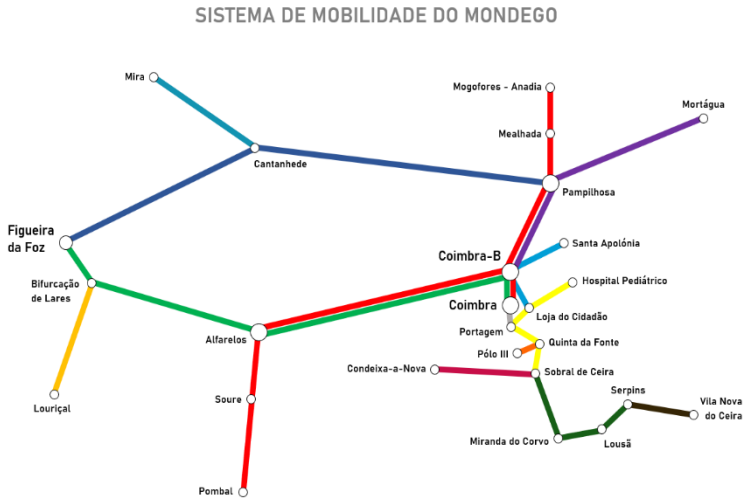
Trabalho Prático

Projeto: *Metro Mondego*



LEI – 2022/23

Unidade Curricular: Programação

Docente: Prof. Francisco José Baptista Pereira

Autor: Pedro Pereira – a2021130905

ISEC ENGENHARIA

CLion

Índice

1. Introdução

Este trabalho tem como objetivo a criação de um programa desenvolvido na linguagem de programação C que permita a gestão de uma rede de transportes composta por várias linhas e paragens, sendo usado o exemplo do Metro Mondego. Foi utilizado o CLion como IDE.

O programa tem uma interface simples em que inicialmente é apresentado um menu com as várias opções disponíveis sendo que existem três grupos principais: “Paragens”, “Linhas” e “Outros”.

No grupo “Paragens” tal como o nome indica é possível fazer a gestão do que tem a ver com as paragens do sistema de mobilidade. É possível registar uma nova paragem, eliminar uma paragem já registada e visualizar a lista de paragens registadas.

No grupo “Linhas” as funcionalidades são semelhantes às paragens, permitindo gerir as paragens que pertencem a cada linha, seja removendo ou adicionando novas paragens.

Por fim, no grupo “Outros” tem as opções de mostrar o percurso entre duas paragens e a opção de terminar o programa.

2. Desenvolvimento

2.1 Estruturas

Este programa tem duas estruturas essenciais para o seu funcionamento, a estrutura ‘Paragem’ e a estrutura ‘Linha’.

A estrutura ‘Paragem’ é necessária para armazenar o nome, código alfanumérico e quantidade de linhas a que pertence cada paragem que for registada. Cada estrutura ‘Paragem’ vai fazer parte de um *array* de estruturas dinâmico.

```
typedef struct paragem Paragem;

struct paragem{
    char nome[50];
    char codigo[5];
    int quantLinhas;
};
```

Figura 1 – Estrutura do tipo Paragem
(está no ficheiro paragens.h)

A estrutura ‘Linha’ é essencial para guardar o nome da linha, número de paragens que pertencem a essa linha, para além dos campos relacionados com a gestão de cada estrutura que são o ponteiro para a primeira paragem da linha e o ponteiro para a próxima linha/nó da estrutura dinâmica do tipo lista ligada.

```
typedef struct linha Linha;       // struct linha <==> Linha

struct linha {
    char nome[50];
    int numParagens;
    Paragem* primeiraParagem;         // ponteiro para a primeira paragem da linha
    struct linha* proximaLinha;       // ponteiro para a próxima linha/nó
};
```

Figura 2 – Estrutura do tipo Linha
(está no ficheiro linhas.h)

2.2 Ficheiros

São utilizados dois tipos diferentes de ficheiros, **ficheiro binário** e **ficheiro de texto**.

O **ficheiro binário** guarda a informação das paragens (nome, código alfanumérico e número de linhas a que a paragem pertence) e das linhas para que seja possível reconstruir as estruturas dinâmicas, *array* de estruturas dinâmico e estrutura dinâmica do tipo lista ligada, quando o programa retomar a execução.

O **ficheiro de texto** vai conter a informação relativa a uma linha (nome, número de paragens e as respetivas paragens identificadas através do seu nome e código alfanumérico) e servirá para que seja adicionada uma nova linha sem que seja necessário o *input* do nome da linha e das paragens por parte do utilizador.

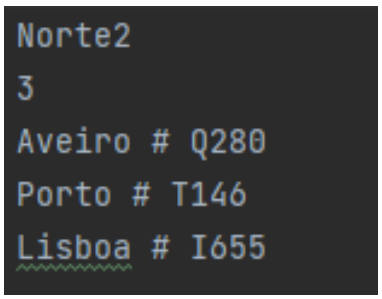


Figura 3 – Exemplo da informação de um ficheiro de texto

2.3 Funções

O programa é constituído por várias funções sendo que a maioria delas é para satisfazer os requisitos do programa, mas existem também funções que são usadas por outras para tornar o código mais fluído, claro e limpo.

Essas funções “auxiliares” estão presentes no arquivo **utils.c** e respetivo header **utils.h**.

2.3.1 Ficheiros paragens.c / paragens.h

No ficheiro **paragens.c** e respetivo header **paragens.h** é onde estão as funções que permitem a gestão das paragens.

- **void registaParagem(Paragem** paragens, int* tamanho_array, int* capacidade_array)**
  - Função que permite ao utilizador adicionar uma nova paragem ao array dinâmico de paragens

- **void eliminaParagem(Paragem** paragens, int* tamanho_array)**
  - Função que permite eliminar uma paragem específica do array dinâmico de paragens
  - O utilizador tem de introduzir o código alfanumérico correspondente à paragem que pretende eliminar
  - Caso a paragem pertença a uma ou mais linhas registadas, então não pode ser eliminada. Tem de ser removida primeiro da(s) linha(s) para ser possível eliminar

- **void visualizaParagem(Paragem* paragens, int tamanho_array)**
  - Função que permite visualizar as informações de todas as paragens presentes no array dinâmico de paragens

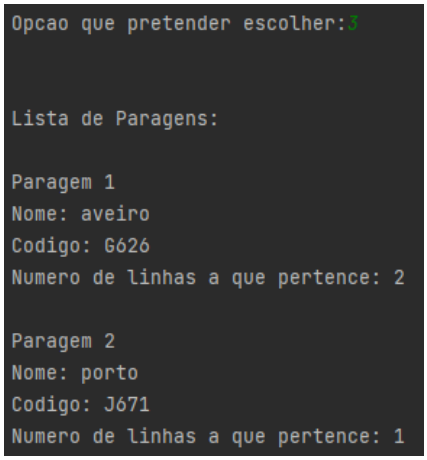


Figura 4 – Exemplo de uma lista de paragens

2.3.2 Ficheiros linhas.c / linhas.h

No ficheiro **linhas.c** e respetivo header **linhas.h** é onde estão as funções que permitem a gestão das linhas e as suas respetivas paragens através da manipulação da lista ligada das linhas.

- **void adicionaLinha(struct linha** listaLinhas, Paragem* paragens, int tamanho_array)**
  - Função que permite adicionar uma nova linha à lista ligada da lista de linhas

  - Tem duas opções:
    - 1 - Adicionar linha através de input
    - 2 - Adicionar linha através de ficheiro de texto

```
Opcao que pretender escolher:4

1 - Adicionar linha atraves de input
2 - Adicionar linha atraves de ficheiro de texto
```

Figura 5 – Escolha do tipo de método para adicionar linha

  - A segunda opção permite ao utilizador adicionar uma linha ao programa através de um ficheiro de texto que contenha o nome da linha, número de paragens e respetivos nomes e códigos alfanuméricos. Apenas necessita de introduzir o nome da linha no formato 'nome.txt' e o programa verifica se existe um ficheiro de texto com o nome introduzido.

- A linha não é adicionada quando:
  - Não existe um ficheiro de texto com esse nome
  - Existe uma linha já registada com o nome da linha do ficheiro de texto
  - Uma ou mais paragens da linha do ficheiro de texto não estão registadas no programa

- **void atualizaLinha(struct linha* listaLinhas, Paragem* paragens, int tamanho_array)**
  - Função que possibilita ao utilizador atualizar as paragens das linhas

  - Tem duas opções:
    - 1 – Adicionar paragem
    - 2 – Remover paragem

```
Opcao que pretender escolher:5
 Insira o nome da linha que deseja atualizar:norte

1 - Adicionar paragem
2 - Remover paragem
```

Figura 6 – Escolha do tipo de operação na linha (exemplo usando a linha 'norte')

- **void visualizaLinhas(struct linha* listaLinhas, Paragem* listaParagens)**
  - Função que tem como objetivo visualizar informações acerca das linhas e/ou das paragens associadas a linhas

  - Tem duas opções:

- 1 – Ver lista completa de linhas

```
Opcao que pretender escolher:0

1 - Ver lista completa de linhas
2 - Ver linhas que passam numa determinada paragem

Opcao que pretender escolher:1

Lista de linhas:
----------------------
Nome: norte

Paragem 1: porto
Paragem 2: aveiro
----------------------
```

Figura 7 – Exemplo de uma lista de linhas

- 2 – Ver linhas que passam numa determinada paragem

```
Opcao que pretender escolher:2
 Insira o nome da paragem:aveiro

Linhas que passam pela paragem 'aveiro':
------------------------
Nome: norte

Paragem 1: porto
Paragem 2: aveiro
------------------------
Nome: centro

Paragem 1: aveiro
Paragem 2: coimbra
------------------------
```

Figura 8 – Exemplo de uma lista de linhas que passam por uma determinada paragem (exemplo para a paragem 'aveiro')

2.3.2 Ficheiros utils.c / utils.h

No ficheiro **utils.c** e respetivo header **utils.h** é onde estão as funções auxiliares que estão presentes noutras funções e que servem para simplificar o código nos restantes ficheiros de código.

- **int comparaNome(Paragem* paragens, int tamanho_array, char* nome)**
  - Função que serve para fazer a validação do nome introduzido pelo utilizador quando tenta adicionar uma nova paragem
  - Compara esse nome com todos os nomes das paragens registadas para evitar duplicação de paragens com o mesmo nome
  - É usada no ficheiro **paragens.c**

- **void gerarCodigo(char* codigo)**
  - Função que gera um código alfanumérico aleatório de quatro caracteres (uma letra maiúscula e três números)
  - Esse código vai ser atribuído a uma nova paragem (é único para essa paragem)

- **int comparaCodigo(Paragem* paragens, int tamanho_array, char* codigo)**
  - Função que serve para fazer a validação do código alfanumérico gerado pela função **gerarCodigo**
  - Compara esse código gerado com todos os códigos das paragens registadas para evitar duplicação de paragens com o mesmo código
  - É usada no ficheiro **paragens.c**

- **int comparaNomeLinhas(struct linha* listaLinhas, char* nome)**
  - Função que serve para fazer a validação do nome introduzido pelo utilizador quando tenta adicionar uma nova linha
  - Compara esse nome com todos os nomes das linhas registadas para evitar duplicação de linhas com o mesmo nome
  - É usada no ficheiro **linhas.c**

- **void cortaEspacos(char *nomeP)**
  - Função que remove espaços em branco de uma string, seja à direita ou à esquerda
  - Recebe um ponteiro para a string que é necessário "formatar"
  - É usada no ficheiro **utils.c** dentro da função **guardarInfo** (referenciada mais à frente)

- **void calculaPercurso(struct linha* listaLinhas);**
  - Função que mostra o percurso entre duas paragens
  - Mostra nos dois sentidos: Paragem A para Paragem B e vice-versa
  - É usada no ficheiro **utils.c**

```
Opcao que pretender escolher:7
 Digite a paragem de partida:porto
 Digite a paragem de chegada:aveiro

Percurso encontrado usando a linha norte:
porto -> aveiro ->
```

Figura 9 – Exemplo do "cálculo" de um percurso (usado o exemplo entre 'porto' e 'aveiro')

- **void guardarInfo(Paragem* paragens, int tamanho_array, struct linha* listaLinhas, char* nome_arquivo)**
  - Função que armazena as informações do array de estruturas dinâmico (que contem as paragens e os seus dados) e da lista ligada (que contem as linhas e os seus dados) num ficheiro binário
  - O nome definido para esse ficheiro binário foi info.bin
  - É usada no ficheiro **main.c**

1. **void reconstruirEstruturas(Paragem** paragens, int* tamanho_array, Linha** listaLinhas, char* nome_ficheiro)**

- Função que “reconstrui” as estruturas: o array de paragens (*array* de estruturas dinâmico) e a lista de linhas (estrutura dinâmica do tipo lista ligada)
- Faz isto através do ficheiro binário info.bin que foi criado/atualizado pela função **guardarInfo**
- É usada no ficheiro **main.c**

2.4 Menu e funcionalidades do programa

No **main.c** é onde se cria o *array* de estruturas dinâmico e a estrutura dinâmica do tipo lista ligada e onde se tenta carregar o ficheiro **info.bin** para carregar as informações das paragens e linhas. No caso de não ser possível carregar o ficheiro, por exemplo caso ele não tenha sido criado anteriormente, o programa mostra a mensagem de erro “Erro ao abrir o arquivo info.bin”

Após isso está um *switch case* que serve para imprimir todas as opções que chamam funções diferentes dependendo do que o utilizador pretende fazer. Isto permite apresentar um menu com interface simples e intuitivo para o utilizador.

```
-------Paragens-------
1 - Registar Paragem
2 - Eliminar Paragem
3 - Visualizar Paragens
--------Linhas--------
4 - Adicionar Linha
5 - Atualizar Linha
6 - Visualizar Linhas
--------Outros--------
7 - Calcular Percurso
8 - Sair
----------------------

Opcao que pretender escolher:
```

Figura 10 – Interface do menu

Quando o utilizador escolhe a opção “8 – Sair”, o *switch case* termina e é guardada a informação relevante do programa no ficheiro **info.bin** e libertada a memória ocupada.

3. Conclusão

Através do programa apresentado neste relatório é possível fazer uma gestão eficiente de uma rede de mobilidade/transporte através de um sistema em que todas as paragens têm nomes e códigos alfanuméricos distintos sendo por isso facilmente manipulados seja adicionando novas paragens e eliminando paragens de uma lista ou então fazer as mesmas operações, mas para paragens que façam parte de linhas.

As linhas também têm a mesma característica das paragens, ou seja, todas são únicas (todas com nomes distintas) e igualmente simples de manipular. As diferentes opções nomeadamente na forma de adicionar novas linhas é uma vantagem deste programa pois permite poupar algum tempo ao utilizador que assim não necessita de inserir paragens uma a uma.

O facto de o programa também ser capaz de guardar e carregar as informações de uma execução para execuções futuras também é uma grande mais-valia evitando assim a perda de dados e dispensa a necessidade de voltar a introduzir os mesmos dados sempre que o programa é iniciado.

4. Bibliografia

1. **Stack Overflow–** https://stackoverflow.com
2. **TutorialSpoint –** www.tutorialspoint.com
3. **ChatGPT –** https://chat.openai.com
4. **Moodle ISEC –** https://moodle.isec.pt/moodle/