DEPARTAMENTO DE INFORMÁTICA

ALGORITMOS E ESTRUTURAS DE DADOS

Relatório do trabalho

FCT Boleia

55091 João Naves MIEEC

Professor das aulas práticas: Fernanda Barbosa

Turno prático: P7

3 de Junho de 2019

Declaração

Declaramos que o trabalho apresentado foi realizado pelos membros do grupo e que a distribuição das tarefas pelos membros do grupo foi equilibrada.

(Assinaturas .....)

Índice

1 Introdução

O objectivo deste trabalho é o desenvolvimento de um sistema de partilha de carros, em que se oferecem e pedem “boleias”.

Para este programa existem dois modos de funcionamento:

- Modo default : Neste modo é onde os users se podem registar e fazerem o seu Log in bem como terminar a execução da linha de comandos
- Modo session: Neste modo é onde os users já realizaram o seu Log in e podem então executar ações relacionadas com viagens como criar uma viagem listar viagens assim como registar se em viagens já existentes, e ainda poder terminar a sua sessão.

E foram estas as fundações que levaram à criação deste trabalho.

2 Tipos Abstractos de Dados

2.1 Enquadramento

Uma das carecterísticas mais condicionantes deste trabalho foi a existência de apenas uma deslocação por dia por utilizador. Isto influenciou o facto de ter de existir uma estrutura de utilizador para guardar de uma forma mais eficiente as viagens que este utilizador iria criar e registar -se em.

Tanto este como outras condições levaram a tomadas de decisões relacionadas com quais as estruturas de dados mais apropriadas para certas situações.

2.2 Definição dos Tipos Abstractos de Dados

2.2.1 Sessão (session.h & session.c)

- **initialize()** – comeca uma sessao do programa
  - Parametros:
    - capUsers (capacidade maxima prevista de users)
  - Retorno:
    - (struct) session
- **registUser()** – regista um user na base de dados
  - Parametros:
    - user (mail)
    - user (nome)
    - password
    - session

- **userLogin()** – faz login de um user
  - Parametros:
    - user (mail)
    - password
    - session
  - Retorno:
    - 1 – se o login foi feito com sucesso
    - 0 – se o contrario
- **userCheck()** – verifica se o user existe
  - Parametros:
    - user (mail)
    - session
  - Retorno:
    - 1 – caso exista
    - 0 – caso contrario
- **userName()** – devolve o nome do user
  - Parametros:
    - user (mail)
    - session
  - Retorno:
    - (char *) nome do user
- **userMail()** – devolve o mail de um user
  - Parametros:
    - user (struct)
  - Retorno:
    - (char *) mail do user

- **userCheckBol()** – verifica se um user esta registado numa viagem
  - Parametros:
    - user (mail)
    - data (da viagem)
    - session
  - Retorno:
    - 1 – se existe
    - 0 – caso contrario
- **newDeslocacao()** – regista uma nova deslocacao para um user
  - Parametros:
    - user (mail)
    - session
    - local de origem
    - local de destino
    - datacmd (uma string com a data hora duracao e numero de lugares)
- **delDeslocacao()** – elimina uma deslocacao
  - Parametros:
    - user (mail)
    - data (da viagem)
    - session

- **checkDeslocacao()** – verifica se existe uma deslocacao de um user numa data
  - Parametros:
    - user (mail)
    - data
    - session
  - Retorno:
    - 1 – caso exista
    - 0 – caso contrario
- **numEmptySeats()** – devolve o numero de lugares livres numa viagem
  - Parametros:
    - user (mail)
    - data
    - session
  - Retorno:
    - (int) numero de lugares livres
- **newPendura()** – adiciona um novo pendura a viagem
  - Parametros:
    - pendura (mail)
    - condutor (mail)
    - data
    - session

- **existUsersReg()** – verifica se existem useres registados numa viagem
  - Parametros:
    - user (mail)
    - data
    - session
  - Retorno:
    - 1 – caso exista
    - 0 – caso contrario
- **delBoleia()** – retira um user da boleia
  - Parametros:
    - user (mail)
    - data
    - session
- **listDeslocacoes()** – devolve iterador para listar as deslocacoes
  - Parametros:
    - user (mail)
    - session
  - Retorno:
    - (iterador) iterador de deslocacoes
- **listBoleias()** – devolve iterador para listar as boleias em que esta registado
  - Parametros:
    - user (mail)
    - session
  - Retorno:
    - (iterador) iterador de boleias

- **listDatas()** – devolve iterador para listar as viagens de uma data
  - Parametros:
    - data
    - session
  - Retorno:
    - iterador) iterador de viagens de uma data
- **listUsersReg()** – iterador para listar users registados numa boleia
  - Parametros:
    - boleia
  - Retorno:
    - (iterador) iterador de users registados

2.2.2 UTILIZADORES (USER.H & USER.C)

- **fillUser()** – preenche uma estrutura de user
  - Parametros:
    - mail
    - nome
    - password
  - Retorno:
    - (struct) user
- **getDeslocacao()** – devolve a viagem relacionada a data
  - Parametros:
    - user
    - data
  - Retorno:
    - (struct) boleia

- **getBoleia()** – devolve a boleia relacionada a data
  - Parametros:
    - user
    - data
  - Retorno:
    - (struct) boleia
- **getVecBoleia()** – devolve um vetor com as viagens de um iterador
  - Parametros:
    - iterador
    - tamanho do vetor
  - Retorno:
    - (boleia *) vetor de viagens

- **addDeslocacao()** – adicona uma nova deslocacao ao user
  - Parametros:
    - user
    - boleia
- **remDeslocacao()** – remove uma deslocacao do user
  - Parametros:
    - user
    - data
- **addBoleia()** – adiciona uma boleia a lista de boleias a que o user esta relacionado
  - Parametros:
    - user
    - boleia
- **remBoleia()** – remove o user de uma boleia em que estava registado
  - Parametros:
    - user
    - data
- **getnDeslocacoes()** – devolve o numero de deslocacoes que o user tem
  - Parametros:
    - user
  - Retorno:
    - (int) numero de deslocacoes

- **getDeslocacaoOrd()** – devolve um iterador com as deslocacoes do user ordenadas por data
  - Parametros:
    - user
  - Retorno:
    - (iterador) lista de deslocacoes
- **getBoleiasOrd()** – devolve um iterador com as boleias em que o user esta registado ordenadas por data
  - Parametros:
    - user
  - Retorno:
    - (iterador) lista de viagens registadas
- **binarySearch()** – faz a procura do local onde o elemento tem de ser inserido
  - Parametros:
    - vetor de viagens
    - elemento a pesquisar
    - minimo da procura
    - maximo da procura
    - funcao de comparacao com outputs e inputs semelhantes ao strcmp
    - funcao de recolha de dados com output char*
  - Retorno:
    - (int) posicao a ser inserido

- **insertionSort()** – ordena viagens por data
  - Parametros:
    - vetor de viagens
    - tamanho do vetor
    - funcao de comparacao com outputs e inputs semelhantes ao strcmp
    - funcao de recolha de dados com output char*
- **compareDate()** – compara duas datas em formato string
  - Parametros:
    - data1
    - data2
  - Retorno:
    - –1 – data1<data2
    - 0 – data1=data2
    - 1 - data1>data2
- **checkpass()** – verifica se a pass do user e igual a inserida
  - Parametros:
    - user
    - password a verficar
  - Retorno:
    - 1 – caso esteja correta
    - 0 – caso contrario
- **getName()** – devolve o nome do user
  - Parametros:
    - user
  - Retorno:
    - (char *) nome do user

- **getMail()** – devolve o mail do user
  - Parametros:
    - user
  - Retorno:
    - (char *) mail do user

2.2.3 VIAGENS(BOLEIA.H & BOLEIA.C)

- **fillBoleia()** – preenche uma boleia e devolve
  - Parametros:
    - mail
    - origem
    - destino
    - data (toda a informacao relacionada com a data, hora, duracao e lugares)
  - Retorno:
    - (struct) boleia
- **addPendura()** – adiciona um pendura a viagem
  - Parametros:
    - boleia
    - (user)pendura (usou se o void * para evita recursividade)
- **remPendura()** – remove um pendura da viagem
  - Parametros:
    - boleia
    - pos (posicao do user a retirar)
- **getPosUser()** – devolve a posicao de um user numa boleia
  - Parametros:
    - boleia
    - email do user
  - Retorno:
    - (int) posicao do user na boleia

- **seqPenduras()** – devolve um iterador da sequencia de penduras
  - Parametros:
    - boleia
  - Retorno:
    - (iterador) iterador da Sequencia
- **giveMaster()** – devolve o dono da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (char *) email do dono da viagem
- **giveOrigem()** – devolve o local de comeco da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (char *) origem da viagem
- **giveDestino()** – devolve o local de destino da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (char *) destino da viagem
- **giveDate()** – devolve a data da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (char *) data da viagem

- **giveHorah()** – devolve a hora da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (int) hora da viagem
- **giveHoram()** – devolve os minutos da hora de viagem
  - Parametros:
    - boleia
  - Retorno:
    - (int) minutos da hora viagem
- **giveDuracao()** – devolve a duracao da viagem
  - Parametros:
    - boleia
  - Retorno:
    - (int) duracao da viagem
- **giveLugares()** – devolve o numero de lugares da viagem (no momento inicial)
  - Parametros:
    - boleia
  - Retorno:
    - (int) numero de lugares
- **givenumPenduras()** – devolve o numero de penduras inscritos
  - Parametros:
    - boleia
  - Retorno:
    - (int) numero de penduras inscritos

2.3 Implementação dos Tipos Abstractos de Dados

2.3.1 Sessão (session.h & session.c)

Neste TAD é onde é feita a gerencição dos dados relacionados com o programa geral. Semelhante a como seria feito numa base de dados é guardado as informações da seguinte forma:

- Dicionário de utilizadores
- Dicionario “Duplo” de apontadores para Viagens ordenadas por data – Neste dicionario em cada chave de data irá ser criado um outro dicionario em que a chave seria o utilizador. Desta forma temos uma forma rapida de aceder as Viagens de uma certa data sem gastar espaco extra devido ao uso de apontadores

As funções presentes são:

- **initialize()** - função que cria uma sessão do programa onde vao ser guardados todos os dados sobre todos os utilizadores existentes e as subsquentes viagens criadas por estes bem como informacoes sobre as viagens em que esta registado.
- **registUser()** - função que vai criar um utilizador com o auxilio da TAD de utilizador e depois guarda o no dicionario da sessao
- **userCheck()** - funcao que verifica se um utilizador existe no dicionario da sessao
- **userLogin()** - funcao que vai tratar do login de um utilizador com o auxilio da TAD de utilizador
- **userName()** - funcao que vai retornar o nome de um utilizador com um certo e-mail com o auxilio da TAD de utilizador
- **userCheckBol()** - funcao que verifica se um utilizador já se registou uma boleia numa certa data sendo auxiliado pelo TAD de utilizador e TAD de viagens
- **checkDeslocacao()** - funcao que verifica se já existe uma viagem deste utilizador para uma certa data tendo como auxilio a TAD de utilizador
- **newDeslocacao()** - funcao que vai preencher uma viagem para um certo utilizador e por sua vez guarda la na estrutura do utilizador e na estrutura da sessão tendo como auxilio as TADs de utilizador e de viagens
- **delDeslocacao()** - funcao que vai eleminar uma viagem de todas as estruturas onde estava guardada tendo como auxilio a TAD de utilizador

- **numEmptySeats()** - funcao que vai retornar o numero de lugares livres de uma viagem com uma certa data tendo como auxilio as TADs de utilizador e de viagem
- **newPendura()** - funcao que vai adicioar um utilizador como pendura a viagem de outro user registando o nos respetivos lugares nas estruturas dos utilizadores e da viagem com o auxilio dos TADs de utilizador e de viagem
- **existUsersReg()** – funcao que verifica se existem utilizadores registados numa viagem com o auxilio das TADS de utilizador e de viagem
- **delBoleia()** - funcao que remove o registo de um user de uma viagem de um certo dia
- **listDeslocacoes()** - funcao que retorna um iterador das deslocacoes de um user tendo o auxilio da TAD de utilizador
- **listBoleias()** - funcao que retorna um iterador das boleias em que um user esta registado tendo o auxilio da TAD de utilizador
- **listDatas()** - funcao que retorna um iterador das viagens que existem numa certa data acedendo ao dicionario “duplo” e depois organizando as viagens por ordem alfabetica
- **listUsersReg()** - funcao que retorna um iterador dos e-mails dos utilizadores registados numa certa viagem

As complexidades deste TAD são:

initialize()	O(1)
registUser()	O(1)
userLogin()	O(1)
userCheck()	O(1)
userName()	O(1)
userMail()	O(1)
userCheckBol()	O(n) – n penduras
checkDeslocacao()	O(1)
newDeslocacao()	O(1)
delDeslocacao()	O(1)
numEmptySeats()	O(1)
newPendura()	O(1)
existUsersReg()	O(1)
delBoleia()	O(n) – n penduras
listDeslocacoes()	O(nlogn) – n deslocacoes
listBoleias()	O(nlogn) – n boleias registadas
listDatas()	O(nlogn) – n deslocacoes
listUsersReg()	O(n) – n penduras

2.3.2 UTILIZADORES (USER.H & USER.C)

Neste TAD é onde é feita a gerenciacao dos dados especificos de cada utilizador e onde existem funcoes relacionadas com a gerenciação desses dados. A estrutura presente é:

- Email do utilizador
- Nome do utilizador
- Password (encriptada)
- Dicionario com as Viagens criadas pelo utilizador
- Número de Viagens registadas
- Dicionario com as Viagens onde o utilizador esta registado
- Número de Viagens onde esta registado

As funções presentes são:

- **xorstring()** - uma das funcoes responsaveis pela encriptacao da password. Esta funcao em particular faz a operacao xor de uma string com a chave de encriptação
- **encryption()** - a funcao responsavel pela encriptação da password e guarda la na estrutura
- **fillUser()** - funcao responsavel por preencher as informações sobre um utilizador e guarda las na estrutura do utilizador
- **getDeslocacao()** - funcao responsavel por devolver uma viagem com uma certa data
- **getBoleia()** - funcao responsavel por devolver uma viagem com uma certa data em que o utilizador estava registado
- **getVecBoleia()** - funcao que devolve um vetor com as viagens presentes num iterador
- **addDeslocacao()** - funcao que adiciona uma viagem a estrutura do utilizador
- **remDeslocacao()** - funcao que remove uma viagem da estrutura do utilizador
- **addBoleia()** - funcao que adiciona uma boleia a estrututa do utilizador
- **remBoleia()** - funcao que remove uma boleia da estrutura da estrutura do utilizador
- **getnDeslocacoes()** - funcao que retorna o número de deslocacoes que um utlizador tem registadas

- **getDeslocacaoOrd()** - funcao que atraves de um vetor de viagens que um utiliador registou, vetor este retirado de um iterador, vai ser ordenado e criado um novo iterador com este vetor ordenado
- **getBoleiasOrd()** - funcao que atraves de um vetor de viagens que um utiliador se registou em, vetor este retirado de um iterador, vai ser ordenado e criado um novo iterador com este vetor ordenado
- **binarySearch()** - funcao de procura dictomica que ira procurar a posicao correta para um elemento ser inserido de forma ordenada
- **insertionSort()** - funcao que ira ordenar um vetor de viagens atraves de inserção ordenada auxiliada por procura dictomica
- **compareDate()** - funcao que compara duas datas dadas em forma de string
- **checkpass()** - funcao que verifica se a password inserida está correta segundo a estrutura do utilizador
- **getName()** - funcao que devolve o nome do utilizador
- **getMail()** - funcao que devolve o e-mail do utlizador

As complexidades deste TAD são:

xorstring()	O(1)
encryption()	O(1)
fillUser()	O(1)
getDeslocacao()	O(1)
getBoleia()	O(1)
getVecBoleia()	O(n) – n viagens
addDeslocacao()	O(1)
remDeslocacao()	O(1)
addBoleia()	O(1)
remBoleia()	O(1)
getnDeslocacoes()	O(1)
getDeslocacaoOrd()	O(nlogn) – n viagens
getBoleiasOrd()	O(nlogn) – n viagens
binarySearch()	O(logn)
insertionSort()	O(nlogn) – n viagens
compareDate()	O(1)
checkpass()	O(1)
getName()	O(1)
getMail()	O(1)

2.3.3 VIAGENS(BOLEIA.H & BOLEIA.C)

Neste TAD é onde é feita a gerenciação dos dados sobre as viagens. A estrutura que está definida é:

- Email do dono da viagem
- Origem
- Destino
- Data
- Hora de inicio
- Minuto de inicio
- Duração
- Lugares livres inicialmente
- Sequencia dos e-mails dos penduras
- Número de penduras já estão inscritos

As funcoes presentes são:

- **fillBoleia()** - funcao que preenche a estrutura de uma viagem
- **addPendura()** - funcao que adiciona um e-mail de um pendura a viagem
- **remPendura()** - funcao que remove um e-mail de um pendura a viagem
- **getPosUser()** - funcao que devolve a posicao de um pendura na sequencia de pendura
- **seqPenduras()** - funcao que devolve um iterador com os e-mails dos penduras
- **giveMaster()** - funcao que devolve o utilizador dono da viagem
- **giveOrigem()** – funcao que devolve a origem da viagem
- **giveDestino()** - funcao que devolve o destino da viagem
- **giveDate()** - funcao que devolve a data da viagem
- **giveHorah()** - funcao que devolve a hora da viagem
- **giveHoram()** - funcao que devolve os minutos da viagem
- **giveDuracao()** - funcao que devolve a duracaoda viagem
- **giveLugares()** - funcao que devolve o numero de lugares inicial da viagem
- **givenumPenduras()** - funcao que devolve o numero de penduras da viagem

As complexidades deste TAD são:

fillBoleia()	O(1)
addPendura()	O(1)
remPendura()	O(1)
getPosUser()	O(n) – n penduras
seqPenduras()	O(n) – n penduras
giveMaster()	O(1)
giveOrigem()	O(1)
giveDestino()	O(1)
giveDate()	O(1)
giveHorah()	O(1)
giveHoram()	O(1)
giveDuracao()	O(1)
giveLugares()	O(1)
givenumPenduras()	O(1)

3 Programa Principal

No programa principal é onde se encontram as respostas aos seguintes comandos:

Quando em modo inicial:

- Ajuda - Mostra os comandos disponiveis neste modo
- Regista - Regista um utilizador
  - Parametros:
    - Comando escrito
    - Sessao do programa
- Entrada - Faz o Login de um utilizador
  - Parametros:
    - Utilizador
    - Comando escrito
    - Sessao do programa
  - Retorno:
    - 1 - se foi bem sucedido
    - 0 - caso contrario
- Termina - Termina o programa

Quando em modo de sessão:

- Ajuda - Mostra os comandos disponiveis neste modo
- Nova - Cria uma nova viagem para o utilizador
  - Parametros:
    - Utilizador
    - Sessao do programa
- Boleia - Regista o utilizador numa viagem de outro
  - Parametros:
    - Utilizador
    - Comando escrito
    - Sessao do programa

- Lista: ⇒ Parametros : Utilizador, Comando escrito, Sessao do programa
  - Minhas - Mostra as deslocacoes do utilizador
    - Parametros:
      - Utilizador
      - Sessao do programa
  - Boleias - Mostra as viagens em que o utilizador se registou
    - Parametros:
      - Utilizador
      - Sessao do programa
  - [User] - Mostra as deslocacoes de um utilizador
    - Parametros:
      - Utilizador descrito do comando
      - Sessao do programa
  - [Data] - Mostra as viagens de uma certa data
    - Parametros:
      - Data descrita do comando
      - Utilizador
      - Sessao do programa
- Retira - Retira o utilizador de uma viagem em que esteja registado
  - Parametros:
    - Utilizador
    - Comando escrito
    - Sessao do programa
- Remove - Elimina uma deslocacao se não houver penduras inscritos
  - Parametros:
    - Utilizador
    - Comando escrito
    - Sessao do programa
- Sai - Termina a sessão do utilizador

Estes são os comandos que a linha de comando nos permite usar pelo que se não for este os comandos inseridos o programa retorna : “Comando invalido.”

No programa inicial para alem dos TADs da professora iterador e sequencia ainda temos os TADs de viagens para dar display das viagens e de sessao para poder gerir os utilizadores e guardar as suas respetivas informações.

Complexidades Temporais de cada comando:

- Modo Inicial:

Ajuda	O(1)
Regista	O(1)
Entrada	O(1)
Termina	O(1)

- Modo de Sessão:

Ajuda	O(1)
Nova	O(1)
Boleia	O(n)
Lista	$O(n_{[deslocacoes]}*n_{[penduras]})$
Retira	O(1)
Remove	O(1)
Sai	O(1)

Devido a ter pouco conhecimento sobre como exprimir a complexidade espacial não vou me exprimir acerca deste ponto.

4 Conclusões

Apreciei muito a ideia deste trabalho no entanto acredito que se houvesse mais tempo para se abordar mais pontos sobre certos Tipos Abstratos de Dados poderia ser um trabalho ainda mais apelativo.

No entanto, vários pontos poderiam ter sido melhorados, havendo tempo para tal, por exemplo, a pesquisa da posição de um utilizador na sequencia poderia ser feito talvez atraves de uma pesquisa dictómica.

Apesar disso, foi um trabalho desafiador em muitos aspetos e claramente há particularidades que poderiam ser melhoradas neste trabalho.

5 Bibliografia

5.1 MAIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#include "iterador.h"
#include "sequencia.h"

#include "boleia.h"
#include "user.h"
#include "session.h"

#define MAXL 50 //MAX LINE CHARACTERS
#define MAXC 10 //MAX COMMAND CHARACTERS

#define MINPASS 4 //PASS MINIMUM
#define MAXPASS 6 //PASS MAXIMUM

#define MAXUSERS 10000 //MAX USERS

//MENUS
void inicialmenu(session s);
void sessionmenu(char * user, session s);
int imenuselection(char * cmd);
int smenuselection(char * cmd);

//INICIAL SELECTIONS
void selecAjudaI();
int selecEntrada(char * user,char * cmd, session s);
int selecRegista(char * cmd, session s);
void selecTermina();

//SESSION SELECTIONS
void selecAjudaS();
void selecNova(char * usr, session s);
void selecBoleia(char * cmd, session s, char * mail);
void selecRetira(char * user, char * cmd, session s);
void selecRemove(char * user, char * cmd, session s);
void selecLista(char * cmd, session s, char * user);
void listMenu(int selection, session s, char * user, char *
option);
void listaMinhas(session s, char * user);
void listaUser(session s, char * user);
void listaBoleias(session s, char * user);
void listaDate(session s, char * date, char * user);
void selecSai(char * user, session s);
```

```
//HELPER FUNCTIONS
int verifypass(char * cmd);
int checkstrdate(char * date);
int checkdate(int dia, int mes, int ano);
int checkhoranminuto(int hora,int minuto);
char * lowerIN(char * input, int size);

int main(){
    session s = initialize(MAXUSERS);
    inicialmenu(s);
    return 0;
}

//*INICIAL MENU
void inicialmenu(session s){
    char cmdi[MAXL];
    char selecti[MAXC] = "NULL";
    int selecinti = '\0';
    char * user = (char*) malloc(MAXL);
    while(selecinti ≠ 3){
        selecti[0] = '\0';
        printf("> ");
        fgets(cmdi,MAXL-1,stdin);
        sscanf(cmdi,"%s",selecti);
        selecinti = imenuselection(selecti);
        switch(selecinti){
            case 0:
                selecAjudaI();
                break;
            case 1:
                sscanf(cmdi,"%*s %s",user);
                if(selecEntrada(user,cmdi,s)){
                    sessionmenu(user,s);
                }
                break;
            case 2:
                if(selecRegista(cmdi,s)){
                    printf("Registo efetuado.\n");
                }
                else{
                    printf("Registo nao efetuado.\n");
                }
                break;
            case 3:
                selecTermina();
                break;
            default:
                printf("Comando inexistente.\n");
                break;
        }
    }
}

//*SESSION MENU
```

```
void sessionmenu(char * user, session s){
    char cmds[MAXL];
    char selects[MAXC] = "NULL";
    int selecints = '\0';
    while(selecints ≠ 6){
        selects[0] = '\0';
        printf("%s> ",user);
        fgets(cmds,MAXL-1,stdin);
        sscanf(cmds,"%s",selects);
        selecints = smenuselection(selects);
        switch(selecints){
            case 0:
                selecAjudaS();
                break;
            case 1:
                selecNova(user,s);
                break;
            case 2:
                selecBoleia(cmds,s,user);
                break;
            case 3:
                selecRetira(user,cmds,s);
                break;
            case 4:
                selecRemove(user,cmds,s);
                break;
            case 5:
                selecLista(cmds,s,user);
                break;
            case 6:
                selecSai(user,s);
                break;
            default:
                printf("Comando inexistente.\n");
                break;
        }
    }
}

//* MENU FUNCTIONS NUMBER ATRIBUTION

int imenuselection(char * cmd){
    int cmdint = -1;
    char * cmdl = lowerIN(cmd, strlen(cmd));
    char * cmds[] = {"ajuda","entrada","regista","termina"};
    for(int i=0; i<(sizeof(cmds)/8);i++){
        if(!strcmp(cmdl,cmds[i])){
            cmdint = i;
        }
    }
    return cmdint;
}

int smenuselection(char * cmd){
```

```
    int cmdint = -1;
    char * cmdl = lowerIN(cmd, strlen(cmd));
                                char       *       cmds[]       =
{"ajuda","nova","boleia","retira","remove","lista","sai"};
    for(int i=0; i<(sizeof(cmds)/8);i++){
        if(!strcmp(cmdl,cmds[i])){
            cmdint = i;
        }
    }
    return cmdint;
}

//* INICIAL SELECT

void selecAjudaI(){
    printf("ajuda - Mostra os comandos existentes\n");
    printf("termina - Termina a execucao do programa\n");
    printf("regista - Regista um novo utilizador no programa\
n");
       printf("entrada - Permite a entrada (\"login\") dum
utilizador no programa\n");
}

int selecEntrada(char * user,char * cmd, session s){
    int tries = 1;
    char * pass = (char*) malloc(MAXPASS);
    if(userCheck(user,s)){
        printf("password: ");
        fgets(pass,MAXL,stdin);
        while(!userLogin(user,pass,s)){
            printf("Password incorrecta.\n");
            if(tries == 3){
                return 0;
            }
            printf("password: ");
            fgets(pass,MAXL,stdin);
            tries+=1;
        }
    }
    else{
        printf("Utilizador nao existente.\n");
        return 0;
    }
    free(pass);
    return 1;
}

int selecRegista(char * cmd, session s){
    int finish = -1;
    char * user = (char * ) malloc(MAXL);
    char * name = (char * ) malloc(MAXL);
    char * passtmp = (char *) malloc(MAXL);
    char pass[MAXPASS];
    int tries = 1;
```

```
    sscanf(cmd,"%*s %s", user);
    if(userCheck(user,s)){
        printf("Utilizador ja existente.\n");
        finish = 0;
    }
    else{
        printf("nome (maximo 50 caracteres): ");
        fgets(name,MAXL,stdin);
        name[strlen(name)-1] = '\0';


         printf("password (entre 4 e 6 caracteres - digitos e
letras): ");
        fgets(passtmp,MAXL,stdin);
        passtmp[strlen(passtmp)-1] = '\0';

        while (!(verifypass(passtmp)) && tries++≠3){
            printf("Password incorrecta.\n");
                 printf("password (entre 4 e 6 caracteres -
digitos e letras): ");
            fgets(passtmp,MAXL,stdin);
            passtmp[strlen(passtmp)-1] = '\0';
        }
        if(tries≦3){
            strncpy(pass,passtmp,MAXPASS);
            registUser(user,name,pass,s);
            finish = 1;
        }
        else{
            printf("Password incorrecta.\n");
            finish = 0;
        }
    }
    return finish;
}

void selecTermina(){
    printf("Obrigado. Ate a proxima.\n");
}

//* SESSION SELECT

void selecAjudaS(){
    printf("ajuda - Mostra os comandos existentes\n");
       printf("sai  - Termina a sessao deste utilizador no
programa\n");
    printf("nova - Regista uma nova deslocacao\n");
       printf("lista  - Lista todas ou algumas deslocacoes
registadas\n");
       printf("boleia  - Regista uma boleia para uma dada
deslocacao\n");
    printf("retira - Retira uma dada boleia\n");
    printf("remove - Elimina uma dada deslocacao\n");
}
```

```
void selecNova(char * usr, session s){
    char * origem = (char *) malloc(MAXL);
    char * destino = (char *) malloc(MAXL);
    char * datacmd = (char *) malloc(MAXL);
    char * datastr = (char *) malloc(MAXL);
    int dia,mes,ano;
    int hora,minuto = -1;
    int duracao = -1;
    int numLugares = -1;

    fgets(origem,MAXL,stdin);
    origem[strlen(origem)-1] = '\0';
    fgets(destino,MAXL,stdin);
    destino[strlen(destino)-1] = '\0';
    fgets(datacmd,MAXL,stdin);

    sscanf(datacmd,"%d-%d-%d",&dia, &mes, &ano);
    sprintf(datastr,"%d-%d-%d",dia,mes,ano);

                    if((sscanf(datacmd,"%d-%d-%d    %d:%d    %d
%d",&dia,&mes,&ano,&hora,&minuto,&duracao,&numLugares)) ≠ 7)
{
        printf("Dados invalidos.\n");
        printf("Deslocacao nao registada.\n");
    }
                  else    if(!checkdate(dia,mes,ano)    ||    !
checkhoranminuto(hora,minuto) || duracao<0 || numLugares<0){
        printf("Dados invalidos.\n");
        printf("Deslocacao nao registada.\n");
    }
    else if(checkDeslocacao(usr,datastr,s)){
            printf("%s ja tem uma deslocacao registada nesta
data.\n",userName(usr,s));
        printf("Deslocacao nao registada.\n");
    }
    else{
                          sprintf(datacmd,"%d-%d-%d   %d:%d   %d
%d",dia,mes,ano,hora,minuto,duracao,numLugares);
        newDeslocacao(usr,s,origem,destino,datacmd);
             printf("Deslocacao registada. Obrigado %s.\n",
userName(usr,s));
    }

}

void selecBoleia(char * cmd, session s, char * mail){
    char * master = (char*) malloc(sizeof(char)*MAXL);
    char * data = (char*) malloc(sizeof(char)*MAXL);

    sscanf(cmd,"%*s %s %s", master, data);

    if(!userCheck(master,s)){
        printf("Utilizador nao existente.\n");
    }
```

```
    else if(!checkstrdate(data)){
        printf("Data invalida.\n");
    }
    else if(strcmp(mail,master)==0){
         printf("%s nao pode dar boleia a si proprio. Boleia
nao registada.\n",userName(mail,s));
    }
    else if(!checkDeslocacao(master,data,s)){
        printf("Deslocacao nao existe.\n");
    }
    else if(userCheckBol(mail,data,s)){
         printf("%s ja registou uma boleia nesta data. Boleia
nao registada.\n",userName(mail,s));
    }
    else if(numEmptySeats(master,data,s)==0){
          printf("%s nao existe lugar. Boleia nao registada.\
n",userName(mail,s));
    }
    else{
        newPendura(mail,master,data,s);
        printf("Boleia registada.\n");
    }
}

void selecRetira(char * user, char * cmd, session s){
    char * date = (char*) malloc(sizeof(char)*MAXL);

    sscanf(cmd,"%*s %s", date);

    if(!checkstrdate(date)){
        printf("Data invalida.\n");
    }
    else if(!userCheckBol(user,date,s)){
           printf("%s nesta data nao tem registo de boleia.\
n",userName(user,s));
    }
    else{
        delBoleia(user,date,s);
        printf("%s boleia retirada.\n", userName(user,s));
    }
}

void selecRemove(char * user, char * cmd, session s){
    char * date = (char*) malloc(sizeof(char)*MAXL);

    sscanf(cmd,"%*s %s",date);

    if(!checkstrdate(date)){
        printf("Data invalida.\n");
    }
    else if(!checkDeslocacao(user,date,s)){
         printf("%s nesta data nao tem registo de deslocacao.\
n",userName(user,s));
    }
    else if(existUsersReg(user,date,s)){
```

```
            printf("%s ja nao pode eliminar esta deslocacao.\
n",userName(user,s));
    }
    else{
        delDeslocacao(user,date,s);
        printf("Deslocacao removida.\n");
    }
}

//* LISTA OPTIONS

void selecLista(char * cmd, session s, char * user){
    char * option = (char *) malloc(MAXL);
    int selec = -1;

    sscanf(cmd,"%*s %s",option);

    //KEYWORDS
    char * cmds[] = {"minhas","boleias",};
    for(int i=0; i<(sizeof(cmds)/8);i++){
        if(!strcmp(option,cmds[i])){
            selec = i;
        }
    }
    //MAIL
    if(strchr(option,'@')≠NULL){
        selec = 2;
    }
    //DATE
    else if(strchr(option,'-')≠NULL){
        selec = 3;
    }
    listMenu(selec,s,user,option);
}

//* LISTA SELECTER

void listMenu(int selection, session s, char * master,char *
data){
    switch(selection){
        case 0:
            listaMinhas(s,master);
            break;
        case 1:
            listaBoleias(s,master);
            break;
        case 2:
            listaUser(s,data);
            break;
        case 3:
            listaDate(s,data,master);
            break;
        default:
            listaUser(s,data);
            break;
```

```
    }
}

//* LIST DISPLAY FUNCTION

void displayViagens(boleia bol){
    printf("%s\n",giveMaster(bol));
    printf("%s\n%s\n",giveOrigem(bol),giveDestino(bol));
                          printf("%s      %d:%d      %d      %d\
n",giveDate(bol),giveHorah(bol),giveHoram(bol),giveDuracao(bo
l),giveLugares(bol));
            printf("Lugares   vagos:   %d\n",(giveLugares(bol)-
givenumPenduras(bol)));
    if(givenumPenduras(bol)>0){
        printf("Boleias:");
        iterador penduras = seqPenduras(bol);
        while(temSeguinteIterador(penduras)){
            char * username = seguinteIterador(penduras);
            printf(" %s", username);
            if(temSeguinteIterador(penduras)){
                printf(";");
            }
        }
        printf("\n");
        destroiIterador(penduras);
    }
    else{
        printf("Sem boleias registadas.\n");
    }
    printf("\n");
}

//* LISTA FUNCTIONS

void listaMinhas(session s, char * user){
    iterador deslocacoes = listDeslocacoes(user,s);

    if(temSeguinteIterador(deslocacoes)==0){
                printf("%s  nao  tem  deslocacoes  registadas.\
n",userName(user,s));
    }
    while(temSeguinteIterador(deslocacoes)){
         displayViagens((boleia)seguinteIterador(deslocacoes))
;
    }
    destroiIterador(deslocacoes);
}

void listaUser(session s, char * user){
    iterador deslocacoes = listDeslocacoes(user,s);

    if(temSeguinteIterador(deslocacoes)==0){
         printf("Nao existem deslocacoes registadas para esse
utilizador.\n");
    }
```

```
    while(temSeguinteIterador(deslocacoes)){
        displayViagens((boleia)seguinteIterador(deslocacoes))
;
    }
    destroiIterador(deslocacoes);
}

void listaBoleias(session s, char * user){
    iterador deslocacoes = listBoleias(user,s);

    if(temSeguinteIterador(deslocacoes)==0){
                printf("%s nao tem boleias registadas.\
n",userName(user,s));
    }
    while(temSeguinteIterador(deslocacoes)){
        displayViagens((boleia)seguinteIterador(deslocacoes))
;
    }
    destroiIterador(deslocacoes);
}

void listaDate(session s, char * date, char * user){
    iterador deslocacoes = listDatas(date,s);

    if(temSeguinteIterador(deslocacoes)==0){
          printf("%s nao existem deslocacoes registadas para
%s.\n",userName(user,s), date);
    }
    while(temSeguinteIterador(deslocacoes)){
        displayViagens((boleia)seguinteIterador(deslocacoes))
;
    }
    destroiIterador(deslocacoes);
}

void selecSai(char * user, session s){
             printf("Fim de sessao. Obrigado %s.\n",
userName(user,s));
}

//* HELPER FUNCTIONS

//* PASSWORD CONDITIONS VERIFIER

int verifypass(char * pass){
    int valid = 0;
    int numeric = 0, letters = 0, others = 0;

    for(int i = 0; i < strlen(pass); i++){
        if(isalpha(pass[i])){
            letters++;
        }
        if(isdigit(pass[i])){
            numeric++;
        }
```

```
            if(ispunct(pass[i])){
                others++;
            }
        }

            if(letters>0  &&  numeric  >  0  &&  others  =  0  &&
strlen(pass)≥MINPASS && strlen(pass)≤MAXPASS){
            valid = 1;
        }

        return valid;
    }

    //* DATE CHECKERS

    int checkstrdate(char * date){
        int dia,mes,ano;

        sscanf(date,"%d-%d-%d",&dia,&mes,&ano);

        return (checkdate(dia,mes,ano));
    }

    int checkdate(int dia, int mes, int ano){
        int bisexto = 0;
        int valid = 1;

        if (ano ≥ 1800 && ano ≤ 9999){
            if ((ano % 4 = 0 && ano % 100 ≠ 0) || (ano % 400 =
0)){
                bisexto = 1;
            }
            if(mes ≥ 1 && mes ≤ 12){
                if (mes = 2){
                    if (bisexto && dia = 29){
                        valid = 1;
                    }
                    else if (dia > 28){
                        valid = 0;
                    }
                }
                 else if (mes = 4 || mes = 6 || mes = 9 || mes
= 11){
                    if (dia > 30){
                        valid = 0;
                    }
                }
                else if(dia > 31){
                    valid = 0;
                }
            }
            else{
                valid = 0;
            }
        }
```

```
    else{
        valid = 0;
    }

    if(ano≤0||mes≤0||dia≤0){
        valid = 0;
    }

    return valid;
}

//* TIME CHECKER

int checkhoranminuto(int hora,int minuto){
    int valid;

    if(hora≥0 && hora≤24){
        if(minuto≥0 && minuto<60){
            valid = 1;
        }
        else{
            valid = 0;
        }
    }
    else{
        valid = 0;
    }

    return valid;
}
//* LOWER INPUT FUNCTION
char* lowerIN(char * input, int size){
    char *out = malloc(size);
    int i;

    for(i = 0; i<size; i++){
        out[i] = tolower(input[i]);
    }

    out[i] = '\0';

    return out;
}
```

5.2 SESSION.H

```
#ifndef _H_SESSION
#define _H_SESSION

#include "boleia.h"
#include "user.h"

typedef struct _sess *session;

/**************************************
initialize - comeca uma sessao do programa
Parametros:
    - capUsers (capacidade maxima prevista de users)
Retorno:
    (struct) session
**************************************/
session initialize(int capUsers);

/**************************************
registUser - regista um user na base de dados
Parametros:
    - user (mail)
    - user (nome)
    - password
    - session
**************************************/
void registUser(char * usr, char * name, char * pass, session
s);

/**************************************
userLogin - faz login de um user
Parametros:
    - user (mail)
    - password
    - session
Retorno:
    1 - se o login foi feito com sucesso
    0 - se o contrario
**************************************/
int userLogin(char * username, char * pass, session s);

/**************************************
userCheck - verifica se o user existe
Parametros:
    - user (mail)
    - session
Retorno:
    1 - caso exista
    0 - caso contrario
**************************************/
int userCheck(char * mail, session s);

/**************************************
```

```
userName - devolve o nome do user
Parametros:
    - user (mail)
    - session
Retorno:
    (char *) nome do user
*************************************/
char * userName(char * mail, session s);

/*************************************
userMail - devolve o mail de um user
Parametros:
    - user (struct)
Retorno:
    (char *) mail do user
*************************************/
char * userMail(user us);

/*************************************
userCheckBol - verifica se um user esta registado numa viagem
Parametros:
    - user (mail)
    - data (da viagem)
    - session
Retorno:
    1 - se existe
    0 - caso contrario
*************************************/
int userCheckBol(char * mail, char * date, session s);

/*************************************
newDeslocacao - regista uma nova deslocacao para um user
Parametros:
    - user (mail)
    - session
    - local de origem
    - local de destino
    - datacmd (uma string com a data hora duracao e numero de
lugares)
*************************************/
void newDeslocacao(char * usr, session s, char * origem, char
* destino, char * datacmd);

/*************************************
delDeslocacao - elimina uma deslocacao
Parametros:
    - user (mail)
    - data (da viagem)
    - session
*************************************/
void delDeslocacao(char * usr, char * date, session s);

/*************************************
```

```
checkDeslocacao - verifica se existe uma deslocacao de um
user numa data
Parametros:
    - user (mail)
    - data
    - session
Retorno:
    1 - caso exista
    0 - caso contrario
*************************************/
int checkDeslocacao(char *usr, char * date, session s);

/*************************************
numEmptySeats - devolve o numero de lugares livres numa
viagem
Parametros:
    - user (mail)
    - data
    - session
Retorno:
    (int) numero de lugares livres
*************************************/
int numEmptySeats(char * mail, char * date, session s);

/*************************************
newPendura - adiciona um novo pendura a viagem
Parametros:
    - pendura (mail)
    - condutor (mail)
    - data
    - session
*************************************/
void newPendura(char * pendura, char * condutor, char * data,
session s);

/*************************************
existUsersReg - verifica se existem useres registados numa
viagem
Parametros:
    - user (mail)
    - data
    - session
Retorno:
    1 - caso exista
    0 - caso contrario
*************************************/
int existUsersReg(char *usr, char * date, session s);

/*************************************
delBoleia - retira um user da boleia
Parametros:
    - user (mail)
    - data
    - session
```

```
**************************************/
void delBoleia(char * usr, char * date, session s);

/**************************************
listDeslocacoes - devolve iterador para listar as deslocacoes
Parametros:
    - user (mail)
    - session
Retorno:
    (iterador) iterador de deslocacoes
**************************************/
iterador listDeslocacoes(char * usr, session s);

/**************************************
listBoleias - devolve iterador para listar as boleias em que
esta registado
Parametros:
    - user (mail)
    - session
Retorno:
    (iterador) iterador de boleias
**************************************/
iterador listBoleias(char * usr, session s);

/**************************************
listDatas - devolve iterador para listar as viagens de uma
data
Parametros:
    - data
    - session
Retorno:
    (iterador) iterador de viagens de uma data
**************************************/
iterador listDatas(char * date, session s);


/**************************************
listUsersReg - iterador para listar users registados numa
boleia
Parametros:
    - boleia
Retorno:
    (iterador) iterador de users registados
**************************************/
iterador listUsersReg(boleia bol);

#endif
```

5.3 SESSION.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "iterador.h"
#include "sequencia.h"
#include "dicionario.h"

#include "boleia.h"
#include "user.h"
#include "session.h"

#define MAXL 50
#define MAXC 10

#define MINPASS 4
#define MAXPASS 6

#define MAXUSERS 10000

struct _sess{
    //Users dicionario
    dicionario users;
    //Deslocacoes por data
    dicionario datalist;
};

//*INITIALIZER
session initialize(int capUsers){
    session s;
    s = (session) malloc(sizeof(struct _sess));
    s→users = criaDicionario(capUsers,1);
    if(s→users == NULL){
        return NULL;
    }
    s→datalist = criaDicionario(capUsers,1);
    if(s→datalist == NULL){
        return NULL;
    }
    return s;
}

//*USER RELATED ACTIONS
void registUser(char * usr, char * name, char * pass, session
s){
    user us = fillUser(usr,name,pass);
    adicionaElemDicionario(s→users,usr,us);
}

int userCheck(char * mail, session s){
    return existeElemDicionario(s→users,mail);
}
```

```
int userLogin(char * mail, char * pass, session s){
    user us;
    us = elementoDicionario(s→users,mail);
    return (checkpass(us,pass));
}

char * userName(char * mail, session s){
    user us;
    us = elementoDicionario(s→users,mail);
    return getName(us);
}

char * userMail(user us){
    return getMail(us);
}

int userCheckBol(char * mail, char * date, session s){
    int existe = 0;
    user us = elementoDicionario(s→users,mail);
    boleia bol = getBoleia(us,date);
    if(bol ≠ NULL){
        if(getPosUser(bol,mail)≠0){
            existe = 1;
        }
    }
    return existe;
}

//*DESLOCACOES RELATED ACTIONS

int checkDeslocacao(char *usr, char * date, session s){
    int exist = 0;
    user us = elementoDicionario(s→users,usr);
    boleia bol = getDeslocacao(us,date);
    if(bol ≠ NULL){
        exist = 1;
    }
    return exist;
}

void newDeslocacao(char * usr, session s, char * origem, char
* destino, char * datacmd){
    user us = elementoDicionario(s→users,usr);
                                        boleia        bol        =
fillBoleia(getMail(us),origem,destino,datacmd);
    addDeslocacao(us,bol);
    char * date = (char*) malloc(sizeof(char)* MAXL);
    sscanf(datacmd,"%s",date);
    //If Date doesnt exist in database
    if(!existeElemDicionario(s→datalist,date)){
                                          adicionaElemDicionario(s-
>datalist,date,criaDicionario(MAXUSERS,1));
        dicionario d = elementoDicionario(s→datalist,date);
```

```
            adicionaElemDicionario(d,usr,bol);
        }
        else{
            dicionario d = elementoDicionario(s→datalist,date);
            adicionaElemDicionario(d,usr,bol);
        }
    }

    void delDeslocacao(char * usr, char * date, session s){
        user us = elementoDicionario(s→users,usr);
        remDeslocacao(us,date);
        if(existeElemDicionario(s→datalist,date)){
            dicionario d = elementoDicionario(s→datalist,date);
            removeElemDicionario(d,usr);
        }
    }

    int numEmptySeats(char * mail, char * date, session s){
        user master = elementoDicionario(s→users, mail);
        boleia bol = getDeslocacao(master,date);
        return (giveLugares(bol) - givenumPenduras(bol));
    }

    void newPendura(char * pendura, char * condutor, char * date,
    session s){
        user condt = elementoDicionario(s→users,condutor);
        boleia bol = getDeslocacao(condt, date);
        user pend = elementoDicionario(s→users,pendura);
        addPendura(bol,(void*)pend);
        addBoleia(pend,bol);
    }

    int existUsersReg(char *usr, char * date, session s){
        int exist = 0;
        user us = elementoDicionario(s→users,usr);
        boleia bol = getDeslocacao(us,date);
        if(givenumPenduras(bol)>0){
            exist = 1;
        }
        return exist;
    }

    void delBoleia(char * usr, char * date, session s){
        user us = elementoDicionario(s→users,usr);
        remBoleia(us,date);
    }

    //* LIST FUNCTIONS

    iterador listDeslocacoes(char * usr, session s){
        user us = elementoDicionario(s→users,usr);
        return getDeslocacaoOrd(us);
    }
```

```
iterador listBoleias(char * usr, session s){
    user us = elementoDicionario(s→users,usr);
    return getBoleiasOrd(us);
}

iterador listDatas(char * date, session s){
    int size = 0;
                  boleia    *    vetor    =    (boleia    *)
malloc(sizeof(boleia)*MAXUSERS);
    if(existeElemDicionario(s→datalist,date)){
        dicionario d = elementoDicionario(s→datalist,date);
        iterador it = iteradorDicionario(d);
        size = tamanhoDicionario(d);
        vetor = getVecBoleia(it,size);
        insertionSort(vetor,size,strcmp,giveMaster);
    }

    return (criaIterador((void **)vetor,size));
}

iterador listUsersReg(boleia bol){
    return seqPenduras(bol);
}
```

5.4 USER.H

```
#ifndef _H_USER
#define _H_USER

typedef struct _usr *user;

/**************************************
fillUser - preenche uma estrutura de user
Parametros:
    - mail
    - nome
    - password
Retorno:
    (struct) user
**************************************/
user fillUser(char * mail, char * name, char * pass);

/**************************************
getDeslocacao - devolve a viagem relacionada a data
Parametros:
    - user
    - data
Retorno:
    (struct) boleia
**************************************/
boleia getDeslocacao(user us, char * date);

/**************************************
getBoleia - devolve a boleia relacionada a data
Parametros:
    - user
    - data
Retorno:
    (struct) boleia
**************************************/
boleia getBoleia(user us, char * date);

/**************************************
getVecBoleia - devolve um vetor com as viagens de um iterador
Parametros:
    - iterador
    - tamanho do vetor
Retorno:
    (boleia *) vetor de viagens
**************************************/
boleia * getVecBoleia (iterador it,int size);

/**************************************
addDeslocacao - adicona uma nova deslocacao ao user
Parametros:
    - user
    - boleia
**************************************/
void addDeslocacao(user us, boleia bol);
```

```
/**************************************
remDeslocacao - remove uma deslocacao do user
Parametros:
    - user
    - data
**************************************/
void remDeslocacao(user us, char * date);

/**************************************
addBoleia - adiciona uma boleia a lista de boleias a que o
user esta relacionado
Parametros:
    - user
    - boleia
**************************************/
void addBoleia(user master, boleia bol);

/**************************************
remBoleia - remove o user de uma boleia em que estava
registado
Parametros:
    - user
    - data
**************************************/
void remBoleia(user us, char * date);

/**************************************
getnDeslocacoes - devolve o numero de deslocacoes que o user
tem
Parametros:
    - user
Retorno:
    (int) numero de deslocacoes
**************************************/
int getnDeslocacoes(user us);

/**************************************
getDeslocacaoOrd - devolve um iterador com as deslocacoes do
user ordenadas por data
Parametros:
    - user
Retorno:
    (iterador) lista de deslocacoes
**************************************/
iterador getDeslocacaoOrd(user us);

/**************************************
getBoleiasOrd - devolve um iterador com as boleias em que o
user esta registado ordenadas por data
Parametros:
    - user
Retorno:
    (iterador) lista de viagens registadas
```

```
**************************************/
iterador getBoleiasOrd(user us);

/**************************************
binarySearch - faz a procura do local onde o elemento tem de
ser inserido
Parametros:
    - vetor de viagens
    - elemento a pesquisar
    - minimo da procura
    - maximo da procura
     - funcao de comparacao com outputs e inputs semelhantes
ao strcmp
    - funcao de recolha de dados com output char*
Retorno:
    (int) posicao a ser inserido
**************************************/
int binarySearch(boleia * vetor, boleia elem, int low, int
high, int compare(const char*, const char*), char*
giveFunc(boleia));

/**************************************
insertionSort - ordena viagens por data
Parametros:
    - vetor de viagens
    - tamanho do vetor
     - funcao de comparacao com outputs e inputs semelhantes
ao strcmp
    - funcao de recolha de dados com output char*
**************************************/
void insertionSort(boleia * vetor, int size, int
compare(const char*, const char*), char* giveFunc(boleia));

/**************************************
compareDate - compara duas datas em formato string
Parametros:
    - data1
    - data2
Retorno:
    -1 - data1<data2
    0 - data1=data2
    1 - data1>data2
**************************************/
int compareDate(const char * date1, const char * date2);

/**************************************
checkpass - verifica se a pass do user e igual a inserida
Parametros:
    - user
    - password a verficar
Retorno:
    1 - caso esteja correta
    0 - caso contrario
**************************************/
```

```
int checkpass(user us, char *pass);

/**************************************
getName - devolve o nome do user
Parametros:
    - user
Retorno:
    (char *) nome do user
**************************************/
char * getName(user us);

/**************************************
getMail - devolve o mail do user
Parametros:
    - user
Retorno:
    (char *) mail do user
**************************************/
char * getMail(user us);


#endif
```

5.5 USER.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "iterador.h"
#include "sequencia.h"
#include "dicionario.h"

#include "boleia.h"
#include "user.h"

#define MAXMAIL 20
#define PASS 6

//SECRET ENCRYPTION KEY
const char * key = "aed18!";

//ENCRYPTION FUNCTIONS
void xorstring(char * keyword, int max , char * out);
void encryption(user us,char * pass);

struct _usr{
    char mail[MAXMAIL];
    char * nome;
    char hashedPass[PASS];
    //Dictionary of deslocacoes
    dicionario dicboleias;
    int numDeslocacoes;
    //Dictionary of boleias
    dicionario boleiasregistadas;
    int numBoleias;
};

void xorstring(char * keyword, int max , char * out){
    for(int i = 0; i<max;i++){
        out[i]=key[i]^keyword[i];
    }
}

void encryption(user us,char * pass){
    xorstring(pass,PASS,us->hashedPass);
}

//* USER FORM

user fillUser(char * mail, char * name, char * pass){
    user us = (user) malloc(sizeof(struct _usr));
    us->nome = (char * ) malloc(strlen(name));
    strcpy(us->mail, mail);
    strcpy(us->nome, name);
    us->dicboleias = criaDicionario(7,1);
    us->boleiasregistadas = criaDicionario(7,1);
```

```
    encryption(us,pass);
    us→numDeslocacoes = 0;
    us→numBoleias = 0;
    return us;
}

//* BOLEIA GETTERS

boleia getDeslocacao(user us, char * date){
    if(existeElemDicionario(us→dicboleias,date)){
                        return   (boleia)elementoDicionario(us-
>dicboleias,date);
    }
    return NULL;
}

boleia getBoleia(user us, char * date){
    if(existeElemDicionario(us→boleiasregistadas,date)){
                        return   (boleia)elementoDicionario(us-
>boleiasregistadas,date);
    }
    return NULL;
}

boleia * getVecBoleia (iterador it,int size){
    boleia * vetor = (boleia *) malloc(sizeof(boleia)*size);
    int id=0;
    while(temSeguinteIterador(it)){
        vetor[id]=seguinteIterador(it);
        id++;
    }
    destroiIterador(it);
    return vetor;
}

//* DESLOCACOES RELATED FUNCTIONS

void addDeslocacao(user us, boleia bol){
    adicionaElemDicionario(us→dicboleias,giveDate(bol),bol);
    us→numDeslocacoes++;
}

void remDeslocacao(user us, char * date){
    removeElemDicionario(us→dicboleias,date);
    us→numDeslocacoes--;
}

void addBoleia(user us, boleia bol){
                                    adicionaElemDicionario(us-
>boleiasregistadas,giveDate(bol),bol);
    us→numBoleias++;
}

void remBoleia(user us, char * date){
```

```
                boleia    bol    =    removeElemDicionario(us-
>boleiasregistadas,date);
    us→numBoleias--;
    int pos = getPosUser(bol,getMail(us));
    remPendura(bol,pos);
}

int getnDeslocacoes(user us){
    return us→numDeslocacoes;
}

//* BOLEIAS SORT FUNCTIONS

iterador getDeslocacaoOrd(user us){
    iterador it = iteradorDicionario(us→dicboleias);
    int size = us→numDeslocacoes;
    boleia * vetor = getVecBoleia(it,size);
    insertionSort(vetor,size,compareDate,giveDate);
    return (criaIterador((void **)vetor,size));
}

iterador getBoleiasOrd(user us){
    iterador it = iteradorDicionario(us→boleiasregistadas);
    int size = us→numBoleias;
    boleia * vetor = getVecBoleia(it,size);
    insertionSort(vetor,size,compareDate,giveDate);
    return (criaIterador((void **)vetor,size));
}

//* SORT ALGORITHM

int binarySearch(boleia * vetor, boleia elem, int low, int
high,   int   compare(const   char*,   const   char*),   char*
giveFunc(boleia)){
    char * elemCh = giveFunc(elem);

    if(high ≤ low){
        if(compare(elemCh,giveFunc(vetor[low]))>0){
            return low+1;
        }
        else{
            return low;
        }
    }

    int mid = (low + high)/2;

    if(compare(elemCh,giveFunc(vetor[mid]))=0){
        return mid+1;
    }

    if(compare(elemCh,giveFunc(vetor[mid]))>0){
              return  binarySearch(vetor,  elem,  mid+1,  high,
compare, giveFunc);
```

```
    }

      return binarySearch(vetor, elem, low, mid-1, compare,
giveFunc);
}

void   insertionSort(boleia   *   vetor,   int   size,   int
compare(const char*, const char*), char* giveFunc(boleia)){
    boleia selected;
    int i, loc, j;
    for (i = 1; i < size; i++) {
        j = i - 1;
        selected = vetor[i];

        // find location where selected sould be inseretd
                  loc  =  binarySearch(vetor,  selected,  0,
j,compare,giveFunc);

        // Move all elements after location to create space
        while (j ≥ loc)
        {
            vetor[j+1] = vetor[j];
            j--;
        }
        vetor[j+1] = selected;
    }
}

//* DATE COMPARER

int compareDate(const char * date1, const char * date2){
    int result;
    int dia1,mes1,ano1;
    int dia2,mes2,ano2;
    sscanf(date1,"%d-%d-%d",&dia1,&mes1,&ano1);
    sscanf(date2,"%d-%d-%d",&dia2,&mes2,&ano2);
    if(ano1<ano2){
        result = -1;
    }
    else if(ano1=ano2){
        if(mes1<mes2){
            result = -1;
        }
        else if(mes1=mes2){
            if(dia1<dia2){
                result = -1;
            }
            else if(dia1 = dia2){
                result = 0;
            }
            else{
                result = 1;
            }
        }
```

```
            else{
                result = 1;
            }
        }
        else{
            result = 1;
        }
        return result;
}

//* PASSWORD CHECKER FUNCTION

int checkpass(user us, char *pass){
        pass[strlen(pass)-1] = '\0';
        char * test = (char *) malloc(PASS);
        xorstring(us->hashedPass,PASS,test);
        if(strncmp(test,pass,7)==0){
            return 1;
        }
        else{
            return 0;
        }
}

//* USER INFO GETTERS

char * getName(user us){
        return us->nome;
}
char * getMail(user us){
        return us->mail;
}
```

5.6 BOLEIA.H

```
#ifndef _H_BOLEIA
#define _H_BOLEIA

typedef struct _bol *boleia;

/**************************************
fillBoleia - preenche uma boleia e devolve
Parametros:
    - mail
    - origem
    - destino
     - data (toda a informacao relacionada com a data, hora,
duracao e lugares)
Retorno:
    (struct) boleia
**************************************/
boleia fillBoleia(char * mail, char * origem, char *
destino,char * data);

/**************************************
addPendura - adiciona um pendura a viagem
Parametros:
    - boleia
        - (user)pendura (usou se o void * para evitar
recursividade)
**************************************/
void addPendura(boleia bol, void * pendura);

/**************************************
remPendura - remove um pendura da viagem
Parametros:
    - boleia
    - pos (posicao do user a retirar)
**************************************/
void remPendura(boleia bol, int pos);

/**************************************
getPosUser - devolve a posicao de um user numa boleia
Parametros:
    - boleia
    - email do user
Retorno:
    (int) posicao do user na boleia
**************************************/
int getPosUser(boleia bol, char * us);

/**************************************
seqPenduras - devolve um iterador da sequencia de penduras
Parametros:
    - boleia
Retorno:
    (iterador) iterador da Sequencia
**************************************/
```

```
iterador seqPenduras(boleia bol);

/**************************************
giveMaster - devolve o dono da viagem
Parametros:
    - boleia
Retorno:
    (char *) email do dono da viagem
**************************************/
char * giveMaster(boleia bol);

/**************************************
giveOrigem - devolve o local de comeco da viagem
Parametros:
    - boleia
Retorno:
    (char *) origem da viagem
**************************************/
char * giveOrigem(boleia bol);

/**************************************
giveDestino - devolve o local de destino da viagem
Parametros:
    - boleia
Retorno:
    (char *) destino da viagem
**************************************/
char * giveDestino(boleia bol);

/**************************************
giveDate - devolve a data da viagem
Parametros:
    - boleia
Retorno:
    (char *) data da viagem
**************************************/
char * giveDate(boleia bol);

/**************************************
giveHorah - devolve a hora da viagem
Parametros:
    - boleia
Retorno:
    (int) hora da viagem
**************************************/
int giveHorah(boleia bol);

/**************************************
giveHoram - devolve os minutos da hora de viagem
Parametros:
    - boleia
Retorno:
    (int) minutos da hora viagem
**************************************/
```

```
int giveHoram(boleia bol);

/**************************************
giveDuracao - devolve a duracao da viagem
Parametros:
    - boleia
Retorno:
    (int) duracao da viagem
**************************************/
int giveDuracao(boleia bol);

/**************************************
giveLugares - devolve o numero de lugares da viagem (no
momento inicial)
Parametros:
    - boleia
Retorno:
    (int) numero de lugares
**************************************/
int giveLugares(boleia bol);

/**************************************
givenumPenduras - devolve o numero de penduras inscritos
Parametros:
    - boleia
Retorno:
    (int) numero de penduras inscritos
**************************************/
int givenumPenduras(boleia bol);

#endif
```

5.7 BOLEIA.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "iterador.h"
#include "sequencia.h"
#include "dicionario.h"

#include "boleia.h"
#include "user.h"

struct _bol{
    char * mail;
    char * origem;
    char * destino;
    char data[11];
    int horaH;
    int horaM;
    int duracao;
    int lugaresLivres;
    //sequencia de users
    sequencia penduras;
    int numPenduras;
};

//* DESLOCACAO FORM

boleia fillBoleia(char * mail, char * origem, char *
destino,char * data){
    boleia bol = (boleia) malloc(sizeof(struct _bol));
    bol->origem = (char *) malloc(strlen(origem));
    bol->destino = (char *) malloc(strlen(destino));
    bol->mail = mail;
    strcpy(bol->origem, origem);
    strcpy(bol->destino,destino);
     sscanf(data,"%s %d:%d %d %d",bol->data,&bol->horaH,&bol-
>horaM,&bol->duracao,&bol->lugaresLivres);
    bol->penduras = criaSequencia(bol->lugaresLivres);
    bol->numPenduras = 0;
    return bol;
}

//* PENDURA FUNCTIONS

void addPendura(boleia bol, void * pendura){
    bol->numPenduras++;
                adicionaPosSequencia(bol->penduras,pendura,bol-
>numPenduras);

}

void remPendura(boleia bol, int pos){
```

```
    removePosSequencia(bol→penduras,pos);
    bol→numPenduras--;
}

int getPosUser(boleia bol, char * us){
    iterador it = iteradorSequencia(bol→penduras);
    int pos = 0;
    int id = 0;
    while(temSeguinteIterador(it) && pos = 0){
        id++;
        char * m = getMail((user)seguinteIterador(it));
        if(strcmp(m,us) = 0){
            pos = id;
        }
    }
    destroiIterador(it);
    return pos;
}

iterador seqPenduras(boleia bol){
    return iteradorSequencia(bol→penduras);
}

//* GIVE FUNCTIONS

char * giveMaster(boleia bol){
    return bol→mail;
}
char * giveOrigem(boleia bol){
    return bol→origem;
}
char * giveDestino(boleia bol){
    return bol→destino;
}
char * giveDate(boleia bol){
    return bol→data;
}
int giveHorah(boleia bol){
    return bol→horaH;
}
int giveHoram(boleia bol){
    return bol→horaM;
}
int giveDuracao(boleia bol){
    return bol→duracao;
}
int giveLugares(boleia bol){
    return bol→lugaresLivres;
}
int givenumPenduras(boleia bol){
    return bol→numPenduras;
}
```