MANUAL DO PROGRAMADOR

Este projeto foi concebido no âmbito da cadeira de Princípios de Programação Procedimental, e tem como objetivo simular um programa que auxilie uma agência de viagens. Ao longo deste manual iremos explicar como funciona a dinâmica do nosso programa e como o estruturámos através de código-fonte. No anexo que encontra no final deste manual, iremos explicar a organização dos ficheiros de código-fonte.

NOTA: Para o bom funcionamento do programa, deve estar sempre os ficheiros .txt na mesma diretoria do executável.

MENU

No menu principal, temos 6 opções:

1 — Listar viagens;
2 — Adicionar cliente/viagem;
3 — Adquirir viagens;
4 — Cancelar viagem;
5 — Lista de clientes;
6 — Sair;

E ainda dentro da opção 1, podemos listar as viagens por destino ou por cliente. Na opção 2, podemos adicionar cliente ou adicionar viagem.

Na implementação deste menu, usámos um switch case. E dentro de cada uma das opções do menu principal usámos a operação if, a ser auxiliado por um ciclo while.

1 — Listar viagens

- **Listar viagens por destino**

No início da função, é pedido ao utilizador que insira o destino da viagem. É criada uma lista que contém todas as viagens pra esse destino, se não for encontrada nenhuma viagem é devolvida uma lista em que o parâmetro *codigo* é igual a -1, imprimindo ao utilizador que não existe o destino inserido. Depois de criada a lista com todas as viagens para o destino inserido é feita a ordenação da lista, usando o *Bubble Sort.*

O formato de data que usámos foi o seguinte: AAAA/MM/DD. Lemos a partir do ficheiro “viagens.txt” a data como uma string, para podermos usar a função *strcmp.*

Em cada iteração do ciclo for é comparado em primeiro lugar a data de dois nodes adjacentes, se o primeiro node a comparar for maior do que o segundo faz a troca de posições dos nodes. No caso de haverem datas iguais, usamos o mesmo procedimento mas comparando agora as horas. Depois do ordenamento da lista, esta é imprimida. Para comparar as datas usámos a função *strcmp,* que retorna um valor menor que zero se a primeira string comparada é menor do que a segunda. Devolve um valor maior que zero, se a segunda for menor do que a primeira. E devolve zero, se as strings forem iguais.

De seguida, é libertado o espaço em memória da lista.

Neste caso, foi utilizada uma estrutura mais simples onde não foi necessário guardar a informação dos clientes que tinham adquirido esta viagem por não ser necessário à resolução deste problema.

- **Listar viagens por cliente**

Esta função permite listar para um dado cliente todas as viagens adquiridas, sendo as mais antigas primeiro.

No início da função é pedido ao utilizador que insira o primeiro e último nome do cliente. É criada uma lista com informação do cliente, caso esse cliente não exista, o parâmetro *codigo* é igual a -1. De seguida é criada uma lista com todas as viagens adquiridas pelo cliente. As viagens do cliente estão armazenadas em dois array's com tamanho máximo de 20, sendo um array destinado a guardar informação das viagens em que o cliente está na lista de espera, e o outro contém a informação das viagens em que o cliente está a lista de reserva. Cada elemento do array contém o código da viagem adquirida pelo cliente.

É usado um ciclo *while* para percorrer o array das viagens adquiridas. É lida a informação da viagem e é criado um node caso essa viagem ainda não esteja na lista de viagens. De seguida, procede-se da mesma forma, para o array das viagens em que o cliente está na lista de espera.

A ordenação da lista segue os mesmos modos que a ordenação da lista das viagens usada para listar viagens por destino.

Por fim, segue-se a impressão da lista de viagens já ordenada, seguida da libertação em memória da lista de clientes e das viagens.

2 — Adicionar cliente/viagem

- **Adicionar cliente**

Assim que a função começa é pedido ao utilizador, o primeiro e último nome do novo cliente, pede-se também o seu número do Cartão de Cidadão. Há verificação se o nº do CC é válido (9 dígitos). Segue-se a atualização do ficheiro "clientes.txt". Para isso, fazemos a leitura do ficheiro para determinar o último código de cliente já atribuído. Reabre-se o ficheiro "cliente.txt" no modo *append,* e escreve-se a informação do novo cliente. Para a leitura do ficheiro, foi criada uma função chamada "le_linha" que recebe como parâmetros: *file pointer* (*fp), e *long *num.* Vai-se para a posição ainda não lida do ficheiro e lê-se o código de um cliente. De seguida, há leitura de caracter a caracter até ao final da linha, atualiza-se a variável num com a função *ftell* (que devolve a posição atual do file pointer). Esta função devolve *codigo* lido. Por fim, surge uma mensagem com a informação de que o cliente foi adicionado, perguntando também se deseja adicionar ou não mais clientes.

- **Adicionar viagem**

No início da função é pedida a origem e destino da viagem, seguida da data no seguinte formato: AAAA/MM/DD, e a hora no formato HH:MM. Por fim, é pedido ao utilizador, o número total de lugares da viagem. Segue-se a escrita dos dados da nova viagem, com um procedimento de escrita semelhante ao de adicionar cliente. Para uma maior validação dos dados, nós sujeitámos os dados a uma verificação se se enquadram nos parâmetros corretamente estabelecidos.

3 — Adquirir viagens

Para começar, criámos um array que vai ter todos os códigos das viagens. De seguida, pedimos a origem e o destino da viagem, através de *printf* e *scanf,* este último que contém um

modificador (%[^n]s) , que consiste em ler todos os caracteres até encontrar um “\n”. Criamos a função *verifica_viagem,* que percorre a lista com todas as viagens existentes, pondo no array criado no início da função. À medida que vai percorrendo a lista, se encontrar uma viagem que satisfaça a origem e o destino introduzidos imprime-se a informação da viagem. Todas as variáveis são recebidas como ponteiros nos respectivos tipos para que seja possível se necessário alterar-se o seu conteúdo. Por fim, é devolvido o número total de viagens.

Se o número total de viagens for diferente de zero, é pedido ao utilizador que insira o código da viagem pretendida. Seguidamente, é verificado se o código introduzido é válido, se for válido imprime todos os clientes registados, se não, informa que o utilizador inseriu um código incorreto. E pergunta se deseja voltar a escrever um novo código. Caso, o código introduzido seja válido, é pedido o código do cliente, e se verifica se é válido, sendo maior ou igual que 1 e menor ou igual que o código máximo de clientes.

Com o código válido, é chamada a função *adquire* que tem como parâmetros: a lista de viagens, lista de clientes, e os códigos introduzidos. Dentro da função *adquire,* são percorridas as listas até aos nodes com a informação da viagem pretendida e do cliente introduzido.

De seguida, é perguntado ao utilizador se deseja ir pra uma lista de espera ou para uma lista de adquiridos. Se desejar ir para a lista de espera, vai pra o final da lista, se não, vai pra a lista de adquiridos. É verificado se existem vagas, através da função *verifica_vagas,* que percorre a lista de adquiridos e verifica se o número de iterações que fez se é maior ou igual que o número total de lugares. Se for maior, devolve zero, caso contrário, devolve 1. Se houver vagas, atualiza-se as listas que contêm a informação dos clientes que adquiriram a viagem. Para a actualização dessa informação são utilizadas a função *add_reserva* e a função *atualiza_array.*

A função *add_reserva* recebe como parâmetros: uma lista de adquiridos ou de espera, e a lista de cliente. Percorre-se a lista de espera/adquiridos até ao fim. Se a informação do node atual com a variável *cod_cliente* for igual a zero, significa que não há nenhum cliente em lista de adquiridos/espera. Neste caso, é alterada essa variável para o código do cliente que quer adquirir a viagem, se não, é criada um novo node para a lista de adquiridos/espera que vai conter informação do cliente.

A função *atualiza_array* recebe como parâmetros: um ponteiro para um array de inteiros, um código da viagem, e um modo. Se o modo for igual a zero, a função elimina informação do array (usada no cancelamento de viagens, que será explicado mais à frente), caso contrário, adiciona informação. Para a adicionar, é percorrido um array até encontrar um zero, que significa que já não há mais informação necessária a preservar. Na posição do array em que está um zero, é colocado o código da viagem, e na seguinte, por precaução é colocado um zero.

Após todas estas operações, são atualizados os ficheiros: “viagens.txt” e “clientes.txt”, em que são escritas as listas, que contém uma codificação própria, que será explicada mais à frente.

Para concluir, é perguntado ao utilizador se deseja adquirir mais viagens, caso introduza uma opção fora das válidas (0 - não; 1 - sim), é detetado um erro, que informa o utilizador que colocou uma opção inválida.

4 — Cancelar viagem

A função cancelar_viagens recebe como parâmetros: a lista de clientes, lista de viagens, e o código máximo de clientes. Para ajuda da resolução deste problema que a própria função assim o exigia foram criadas duas variáveis do tipo lista, uma para a viagem, e outra para os clientes.

Pra o funcionamento da função, pedimos ao utilizador que insira que tipo de cancelamento deseja, se quer cancelar viagens adquiridas ou se quer cancelar da lista de espera. Se introduzir uma opção válida, imprime uma lista dos clientes, e pede o código do cliente em questão. Caso o

código seja válido, procura-se o cliente na lista. Do mesmo modo, imprime-se a lista das viagens, perguntando de seguida qual o código a inserir da viagem que o cliente pretende cancelar. Através de um ciclo *while,* procura o código da viagem introduzida na lista, caso não o encontre surge uma mensagem de erro. Independentemente das opções, há verificação de se o cliente está na viagem introduzida, caso não esteja é imprimida uma mensagem de erro. Caso esteja na viagem introduza é feito o cancelamento da viagem, para isso foi usada a função *cancela,* que recebe uma lista de clientes que têm a viagem adquirida ou que estão na lista de espera, e o código do cliente em questão. É procurada na lista que o cliente quer cancelar viagem e que faça a actualização da lista. Se o cliente for o primeiro da lista usa-se uma variável auxiliar que aponta para o segundo elemento da lista, faz-se a libertação do primeiro elemento da lista e devolve-se o segundo. Se o cliente não for o primeiro da lista, atualiza-se os ponteiros da lista de forma a que o elemento anterior ao cliente passe a apontar para o elemento a seguir ao cliente, de seguida faz-se a "libertação" do cliente, e devolve-se nova lista.

Segue-se a atualização da lista de espera no qual o primeiro elemento da lista de espera passe para o último lugar da lista de adquiridos, e no fim devolve-se a lista de espera atualizada. De seguida, faz-se a atualização do array que contém a informação das viagens do cliente usando a função *atualiza_array*. Desta vez, a função *atualiza_array,* inicia-se no modo zero, que significa que vai eliminar informação do array, pra a eliminar-mos percorremos o array até encontrarmos o código da viagem que o utilizador cancelou, e faz-se uma cópia da restante informação do array eliminando o código da viagem que foi cancelada.

Para concluir a função, faz-se a actualização dos ficheiros, usando as funções: *escreve_lista_clientes_fich* e *escreve_lista_viagem_fich,* já explicadas anteriormente.

5 — Lista de clientes

É criada uma lista de todos os clientes que adquiram ou estão em espera para viagens. Para isso, é lido o ficheiro "clientes.txt", e verifica-se se o cliente lido tem viagens adquiridas ou se está na lista de espera de alguma viagem. Para esta verificação, procura-se no array de viagens adquiridas e de espera um código de viagem, se não encontrar ignora-se todos os dados lidos. Imprime-se a lista de clientes com a função *imprime_clientes,* que consiste na impressão dos dados do cliente percorrendo a lista, imprimindo como inteiro o código do cliente, como string o primeiro e último nome, e por fim, como inteiro o número do cartão de cidadão.

Por fim, realiza-se a libertação da lista criada nesta função.

ANEXO

O ficheiro adicionar.c tem estas duas funções:

- int le_linha(FILE *fp, long *num)
- long tamanho_fich (FILE *fp)

Estas duas funções são compartilhadas por: “Adicionar_Clientes.c” e por “Adicionar_Viagem.c”.

O ficheiro listar.c tem as seguintes funções: ordena_lista e imprime_lista_viagem, contem ainda a definição da estrutura Lnode_Viagens_dest, usada na função Listar_Viagens.c.

O ficheiro Funcoes.c contém:

- void inicializa(int *array_esp, int *array_adq);
- void le_nome(char *output, char *input, int *start);
- void copia_para_array(int *array_dest, char *array_orig, int *start);
- int det_tam(Lista_Viagens x);
- void imprime_clientes(Lista_Clientes x);
- void escreve_array(FILE *fp, int *x);
- Lista_Clientes cria_lista_clientes(int *cod);
- void escreve_lista_clientes_fich(Lista_Clientes x);
- Lista add_Lista_Esp_Adq(int *array_esp_adq, int lugar, Lista_Clientes lista_clien);
- void escreve_lista(FILE *fp, Lista x);
- void escreve_lista_viagens_fich(Lista_Viagens x);
- Lista_Viagens cria_lista_viagens(Lista_Clientes list_clien);
- void liberta_listas(Lista_Clientes lista_clien, Lista_Viagens lista_viag);
- void atualiza_array(int *array, int codigo_viagem, int modo);

Este ficheiro contém também as definições das estruturas de listas de clientes e lista de viagens.

Todos os ficheiros têm um respectivo header file, que contem o protótipo da função principal do ficheiro e possíveis funções partilhadas entre ficheiros. O ficheiro main.c é o principal, que contém o menu principal do programa.

Para o bom funcionamento do programa, deve estar sempre os ficheiros .txt na mesma diretoria do executável.