCATION]
- float distance
- struct adjacent * next

4.1.1 Field Documentation

4.1.1.1 distance

```
float distance
```

4.1.1.2 id

```
char id[SIZE_LOCATION]
```

4.1.1.3 next

```
struct adjacent* next
```

The documentation for this struct was generated from the following file:

- header.h

4.2 client Struct Reference

`#include <header.h>`

Data Fields

- int id
- char username [SIZE_USERNAME]
- char password [SIZE_PASSWORD]
- char name [SIZE_NAME]
- int nif
- char location [SIZE_LOCATION]
- float balance
- int available
- struct client * next

4.2.1 Field Documentation

4.2.1.1 available

`int available`

4.2.1.2 balance

`float balance`

4.2.1.3 id

`int id`

4.2.1.4 location

`char location[SIZE_LOCATION]`

4.2.1.5 name

```
char name[SIZE_NAME]
```

4.2.1.6 next

```
struct client* next
```

4.2.1.7 nif

```
int nif
```

4.2.1.8 password

```
char password[SIZE_PASSWORD]
```

4.2.1.9 username

```
char username[SIZE_USERNAME]
```

The documentation for this struct was generated from the following file:

- header.h

4.3 collection Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- char startLocation [SIZE_LOCATION]
- time_t datetime
- int manager
- struct point * points
- struct collection * next

4.3.1 Field Documentation

4.3.1.1 datetime

```
time_t datetime
```

4.3.1.2 id

```
int id
```

4.3.1.3 manager

```
int manager
```

4.3.1.4 next

```
struct collection* next
```

4.3.1.5 points

```
struct point* points
```

4.3.1.6 startLocation

```
char startLocation[SIZE_LOCATION]
```

The documentation for this struct was generated from the following file:

- header.h

4.4 integer Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- struct integer * next

4.4.1 Field Documentation

4.4.1.1 id

```
int id
```

4.4.1.2 next

```
struct integer* next
```

The documentation for this struct was generated from the following file:

- header.h

4.5 location Struct Reference

```
#include <header.h>
```

Data Fields

- char id [SIZE_LOCATION]
- char name [SIZE_LOCATION]
- struct adjacent * adjacents
- struct location * next

4.5.1 Field Documentation

4.5.1.1 adjacents

```
struct adjacent* adjacents
```

4.5.1.2 id

```
char id[SIZE_LOCATION]
```

4.5.1.3 name

```
char name[SIZE_LOCATION]
```

4.5.1.4 next

```
struct location* next
```

The documentation for this struct was generated from the following file:

- header.h

4.6 manager Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- char username [SIZE_USERNAME]
- char password [SIZE_PASSWORD]
- char name [SIZE_NAME]
- struct manager * next

4.6.1 Field Documentation

4.6.1.1 id

```
int id
```

4.6.1.2 name

```
char name[SIZE_NAME]
```

4.6.1.3 next

```
struct manager* next
```

4.6.1.4 password

```
char password[SIZE_PASSWORD]
```

4.6.1.5 username

```
char username[SIZE_USERNAME]
```

The documentation for this struct was generated from the following file:

- header.h

4.7 point Struct Reference

```
#include <header.h>
```

Data Fields

- char id [SIZE_LOCATION]
- struct integer * collected
- struct point * next

4.7.1 Field Documentation

4.7.1.1 collected

```
struct integer* collected
```

4.7.1.2 id

```
char id[SIZE_LOCATION]
```

4.7.1.3 next

```
struct point* next
```

The documentation for this struct was generated from the following file:

- header.h

4.8 ride Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- int vehicle
- int client
- time_t startTime
- time_t endTime
- char startLocation [SIZE_LOCATION]
- char endLocation [SIZE_LOCATION]
- float cost
- float distance
- struct ride * next

4.8.1 Field Documentation

4.8.1.1 client

```
int client
```

4.8.1.2 cost

```
float cost
```

4.8.1.3 distance

```
float distance
```

4.8.1.4 endLocation

```
char endLocation[SIZE_LOCATION]
```

4.8.1.5 endTime

```
time_t endTime
```

4.8.1.6 id

```
int id
```

4.8.1.7 next

```
struct ride* next
```

4.8.1.8 startLocation

```
char startLocation[SIZE_LOCATION]
```

4.8.1.9 startTime

```
time_t startTime
```

4.8.1.10 vehicle

```
int vehicle
```

The documentation for this struct was generated from the following file:

- header.h

4.9 type Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- char name [SIZE_NAME]
- float cost
- struct type * next

4.9.1 Field Documentation

4.9.1.1 cost

```
float cost
```

4.9.1.2 id

```
int id
```

4.9.1.3 name

```
char name[SIZE_NAME]
```

4.9.1.4 next

```
struct type* next
```

The documentation for this struct was generated from the following file:

- header.h

4.10 vehicle Struct Reference

```
#include <header.h>
```

Data Fields

- int id
- int type
- float battery
- float range
- char location [SIZE_LOCATION]
- int available
- struct vehicle * next

4.10.1 Field Documentation

4.10.1.1 available

```
int available
```

4.10.1.2 battery

```
float battery
```

4.10.1.3 id

```
int id
```

4.10.1.4 location

```
char location[SIZE_LOCATION]
```

4.10.1.5 next

```
struct vehicle* next
```

4.10.1.6 range

```
float range
```

4.10.1.7 type

```
int type
```

The documentation for this struct was generated from the following file:

- header.h

4.11 visited Struct Reference

```
#include <header.h>
```

Data Fields

- char id [SIZE_LOCATION]
- struct visited * next

4.11.1 Field Documentation

4.11.1.1 id

```
char id[SIZE_LOCATION]
```

4.11.1.2 next

```
struct visited* next
```

The documentation for this struct was generated from the following file:

- header.h

Chapter 5

File Documentation

5.1 header.h File Reference

```
#include <time.h>
```

Data Structures

- struct integer
- struct type
- struct vehicle
- struct client
- struct manager
- struct ride
- struct point
- struct collection
- struct adjacent
- struct location
- struct visited

Macros

- #define DATA_DIR "data/"
- #define SIZE_USERNAME 40
- #define SIZE_PASSWORD 40
- #define SIZE_NAME 60
- #define SIZE_LOCATION 60
- #define SIZE_TYPE 5
- #define SIZE_BATTERY 15
- #define SIZE_RANGE 15
- #define SIZE_NIF 15
- #define SIZE_DATETIME 20
- #define HQ "tatica.ideia.morno"
- #define RED "\x1B[31m"
- #define GREEN "\x1B[32m"
- #define YELLOW "\x1B[33m"
- #define BLUE "\x1B[34m"
- #define MAGENTA "\x1B[35m"
- #define CYAN "\x1B[36m"
- #define WHITE "\x1B[37m"
- #define RESET "\x1B[0m"

Typedefs

- typedef struct integer Integer
- typedef struct type Type
- typedef struct vehicle Vehicle
- typedef struct client Client
- typedef struct manager Manager
- typedef struct ride Ride
- typedef struct point Point
- typedef struct collection Collection
- typedef struct adjacent Adjacent
- typedef struct location Location
- typedef struct visited Visited

Functions

- void ridesMain ()
- Ride * insertRide (Ride *head, int id, int vehicle, int client, int startTime, int endTime, char startLocation[ ], char endLocation[ ], float cost, float distance)
- Ride * startRide (Ride *head, Vehicle *headVehicles, Type *headTypes, Client *headClients, int id, int vehicle, int client)
- void endRide (Ride *head, Vehicle *headVehicles, Type *headTypes, Client *headClients, Location *head↩ Locations, int id, char endLocation[ ])
- int listRides (Ride *head, Client *headClients)
- int listRidesClient (Ride *head, Client *headClients, int id)
- int assignRideId (Ride *head)
- int currentRide (Ride *head, int id)
- void showRide (Ride *head, int id)
- int saveRides (Ride *head)
- Ride * readRides ()
- void vehiclesMain ()
- Vehicle * insertVehicle (Vehicle *head, int id, int type, float battery, float range, int available, char location[ ])
- Vehicle * removeVehicle (Vehicle *head, int id)
- void editVehicle (Vehicle *head, Type *headTypes, int id, int type, float battery, float range, char location[ ])
- int listVehicles (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int listVehiclesByRange (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int listVehiclesByBattery (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int listVehiclesInLocation (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int listVehiclesByDistance (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int listVehiclesInRadius (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ], float radius)
- int listVehiclesByTypeInRadius (Vehicle *head, Type *headTypes, Location *headLocations, int type, char location[ ], float radius)
- int listVehiclesByBatteryHalfCharged (Vehicle *head, Type *headTypes, Location *headLocations, char location[ ])
- int existVehicle (Vehicle *head, int id)
- int assignVehicleId (Vehicle *head)
- int isVehicleAvailable (Vehicle *head, int id)
- int isVehicleCharged (Vehicle *head, int id)
- void updateVehicleLocation (Vehicle *head, int id, char location[ ])
- Vehicle * chargeVehicles (Vehicle *head, char location[ ])
- Vehicle * copyLinkedList (Vehicle *head)
- int saveVehicles (Vehicle *head)
- Vehicle * readVehicles ()

- char * getVehicleTypeName (Vehicle *head, Type *headTypes, int id)
- float getVehicleBattery (Vehicle *head, int id)
- char * getVehicleLocation (Vehicle *head, int id)
- float getVehicleCost (Vehicle *head, Type *headTypes, int id)
- float getTypeCost (Type *head, int id)
- char * getTypeName (Type *head, int id)
- Type * insertType (Type *head, int id, char name[ ], float cost)
- int listTypes (Type *head)
- int existType (Type *head, int id)
- int saveTypes (Type *head)
- Type * readTypes ()
- void locationsMain ()
- Location * createLocation (Location *head, char id[ ], char name[ ])
- int existLocation (Location *head, char id[ ])
- char * getLocationName (Location *head, char id[ ])
- float getDistance (Location *head, char origin[ ], char destination[ ])
- Location * createEdge (Location *head, char origin[ ], char destination[ ], float distance)
- void listAdjacents (Location *head, char id[ ])
- void listGraph (Location *head)
- Location * readLocations ()
- void collectionsMain (int manager)
- Collection * collect (Collection *head, Vehicle *headVehicles, Location *headLocations, char startLocation[ ], int manager)
- Collection * insertCollection (Collection *head, int id, char startLocation[ ], time_t datetime, int manager)
- Collection * insertPoint (Collection *head, int id, char location[ ])
- Collection * insertCollected (Collection *head, int id, char location[ ], int vehicle)
- Visited * insertVisited (Visited *head, char location[ ])
- void listCollections (Collection *head, Vehicle *headVehicles, Type *headTypes)
- void listLatestCollection (Collection *head, Vehicle *headVehicles, Type *headTypes)
- int assignCollectionId (Collection *head)
- int isVisited (Visited *head, char location[ ])
- int saveCollections (Collection *head)
- Collection * loadCollections ()
- void clientsMain ()
- Client * insertClient (Client *head, int id, char username[ ], char password[ ], char name[ ], int nif, char location[ ], float balance, int available)
- Client * removeClient (Client *head, int id)
- void editClient (Client *head, int id, char username[ ], char password[ ], char name[ ], int nif, char location[ ])
- int listClients (Client *head)
- int listClient (Client *head, int id)
- char * getClientName (Client *head, int id)
- char * getClientUsername (Client *head, int id)
- char * getClientLocation (Client *head, int id)
- int existClientUsername (Client *head, char username[ ])
- int existClient (Client *head, int id)
- int assignClientId (Client *head)
- int isClientAvailable (Client *head, int id)
- void addBalance (Client *head, int id, float balance)
- void removeBalance (Client *head, int id, float balance)
- void editBalance (Client *head, int id, float balance)
- int hasBalance (Client *head, int id)
- int saveClients (Client *head)
- Client * readClients ()
- void managersMain ()
- Manager * insertManager (Manager *head, int id, char username[ ], char password[ ], char name[ ])

- Manager ∗ removeManager (Manager ∗head, int id)
- void editManager (Manager ∗head, int id, char username[ ], char password[ ], char name[ ])
- int listManagers (Manager ∗head)
- char ∗ getManagerName (Manager ∗head, int id)
- int existManagerUsername (Manager ∗head, char username[ ])
- int existManager (Manager ∗head, int id)
- int assignManagerId (Manager ∗head)
- int saveManagers (Manager ∗head)
- Manager ∗ readManagers ()
- void encrypt (char password[ ])
- int authClient (Client ∗head, char username[ ], char password[ ])
- int authManager (Manager ∗head, char username[ ], char password[ ])
- void menuApp ()
- void menuMain ()
- void menuMainClients (int available)
- void menuLine ()
- void menuAuth ()
- void menuAuthClients ()
- void menuAuthManagers ()
- void menuHeaderRides ()
- void menuHeaderRidesClient ()
- void menuHeaderVehicles ()
- void menuHeaderClients ()
- void menuHeaderClient ()
- void menuHeaderManagers ()
- void menuFooterRides ()
- void menuFooterVehicles ()
- void menuFooterCollections ()
- void menuFooterClients ()
- void menuFooterManagers ()
- void menuTitleInsertVehicle ()
- void menuTitleRemoveVehicle ()
- void menuTitleEditVehicle ()
- void menuTitleInsertClient ()
- void menuTitleRemoveClient ()
- void menuTitleEditClient ()
- void menuTitleAddBalance ()
- void menuTitleRemoveBalance ()
- void menuTitleInsertManager ()
- void menuTitleRemoveManager ()
- void menuTitleEditManager ()
- void clrscr ()
- void clrbuffer ()
- void enterToContinue ()
- void showCount (int count)

5.1.1 Macro Definition Documentation

5.1.1.1 BLUE

```
#define BLUE "\x1B[34m"
```

5.1.1.2 CYAN

```
#define CYAN "\x1B[36m"
```

5.1.1.3 DATA_DIR

```
#define DATA_DIR "data/"
```

5.1.1.4 GREEN

```
#define GREEN "\x1B[32m"
```

5.1.1.5 HQ

```
#define HQ "tatica.ideia.morno"
```

5.1.1.6 MAGENTA

```
#define MAGENTA "\x1B[35m"
```

5.1.1.7 RED

```
#define RED "\x1B[31m"
```

5.1.1.8 RESET

```
#define RESET "\x1B[0m"
```

5.1.1.9 SIZE_BATTERY

```
#define SIZE_BATTERY 15
```

5.1.1.10 SIZE_DATETIME

```
#define SIZE_DATETIME 20
```

5.1.1.11 SIZE_LOCATION

```
#define SIZE_LOCATION 60
```

5.1.1.12 SIZE_NAME

```
#define SIZE_NAME 60
```

5.1.1.13 SIZE_NIF

```
#define SIZE_NIF 15
```

5.1.1.14 SIZE_PASSWORD

```
#define SIZE_PASSWORD 40
```

5.1.1.15 SIZE_RANGE

```
#define SIZE_RANGE 15
```

5.1.1.16 SIZE_TYPE

```
#define SIZE_TYPE 5
```

5.1.1.17 SIZE_USERNAME

```
#define SIZE_USERNAME 40
```

5.1.1.18 WHITE

```
#define WHITE "\x1B[37m"
```

5.1.1.19 YELLOW

```
#define YELLOW "\x1B[33m"
```

5.1.2 Typedef Documentation

5.1.2.1 Adjacent

```
typedef struct adjacent Adjacent
```

5.1.2.2 Client

```
typedef struct client Client
```

5.1.2.3 Collection

```
typedef struct collection Collection
```

5.1.2.4 Integer

```
typedef struct integer Integer
```

5.1.2.5 Location

```
typedef struct location Location
```

5.1.2.6 Manager

```
typedef struct manager Manager
```

5.1.2.7 Point

```
typedef struct point Point
```

5.1.2.8 Ride

```
typedef struct ride Ride
```

5.1.2.9 Type

```
typedef struct type Type
```

5.1.2.10 Vehicle

```
typedef struct vehicle Vehicle
```

5.1.2.11 Visited

```
typedef struct visited Visited
```

5.1.3 Function Documentation

5.1.3.1 addBalance()

```
void addBalance (
            Client * head,
            int id,
            float balance )
```

It adds the balance to the client with the given id

Parameters

head	The head of the linked list
id	The id of the client
balance	The amount of money to add to the client's balance

5.1.3.2 assignClientId()

```
int assignClientId (
            Client * head )
```

It returns the next available client id

Parameters

head	The head of the linked list

Returns

The next available client ID.

5.1.3.3 assignCollectionId()

```
int assignCollectionId (
            Collection * head )
```

This function assigns a unique ID to a new element in a linked list.

Parameters

head	A pointer to the head of a linked list of Collection structs.

Returns

an integer value, which is the next available ID for a new collection.

5.1.3.4 assignManagerId()

```
int assignManagerId (
            Manager * head )
```

It returns the next available manager id.

Parameters

head	The head of the linked list

Returns

The id of the last manager in the list.

5.1.3.5 assignRideId()

```
int assignRideId (
            Ride * head )
```

It returns the next available ride id

Parameters

head	The head of the linked list

Returns

The next available ride id.

5.1.3.6 assignVehicleId()

```
int assignVehicleId (
            Vehicle * head )
```

It returns the next available vehicle id

Parameters

head	The head of the linked list

Returns

The next available ID number.

5.1.3.7 authClient()

```
int authClient (
            Client * head,
```

```
            char username[],
            char password[] )
```

It takes a pointer to the head of a linked list of clients, a username and a password, encrypts the password, and returns the id of the client if the username and password match, or 0 if they don't

Parameters

head	The head of the linked list
username	the username of the client
password	the password to be encrypted

Returns

The ID of the client.

5.1.3.8 authManager()

```
int authManager (
            Manager * head,
            char username[],
            char password[] )
```

It takes a pointer to a linked list of managers, a username and a password, encrypts the password, and then compares the username and password to the username and password of each manager in the linked list. If it finds a match, it returns the manager's ID. If it doesn't find a match, it returns 0

Parameters

head	pointer to the first node of the linked list
username	the username of the manager
password	the password to be encrypted

Returns

The ID of the manager.

5.1.3.9 chargeVehicles()

```
Vehicle * chargeVehicles (
            Vehicle * head,
            char location[] )
```

The function charges all vehicles located in a specific location by setting their battery to 100% and updating their range accordingly.

Parameters

head	A pointer to the head of a linked list of Vehicle structures.
location	The location where the vehicles are currently parked or located.

Returns

a pointer to the head of the linked list of vehicles.

5.1.3.10 clientsMain()

```
void clientsMain ( )
```

5.1.3.11 clrbuffer()

```
void clrbuffer ( )
```

It clears the input buffer

5.1.3.12 clrscr()

```
void clrscr ( )
```

It clears the screen

5.1.3.13 collect()

```
Collection * collect (
            Collection * head,
            Vehicle * headVehicles,
            Location * headLocations,
            char startLocation[],
            int manager )
```

The function collects data from a given starting location by visiting adjacent locations using vehicles and saves the collected data.

Parameters

head	A pointer to the head of a linked list of Collection structs.
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headLocations	A pointer to the head of a linked list of Location structs, representing all the locations in the system.
startLocation	The starting location for the collection route.
manager	An integer representing the ID of the manager responsible for the collection.

Returns

a pointer to a Collection, which is the updated head of the linked list of collections.

5.1.3.14 collectionsMain()

```
void collectionsMain (
            int manager )
```

5.1.3.15 copyLinkedList()

```
Vehicle * copyLinkedList (
            Vehicle * head )
```

It creates a new linked list, and copies the contents of the original linked list into the new linked list

Parameters

head	The head of the linked list

Returns

The head of the copied linked list.

5.1.3.16 createEdge()

```
Location * createEdge (
            Location * head,
            char origin[],
            char destination[],
            float distance )
```

The function creates an edge between two locations in a graph data structure.

Parameters

head	A pointer to the head of the linked list of locations.
origin	A string representing the ID of the origin location for the edge being created.
destination	The name or identifier of the location that is being connected to the origin location by the new edge.
distance	distance is a float variable that represents the distance between two locations in a graph. It is used in the function createEdge() to create a new adjacent node between two existing locations.

Returns

a pointer to the head of the Location linked list.

5.1.3.17 createLocation()

```
Location * createLocation (
            Location * head,
            char id[],
            char name[] )
```

The function creates a new location and adds it to the linked list of locations if it does not already exist.

Parameters

head	A pointer to the head of a linked list of Location structs.
id	A character array representing the unique identifier of the location being created.
name	The name of the location that we want to create.

Returns

a pointer to the head of the linked list of locations.

5.1.3.18 currentRide()

```
int currentRide (
            Ride * head,
            int id )
```

It returns the id of the ride that the client is currently on, or -1 if the client is not on a ride

Parameters

head	The head of the linked list
id	The id of the client

Returns

The id of the ride that the client is currently on.

5.1.3.19 editBalance()

```
void editBalance (
            Client * head,
```

```
            int id,
            float balance )
```

It loops through the linked list until it finds the client with the matching id, then it sets the balance to the new balance

Parameters

head	The head of the linked list
id	The id of the client to edit
balance	The new balance

5.1.3.20 editClient()

```
void editClient (
            Client * head,
            int id,
            char username[],
            char password[],
            char name[],
            int nif,
            char location[] )
```

It edits a client's information

Parameters

head	The head of the linked list
id	The id of the client to edit
username	The username of the client
password	The password of the client
name	The name of the client
nif	The tax identification number of the client
location	The location of the client

5.1.3.21 editManager()

```
void editManager (
            Manager * head,
            int id,
            char username[],
            char password[],
            char name[] )
```

It's a function that edits a manager's information

Parameters

head	The head of the linked list
id	The id of the manager to edit
username	The username
password	The password
name	The name

5.1.3.22 editVehicle()

```
void editVehicle (
            Vehicle * head,
            Type * headTypes,
            int id,
            int type,
            float battery,
            float range,
            char location[] )
```

It edits a vehicle's information

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list
id	The id of the vehicle to edit
type	The type of vehicle
battery	The battery of the vehicle
range	The range of the vehicle
location	The location of the vehicle

5.1.3.23 encrypt()

```
void encrypt (
            char password[] )
```

It takes a string, and adds a key to each character in the string.

The key is 18445, but it's multiplied by 4 if the character is in an even position, and multiplied by 2 if the character is in an odd position.

The key is then added to the character.

The result is stored in the same position in the string.

The function returns nothing.

Parameters

password	The password to be encrypted.

5.1.3.24 endRide()

```
void endRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            Location * headLocations,
            int id,
            char endLocation[] )
```

It takes a ride, a vehicle, a type, a client, an id, and an end location, and then it sets the end time, end location, cost, distance, and range of the ride

Parameters

head	The head of the linked list
headVehicles	Pointer to the first vehicle in the linked list
headTypes	Pointer to the first type of vehicle in the linked list
headClients	Pointer to the first client in the linked list
id	The id of the ride
endLocation	The end location of the ride

5.1.3.25 enterToContinue()

```
void enterToContinue ( )
```

It clears the buffer and prints a message to the user, then waits for the user to press a key

5.1.3.26 existClient()

```
int existClient (
            Client * head,
            int id )
```

It checks if a client with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the client

Returns

1 if the client exists in the list, otherwise it returns 0.

5.1.3.27 existClientUsername()

```
int existClientUsername (
            Client * head,
            char username[] )
```

It returns 1 if the username exists in the linked list, otherwise it returns 0

Parameters

head	The head of the linked list
username	The username

Returns

1 if the username exists in the list, otherwise it returns 0.

5.1.3.28 existLocation()

```
int existLocation (
            Location * head,
            char id[] )
```

The function checks if a given location ID exists in a linked list of locations.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The parameter "id" is a character array that represents the ID of a location. It is used to search for a location in a linked list of locations.

Returns

The function `existLocation` returns an integer value of 1 if a location with the given `id` exists in the linked list starting from the `head` node, and 0 otherwise.

5.1.3.29 existManager()

```
int existManager (
            Manager * head,
            int id )
```

It checks if a manager with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the manager

Returns

1 if the manager exists in the list, otherwise it returns 0.

5.1.3.30 existManagerUsername()

```
int existManagerUsername (
            Manager * head,
            char username[] )
```

It returns 1 if the username exists in the linked list, otherwise it returns 0

Parameters

head	The head of the linked list
username	The username

Returns

1 if the username exists in the list, otherwise it returns 0.

5.1.3.31 existType()

```
int existType (
            Type * head,
            int id )
```

It checks if a type with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the type

Returns

1 if the type exists in the list, otherwise it returns 0.

5.1.3.32 existVehicle()

```
int existVehicle (
            Vehicle * head,
            int id )
```

It returns 1 if the vehicle with the given id exists in the list, otherwise it returns 0

Parameters

head	The head of the linked list
id	The id of the vehicle to be added

Returns

1 if the vehicle exists in the list, otherwise it returns 0.

5.1.3.33 getClientLocation()

```
char * getClientLocation (
            Client * head,
            int id )
```

The function returns the location of a client with a given ID, or "*********" if the client is not found.

Parameters

head	A pointer to the head of a linked list of Client structs.
id	The parameter "id" is an integer representing the unique identifier of a client.

Returns

The function `getClientLocation` returns a `char*` which is either the location of the client with the given `id` or the string `"*********"` if the client with the given `id` is not found in the linked list.

5.1.3.34 getClientName()

```
char * getClientName (
            Client * head,
            int id )
```

It returns the name of the client with the given id, or "*********" if the client doesn't exist

Parameters

head	The head of the linked list
id	The id of the client you want to get the name of

Returns

The name of the client with the given id.

5.1.3.35 getClientUsername()

```
char * getClientUsername (
            Client * head,
            int id )
```

Get the username of the client with the given id.

Parameters

head	The head of the linked list
id	The id of the client you want to get the username of

Returns

The username of the client with the given id.

5.1.3.36 getDistance()

```
float getDistance (
            Location * head,
            char origin[],
            char destination[] )
```

The function calculates the distance between two locations in a graph.

Parameters

head	a pointer to the head of a linked list of Location structs
origin	A string representing the ID of the starting location.
destination	The destination parameter is a character array that represents the ID of the location to which the distance is being calculated.

Returns

a float value which represents the distance between two locations. If either the origin or destination location does not exist in the linked list of locations, the function returns -1. If the origin and destination are the same location, the function returns 0.

5.1.3.37 getLocationName()

```
char * getLocationName (
            Location * head,
            char id[] )
```

The function returns the name of a location given its ID, or "*********" if the ID is not found.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The id parameter is a character array that represents the unique identifier of a location.

Returns

If a location with the given id is found in the linked list, its name is returned as a character pointer. If no location with the given id is found, the string "*********" is returned.

5.1.3.38 getManagerName()

```
char * getManagerName (
            Manager * head,
            int id )
```

It returns the name of the manager with the given id, or "*********" if no manager with that id exists

Parameters

head	The head of the linked list
id	The id of the manager you want to get the name of

Returns

The name of the manager with the given id.

5.1.3.39 getTypeCost()

```
float getTypeCost (
            Type * head,
            int id )
```

It returns the cost of a type with a given id

Parameters

head	The head of the linked list
id	The id of the type you want to get the cost of.

Returns

The cost of the type with the given id.

5.1.3.40 getTypeName()

```
char * getTypeName (
            Type * head,
            int id )
```

It returns the name of the type with the given id, or "*********" if the type doesn't exist

Parameters

head	The head of the linked list
id	The id of the type you want to get the name of.

Returns

The name of the type with the given id.

5.1.3.41 getVehicleBattery()

```
float getVehicleBattery (
            Vehicle * head,
            int id )
```

The function returns the battery level of a vehicle with a given ID from a linked list of vehicles.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

The function `getVehicleBattery` returns a float value representing the battery level of a vehicle with the given `id`. If a vehicle with the given `id` is found in the linked list pointed to by `head`, the function returns the battery level of that vehicle. If no vehicle with the given `id` is found, the function returns -1.

5.1.3.42 getVehicleCost()

```
float getVehicleCost (
            Vehicle * head,
            Type * headTypes,
            int id )
```

It loops through the linked list of vehicles, and if the vehicle's id matches the id passed in, it returns the cost of the vehicle's type

Parameters

head	The head of the linked list of vehicles
headTypes	The head of the linked list of types
id	The id of the vehicle you want to get the cost of.

Returns

The cost of the vehicle.

5.1.3.43 getVehicleLocation()

```
char * getVehicleLocation (
            Vehicle * head,
            int id )
```

The function returns the location of a vehicle with a given ID from a linked list of vehicles.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

If a vehicle with the given ID is found in the linked list, the function returns the location of that vehicle as a string. If no vehicle with the given ID is found, the function returns the string "*********".

5.1.3.44 getVehicleTypeName()

```
char * getVehicleTypeName (
            Vehicle * head,
            Type * headTypes,
            int id )
```

The function returns the name of a vehicle type given its ID, by iterating through a linked list of vehicles and using a separate linked list of types.

Parameters

head	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs.
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

a string that represents the name of the vehicle type associated with the given ID. If the ID is not found in the linked list of vehicles, the function returns a string of asterisks.

5.1.3.45 hasBalance()

```
int hasBalance (
            Client * head,
            int id )
```

If the client with the given id has a balance greater than 0, return 1, otherwise return 0

Parameters

head	The head of the linked list
id	The id of the client

Returns

The value of the boolean expression.

5.1.3.46 insertClient()

```
Client * insertClient (
            Client * head,
            int id,
            char username[],
            char password[],
```

```
            char name[],
            int nif,
            char location[],
            float balance,
            int available )
```

It inserts a new client at the end of the list

Parameters

head	The head of the linked list
id	The id of the client
username	The username of the client
password	The password of the client
name	The name of the client
nif	The tax identification number of the client
location	The location of the client
balance	The balance of the client
available	0 = not available, 1 = available

Returns

The head of the list.

5.1.3.47 insertCollected()

```
Collection * insertCollected (
            Collection * head,
            int id,
            char location[],
            int vehicle )
```

The function inserts a new vehicle ID into a specific location's collection within a given collection list.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	The ID of the collection to which the new collected item will be added.
location	A string representing the location where an item was collected.
vehicle	The parameter "vehicle" is an integer representing the ID of the vehicle that collected the item at the specified location.

Returns

a pointer to the head of the Collection linked list.

5.1.3.48 insertCollection()

```
Collection * insertCollection (
             Collection * head,
             int id,
             char startLocation[],
             time_t datetime,
             int manager )
```

The function inserts a new collection into a linked list of collections.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	an integer representing the unique identifier of the collection
startLocation	A character array that represents the starting location of the collection.
datetime	The datetime parameter is a variable of type time_t, which represents the date and time of the collection. It is likely stored as a Unix timestamp, which is the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC.
manager	The parameter "manager" is an integer that represents the ID of the user who is managing the collection.

Returns

a pointer to the head of the linked list of collections.

5.1.3.49 insertManager()

```
Manager * insertManager (
             Manager * head,
             int id,
             char username[],
             char password[],
             char name[] )
```

It inserts a new manager at the end of the list

Parameters

head	The head of the linked list
id	The id
username	The username
password	The password
name	The name

Returns

The head of the list.

5.1.3.50 insertPoint()

```
Collection * insertPoint (
            Collection * head,
            int id,
            char location[] )
```

The function inserts a new point into a collection with a given ID and location.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	The ID of the collection where the new point will be inserted.
location	The location parameter is a string that represents the ID or name of a point that is being inserted into a collection.

Returns

a pointer to a Collection, which is the head of the linked list.

5.1.3.51 insertRide()

```
Ride * insertRide (
            Ride * head,
            int id,
            int vehicle,
            int client,
            int startTime,
            int endTime,
            char startLocation[],
            char endLocation[],
            float cost,
            float distance )
```

It inserts a new ride into the linked list of rides

Parameters

head	The head of the linked list
id	The id of the ride
vehicle	The id of the vehicle
client	The id of the client
startTime	The start time of the ride
endTime	The end time of the ride
startLocation	The start location of the ride
endLocation	The end location of the ride
cost	The cost of the ride
distance	The distance of the ride

Returns

The head of the list.

5.1.3.52 insertType()

```
Type * insertType (
            Type * head,
            int id,
            char name[],
            float cost )
```

It inserts a new client at the end of the list

Parameters

head	The head of the linked list
id	The id of the type of vehicle
name	The name of the type of vehicle
cost	The cost of the type of vehicle

Returns

The head of the list.

5.1.3.53 insertVehicle()

```
Vehicle * insertVehicle (
            Vehicle * head,
            int id,
            int type,
            float battery,
            float range,
            int available,
            char location[] )
```

It inserts a new vehicle at the end of the list

Parameters

head	The head of the linked list
id	The id of the vehicle
type	The type of the vehicle
battery	The battery of the vehicle
range	The range of the vehicle
available	0 = not available, 1 = available
location	The location of the vehicle

Returns

The head of the list.

5.1.3.54 insertVisited()

```
Visited * insertVisited (
            Visited * head,
            char location[] )
```

The function inserts a new visited location into a linked list.

Parameters

head	a pointer to the head of a linked list of Visited nodes.
location	a string representing the ID of a location that the user has visited.

Returns

a pointer to the head of the linked list of visited locations.

5.1.3.55 isClientAvailable()

```
int isClientAvailable (
            Client * head,
            int id )
```

It checks if a client is available

Parameters

head	The head of the linked list
id	The id of the client

Returns

The value of the head->available variable.

5.1.3.56 isVehicleAvailable()

```
int isVehicleAvailable (
            Vehicle * head,
            int id )
```

It checks if a vehicle is available

Parameters

head	The head of the linked list
id	The id of the vehicle to check

Returns

1 if the vehicle is available, otherwise it returns 0.

5.1.3.57 isVehicleCharged()

```
int isVehicleCharged (
            Vehicle * head,
            int id )
```

It checks if the vehicle is charged and has a range greater than 0

Parameters

head	The head of the linked list
id	The id of the vehicle to check

Returns

1 if the vehicle has any battery, otherwise it returns 0.

5.1.3.58 isVisited()

```
int isVisited (
            Visited * head,
            char location[] )
```

The function checks if a given location has been visited before by searching through a linked list of visited locations.

Parameters

head	a pointer to the head of a linked list of Visited nodes
location	A character array representing the ID of a location that we want to check if it has been visited before.

Returns

an integer value of either 1 or 0. The value 1 indicates that the location passed as an argument has been visited before and is present in the linked list pointed to by the head parameter. The value 0 indicates that the location has not been visited before and is not present in the linked list.

5.1.3.59 listAdjacents()

```
void listAdjacents (
            Location * head,
            char id[] )
```

The function lists the adjacent locations and their distances from a given location.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The id parameter is a string that represents the id of a location for which we want to list its adjacent locations.

5.1.3.60 listClient()

```
int listClient (
            Client * head,
            int id )
```

It prints the client's information if the client's id matches the id passed as an argument

Parameters

head	The head of the linked list
id	The id of the client

Returns

The number of clients with the same id.

5.1.3.61 listClients()

```
int listClients (
            Client * head )
```

It prints the contents of a linked list of clients

Parameters

head	The head of the linked list

Returns

The number of clients in the list.

5.1.3.62 listCollections()

```
void listCollections (
            Collection * head,
            Vehicle * headVehicles,
            Type * headTypes )
```

The function lists the collections, their start location, date and time, and the collected vehicles at each collection point.

Parameters

head	A pointer to the head of a linked list of Collection structs.
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs.

Returns

The function does not return anything, it only prints information about the collections and their points of collection.

5.1.3.63 listGraph()

```
void listGraph (
            Location * head )
```

The function prints out a list of locations and their adjacent locations with their respective distances.

Parameters

head	The head pointer of a linked list of Location structs.

5.1.3.64 listLatestCollection()

```
void listLatestCollection (
            Collection * head,
            Vehicle * headVehicles,
            Type * headTypes )
```

The function lists the details of the latest collection, including the start location, date and time, and the collected vehicles at each point of collection.

Parameters

head	a pointer to the head of a linked list of Collection structs
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs, which contain information about the types of vehicles available in the system.

Returns

The function does not return anything, it only prints information about the latest collection.

5.1.3.65 listManagers()

```
int listManagers (
            Manager * head )
```

It prints the id, name, and username of each manager in the list

Parameters

head	The head of the linked list

Returns

The number of managers in the list.

5.1.3.66 listRides()

```
int listRides (
            Ride * head,
            Client * headClients )
```

It prints the list of rides

Parameters

head	The head of the linked list
headClients	Pointer to the first client in the linked list

Returns

The number of rides in the list.

5.1.3.67 listRidesClient()

```
int listRidesClient (
            Ride * head,
            Client * headClients,
            int id )
```

It prints out the rides of a client

Parameters

head	The head of the linked list
headClients	Pointer to the first node of the clients linked list
id	The id of the client

Returns

The number of rides that the client has.

5.1.3.68 listTypes()

```
int listTypes (
            Type * head )
```

It prints the contents of a linked list of types

Parameters

head	The head of the linked list

Returns

The number of items in the list.

5.1.3.69 listVehicles()

```
int listVehicles (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It prints a list of vehicles

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list

Returns

The number of vehicles in the list.

5.1.3.70 listVehiclesByBattery()

```
int listVehiclesByBattery (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function sorts a linked list of vehicles by their battery level and then lists them.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	A pointer to the head of a linked list of Type structs, which contain information about the types of vehicles (e.g. electric, hybrid, gas).
headLocations	A pointer to the head of a linked list of Location structs.
location	The parameter "location" is a string that represents the location where the vehicles are located. It is used as a filter to only list the vehicles that are located in that specific location.

Returns

the result of calling the function `listVehicles` with the sorted linked list of vehicles as its first argument, and the other arguments passed to the function as well.

5.1.3.71 listVehiclesByBatteryHalfCharged()

```
int listVehiclesByBatteryHalfCharged (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function filters vehicles with battery levels below 50% and lists them by location.

Parameters

head	a pointer to the head of a linked list of Vehicle structs

Parameters

headTypes	A pointer to the head of the linked list of vehicle types.
headLocations	A pointer to the head of a linked list of Location structs, which contains information about the locations of vehicles.
location	The parameter "location" is a string that represents the name of a location. It is used as a filter to list only the vehicles that are located in that specific location.

Returns

the result of calling the function `listVehiclesByBattery()` with the filtered list of vehicles as the first argument, along with the other arguments passed to the function.

5.1.3.72 listVehiclesByDistance()

```
int listVehiclesByDistance (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function lists vehicles in ascending order of distance from a given location, with ties broken by range.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	a pointer to the head of a linked list of Type structs
headLocations	a pointer to the head of a linked list of Location structs
location	A string representing the location for which the vehicles need to be listed in order of increasing distance.

Returns

an integer value, which is the result of calling the function `listVehicles` with the sorted linked list of vehicles as its first argument, and the other linked lists as the remaining arguments.

5.1.3.73 listVehiclesByRange()

```
int listVehiclesByRange (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It sorts the linked list by range, then lists the vehicles

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list

Returns

The return value is the result of the function listVehicles.

5.1.3.74 listVehiclesByTypeInRadius()

```
int listVehiclesByTypeInRadius (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            int type,
            char location[],
            float radius )
```

This function filters a linked list of vehicles by type and location within a certain radius and then lists them by distance.

Parameters

head	A pointer to the head of a linked list of Vehicle structs.
headTypes	It is a pointer to the head of a linked list of Type structs.
headLocations	A pointer to the head of a linked list of Location structs, which contains information about the locations of vehicles.
type	an integer representing the type of vehicle to filter by. If set to 0, all types of vehicles will be included in the result.
location	The location parameter is a string that represents the location from which the distance to the vehicles will be calculated.
radius	The radius is a float value that represents the maximum distance from a given location within which vehicles of a certain type should be listed.

Returns

the result of calling the function `listVehiclesByDistance` with the filtered list of vehicles as the first argument, and the head of the types and locations lists and the specified location as the remaining arguments.

5.1.3.75 listVehiclesInLocation()

```
int listVehiclesInLocation (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It filters the linked list by location, then lists the vehicles sorted by range

Parameters

head	pointer to the first element of the linked list
headTypes	a linked list of types
location	The location of the vehicle

Returns

The return value is the number of vehicles that were listed.

5.1.3.76 listVehiclesInRadius()

```
int listVehiclesInRadius (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[],
            float radius )
```

The function lists all vehicles within a certain radius of a given location.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	A pointer to the head of a linked list of vehicle types.
headLocations	A linked list of Location structs containing information about the locations of vehicles.
location	The location parameter is a string that represents the starting location from which the distance to each vehicle's location will be calculated.
radius	The radius is a float value that represents the maximum distance from a given location within which vehicles should be listed.

Returns

the result of calling the function `listVehiclesByDistance` with the filtered list of vehicles as the first argument, and the head of the types and locations linked lists, as well as a location string, as the remaining arguments.

5.1.3.77 loadCollections()

```
Collection * loadCollections ( )
```

This function loads collections from a binary file and returns a pointer to the head of the linked list.

Returns

a pointer to the head of a linked list of Collection structs.

5.1.3.78 locationsMain()

```
void locationsMain ( )
```

5.1.3.79 managersMain()

```
void managersMain ( )
```

5.1.3.80 menuApp()

```
void menuApp ( )
```

5.1.3.81 menuAuth()

```
void menuAuth ( )
```

5.1.3.82 menuAuthClients()

```
void menuAuthClients ( )
```

5.1.3.83 menuAuthManagers()

```
void menuAuthManagers ( )
```

5.1.3.84 menuFooterClients()

```
void menuFooterClients ( )
```

5.1.3.85 menuFooterCollections()

```
void menuFooterCollections ( )
```

5.1.3.86 menuFooterManagers()

```
void menuFooterManagers ( )
```

5.1.3.87 menuFooterRides()

```
void menuFooterRides ( )
```

5.1.3.88 menuFooterVehicles()

```
void menuFooterVehicles ( )
```

5.1.3.89 menuHeaderClient()

```
void menuHeaderClient ( )
```

5.1.3.90 menuHeaderClients()

```
void menuHeaderClients ( )
```

5.1.3.91 menuHeaderManagers()

```
void menuHeaderManagers ( )
```

5.1.3.92 menuHeaderRides()

```
void menuHeaderRides ( )
```

5.1.3.93 menuHeaderRidesClient()

```
void menuHeaderRidesClient ( )
```

5.1.3.94 menuHeaderVehicles()

```
void menuHeaderVehicles ( )
```

5.1.3.95 menuLine()

```
void menuLine ( )
```

5.1.3.96 menuMain()

```
void menuMain ( )
```

5.1.3.97 menuMainClients()

```
void menuMainClients (
             int available )
```

5.1.3.98 menuTitleAddBalance()

```
void menuTitleAddBalance ( )
```

5.1.3.99 menuTitleEditClient()

```
void menuTitleEditClient ( )
```

5.1.3.100 menuTitleEditManager()

```
void menuTitleEditManager ( )
```

5.1.3.101 menuTitleEditVehicle()

```
void menuTitleEditVehicle ( )
```

5.1.3.102 menuTitleInsertClient()

```
void menuTitleInsertClient ( )
```

5.1.3.103 menuTitleInsertManager()

```
void menuTitleInsertManager ( )
```

5.1.3.104 menuTitleInsertVehicle()

```
void menuTitleInsertVehicle ( )
```

5.1.3.105 menuTitleRemoveBalance()

```
void menuTitleRemoveBalance ( )
```

5.1.3.106 menuTitleRemoveClient()

```
void menuTitleRemoveClient ( )
```

5.1.3.107 menuTitleRemoveManager()

```
void menuTitleRemoveManager ( )
```

5.1.3.108 menuTitleRemoveVehicle()

```
void menuTitleRemoveVehicle ( )
```

5.1.3.109 readClients()

```
Client * readClients ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Client struct.

5.1.3.110 readLocations()

```
Location * readLocations ( )
```

The function reads location and edge data from text files and creates a linked list of locations with edges between them.

Returns

a pointer to a Location struct.

5.1.3.111 readManagers()

```
Manager * readManagers ( )
```

It reads a file and creates a linked list of managers

Returns

A pointer to a Manager struct.

5.1.3.112 readRides()

```
Ride * readRides ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Ride struct.

5.1.3.113 readTypes()

```
Type * readTypes ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Type struct.

5.1.3.114 readVehicles()

```
Vehicle * readVehicles ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Vehicle struct.

5.1.3.115 removeBalance()

```
void removeBalance (
            Client * head,
            int id,
            float balance )
```

It removes the balance from the client with the given id

Parameters

head	The head of the linked list
id	The id of the client
balance	The amount of money to be removed from the client's balance

5.1.3.116 removeClient()

```
Client * removeClient (
            Client * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, free it and return the second element. Otherwise, find the element to be removed and free it

Parameters

head	The head of the linked list
id	The id of the client to be removed

Returns

The head of the list.

5.1.3.117 removeManager()

```
Manager * removeManager (
            Manager * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, remove it and return the new head. Otherwise, find the element to be removed and remove it

Parameters

head	The head of the linked list
id	The id of the manager to be removed

Returns

The head of the list.

5.1.3.118 removeVehicle()

```
Vehicle * removeVehicle (
            Vehicle * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, free it and return the second element. Otherwise, find the element to be removed and free it

Parameters

head	The head of the linked list
id	The id of the vehicle to be removed

Returns

The head of the list.

5.1.3.119 ridesMain()

```
void ridesMain ( )
```

5.1.3.120 saveClients()

```
int saveClients (
            Client * head )
```

It saves the clients to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, or 0 if it wasn't.

5.1.3.121 saveCollections()

```
int saveCollections (
            Collection * head )
```

The function saves a linked list of collections, along with their points and collected integers, to a binary file.

Parameters

head	A pointer to the head of a linked list of Collection structs.

Returns

an integer value. If the file "collections.bin" is successfully opened for writing, the function returns 1. If the file cannot be opened, the function returns 0.

5.1.3.122 saveManagers()

```
int saveManagers (
            Manager * head )
```

It saves the managers to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, and 0 if it wasn't.

5.1.3.123 saveRides()

```
int saveRides (
            Ride * head )
```

It saves the linked list of rides to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, and 0 if it wasn't.

5.1.3.124 saveTypes()

```
int saveTypes (
            Type * head )
```

It saves the types to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, or 0 if it wasn't.

5.1.3.125 saveVehicles()

```
int saveVehicles (
            Vehicle * head )
```

It saves the vehicles to a file

Parameters

head	The head of the linked list

Returns

1 if the file was successfully saved, and 0 if it was not.

5.1.3.126 showCount()

```
void showCount (
            int count )
```

It prints a message to the user, telling them how many results were found

Parameters

count	The number of results to be shown.

5.1.3.127 showRide()

```
void showRide (
            Ride * head,
            int id )
```

It prints the information of a ride given its id

Parameters

head	The head of the linked list
id	The id of the ride

5.1.3.128 startRide()

```
Ride * startRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            int id,
            int vehicle,
            int client )
```

It takes a ride, a vehicle, a type, a client, and an id, and returns a ride

Parameters

head	The head of the linked list
headVehicles	Pointer to the first vehicle in the linked list
headTypes	Pointer to the first type of vehicle in the linked list
headClients	Pointer to the first client in the linked list
id	The id of the ride
vehicle	The id of the vehicle
client	The id of the client

Returns

The head of the list.

5.1.3.129 updateVehicleLocation()

```
void updateVehicleLocation (
            Vehicle * head,
            int id,
            char location[] )
```

The function updates the location of a vehicle with a given ID in a linked list.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.
location	The parameter "location" is a character array that represents the new location of a vehicle. This function updates the location of a vehicle with the given "id" to the new location provided in the "location" parameter.

5.1.3.130 vehiclesMain()

```
void vehiclesMain ( )
```

5.2 header.h

Go to the documentation of this file.

```
00001 #ifndef HEADER_H_
00002 #define HEADER_H_
00003
00004 #define DATA_DIR "data/"
00005
00006 #define SIZE_USERNAME 40
00007 #define SIZE_PASSWORD 40
00008 #define SIZE_NAME 60
```

```
00009 #define SIZE_LOCATION 60
00010 #define SIZE_TYPE 5
00011 #define SIZE_BATTERY 15
00012 #define SIZE_RANGE 15
00013 #define SIZE_NIF 15
00014 #define SIZE_DATETIME 20
00015
00016 #define HQ "tatica.ideia.morno"
00017
00018 #define RED     "\x1B[31m"
00019 #define GREEN   "\x1B[32m"
00020 #define YELLOW  "\x1B[33m"
00021 #define BLUE    "\x1B[34m"
00022 #define MAGENTA "\x1B[35m"
00023 #define CYAN    "\x1B[36m"
00024 #define WHITE   "\x1B[37m"
00025 #define RESET   "\x1B[0m"
00026
00027 #include <time.h>
00028
00029 typedef struct integer {
00030     int id;
00031     struct integer* next;
00032
00033 } Integer;
00034
00035 typedef struct type {
00036     int id; // 1 - Trotinete; 2 - Bicicleta
00037     char name[SIZE_NAME];
00038     float cost;
00039     struct type* next;
00040
00041 } Type;
00042
00043 typedef struct vehicle {
00044     int id;
00045     int type;
00046     float battery;
00047     float range;
00048     char location[SIZE_LOCATION];
00049     int available;
00050     struct vehicle* next;
00051
00052 } Vehicle;
00053
00054 typedef struct client {
00055     int id;
00056     char username[SIZE_USERNAME];
00057     char password[SIZE_PASSWORD];
00058     char name[SIZE_NAME];
00059     int nif;
00060     char location[SIZE_LOCATION];
00061     float balance;
00062     int available;
00063     struct client* next;
00064
00065 } Client;
00066
00067 typedef struct manager {
00068     int id;
00069     char username[SIZE_USERNAME];
00070     char password[SIZE_PASSWORD];
00071     char name[SIZE_NAME];
00072     struct manager* next;
00073
00074 } Manager;
00075
00076 typedef struct ride {
00077     int id;
00078     int vehicle;
00079     int client;
00080     time_t startTime;
00081     time_t endTime;
00082     char startLocation[SIZE_LOCATION];
00083     char endLocation[SIZE_LOCATION];
00084     float cost;
00085     float distance;
00086     struct ride* next;
00087
00088 } Ride;
00089
00090 typedef struct point {
00091     char id[SIZE_LOCATION];
00092     struct integer* collected;
00093     struct point* next;
00094
00095 } Point;
```

```
00096
00097 typedef struct collection {
00098     int id;
00099     char startLocation[SIZE_LOCATION];
00100     time_t datetime;
00101     int manager;
00102     struct point* points;
00103     struct collection* next;
00104
00105 } Collection;
00106
00107 typedef struct adjacent {
00108     char id[SIZE_LOCATION];
00109     float distance;
00110     struct adjacent* next;
00111
00112 } Adjacent;
00113
00114 typedef struct location {
00115     char id[SIZE_LOCATION];
00116     char name[SIZE_LOCATION];
00117     struct adjacent* adjacents;
00118     struct location* next;
00119
00120 } Location;
00121
00122 typedef struct visited {
00123     char id[SIZE_LOCATION];
00124     struct visited* next;
00125
00126 } Visited;
00127
00128 /*Rides*/
00129 void ridesMain();
00130 Ride* insertRide(Ride* head, int id, int vehicle, int client, int startTime, int endTime, char
      startLocation[], char endLocation[], float cost, float distance);
00131 Ride* startRide(Ride* head, Vehicle* headVehicles, Type* headTypes, Client* headClients, int id, int
      vehicle, int client);
00132 void endRide(Ride* head, Vehicle* headVehicles, Type* headTypes, Client* headClients, Location*
      headLocations, int id, char endLocation[]);
00133 int listRides(Ride* head, Client* headClients);
00134 int listRidesClient(Ride* head, Client* headClients, int id);
00135 int assignRideId(Ride* head);
00136 int currentRide(Ride* head, int id);
00137 void showRide(Ride* head, int id);
00138 int saveRides(Ride* head);
00139 Ride* readRides();
00140
00141 /*Vehicles*/
00142 void vehiclesMain();
00143 Vehicle* insertVehicle(Vehicle* head, int id, int type, float battery, float range, int available,
      char location[]);
00144 Vehicle* removeVehicle(Vehicle* head, int id);
00145 void editVehicle(Vehicle* head, Type* headTypes, int id, int type, float battery, float range, char
      location[]);
00146 int listVehicles(Vehicle* head, Type* headTypes, Location* headLocations, char location[]);
00147 int listVehiclesByRange(Vehicle* head, Type* headTypes, Location* headLocations, char location[]);
00148 int listVehiclesByBattery(Vehicle* head, Type* headTypes, Location* headLocations, char location[]);
00149 int listVehiclesInLocation(Vehicle* head, Type* headTypes, Location* headLocations, char location[]);
00150 int listVehiclesByDistance(Vehicle* head, Type* headTypes, Location* headLocations, char location[]);
00151 int listVehiclesInRadius(Vehicle* head, Type* headTypes, Location* headLocations, char location[],
      float radius);
00152 int listVehiclesByTypeInRadius(Vehicle* head, Type* headTypes, Location* headLocations, int type, char
      location[], float radius);
00153 int listVehiclesByBatteryHalfCharged(Vehicle* head, Type* headTypes, Location* headLocations, char
      location[]);
00154 int existVehicle(Vehicle* head, int id);
00155 int assignVehicleId(Vehicle* head);
00156 int isVehicleAvailable(Vehicle* head, int id);
00157 int isVehicleCharged(Vehicle* head, int id);
00158 void updateVehicleLocation(Vehicle* head, int id, char location[]);
00159 Vehicle* chargeVehicles(Vehicle* head, char location[]);
00160 Vehicle* copyLinkedList(Vehicle* head);
00161 int saveVehicles(Vehicle* head);
00162 Vehicle* readVehicles();
00163 char* getVehicleTypeName(Vehicle* head, Type* headTypes, int id);
00164 float getVehicleBattery(Vehicle* head, int id);
00165 char* getVehicleLocation(Vehicle* head, int id);
00166 float getVehicleCost(Vehicle* head, Type* headTypes, int id);
00167 float getTypeCost(Type* head, int id);
00168 char* getTypeName(Type* head, int id);
00169 Type* insertType(Type* head, int id, char name[], float cost);
00170 int listTypes(Type* head);
00171 int existType(Type* head, int id);
00172 int saveTypes(Type* head);
00173 Type* readTypes();
00174
```

```
00175 /*Locations*/
00176 void locationsMain();
00177 Location* createLocation(Location* head, char id[], char name[]);
00178 int existLocation(Location* head, char id[]);
00179 char* getLocationName(Location* head, char id[]);
00180 float getDistance(Location* head, char origin[], char destination[]);
00181 Location* createEdge(Location* head, char origin[], char destination[], float distance);
00182 void listAdjacents(Location* head, char id[]);
00183 void listGraph(Location* head);
00184 Location* readLocations();
00185
00186 /*Collections*/
00187 void collectionsMain(int manager);
00188 Collection* collect(Collection* head, Vehicle* headVehicles, Location* headLocations, char
      startLocation[], int manager);
00189 Collection* insertCollection(Collection* head, int id, char startLocation[], time_t datetime, int
      manager);
00190 Collection* insertPoint(Collection* head, int id, char location[]);
00191 Collection* insertCollected(Collection* head, int id, char location[], int vehicle);
00192 Visited* insertVisited(Visited* head, char location[]);
00193 void listCollections(Collection* head, Vehicle* headVehicles, Type* headTypes);
00194 void listLatestCollection(Collection* head, Vehicle* headVehicles, Type* headTypes);
00195 int assignCollectionId(Collection* head);
00196 int isVisited(Visited* head, char location[]);
00197 int saveCollections(Collection* head);
00198 Collection* loadCollections();
00199
00200 /*Clients*/
00201 void clientsMain();
00202 Client* insertClient(Client* head, int id, char username[], char password[], char name[], int nif,
      char location[], float balance, int available);
00203 Client* removeClient(Client* head, int id);
00204 void editClient(Client* head, int id, char username[], char password[], char name[], int nif, char
      location[]);
00205 int listClients(Client* head);
00206 int listClient(Client* head, int id);
00207 char* getClientName(Client* head, int id);
00208 char* getClientUsername(Client* head, int id);
00209 char* getClientLocation(Client* head, int id);
00210 int existClientUsername(Client* head, char username[]);
00211 int existClient(Client* head, int id);
00212 int assignClientId(Client* head);
00213 int isClientAvailable(Client* head, int id);
00214 void addBalance(Client* head, int id, float balance);
00215 void removeBalance(Client* head, int id, float balance);
00216 void editBalance(Client* head, int id, float balance);
00217 int hasBalance(Client* head, int id);
00218 int saveClients(Client* head);
00219 Client* readClients();
00220
00221 /*Managers*/
00222 void managersMain();
00223 Manager* insertManager(Manager* head, int id, char username[], char password[], char name[]);
00224 Manager* removeManager(Manager* head, int id);
00225 void editManager(Manager* head, int id, char username[], char password[], char name[]);
00226 int listManagers(Manager* head);
00227 char* getManagerName(Manager* head, int id);
00228 int existManagerUsername(Manager* head, char username[]);
00229 int existManager(Manager* head, int id);
00230 int assignManagerId(Manager* head);
00231 int saveManagers(Manager* head);
00232 Manager* readManagers();
00233
00234 /*Auth*/
00235 void encrypt(char password[]);
00236 int authClient(Client* head, char username[], char password[]);
00237 int authManager(Manager* head, char username[], char password[]);
00238
00239 /*Menus*/
00240 void menuApp();
00241 void menuMain();
00242 void menuMainClients(int available);
00243 void menuLine();
00244 void menuAuth();
00245 void menuAuthClients();
00246 void menuAuthManagers();
00247 void menuHeaderRides();
00248 void menuHeaderRidesClient();
00249 void menuHeaderVehicles();
00250 void menuHeaderClients();
00251 void menuHeaderClient();
00252 void menuHeaderManagers();
00253 void menuFooterRides();
00254 void menuFooterVehicles();
00255 void menuFooterCollections();
00256 void menuFooterClients();
00257 void menuFooterManagers();
```

```
00258 void menuTitleInsertVehicle();
00259 void menuTitleRemoveVehicle();
00260 void menuTitleEditVehicle();
00261 void menuTitleInsertClient();
00262 void menuTitleRemoveClient();
00263 void menuTitleEditClient();
00264 void menuTitleAddBalance();
00265 void menuTitleRemoveBalance();
00266 void menuTitleInsertManager();
00267 void menuTitleRemoveManager();
00268 void menuTitleEditManager();
00269
00270 /*Utilities*/
00271 void clrscr();
00272 void clrbuffer();
00273 void enterToContinue();
00274 void showCount(int count);
00275
00276 #endif
```

5.3 README.md File Reference

5.4 auth.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void encrypt (char password[ ])
- int authClient (Client *head, char username[ ], char password[ ])
- int authManager (Manager *head, char username[ ], char password[ ])

5.4.1 Function Documentation

5.4.1.1 authClient()

```
int authClient (
            Client * head,
            char username[ ],
            char password[ ] )
```

It takes a pointer to the head of a linked list of clients, a username and a password, encrypts the password, and returns the id of the client if the username and password match, or 0 if they don't

Parameters

head	The head of the linked list
username	the username of the client
password	the password to be encrypted

Returns

The ID of the client.

5.4.1.2 authManager()

```
int authManager (
            Manager * head,
            char username[],
            char password[] )
```

It takes a pointer to a linked list of managers, a username and a password, encrypts the password, and then compares the username and password to the username and password of each manager in the linked list. If it finds a match, it returns the manager's ID. If it doesn't find a match, it returns 0

Parameters

head	pointer to the first node of the linked list
username	the username of the manager
password	the password to be encrypted

Returns

The ID of the manager.

5.4.1.3 encrypt()

```
void encrypt (
            char password[] )
```

It takes a string, and adds a key to each character in the string.

The key is 18445, but it's multiplied by 4 if the character is in an even position, and multiplied by 2 if the character is in an odd position.

The key is then added to the character.

The result is stored in the same position in the string.

The function returns nothing.

Parameters

password	The password to be encrypted.

5.5 clients.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void clientsMain ()
- Client * insertClient (Client *head, int id, char username[], char password[], char name[], int nif, char location[], float balance, int available)
- Client * removeClient (Client *head, int id)
- void editClient (Client *head, int id, char username[], char password[], char name[], int nif, char location[])
- int listClients (Client *head)
- int listClient (Client *head, int id)
- char * getClientName (Client *head, int id)
- char * getClientUsername (Client *head, int id)
- char * getClientLocation (Client *head, int id)
- int existClientUsername (Client *head, char username[])
- int existClient (Client *head, int id)
- int assignClientId (Client *head)
- int isClientAvailable (Client *head, int id)
- void addBalance (Client *head, int id, float balance)
- void removeBalance (Client *head, int id, float balance)
- void editBalance (Client *head, int id, float balance)
- int hasBalance (Client *head, int id)
- int saveClients (Client *head)
- Client * readClients ()

5.5.1 Function Documentation

5.5.1.1 addBalance()

```
void addBalance (
            Client * head,
            int id,
            float balance )
```

It adds the balance to the client with the given id

Parameters

head	The head of the linked list
id	The id of the client
balance	The amount of money to add to the client's balance

5.5.1.2 assignClientId()

```
int assignClientId (
            Client * head )
```

It returns the next available client id

Parameters

head	The head of the linked list

Returns

The next available client ID.

5.5.1.3 clientsMain()

```
void clientsMain ( )
```

5.5.1.4 editBalance()

```
void editBalance (
            Client * head,
            int id,
            float balance )
```

It loops through the linked list until it finds the client with the matching id, then it sets the balance to the new balance

Parameters

head	The head of the linked list
id	The id of the client to edit
balance	The new balance

5.5.1.5 editClient()

```
void editClient (
            Client * head,
            int id,
```

```
            char username[],
            char password[],
            char name[],
            int nif,
            char location[] )
```

It edits a client's information

Parameters

head	The head of the linked list
id	The id of the client to edit
username	The username of the client
password	The password of the client
name	The name of the client
nif	The tax identification number of the client
location	The location of the client

5.5.1.6 existClient()

```
int existClient (
            Client * head,
            int id )
```

It checks if a client with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the client

Returns

1 if the client exists in the list, otherwise it returns 0.

5.5.1.7 existClientUsername()

```
int existClientUsername (
            Client * head,
            char username[] )
```

It returns 1 if the username exists in the linked list, otherwise it returns 0

Parameters

head	The head of the linked list
username	The username

Returns

1 if the username exists in the list, otherwise it returns 0.

5.5.1.8 getClientLocation()

```
char * getClientLocation (
            Client * head,
            int id )
```

The function returns the location of a client with a given ID, or "*********" if the client is not found.

Parameters

head	A pointer to the head of a linked list of Client structs.
id	The parameter "id" is an integer representing the unique identifier of a client.

Returns

The function `getClientLocation` returns a `char*` which is either the location of the client with the given `id` or the string `"*********"` if the client with the given `id` is not found in the linked list.

5.5.1.9 getClientName()

```
char * getClientName (
            Client * head,
            int id )
```

It returns the name of the client with the given id, or "*********" if the client doesn't exist

Parameters

head	The head of the linked list
id	The id of the client you want to get the name of

Returns

The name of the client with the given id.

5.5.1.10 getClientUsername()

```
char * getClientUsername (
            Client * head,
            int id )
```

Get the username of the client with the given id.

Parameters

head	The head of the linked list
id	The id of the client you want to get the username of

Returns

The username of the client with the given id.

5.5.1.11 hasBalance()

```
int hasBalance (
            Client * head,
            int id )
```

If the client with the given id has a balance greater than 0, return 1, otherwise return 0

Parameters

head	The head of the linked list
id	The id of the client

Returns

The value of the boolean expression.

5.5.1.12 insertClient()

```
Client * insertClient (
            Client * head,
            int id,
            char username[],
            char password[],
            char name[],
            int nif,
            char location[],
            float balance,
            int available )
```

It inserts a new client at the end of the list

Parameters

head	The head of the linked list
id	The id of the client
username	The username of the client

Parameters

password	The password of the client
name	The name of the client
nif	The tax identification number of the client
location	The location of the client
balance	The balance of the client
available	0 = not available, 1 = available

Returns

The head of the list.

5.5.1.13 isClientAvailable()

```
int isClientAvailable (
            Client * head,
            int id )
```

It checks if a client is available

Parameters

head	The head of the linked list
id	The id of the client

Returns

The value of the head->available variable.

5.5.1.14 listClient()

```
int listClient (
            Client * head,
            int id )
```

It prints the client's information if the client's id matches the id passed as an argument

Parameters

head	The head of the linked list
id	The id of the client

Returns

The number of clients with the same id.

5.5.1.15 listClients()

```
int listClients (
            Client * head )
```

It prints the contents of a linked list of clients

Parameters

head	The head of the linked list

Returns

The number of clients in the list.

5.5.1.16 readClients()

```
Client * readClients ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Client struct.

5.5.1.17 removeBalance()

```
void removeBalance (
            Client * head,
            int id,
            float balance )
```

It removes the balance from the client with the given id

Parameters

head	The head of the linked list
id	The id of the client
balance	The amount of money to be removed from the client's balance

5.5.1.18 removeClient()

```
Client * removeClient (
            Client * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, free it and return the second element. Otherwise, find the element to be removed and free it

Parameters

head	The head of the linked list
id	The id of the client to be removed

Returns

The head of the list.

5.5.1.19 saveClients()

```
int saveClients (
            Client * head )
```

It saves the clients to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, or 0 if it wasn't.

5.6 collections.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "../inc/header.h"
```

Functions

- void collectionsMain (int manager)
- Collection * collect (Collection *head, Vehicle *headVehicles, Location *headLocations, char startLocation[ ], int manager)
- Collection * insertCollection (Collection *head, int id, char startLocation[ ], time_t datetime, int manager)
- Collection * insertPoint (Collection *head, int id, char location[ ])
- Collection * insertCollected (Collection *head, int id, char location[ ], int vehicle)
- Visited * insertVisited (Visited *head, char location[ ])
- void listCollections (Collection *head, Vehicle *headVehicles, Type *headTypes)
- void listLatestCollection (Collection *head, Vehicle *headVehicles, Type *headTypes)
- int assignCollectionId (Collection *head)
- int isVisited (Visited *head, char location[ ])
- int saveCollections (Collection *head)
- Collection * loadCollections ()

5.6.1 Function Documentation

5.6.1.1 assignCollectionId()

```
int assignCollectionId (
            Collection * head )
```

This function assigns a unique ID to a new element in a linked list.

Parameters

head	A pointer to the head of a linked list of Collection structs.

Returns

an integer value, which is the next available ID for a new collection.

5.6.1.2 collect()

```
Collection * collect (
            Collection * head,
            Vehicle * headVehicles,
            Location * headLocations,
            char startLocation[],
            int manager )
```

The function collects data from a given starting location by visiting adjacent locations using vehicles and saves the collected data.

Parameters

head	A pointer to the head of a linked list of Collection structs.
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headLocations	A pointer to the head of a linked list of Location structs, representing all the locations in the system.
startLocation	The starting location for the collection route.
manager	An integer representing the ID of the manager responsible for the collection.

Returns

a pointer to a Collection, which is the updated head of the linked list of collections.

5.6.1.3 collectionsMain()

```
void collectionsMain (
             int manager )
```

5.6.1.4 insertCollected()

```
Collection * insertCollected (
             Collection * head,
             int id,
             char location[],
             int vehicle )
```

The function inserts a new vehicle ID into a specific location's collection within a given collection list.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	The ID of the collection to which the new collected item will be added.
location	A string representing the location where an item was collected.
vehicle	The parameter "vehicle" is an integer representing the ID of the vehicle that collected the item at the specified location.

Returns

a pointer to the head of the Collection linked list.

5.6.1.5 insertCollection()

```
Collection * insertCollection (
             Collection * head,
             int id,
             char startLocation[],
             time_t datetime,
             int manager )
```

The function inserts a new collection into a linked list of collections.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	an integer representing the unique identifier of the collection
startLocation	A character array that represents the starting location of the collection.
datetime	The datetime parameter is a variable of type time_t, which represents the date and time of the collection. It is likely stored as a Unix timestamp, which is the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC.
manager	The parameter "manager" is an integer that represents the ID of the user who is managing the collection.

Returns

a pointer to the head of the linked list of collections.

5.6.1.6 insertPoint()

```
Collection * insertPoint (
             Collection * head,
             int id,
             char location[] )
```

The function inserts a new point into a collection with a given ID and location.

Parameters

head	A pointer to the head of a linked list of Collection structs.
id	The ID of the collection where the new point will be inserted.
location	The location parameter is a string that represents the ID or name of a point that is being inserted into a collection.

Returns

a pointer to a Collection, which is the head of the linked list.

5.6.1.7 insertVisited()

```
Visited * insertVisited (
            Visited * head,
            char location[] )
```

The function inserts a new visited location into a linked list.

Parameters

head	a pointer to the head of a linked list of Visited nodes.
location	a string representing the ID of a location that the user has visited.

Returns

a pointer to the head of the linked list of visited locations.

5.6.1.8 isVisited()

```
int isVisited (
            Visited * head,
            char location[] )
```

The function checks if a given location has been visited before by searching through a linked list of visited locations.

Parameters

head	a pointer to the head of a linked list of Visited nodes
location	A character array representing the ID of a location that we want to check if it has been visited before.

Returns

an integer value of either 1 or 0. The value 1 indicates that the location passed as an argument has been visited before and is present in the linked list pointed to by the head parameter. The value 0 indicates that the location has not been visited before and is not present in the linked list.

5.6.1.9 listCollections()

```
void listCollections (
            Collection * head,
            Vehicle * headVehicles,
            Type * headTypes )
```

The function lists the collections, their start location, date and time, and the collected vehicles at each collection point.

Parameters

head	A pointer to the head of a linked list of Collection structs.
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs.

Returns

The function does not return anything, it only prints information about the collections and their points of collection.

5.6.1.10 listLatestCollection()

```
void listLatestCollection (
            Collection * head,
            Vehicle * headVehicles,
            Type * headTypes )
```

The function lists the details of the latest collection, including the start location, date and time, and the collected vehicles at each point of collection.

Parameters

head	a pointer to the head of a linked list of Collection structs
headVehicles	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs, which contain information about the types of vehicles available in the system.

Returns

The function does not return anything, it only prints information about the latest collection.

5.6.1.11 loadCollections()

```
Collection * loadCollections ( )
```

This function loads collections from a binary file and returns a pointer to the head of the linked list.

Returns

a pointer to the head of a linked list of Collection structs.

5.6.1.12 saveCollections()

```
int saveCollections (
            Collection * head )
```

The function saves a linked list of collections, along with their points and collected integers, to a binary file.

Parameters

head	A pointer to the head of a linked list of Collection structs.

Returns

an integer value. If the file "collections.bin" is successfully opened for writing, the function returns 1. If the file cannot be opened, the function returns 0.

5.7 locations.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void locationsMain ()
- Location ∗ createLocation (Location ∗head, char id[ ], char name[ ])
- Location ∗ createEdge (Location ∗head, char origin[ ], char destination[ ], float distance)
- int existLocation (Location ∗head, char id[ ])
- char ∗ getLocationName (Location ∗head, char id[ ])
- float getDistance (Location ∗head, char origin[ ], char destination[ ])
- void listAdjacents (Location ∗head, char id[ ])
- void listGraph (Location ∗head)
- Location ∗ readLocations ()

5.7.1 Function Documentation

5.7.1.1 createEdge()

```
Location * createEdge (
            Location * head,
            char origin[],
            char destination[],
            float distance )
```

The function creates an edge between two locations in a graph data structure.

Parameters

head	A pointer to the head of the linked list of locations.
origin	A string representing the ID of the origin location for the edge being created.
destination	The name or identifier of the location that is being connected to the origin location by the new edge.
distance	distance is a float variable that represents the distance between two locations in a graph. It is used in the function createEdge() to create a new adjacent node between two existing locations.

Returns

a pointer to the head of the Location linked list.

5.7.1.2 createLocation()

```
Location * createLocation (
            Location * head,
            char id[],
            char name[] )
```

The function creates a new location and adds it to the linked list of locations if it does not already exist.

Parameters

head	A pointer to the head of a linked list of Location structs.
id	A character array representing the unique identifier of the location being created.
name	The name of the location that we want to create.

Returns

a pointer to the head of the linked list of locations.

5.7.1.3 existLocation()

```
int existLocation (
            Location * head,
            char id[] )
```

The function checks if a given location ID exists in a linked list of locations.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The parameter "id" is a character array that represents the ID of a location. It is used to search for a location in a linked list of locations.

Returns

The function `existLocation` returns an integer value of 1 if a location with the given `id` exists in the linked list starting from the `head` node, and 0 otherwise.

5.7.1.4 getDistance()

```
float getDistance (
            Location * head,
            char origin[],
            char destination[] )
```

The function calculates the distance between two locations in a graph.

Parameters

head	a pointer to the head of a linked list of Location structs
origin	A string representing the ID of the starting location.
destination	The destination parameter is a character array that represents the ID of the location to which the distance is being calculated.

Returns

a float value which represents the distance between two locations. If either the origin or destination location does not exist in the linked list of locations, the function returns -1. If the origin and destination are the same location, the function returns 0.

5.7.1.5 getLocationName()

```
char * getLocationName (
            Location * head,
            char id[] )
```

The function returns the name of a location given its ID, or "*********" if the ID is not found.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The id parameter is a character array that represents the unique identifier of a location.

Returns

If a location with the given id is found in the linked list, its name is returned as a character pointer. If no location with the given id is found, the string "*********" is returned.

5.7.1.6 listAdjacents()

```
void listAdjacents (
            Location * head,
            char id[] )
```

The function lists the adjacent locations and their distances from a given location.

Parameters

head	a pointer to the head of a linked list of Location structs
id	The id parameter is a string that represents the id of a location for which we want to list its adjacent locations.

5.7.1.7 listGraph()

```
void listGraph (
            Location * head )
```

The function prints out a list of locations and their adjacent locations with their respective distances.

Parameters

head	The head pointer of a linked list of Location structs.

5.7.1.8 locationsMain()

```
void locationsMain ( )
```

5.7.1.9 readLocations()

```
Location * readLocations ( )
```

The function reads location and edge data from text files and creates a linked list of locations with edges between them.

Returns

a pointer to a Location struct.

5.8 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "../inc/header.h"
```

Functions

- int main ()

5.8.1 Function Documentation

5.8.1.1 main()

```
int main ( )
```

5.9 managers.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void managersMain ()
- Manager * insertManager (Manager *head, int id, char username[ ], char password[ ], char name[ ])
- Manager * removeManager (Manager *head, int id)
- void editManager (Manager *head, int id, char username[ ], char password[ ], char name[ ])
- int listManagers (Manager *head)
- char * getManagerName (Manager *head, int id)
- int existManagerUsername (Manager *head, char username[ ])
- int existManager (Manager *head, int id)
- int assignManagerId (Manager *head)
- int saveManagers (Manager *head)
- Manager * readManagers ()

5.9.1 Function Documentation

5.9.1.1 assignManagerId()

```
int assignManagerId (
            Manager * head )
```

It returns the next available manager id.

Parameters

head	The head of the linked list

Returns

The id of the last manager in the list.

5.9.1.2 editManager()

```
void editManager (
            Manager * head,
            int id,
            char username[],
            char password[],
            char name[] )
```

It's a function that edits a manager's information

Parameters

head	The head of the linked list
id	The id of the manager to edit
username	The username
password	The password
name	The name

5.9.1.3 existManager()

```
int existManager (
            Manager * head,
            int id )
```

It checks if a manager with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the manager

Returns

1 if the manager exists in the list, otherwise it returns 0.

5.9.1.4 existManagerUsername()

```
int existManagerUsername (
            Manager * head,
            char username[] )
```

It returns 1 if the username exists in the linked list, otherwise it returns 0

Parameters

head	The head of the linked list
username	The username

Returns

1 if the username exists in the list, otherwise it returns 0.

5.9.1.5 getManagerName()

```
char * getManagerName (
            Manager * head,
            int id )
```

It returns the name of the manager with the given id, or "*********" if no manager with that id exists

Parameters

head	The head of the linked list
id	The id of the manager you want to get the name of

Returns

The name of the manager with the given id.

5.9.1.6 insertManager()

```
Manager * insertManager (
            Manager * head,
            int id,
            char username[],
            char password[],
            char name[] )
```

It inserts a new manager at the end of the list

Parameters

head	The head of the linked list
id	The id
username	The username
password	The password
name	The name

Returns

The head of the list.

5.9.1.7 listManagers()

```
int listManagers (
            Manager * head )
```

It prints the id, name, and username of each manager in the list

Parameters

head	The head of the linked list

Returns

The number of managers in the list.

5.9.1.8 managersMain()

```
void managersMain ( )
```

5.9.1.9 readManagers()

```
Manager * readManagers ( )
```

It reads a file and creates a linked list of managers

Returns

A pointer to a Manager struct.

5.9.1.10 removeManager()

```
Manager * removeManager (
            Manager * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, remove it and return the new head. Otherwise, find the element to be removed and remove it

Parameters

head	The head of the linked list
id	The id of the manager to be removed

Returns

The head of the list.

5.9.1.11 saveManagers()

```
int saveManagers (
            Manager * head )
```

It saves the managers to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, and 0 if it wasn't.

5.10 menus.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void menuApp ()
- void menuMain ()
- void menuMainClients (int available)

- void menuLine ()
- void menuAuth ()
- void menuAuthClients ()
- void menuAuthManagers ()
- void menuHeaderRides ()
- void menuHeaderRidesClient ()
- void menuHeaderVehicles ()
- void menuHeaderClients ()
- void menuHeaderClient ()
- void menuHeaderManagers ()
- void menuFooterRides ()
- void menuFooterVehicles ()
- void menuFooterCollections ()
- void menuFooterClients ()
- void menuFooterManagers ()
- void menuTitleInsertVehicle ()
- void menuTitleRemoveVehicle ()
- void menuTitleEditVehicle ()
- void menuTitleInsertClient ()
- void menuTitleRemoveClient ()
- void menuTitleEditClient ()
- void menuTitleAddBalance ()
- void menuTitleRemoveBalance ()
- void menuTitleInsertManager ()
- void menuTitleRemoveManager ()
- void menuTitleEditManager ()

5.10.1 Function Documentation

5.10.1.1 menuApp()

```
void menuApp ( )
```

5.10.1.2 menuAuth()

```
void menuAuth ( )
```

5.10.1.3 menuAuthClients()

```
void menuAuthClients ( )
```

5.10.1.4 menuAuthManagers()

```
void menuAuthManagers ( )
```

5.10.1.5 menuFooterClients()

```
void menuFooterClients ( )
```

5.10.1.6 menuFooterCollections()

```
void menuFooterCollections ( )
```

5.10.1.7 menuFooterManagers()

```
void menuFooterManagers ( )
```

5.10.1.8 menuFooterRides()

```
void menuFooterRides ( )
```

5.10.1.9 menuFooterVehicles()

```
void menuFooterVehicles ( )
```

5.10.1.10 menuHeaderClient()

```
void menuHeaderClient ( )
```

5.10.1.11 menuHeaderClients()

```
void menuHeaderClients ( )
```

5.10.1.12 menuHeaderManagers()

```
void menuHeaderManagers ( )
```

5.10.1.13 menuHeaderRides()

```
void menuHeaderRides ( )
```

5.10.1.14 menuHeaderRidesClient()

```
void menuHeaderRidesClient ( )
```

5.10.1.15 menuHeaderVehicles()

```
void menuHeaderVehicles ( )
```

5.10.1.16 menuLine()

```
void menuLine ( )
```

5.10.1.17 menuMain()

```
void menuMain ( )
```

5.10.1.18 menuMainClients()

```
void menuMainClients (
             int available )
```

5.10.1.19 menuTitleAddBalance()

```
void menuTitleAddBalance ( )
```

5.10.1.20 menuTitleEditClient()

```
void menuTitleEditClient ( )
```

5.10.1.21 menuTitleEditManager()

```
void menuTitleEditManager ( )
```

5.10.1.22 menuTitleEditVehicle()

```
void menuTitleEditVehicle ( )
```

5.10.1.23 menuTitleInsertClient()

```
void menuTitleInsertClient ( )
```

5.10.1.24 menuTitleInsertManager()

```
void menuTitleInsertManager ( )
```

5.10.1.25 menuTitleInsertVehicle()

```
void menuTitleInsertVehicle ( )
```

5.10.1.26 menuTitleRemoveBalance()

```
void menuTitleRemoveBalance ( )
```

5.10.1.27 menuTitleRemoveClient()

```
void menuTitleRemoveClient ( )
```

5.10.1.28 menuTitleRemoveManager()

```
void menuTitleRemoveManager ( )
```

5.10.1.29 menuTitleRemoveVehicle()

```
void menuTitleRemoveVehicle ( )
```

5.11 rides.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "../inc/header.h"
```

Functions

- void ridesMain ()
- Ride * insertRide (Ride *head, int id, int vehicle, int client, int startTime, int endTime, char startLocation[ ], char endLocation[ ], float cost, float distance)
- Ride * startRide (Ride *head, Vehicle *headVehicles, Type *headTypes, Client *headClients, int id, int vehicle, int client)
- void endRide (Ride *head, Vehicle *headVehicles, Type *headTypes, Client *headClients, Location *head↩ Locations, int id, char endLocation[ ])
- int listRides (Ride *head, Client *headClients)
- int listRidesClient (Ride *head, Client *headClients, int id)
- int assignRideId (Ride *head)
- int currentRide (Ride *head, int id)
- void showRide (Ride *head, int id)
- int saveRides (Ride *head)
- Ride * readRides ()

5.11.1 Function Documentation

5.11.1.1 assignRideId()

```
int assignRideId (
            Ride * head )
```

It returns the next available ride id

Parameters

head	The head of the linked list

Returns

The next available ride id.

5.11.1.2 currentRide()

```
int currentRide (
            Ride * head,
            int id )
```

It returns the id of the ride that the client is currently on, or -1 if the client is not on a ride

Parameters

head	The head of the linked list
id	The id of the client

Returns

The id of the ride that the client is currently on.

5.11.1.3 endRide()

```
void endRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            Location * headLocations,
            int id,
            char endLocation[] )
```

It takes a ride, a vehicle, a type, a client, an id, and an end location, and then it sets the end time, end location, cost, distance, and range of the ride

Parameters

head	The head of the linked list
headVehicles	Pointer to the first vehicle in the linked list
headTypes	Pointer to the first type of vehicle in the linked list
headClients	Pointer to the first client in the linked list
id	The id of the ride
endLocation	The end location of the ride

5.11.1.4 insertRide()

```
Ride * insertRide (
            Ride * head,
            int id,
            int vehicle,
            int client,
            int startTime,
            int endTime,
            char startLocation[],
            char endLocation[],
            float cost,
            float distance )
```

It inserts a new ride into the linked list of rides

Parameters

head	The head of the linked list
id	The id of the ride
vehicle	The id of the vehicle
client	The id of the client
startTime	The start time of the ride
endTime	The end time of the ride
startLocation	The start location of the ride
endLocation	The end location of the ride
cost	The cost of the ride
distance	The distance of the ride

Returns

The head of the list.

5.11.1.5 listRides()

```
int listRides (
            Ride * head,
            Client * headClients )
```

It prints the list of rides

Parameters

head	The head of the linked list
headClients	Pointer to the first client in the linked list

Returns

The number of rides in the list.

5.11.1.6 listRidesClient()

```
int listRidesClient (
            Ride * head,
            Client * headClients,
            int id )
```

It prints out the rides of a client

Parameters

head	The head of the linked list
headClients	Pointer to the first node of the clients linked list
id	The id of the client

Returns

The number of rides that the client has.

5.11.1.7 readRides()

```
Ride * readRides ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Ride struct.

5.11.1.8 ridesMain()

```
void ridesMain ( )
```

5.11.1.9 saveRides()

```
int saveRides (
            Ride * head )
```

It saves the linked list of rides to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, and 0 if it wasn't.

5.11.1.10 showRide()

```
void showRide (
            Ride * head,
            int id )
```

It prints the information of a ride given its id

Parameters

head	The head of the linked list
id	The id of the ride

5.11.1.11 startRide()

```
Ride * startRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            int id,
            int vehicle,
            int client )
```

It takes a ride, a vehicle, a type, a client, and an id, and returns a ride

Parameters

head	The head of the linked list
headVehicles	Pointer to the first vehicle in the linked list
headTypes	Pointer to the first type of vehicle in the linked list
headClients	Pointer to the first client in the linked list
id	The id of the ride
vehicle	The id of the vehicle
client	The id of the client

Returns

The head of the list.

5.12 utilities.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void clrscr ()
- void clrbuffer ()
- void enterToContinue ()
- void showCount (int count)

5.12.1 Function Documentation

5.12.1.1 clrbuffer()

```
void clrbuffer ( )
```

It clears the input buffer

5.12.1.2 clrscr()

```
void clrscr ( )
```

It clears the screen

5.12.1.3 enterToContinue()

```
void enterToContinue ( )
```

It clears the buffer and prints a message to the user, then waits for the user to press a key

5.12.1.4 showCount()

```
void showCount (
            int count )
```

It prints a message to the user, telling them how many results were found

Parameters

count	The number of results to be shown.

5.13 vehicles.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/header.h"
```

Functions

- void vehiclesMain ()
- Vehicle * insertVehicle (Vehicle *head, int id, int type, float battery, float range, int available, char location[])
- Vehicle * removeVehicle (Vehicle *head, int id)
- void editVehicle (Vehicle *head, Type *headTypes, int id, int type, float battery, float range, char location[])
- int listVehicles (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int listVehiclesByRange (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int listVehiclesByBattery (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int listVehiclesByDistance (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int listVehiclesInLocation (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int listVehiclesInRadius (Vehicle *head, Type *headTypes, Location *headLocations, char location[], float radius)
- int listVehiclesByTypeInRadius (Vehicle *head, Type *headTypes, Location *headLocations, int type, char location[], float radius)
- int listVehiclesByBatteryHalfCharged (Vehicle *head, Type *headTypes, Location *headLocations, char location[])
- int existVehicle (Vehicle *head, int id)
- int assignVehicleId (Vehicle *head)
- int isVehicleAvailable (Vehicle *head, int id)
- int isVehicleCharged (Vehicle *head, int id)
- void updateVehicleLocation (Vehicle *head, int id, char location[])
- Vehicle * chargeVehicles (Vehicle *head, char location[])
- Vehicle * copyLinkedList (Vehicle *head)
- int saveVehicles (Vehicle *head)
- Vehicle * readVehicles ()
- char * getVehicleTypeName (Vehicle *head, Type *headTypes, int id)
- float getVehicleBattery (Vehicle *head, int id)
- char * getVehicleLocation (Vehicle *head, int id)
- float getVehicleCost (Vehicle *head, Type *headTypes, int id)
- float getTypeCost (Type *head, int id)
- char * getTypeName (Type *head, int id)
- Type * insertType (Type *head, int id, char name[], float cost)
- int listTypes (Type *head)
- int existType (Type *head, int id)
- int saveTypes (Type *head)
- Type * readTypes ()

5.13.1 Function Documentation

5.13.1.1 assignVehicleId()

```
int assignVehicleId (
            Vehicle * head )
```

It returns the next available vehicle id

Parameters

head	The head of the linked list

Returns

The next available ID number.

5.13.1.2 chargeVehicles()

```
Vehicle * chargeVehicles (
            Vehicle * head,
            char location[] )
```

The function charges all vehicles located in a specific location by setting their battery to 100% and updating their range accordingly.

Parameters

head	A pointer to the head of a linked list of Vehicle structures.
location	The location where the vehicles are currently parked or located.

Returns

a pointer to the head of the linked list of vehicles.

5.13.1.3 copyLinkedList()

```
Vehicle * copyLinkedList (
            Vehicle * head )
```

It creates a new linked list, and copies the contents of the original linked list into the new linked list

Parameters

head	The head of the linked list

Returns

The head of the copied linked list.

5.13.1.4 editVehicle()

```
void editVehicle (
            Vehicle * head,
            Type * headTypes,
            int id,
            int type,
            float battery,
            float range,
            char location[] )
```

It edits a vehicle's information

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list
id	The id of the vehicle to edit
type	The type of vehicle
battery	The battery of the vehicle
range	The range of the vehicle
location	The location of the vehicle

5.13.1.5 existType()

```
int existType (
            Type * head,
            int id )
```

It checks if a type with the given id exists in the list

Parameters

head	The head of the linked list
id	The id of the type

Returns

1 if the type exists in the list, otherwise it returns 0.

5.13.1.6 existVehicle()

```
int existVehicle (
            Vehicle * head,
            int id )
```

It returns 1 if the vehicle with the given id exists in the list, otherwise it returns 0

Parameters

head	The head of the linked list
id	The id of the vehicle to be added

Returns

1 if the vehicle exists in the list, otherwise it returns 0.

5.13.1.7 getTypeCost()

```
float getTypeCost (
            Type * head,
            int id )
```

It returns the cost of a type with a given id

Parameters

head	The head of the linked list
id	The id of the type you want to get the cost of.

Returns

The cost of the type with the given id.

5.13.1.8 getTypeName()

```
char * getTypeName (
            Type * head,
            int id )
```

It returns the name of the type with the given id, or "*********" if the type doesn't exist

Parameters

head	The head of the linked list
id	The id of the type you want to get the name of.

Returns

The name of the type with the given id.

5.13.1.9 getVehicleBattery()

```
float getVehicleBattery (
            Vehicle * head,
            int id )
```

The function returns the battery level of a vehicle with a given ID from a linked list of vehicles.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

The function `getVehicleBattery` returns a float value representing the battery level of a vehicle with the given `id`. If a vehicle with the given `id` is found in the linked list pointed to by `head`, the function returns the battery level of that vehicle. If no vehicle with the given `id` is found, the function returns -1.

5.13.1.10 getVehicleCost()

```
float getVehicleCost (
            Vehicle * head,
            Type * headTypes,
            int id )
```

It loops through the linked list of vehicles, and if the vehicle's id matches the id passed in, it returns the cost of the vehicle's type

Parameters

head	The head of the linked list of vehicles
headTypes	The head of the linked list of types
id	The id of the vehicle you want to get the cost of.

Returns

The cost of the vehicle.

5.13.1.11 getVehicleLocation()

```
char * getVehicleLocation (
            Vehicle * head,
            int id )
```

The function returns the location of a vehicle with a given ID from a linked list of vehicles.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

If a vehicle with the given ID is found in the linked list, the function returns the location of that vehicle as a string. If no vehicle with the given ID is found, the function returns the string "*********".

5.13.1.12 getVehicleTypeName()

```
char * getVehicleTypeName (
            Vehicle * head,
            Type * headTypes,
            int id )
```

The function returns the name of a vehicle type given its ID, by iterating through a linked list of vehicles and using a separate linked list of types.

Parameters

head	A pointer to the head of a linked list of Vehicle structs.
headTypes	A pointer to the head of a linked list of Type structs.
id	The id parameter is an integer that represents the unique identifier of a vehicle.

Returns

a string that represents the name of the vehicle type associated with the given ID. If the ID is not found in the linked list of vehicles, the function returns a string of asterisks.

5.13.1.13 insertType()

```
Type * insertType (
            Type * head,
            int id,
            char name[],
            float cost )
```

It inserts a new client at the end of the list

Parameters

head	The head of the linked list
id	The id of the type of vehicle
name	The name of the type of vehicle
cost	The cost of the type of vehicle

Returns

The head of the list.

5.13.1.14 insertVehicle()

```
Vehicle * insertVehicle (
            Vehicle * head,
            int id,
            int type,
            float battery,
            float range,
            int available,
            char location[] )
```

It inserts a new vehicle at the end of the list

Parameters

head	The head of the linked list
id	The id of the vehicle
type	The type of the vehicle
battery	The battery of the vehicle
range	The range of the vehicle
available	0 = not available, 1 = available
location	The location of the vehicle

Returns

The head of the list.

5.13.1.15 isVehicleAvailable()

```
int isVehicleAvailable (
            Vehicle * head,
            int id )
```

It checks if a vehicle is available

Parameters

head	The head of the linked list
id	The id of the vehicle to check

Returns

1 if the vehicle is available, otherwise it returns 0.

5.13.1.16 isVehicleCharged()

```
int isVehicleCharged (
            Vehicle * head,
            int id )
```

It checks if the vehicle is charged and has a range greater than 0

Parameters

head	The head of the linked list
id	The id of the vehicle to check

Returns

1 if the vehicle has any battery, otherwise it returns 0.

5.13.1.17 listTypes()

```
int listTypes (
            Type * head )
```

It prints the contents of a linked list of types

Parameters

head	The head of the linked list

Returns

The number of items in the list.

5.13.1.18 listVehicles()

```
int listVehicles (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It prints a list of vehicles

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list

Returns

The number of vehicles in the list.

5.13.1.19 listVehiclesByBattery()

```
int listVehiclesByBattery (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function sorts a linked list of vehicles by their battery level and then lists them.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	A pointer to the head of a linked list of Type structs, which contain information about the types of vehicles (e.g. electric, hybrid, gas).
headLocations	A pointer to the head of a linked list of Location structs.
location	The parameter "location" is a string that represents the location where the vehicles are located. It is used as a filter to only list the vehicles that are located in that specific location.

Returns

the result of calling the function `listVehicles` with the sorted linked list of vehicles as its first argument, and the other arguments passed to the function as well.

5.13.1.20 listVehiclesByBatteryHalfCharged()

```
int listVehiclesByBatteryHalfCharged (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function filters vehicles with battery levels below 50% and lists them by location.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	A pointer to the head of the linked list of vehicle types.
headLocations	A pointer to the head of a linked list of Location structs, which contains information about the locations of vehicles.
location	The parameter "location" is a string that represents the name of a location. It is used as a filter to list only the vehicles that are located in that specific location.

Returns

the result of calling the function `listVehiclesByBattery()` with the filtered list of vehicles as the first argument, along with the other arguments passed to the function.

5.13.1.21 listVehiclesByDistance()

```
int listVehiclesByDistance (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

This function lists vehicles in ascending order of distance from a given location, with ties broken by range.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	a pointer to the head of a linked list of Type structs
headLocations	a pointer to the head of a linked list of Location structs
location	A string representing the location for which the vehicles need to be listed in order of increasing distance.

Returns

an integer value, which is the result of calling the function `listVehicles` with the sorted linked list of vehicles as its first argument, and the other linked lists as the remaining arguments.

5.13.1.22 listVehiclesByRange()

```
int listVehiclesByRange (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It sorts the linked list by range, then lists the vehicles

Parameters

head	The head of the linked list
headTypes	Pointer to the first type of vehicle in the linked list

Returns

The return value is the result of the function listVehicles.

5.13.1.23 listVehiclesByTypeInRadius()

```
int listVehiclesByTypeInRadius (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            int type,
            char location[],
            float radius )
```

This function filters a linked list of vehicles by type and location within a certain radius and then lists them by distance.

Parameters

head	A pointer to the head of a linked list of Vehicle structs.
headTypes	It is a pointer to the head of a linked list of Type structs.
headLocations	A pointer to the head of a linked list of Location structs, which contains information about the locations of vehicles.
type	an integer representing the type of vehicle to filter by. If set to 0, all types of vehicles will be included in the result.
location	The location parameter is a string that represents the location from which the distance to the vehicles will be calculated.
radius	The radius is a float value that represents the maximum distance from a given location within which vehicles of a certain type should be listed.

Returns

the result of calling the function `listVehiclesByDistance` with the filtered list of vehicles as the first argument, and the head of the types and locations lists and the specified location as the remaining arguments.

5.13.1.24 listVehiclesInLocation()

```
int listVehiclesInLocation (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[] )
```

It filters the linked list by location, then lists the vehicles sorted by range

Parameters

head	pointer to the first element of the linked list
headTypes	a linked list of types
location	The location of the vehicle

Returns

The return value is the number of vehicles that were listed.

5.13.1.25 listVehiclesInRadius()

```
int listVehiclesInRadius (
            Vehicle * head,
            Type * headTypes,
            Location * headLocations,
            char location[],
            float radius )
```

The function lists all vehicles within a certain radius of a given location.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
headTypes	A pointer to the head of a linked list of vehicle types.
headLocations	A linked list of Location structs containing information about the locations of vehicles.
location	The location parameter is a string that represents the starting location from which the distance to each vehicle's location will be calculated.
radius	The radius is a float value that represents the maximum distance from a given location within which vehicles should be listed.

Returns

the result of calling the function `listVehiclesByDistance` with the filtered list of vehicles as the first argument, and the head of the types and locations linked lists, as well as a location string, as the remaining arguments.

5.13.1.26 readTypes()

```
Type * readTypes ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Type struct.

5.13.1.27 readVehicles()

```
Vehicle * readVehicles ( )
```

It reads a file and inserts the data into a linked list

Returns

A pointer to a Vehicle struct.

5.13.1.28 removeVehicle()

```
Vehicle * removeVehicle (
            Vehicle * head,
            int id )
```

If the list is empty, return NULL. If the first element is the one to be removed, free it and return the second element. Otherwise, find the element to be removed and free it

Parameters

head	The head of the linked list
id	The id of the vehicle to be removed

Returns

The head of the list.

5.13.1.29 saveTypes()

```
int saveTypes (
            Type * head )
```

It saves the types to a file

Parameters

head	The head of the linked list

Returns

1 if the file was saved successfully, or 0 if it wasn't.

5.13.1.30 saveVehicles()

```
int saveVehicles (
            Vehicle * head )
```

It saves the vehicles to a file

Parameters

head	The head of the linked list

Returns

1 if the file was successfully saved, and 0 if it was not.

5.13.1.31 updateVehicleLocation()

```
void updateVehicleLocation (
            Vehicle * head,
            int id,
            char location[] )
```

The function updates the location of a vehicle with a given ID in a linked list.

Parameters

head	a pointer to the head of a linked list of Vehicle structs
id	The id parameter is an integer that represents the unique identifier of a vehicle.
location	The parameter "location" is a character array that represents the new location of a vehicle. This function updates the location of a vehicle with the given "id" to the new location provided in the "location" parameter.

5.13.1.32 vehiclesMain()

```
void vehiclesMain ( )
```

Index