



**FACULDADE DE CIÊNCIAS E TECNOLOGIA**

**UNIVERSIDADE DE COIMBRA**

**PRINCÍPIOS DE PROGRAMAÇÃO PROCEDIMENTAL**  
**LICENCIATURA EM ENGENHARIA INFORMÁTICA**  
**2018/2019– 2º SEMESTRE**

**PROJETO**

**“Planeamento de Viagens no DEI”**

2018274233 – Ana Beatriz Ribeiro Correia Ferreira Marques

2018295474 – Bruno Ricardo Leitão Faria

## INTRODUÇÃO

No âmbito da cadeira de Princípios de Programação Procedimental foi-nos pedido que em linguagem C implementássemos uma aplicação para planear uma viagem do DEI.

Para a realização deste projeto tivemos de explorar profundamente listas ligadas, ficheiros e alocação de memória, usando funções do sistema tal como novas, criadas por nós. Em seguida são descritas as estruturas, listas ligadas, ficheiros, bem como o modo de funcionamento do programa.

### ESQUEMA COM AS ESTRUTURAS DE DADOS UTILIZADAS

Structs para obter lista de cidades e respetivos pdis

```
typedef struct Cities{  
    char *name;  
    PDI *pdi;  
    int pop;  
    struct Cities *next;  
}CITIES;
```

```
typedef struct Pdi{  
    char *name, *info;  
    int pop,hot;  
    struct Pdi *next;  
}PDI;
```

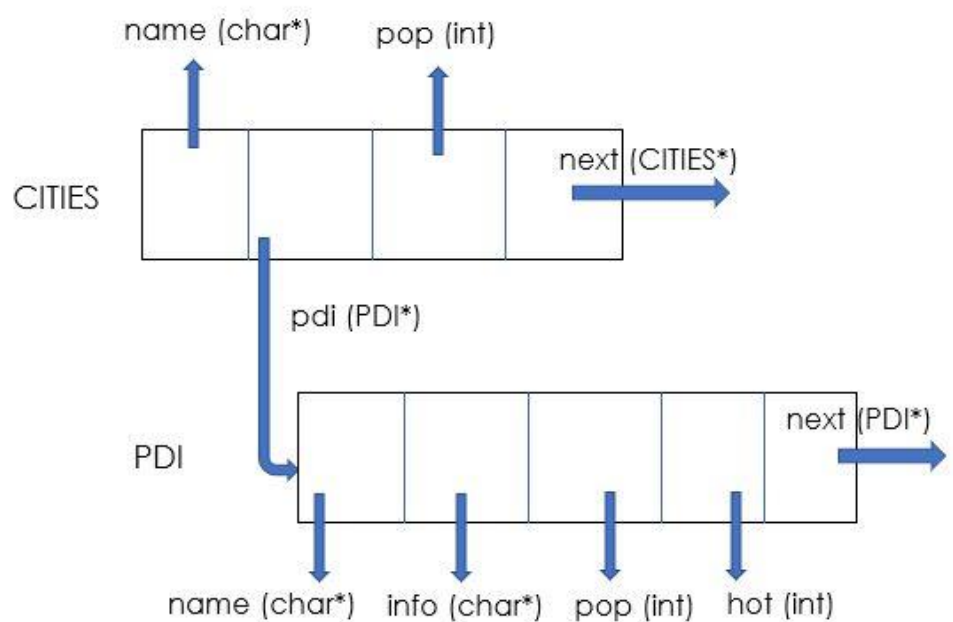


fig. 1

Structs que guardam as informações do user

```
typedef struct USER{  
    char *name, *address,  
    *date_of_birth,  
    *phone_number;  
    USERInfo info;  
    struct USER *next;  
}User;
```

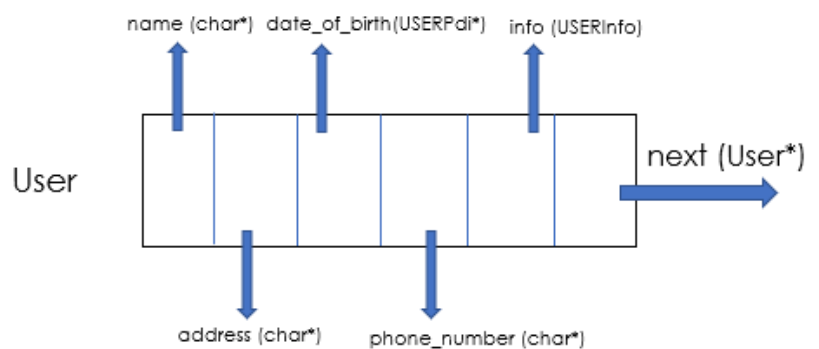


fig. 2

```
typedef struct {
    char *hot, *hot_city;
    USERPdi *pdi;
    USERCity *cities;
}USERInfo;
```

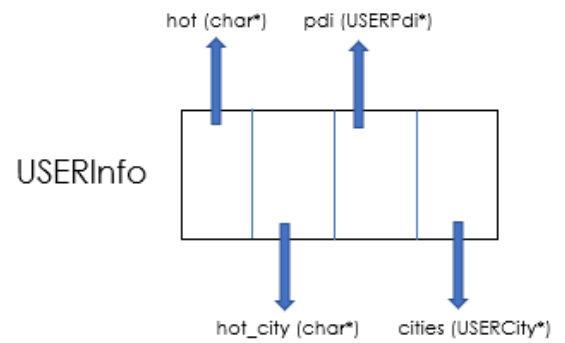


fig. 3

Structs que guardam as preferências do user

```
typedef struct UserCity{
    char *name;
    struct UserCity *next;
}USERCity;
```

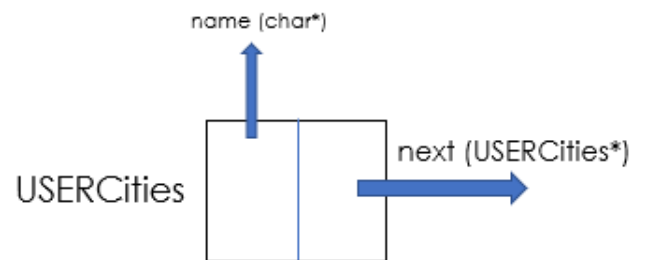


fig. 4

```
typedef struct UserPdi{
    char *name, *city;
    struct UserPdi *next;
}USERPdi;
```

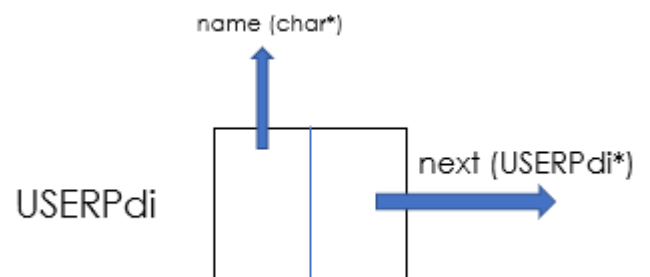


fig. 5

Structs de users e as suas preferências de cidades e pdis (Resumo)

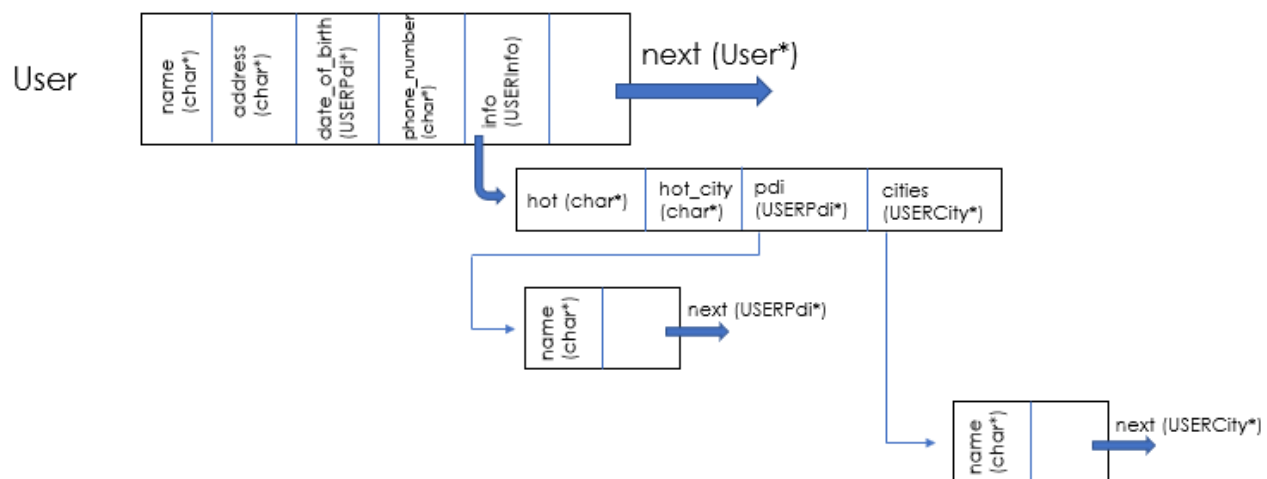


fig. 6

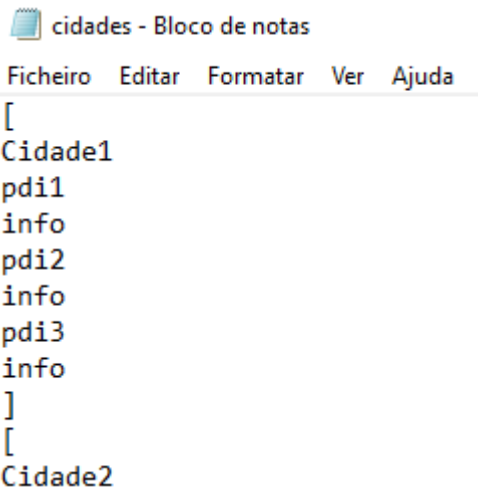
## EXPLICAÇÃO DO CÓDIGO:

Como primeiro passo, de forma a manter o projeto e o código organizado, criamos vários ficheiros header de forma a separar as diferentes funções consoante a sua utilidade, obtemos assim:

func.h – declarar variáveis globais e declarar as structs.

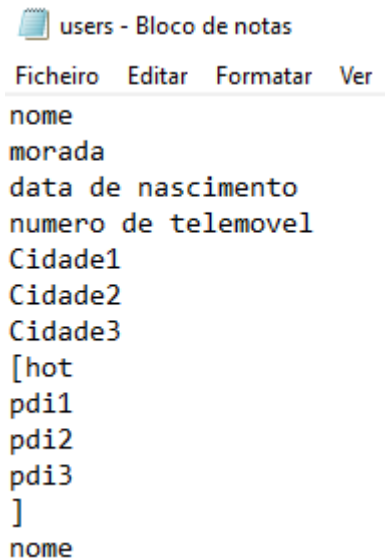
read\_file.h- funções de leitura de ficheiros, colocando a informação de um ficheiro numa lista ligada e nas respetivas structs, tendo em conta os seguintes formatos:

formato das cidades.txt



```
[
Cidade1
pdi1
info
pdi2
info
pdi3
info
]
[
Cidade2
```

formato dos users.txt



```
nome
morada
data de nascimento
numero de telemovel
Cidade1
Cidade2
Cidade3
[hot
pdi1
pdi2
pdi3
]
nome
```

fig. 7

insert\_sort.h- funções de organização de listas ligadas, bubblesort por ordem alfabética e bubblesort por popularidade.

write\_file.h- escrita no ficheiro de users, colocando as informações dos structs da lista ligada no formato em cima referido (fig.7- users.txt).

format.h –funções que verificam o conteúdo das strings inseridas pelo utilizador: nome, data de nascimento, numero de telemóvel e choice.

### Para além disso temos no projeto:

main.c – menus de seleção e as funções necessárias para o funcionamento do código;

cidades.txt- ficheiro fixo já dado e nunca alterado nem ordenado, com todas as cidades e respetivos pdis. A leitura deste ficheiro cria uma lista ligada das cidades e pdis (L1) e é depois ordenada alfabeticamente.

users.txt- ficheiro que guarda as informações do user e as suas preferências (hot, pdis, cidades). Ao contrário de cidades.txt, é um ficheiro vazio inicialmente e sempre que o programa termina é inserido a lista de utilizadores.

**NOTA:** os formatos de cidades.txt e users.txt estão representados na fig.7.

Através de múltiplos structs definimos assim 3 listas ligadas:

L1- lista das cidades e respetivos pdis, representado na figura 1.

L2-lista de utilizadores, informações e preferências.

L3-lista das cidades e respetivos pdis ordenado pelas preferências dos utilizadores.

## MODO DE FUNCIONAMENTO

Quando se inicia o programa, são lidos os ficheiros cidades.txt e users.txt usando, respetivamente, **read\_file()** e **read\_user()**. Em seguida, utilizamos a função **AlfabetiBubbleSort()**, que ordena por ordem alfabética a lista de cidades e pdis. Após isso, é alocado memória para as preferências do utilizador e o telemóvel do user (é a partir deste que procuramos na lista de users para poder alterar). Após isso, temos um menu com 3 opções:

- 0- sair
- 1- criar novo user
- 2- escolher um user já existente na lista

(NOTA: sempre que o utilizador seleciona um char que não é 0,1 ou 2 (isto é, default) volta ao menu.)

Em 0, sai-se do menu, abre-se o users.txt e é escrito a lista ligada dos users, ou seja, é guardado todas as informações e preferências de todos os users, **WriteUserFile()**. É libertada a memória de todas as listas ligadas e as informações do user (se existir algum user selecionado). Deste modo o programa é terminado.

Em 2, a função **FindUser()** verifica se o número de telemóvel pertence à lista de utilizadores.

Caso esteja: vamos procurar as preferências do user, **GetUserCities()** e **GetUserPdi()**, e iniciamos a função **UserInterface()**.

Caso não esteja: pergunta ao utilizador se pretende registar um novo utilizador e continua para a opção 1.

Em 1, é iniciada a função **Resgister()** na qual o user coloca a informação, e apenas termina a função quando nome, morada, data de nascimento e telemóvel estiver no formato correto, **format\_user()**, evitando possíveis erros de input. Após isso, inicia a função **UserInterface()**.

Continuando com o programa, na entrada na função **UserInterface()**, por outras palavras, o menu principal, temos as seguintes opções:

- 0- sair
- 1- editar o user
- 2- ver as informações do user
- 3- adicionar cidade às preferências
- 4- remover cidade das preferências
- 5- adicionar pdi às preferências
- 6- remove pdi das preferências

- 7- escolher hot
- 8- remover hot
- 9- construir viagem

Em 0, recorremos à função **free()** para libertar a memória do input do utilizador. De seguida, saímos do menu principal e voltamos ao menu inicial.

Em 1, é redirecionado para um novo menu, **EditUser()**, onde poderá escolher 0-sair,1-nome,2-morada,3 data de nascimento, 4-numero de telemóvel. Assim altera a opção seleccionada no user, de acordo com o formato correto **format\_user()**.

Em 2, a função **PrintUserInfo()** faz print de todas as informações do user.

Em 4, lemos a lista de cidades preferidas para verificar se está vazia.

Caso esteja: pergunta se deseja adicionar nova cidade às preferências e continua para a opção 3.

Caso não esteja: é iniciada **RemoveCity()** que faz print das cidades preferidas e pergunta qual deseja apagar.

Em 3, usamos a função **Addcity()** para adicionar uma nova cidade à lista de preferências. Sendo que apenas podemos ter 3 cidades preferidas, esta função também verifica se existem no máximo duas cidades no user.

Em 6, lemos a lista de pdis para verificar se está vazia.

Caso esteja: pergunta se deseja adicionar nova cidade às preferências e continua para a opção 3.

Caso não esteja: é iniciada **RemovePdi()** que faz print das cidades preferidas e pergunta qual deseja apagar.

Em 5, adicionamos novo pdi às preferências do user com **AddPdi()**.

Em 8, verificamos se hot está vazio.

Caso esteja: pergunta se deseja adicionar novo hot e se desejar adicionar continua para a opção 7.

Caso não esteja: é iniciada **RemoveHot()**.

Em 7, se hot estiver vazio (apenas existe um hot por utilizador), adicionamos através de **AddHot()**.

Em 9, recorrendo as funções **SaveUser()**, **GetNumUsers()** e **GetPopularity()**, respetivamente, guardamos a informação do user, obtemos o numero de users e adicionamos à lista da viagem, em cada pdi e

cidade, o número de utilizadores que seleccionaram como preferência. De seguida, ordenamos a lista L1 pela popularidade, **PopBubbleSort()** e criamos a viagem do user e verificamos a sua popularidade, **MakeTrip()** e **RateTrip()**. No final, voltamos a organizar a lista L1 por ordem alfabética, **AlfabeticBubbleSort()**.

A função **MakeTrip()** apenas é possível quando estão seleccionadas pro user 3 cidades preferidas, e utiliza a função

**NOTA:** a viagem é composta por 3 cidades cada ela com 3 pdis a visitar

Prioridades da viagem: 1º hot, 2º pdis do user ordenados por popularidade, 3ª pdis mais populares que não foram seleccionados pelo user (apenas possível se para a cidade seleccionada o user apenas escolher menos de 3 hot/pdis)

Exemplo:

Lista popularidade Cidade1- pdi1, pdi2, pd3, pd4, pdi5

Cidade1 Hot- pdi5

Cidade1 Pdis- pdi2

Os pdis seleccionados para a viagem na Cidade1 foram, por ordem, pdi5, pdi2 e pdi1.

A função **RateTrip()** é chamada no final da função **MakeTrip()** e calcula a popularidade da viagem criada pelo user, sendo a taxa de popularidade:

$$\left( \% \text{ de users que } >1 \text{ cidades favoritas na viagem} + \% \text{ de users que hot está na viagem} + \%(\text{pontos}^1 \text{ na viagem} / \text{pontos}^1 \text{ total}) \right) / 3$$

pontos<sup>1</sup> - soma de todos os pdis e hots de todos os users

Exemplo:

Viagem – Cidade1: pdi5, pdi2, pdi1, Cidade2: pdi3, pdi2, pdi4, Cidade3: pdi2, pdi1, pdi4

Número de users= 4

Cidades preferidas dos users: user1: , user2: Cidade1, Cidade4, user3: Cidade2 , user4: Cidade1, Cidade2, Cidade3

Hots dos users: user1: Cidade1-pdi4, user2: Cidade3-pdi3, user3: Cidade2-pdi3, user4: Cidade1-pdi5

Pontos no total=30

Pontos na viagem: Cidade1: 2- pdi5, 1- pdi2, 4-pdi1, Cidade2: 1-pdi3, 3-pdi2, 3-pdi4, Cidade 3: 3-pdi3, 2-pdi2, 1-pdi4

Popularidade da viagem= ( ( 3 / 4 + 2 / 4 + (2+1+4+1+3+3+3+2+1)/30 ) \*100 ) / 3=  
= ( ( 0.75+0.5+ 0.67)\*100) / 3= 64%

## ALGUMAS FUNÇÕES (E BREVE EXPLICAÇÃO)

**Len()**- devolve o tamanho de um array

**AddUserCP()**- chamada pelas funções **AddCity()** e **AddPdi()**, adiciona o novo local/pdi à lista de locais/pdis do user

**FixInput()**- retira o '\n' no final do array

**format()**- verifica se o input são apenas números

**CleanInput()**- quando o input ultrapassa o máximo de caracteres, esta função limpa os que estão a mais

**GetNumUsers()**- devolve o número de utilizadores registados

**FindUser()**- determina se o utilizador já está registado através do numero de telemóvel

**Register()**- registo de um novo utilizador

**EditUser()**- altera as informações do utilizador

**UserInterface()**- menu principal

**PrintUserInfo()**- apresenta ao utilizador as informações e seleções já realizadas

**SaveUser()**- guarda o utilizador na lista ligada de users

**format\_user()**- verifica o formato das informações do user

**AddCity()**- adiciona uma cidade às preferências do user

**RemoveCity()**- retira uma cidade das preferências do user

**AddPdi()**-adiciona um pdi às preferências do user

**RemovePdi**- retira um pdi das preferências do user

**AddHot**- adiciona hot às preferências do user

**RemoveHot**- retira hot das preferências do user

**GetPopularity()**- atribui a cada cidade/pdi o número de user que o seleccionaram