FCT Boleia

Algoritmo e Estruturas de Dados
Enunciado do Trabalho Prático, versão 1.0 – 2019-4-5

Notas importantes

Trabalho em Grupo de 2 alunos a desenvolver respeitando os princípios do *Código de Ética* do departamento de informática.

Trabalho com 2 Fases de entrega. A 1ª fase consta de um programa desenvolvido com os TADs colecção implementados com vectores, e que se encontram na página da cadeira (25% da nota final na cadeira). A 2ª fase consta dum relatório escrito e um programa, o qual é desenvolvida com os TADs coleção implementados com as estruturas de dados adequadas, e implementadas pelos os alunos (15% da nota final na cadeira).

Prazos de entrega das 2 fases. A 1º fase deve ser entregue até às 17:00 horas do dia 20 de Maio de 2019, ou após esta data com uma penalização de **3 valores na nota da componente prática** da cadeira. A 2º fase deve ser entregue até às 17:00 horas do dia 3 de Junho de 2019.

Frequência na cadeira de AED é dada pela 1ª fase do trabalho. Para obter frequência a 1ª fase deve ter uma nota superior ou igual a 9.5 valores em 20.

Submissão e aceitação do código fonte pelo sistema de avaliação automática Mooshak. O relatório escrito na secretaria do departamento de Informática.

1 Introdução

Este documento descreve o trabalho prático da disciplina de Algoritmos e Estruturas de Dados do 1º ano do Mestrado Integrado em Engenharia Electrotécnica e de Computadores. Os alunos podem e devem tirar todas as dúvidas com os docentes da disciplina.

2 Desenvolvimento da Aplicação *FCT Boleia*

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

2.1 Descrição do Problema

Dar e pedir boleia é uma coisa que já provavelmente fez com alguns dos seus amigos: é simplesmente o facto de deslocar-se com pessoas partilhando um carro e os custos da deslocação (combustível e portagens).

Pretende-se criar uma aplicação para facilitar o uso desta forma de viajar.

A aplicação deve permitir que as pessoas encontrem e/ou partilhem um carro para se deslocarem juntos, "à boleia". Esta aplicação tem como funcionalidades principais:

- registar uma dada deslocação, que poderá ser partilhada;
- seleccionar uma boleia para uma dada deslocação;
- consultar as possíveis deslocações existentes.

Para poder usar estas funcionalidades, o utilizador deve registar-se na aplicação com o seu email. Assim sendo, o programa pode encontrar-se em dois modos:

- inicial: aqui é possível fazer registos de novos utilizadores, iniciar uma sessão com um utilizador e terminar a execução do programa;
- sessão: aqui é possível registar, partilhar e consultar viagens.

Prevê-se que existam mais de 10000 utilizadores registados, e que cada utilizador tenha quase sempre uma deslocação diária registada. As funcionalidades mais utilizadas são: a listagem das deslocações dado um dia específico, e o registo de boleias.

A interação do utilizador com o programa é feita sempre através de comandos, descritos na secção seguinte.

3 Comandos

Nesta secção apresentam-se os vários comandos que o sistema deve ser capaz de interpretar e executar. Nos exemplos apresentados, diferenciamos o `texto escrito pelo utilizador` da retroacção escrita pelo programa na consola, ao executar o comando. Pode assumir que o utilizador não cometerá erros na introdução de argumentos nos comandos, para além dos descritos neste enunciado, ou seja, apenas tem de tratar as situações de erro descritas aqui, pela ordem que são descritas.

Na leitura de comandos o interpretador não deve fazer distinção entre maiúsculas e minúsculas. Por exemplo, para o comando `sai`, o interpretador deve aceitar como válidas as alternativas `SAI`, `sai`, `Sai`, `sAI`, etc. Nos vários exemplos que se seguem, o símbolo ↵ denota a mudança de linha.

Caso o utilizador introduza um comando inexistente, o programa escreve na consola a mensagem Comando inexistente.. Por exemplo o comando inexistente `desconhecido` teria este efeito:

```
> desconhecido↵
Comando inexistente.↵
>
```

Este erro "Comando inexistente."é o primeiro erro a verificar em todos os casos.

Vários comandos têm argumentos. Pode assumir que o utilizador apenas usa argumentos em número e de tipos correctos. No entanto, pode acontecer que os argumentos passados a um desses comandos tenham algum valor inválido no contexto do problema. Por esse motivo, teremos de testar esses argumentos exactamente pela ordem especificada neste enunciado.

3.1 Comando `ajuda`

Informa sobre os comandos disponíveis no programa. O comando não necessita de argumentos e tem sempre sucesso. Este comando tem 2 formatos distintos, dependendo do modo em que o programa está. No caso de ainda não ter-se realizado um login (modo inicial), o formato do comando é o seguinte:

```
> ajuda↵
ajuda - Mostra os comandos existentes↵
termina - Termina a execucao do programa↵
regista - Regista um novo utilizador no programa↵
entrada - Permite a entrada ("login") dum utilizador no programa↵
>
```

No caso do programa estar em modo sessão (já foi feito o comando "entrada"), o formato do comando é o seguinte:

```
emailUtilizador > ajuda↵
ajuda - Mostra os comandos existentes↵
sai - Termina a sessao deste utilizador no programa↵
nova - Regista uma nova deslocacao↵
lista - Lista todas ou algumas deslocacoes registadas↵
boleia - Regista uma boleia para uma dada deslocacao↵
retira - Retira uma dada boleia↵
remove - Elimina uma dada deslocacao↵
emailUtilizador >
```

Note que, quando o programa está em modo sessão no prompt está o email do utilizador que fez login (comando entrada). Caso seja o utilizador com email *aa@fct.unl.pt* a executar o comando ajuda, o formato seria:

```
aa@fct.unl.pt > ajuda↵
ajuda - Mostra os comandos existentes↵
sai - Termina a sessao do utilizador no programa↵
nova - Regista uma nova deslocacao↵
lista - Lista todas ou algumas deslocacoes registadas↵
boleia - Regista uma boleia para uma dada deslocacao↵
consulta - Lista a informacao de uma dada deslocacao↵
remove - Retira uma dada deslocacao↵
aa@fct.unl.pt >
```

3.2 Comando `termina`

Termina a execução do programa. O comando não necessita de argumentos. Este comando só tem sucesso se o programa estiver em modo inicial.

No caso do programa estar em modo inicial, o formato do comando é o seguinte:

```
> termina↵
Obrigado. Ate a proxima.↵
>
```

Um exemplo da execução do comando `termina`, quando o programa está em modo sessão com o utilizador aa@fct.unl.pt.

```
aa@fct.unl.pt > termina↵
Comando inexistente.↵
aa@fct.unl.pt >
```

3.3 Comando **regista**

Regista um novo utilizador no programa. Este comando só tem sucesso se o programa estiver em modo inicial, e não exisitir um utilizador com o email dado. Para realizar um registo deve ser dado sempre o email. Este email tem que ser único, e nesse caso é pedido o nome e a password do utilizador.

No caso do programa estar em modo inicial e o email ser único, o formato do comando é o seguinte (parâmetros com estilo **bold**):

```
> regista email utilizador↵
nome (maximo 50 caracteres): nome completo utilizador↵
password (entre 4 e 6 caracteres - digitos e letras): password utilizador↵
Registo efetuado.↵
>
```

Assuma que o nome não ultrapassa os 50 caracteres e é sempre válido. No entanto, a password deve ser verificada. A password para ser válida deve ter uma dimensão entre 4 e 6 caracteres, e estes devem ser letras(maiúsculas ou minúsculas) e digitos. Caso não seja válida, o programa deve notificar com a mensagem "Password incorrecta.", e voltar a pedir a mesma um máximo de 3 vezes. Por exemplo:

```
> regista aa@fct.unl.pt↵
nome (maximo 50 caracteres): Fernanda AA↵
password (entre 4 e 6 caracteres - digitos e letras): aed↵
Password incorrecta.↵
password (entre 4 e 6 caracteres - digitos e letras): aed_2↵
Password incorrecta.↵
password (entre 4 e 6 caracteres - digitos e letras): aed18↵
Registo efetuado.↵
>
```

Caso não seja dada uma password válida nas 3 tentativas o registo não será efetuado.

```
> regista aa@fct.unl.pt↵
nome (maximo 50 caracteres): Fernanda AA↵
password (entre 4 e 6 caracteres - digitos e letras): aed↵
Password incorrecta.↵
password (entre 4 e 6 caracteres - digitos e letras): aed2018↵
Password incorrecta.↵
password (entre 4 e 6 caracteres - digitos e letras): aeda↵
Password incorrecta.↵
Registo nao efetuado.↵
>
```

Um exemplo da execução do comando `regista`, em que o email do utilizador **aa@fct.unl.pt** já existe. Neste caso, escreve-se na consola "Utilizador ja existente.".

```
> regista aa@fct.unl.pt↵
Utilizador ja existente.↵
Registo nao efetuado.↵
>
```

Um exemplo da execução do comando `regista`, quando o programa está em modo sessão com o utilizador **aa@fct.unl.pt**.

```
aa@fct.unl.pt > regista email utilizador↵
Comando inexistente.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"; (2) o utilizador já existe e (3) não ter dado uma password válida nas 3 tentativas.

3.4 Comando `entrada`

Permite a entrada ("login") dum utilizador no programa. Este comando só tem sucesso se o programa estiver em modo inicial, e existir um utilizador com o email dado. Para realizar a entrada na aplicação deve ser dado sempre o email. Este email tem que ser de um utilizador já registado. Neste caso é pedido a password, tendo o utilizador 3 tentativas para a editar correctamente.

No caso do programa estar em modo inicial e o email ser dum utilizador registado, o formato do comando é o seguinte:

```
> entrada email utilizador↵
password: password utilizador↵
emailUtilizador >
```

Note que, quando o comando `entrada` tem sucesso, o prompt seguinte tem o email do utilizador que fez login. Caso a password não seja correcta, o programa deve notificar com a mensagem "Password incorrecta.", e voltar a pedir (no máximo mais 2 vezes) até que seja dado um valor correcto. Por exemplo:

```
> entrada aa@fct.unl.pt↵
password: aed↵
Password incorrecta.↵
password: aeda↵
Password incorrecta.↵
password: aed18↵
aa@fct.unl.pt >
```

Caso não seja dada a password correcta, a entrada não é efetuada.

```
> entrada aa@fct.unl.pt↵
password: aed22↵
Password incorrecta.↵
password: aed2018↵
Password incorrecta.↵
password: aed1↵
Password incorrecta.↵
>
```

Um exemplo da execução do comando `entrada`, em que o email do utilizador **aa@fct.unl.pt** não existe. Neste caso, escreve-se na consola "Utilizador nao existente.".

```
> entrada aa@fct.unl.pt↵
Utilizador nao existente.↵
>
```

Um exemplo da execução do comando `entrada`, quando o programa está em modo sessão com o utilizador **aa@fct.unl.pt**.

```
aa@fct.unl.pt > entrada email utilizador↵
Comando inexistente.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) estar em modo "sessão"; (2) o utilizador não existe e (3) não ter dado a password válida nas 3 tentativas.

3.5 Comando `sai`

Termina a sessao do utilizador no programa. O comando não necessita de argumentos. Este comando só tem sucesso se o programa estiver em modo sessão.

Um exemplo da execução do comando `sai`, quando o programa está em modo sessão com o utilizador **aa@fct.unl.pt**.

```
aa@fct.unl.pt > sai↵
Fim de sessao. Obrigado Fernanda AA.↵
>
```

Um exemplo da execução do comando `sai`, quando o programa está em modo inicial.

```
> sai↵
Comando inexistente.↵
>
```

3.6 Comando `nova`

Regista uma nova deslocacao. Pode-se registar uma `nova` deslocação sempre e quando o programa esteja em modo sessão. Esta deslocação é registada pelo utilizador da sessão..

Quando se regista a deslocação deve ser dada a seguinte informação: locais de origem e destino, data e horário da viagem (dia, mês, ano, hora e minutos estimados de partida e duração em minutos estimada de viagem) e número de lugares do carro disponíveis para boleia. Note que a hora estimada está no formato hh:mm (hh é um valor inteiro entre 0 e 23 e mm é um valor inteiro entre 0 e 59), e a duração é um valor inteiro positivo (maior que zero). Este registo tem sucesso se não existir uma deslocação deste utilizador no mesmo dia, isto é no sistema o utilizador só pode ter uma deslocação por dia.

Se o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) estiver a registar uma deslocação, o formato geral do comando é o seguinte:

```
aa@fct.unl.pt > nova↵
local origem↵
local destino↵
dia-mês-ano hora:minutos duração número lugares para boleia↵
Deslocacao registada. Obrigado Fernanda AA.↵
aa@fct.unl.pt >
```

Exemplo de um registo de deslocação poderia ser:

```
aa@fct.unl.pt > nova↵
Lisboa↵
Alvalade do Sado↵
20-12-2018 15:00 90 5↵
Deslocacao registada. Obrigado Fernanda AA.↵
fb@fct.unl.pt >
```

Por exemplo, suponha agora que o mesmo utilizador pretende registar uma deslocação entre Porto e Monte da Caparica no dia 20-12-2018 pelas 9:30 horas com uma duração estimada de 180 minutos, e 5 lugares disponíveis. Neste caso, o registo não é efetuado pois já existe uma deslocação nessa data para este utilizador.

```
aa@fct.unl.pt > nova↵
Porto↵
Monte da Caparica↵
20-12-2018 9:30 180 5↵
Fernanda AA ja tem uma deslocacao registada nesta data.↵
Deslocacao nao registada.↵
aa@fct.unl.pt >
```

Caso exista erros nos dados fornecidos pelo utilizador, o registo não é efetuado e escreve a mensagem de erro "Dados invalidos". Os erros possíveis são: data invalida, horario invalido, duração e lugares disponíveis com valores não positivos. Um exemplo desta situação é:

```
aa@fct.unl.pt > nova↵
Porto↵
Monte da Caparica↵
20-13-2018 15:78 180 5↵
Dados invalidos.↵
Deslocacao nao registada.↵
aa@fct.unl.pt >
```

Um exemplo da execução do comando `nova`, quando o programa está em modo inicial.

```
> nova↵
Comando inexistente.↵
>
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"; (2) ter dados invalidos e (3) ter uma deslocação na data indicada.

3.7 Comando `lista`

Lista todas ou algumas deslocacoes registadas. Este comando só terá sucesso se o programa estiver em modo sessão. Existem três modos de listagem:

- Caso seja escrita a palavra "minhas", deve-se listar a informação referente às deslocações do utilizador da sessão;
- Caso seja escrita a palavra "boleias", deve-se listar a informação referente às boleias registadas do utilizador da sessão;

- Caso seja dado um email, deve-se listar a informação referente às deslocações do utilizador com o email dado;
- Caso seja dado uma data, deve-se listar a informação referente às deslocações de todos os utilizadores na data indicada;

AS listagens das deslocações estão ordenadas por data, e em caso de empate na data pela ordem alfabética crescente do email do utilizador que as registou. As listagens de emails (boleias numa dada deslocação) estão ordenadas por ordem de registo da boleia.

Exemplos da execução do comando `lista`, quando o programa está em modo inicial.

```
> lista minhas↵
Comando inexistente.↵
>
> lista boleias↵
Comando inexistente.↵
>
> lista aa@fct.unl.pt↵
Comando inexistente.↵
> lista 13-4-2017↵
Comando inexistente.↵
>
```

Assuma que o argumento do comando `lista`, é sempre: (1) uma das palavras: "minhas"ou "boleias"; ou (2) uma data; ou (3) um email.

3.7.1 Listar as minhas deslocações registadas

O utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista` com o parâmetro "minhas". Caso não tenha registado nenhuma deslocação:

```
aa@fct.unl.pt > lista minhas↵
Fernanda AA nao tem deslocacoes registadas.↵
aa@fct.unl.pt >
```

Caso tenha registado 2 deslocações:

```
aa@fct.unl.pt > lista minhas↵
aa@fct.unl.pt↵
Lisboa↵
Alvalade do Sado↵
20-12-2018 15:30 90 5↵
Lugares vagos: 5↵
Sem boleias registadas.↵
↵
aa@fct.unl.pt↵
Porto↵
Monte da Caparica↵
21-12-2018 10:45 120 5↵
Lugares vagos: 3↵
Boleias: bb@fct.unl.pt; cc@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista todos` é: (1) não estar em modo "sessão".

3.7.2 Listar as minhas boleias registadas

O utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista` com o parâmetro "boleias". Caso não tenha registado nenhuma boleia:

```
aa@fct.unl.pt > lista boleias↵
Fernanda AA nao tem boleias registadas.↵
aa@fct.unl.pt >
```

Caso tenha registado 2 boleias:

```
aa@fct.unl.pt > lista boleias↵
bb@fct.unl.pt↵
Lisboa↵
Beja↵
12-12-2018 15:30 90 5↵
Lugares vagos: 4↵
Boleias: aa@fct.unl.pt↵
↵
dd@fct.unl.pt↵
Porto↵
Lisboa↵
15-12-2018 10:45 120 5↵
Lugares vagos: 2↵
Boleias: bb@fct.unl.pt; cc@fct.unl.pt; aa@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista boleias` é: (1) não estar em modo "sessão".

3.7.3 Listar as deslocações registadas por um dado email

O utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista` com um parâmetro que é o email dum dado utilizador. Caso o utilizador indicado *bb@fct.unl.pt* (utilizador com nome *Fernanda BB*, dado no registo de utilizador) não tenha registado nenhuma deslocação:

```
aa@fct.unl.pt > lista bb@fct.unl.pt↵
Nao existem deslocacoes registadas para esse utilizador.↵
aa@fct.unl.pt >
```

Caso tenha registado uma deslocação:

```
aa@fct.unl.pt > lista bb@fct.unl.pt↵
bb@fct.unl.pt↵
Lisboa↵
Alvalade do Sado↵
20-12-2018 15:30 90 5↵
Lugares vagos: 5↵
Sem boleias registadas.↵
↵
aa@fct.unl.pt >
```

Caso o utilizador indicado *cccc@fct.unl.pt* não exista:

```
aa@fct.unl.pt > lista cccc@fct.unl.pt↵
Nao existe o utilizador dado.↵
aa@fct.unl.pt >
```

Caso o utilizador indicado seja o último utilizador que entrou no programa com o comando `entrada`, o comando é equivalente ao comando `lista todos`.

A ordem de verificação de erros neste comando `lista` com o argumento dum email é: (1) não estar em modo "sessão"; (2) o email dado não ser dum utilizador registado.

3.7.4 Listar as deslocações numa dada data

Este comando é um dos mais usados, pelo que a sua implementação deve ser cuidada de modo a ser eficiente.
O utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista`, dando uma data. Caso não existam deslocações para a data indicada:

```
aa@fct.unl.pt > lista 13-4-2018↵
Fernanda AA nao existem deslocacoes registadas para 13-4-2018.↵
aa@fct.unl.pt >
```

Caso exista 4 deslocações registadas nessa data. Os utilizadores que registaram tem os seguintes emails *aa@fct.unl.pt*, *bb@fct.unl.pt*, *cc@fct.unl.pt* e *dd@fct.unl.pt*:

```
aa@fct.unl.pt > lista 20-12-2018↵
aa@fct.unl.pt↵
Lisboa↵
Costa da Caparica↵
20-12-2018 15:50 25 5↵
Lugares vagos: 5↵
Sem boleias registadas.↵
↵
bb@fct.unl.pt↵
Porto↵
Monte da Caparica↵
20-12-2018 10:45 120 5↵
Lugares vagos: 3↵
Boleias: cc@fct.unl.pt; aa@fct.unl.pt↵
↵
cc@fct.unl.pt↵
Porto↵
Monte da Caparica↵
20-12-2018 10:15 120 5↵
Lugares vagos: 4↵
Boleias: bb@fct.unl.pt↵
↵
dd@fct.unl.pt↵
Braga↵
Monte da Caparica↵
20-12-2018 10:05 140 5↵
Lugares vagos: 2↵
Boleias: ee@fct.unl.pt; kk@fct.unl.pt; gg@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > lista 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista` com um argumento que é uma data, é: (1) não estar em modo "sessão"; (2) ter uma data invalida.

3.8 Comando `boleia`

Regista uma boleia para uma dada deslocacao. Regista mais um lugar ocupado para uma `dada` deslocação sempre e quando o programa esteja em modo sessão, e essa deslocação exista, tenha lugares vagos, não tenha sido registada pelo último utilizador a executar o comando `entrada`, e não haja já uma boleia registada nesse dia para o último utilizador a executar o comando `entrada`.
Este comando é um dos mais usados, pelo que a sua implementação deve ser cuidada de modo a ser eficiente.
Se o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executar o comando `boleia`, indicando o email do utilizador que registou a deslocação, e a data da deslocação, é registada a boleia, caso exista lugares disponíveis para a

deslocação e esse utilizador *aa@fct.unl.pt* não tenha uma boleia ou deslocação nesse dia.
Vejamos um exemplo desta situação (assuma que existe uma deslocação do utilizador bb@fct.unl.pt no dia 20-12-2018 com 2 lugares vagos):

```
aa@fct.unl.pt > lista bb@fct.unl.pt↵
bb@fct.unl.pt↵
Braga↵
Monte da Caparica↵
20-12-2018 10:00 90 5↵
Lugares vagos: 2↵
Boleias: cc@fct.unl.pt; hh@fct.unl.pt; gg@fct.unl.pt↵
...↵
aa@fct.unl.pt >
```

Agora o utilizador *aa@fct.unl.pt* trata de registar a boleia:

```
aa@fct.unl.pt >boleia bb@fct.unl.pt 20-12-2018↵
Boleia registada.↵
aa@fct.unl.pt >
```

Agora se listarmos as deslocações desse utilizador, podemos ver que a boleia foi registada, ficando de último nos registos da boleia:

```
aa@fct.unl.pt >lista bb@fct.unl.pt ↵
bb@fct.unl.pt↵
Braga↵
Monte da Caparica↵
20-12-2018 10:00 90 5↵
Lugares vagos: 1↵
Boleias: cc@fct.unl.pt; hh@fct.unl.pt; gg@fct.unl.pt; aa@fct.unl.pt↵
...↵
aa@fct.unl.pt >
```

Caso exista a deslocação mas não tenha lugares vagos, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia bb@fct.unl.pt 20-12-2018↵
Fernanda AA nao existe lugar. Boleia nao registada.↵
aa@fct.unl.pt >
```

Caso exista a deslocação mas tenha sido registada pelo próprio utilizador, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia aa@fct.unl.pt 20-12-2018↵
Fernanda AA nao pode dar boleia a si proprio. Boleia nao registada.↵
aa@fct.unl.pt >
```

Caso exista a deslocação mas o utilizador já registou antes uma boleia ou uma deslocação nessa data, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia bb@fct.unl.pt 20-12-2018↵
Fernanda AA ja registou uma boleia nesta data. Boleia nao registada.↵
aa@fct.unl.pt >
```

Caso não exista a deslocação, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia bb@fct.unl.pt 21-12-2017↵
Deslocacao nao existe.↵
aa@fct.unl.pt >
```

Caso não exista o utilizador dado, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia naoexiste@gmail.com 21-12-2018↵
Utilizador inexistente.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
fb@fct.unl.pt > boleia mtmp@fct.unl.pt 0-12-2018↵
Data invalida.↵
fb@fct.unl.pt >
```

Exemplo da execução do comando `boleia`, quando o programa está em modo inicial.

```
> boleia mtmp@fct.unl.pt 21-12-2018↵
Comando inexistente.↵
>
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"; (2) não existir o utilizador dado; (3) data invalida; (4) ser uma deslocação do próprio; (5) não existir a deslocação dada; (6) já ter registado uma boleia nesse dia; e (7) não ter vaga na deslocação.

3.9 Comando `retira`

Retira uma dada boleia. Pode-se eliminar a boleia registada para o dia indicado, sempre e quando o programa esteja em modo sessão, essa boleia tenha sido registada pelo utilizador da sessão.

Caso o utilizador tenha as seguintes boleias registadas:

```
aa@fct.unl.pt > lista boleias↵
nn@fct.unl.pt↵
Lisboa↵
Alvalade do Sado↵
23-12-2018 15:20 20 5↵
Lugares vagos: 3↵
Boleias registadas: aa@fct.unl.pt; bb@fct.unl.pt;↵
↵
hh@fct.unl.pt↵
Porto↵
Monte da Caparica↵
25-12-2018 10:05 145 5↵
Lugares vagos: 2↵
Boleias registadas: dd@fct.unl.pt; aa@fct.unl.pt; bb@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso o utilizador trate de retirar o registo de boleia para o dia 23-12-2018, e esse utilizador tenha uma boleia registada nesse dia, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > retira 23-12-2018↵
Fernanda AA boleia retirada.↵
aa@fct.unl.pt >
```

Caso o utilizador volte a listar as boleias registadas:

```
aa@fct.unl.pt > lista boleias↵
hh@fct.unl.pt↵
Porto↵
Monte da Caparica↵
25-12-2018 10:05 145 5↵
Lugares vagos: 2↵
Boleias registadas: dd@fct.unl.pt; bb@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso o utilizador trate de retirar um registo de boleia para o dia 22-12-2018, e esse utilizador não tenha uma boleia registada nesse dia, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > retira 22-12-2018↵
Fernanda AA nesta data nao tem registo de boleia.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > retira 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

Um exemplo da execução do comando `retira`, quando o programa está em modo inicial.

```
> retira 23-12-2018↵
Comando inexistente.↵
>
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"; (2) data invalida; e (3) não existir uma boleia nesse dia para o utilizador.

3.10 Remover uma deslocação

Caso o utilizador tenha registado uma deslocação nesse dia, essa deslocação será eliminada se não houver registo de boleias para essa deslocação. Por exemplo, se o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) tiver as seguintes deslocações:

```
aa@fct.unl.pt > lista minhas↵
aa@fct.unl.pt↵
Lisboa↵
Alvalade do Sado↵
23-12-2018 15:20 110 5↵
Lugares vagos: 5↵
Sem boleias registadas.↵
↵
aa@fct.unl.pt↵
Porto↵
Monte da Caparica↵
25-12-2018 10:10 120 5↵
Lugares vagos: 2↵
Boleias registadas: bb@fct.unl.pt; cc@fct.unl.pt; dd@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso o utilizador trate de retirar a deslocação do dia 23-12-2018, esta é eliminada com sucesso:

```
aa@fct.unl.pt > remove 23-12-2018↵
Deslocacao removida.↵
aa@fct.unl.pt > lista minhas↵
aa@fct.unl.pt↵
Porto↵
Monte da Caparica↵
25-12-2018 10:10 120 5↵
Lugares vagos: 2↵
Boleias registadas: bb@fct.unl.pt; cc@fct.unl.pt; dd@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso o utilizador trate de remover a deslocação do dia 25-12-2018, a deslocação não é eliminada porque tem boleias registadas:

```
aa@fct.unl.pt > remove 25-12-2018↵
Fernanda AA ja nao pode eliminar esta deslocacao.↵
aa@fct.unl.pt >
```

Caso o utilizador trate de remover um registo de deslocação para o dia 22-12-2018, e esse utilizador não tenha uma deslocação registada nesse dia, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > remove 22-12-2018↵
Fernanda AA nesta data nao tem registo de deslocacao.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > remove 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

Um exemplo da execução do comando `remove`, quando o programa está em modo inicial.

```
> remove 23-12-2018↵
Comando inexistente.↵
>
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"; (2) data invalida; e (3) não existir uma deslocação nesse dia para o utilizador; (4) a deslocação tem boleias já registadas.

4 A Entrega

O trabalho tem 2 fases, sendo que ambas tem uma tarefa de análise e desenho do problema comúm. Ambas as fases envolve a entrega de um programa, o qual deve ser entregue nos concursos do Mooshak para o efeito.

Cada grupo de 2 alunos (equipa de trabalho) deve registar-se nesses concursos. O nome do utilizador deve ser o seu **número de aluno_número de aluno** (o primeiro número a colocar é o menor) no grupo correspondente ao seu turno (por exemplo, os alunos nº 3435 e nº 3434 do turno p1 deve ter como nome utilizador no mooshak **3434_3435** no grupo **P1**). Só serão aceites como entregas para avaliação, os programas cujo utilizador no concurso do Mooshak siga estas regras.

Para os programas e relatório serem avaliados, os alunos devem ter cumprido o código de ética do DI, e ter, no final do prazo, uma entrega dum programa no concurso do mooshak associado a essa entrega. Pode submeter o seu código mais que uma vez. Será a última submissão que será avaliada.

4.1 1ª Fase

A 1ª fase consta de um programa desenvolvido com os TADs colecção implementados com vectores, e que se encontram na página da cadeira no moodle. Esta fase tem um peso de 25% da nota final na cadeira, e faz parte da frequência da cadeira. Para obter frequência na cadeira, a 1ª fase do trabalho prático deve ter uma nota superior ou igual a 9.5 valores em 20.

Esta fase deve ser entregue até às 17:00 horas do dia 20 de Maio de 2019, ou após esta data com uma penalização de 3 valores na nota da componente prática, que no máximo pode ter 8 valores. O concurso mooshak correspondente a esta fase é "MIEEC_AED_TP1".

A avaliação do seu trabalho tem 2 componentes, sendo a nota final calculada como a soma das notas dessas componentes:

- Avaliação da correcção dos resultados produzidos: até 9 valores

  - Submissão ao concurso obtendo a pontuação máxima (100 pontos), com pelo menos 1 TAD do problema bem definido, em que usa pelo menos um TAD coleção (implementado em vector), terá 9 valores nesta componente. Note que muito provavelmente vai necessitar de mais TADs para fazer este programa **bem**.
  - Uma submissão com erros em algumas das funcionalidades terá uma classificação correspondente às funcionalidades correctamente implementadas. Por exemplo, um aluno que entregue uma aplicação que receba 90 pontos terá 90% dos 9 pontos.

- Avaliação da qualidade do código: 11 valores. Esta avaliação tem em conta a utilização dos TADs coleção adequadas à necessidade do problema e a estruturação da solução em TADs do problema (por exemplo, deslocação).

O concurso mooshak "MIEEC_AED_TP1"abre no dia 8 de Abril e fecha às 17:00 horas do dia 20 de Maio de 2019. Para entregas após esta data (com a penalização indicada anteriormente) deve usar a página do moodle (entregas fase 1 com penalização). Estes programas depois serão submetidos no concurso do mooshak pelos docentes.

4.2 2ª Fase

A 2ª fase consta dum relatório escrito e um programa, o qual é desenvolvida com os TADs coleção implementados com as estruturas de dados adequadas, e implementadas pelos os alunos (15% da nota final na cadeira).
Esta fase deve ser entregue até às 17:00 horas do dia 3 de Junho de 2019. O concurso mooshak correspondente a esta fase é "MIEEC_AED_TP2".
A avaliação do seu trabalho tem 3 componentes, sendo a nota final calculada como a soma das notas dessas componentes:

- Avaliação da correcção dos resultados produzidos: até 7 valores

  - Submissão ao concurso obtendo a pontuação máxima (100 pontos), com pelo menos 1 TAD bem definido terá 6 valores nesta componente. Note que muito provavelmente vai necessitar de mais TADs para fazer este programa **bem**. Nomeadamente os TADs coleção implementados com estruturas de dados adequadas.
  - Uma submissão com erros em algumas das funcionalidades terá uma classificação correspondente às funcionalidades correctamente implementadas. Por exemplo, um aluno que entregue uma aplicação que receba 90 pontos terá 90% dos 7 pontos.

- Avaliação da qualidade do código: 10 valores. Esta avaliação tem em conta a utilização correcta das estruturas de dados nos TADs usados e a estruturação da solução em TADs do problema (por exemplo, deslocação).

- Qualidade do relatório: 3 valores. Descrição da solução implementada e estudo da análise da complexidade do problema.

CUIDADO: Caso o aluno entregue na 2ª fase o mesmo código da 1ª fase, a nota máxima da 2ª Fase será de 3 valores (qualidade do relatório).