# UniBedrooms - Residências de estudantes

Algoritmos e Estruturas de Dados – LEEC
Ano lectivo 2022/2023
Enunciado do Trabalho Prático

versão 1.0

5 maio de 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTAMENTO
DE INFORMÁTICA

DEPARTMENT OF
COMPUTER SCIENCE

# Conteúdo



**Changelog**:

- 5/5/2023 - primeira versão 1.0

# 1 Introdução e notas importantes

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

**Trabalho em grupo de 2 alunos** a desenvolver respeitando os princípios do *Código de Ética* do Departamento de Informática. Não são aceites grupos individuais sem autorização por escrito (por ex. por email) de um docente.

**Trabalho com 2 fases** de entrega. Ver detalhes sobre submissão, datas de entrega e versões possíveis nas secções 4, 5 e 6.

# 2 Desenvolvimento da aplicação *UniBedrooms*

O objetivo deste trabalho é o desenvolvimento de um sistema de arrendamento de quartos em residências de estudantes de universidades da União Europeia.

## 2.1 Descrição do problema

Pretende-se criar uma aplicação que facilite procurar e arrendar quartos em residências de estudantes em universidades da União Europeia.

Há dois tipos de utilizadores: os estudantes universitários e os gerentes das residências. Os estudantes podem candidatar-se a um quarto em várias residências da sua escolha. Os estudantes não estão limitados a candidatarem-se a quartos de residências da sua universidade. Podem candidatar-se ao aluguer de quartos de residências de outras universidades.

Os gerentes usam o sistema para introduzir quartos (que são de residências das suas universidades), e introduzir vagas nesses quartos. A atribuição de quartos aos estudantes é feita pelos gerentes das residências e estes podem escolher qualquer um dos estudantes que se candidatou ao quarto.

Esta aplicação tem como funcionalidades principais:

- registar um estudante $e$ ou gerente $g$;
- registar um quarto $q$ numa residência;
- registar uma vaga, isto é, indicar que um determinado quarto $q$ está livre;
- consultar quartos disponíveis numa localidade $l$;
- registar candidatura de estudante $e$ ao aluguer de um quarto $q$;
- consultar candidatos a um quarto $q$;
- atribuir um quarto $q$ a um candidato $e$.

## 2.2 Conceitos e definições

Existe um conjunto de conceitos associados ao sistema a desenvolver e que se descreve a seguir:

**Estudante** – Um estudante é um utilizador que pretende alugar um quarto. É identificado por um *login* (uma cadeia de caracteres sem espaços).

**Gerente** – Um gerente é um utilizador que gere uma residência de estudantes. É identificado por um *login* (uma cadeia de caracteres sem espaços).

Gerentes e estudantes não podem ter *login*s iguais. Por exemplo, se tivermos um gerente com *login jpsousa*, nenhum estudante pode ter este *login*.

**Quarto** – O gerente pode registar quartos de residências no sistema e gerir o seu aluguer. Os quartos são identificados por um código criado pelo gerente. Por exemplo, um quarto no 1º andar da residência Fraústo da Silva pode ser registado com o nome *NOVA-RFS-Q101*.

**Localidade** – Localidade onde existe pelo menos um quarto. Podem existir várias residências, com vários quartos, na mesma localidade.

**Candidatura** – Uma candidatura é uma relação entre um estudante e um quarto. Um estudante não pode ter mais que uma candidatura activa para cada quarto. Um quarto pode ter candidaturas de vários estudantes. O gerente escolhe qual dos candidatos fica com o quarto, ou seja, o gerente escolhe qual das candidaturas ao quarto é aceite.

## 2.3 Sintaxe

Pretende-se que a interface da aplicação seja muito simples, de modo a poder ser utilizada em ambientes diversos e, no caso da saída, para permitir automatizar o processo de teste. Por estes motivos, a entrada e a saída deverá respeitar o formato rígido que se indica na Secção 3.

Convém referir que o símbolo ↵ representa uma mudança de linha e que cada comando termina com duas mudanças de linha. Antes de cada comando de entrada deverá ser impresso o símbolo > seguido de um espaço. Poderá admitir que a entrada está sempre sintaticamente correta e que os dados satisfazem as restrições enunciadas na Secção 2.4. Não existe necessidade de, durante a execução, verificar se os dados inseridos são dos tipos definidos.

## 2.4 Dados e resultados

Os nomes de estudantes, gerentes, residências, universidades e localidades têm 50 caracteres e todos os textos (descrições) têm até 200 caracteres. As restantes cadeias de caracteres terão um comprimento de 20. As cadeias de caracteres de dados a inserir no sistema não deverão conter mudanças de linha (símbolo ↵).

A *idade* dos estudantes é um inteiro. O *andar* onde se situa um quarto também é um inteiro.

O número de estudantes, gerentes, quartos e localidades não é limitado superiormente. Poderemos, no entanto, prever um número de 20 000 estudantes, 1 000 gerentes, 10 000 quartos e 500 localidades. O número de candidatos a um quarto não tem limites mas cada estudante pode candidatar-se, no máximo, a **10 quartos**.

O sistema faz distinção entre maiúsculas e minúsculas. Todos os comandos estão em maiúsculas. Não serão incluídas palavras com acentos ou cedilha. No entanto, o output da informação introduzida deverá ser gerado tal como foi inserido. Isto significa que, no exemplo dado, se o código do quarto foi inserido como (*NOVA-rfs-Q101*), o sistema deverá listá-lo nesta forma, quando gerar o seu output.

## 2.5 Especificação do sistema

A interação do utilizador com o programa é feita sempre através de comandos, descritos na próxima secção. O sistema deverá ler comandos da entrada padrão (stdin), processando-os um a um e enviando os resultados para a saída padrão (stdout).

# 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.

Deverão desenvolver e testar os comandos incrementalmente. Alguns comandos apenas serão necessários para a Fase 2.

Vários comandos têm parâmetros. A entrada e a saída deverão respeitar o formato rígido que se indica neste documento. 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.

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.

Nos vários exemplos que se seguem, o símbolo ↵ denota a mudança de linha. Cada comando termina com duas mudanças de linha.

A Tabela 1 apresenta os comandos do sistema, cujos detalhes são apresentados nas secções seguintes. Caso o utilizador introduza um comando inexistente, o programa deverá escrever Comando invalido.. Por exemplo o comando inexistente `desconhecido` teria este efeito:

```
> desconhecido↵
Comando invalido.↵
↵
>
```

Caso o utilizador coloque o símbolo # tudo o que for colocado a seguir será um comentário e não terá efeito:

```
> # um comentario ...  ↵
↵
>
```

Tabela 1: Comandos

| Comando | Descrição | Detalhes |
|---|---|---|
| IE | inserção de novo estudante | Secção 3.1 |
| DE | informação sobre um estudante | Secção 3.2 |
| IG | inserção de novo gerente | Secção 3.3 |
| DG | informação sobre um gerente | Secção 3.4 |
| IQ | inserção de novo quarto | Secção 3.5 |
| DQ | informação sobre um quarto | Secção 3.6 |
| MQ | modificação do estado de um quarto | Secção 3.7 |
| RQ | remoção de um quarto | Secção 3.8 |
| IC | inserção de candidatura de um estudante a um quarto | Secção 3.9 |
| AC | aceitaçao de candidatura (ou seja, atribuição de quarto a um estudante) | Secção 3.10 |
| LC | listagem de candidaturas a um quarto | Secção 3.11 |
| XS | termina a execução do programa | Secção 3.15 |
| Fase 2 | | |
| LQ | listagem de todos os quartos | Secção 3.12 |
| LL | listagem de quartos vagos de uma localidade | Secção 3.13 |
| LT | listagem dos top 3 gerentes | Secção 3.14 |

## 3.1 Comando **IE** – inserir novo estudante

**Regista um novo estudante no sistema.** Para realizar um registo de estudante deve ser dado sempre o *login*, que tem que ser único. São também introduzidos o nome do estudante, a sua idade, localidade de residência e universidade.

O sistema poderá guardar vários estudantes. O acesso a cada estudante será conseguido por *login*. O *login* identifica o estudante de forma única.

O formato geral do comando é o seguinte (parâmetros com estilo **bold**):

```
> IE login nomeEstudante↵
idade localidade↵
universidade↵
Registo de estudante executado.↵
↵
>
```

Este comando só tem sucesso se não existir um utilizador (estudante ou gerente) com o *login* dado. Exemplo da execução do comando IE quando o *login* *aalmeida* já existe no sistema. Neste caso, escreve-se na consola Utilizador ja existente.

```
> IE aalmeida Antonio Almeida↵
19 Leiria↵
Universidade Nova de Lisboa↵
Utilizador ja existente.↵
↵
>
```

## 3.2 Comando **DE** - consulta dados de estudante

**Consulta os dados de um dado estudante.** Quando este comando é executado, a informação do estudante é escrita na consola.

O formato geral do comando é o seguinte:

```
> DE login↵
login, nomeEstudante, idade anos, localidade↵
universidade↵
↵
>
```

Este comando só tem sucesso se o *login* do estudante existir. Exemplo da execução do comando DE quando o estudante com o *login* dado (ex. *ccorreia*) não existe. (Nota: pode existir um gerente com *login* dado.)

```
> DE ccorreia↵
Inexistencia do estudante referido.
↵
>
```

### 3.3 Comando `IG` – inserir novo gerente

**Regista um novo gerente no sistema.** Para realizar um registo de gerente deve ser dado sempre o *login*, que tem que ser único. São também introduzidos o nome do gerente e a universidade a que pertence.

O sistema poderá guardar vários gerentes. O acesso a cada gerente será conseguido por *login*. O *login* identifica o gerente de forma única.

O formato geral do comando é o seguinte:

```
> IG login nomeGerente↵
universidade↵
Registo de gerente executado.↵
↵
>
```

Este comando só tem sucesso se não exisitir um estudante nem um gerente com o *login* dado. Exemplo da execução do comando `IG` quando o *login* *rgomes* já existe no sistema. Neste caso, escreve-se na consola Utilizador ja existente.

```
> IG rgomes Raquel Gomes↵
Universidade Nova de Lisboa↵
Utilizador ja existente.↵
↵
>
```

### 3.4 Comando `DG` - consulta dados de gerente

**Consulta os dados de um dado gerente.** Quando este comando é executado, a informação do gerente é escrita na consola.

O formato geral do comando é o seguinte:

```
> login↵
login, nomeGerente↵
universidade↵
↵
>
```

Este comando só tem sucesso se o gerente existir. Exemplo da execução do comando `DG` quando o gerente com o *login* dado (ex. *aantunes*) não existe. (Nota: pode existir um estudante com *login* dado.)

```
> aantunes↵
Inexistencia do gerente referido.↵
↵
>
```

### 3.5 Comando `IQ` – inserir novo quarto

**Regista um novo quarto no sistema.** Os gerentes podem registar novos quartos no sistema. Para realizar um registo deve ser dado sempre o código do quarto, que tem que ser único, e o *login* do gerente. São também introduzidos o nome da residência, a universidade, a localidade, andar e uma descrição. Quando um quarto é registado, fica automaticamente registado como vago.

O sistema poderá guardar vários quartos. O acesso a cada quarto será conseguido por código do quarto. O código identifica o quarto de forma única.

O formato geral do comando é o seguinte:

```
> IQ codigo loginGerente↵
nomeResidencia↵
universidade↵
localidade↵
andar↵
descricao↵
Registo de quarto executado.↵
↵
>
```

Este comando só tem sucesso se (1) não existir um quarto com o código dado, se (2) existir um gerente com *login* dado e se (3) o quarto pertencer à universidade do gerente. A ordem de verificação de erros neste comando é: (1) existência do quarto, (2) existência de gerente com o *login* dado, (3) verificação da universidade.

Exemplo da execução do comando `IQ` quando o código *NOVA-RFS-Q101* já existe no sistema. Neste caso, escreve-se na consola Quarto existente.

```
> IQ NOVA-RFS-Q101 aantunes↵
Residencia Frausto da Silva↵
Universidade Nova de Lisboa↵
Caparica↵
1↵
Quarto individual com casa de banho privada.  Janela orientada a poente.↵
Quarto existente.↵
↵
>
```

Exemplo da execução do comando `IQ` quando o código *NOVA-RFS-Q901* não existe no sistema mas o gerente *aantunes* não existe. Neste caso, escreve-se na consola Inexistencia do gerente referido. (Nota, pode existir um estudante com o *login* referido.)

```
> IQ NOVA-RFS-Q901 aantunes↵
Residencia Frausto da Silva↵
Universidade Nova de Lisboa↵
Caparica↵
1↵
Quarto individual com casa de banho privada.  Janela orientada a poente.↵
Inexistencia do gerente referido.↵
↵
>
```

Exemplo da execução do comando `IQ` quando o código *NOVA-RFS-Q901* não existe no sistema, o gerente *tpaiva* existe e é da *Universidade de Evora*. Neste caso, escreve-se na consola Operacao nao autorizada.

```
> IQ NOVA-RFS-Q901 tpaiva↵
Residencia Frausto da Silva↵
Universidade Nova de Lisboa↵
Caparica↵
1↵
Quarto individual com casa de banho privada.  Janela orientada a poente.↵
Operacao nao autorizada.↵
↵
>
```

### 3.6 Comando **DQ** - consulta dados de quarto

**Consulta os dados de um dado quarto.** Quando este comando é executado, a informação do quarto é escrita na consola. Este comando só tem sucesso se o quarto existir.

O formato geral do comando é o seguinte:

```
> DQ codigo↵
codigo, nomeResidencia
universidade
localidade
andar
descricao
estado
↵
>
```

A informação sobre o estado indica se o quarto está livre ou ocupado.

Exemplo da execução do comando DQ quando o quarto com o código dado (ex. *NOVA-RFS-Q101*) existe e está livre.

```
> DQ NOVA-RFS-Q101↵
NOVA-RFS-Q101, Residencia Frausto da Silva
Universidade Nova de Lisboa
Caparica
1
Quarto individual com casa de banho privada. Janela orientada a poente.
livre
↵
>
```

Exemplo da execução do comando DQ quando o quarto com o código dado (ex. *NOVA-RFS-Q102*) existe e está ocupado.

```
> DQ NOVA-RFS-Q102↵
NOVA-RFS-Q101, Residencia Frausto da Silva
Universidade Nova de Lisboa
Caparica
1
Quarto individual com casa de banho privada. Janela orientada a nascente.
ocupado
↵
>
```

Exemplo da execução do comando DQ quando o quarto com o código dado (ex. *NOVA-RFS-Q901*) não existe.

```
> DQ NOVA-RFS-Q901↵
Inexistencia do quarto referido.
↵
>
```

## 3.7 Comando MQ – modificar estado de um quarto

**Modifica estado de um quarto.** O gerente de um quarto pode alterar o seu estado para *livre* ou *ocupado*. Para isso, o gerente indica o seu login e o código do quarto.

O formato geral do comando é o seguinte, onde *estado* pode ser *livre* ou *ocupado*:

```
> MQ codigoQuarto loginGerente estado↵
Estado de quarto atualizado.↵
↵
>
```

Este comando só tem sucesso se (1) existir um quarto com o código dado, se (2) o login dado corresponde ao login do gerente deste quarto e, (3) no caso do novo estado ser *ocupado*, se o quarto não tiver candidaturas activas. A ordem de verificação de erros neste comando é: (1) existência do quarto, (2) login dado é o login do gerente do quarto, e (3) no caso do parâmetro *estado* ter o valor *ocupado*, inexistência de candidaturas activas.

Quando o quarto não existe, escreve-se na consola Inexistencia do quarto referido. Exemplo da execução do comando MQ quando o código *NOVA-RFS-Q901* não existe no sistema.

```
> MQ NOVA-RFS-Q901 jpsousa livre↵
Inexistencia do quarto referido.↵
↵
>
```

Quando o quarto existe mas o login dado não corresponde ao login do seu gerente, escreve-se na consola Operacao nao autorizada. Exemplo da execução do comando MQ quando o código *NOVA-RFS-Q101* existe no sistema e foi registado pelo gerente *jpsousa*.

```
> MQ NOVA-RFS-Q101 aantunes ocupado↵
Operacao nao autorizada.↵
↵
>
```

Quando o quarto existe, o login dado corresponde ao login do seu gerente, mas o quarto tem candidaturas activas e o parâmetro *estado* tem o valor *ocupado*, escreve-se na consola Candiaturas activas. Exemplo da execução do comando MQ quando o código *NOVA-RFS-Q103* existe no sistema, foi registado pelo gerente *jpsousa* e tem candidaturas activas.

```
> MQ NOVA-RFS-Q103 jpsousa ocupado↵
Candidaturas activas.↵
↵
>
```

## 3.8 Comando `RQ` - remoção de um quarto

**Remove um quarto do sistema.** O gerente de um quarto pode removê-lo do sistema. Para isso, o gerente indica o seu login e o código do quarto.

O formato geral do comando é o seguinte:

```
> RQ codigoQuarto loginGerente↵
Remocao de quarto executada.↵
↵
>
```

Este comando só tem sucesso se (1) existir um quarto com o código dado, se (2) o login dado corresponde ao login do gerente deste quarto e se (3) não existirem candidaturas a este quarto. A ordem de verificação de erros neste comando é: (1) existência do quarto, (2) login dado é o login do gerente do quarto, (3) existência de candidaturas a este quarto.

Quando o quarto não existe, escreve-se na consola Inexistencia do quarto referido. Exemplo da execução do comando RQ quando o código *NOVA-RFS-Q901* não existe no sistema.

```
> RQ NOVA-RFS-Q901 jpsousa↵
Inexistencia do quarto referido.↵
↵
>
```

Quando o quarto existe mas o login dado não corresponde ao login do seu gerente, escreve-se na consola Operacao nao autorizada. Exemplo da execução do comando RQ quando o código *NOVA-RFS-Q101* existe no sistema e foi registado pelo gerente *jpsousa*.

```
> RQ NOVA-RFS-Q101 aantunes↵
Operacao nao autorizada.↵
↵
>
```

Quando o quarto existe e o login dado corresponde ao login do seu gerente, mas o quarto tem candidaturas activas, escreve-se na consola Candidaturas activas. Exemplo da execução do comando RQ quando o código *NOVA-RFS-Q103* existe no sistema, foi registado pelo gerente *jpsousa* e tem candidaturas activas.

```
> RQ NOVA-RFS-Q103 jpsousa↵
Candidaturas activas.↵
↵
>
```

## 3.9 Comando `IC` – inserir candidatura

**Regista candidatura de um estudante a um quarto.** Para registar a candidatura a um quarto devem ser dados o *login* do estudante e o código do quarto.

O sistema poderá registar várias candidaturas. Um quarto pode ter candidaturas activas de vários estudantes (sem limite superior). Quando um quarto que estava alugado volta a estar disponível, qualquer estudante se pode candidatar a ele (mesmo que no passado já se tenha candidato).

O formato geral do comando é o seguinte:

```
> IC loginEstudante codigoQuarto↵
Registo de candidatura executado.↵
↵
>
```

Este comando só tem sucesso se (1) o estudante existir, se (2) o estudante tiver menos de 10 candidaturas activas, se (3) o quarto existir, (4) se o quarto estiver livre, e se (5) não houver uma candidatura activa do estudante referido ao quarto referido. A ordem de verificação de erros neste comando é: (1) existência do estudante, (2) estudante com menos de 10 candidaturas activas, (3) existência de quarto, (4) estado do quarto, (5) inexistência de candidatura activa do estudante ao quarto.

Quando o estudante não existe no sistema, escreve-se na consola Inexistencia do estudante referido. Exemplo da execução do comando `IC` quando o estudante com o *login ccorreia* não existe no sistema. (Nota: pode existir um gerente com *login* dado.)

```
> IC ccorreia NOVA-RFS-Q101↵
Inexistencia do estudante referido.↵
↵
>
```

Exemplo da execução do comando `IC` quando o estudante com o *login bmoreira* foi registado mas tem 10 candidaturas activas.

```
> IC bmoreira NOVA-RFS-Q101↵
Operacao nao autorizada.↵
↵
>
```

Exemplo da execução do comando `IC` quando o estudante com o *login bmadeira* foi registado, tem menos de 10 candidaturas activas, mas o quarto *NOVA-RFS-Q901* não foi registado.

```
> IC bmadeira NOVA-RFS-Q901↵
Inexistencia do quarto referido.↵
↵
>
```

Exemplo da execução do comando `IC` quando o estudante com o *login bmadeira* foi registado, tem menos de 10 candidaturas activas, o quarto *NOVA-RFS-Q102* foi registado, mas está ocupado.

```
> IC bmadeira NOVA-RFS-Q102↵
Quarto ocupado.↵
↵
>
```

Exemplo da execução do comando `IC` quando o estudante com o *login bmadeira* foi registado, tem menos de 10 candidaturas activas, o quarto *NOVA-RFS-Q101* foi registado e está livre, mas o estudante tem uma candidatura activa a este quarto.

```
> IC bmadeira NOVA-RFS-Q101↵
Candidatura existente.↵
↵
>
```

### 3.10 Comando **AC** – aceitar candidatura

**Aceita candidatura de um estudante a um quarto.** O gerente de um quarto pode atribuí-lo a um estudante que se tenha candidatado ao quarto. Para isso, ao chamar o comando AC, o gerente deve dar o seu login, o código do quarto e o login do estudante.

Quando um quarto $q$ é atribuído a um estudante $e$: (1) o estado do quarto passa a ser *ocupado*, (2) o estudante $e$ deixa de ter candidaturas activas, (3) o quarto deixa de ter outras candidaturas activas e (4) todos os outros estudantes com candidatura activa ao quarto $q$, deixam de ter essa candidatura.

O formato geral do comando é o seguinte:

```
> AC codigoQuarto loginGerente loginEstudante↵
Aceitacao de candidatura executada.↵
↵
>
```

Este comando só tem sucesso se (1) o quarto referido existir, se (2) *loginGerente* corresponde ao login do gerente deste quarto e se (3) houver uma candidatura activa do estudante com login *loginEstudante* ao quarto. A ordem de verificação de erros neste comando é: (1) existência do quarto, (2) *loginGerente* é o login do gerente do quarto e (3) existência de candidatura activa do estudante ao quarto.

Exemplo da execução do comando AC quando o quarto *NOVA-RFS-Q901* não foi registado.

```
> AC NOVA-RFS-Q901 aantunes ccorreia↵
Inexistencia do quarto referido.↵
↵
>
```

Quando o quarto existe mas o login de gerente dado não corresponde ao login do seu gerente, escreve-se na consola Operacao nao autorizada. Exemplo da execução do comando AC quando o quarto com o código *NOVA-RFS-Q101* existe no sistema e foi registado pelo gerente *jpsousa*.

```
> AC NOVA-RFS-Q101 aantunes ccorreia↵
Operacao nao autorizada.↵
↵
>
```

Exemplo da execução do comando AC quando o estudante com o login *ccorreia* não tem uma candidatura a este quarto. (Nota: o estudante pode existir ou não existir no sistema.)

```
> AC NOVA-RFS-Q101 jpsousa ccorreia↵
Inexistencia da candidatura referida.↵
↵
>
```

### 3.11 Comando `LC` – listagem de candidaturas a um quarto

**Lista os candidatos a um quarto.** O gerente de um quarto pode consultar os candidatos a esse quarto através da listagem dos estudantes com candidatura activa ao quarto. A listagem é dada por ordem cronológica de criação de candidatura. Para chamar este comando deve ser dado o login do gerente do quarto e o código do quarto. Na listagem, para cada candidato, escreve-se na consola o seu login, nome e universidade.

O formato geral do comando é o seguinte:

```
> LC codigoQuarto loginGerente↵
loginE1, nomeE1, universidadeE1↵
loginE2, nomeE2, universidadeE2↵
...
loginEn, nomeEn, universidadeEn↵
↵
>
```

*loginEi*, *nomeEi* e *universidadeEi* representam o login, nome e universidade do *i*-ésimo estudante que tem candidatura activa ao quarto com o código *codigoQuarto*.

Este comando só tem sucesso se (1) o quarto com código *codigoQuarto* existir, se (2) o *loginGerente* corresponde ao login do gerente deste quarto, e se (3) o quarto tem candidaturas registadas. A ordem de verificação de erros neste comando é: (1) existência do quarto, (2) login dado é o login do gerente do quarto, (3) existência de candidaturas ao quarto referido.

Quando o quarto não existe no sistema, escreve-se na consola Inexistencia do quarto referido. Exemplo da execução do comando `LC` quando o quarto com o código *NOVA-RFS-Q901* nao existe no sistema.

```
> LC NOVA-RFS-Q901 jpsousa↵
Inexistencia do quarto referido.↵
↵
>
```

Quando o quarto existe mas o login dado não corresponde ao login do seu gerente, escreve-se na consola Operacao nao autorizada. Exemplo da execução do comando `LC` quando o quarto com o código *NOVA-RFS-Q101* existe no sistema e foi registado pelo gerente *jpsousa*.

```
> LC NOVA-RFS-Q101 aantunes↵
Operacao nao autorizada.↵
↵
>
```

Quando o quarto não tem candidaturas, escreve-se na consola Inexistencia de candidaturas. Exemplo da execução do comando `LC` quando o quarto *NOVA-RFS-Q201* foi registado no sistema pelo gerente *jpsousa* mas não tem candidaturas registadas. (Nota: o quarto pode estar livre ou ocupado. Se o quarto está ocupado, não tem candidaturas activas.)

```
> LC NOVA-RFS-Q201 jpsousa↵
Inexistencia de candidaturas.↵
↵
>
```

## 3.12 Comando `LQ` – listagem de todos os quartos

**Lista os quartos do sistema.** Listagem de todos os quartos existentes no sistema. A listagem é feita por ordem lexicográfica de localidade e código de quarto. Ou seja, a listagem é feita usando a ordem lexicográfica de localidade, e para cada localidade listam-se todos os quartos desse local por ordem lexicográfica de código de quarto.

O comando não tem parâmetros. Para cada quarto, escreve-se na consola a localidade, código, nome da universidade e nome da residência.

O formato geral do comando é o seguinte:

```
> LQ↵
localidadeQ1 codigoQ1↵
universidadeQ1↵
residenciaQ1↵
↵
localidadeQ2 codigoQ2↵
universidadeQ2↵
residenciaQ2↵
↵
...
localidadeQn codigoQn↵
universidadeQn↵
residenciaQn↵
↵
>
```

*localidadeQi*, *codigoQi*, *universidadeQi* e *residenciaQi* representam o localidade, código, nome da universidade e nome da residência e do $i$-ésimo quarto.

Este comando só tem sucesso se existirem quartos registados no sistema. Quando não há quartos registados, escreve-se na consola Inexistencia de quartos. Exemplo da execução do comando `LQ` quando não há nenhum quarto registado no sistema.

```
> LQ↵
Inexistencia de quartos.↵
↵
>
```

**Fase 1 e fase 2**

Este comando só será testado na fase 2.

## 3.13 Comando `LL` – listagem de quartos vagos de uma localidade

**Lista os quartos vagos numa localidade.** A listagem é feita por ordem lexicográfica de código de quarto. Para chamar este comando deve ser dado o nome da localidade.

Para cada quarto, escreve-se na consola a localidade, código do quarto, nome da universidade e nome da residência.

O formato geral do comando é o seguinte:

```
> LL localidade↵
localidade codigoQ1↵
universidadeQ1↵
residenciaQ1↵
↵
localidade codigoQ2↵
universidadeQ2↵
residenciaQ2↵
↵
...
localidade codigoQn↵
universidadeQn↵
residenciaQn↵
↵
>
```

*codigoQi*, *universidadeQi* e *residenciaQi* representam o código, nome da universidade e nome da residência e do $i$-ésimo quarto registado em *localidade*.

Este comando só tem sucesso se a localidade com nome *localidade* existir e tiver quartos registados. Há três situações em que o comando não tem sucesso: (1) quando nunca foi registado um quarto nessa localidade, (2) quando houve quartos registados na localidade (com o comando IQ) mas que entretanto foram removidos (com o comando RQ), e (3) quando há quartos na localidade mas estão todos ocupados (ou seja, não há quartos livres na localidade). Em qualquer uma destas situações, escreve-se na consola Inexistencia de quartos na localidade referida. Exemplo da execução do comando LL quando a localidade *Para la do sol posto* não existe no sistema.

```
> LL Para la do sol posto↵
Inexistencia de quartos na localidade referida.↵
↵
>
```

**Fase 1 e fase 2**

Este comando só será testado na fase 2.

## 3.14 Comando LT – listagem de Top 3 gerentes

**Lista os top 3 gerentes do sistema.** Listagem dos 3 gerentes com mais quartos alugados no sistema por ordem descrescente. Em caso de empate a listagem é feita por ordem lexicográfica de código de gerente. Se não existirem 3 gerentes imprime só os que existirem. Considera-se que um quarto foi alugado apenas quando foi efetuado o comando de aceitar candidatura.

O comando não tem parâmetros. Para cada gerente, escreve-se na consola o seu login e quantos quartos alugou.

O formato geral do comando é o seguinte:

```
> LT↵
codigoGerente1 alugou n1 quartos↵
codigoGerente2 alugou n2 quartos↵
codigoGerente3 alugou n3 quartos↵
↵
>
```

Este comando só tem sucesso se existirem gerentes registados no sistema. Quando não há gerentes registados, escreve-se na consola Inexistencia de gerentes. Exemplo da execução do comando LT quando não há nenhum gerente registado no sistema.

```
> LT↵
Inexistencia de gerentes.↵
↵
>
```

**Fase 1 e fase 2**

Este comando só será testado na fase 2.

### 3.15 Comando XS

**Termina a execução do programa.** O comando não tem parâmetros e tem sempre sucesso.

O formato do comando é o seguinte:

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

## 4 Desenvolvimento

**Todos os estudantes que realizam o trabalho prático têm de estar registados num turno prático.**

O trabalho é realizado em **grupos de dois alunos.** (**NOTA**: tal como referido nas aulas, **não são aceites grupos individuais** sem autorização por escrito dos docentes. Esta autorização não será dada na véspera da entrega. Portanto, se um aluno pretender fazer o trabalho individual porque tem uma justificação de força maior, deve contactar o docente do seu turno prático o mais brevemente possível.)

O trabalho é implementado em C, com o compilador GNU GCC.

### 4.1 Entregas e faseamento

Como já descrito acima, o trabalho será desenvolvido incrementalmente, em duas fases diferentes, com entregas marcadas.

Na primeira fase, os estudantes deverão desenhar a sua solução utilizando os tipos abstractos de dados e respectivas estruturas de dados fornecidos implementadas em vector.

Todas as operações deverão ser implementadas na fase dois. A escolha dos tipos abstractos de dados deve ser revista na fase dois para melhorar a performance do trabalho da fase 1 para a fase 2. É expectável que alguns dos tipos abstractos de dados usados na solução da fase 1 sejam alterados na fase 2. (Será dada mais informação sobre este assunto nas aulas práticas e teóricas.)

O programa deve ser entregue nos concursos do Mooshak para o efeito. Adicionalmente, na última fase deverá ainda ser entregue junto do código, um relatório final do trabalho em .pdf (Secção 4.2).

Cada grupo de **dois alunos** (equipa de trabalho) deve registar-se nesses concursos. O nome do utilizador (*login* no mooshak) 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 do turno P1 e nº 3434 do turno P4 devem ter como nome utilizador no Mooshak **3434_3435**). **Só serão aceites como entregues para avaliação, os programas cujo utilizador no concurso do Mooshak siga estas regras.**

A entrega no Mooshak, é constituída por um zip contendo o código fonte devidamente comentado. O nome e número dos alunos que compõem o grupo deverá ser inserido no cabeçalho de todos os ficheiros entregues, da seguinte forma:

```
/**
* @author (NUMEROALUNO1) NOMEALUNO1 emailALUNO1
* @author (NUMEROALUNO2) NOMEALUNO2 emailALUNO2
*/
```

Para os programas e relatório serem avaliados, os alunos devem ter cumprido o Código de Ética do DI e o Regulamento de Avaliação de Conhecimentos da FCT, 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, até 20 vezes. Só a submissão mais alta será avaliada.

### 4.2 Relatório

Na última fase deverá ainda ser entregue, no Moodle, um relatório final do trabalho, contendo o seguinte:

- Para cada um dos Tipos Abstractos de Dados definidos no trabalho, uma justificação curta para as implementações realizadas para os mesmos, especificamente para as estruturas de dados escolhidas;
- Para cada uma das operações descritas na Secção 3, uma descrição do comportamento do trabalho, em termos das operações efetuadas sobre as estruturas de dados;
- O estudo das complexidades temporais e espaciais das operações descritas na Secção 3, no no pior caso e no caso esperado, de acordo com *template* a fornecer.

Será fornecido proximamente um *template* para o relatório.

## 5 Requisitos do Mooshak

Para submeter o trabalho ao Mooshak, é necessário que o programa fonte respeite as quatro regras seguintes:

- O código fonte completo (ficheiros *.c e *.h) tem de ser guardado num único arquivo ZIP.
- O programa principal deverá chamar-se roomsMain.c .
- Para evitar problemas causados pela dimensão do ficheiro submetido, somente o código fonte (ficheiros *.c e *.h) deve ser enviado para o servidor.
- Deverá seguir a Metodologia de Programação da cadeira (ver Secção Moodle).

## 6 Avaliação

O trabalho é obrigatório para todos os alunos que não têm frequência e dá frequência. A avaliação incidirá sobre todos os aspectos: concepção, qualidade e eficiência da solução, modularidade, estrutura e documentação do código, qualidade do relatório final, etc. A fase 1 vale 25% da nota final da disciplina e a fase 2 vale 15% da mesma nota. Se uma das fases não for entregue dentro do prazo definido, a nota associada a essa fase será 0 (zero). A avaliação de cada fase é composta por **9 valores** com avaliação automática pelo Mooshak e por **11 valores** de avaliação qualitativa dada pelo docente. A nota da avaliação qualitativa poderá ser negativa, caso o projecto não siga a metodologia de programação da disciplina. A nota final é a soma das notas destas duas componentes.

## 6.1 Testes de avaliação

O trabalho terá de satisfazer todos os requisitos especificados na Secção 3. Essa verificação será efetuada automaticamente pelo sistema Mooshak, com os Testes de Avaliação.

Em cada fase:

- se o programa não tiver pelo menos 60 pontos no mooshak, os seus autores obterão a nota de 0 (zero) na fase em questão;
- se o programa tiver 60 ou mais, os docentes farão uma avaliação preliminar da fase, que deverá ser confirmada no final do semestre, numa discussão final.

## 6.2 Verificação de autoria

**Código de Ética.** De acordo com o Regulamento de avaliação de conhecimentos FCT NOVA:

- existe fraude quando se:
  (a) usa ou tenta usar em qualquer forma, num teste, exame, ou outro elemento de avaliação, presencial ou remoto, informação ou equipamento não autorizado;
  (b) fornece ou recebe colaboração não autorizada para realizar um teste, exame, ou outro elemento de avaliação;
  (c) fornece ou recebe colaboração não autorizada pelas regras, definidas em cada caso, para realozar trabalhos práticos, relatórios ou outros elementos de avaliação.
- Estudantes directamente envolvidos em fraude não aprovam na UC e não terão frequência.

**Requisitos éticos.** O código submetido pelos alunos como fazendo parte do seu trabalho deve ser desenvolvido **de raiz pelos mesmos**, expressamente para o trabalho em questão. As exceções a esta regra serão os TADs disponibilizados na página Moodle da disciplina.

No final do semestre, todos os grupos em posição de obter frequência poderão ser submetidos a uma discussão do trabalho, para verificação de autoria. Nesta discussão, os membros do grupo poderão ser questionados sobre **qualquer parte de código submetida ao Mooshak em qualquer das fases de submissão**. Se se detetar:

- que um trabalho não foi realizado apenas pelos alunos que o assinaram;
- que um aluno assinou um trabalho que não realizou; ou
- que a distribuição das tarefas pelos membros do grupo foi muito desiquilibrada,

esse trabalho será anulado e **nenhum** dos elementos do(s) grupo(s) envolvido(s) obterá frequência. Se um aluno faltar à discussão do trabalho que assinou, não obterá frequência, reprovando à disciplina.

## 6.3 Resumo das datas importantes

- Submissão da fase 1 do trabalho, no Mooshak: **17h00, 29 de Maio**.
- Submissão da fase 2 do trabalho, no Mooshak: **17h00, 12 de Junho**.