

## Mostrar veículos num raio

- Pedir localização do utilizador
- Pedir raio máximo
- Mostrar veículos no ponto de localização
- distor adjacentes
- Verificar se estão dentro do raio
- Ordenar por ordem crescente de distância
- Correr lista ligada e apresentar veículos presentes nos vértices adjacentes

## Caipeiro viajante

- Correr lista ligada de veículos e registrar veículos com autonomia < 50
- Guardar os veículos numa matriz
- Apresentar os veículos que necessitam de ser recolhidos
- Simplificar a lista para apenas guardar uma vez cada vértice
- Correr algoritmo dijkstra para ver vértice mais perto que é necessário ir
- Apresenta o menor caminho e retira esse vértice da lista
- Voltar a correr o algoritmo com o vértice inicial e último vértice visitado, retirar da lista e apresentar caminho
- Até lista acabar, após acabar adiciona a lista de vértices a visitar o vértice 21 que é o armazém para retroceder ao vértice inicial

## Adicionar caminhos

- Perguntar vértice inicial, vértice final e a distância
- Adicionar à lista ligada o caminho e escrever o mesmo no Ficheiro de Texto

# Data Structures

Here are the data structures with brief descriptions:

C <b>gestores</b>	Estrutura equivalente à dos clientes, apenas respectiva aos gestores dos programas
C <b>HistAluger</b>	Estrutura de registo dos alugueres, onde irão ser colocados os dados do respectivo aluguer
C <b>HistOp</b>	Estrutura de Históricos de Operação, sera registado sempre que o utilizador faça um pagamento ou uma adição de saldo
C <b>reg</b>	Estrutura onde irão ser guardados o valor do vertice mais proximo
C <b>registro</b>	Lista ligada onde é armazenada o peso dos caminhos entre vertices
C <b>sListElem</b>	Estrutura de lista ligada, no primeiro campo sera guardada a informação e no campo next será guardado o endereço do proximo elemento da lista ligada
C <b>utilizadores</b>	Estrutura de Clientes, é guardado dados como o nome, o email, o nif, o email, a morada, o saldo, a password, as permissões,a ultima operação(aluguer ou devolução) e o veiculo em questão
C <b>veiculos</b>	Estruturas de veiculos, onde é guardado o tipo, o identificador, a bateria, a autonomia, o custo do aluguer, a localização e a disponibilidade

Generated by  1.9.6

## main.c File Reference

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "11.h"
#include <locale.h>
```

## Data Structures

---

### struct gestores

Estrutura equivalente à dos clientes, apenas respectiva aos gestores dos programas. More...

### struct HistAluger

Estrutura de registo dos alugeres, onde irão ser colocados os dados do respectivo aluguer. More...

### struct HistOp

Estrutura de Históricos de Operação, será registado sempre que o utilizador faça um pagamento ou uma adição de saldo. More...

### struct reg

Estrutura onde irão ser guardados o valor do vértice mais próximo. More...

### struct registo

Lista ligada onde é armazenada o peso dos caminhos entre vértices. More...

### struct utilizadores

Estrutura de Clientes, é guardado dados como o nome, o email, o nif, o email, a morada, o saldo, a password, as permissões, a última operação (aluguer ou devolução) e o veículo em questão. More...

### struct veiculos

Estruturas de veículos, onde é guardado o tipo, o identificador, a bateria, a autonomia, o custo do aluguer, a localização e a disponibilidade. More...

## Macros

---

```
#define _CRT_SECURE_NO_WARNINGS
#define INFINITY 9999
#define MAX 40
#define VERTICES 40
```

## Typedefs

---

```
typedef struct registo * Adjacentes
```

### typedef struct utilizadores \* Clientes

Estrutura de Clientes, é guardado dados como o nome, o email, o nif, o email, a morada, o saldo, a password, as permissões, a última operação (aluguer ou devolução) e o veículo em questão.

```
typedef struct gestores * Gestores
```

Estrutura equivalente à dos clientes, apenas respectiva aos gestores dos programas.

---

**typedef struct HistAluger \*** **Historicoaluger**

Estrutura de registo dos alugeres, onde irão ser colocados os dados do respetivo aluguer.

---

**typedef struct HistOp \*** **Historicoopera**

Estrutura de Históricos de Operação, sera registado sempre que o utilizador faça um pagamento ou uma adição de saldo.

---

**typedef struct registro** **Lista**

Lista ligada onde é armazenada o peso dos caminhos entre vertices.

---

**typedef struct veiculos \*** **mobilidade**

Estruturas de veiculos, onde é guardado o tipo, o identificador, a bateria, a autonomia, o custo do aluguer, a localização e a disponibilidade.

---

**typedef struct reg \*** **Pilha**

Estrutura onde irão ser guardados o valor do vertice mais proximo.

## Functions

---

**int alterardados (void \*data, char NIF[10], char string[], int op, int perm)**

Função de alterar dados, recebe como entrada os dados da lista ligada, o nif do utilizador e a alteração. Os inteiros que recebem de seguida indicam de que tipo de operação e do tipo de utilizador.

---

**int alterardadosveiculos (void \*data, int identificador, char string[], int inteiro, float custo, int op)**

Função de alterar dados dos veiculos, recebe como variaveis de entrada da lista ligada, o identificador do veiculo a alterar, recebe uma string, um inteiro e um float consoante o valor do inteiro da operação, esta função reescreve na lista ligada a informação que foi alterada.

---

**int alterarlastopid (void \*data, char NIF[10], int numero, int op, int perm)**

Alteração dos valores da lista ligada de ultima operação e de ultimo identificador usado pelo utilizador.

---

**int altsaldo (void \*data, char NIF[10], float \*quantia, int op, int perm)**

Função de alterar saldo dentro das listas ligadas, caso seja para alterar positivamente ou negativamenete, registando esta operação na lista ligada do historico de operações Tambem regista no ficheiro de texto das operações.

---

**void apagaruser (char NIF[], int op)**

Apaga utilizador da lista ligada, seja ele cliente ou gestor, reescrevendo de seguida os ficheiros de texto com a informação.

---

**void apagarveiculo (int identificador)**

Apaga veiculo da lista ligada.

---

**void cleantxt (char txt[])**

Função para limpeza dos ficheiros txt que armazena os dados utilizados pela aplicação.

---

**int comparar (void \*data1, void \*data2)**

---

**int compararal (void \*data1, void \*data2)**

---

**int compararclientes (void \*data1, void \*data2)**

---

**int comparargestores (void \*data1, void \*data2)**

---

**int compararhist (void \*data1, void \*data2)**

---

```
int compararveiculos (void *data1, void *data2)
```

```
void converter (Adjacentes Grafo[], float Matriz[][VERTICES])
```

Função que transforma o grafo recebido por parametro, numa matriz de adjacencia, primeiramente é escrita a matriz com valores nulos e de seguida é preenchida com o peso da adjacencia de cada vertice.

---

```
void devolverlocalizacao (int id_read, int op)
```

Função de leitura de ficheiro de texto, que recebe por parametro, o id do vertice e um inteiro op, esta função irá ler o ficheiro de texto e comparar com o id de entrada da função e realizará outra função consoante o op.

---

```
int devolvermenor (float Matriz[VERTICES][2])
```

Recebe como entrada uma Matriz, onde vão estar escritos os vertices adjacentes recolhidos e os pesos, e vai retornar a sua saída o vertice em que o caminho é menor.

---

```
int dijkstra (float G[VERTICES][VERTICES], int n, int startnode)
```

Função que recebe por parametro a matriz adjacencia, o numero de vertices e o vertice inicial Verifica qual o vertice mais perto na lista de vertices ao qual é necessário e apaga o vertice que está mais perto da lista Apresenta por ultimo o caminho mais curto para o vertice mais perto.

---

```
int igual (void *data1, void *data2)
```

```
int igual_clientes (void *data1, void *data2)
```

```
void importsfromtxt ()
```

Importa dos ficheiros de texto os valores todos de utilização.

---

```
void inicializarGrafo (Adjacentes Grafo[])
```

Função que inicia o grafo, coloca todas as distancias em zero.

---

```
int inserirAresta (Adjacentes Grafo[], int v1, int v2, float p)
```

Função inicial de ler os ficheiros de texto e colocar os dados registados para dentro das listas ligadas.

---

```
void listar_ (Adjacentes Grafo[], int vertice, int raiomaximo)
```

Função que lista os vertices adjacentes presentes num grafo do vertice escolhido, após isso faz a seleção dos vertices que estão até ao raio escolhido pelo utilizador Chama depois a função que para apresentar os veiculos em.

---

```
int main ()
```

```
int mostrarveiculosnasproximidades (void *data, char localizacao[100])
```

Função escreve na consola os detalhes do veiculo, presentes na lista ligada caso estes entrem nos parametros de localização e disponibilidade.

---

```
void registrar (char nome[], char email[], char NIF[], char morada[], float saldo, char password[], int permissoes, int lastop, int lastid)
```

Função de registrar clientes e gestores, recebe os dados a registar para dentro das listas ligadas.

---

```
void registaral (char nome[50], char NIF[10], int ident, char tipo[50], char localizacao[100], float custo)
```

Função de registar aluguer na lista ligada-.

---

```
void registrarop (char nome[50], char NIF[10], char operacao[10], float quantia)
```

Regista a operação (pagamento ou carregamento) na lista ligada.

---

```
void registrarveiculo (char tipo_de_veiculo[50], int identificador, int capacidade_bateria, int autonomia, float custo_aluguer, char localizacao[100], int disponibilidade)
```

Regista dentro da lista ligada por ordem decrescente.

---

---

```
void Show (void *data, int op)
```

Função do tipo Void, recebe como variaveis de entrada os dados das listas ligadas e um inteiro, que seleciona o que vai ser mostrado na consola.

---

```
int verificarexistenciacliente (void *data, char *nif)
```

Verifica Existencia dos clientes, caso exista passa os dados da lista ligada para variaveis auxliares para posterior edição das mesmas.

---

```
int verificarexistenciagestor (void *data, char *nif)
```

Verifica a existencia do gestor na lista ligada, recebe apenas como variaveis de entrada os dados da lista ligada e o nif do utlizador em questão. Caso este esteja inserido dentro da lista ligada, copia os dados do mesmo para variaveis auxiliares para posterior alteração.

---

```
int verificarexistenciapoucabateira (void *data, int i)
```

---

```
int verificarexistenciaveiculo (void *data, int identificador)
```

Verifica a existencia de veiculo na lista ligada, copiando de seguida os dados do mesmo, se existir, para variaveis auxiliares para posterior alteração se for necessário.

---

```
int verificarlogginclientes (void *data, char email[50], char password[15])
```

Verifica loggin dos clientes, caso exista passa os dados da lista ligada para variaveis do utilizador.

---

```
int verificarloggingestores (void *data, char email[50], char password[15])
```

Função para verificar loggin dos gestores, esta função recebe como entrada a lista lgiada, os dados de e-mail e a password caso ele exista dentro da lista ligada e a password for igual a registada na lista ligada os dados do gestor são copiados para as variaveis de utilizador.

---

```
int visitado (int sequencia[], int pos, int id)
```

---

```
void writeal (char nome[50], char NIF[10], int ident, char tipo[50], char localizacao[100], float custo)
```

Escreve no ficheiro de texto o aluger que foi previamente registado na lista ligada.

---

```
void writecaminho (int vertice1, int vertice2, float distancia)
```

---

```
void writeope (char nome[50], char NIF[10], char operacao[20], float quantia)
```

Regista no ficheiro de Texto a operação que foi declarada anteriormente na lista ligada.

---

```
int writetxt (void *data, int op)
```

Função para escrever conteudo das listas ligadas em ficheiros de texto, recebe como argumentos de entrada o conteudo das listas ligadas e a operacao a executar.

---

```
void WriteuserFile (char nome[50], char email[50], char NIF[10], char morada[50], float saldo, char password[50], int permissoes, int lastop, int lastid)
```

Adiciona ao ficheiro de texto dos utilizadores o utilizador que foi registado novo.

---

```
void writeveiculo (char tipo_de_veiculo[50], int identificador, int capacidade_bateria, int autonomia, float custo_aluguer, char localizacao[100], int disponibilidade)
```

Escreve o meio no txt respectivo.

---

## Variables

---

```
int autonomia_AUX = 0
```

---

```
int capacidade_bateria_AUX = 0
```

---

```
ListElem ClientList = NULL
```

Declaração das listas ligadas.

---

```
float custo_aluguer_AUX = 0
```

```

char email_AUX [100] = ""
char email_utilizador [100] = ""
char FileLocal [] = "Documentos/Locais.txt"
char FileMeios [] = "Documentos/MeiosFile.txt"
char FileRout [] = "Documentos/RoutFile.txt"
ListElem GestorList = NULL
ListElem HistAlug = NULL
char HistAluger [] = "Documentos/HistAluger.txt"
char HistOper [] = "Documentos/HistOpera.txt"
ListElem HistOpera = NULL
int id_AUX
int lastid
int lastid_AUX
int lastop_AUX
int lastoperacao
char localizacao_AUX [100] = ""
float Matriz [VERTICES][VERTICES]
ListElem MeiosList = NULL
char morada_AUX [100] = ""
char morada_utilizador [100] = ""
char NIF_AUX [10] = ""
char NIF_utilizador [10] = ""
char nome_AUX [50] = ""
char password_AUX [100] = ""
char password_utilizador [100] = ""
int permissões_AUX = 0
int permissões_utilizador = 0
float saldo_AUX = 0.00
float saldo_utilizador = 0.00
char tipo_AUX [50]
char UserFile [] = "Documentos/UsersFile.txt"
char utilizador [50] = ""
int Veiculoscompoocabateria [100][3]
int vertices [VERTICES]

```

## Macro Definition Documentation

◆ \_CRT\_SECURE\_NO\_WARNINGS

```
#define _CRT_SECURE_NO_WARNINGS
```

## ◆ INFINITY

```
#define INFINITY 9999
```

## ◆ MAX

```
#define MAX 40
```

## ◆ VERTICES

```
#define VERTICES 40
```

# Typedef Documentation

## ◆ Adjacentes

```
typedef struct registro * Adjacentes
```

## ◆ Clientes

```
typedef struct utilizadores* Clientes
```

Estrutura de Clientes, é guardado dados como o nome, o email, o nif, o email, a morada, o saldo, a password, as permissões,a ultima operação(aluguer ou devolução) e o veiculo em questão.

## ◆ Gestores

```
typedef struct gestores* Gestores
```

Estrutura equivalente à dos clientes, apenas respectiva aos gestores dos programas.

## ◆ Historicoaluguer

```
typedef struct HistAluger* Historicoaluger
```

Estrutura de registo dos alugeres, onde irão ser colocados os dados do respectivo aluguer.

///

### ◆ Historicopera

```
typedef struct HistOp* Historicopera
```

Estrutura de Históricos de Operação, será registado sempre que o utilizador faça um pagamento ou uma adição de saldo.

### ◆ Lista

```
typedef struct registo Lista
```

Lista ligada onde é armazenada o peso dos caminhos entre vértices.

### ◆ mobilidade

```
typedef struct veiculos* mobilidade
```

Estruturas de veículos, onde é guardado o tipo, o identificador, a bateria, a autonomia, o custo do aluguer, a localização e a disponibilidade.

### ◆ Pilha

```
typedef struct reg * Pilha
```

Estrutura onde irão ser guardados o valor do vértice mais próximo.

## Function Documentation

### ◆ alterardados()

```
int alterardados ( void * data,
                    char   NIF[10],
                    char   string[],
                    int    op,
                    int    perm
                )
```

Função de alterar dados, recebe como entrada os dados da lista ligada, o nif do utilizador e a alteração. Os inteiros que recebem de seguida indicam de que tipo de operação e do tipo de utilizador.

#### Parameters

**data**

**NIF**

**string**

**op**

**perm**

#### Returns

### ◆ alterardadosveiculos()

```
int alterardadosveiculos ( void * data,
                           int     identificador,
                           char   string[],
                           int     inteiro,
                           float   custo,
                           int     op
)
```

Função de alterar dados dos veículos, recebe como variáveis de entrada da lista ligada, o identificador do veículo a alterar, recebe uma string, um inteiro e um float consoante o valor do inteiro da operação, esta função reescreve na lista ligada a informação que foi alterada.

#### Parameters

- data**
- identificador**
- string**
- inteiro**
- custo**
- op**

#### Returns

### ◆ alterarlastopid()

```
int alterarlastopid ( void * data,
                      char   NIF[10],
                      int    numero,
                      int    op,
                      int    perm
)
```

Alteração dos valores da lista ligada de ultima operação e de ultimo identificador usado pelo utilizador.

#### Parameters

- data**
- NIF**
- numero**
- op**
- perm**

#### Returns

## ◆ altsaldo()

```
int altsaldo ( void * data,  
               char   NIF[10],  
               float * quantia,  
               int    op,  
               int    perm  
             )
```

Função de alterar saldo dentro das listas ligadas, caso seja para alterar positivamente ou negativamente, registando esta operação na lista ligada do histórico de operações. Também regista no ficheiro de texto das operações.

### Parameters

**data**  
**NIF**  
**quantia**  
**op**  
**perm**

### Returns

## ◆ apagaruser()

```
void apagaruser ( char NIF[],  
                  int   op  
                )
```

Apaga utilizador da lista ligada, seja ele cliente ou gestor, reescrevendo de seguida os ficheiros de texto com a informação.

### Parameters

**NIF**  
**op**

## ◆ apagarveiculo()

```
void apagarveiculo ( int identificador )
```

Apaga veiculo da lista ligada.

**Parameters**

identificador

**◆ cleantxt()**

```
void cleantxt ( char txt[] )
```

Função para limpeza dos ficheiros txt que armazenam os dados utilizados pela aplicação.

**Parameters**

txt

**◆ comparar()**

```
int comparar ( void * data1,  
                void * data2  
            )
```

**◆ compararal()**

```
int compararal ( void * data1,  
                  void * data2  
              )
```

**◆ compararclientes()**

```
int compararclientes ( void * data1,  
                       void * data2  
                   )
```

**◆ comparargestores()**

```
int comparargestores ( void * data1,  
                      void * data2  
)
```

### ◆ compararhist()

```
int compararhist ( void * data1,  
                   void * data2  
)
```

### ◆ compararveiculos()

```
int compararveiculos ( void * data1,  
                       void * data2  
)
```

### ◆ converter()

```
void converter ( Adjacentes Grafo[],  
                 float          Matriz[][][VERTICES]  
)
```

Função que transforma o grafo recebido por parametro, numa matriz de adjacencia, primeiramente é escrita a matriz com valores nulos e de seguida é preenchida com o peso da adjacencia de cada vertice.

#### Parameters

**Grafo**

Grafo a ser convertido

#### Parameters

**Matriz**

Matriz na qual é escrita a matriz adjacencia

### ◆ devolverlocalizacao()

```
void devolverlocalizacao ( int id_read,  
                           int op  
                         )
```

Função de leitura de ficheiro de texto, que recebe por parametro, o id do vertice e um inteiro op, esta função irá ler o ficheiro de texto e comparar com o id de entrada da função e realizará outra função consoante o op.

#### Parameters

**id\_read**

ID de descodificação do vertice

#### Parameters

**op**

Simbolo da operação que realizará após leitura do vertice

### ◆ devolvermenor()

```
int devolvermenor ( float Matriz[VERTICES][2] )
```

Recebe como entrada uma Matriz, onde vão estar escritos os vertices adjacentes recolhidos e os pesos, e vai retornar a sua saída o vertice em que o caminho é menor.

#### Parameters

**Matriz**

#### Returns

Retorna vertice que o caminho é menor

### ◆ dijkstra()

```
int dijkstra ( float G[VERTICES][VERTICES],  
    int n,  
    int startnode  
)
```

Função que recebe por parametro a matriz adjacencia, o numero de vertices e o vertice inicial. Verifica qual o vertice mais perto na lista de vertices ao qual é necessário e apaga o vertice que está mais perto da lista. Apresenta por ultimo o caminho mais curto para o vertice mais perto.

#### Parameters

**G**

Matriz adjacencia

#### Parameters

**n**

Numero de Vertices

#### Parameters

**startnode**

Vertice inicial para verificar o mais perto

#### Returns

#### ◆ igual()

```
int igual ( void * data1,  
    void * data2  
)
```

#### ◆ igual\_clientes()

```
int igual_clientes ( void * data1,  
    void * data2  
)
```

#### ◆ importsfromtxt()

```
void importsfromtxt ( )
```

Importa dos ficheiros de texto os valores todos de utilização.

### ◆ inicializarGrafo()

```
void inicializarGrafo ( Adjacentes Grafo[] )
```

Função que inicia o grafo, coloca todas as distancias em zero.

#### Parameters

**Grafo**

Grafo a ser inicializado

### ◆ inserirAresta()

```
int inserirAresta ( Adjacentes Grafo[],  
                    int          v1,  
                    int          v2,  
                    float        p  
)
```

Função inicial de ler os ficheiros de texto e colocar os dados registados para dentro das listas ligadas.

**Parameters****Users****meios**

Função para registo de arestas do grafo na lista ligada

**Parameters****Grafo**

Grafo ao qual vai ser adicionado a aresta

**Parameters****v1**

vertice inicial

**Parameters****v2**

vertice final

**Parameters****p**

distancia do caminho

**Returns**

◆ **listar\_()**

```
void listar_ ( Adjacentes Grafo[],  
              int         vertice,  
              int         raiomaximo  
            )
```

Função que lista os vertices adjacentes presentes num grafo do vertice escolhido, após isso faz a seleção dos vertices que estão até ao raio escolhido pelo utilizador Chama depois a função que para apresentar os veículos em.

#### Parameters

**Grafo**

Grafo a procurar vertices adjacentes

#### Parameters

**vertice**

Vertice ao qual vamos procurar os adjacentes

#### Parameters

**raiomaximo**

Raio máximo admitida percorrer pelo cliente

#### ◆ main()

```
int main ( )
```

#### ◆ mostrarveiculosnasproximidades()

```
int mostrarveiculosnasproximidades ( void * data,
                                         char   localizacao[100]
                                         )
```

Função escreve na consola os detalhes do veiculo, presentes na lista ligada caso estes entrem nos parametros de localização e disponibilidade.

#### Parameters

**data**

Espaço de memoria onde esta alocada a informação relativa ao veiculo

#### Parameters

**localizacao**

localização para a qual tem de coincidir os veiculos para serem apresentados

#### Returns

◆ **registar()**

```
void registrar ( char nome[],  
                 char email[],  
                 char NIF[],  
                 char morada[],  
                 float saldo,  
                 char password[],  
                 int permissoes,  
                 int lastop,  
                 int lastid  
)
```

Função de registar clientes e gestores, recebe os dados a registar para dentro das listas ligadas.

#### Parameters

- nome**
- email**
- NIF**
- morada**
- saldo**
- password**
- permissoes**
- lastop**
- lastid**

◆ **registrar()**

```
void registral ( char nome[50],  
                 char NIF[10],  
                 int ident,  
                 char tipo[50],  
                 char localizacao[100],  
                 float custo  
             )
```

Função de registar aluger na lista ligada-.

#### Parameters

**nome**  
**NIF**  
**ident**  
**tipo**  
**localizacao**  
**custo**

### ◆ registrarop()

```
void registrarop ( char nome[50],  
                  char NIF[10],  
                  char operacao[10],  
                  float quantia  
                )
```

Regista a operação (pagamento ou carregamento) na lista ligada.

#### Parameters

**nome**  
**NIF**  
**operacao**  
**quantia**

### ◆ registrarveiculo()

```
void registrarveiculo ( char tipo_de_veiculo[50],  
                        int identificador,  
                        int capacidade_bateria,  
                        int autonomia,  
                        float custo_aluguer,  
                        char localizacao[100],  
                        int disponibilidade  
)
```

Regista dentro da lista ligada por ordem decrescente.

#### Parameters

- tipo\_de\_veiculo**
- identificador**
- capacidade\_bateria**
- autonomia**
- custo\_aluguer**
- localizacao**
- disponibilidade**

### ◆ Show()

```
void Show ( void * data,  
            int op  
)
```

Função do tipo Void, recebe como variaveis de entrada os dados das listas ligadas e um inteiro, que seleciona o que vai ser mostrado na consola.

#### Parameters

- data**
- op**

### ◆ verificarexistenciacliente()

```
int verificarexistenciacliente ( void * data,  
                                char * nif  
                                )
```

Verifica Existencia dos clientes, caso exista passa os dados da lista ligada para variaveis auxiliares para posterior edição das mesmas.

#### Parameters

data  
nif

#### Returns

### ◆ verificarexistenciagestor()

```
int verificarexistenciagestor ( void * data,  
                               char * nif  
                               )
```

Verifica a existencia do gestor na lista ligada, recebe apenas como variaveis de entrada os dados da lista ligada e o nif do utilizador em questão. Caso este esteja inserido dentro da lista ligada, copia os dados do mesmo para variaveis auxiliares para posterior alteração.

#### Parameters

data  
nif

#### Returns

### ◆ verificarexistenciapoucabateira()

```
int verificarexistenciapoucabateira ( void * data,  
                                      int      i  
                                      )
```

### ◆ verificarexistenciaveiculo()

```
int verificarexistenciateveiculo ( void * data,  
                                int      identificador  
                                )
```

Verifica a existencia de veiculo na lista ligada, copiando de seguida os dados do mesmo, se existir, para variaveis auxiliares para posterior alteração se for necessário.

**Parameters**

**data**  
**identificador**

**Returns****◆ verificarlogginclientes()**

```
int verificarlogginclientes ( void * data,  
                            char   email[50],  
                            char   password[15]  
                            )
```

Verifica loggin dos clientes, caso exista passa os dados da lista ligada para variaveis do utilizador.

**Parameters**

**data**  
**email**  
**password**

**Returns****◆ verificarloggingestores()**

```
int verificarloggingestores ( void * data,  
                           char email[50],  
                           char password[15]  
                         )
```

Função para verificar loggin dos gestores, esta função recebe como entrada a lista ligada, os dados de e-mail e a password caso ele exista dentro da lista ligada e a password for igual a registada na lista ligada os dados do gestor são copiados para as variaveis de utilizador.

#### Parameters

**data**  
**email**  
**password**

#### Returns

### ◆ visitado()

```
int visitado ( int sequencia[],  
               int pos,  
               int id  
             )
```

#### Parameters

**sequencia**  
**pos**  
**id**

#### Returns

### ◆ writeal()

```
void writeal ( char nome[50],  
               char NIF[10],  
               int ident,  
               char tipo[50],  
               char localizacao[100],  
               float custo  
 )
```

Escreve no ficheiro de texto o aluger que foi previamente registado na lista ligada.

#### Parameters

**nome**  
**NIF**  
**ident**  
**tipo**  
**localizacao**  
**custo**

#### ◆ writecaminho()

```
void writecaminho ( int vertice1,  
                    int vertice2,  
                    float distancia  
 )
```

#### ◆ writeope()

```
void writeope ( char nome[50],  
                char NIF[10],  
                char operacao[20],  
                float quantia  
        )
```

Regista no ficheiro de Texto a operação que foi declarada anteriormente na lista ligada.

#### Parameters

**nome**  
**NIF**  
**operacao**  
**quantia**

### ◆ writetxt()

```
int writetxt ( void * data,  
              int     op  
        )
```

Função para escrever conteúdo das listas ligadas em ficheiros de texto, recebe como argumentos de entrada o conteúdo das listas ligadas e a operacao a executar.

#### Parameters

**data**  
**op**

### ◆ WriteuserFile()

```
void WriteuserFile ( char nome[50],  
                     char email[50],  
                     char NIF[10],  
                     char morada[50],  
                     float saldo,  
                     char password[50],  
                     int permissoes,  
                     int lastop,  
                     int lastid  
)
```

Adiciona ao ficheiro de texto dos utilizadores o utilizador que foi registado novo.

#### Parameters

**nome**  
**email**  
**NIF**  
**morada**  
**saldo**  
**password**  
**permissoes**  
**lastop**  
**lastid**

#### ◆ writeveiculo()

```
void writeveiculo ( char tipo_de_veiculo[50],  
                    int identificador,  
                    int capacidade_bateria,  
                    int autonomia,  
                    float custo_aluguer,  
                    char localizacao[100],  
                    int disponibilidade  
    )
```

Escreve o meio no txt respectivo.

#### Parameters

**tipo\_de\_veiculo**

**identificador**

**capacidade\_bateria**

**autonomia**

**custo\_aluguer**

**localizacao**

**disponibilidade**

## Variable Documentation

---

### ◆ autonomia\_AUX

```
int autonomia_AUX = 0
```

### ◆ capacidade\_bateria\_AUX

```
int capacidade_bateria_AUX = 0
```

### ◆ ClientList

```
ListElem ClientList = NULL
```

Declaração das listas ligadas.

### ◆ custo\_aluger\_AUX

```
float custo_aluger_AUX = 0
```

### ◆ email\_AUX

```
char email_AUX[100] = ""
```

### ◆ email\_utilizador

```
char email_utilizador[100] = ""
```

### ◆ FileLocal

```
char FileLocal[] = "Documentos/Locais.txt"
```

### ◆ FileMeios

```
char FileMeios[] = "Documentos/MeiosFile.txt"
```

### ◆ FileRout

```
char FileRout[] = "Documentos/RoutFile.txt"
```

### ◆ GestorList

```
ListElem GestorList = NULL
```

### ◆ HistAlug

```
ListElem HistAlug = NULL
```

### ◆ HistAluger

```
char HistAluger[] = "Documentos/HistAluger.txt"
```

### ◆ HistOper

```
char HistOper[] = "Documentos/HistOpera.txt"
```

### ◆ HistOpera

```
ListElem HistOpera = NULL
```

### ◆ id\_AUX

```
int id_AUX
```

### ◆ lastid

```
int lastid
```

### ◆ lastid\_AUX

```
int lastid_AUX
```

### ◆ lastop\_AUX

```
int lastop_AUX
```

### ◆ lastoperacao

```
int lastoperacao
```

### ◆ localizacao\_AUX

```
char localizacao_AUX[100] = ""
```

### ◆ Matriz

```
float Matriz[VERTICES][VERTICES]
```

### ◆ MeiosList

```
ListElem MeiosList = NULL
```

### ◆ morada\_AUX

```
char morada_AUX[100] = ""
```

### ◆ morada\_utilizador

```
char morada_utilizador[100] = ""
```

### ◆ NIF\_AUX

```
char NIF_AUX[10] = ""
```

### ◆ NIF\_utilizador

```
char NIF_utilizador[10] = ""
```

### ◆ nome\_AUX

```
char nome_AUX[50] = ""
```

### ◆ password\_AUX

```
char password_AUX[100] = ""
```

### ◆ password\_utilizador

```
char password_utilizador[100] = ""
```

### ◆ permissões\_AUX

```
int permissões_AUX = 0
```

### ◆ permissões\_utilizador

```
int permissões_utilizador = 0
```

### ◆ saldo\_AUX

```
float saldo_AUX = 0.00
```

### ◆ saldo\_utilizador

```
float saldo_utilizador = 0.00
```

### ◆ tipo\_AUX

```
char tipo_AUX[50]
```

### ◆ UserFile

```
char UserFile[] = "Documentos/UsersFile.txt"
```

### ◆ utilizador

```
char utilizador[50] = ""
```

### ◆ Veiculoscompoucabateria

```
int Veiculoscompoucabateria[100][3]
```

## ◆ vertices

```
int vertices[VERTICES]
```

## II.c File Reference

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ll.h"
```

### Macros

---

```
#define _CRT_SECURE_NO_WARNINGS
```

---

### Functions

---

```
ListElem addItemLastIterative (ListElem head, void *data)
ListElem addItemOrderedIterative (ListElem head, void *data, int(*compare)(void *data1, void *data2))
    int alterar (ListElem head, char NIF[10], char string[], int op, int perm, int(*alterar)(void *data, char
        nif[10], char string[], int op, int perm))
    int alteraraluguer (ListElem head, char NIF[10], int numero, int op, int perm, int(*alterar)(void *data,
        char nif[10], int numero, int op, int perm))
    int alterarsaldo (ListElem head, char NIF[10], float *saldo, int op, int perm, int(*addsaldo)(void *data,
        char nif[10], float *quantia, int op, int perm))
    int alterarveiculos (ListElem head, int identificador, char string[], int inteiro, float custo, int op,
        int(*alterar)(void *data, int identificador, char string[], int inteiro, float custo, int op))
    int loggin (ListElem head, char email[50], char password[15], int(*verificar)(void *data, char email[50],
        char password[15]))
    int menuinicial ()
void menulocalizacao (char FileLocal[])
ListElem removeItemIterative (ListElem head, void *data, int(*compare)(void *data1, void *data2))
    int rewritefile (ListElem head, int op, int(*write)(void *data, int op))
    void showListIterative (ListElem head, int op, void(*show)(void *data, int op))
    int verificarmeios (ListElem head, int id, int(*verificar)(void *data, int id))
    void verificarpoucabit (ListElem head, int(*verificar)(void *data, int i))
    int verificarproximidade (ListElem head, char localizacao[100], int(*verificar)(void *data, char
        localizacao[100]))
    int verificarusers (ListElem head, char num[10], int(*verificar)(void *data, char nif[10]))
```

---

### Macro Definition Documentation

---

#### ◆ \_CRT\_SECURE\_NO\_WARNINGS

```
#define _CRT_SECURE_NO_WARNINGS
```

## Function Documentation

### ◆ addItemLastIterative()

```
ListElem addItemLastIterative ( ListElem head,
                                void *      data
                               )
```

### ◆ addItemOrderedIterative()

```
ListElem addItemOrderedIterative ( ListElem head,
                                   void *      data,
                                   int (*)(void *data1, void *data2) compare
                                 )
```

### ◆ alterar()

```
int alterar ( ListElem head,
              char      NIF[10],
              char      string[],
              int       op,
              int       perm,
              int (*)(void *data, char nif[10], char string[], int op, int perm) alterar
            )
```

### ◆ alteraraluger()

```
int alteraraluger ( ListElem head,
                     char      NIF[10],
                     int       numero,
                     int       op,
                     int       perm,
                     int (*)(void *data, char nif[10], int numero, int op, int perm) alterar
                   )
```

### ◆ alterarsaldo()

```
int alterarsaldo ( ListElem
    char head,
    float * NIF[10],
    int saldo,
    int op,
    int perm,
    int(*)(void *data, char nif[10], float *quantia, int op, int perm) addsaldo
)
)
```

### ◆ alterarveiculos()

```
int alterarveiculos ( ListElem
    int head,
    char identificador,
    int string[],
    int inteiro,
    float custo,
    int op,
    int(*)(void *data, int identificador, char string[], int inteiro, float custo, int op) alterar
)
)
```

### ◆ loggin()

```
int loggin ( ListElem
    char head,
    char email[50],
    char password[15],
    int(*)(void *data, char email[50], char password[15]) verificar
)
)
```

### ◆ menuinicial()

```
int menuinicial ( )
```

### ◆ menulocalizacao()

```
void menulocalizacao ( char FileLocal[] )
```

### ◆ removeItemIterative()

```
ListElem removeItemIterative ( ListElem head,  
                               void * data,  
                               int (*)(void *data1, void *data2) compare  
)
```

### ◆ rewritefile()

```
int rewritefile ( ListElem head,  
                  int op,  
                  int (*)(void *data, int op) write  
)
```

### ◆ showListIterative()

```
void showListIterative ( ListElem head,  
                        int op,  
                        void (*)(void *data, int op) show  
)
```

### ◆ verificarmeios()

```
int verificarmeios ( ListElem head,  
                     int id,  
                     int (*)(void *data, int id) verificar  
)
```

### ◆ verificarpoucabat()

```
void verificarpoucabat ( ListElem head,  
                         int (*)(void *data, int i) verificar  
)
```

### ◆ verificarproximidade()

```
int verificarproximidade ( ListElem head,  
                           char localizacao[100],  
                           int(*)(void *data, char localizacao[100]) verificar  
                         )
```

#### ◆ **verificarusers()**

```
int verificarusers ( ListElem head,
                     char num[10],
                     int(*)(void *data, char nif[10]) verificar
                 )
```

## II.h File Reference

---

Go to the source code of this file.

### Data Structures

---

**struct sListElem**

Estrutura de lista ligada, no primeiro campo sera guardada a informação e no campo next será guardado o endereço do proximo elemento da lista ligada. More...

### Typedefs

---

**typedef struct sListElem \* ListElem**

Estrutura de lista ligada, no primeiro campo sera guardada a informação e no campo next será guardado o endereço do proximo elemento da lista ligada.

**typedef struct sListElem SListElem**

### Functions

---

**ListElem addItemLastIterative (ListElem head, void \*data)**

**ListElem addItemOrderedIterative (ListElem head, void \*data, int(\*compare)(void \*data1, void \*data2))**

**int alterar (ListElem head, char NIF[10], char string[], int op, int perm, int(\*alterar)(void \*data, char nif[10], char string[], int op, int perm))**

**int alteraraluger (ListElem head, char NIF[10], int numero, int op, int perm, int(\*alterar)(void \*data, char nif[10], int numero, int op, int perm))**

**int alterarsaldo (ListElem head, char NIF[10], float \*saldo, int op, int perm, int(\*addsaldo)(void \*data, char nif[10], float \*quantia, int op, int perm))**

**int alterarveiculos (ListElem head, int identificador, char string[], int inteiro, float custo, int op, int(\*alterar)(void \*data, int identificador, char string[], int inteiro, float custo, int op))**

**int menuinicial ()**

**ListElem removeltemIterative (ListElem head, void \*data, int(\*compare)(void \*data1, void \*data2))**

**int rewritefile (ListElem head, int op, int(\*write)(void \*data, int op))**

**void showListIterative (ListElem head, int op, void(\*show)(void \*data, int op))**

**int verificarproximidade (ListElem head, char localizacao[100], int(\*verificar)(void \*data, char localizacao[100]))**

**int verificarusers (ListElem head, char num[10], int(\*verificar)(void \*data, char nif[10]))**

### Typedef Documentation

---

◆ **ListElem**

```
typedef struct sListElem * ListElem
```

Estrutura de lista ligada, no primeiro campo sera guardada a informação e no campo next será guardado o endereço do proximo elemento da lista ligada.

### ◆ SListElem

```
typedef struct sListElem SListElem
```

## Function Documentation

### ◆ addItemLastIterative()

```
ListElem addItemLastIterative ( ListElem head,
                                void *      data
                               )
```

### ◆ addItemOrderedIterative()

```
ListElem addItemOrderedIterative ( ListElem           head,
                                   void *            data,
                                   int (*)(void *data1, void *data2) compare
                                 )
```

### ◆ alterar()

```
int alterar ( ListElem           head,
              char             NIF[10],
              char             string[],
              int              op,
              int              perm,
              int (*)(void *data, char nif[10], char string[], int op, int perm) alterar
            )
```

### ◆ alteraraluger()

```
int alteraraluguer ( ListElem
    char
    int
    int
    int
    int(*)(void *data, char nif[10], int numero, int op, int perm) alterar
)
```

### ◆ alterarsaldo()

```
int alterarsaldo ( ListElem
    char
    float *
    int
    int
    int(*)(void *data, char nif[10], float *quantia, int op, int perm) addsaldo
)
```

### ◆ alterarveiculos()

```
int alterarveiculos ( ListElem
    int
    char
    int
    float
    int
    int(*)(void *data, int identificador, char string[], int inteiro, float custo, int op) alterar
)
```

### ◆ menuinicial()

```
int menuinicial ( )
```

### ◆ removeItemIterative()

```
ListElem removeItemIterative ( ListElem head,
                               void * data,
                               int (*)(void *data1, void *data2) compare
)
```

### ◆ rewritefile()

```
int rewritefile ( ListElem head,
                  int op,
                  int (*)(void *data, int op) write
)
```

### ◆ showListIterative()

```
void showListIterative ( ListElem head,
                        int op,
                        void (*)(void *data, int op) show
)
```

### ◆ verificarproximidade()

```
int verificarproximidade ( ListElem head,
                           char localizacao[100],
                           int (*)(void *data, char localizacao[100]) verificar
)
```

### ◆ verificarusers()

```
int verificarusers ( ListElem head,
                     char num[10],
                     int (*)(void *data, char nif[10]) verificar
)
```