

-Manual de programador -

PRÍNCIPIOS DE PROGRAMAÇÃO PROCEDIMENTAL

André Loras Leite nº 2015250489 TP2

Gonçalo Oliveira Amaral nº 2015249122 TP2

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA | LICENCIATURA EM ENGENHARIA INFORMÁTICA

Função Menu_principal:

```
int menu_principal(Viagem global_viagens, pCliente *adq, pCliente *esp) {
    char op[128];
    int option, cod, data, id, cond;
    char nome[128];
    char destino[128];
    char aux[128];
    int aux_int;

    printf("Menu\n");
    printf("(1) Adquirir uma viagem\n");
    printf("(2) Colocar em fila de espera para uma viagem\n");
    printf("(3) Cancelar viagem\n");
    printf("(4) Cancelar pedido em fila de espera\n");
    printf("(5) Listar viagens de um destino\n");
    printf("(6) Listar viagens de um cliente\n");
    printf("(7) Listar clientes\n");
    printf("(8) Sair\n\n");
    printf("Opcao: ");

    cleanstr(op);
    fgets (op, 128, stdin);
    option=atoi(op);
    if( isdigit(*op) && strlen(op)==2 && option<9 && option>0 ){
    } else {
        clrscr();
        printf("Opcao Invalida!\n\n");
        return menu_principal(global_viagens, adq, esp);
    }
}
```

Esta função tem como principal finalidade dispor ao utilizador uma série de opções, num pequeno e simples menu que dependendo da opção inserida irá permitir manipular de forma fácil e com alto nível de abstração, toda a aplicação. Esta mesma função está protegida caso o elemento introduzido seja um dígito fora do limite [1,8] ou qualquer outro tipo de caracter. É aplicado pela condicionada da forma **if -> else -> return**:

```
if( isdigit(*op) && strlen(op)==2 && option<9 && option>0 ){
} else {
    clrscr();
    printf("Opcao Invalida!\n\n");
    return menu_principal(global_viagens, adq, esp);
}
```

As opções são controladas pela instrução de escolha múltipla **switch** -> **case**. Cada valor está associado a um **case**, e dependendo do valor do parâmetro do **switch**, é executado o respetivo **case**. Vejamos então a implementação desta instrução para a aplicação:

```
switch(option) {
```

CASE 1:

```
case 1:
    clrscr();
    printf("Adquirir Viagem\n");
    cleanstr(destino);
    cod=choose_list_viagem_destinos(&global_viagens,destino);
    clrscr();
    printf("Adquirir Viagem\n");
    data=choose_list_viagem_data(&global_viagens,cod);
    clrscr();
```

Este caso é direcionado para o inteiro (1) uma vez que corresponde á primeira opção (Adquirir um viagem). Caso o inteiro 1 seja o selecionado esta **case** entra em vigor e começa por fazer um print do nome da opção, por conseguinte é chamada a função **choose_list_viagem_destinos**:

```
int choose_list_viagem_destinos(Viagem *lista,char *destino){
```

Esta função cria um pequeno menu com todos os destinos. Para além dispor também permite selecionar um destino sendo o valor devolvido pela função o código da viagem.

Em seguida é novamente chamada uma função para complementar a escolha e aplicar as devidas mudanças e manipulações á aplicação. Desta vez chamamos uma função denominada por **choose_list_viagem_data**:

```
int choose_list_viagem_data(Viagem *lista,int x){
```

Semelhante à função anterior, esta função dispõem as datas para o destino selecionado, devolvendo a data escolhida.

Uma vez feita a distribuição de dados pelas funções acima referidas e explanadas, podemos passar ao registo do passageiro dada a viagem escolhida, com auxílio das funções **choose_list_viagem_destinos** e **choose_list_viagem_data**. Assim, na condição **case 1** e como em todas as outras será possível verificar que é primeiramente sugerida a introdução do número de cartão de cidadão e isto para evitar que nos registos seguintes seja necessária a introdução repetitiva do nome. Desta forma basta fazer um registo de cartão de cidadão -> nome, que irá ficar registada na base de dados, se o cliente tiver adquirido uma viagem ou colocado uma viagem em fila de espera, lembrando ao sistema que este número de cidadão já foi registado, descartando a nova introdução do nome do cliente.

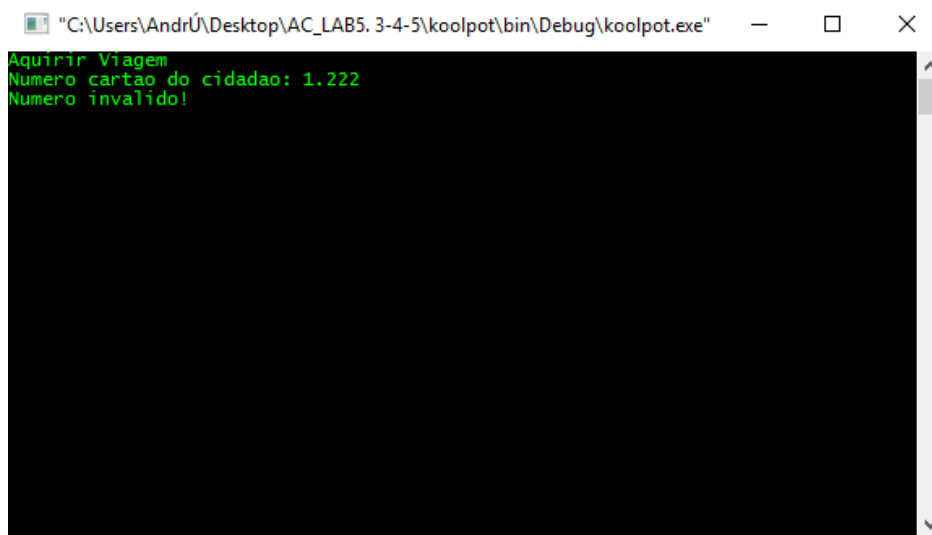
```
while(cond==1){
    printf("Aquirir Viagem\n");
    printf("Numero cartao do cidadao: ");
    fgets (nome, 128, stdin);
    remove_return(nome);
    cond=check_number(nome);
    if( cond == 1 ){
        printf("Numero invalido!");
        clean_input();
        clrscr();
    }
}
id=atoi(nome);
```

Neste **while loop** temos a introdução do número de cartão do cidadão e a sua respetiva verificação. Começando pela receção de uma string com o respetivo input, de seguida é removido o '\n' recorrendo à função **remove_return**.

```
int check_number(char *str){
    int i=0;
    while(str[i]){
        if( !isdigit(str[i]) ) return 1;
        ++i;
    }
    if( i==0 ) return 1;
    return 0;
}
```

Utilizando a função **check_number**, verificamos se o input é um número.

Vejamos um exemplo real com a introdução de um número decimal no espaço para o cartão de cidadão:



De seguida teremos outro **while loop** onde é feito algo muito semelhante mas com uma restrição adicional, **if clause** com a função **get_nome_from_id**. Desta vez, é utilizada a função **check_name** para verificar a string introduzida como parâmetro.

```
if( get_nome_from_id(adq,esp,id,nome)==0 ){
    cond=1;
    while( cond==1 ){
        printf("Nome: ");
        cleanstr(nome);
        cleanstr(aux);
        fgets (nome, 128, stdin);
        remove_return(nome);
        cond=check_name(nome);
        if( cond==1 ) {
            printf("Nome invalido!");
            clean_input();
            clrscr();
            printf("Aquirir Viagem\n");
        }
    }
}
```

O objetivo principal da função **get_nome_from_id** é verificar se o cartão do cidadão introduzido existe na lista de viagens adquiridas ou na lista de viagens em fila de espera.

```
int get_nome_from_id(pCliente *adq,pCliente *esp,int cliente,char *nome)
```

Tem como valor de retorno um inteiro, que é **1** se existir no sistema e **0** caso contrário. Se o número for encontrado, o nome associado ao mesmo é atribuído ao parâmetro nome.

```
int check_name(char *str){
    int i=0;
    while(str[i]){
        if( !isalpha(str[i]) && !isspace(str[i]) ) return 1;
        if(i>0 && isspace(str[i]) && isspace(str[i-1]) ) return 1;
        ++i;
    }
    if( i==0 ) return 1;
    return 0;
}
```

Utilizando a função **check_name**, verificamos se o input é um nome válido.

Vejamos um exemplo real com a introdução de um nome seguido de dígitos, no espaço para o nome:

```
Aquirir Viagem
Numero cartao do cidadao: 123
Nome: Gonçalo 13213231321
Nome invalido!
```

Para finalizar este Case, temos ainda a função **insere_last_adq**:

```
void insert_last_adq(int option, Viagem *global, pCliente *adq, pCliente *esp,
int data, char *destino, int cod_destino, int id, char *nome){
```

Tentamos criar esta função da maneira mais dinâmica possível. O objetivo desta função será inserir no final de uma lista um nó criado através do conjunto de elementos inseridos como parâmetros. Esta função poderá ser cumprir o seu objetivo quer para a lista de viagens adquiridas quer para a lista das viagens em lista de espera através da **option** devendo ser **0** se pretendermos inserir no final da lista de viagens adquiridas e **1** se pretendermos inserir no final da lista de viagens em fila de espera. Para a **option 0**, utilizada no **case 1**, a função para poderá não inserir na lista de viagens adquiridas.

```
switch( option ){
    case 0:
        if( get_viagens_disp_destino(global,data,cod_destino) > 0 ){
            diminuir_disp(global,data,cod_destino);
            enqueue(adq,cod_destino,data,destino,id,nome);
        } else if( get_viagens_disp_destino(global,data,cod_destino)==0 ){
            enqueue(esp,cod_destino,data,destino,id,nome);
        }
        break;
```

A função **recorre** irá inserir na lista de viagens adquiridas se existirem lugares disponíveis. Para verificar isto recorreremos à função **get_viagens_disp_destino**.

```
int get_viagens_disp_destino(Viagem *global,int data,int code){
```

Esta função irá percorrer a lista de viagens e obter os lugares disponíveis para a data e destino introduzidos, usando este valor como valor de retorno.

Se este valor for superior a **0** a função irá inserir na lista de viagens adquiridas, diminuindo a os lugares disponíveis em 1 para a data e destino seleccionados.

Para efetuar esta diminuição recorreremos à função **diminuir_disp**

```
void diminuir_disp(Viagem *global,int data,int code){
```

Para efetuar a inserção no final da lista é usada a função **enqueue**

```
void enqueue(pCliente *lista,int code,int data,char *destino,int id,char *nome) {
```

Se o valor da disponibilidade for **0** a viagem será inserida no final da lista de viagens em fila de espera, recorrendo à **enqueue**.

CASE 2:

Tal como o anterior caso, as funções utilizadas são três, **choose_list_viagem_destinos**, **choose_list_viagem_data**, **remove_return**, **check_name**, **check_number**, **get_nome_from_id** e **insert_last_adq**.

```
insert_last_adq(1,&global_viagens,adq,esp,data,destino,cod,id,nome) ;
```

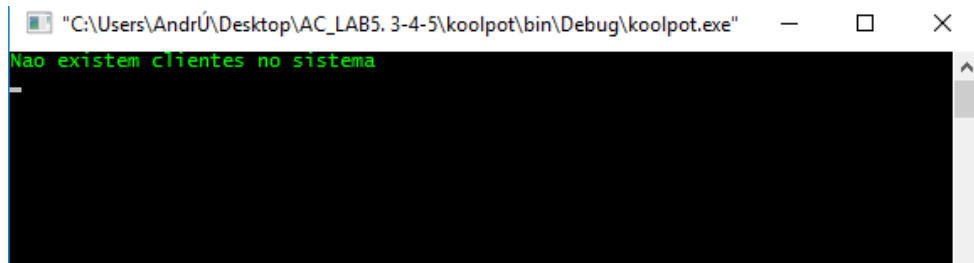
Para perceber a sua funcionalidade basta verificar a explicação da **case 1**, pelo que não existem variâncias de funções, muda somente o objetivo para o qual a informação é dirigida, recorrendo à função **insert_last_adq** com o parâmetro **option** a 1. Como foi referido anteriormente, e aqui é apresentado quão dinâmica a função **insert_last_adq** pode ser.

CASE 3:

Esta case é direcionada á eliminação de um registo de uma viagem, ou seja, cancelar uma viagem. Para que seja possível fazer o cancelamento é necessário ter a certeza que existem clientes e uma viagem. Assim sendo, temos :

```
case 3:
    clrscr();
    if( *adq==NULL && *esp==NULL ){
        printf("Nao existem clientes no sistema\n");
        clean_input();
        break;
    }
```


Isto refere-se ao facto de não existir clientes no sistema. Se existirem clientes é necessário fazer um rastreio às viagens que estão registadas pelo cliente. Assim, temos que:



Esta solução é complementada pela função **in_cliente**, que faz a verificação da existência de um cliente numa lista especifica dado o seu número de cartão de cidadão. Se for encontrado um cliente referente ao número introduzido é feito um **return** de **1**, caso contrário **0**.

```
int in_cliente(pCliente *cliente, int id) {
    pCliente *cliente_org=*cliente;
    while( *cliente!=NULL ){
        if( (*cliente)->id == id ){
            *cliente=cliente_org;
            return 1;
        }
        (*cliente)=(*cliente)->next;
    }
    *cliente=cliente_org;
    return 0;
}
```

Caso o número de cartão de cidadão introduzido não seja referente a nenhuma cliente é imprimida uma notificação de erro pela aplicação “Cliente não existe”:

```
if( in_cliente(adq,id)==0 ){
    printf("Cliente nao existe\n");
    clean_input();
    break;
}
```

A parte fulcral desta opção é dada pela função **remover_viagem**:

```
void remover_viagem(Viagem *global, pCliente *adq, pCliente *esp, int id)
```

Esta função tem como intuito eliminar o registo de uma viagem dado um cliente. Esta função é complementada com as funções **print_cliente_adq_adq**, **dequeue_and_save**, **enqueue**(já referida), **aumentar_disp** e **dequeue**.

print_cliente_adq_adq

```
int print_cliente_adq_adq(pCliente *adq, int cliente, Viagem *lista, int *data_arr, int *codes){  
    ...  
}
```

Esta função dispõe no ecrã todas as viagens adquiridas pelo cliente introduzido, dispondo em primeiro as mais antigas e devolvendo como valor de retorno a quantidade de opções disponíveis. Para fazer esta ordenação recorreremos à função **bubblesort_viagens**, **data_invert** e à função **data_fix**

data_invert

```
int data_invert(int data)
```

Esta função recorre a outra função(**data_to_dia_mes_ano**), que separa a data em dia, mês e ano e inverte a data para o formato ano mês dia para que a função seguinte possa fazer a organização de forma coerente.

bubblesort_viagens

```
void bubblesort_viagens(int vect[], int *codes, int n)
```

Esta função organiza o vetor de inteiros **vect** por ordem crescente, aplicando as mesmas alterações ao vetor **codes**.

data_fix

```
int data_fix(int data)
```

Esta função repõe as mudanças feitas pela função **data_invert**

dequeue

```
void dequeue(pCliente *lista,int id,int code,int data)
```

À semelhança da função **enqueue**, **dequeue**, faz o inverso e remove o primeiro nó da lista introduzida.

dequeue_and_save

```
void dequeue_and_save(pCliente *lista,int *code,int *data,char *destino, int *id, char *  
nome){
```

À semelhança da função **dequeue**, **dequeue_and_save**, faz o inverso e remove o primeiro nó da lista introduzida e guarda a informação do nó nos respetivos argumentos.

aumentar_disp

```
void aumentar_disp(Viagem *global,int data,int code)
```

À semelhança da função **diminuir_disp**, **aumentar_disp** aumenta o valor da disponibilidade de da viagem na data e código de destino usados como argumentos da função.

CASE 4:

Bem como no **case 3**, neste **case** é necessário fazer “check-ups” à existência de clientes neste parâmetro, ou seja, para que seja possível seguir em frente para o cancelamento de listas de espera é prioritário que sejam detetados clientes pois se o ficheiro estiver vazio a aplicação automaticamente irá cancelar o procedimento imprimindo um aviso “Não existem clientes no sistema”.

Por conseguinte e como em todos os casos, é pedido o número de cartão de cidadão que vai servir para registar o cliente ou para verificar a sua existência.

No final deste **case** é realizada, tal como no caso anterior, a eliminação do registo da lista de espera, recorrendo à função **remover_espera**.

```
void remover_espera(Viagem *global, pCliente *adq, pCliente *esp, int id)
```

Esta função tal como a anterior **remover_viagem**, utiliza funções secundárias ou de controlo para permitir a eliminação das listas de espera em ficheiros. Nesta função é permitida o cancelamento por opção, ou seja, é possível escolher a viagem que se quer cancelar, tal como era previsto nos objetivos da aplicação.

CASE 5:

Nesta **case** o objetivo é listar todas as viagens disponíveis de um determinado destino, sendo as mais recentes apresentadas em primeiro. Assim, para este efeito, foram usadas duas funções para aglomerar a informação de forma organizada. As funções são denominadas por **choose_list_viagem_destinos** (anteriormente referida, **case 1**) e **print_viagens_mais_recentes**.

Como já foi explorada a finalidade da função **choose_list_viagem_destinos** passemos à utilidade da função **print_viagens_mais_recentes**.

```
void print_viagens_mais_recentes(Viagem *lista, int cod)
```

Esta função utiliza a função **get_destino_from_cod**, **bubblesort_inv**, **data_invert**(referida anteriormente) e **data_fix**(também já referida) e cria um menu onde selecionamos o destino e é nos apresentado uma lista das datas para esse destino com as mais recentes primeiro.

get_destino_from_cod

```
void get_destino_from_cod(Viagem *lista, int cod, char *destino)
```

Esta função permite, mediante a introdução de um código de destino, a atribuição do nome do destino correspondente à string **destino**.

bubblesort_inv

```
void bubblesort_inv(int vect[], int n)
```

Esta função organiza o vetor **vect** por ordem decrescente.

CASE 6:

Em termos de acessibilidade é muito semelhante às repetidas cases 1 e 2. O método de reconhecimento e procura é igual e só muda a implementação da função **print_cliente_adq**. Esta função é completada por três outras, **print_cliente_adq_adq** (anteriormente referida) e **print_cliente_adq_esp**:

Quanto à função **print_cliente_adq_esp** a sua funcionalidade baseia-se em algo muito semelhante a **print_cliente_adq_adq** mas esta aplica-se à lista de viagens em fila de espera.

A função **print_cliente_adq** não passa da combinação de **print_cliente_adq_adq** e **print_cliente_adq_esp**

CASE 7:

Utilizando a função **print_clientes** é possível listar na consola todos os clientes uma vez que estes já se tenham registado ao efetuar uma compra ou a colocar uma viagem em fila de espera, de outra forma não será possível imprimir quaisquer clientes uma vez que não foram registados no sistema.

```
case 7:
    clrscr();
    printf("Lista de clientes\n");
    print_clientes(adq,esp);
    clean_input();
    break;
```

CASE 8:

Este case existe meramente para dar ao utilizador a opção livre de terminar a aplicação uma vez que a sua opção resulta num **break** que será definido como a paragem total do ciclo envolvente (**Switch case**).

Função main

Na função main são feitas a importação dos ficheiros para listas e após mudanças, as respetivas alterações nas listas nos ficheiros.

Função Export_cliente_to_file

Esta função tem o objetivo de exportar uma lista do tipo **pCliente** para um ficheiro.

Funções destroi_lista_viagem e destroi_lista_cliente:

Tal como o nome sugere, estas funções destroem as listas no final do programa.

Funções CreateGlobalListFromFile, createAdquiridasListFromFile, createEsperaListFromFile

Tem como função importar os dados dos ficheiros para as listas correspondentes

Funções Secundárias

Funções **getline** e **getlinecode**:

```
int getline(char *line, char *aux)
```

Sabendo que a string **line** tem uma determinada estrutura, copia o nome dessa **line** para a string **aux**. Se não foi possível executar a cópia do nome é retornado **0**.

```
int getlinecode(char *line)
```

Sabendo que a string **line** tem uma determinada estrutura, devolve o número como valor de retorno.

Funções print_list_viagem e print_list_clientes:

Estas funções são utilizadas para mero “debug” mostrando de forma organizada a lista introduzida.

Estrutura de ficheiros

viagens-data.txt

```
1 India
-1-01-2017 / 1
-2-02-2017 / 5
-3-03-2017 / 6
-4-04-2017 / 7

7 Alemanha
-4-12-2018 / 21
-6-05-2018 / 12
-5-09-2016 / 9
-5-06-2015 / 34
```

Linha Cabeçalho: (código_destino) (espaço)(destino)(“\n”)

Linha datas e lotação: (“-”)(dia)(“-”)(mes)(“-”)(ano)(espaço)(“/”)(espaço)(lotação)(“\n”)

Linha de separação: (“\n”)

viagens-adquiridas.txt e viagens-espera.txt

```
7 Alemanha
4122018
123 goa

1 India
1012017
123 goa

15 Russia
5092012
123 goa
```

Linha Cabeçalho: (código_destino) (espaço)(destino)(“\n”)

Data: (data)(“\n”)

CC e nome: (CC)(espaço)(nome)(“\n”)

Linha de separação: (“\n”)

Estrutura de variáveis específicas e justificação da sua implementação

Data

Exemplo: 1022015

(dia - 1)(mês - 02)(ano - 2015)

Utilizamos esta estrutura para facilitar a movimentação da data em toda a aplicação em vez de movermos 3 variáveis, dia, mês, ano, apenas movimentamos um formato compactado.

Código destino:

Exemplo: 1 India

Código de destino: 1

Decidimos associar a um destino um código pois é mais eficiente e fácil comparar dois inteiros e usar inteiros como parâmetros do que uma string. Por exemplo, o simples processo de procurar um nó que corresponda ao destino inseria, necessitaria de vários **strcmp**, e não podemos duvidar que comparar um numero seria muito mais rápido que comparar uma string.