Volt

Generated by Doxygen 1.9.6

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 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 address [SIZE_ADDRESS]
- float balance
- int available
- struct client * next

3.1.1 Field Documentation

3.1.1.1 address

`char address[SIZE_ADDRESS]`

3.1.1.2 available

`int available`

3.1.1.3 balance

```
float balance
```

3.1.1.4 id

```
int id
```

3.1.1.5 name

```
char name[SIZE_NAME]
```

3.1.1.6 next

```
struct client* next
```

3.1.1.7 nif

```
int nif
```

3.1.1.8 password

```
char password[SIZE_PASSWORD]
```

3.1.1.9 username

```
char username[SIZE_USERNAME]
```

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

- header.h

3.2 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

3.2.1 Field Documentation

3.2.1.1 id

```
int id
```

3.2.1.2 name

```
char name[SIZE_NAME]
```

3.2.1.3 next

```
struct manager* next
```

3.2.1.4 password

```
char password[SIZE_PASSWORD]
```

3.2.1.5 username

```
char username[SIZE_USERNAME]
```

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

- header.h

3.3 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

3.3.1 Field Documentation

3.3.1.1 client

`int client`

3.3.1.2 cost

`float cost`

3.3.1.3 distance

`float distance`

3.3.1.4 endLocation

`char endLocation[SIZE_LOCATION]`

3.3.1.5 endTime

```
time_t endTime
```

3.3.1.6 id

```
int id
```

3.3.1.7 next

```
struct ride* next
```

3.3.1.8 startLocation

```
char startLocation[SIZE_LOCATION]
```

3.3.1.9 startTime

```
time_t startTime
```

3.3.1.10 vehicle

```
int vehicle
```

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

- header.h

3.4 type Struct Reference

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

Data Fields

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

3.4.1 Field Documentation

3.4.1.1 cost

```
float cost
```

3.4.1.2 id

```
int id
```

3.4.1.3 name

```
char name[SIZE_NAME]
```

3.4.1.4 next

```
struct type* next
```

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

- header.h

3.5 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

3.5.1 Field Documentation

3.5.1.1 available

```
int available
```

3.5.1.2 battery

```
float battery
```

3.5.1.3 id

```
int id
```

3.5.1.4 location

```
char location[SIZE_LOCATION]
```

3.5.1.5 next

```
struct vehicle* next
```

3.5.1.6 range

```
float range
```

3.5.1.7 type

```
int type
```

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

- header.h

Chapter 4

File Documentation

4.1 header.h File Reference

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

Data Structures

- struct type
- struct vehicle
- struct client
- struct manager
- struct ride

Macros

- #define DATA_DIR "data/"
- #define SIZE_USERNAME 40
- #define SIZE_PASSWORD 40
- #define SIZE_NAME 60
- #define SIZE_ADDRESS 150
- #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 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 type Type
- typedef struct vehicle Vehicle
- typedef struct client Client
- typedef struct manager Manager
- typedef struct ride Ride

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, int id, char end↩ Location[ ])
- 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)
- int listVehiclesByRange (Vehicle *head, Type *headTypes)
- int listVehiclesByLocation (Vehicle *head, Type *headTypes, char location[ ])
- int existVehicle (Vehicle *head, int id)
- int assignVehicleId (Vehicle *head)
- int isVehicleAvailable (Vehicle *head, int id)
- int isVehicleCharged (Vehicle *head, int id)
- Vehicle * copyLinkedList (Vehicle *head)
- int saveVehicles (Vehicle *head)
- Vehicle * readVehicles ()
- 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 clientsMain ()
- Client * insertClient (Client *head, int id, char username[ ], char password[ ], char name[ ], int nif, char address[ ], 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 address[ ])
- int listClients (Client *head)
- int listClient (Client *head, int id)
- char * getClientName (Client *head, int id)

- char * getClientUsername (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[])
- void decrypt (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 menuMainClientsLine ()
- void menuAuth ()
- void menuAuthClients ()
- void menuAuthManagers ()
- void menuHeaderRides ()
- void menuHeaderRidesClient ()
- void menuHeaderVehicles ()
- void menuHeaderClients ()
- void menuHeaderClient ()
- void menuHeaderManagers ()
- void menuFooterRides ()
- void menuFooterVehicles ()
- void menuFooterClients ()
- void menuFooterManagers ()
- void menuTitleInsertVehicle ()
- void menuTitleRemoveVehicle ()
- void menuTitleEditVehicle ()
- void menuTitleListVehiclesByLocation ()
- 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)

4.1.1 Macro Definition Documentation

4.1.1.1 BLUE

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

4.1.1.2 CYAN

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

4.1.1.3 DATA_DIR

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

4.1.1.4 GREEN

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

4.1.1.5 MAGENTA

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

4.1.1.6 RED

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

4.1.1.7 RESET

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

4.1.1.8 SIZE_ADDRESS

```
#define SIZE_ADDRESS 150
```

4.1.1.9 SIZE_BATTERY

```
#define SIZE_BATTERY 15
```

4.1.1.10 SIZE_DATETIME

```
#define SIZE_DATETIME 20
```

4.1.1.11 SIZE_LOCATION

```
#define SIZE_LOCATION 60
```

4.1.1.12 SIZE_NAME

```
#define SIZE_NAME 60
```

4.1.1.13 SIZE_NIF

```
#define SIZE_NIF 15
```

4.1.1.14 SIZE_PASSWORD

```
#define SIZE_PASSWORD 40
```

4.1.1.15 SIZE_RANGE

```
#define SIZE_RANGE 15
```

4.1.1.16 SIZE_TYPE

```
#define SIZE_TYPE 5
```

4.1.1.17 SIZE_USERNAME

```
#define SIZE_USERNAME 40
```

4.1.1.18 WHITE

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

4.1.1.19 YELLOW

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

4.1.2 Typedef Documentation

4.1.2.1 Client

```
typedef struct client Client
```

4.1.2.2 Manager

```
typedef struct manager Manager
```

4.1.2.3 Ride

```
typedef struct ride Ride
```

4.1.2.4 Type

```
typedef struct type Type
```

4.1.2.5 Vehicle

```
typedef struct vehicle Vehicle
```

4.1.3 Function Documentation

4.1.3.1 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

4.1.3.2 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.

4.1.3.3 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.

4.1.3.4 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.

4.1.3.5 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.

4.1.3.6 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	"test"
password	the password to be encrypted

Returns

The ID of the client.

4.1.3.7 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.

4.1.3.8 clientsMain()

```
void clientsMain ( )
```

4.1.3.9 clrbuffer()

```
void clrbuffer ( )
```

It clears the input buffer

4.1.3.10 clrscr()

```
void clrscr ( )
```

It clears the screen

4.1.3.11 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.

4.1.3.12 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.

4.1.3.13 decrypt()

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

4.1.3.14 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

4.1.3.15 editClient()

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

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
address	The address of the client

4.1.3.16 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

4.1.3.17 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

Parameters

type	The type of vehicle
battery	The battery of the vehicle
range	The range of the vehicle
location	The location of the vehicle

4.1.3.18 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.

4.1.3.19 endRide()

```
void endRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            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

4.1.3.20 enterToContinue()

```
void enterToContinue ( )
```

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

4.1.3.21 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.

4.1.3.22 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.

4.1.3.23 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.

4.1.3.24 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.

4.1.3.25 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.

4.1.3.26 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.

4.1.3.27 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.

4.1.3.28 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.

4.1.3.29 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.

4.1.3.30 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.

4.1.3.31 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.

4.1.3.32 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.

4.1.3.33 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.

4.1.3.34 insertClient()

```
Client * insertClient (
            Client * head,
            int id,
            char username[],
            char password[],
            char name[],
            int nif,
            char address[],
            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
address	The address of the client
balance	The balance of the client
available	0 = not available, 1 = available

Returns

The head of the list.

4.1.3.35 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.

4.1.3.36 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.

4.1.3.37 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.

4.1.3.38 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.

4.1.3.39 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.

4.1.3.40 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.

4.1.3.41 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.

4.1.3.42 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.

4.1.3.43 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.

4.1.3.44 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.

4.1.3.45 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.

4.1.3.46 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.

4.1.3.47 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.

4.1.3.48 listVehicles()

```
int listVehicles (
            Vehicle * head,
            Type * headTypes )
```

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.

4.1.3.49 listVehiclesByLocation()

```
int listVehiclesByLocation (
            Vehicle * head,
            Type * headTypes,
            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.

4.1.3.50 listVehiclesByRange()

```
int listVehiclesByRange (
            Vehicle * head,
            Type * headTypes )
```

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.

4.1.3.51 managersMain()

```
void managersMain ( )
```

4.1.3.52 menuApp()

```
void menuApp ( )
```

4.1.3.53 menuAuth()

```
void menuAuth ( )
```

4.1.3.54 menuAuthClients()

```
void menuAuthClients ( )
```

4.1.3.55 menuAuthManagers()

```
void menuAuthManagers ( )
```

4.1.3.56 menuFooterClients()

```
void menuFooterClients ( )
```

4.1.3.57 menuFooterManagers()

```
void menuFooterManagers ( )
```

4.1.3.58 menuFooterRides()

```
void menuFooterRides ( )
```

4.1.3.59 menuFooterVehicles()

```
void menuFooterVehicles ( )
```

4.1.3.60 menuHeaderClient()

```
void menuHeaderClient ( )
```

4.1.3.61 menuHeaderClients()

```
void menuHeaderClients ( )
```

4.1.3.62 menuHeaderManagers()

```
void menuHeaderManagers ( )
```

4.1.3.63 menuHeaderRides()

```
void menuHeaderRides ( )
```

4.1.3.64 menuHeaderRidesClient()

```
void menuHeaderRidesClient ( )
```

4.1.3.65 menuHeaderVehicles()

```
void menuHeaderVehicles ( )
```

4.1.3.66 menuMain()

```
void menuMain ( )
```

4.1.3.67 menuMainClients()

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

4.1.3.68 menuMainClientsLine()

```
void menuMainClientsLine ( )
```

4.1.3.69 menuTitleAddBalance()

```
void menuTitleAddBalance ( )
```

4.1.3.70 menuTitleEditClient()

```
void menuTitleEditClient ( )
```

4.1.3.71 menuTitleEditManager()

```
void menuTitleEditManager ( )
```

4.1.3.72 menuTitleEditVehicle()

```
void menuTitleEditVehicle ( )
```

4.1.3.73 menuTitleInsertClient()

```
void menuTitleInsertClient ( )
```

4.1.3.74 menuTitleInsertManager()

```
void menuTitleInsertManager ( )
```

4.1.3.75 menuTitleInsertVehicle()

```
void menuTitleInsertVehicle ( )
```

4.1.3.76 menuTitleListVehiclesByLocation()

```
void menuTitleListVehiclesByLocation ( )
```

4.1.3.77 menuTitleRemoveBalance()

```
void menuTitleRemoveBalance ( )
```

4.1.3.78 menuTitleRemoveClient()

```
void menuTitleRemoveClient ( )
```

4.1.3.79 menuTitleRemoveManager()

```
void menuTitleRemoveManager ( )
```

4.1.3.80 menuTitleRemoveVehicle()

```
void menuTitleRemoveVehicle ( )
```

4.1.3.81 readClients()

```
Client * readClients ( )
```

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

Returns

A pointer to a Client struct.

4.1.3.82 readManagers()

```
Manager * readManagers ( )
```

It reads a file and creates a linked list of managers

Returns

A pointer to a Manager struct.

4.1.3.83 readRides()

```
Ride * readRides ( )
```

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

Returns

A pointer to a Ride struct.

4.1.3.84 readTypes()

```
Type * readTypes ( )
```

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

Returns

A pointer to a Type struct.

4.1.3.85 readVehicles()

```
Vehicle * readVehicles ( )
```

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

Returns

A pointer to a Vehicle struct.

4.1.3.86 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

4.1.3.87 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.

4.1.3.88 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.

4.1.3.89 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.

4.1.3.90 ridesMain()

```
void ridesMain ( )
```

4.1.3.91 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.

4.1.3.92 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.

4.1.3.93 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.

4.1.3.94 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.

4.1.3.95 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.

4.1.3.96 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.

4.1.3.97 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

4.1.3.98 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.

4.1.3.99 vehiclesMain()

```
void vehiclesMain ( )
```

4.2 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_ADDRESS 150
00010 #define SIZE_LOCATION 60
00011 #define SIZE_TYPE 5
00012 #define SIZE_BATTERY 15
00013 #define SIZE_RANGE 15
00014 #define SIZE_NIF 15
00015 #define SIZE_DATETIME 20
00016
00017 #define RED     "\x1B[31m"
00018 #define GREEN   "\x1B[32m"
00019 #define YELLOW  "\x1B[33m"
00020 #define BLUE    "\x1B[34m"
00021 #define MAGENTA "\x1B[35m"
00022 #define CYAN    "\x1B[36m"
00023 #define WHITE   "\x1B[37m"
00024 #define RESET   "\x1B[0m"
00025
00026 #include <time.h>
00027
00028 typedef struct type {
00029     int id; // 1 - Trotinete; 2 - Bicicleta
00030     char name[SIZE_NAME];
00031     float cost;
00032     struct type* next;
00033
00034 } Type;
00035
00036 typedef struct vehicle {
00037     int id;
00038     int type;
00039     float battery;
00040     float range;
00041     char location[SIZE_LOCATION];
00042     int available;
00043     struct vehicle* next;
```

```
00044
00045 } Vehicle;
00046
00047 typedef struct client {
00048     int id;
00049     char username[SIZE_USERNAME];
00050     char password[SIZE_PASSWORD];
00051     char name[SIZE_NAME];
00052     int nif;
00053     char address[SIZE_ADDRESS];
00054     float balance;
00055     int available;
00056     struct client* next;
00057
00058 } Client;
00059
00060 typedef struct manager {
00061     int id;
00062     char username[SIZE_USERNAME];
00063     char password[SIZE_PASSWORD];
00064     char name[SIZE_NAME];
00065     struct manager* next;
00066
00067 } Manager;
00068
00069 typedef struct ride {
00070     int id;
00071     int vehicle;
00072     int client;
00073     time_t startTime;
00074     time_t endTime;
00075     char startLocation[SIZE_LOCATION];
00076     char endLocation[SIZE_LOCATION];
00077     float cost;
00078     float distance;
00079     struct ride* next;
00080
00081 } Ride;
00082
00083 /*Rides*/
00084 void ridesMain();
00085 Ride* insertRide(Ride* head, int id, int vehicle, int client, int startTime, int endTime, char
      startLocation[], char endLocation[], float cost, float distance);
00086 Ride* startRide(Ride* head, Vehicle* headVehicles, Type* headTypes, Client* headClients, int id, int
      vehicle, int client);
00087 void endRide(Ride* head, Vehicle* headVehicles, Type* headTypes, Client* headClients, int id, char
      endLocation[]);
00088 int listRides(Ride* head, Client* headClients);
00089 int listRidesClient(Ride* head, Client* headClients, int id);
00090 int assignRideId(Ride* head);
00091 int currentRide(Ride* head, int id);
00092 void showRide(Ride* head, int id);
00093 int saveRides(Ride* head);
00094 Ride* readRides();
00095
00096 /*Vehicles*/
00097 void vehiclesMain();
00098 Vehicle* insertVehicle(Vehicle* head, int id, int type, float battery, float range, int available,
      char location[]);
00099 Vehicle* removeVehicle(Vehicle* head, int id);
00100 void editVehicle(Vehicle* head, Type* headTypes, int id, int type, float battery, float range, char
      location[]);
00101 int listVehicles(Vehicle* head, Type* headTypes);
00102 int listVehiclesByRange(Vehicle* head, Type* headTypes);
00103 int listVehiclesByLocation(Vehicle* head, Type* headTypes, char location[]);
00104 int existVehicle(Vehicle* head, int id);
00105 int assignVehicleId(Vehicle* head);
00106 int isVehicleAvailable(Vehicle* head, int id);
00107 int isVehicleCharged(Vehicle* head, int id);
00108 Vehicle* copyLinkedList(Vehicle* head);
00109 int saveVehicles(Vehicle* head);
00110 Vehicle* readVehicles();
00111 float getVehicleCost(Vehicle* head, Type* headTypes, int id);
00112 float getTypeCost(Type* head, int id);
00113 char* getTypeName(Type* head, int id);
00114 Type* insertType(Type* head, int id, char name[], float cost);
00115 int listTypes(Type* head);
00116 int existType(Type* head, int id);
00117 int saveTypes(Type* head);
00118 Type* readTypes();
00119
00120 /*Clients*/
00121 void clientsMain();
00122 Client* insertClient(Client* head, int id, char username[], char password[], char name[], int nif,
      char address[], float balance, int available);
00123 Client* removeClient(Client* head, int id);
00124 void editClient(Client* head, int id, char username[], char password[], char name[], int nif, char
```

```
      address[]);
00125 int listClients(Client* head);
00126 int listClient(Client* head, int id);
00127 char* getClientName(Client* head, int id);
00128 char* getClientUsername(Client* head, int id);
00129 int existClientUsername(Client* head, char username[]);
00130 int existClient(Client* head, int id);
00131 int assignClientId(Client* head);
00132 int isClientAvailable(Client* head, int id);
00133 void addBalance(Client* head, int id, float balance);
00134 void removeBalance(Client* head, int id, float balance);
00135 void editBalance(Client* head, int id, float balance);
00136 int hasBalance(Client* head, int id);
00137 int saveClients(Client* head);
00138 Client* readClients();
00139
00140 /*Managers*/
00141 void managersMain();
00142 Manager* insertManager(Manager* head, int id, char username[], char password[], char name[]);
00143 Manager* removeManager(Manager* head, int id);
00144 void editManager(Manager* head, int id, char username[], char password[], char name[]);
00145 int listManagers(Manager* head);
00146 char* getManagerName(Manager* head, int id);
00147 int existManagerUsername(Manager* head, char username[]);
00148 int existManager(Manager* head, int id);
00149 int assignManagerId(Manager* head);
00150 int saveManagers(Manager* head);
00151 Manager* readManagers();
00152
00153 /*Auth*/
00154 void encrypt(char password[]);
00155 void decrypt(char password[]);
00156 int authClient(Client* head, char username[], char password[]);
00157 int authManager(Manager* head, char username[], char password[]);
00158
00159 /*Menus*/
00160 void menuApp();
00161 void menuMain();
00162 void menuMainClients(int available);
00163 void menuMainClientsLine();
00164 void menuAuth();
00165 void menuAuthClients();
00166 void menuAuthManagers();
00167 void menuHeaderRides();
00168 void menuHeaderRidesClient();
00169 void menuHeaderVehicles();
00170 void menuHeaderClients();
00171 void menuHeaderClient();
00172 void menuHeaderManagers();
00173 void menuFooterRides();
00174 void menuFooterVehicles();
00175 void menuFooterClients();
00176 void menuFooterManagers();
00177 void menuTitleInsertVehicle();
00178 void menuTitleRemoveVehicle();
00179 void menuTitleEditVehicle();
00180 void menuTitleListVehiclesByLocation();
00181 void menuTitleInsertClient();
00182 void menuTitleRemoveClient();
00183 void menuTitleEditClient();
00184 void menuTitleAddBalance();
00185 void menuTitleRemoveBalance();
00186 void menuTitleInsertManager();
00187 void menuTitleRemoveManager();
00188 void menuTitleEditManager();
00189
00190 /*Utilities*/
00191 void clrscr();
00192 void clrbuffer();
00193 void enterToContinue();
00194 void showCount(int count);
00195
00196 #endif
```

4.3 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[ ])

4.3.1 Function Documentation

4.3.1.1 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	"test"
password	the password to be encrypted

Returns

The ID of the client.

4.3.1.2 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.

4.3.1.3 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.

4.4 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 address[ ], 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 address[ ])
- int listClients (Client *head)
- int listClient (Client *head, int id)
- char * getClientName (Client *head, int id)
- char * getClientUsername (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 ()

4.4.1 Function Documentation

4.4.1.1 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

4.4.1.2 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.

4.4.1.3 clientsMain()

```
void clientsMain ( )
```

4.4.1.4 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

4.4.1.5 editClient()

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

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
address	The address of the client

4.4.1.6 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.

4.4.1.7 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.

4.4.1.8 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.

4.4.1.9 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.

4.4.1.10 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.

4.4.1.11 insertClient()

```
Client * insertClient (
            Client * head,
            int id,
            char username[],
            char password[],
            char name[],
            int nif,
            char address[],
            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
address	The address of the client
balance	The balance of the client
available	0 = not available, 1 = available

Returns

The head of the list.

4.4.1.12 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.

4.4.1.13 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.

4.4.1.14 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.

4.4.1.15 readClients()

```
Client * readClients ( )
```

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

Returns

A pointer to a Client struct.

4.4.1.16 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

4.4.1.17 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.

4.4.1.18 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.

4.5 main.c File Reference

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

Functions

- int main ()

4.5.1 Function Documentation

4.5.1.1 main()

```
int main ( )
```

4.6 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 ()

4.6.1 Function Documentation

4.6.1.1 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.

4.6.1.2 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

4.6.1.3 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.

4.6.1.4 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.

4.6.1.5 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.

4.6.1.6 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.

4.6.1.7 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.

4.6.1.8 managersMain()

```
void managersMain ( )
```

4.6.1.9 readManagers()

```
Manager * readManagers ( )
```

It reads a file and creates a linked list of managers

Returns

A pointer to a Manager struct.

4.6.1.10 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.

4.6.1.11 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.

4.7 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 menuMainClientsLine ()
- void menuAuth ()
- void menuAuthClients ()
- void menuAuthManagers ()
- void menuHeaderRides ()
- void menuHeaderRidesClient ()
- void menuHeaderVehicles ()
- void menuHeaderClients ()
- void menuHeaderClient ()

- void menuHeaderManagers ()
- void menuFooterRides ()
- void menuFooterVehicles ()
- void menuFooterClients ()
- void menuFooterManagers ()
- void menuTitleInsertVehicle ()
- void menuTitleRemoveVehicle ()
- void menuTitleEditVehicle ()
- void menuTitleListVehiclesByLocation ()
- void menuTitleInsertClient ()
- void menuTitleRemoveClient ()
- void menuTitleEditClient ()
- void menuTitleAddBalance ()
- void menuTitleRemoveBalance ()
- void menuTitleInsertManager ()
- void menuTitleRemoveManager ()
- void menuTitleEditManager ()

4.7.1 Function Documentation

4.7.1.1 menuApp()

```
void menuApp ( )
```

4.7.1.2 menuAuth()

```
void menuAuth ( )
```

4.7.1.3 menuAuthClients()

```
void menuAuthClients ( )
```

4.7.1.4 menuAuthManagers()

```
void menuAuthManagers ( )
```

4.7.1.5 menuFooterClients()

```
void menuFooterClients ( )
```

4.7.1.6 menuFooterManagers()

```
void menuFooterManagers ( )
```

4.7.1.7 menuFooterRides()

```
void menuFooterRides ( )
```

4.7.1.8 menuFooterVehicles()

```
void menuFooterVehicles ( )
```

4.7.1.9 menuHeaderClient()

```
void menuHeaderClient ( )
```

4.7.1.10 menuHeaderClients()

```
void menuHeaderClients ( )
```

4.7.1.11 menuHeaderManagers()

```
void menuHeaderManagers ( )
```

4.7.1.12 menuHeaderRides()

```
void menuHeaderRides ( )
```

4.7.1.13 menuHeaderRidesClient()

```
void menuHeaderRidesClient ( )
```

4.7.1.14 menuHeaderVehicles()

```
void menuHeaderVehicles ( )
```

4.7.1.15 menuMain()

```
void menuMain ( )
```

4.7.1.16 menuMainClients()

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

4.7.1.17 menuMainClientsLine()

```
void menuMainClientsLine ( )
```

4.7.1.18 menuTitleAddBalance()

```
void menuTitleAddBalance ( )
```

4.7.1.19 menuTitleEditClient()

```
void menuTitleEditClient ( )
```

4.7.1.20 menuTitleEditManager()

```
void menuTitleEditManager ( )
```

4.7.1.21 menuTitleEditVehicle()

```
void menuTitleEditVehicle ( )
```

4.7.1.22 menuTitleInsertClient()

```
void menuTitleInsertClient ( )
```

4.7.1.23 menuTitleInsertManager()

```
void menuTitleInsertManager ( )
```

4.7.1.24 menuTitleInsertVehicle()

```
void menuTitleInsertVehicle ( )
```

4.7.1.25 menuTitleListVehiclesByLocation()

```
void menuTitleListVehiclesByLocation ( )
```

4.7.1.26 menuTitleRemoveBalance()

```
void menuTitleRemoveBalance ( )
```

4.7.1.27 menuTitleRemoveClient()

```
void menuTitleRemoveClient ( )
```

4.7.1.28 menuTitleRemoveManager()

```
void menuTitleRemoveManager ( )
```

4.7.1.29 menuTitleRemoveVehicle()

```
void menuTitleRemoveVehicle ( )
```

4.8 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, int id, char end↩ Location[ ])
- 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 ()

4.8.1 Function Documentation

4.8.1.1 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.

4.8.1.2 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.

4.8.1.3 endRide()

```
void endRide (
            Ride * head,
            Vehicle * headVehicles,
            Type * headTypes,
            Client * headClients,
            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

4.8.1.4 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.

4.8.1.5 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.

4.8.1.6 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.

4.8.1.7 readRides()

```
Ride * readRides ( )
```

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

Returns

A pointer to a Ride struct.

4.8.1.8 ridesMain()

```
void ridesMain ( )
```

4.8.1.9 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.

4.8.1.10 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

4.8.1.11 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.

4.9 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)

4.9.1 Function Documentation

4.9.1.1 clrbuffer()

```
void clrbuffer ( )
```

It clears the input buffer

4.9.1.2 clrscr()

```
void clrscr ( )
```

It clears the screen

4.9.1.3 enterToContinue()

```
void enterToContinue ( )
```

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

4.9.1.4 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.

4.10 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)
- int listVehiclesByRange (Vehicle *head, Type *headTypes)
- int listVehiclesByLocation (Vehicle *head, Type *headTypes, char location[ ])
- int existVehicle (Vehicle *head, int id)
- int assignVehicleId (Vehicle *head)
- int isVehicleAvailable (Vehicle *head, int id)
- int isVehicleCharged (Vehicle *head, int id)
- Vehicle * copyLinkedList (Vehicle *head)
- int saveVehicles (Vehicle *head)
- Vehicle * readVehicles ()
- 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 ()

4.10.1 Function Documentation

4.10.1.1 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.

4.10.1.2 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.

4.10.1.3 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

4.10.1.4 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.

4.10.1.5 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.

4.10.1.6 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.

4.10.1.7 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.

4.10.1.8 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.

4.10.1.9 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.

4.10.1.10 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.

4.10.1.11 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.

4.10.1.12 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.

4.10.1.13 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.

4.10.1.14 listVehicles()

```
int listVehicles (
            Vehicle * head,
            Type * headTypes )
```

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.

4.10.1.15 listVehiclesByLocation()

```
int listVehiclesByLocation (
            Vehicle * head,
            Type * headTypes,
            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.

4.10.1.16 listVehiclesByRange()

```
int listVehiclesByRange (
            Vehicle * head,
            Type * headTypes )
```

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.

4.10.1.17 readTypes()

```
Type * readTypes ( )
```

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

Returns

A pointer to a Type struct.

4.10.1.18 readVehicles()

```
Vehicle * readVehicles ( )
```

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

Returns

A pointer to a Vehicle struct.

4.10.1.19 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.

4.10.1.20 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.

4.10.1.21 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.

4.10.1.22 vehiclesMain()

```
void vehiclesMain ( )
```

Index