METRO MONDEGO

Programação 2022/2023

Licenciatura em Engenharia Informática

Dinis Meireles de Sousa Falcão | a2020130403@isec.pt

INDICE

INTRODUÇÃO

A empresa que gere o sistema de mobilidade do Metro Mondego pretende um programa que efetue a gestão das várias linhas existentes. Este programa deverá manter a informação atualizada sobre as linhas que constituem o sistema e sugerir percursos entre duas paragens distintas.

Cada linha do sistema Metro Mondego é constituída por várias paragens. Em cada paragem podem entrar e sair pessoas. As linhas podem ser percorridas nos dois sentidos. Nenhuma das linhas passas na mesma paragem mais do que uma vez, ou seja, não são circulares.

O sistema é constituído por diversas linhas que se cruzam em algumas paragens. Nesses locais, um passageiro pode sair de uma das linhas e entrar noutra.

Foi desenvolvido um programa em linguagem C que permite efetuar a gestão do sistema de mobilidade do Metro Mondego, cumprindo os requisitos e dados referidos anteriormente.

DESCRIÇÃO

Neste ponto serão apresentadas e descritas todas as estruturas existentes no projeto, ficheiros, funções e funcionalidades das mesmas.

Estrutura "paragem":

```
typedef struct paragem pgm, *pg;

struct paragem{
    char codigo_alfanumerico[5];
    char nome[99];
};
```

Estrutura "linha":

```
typedef struct linha lna, *ln;

struct linha{
    char nome[99];
    pg lista;
    ln prox;
    int nParagens;
};
```

A estrutura paragem possui 2 atributos do tipo *char*, que são o Código Alfanumérico e o Nome e que identificam as diferentes paragens existentes. A estrutura linha possui 4 atributos de diferentes tipos (*char*, *pg*, *ln* e *int*). Os atributos Nome e Número de Paragens identificam as diferentes linhas. Os ponteiros para estruturas são

utilizados para que apontar para a linha seguinte e para que haja acesso às paragens.

Ficheiro "paragens.txt":

```
Parque da Cidade # P011
Portagem # P123
Loja do Cidadao # Q998
Casa do Sal # F554
Coimbra-B # H123
```

Este ficheiro contém os nomes e códigos alfanuméricos das paragens que se pretendem inicializar.

Ficheiro "linhas.txt":

```
Linha da Baixa
Parque da Cidade # P011
Portagem # P123
Loja do Cidadao # Q998
Casa do Sal # F554
Coimbra-B # H123
```

Este ficheiro contém os nomes da linha que se pretende inicializar, e os nomes e códigos alfanuméricos das paragens a que ela pertencem.

Ficheiro "dados.dat":

Este ficheiro guarda a informação das paragens e das linhas e permite reconstruir as estruturas dinâmicas quando o programa retoma a sua execução.

Funções "paragem.h" e "paragem.c":

```
pg adicionaParagem(pg tab, int *n);
void visualizaParagens(pg tab, int n);
pg paragemExiste(pg tab, int *n, char *nomeParagem);
void adicionaCodigoParagem(char *codigo, int tamanho);
pg adicionaParagensFicheiro(pg tab, int *n);
void visualizaParagensFicheiro(pg tab, int n);
```

A função *adicionaParagem* recebe uma tabela de paragens *tab* e um ponteiro para a variável *n* que representa o número atual de paragens. Ela pede ao utilizador o nome de uma nova paragem e verifica se esse nome já existe na tabela. Caso exista, mostra uma mensagem de erro. Caso contrário, realoca a memória da tabela para guardar a nova paragem, copia o nome da paragem fornecido pelo utilizador para a tabela, chama a função *adicionaCodigoParagem* para gerar um código alfanumérico para a paragem e incrementa o valor de *n*. Por fim, retorna a tabela atualizada.

A função *visualizaParagens* recebe a tabela de paragens *tab* e o número *n* de paragens. Ela percorre a tabela e imprime o nome e o código alfanumérico de cada paragem.

A função *paragemExiste* recebe a tabela de paragens *tab*, um ponteiro para a variável *n* de paragens e o nome de uma paragem a ser procurada. Ela percorre a tabela e verifica se o nome da paragem fornecido é igual ao nome de alguma paragem na tabela. Se encontrar uma correspondência, imprime uma mensagem indicando que a paragem foi encontrada e retorna um ponteiro para essa paragem. Caso contrário, imprime uma mensagem indicando que a paragem não foi encontrada e retorna NULL.

A função *adicionaParagensFicheiro* recebe uma tabela de paragens *tab* e um ponteiro para a variável *n* que representa o número atual de paragens. Ela solicita ao usuário o nome de um arquivo e tenta abri-lo. Se a abertura do arquivo falhar, retorna NULL e exibe uma mensagem de erro. Caso contrário, a função lê o nome e o código alfanumérico de cada paragem do arquivo. Para cada paragem lida, verifica se o nome já existe na tabela. Se existir, retorna a tabela sem efetuar alterações e exibe uma mensagem de erro. Caso contrário, a função realoca a memória da tabela para guardar a nova paragem, copia o nome e o código da paragem lidos do arquivo para a tabela e incrementa o valor de *n*. A função continua a ler e a adicionar paragens até atingir o final do arquivo. Por fim, retorna a tabela atualizada.

Funções “linha.h” e “linha.c”:

```
pg eliminaParagem(pg tab, int *n, ln linha);

ln adicionaLinha(ln tab, pg par, int *n);

void visualizaLinhas(ln tab);

ln atualizaLinha(ln tab, pg par, int *n);

ln adicionaLinhaFicheiro(ln tab, pg par, int *n);

void armazenaBinario(ln tab, pg par, int *n);

void calculaPercurso(ln tab, pg par, int *n);

void visualizaLinhasParagem(ln tab);
```

A função *eliminaParagem* é responsável por remover uma paragem. Ela recebe como parâmetros um ponteiro para a primeira paragem da lista (*pg tab*), um ponteiro para o número total de paragens (*int *n*), e um ponteiro para a primeira linha da lista (*ln linha*). A função percorre todas as linhas em busca da paragem a ser removida e a exclui-a da lista. Caso a paragem não seja encontrada, a lista de paragens é retornada sem alterações.

A função *adicionaLinha* permite adicionar uma nova linha à lista de linhas existente. Ela recebe como parâmetros um ponteiro para a primeira linha da lista (*ln tab*), um ponteiro para a primeira paragem da lista (*pg par*), e um ponteiro para o número total de paragens (*int *n*). A função solicita ao usuário o nome da nova linha e verifica se o nome já existe na lista. Se o nome já estiver em uso, a função retorna a lista original sem fazer alterações. Caso contrário, a função cria uma nova linha,

permite a adição de paragens à mesma e adiciona-a à lista existente.

A função *visualizaLinhas* mostra todas as linhas existentes juntamente com as suas paragens. Ela recebe como parâmetro um ponteiro para a primeira linha da lista (*ln tab*). A função percorre a lista de linhas e, para cada linha, mostra o nome da linha e todas as suas paragens.

A função *atualizaLinha* permite atualizar uma linha existente, permitindo a adição ou remoção de paragens. Ela recebe como parâmetros um ponteiro para a primeira linha da lista (*ln tab*), um ponteiro para a primeira paragem da lista (*pg par*), e um ponteiro para o número total de paragens (*int *n*). A função solicita ao usuário o nome da linha a ser atualizada e verifica se ela existe na lista. Se a linha não for encontrada, a função retorna a lista original sem fazer alterações. Caso contrário, a função permite ao usuário escolher entre adicionar ou remover paragens da linha e executa a ação solicitada.

A função *adicionaLinhaFicheiro* tem como objetivo adicionar uma nova linha ao sistema de transporte a partir de um ficheiro de texto que contém informações sobre a linha e as suas paragens. Ela lê o nome da linha e as paragens do ficheiro, verifica se o nome da linha já existe na lista atual, e adiciona a nova linha à lista, juntamente com as suas paragens, se forem válidas. Essa função garante a integridade dos dados e a adição correta das informações ao sistema.

A função *armazenaBinario* recebe uma lista de linhas, uma lista de paragens e o número total de paragens. Ela cria um arquivo binário chamado "dados.dat" e armazena as informações das linhas e paragens nesse arquivo. A função percorre cada elemento da lista de linhas. Para cada linha, ela escreve o nome da linha no arquivo usando a função *fwrite*. Em seguida, percorre a lista de paragens da linha e escreve as informações de cada paragem no arquivo, concatenando o nome da paragem com o código alfanumérico, separados pelo caractere *'#'*. No final do processo, o arquivo é fechado. Esta função tem como objetivo armazenar as informações das linhas e paragens em formato binário para posterior recuperação dos dados.

A função *calculaPercurso* recebe uma lista de linhas, uma lista de paragens e o número total de paragens. Ela solicita ao utilizador a origem e o destino desejados. Em seguida, percorre as linhas verificando se a origem e o destino estão presentes. Se encontrados, mostra o percurso entre elas, considerando a ordem das paragens. A função permite calcular e visualizar o percurso entre duas paragens desejadas no sistema de transporte, nos dois sentidos.

A função *visualizaLinhasParagem* recebe uma lista de linha. Ela solicita ao utilizador o nome de uma paragem desejada. Em seguida, percorre as linhas e verifica se a paragem está presente em cada uma delas. Se a paragem for encontrada, exibe o nome da linha correspondente. A

função permite visualizar as linhas que passam por uma determinada paragem.

FUNCIONALIDADE

No "main.c" é onde estão inicializadas as duas estruturas dinâmicas. Estas são inicializadas a NULL. É no mesmo sítio que são chamadas todas as funções necessárias à execução do programa e onde é apresentado o menu de escolha das diferentes funcionalidades. Neste programa é possível executar todas as funções descritas no enunciado do trabalho prático, ou seja, é possível adicionar, ver e remover paragens; adicionar, ver e atualizar (eliminar ou adicionar paragens) linhas; adicionar paragens e linhas através de ficheiros de texto; guardar todas as informações sobre as linhas e as paragens num ficheiro binário; e calcular percursos entre duas paragens distintas.

DIFICULDADES

Não foi possível fazer o cálculo dos percursos com uma mudança de linha. Isto talvez se deva à falta de gestão de tempo na realização do trabalho, ou até mesmo à falta de conhecimento para conseguir alcançar a melhor estratégia para cumprir este ponto. Além disto, também não fui capaz de fazer o *load* do ficheiro binário.

Ainda assim, todos os restantes pontos e objetivos do trabalho foram cumpridos, sem exceção.

BIBLIOGRAFIA

Luís Manuel Dias Dama – *Linguagem C*, 1ª Edição: FCA, 1999. ISBN 9789727221561

FIM