# Estruturas de Dados e Algoritmos II
# Trabalho Prático

Departamento de Informática
Universidade de Évora

2017/2018

v1.0a

## Postas

Um pequeno país, que será aqui referido como Praquistão, não confiando na confidencialidade e segurança do *software* americano, pretende implementar um serviço que permita aos seus cidadãos a divulgação das suas opiniões, através de mensagens curtas (*postas*, na língua local). O sistema a implementar deverá permitir, aos seus utilizadores, enviar mensagens, seguir outros utilizadores, e ler as mensagens por eles enviadas.

Para já, o governo lançou um concurso em que pretende avaliar a qualidade da implementação proposta para a infraestrutura básica do serviço, que contempla a gestão do envio e da leitura de mensagens e dos seguidores dos utilizadores. Posteriormente, será pedido aos autores das melhores propostas, se tiverem qualidade suficiente, que procedam à implementação do sistema completo, incluindo autenticação, a interface para a rede e o tratamento das mensagens.

### 1 Descrição e funcionamento

O sistema deverá suportar a criação e a eliminação de utilizadores, a indicação do envio e da leitura de mensagens, e a possibilidade de um utilizador passar a ou deixar de seguir outro utilizador. Deverá, também, ser possível obter informação sobre o estado de um utilizador do sistema.

O programa que implementa este sistema lerá os comandos a executar da sua entrada normal (*standard input*) e escreverá as respostas na sua saída normal (*standard output*).

O estado do sistema deverá persistir para além da execução do programa. Toda a informação existente no sistema no fim de uma execução do programa estará disponível quando o programa voltar a ser executado.

### 2 Definições e capacidades

O sistema deverá suportar a existência de um milhão de utilizadores activos.

Um utilizador será definido através de um *nick* único e do seu nome. O *nick* é uma sequência de 5 caracteres, de entre letras minúsculas e maiúsculas, sem diacríticos, e algarismos decimais, e serve para idenficar o utilizador perante o sistema. Dois *nicks* são iguais se e só se consistirem na mesma sequência de caracteres. Um nome é uma sequência de até 25 caracteres, incluindo os espaços, e poderá haver vários utilizadores com o mesmo nome.

A relação de ordem entre *nicks* é a ordem lexicográfica obtida considerando que a ordem entre os caracteres que os compõem corresponde à ordem entre os seus códigos ASCII:

$$\texttt{0} < \texttt{1} < \ldots < \texttt{9} < \texttt{A} < \texttt{B} < \ldots < \texttt{Z} < \texttt{a} < \texttt{b} < \ldots < \texttt{z}$$

Assim, `0aaaa` < `12345` < `Zyxvu` < `aBcde` < `abCde`.

O *nick* de um utilizador removido do sistema não poderá ser reutilizado. O sistema deverá funcionar sem degradação perante a existência de até 10% de *nicks* de utilizadores removidos.

Cada utilizador poderá seguir até cem utilizadores do sistema, e poderá ser seguido por qualquer número de utilizadores.

## 3 Comandos

Os comandos a implementar são apresentados a seguir.

Os vários elementos de um comando (e de um nome, se constituído por mais do que uma palavra) são separados, entre si, por um espaço. Nenhum comando começa ou termina com um espaço.

Pode assumir, sem testar, a correcção da entrada do programa, *i.e.*, que todos os comandos estão sintacticamente correctos e que não é violada nenhuma condição estabelecida no enunciado do trabalho, excepto nas situações mencionadas na descrição de cada comando.

Nos exemplos apresentados, o símbolo '⇐' assinala o comando introduzido e o símbolo '↦' assinala a resposta do programa.

**Criar utilizador** Cria um novo utilizador do sistema, com o *nick* e o nome dados. O utilizador criado fica activo até ser removido.

Não tem efeito se o *nick* corresponder ao de um utilizador anteriormente criado.

Entrada

```
U <nick> <nome>
```

Saída

- `+ utilizador <nick> criado`
- `+ nick <nick> usado previamente`
  Se já tiver sido criado um utilizador com *nick* indicado.

Exemplos

```
⇐ U zero7 James Bond
↦ + utilizador zero7 criado
⇐ U zero7 Spectre
↦ + nick zero7 usado previamente
⇐ U Zero7 Spectre
↦ + utilizador Zero7 criado
⇐ U MrPym A Perfect Spy
↦ + utilizador MrPym criado
```

**Remover utilizador** Remove um utilizador do sistema.

Não tem efeito se não houver um utilizador activo com o *nick* indicado.

Entrada

```
R <nick>
```

Saída

- `+ utilizador <nick> removido`
- `+ utilizador <nick> inexistente`

Exemplos

```
⇐ R 00007
↦ + utilizador 00007 inexistente
⇐ R Zero7
↦ + utilizador Zero7 removido
⇐ R Zero7
↦ + utilizador Zero7 inexistente
```

**Seguir utilizador** Um utilizador começa a seguir outro utilizador. Um utilizador só lê as mensagens de um utilizador que segue que sejam enviadas a partir do momento em que o começa a seguir. É possível um utilizador seguir-se a si próprio.

Não tem efeito se algum dos *nicks* não corresponder a um utilizador activo, se o utilizador já segue o outro utilizador, ou se o utilizador já segue o número máximo de utilizadores.

Entrada

```
S <nick1> <nick2>
```

Saída

- `+ <nick1> passou a seguir <nick2>`
- `+ utilizador <nick1> segue <nick2>`
  Se o utilizador com *nick* `<nick1>` já segue o com *nick* `<nick2>`.
- `+ utilizador <nick1> segue o limite`
  Se o utilizador com *nick* `<nick1>` já atingiu o limite de utilizadores que pode seguir.
- `+ utilizador <nick1> inexistente`
- `+ utilizador <nick2> inexistente`
  Só aparece se `<nick1>` corresponder a um utilizador activo.

Exemplos

```
⇐ S 00007 zero7
↦ + utilizador 00007 inexistente
⇐ S Zero7 00007
↦ + utilizador Zero7 inexistente
⇐ S MrPym Zero7
↦ + utilizador Zero7 inexistente
⇐ S zero7 MrPym
↦ + zero7 passou a seguir MrPym
⇐ S zero7 MrPym
↦ + utilizador zero7 segue MrPym
⇐ S zero7 zero7
↦ + zero7 passou a seguir zero7
```

**Deixar de seguir utilizador** Um utilizador deixa de seguir outro utilizador.

Não tem efeito se algum dos *nicks* não corresponder a um utilizador activo ou se o utilizador não segue o outro utilizador.

Entrada

```
D <nick1> <nick2>
```

Saída

- `<nick1> deixou de seguir <nick2>`
- `<nick1> nao segue <nick2>`
- `+ utilizador <nick1> inexistente`
- `+ utilizador <nick2> inexistente`
  Só aparece se `<nick1>` corresponder a um utilizador activo.

Exemplos

```
⇐ D zero7 zero7
↦ + zero7 deixou de seguir zero7
⇐ D zero7 zero7
↦ + utilizador zero7 nao segue zero7
⇐ D Zero7 00007
↦ + utilizador Zero7 inexistente
```

**Enviar mensagem** Regista o envio, por um utilizador, de uma mensagem para o sistema. A cada mensagem de um utilizador é atribuído um número de ordem sequencial, calculado somando 1 ao número de mensagens enviadas anteriormente pelo utilizador.

Não tem efeito se o *nick* não corresponder a um utilizador activo.

Entrada

```
P <nick>
```

Saída

- *Nada*

  Em caso de sucesso, o sistema não apresenta nenhuma mensagem.
- `+ utilizador <nick> inexistente`

Exemplos

```
⇐ P Zero7
↦ + utilizador Zero7 inexistente
⇐ P zero7
⇐ P zero7
⇐ P MrPym
```

**Ler mensagens** O utilizador lê todas as mensagens dos utilizadores activos que segue que ainda não leu, enviadas desde que os começou a seguir pela última vez. Se algum dos utilizadores seguidos foi removido do sistema, o utilizador deixa de o seguir.

Não tem efeito se o *nick* não corresponder a um utilizador activo ou se o utilizador não seguir nenhum utilizador.

Entrada

```
L <nick>
```

Saída

- Para cada utilizador seguido pelo utilizador, por ordem crescente de *nick*, será mostrada uma das linhas seguintes, onde `<nome>` é o nome do utilizador seguido, tal como introduzido na sua criação:

  `sem mensagens novas de <nick> (<nome>)`

  Se não há nenhuma mensagem nova para ler.

  `mensagem nova de <nick> (<nome>): <mensagem>`

  Se a única mensagem não lida é a que tem número `<mensagem>`.

  `mensagens novas de <nick> (<nome>): <mensagem1> a <mensagem2>`

  Se as mensagens não lidas vão da número `<mensagem1>` (a primeira não lida) à número `<mensagem2>` (a última).

  `utilizador <nick> desactivado`

  Se o utilizador foi removido do sistema.
- `+ utilizador <nick> sem seguidos`

  Se o utilizador com o *nick* indicado não segue nenhum utilizador.
- `+ utilizador <nick> inexistente`

Exemplos

```
⇐ L 00007
↦ + utilizador 00007 inexistente
⇐ L zero7
↦ mensagem nova de MrPym (A Perfect Spy): 1
⇐ L zero7
↦ sem mensagens novas de MrPym (A Perfect Spy)
⇐ L MrPym
↦ + utilizador MrPym sem seguidos
```

**Obter informação sobre um utilizador** Mostra a informação relativa a um utilizador do sistema: o nome, quantas mensagens já enviou, quantos seguidores tem e quantos utilizadores segue. Para cada utilizador seguido, mostra o *nick* e o número da última mensagem lida (ou que não lerá, no caso de esse utilizador não ter enviado nenhuma mensagem desde que passou a ser seguido).

Não tem efeito se o *nick* não corresponder a um utilizador activo.

Entrada

```
I <nick>
```

Saída

- ```
  utilizador <nick> (<nome>)
  <mensagens> mensagens, <seguidores> seguidores, segue <seguidos> utilizadores
  nick1 (<mensagens1> lidas)
  ...
  nickK (<mensagensK> lidas)
  ```

  Onde `&lt;mensagens&gt;` é o número de mensagens enviadas pelo utilizador, `&lt;seguidores&gt;` é o número de utilizadores que o seguem, e `&lt;seguidos&gt;` é o número de utilizadores que segue (que inclui aqueles que possam ter sido removidos desde a última vez que o utilizador leu mensagens).

  As linhas da forma "`&lt;nicki&gt; (&lt;mensagensi&gt; lidas)`" serão tantas quantos os utilizadores seguidos, incluindo os removidos desde a última vez que o utilizador leu mensagens, por ordem lexicográfica crescente de *nick*. `&lt;mensagensi&gt;` é o número da última mensagem do utilizador com `&lt;nicki&gt;` considerada lida por este utilizador.
- `+ utilizador <nick> inexistente`

Exemplos

```
⇐ I zero7
↦ utilizador zero7 (James Bond)
↦ 2 mensagens, 0 seguidores, segue 1 utilizadores
↦ MrPym (1 lidas)
⇐ I MrPym
↦ utilizador MrPym (A Perfect Spy)
↦ 1 mensagens, 1 seguidores, segue 0 utilizadores
⇐ I 00007
↦ + utilizador 00007 inexistente
⇐ I Zero7
↦ + utilizador Zero7 inexistente
```

**Terminar a execução** Termina a execução do programa.

Entrada

```
X
```

Saída

- *Nada*

Exemplos

```
⇐ X
```

## 4 Realização e entrega

### 4.1 Realização

O trabalho será realizado individualmente ou por grupos de dois elementos. Só serão considerados os trabalhos de grupos cujos elementos tenham todos obtido a pré-qualificação para o trabalho.

O código C entregue deverá estar de acordo com o *standard* C99, poder ser compilado com o GCC e executado em Linux. O código será compilado, com as opções `-std=gnu99 -Wall -lm`, com o comando
```

```
gcc -std=gnu99 -Wall -lm *.c
```

A opção `-lm` indica ao compilador que deve incluir no executável criado uma biblioteca com funções matemáticas, como a função `sqrt`, que calcula a raiz quadrada de um número. Para usar estas funções, o programa deverá conter uma directiva

```
#include <math.h>
```

Todas as escritas e leituras de informação em disco deverão ser controladas explicitamente pelo programa.

### 4.2 Ambiente de execução e restrições

Na entrega, os programas serão testados numa máquina com sistema operativo Linux, de 64 bits, com um disco magnético normal (HD, e não SSD), e com páginas (de disco) de 4096 *bytes*.

A memória RAM utilizada pelo programa estará limitada a 64 MB e o espaço disponível no disco para dados é de 2 GiB.

### 4.3 Entrega

O trabalho será submetido através do Mooshak, uma aplicação que compilará o código e executará e testará os programas criados, no concurso "EDA2 2017 (Trabalho)". O endereço para acesso ao concurso é:

http://takatakata.di.uevora.pt/~mooshak/cgi-bin/execute?command=login&contest=eda2_2017_trab.

O acesso ao Mooshak faz-se através da identificação do grupo e de uma *password*, que serão fornecidas após o envio da constituição do grupo ao docente de EDA2.

As submissões poderão ter uma de três formas:

- Um ficheiro com todo o código do programa e com extensão `.c`. (Não aconselhado.)
- Um arquivo `tar` comprimido, num ficheiro com extensão `.tgz`, contendo somente os ficheiros com o código do programa (com extensão `.c` ou `.h`).

  Neste caso, os ficheiros deverão ser extraídos para a directoria corrente.

  Um arquivo com estas características pode ser criado através de um comando como o seguinte:

  ```
  tar cvzf nome-do-arquivo.tgz *.[ch]
  ```

- Um arquivo `zip` comprimido, num ficheiro com extensão `.zip`, contendo somente os ficheiros com o código do programa (com extensão `.c` ou `.h`).

  Neste caso também, os ficheiros deverão ser extraídos para a directoria corrente.

No concurso, estarão definidos três problemas:

T (Testes)

Este problema serve para testar o programa com os testes disponíveis em:

http://www.di.uevora.pt/~vp/eda2/testes/.

E (Entrega)

Este é o problema em que é feita a entrega do código.

Um trabalho só será aceite se o programa passar todos os testes deste problema.

Se cumprir todos os restantes requisitos, a classificação de um trabalho aceite neste problema poderá ir de 7 a 14 valores.

P (Proficiente)

Neste problema, o programa será executado com testes mais exigentes do que os usados no problema E (Entrega).

Um trabalho só poderá ter uma nota superior a 14 valores se o programa passar todos os testes deste problema.

**Importante** Qualquer mensagem do compilador e qualquer diferença entre o que o programa escreve e o que devia escrever (e tal como descrito na secção anterior) levará à sua não aceitação. Também poderá impedir a sua aceitação os programas terminarem sem ser por `return 0` (ou equivalente).

**Testar o programa** Se o executável que contém o programa se chamar `postas`, podem verificar se a resposta a um teste — por exemplo, o `teste-A-1` — está correcta através do comando:

```
$ ./postas < teste-A-1.in | diff teste-A-1.out -
```

Se o funcionamento do programa estiver de acordo com o enunciado, o comando terminará sem nada ser escrito no terminal, caso contrário, mostrará as diferenças entre a resposta correcta e a dada pelo programa.

Também é possível redireccionar a saída do programa para um ficheiro e, depois, recorrer ao comando `diff` para comparar o ficheiro obtido e o ficheiro com o resultado esperado:

```
$ ./postas < teste-A-1.in > teste-A-1.o-meu-out
$ diff teste-A-1.out teste-A-1.o-meu-out
```

(Em vez do comando `diff`, pode ser usado qualquer comando que permita comparar o conteúdo de dois ficheiros. No Linux, o comando `meld` permite comparar o conteúdo de dois ficheiros de modo mais amigável.)

Deverão ter em atenção que a máquina em que o programa é testado é relativamente lenta. No entanto, se o programa submetido exceder o tempo permitido, deverão tentar corrigi-lo considerando a complexidade dos algoritmos usados e não procurando poupar alguns $\mu$s alterando as instruções que usam.

### 4.4 Relatório

Além do código submetido e aceite pelo Mooshak, deverá ser entregue um relatório, em papel, com:

- a identificação dos elementos do grupo;
- uma descrição pormenorizada das estruturas de dados usadas (bonecos, também conhecidos como diagramas, são algo que dá muito jeito para descrever estruturas de dados);
- a descrição pormenorizada do formato do(s) ficheiro(s) de dados;
- a descrição do funcionamento do programa, nomeadamente no que toca ao funcionamento das várias operações (devem explicar o que acontece, sem apresentar código nem nomes de funções);
- a justificação de todas as escolhas feitas (suportada na complexidade dos algoritmos usados e na análise dos acessos a disco feitos pelo programa);
- o código dos programas, exactamente como submetido no Mooshak;
- a identificação das fontes consultadas para a realização do trabalho.

### 4.5 Considerações finais

A um trabalho aceite pelo Mooshak corresponderá uma nota igual ou superior a 7, desde que acompanhado de um relatório que cumpra os objectivos.

Na classificação do trabalho serão avaliadas a qualidade das estruturas de dados e dos algoritmos utilizados, a qualidade do código (incluindo a sua clareza, a sua legibilidade, e os comentários), a qualidade do relatório e a qualidade do *trabalho* efectuado.

### 4.6 Datas

A data limite de submissão do **programa** no Mooshak é 3ª-feira, dia 5 de Junho de 2018.

A data limite de entrega do **relatório** é 5ª-feira, dia 7 de Junho de 2018, até às 19h00.

A **discussão** do trabalho será realizada em data a combinar.

### 4.7 TOLERÂNCIA ZERO

EDA2 é uma disciplina de TOLERÂNCIA ZERO no que diz respeito a qualquer tipo de fraude, plágio, cópia ou não participação no trabalho entregue.

*Todos* os elementos entregues pelo grupo deverão ser da autoria de *todos* os membros do grupo. Qualquer elemento usado no trabalho que não seja da autoria dos elementos do grupo, deverá estar devidamente assinalado e a sua origem indicada. (Este uso só é admissível em situações *muito pontuais*, como, por exemplo, na escolha de uma função de *hash*.) Também deverá ser assinalado o código que se tenha inspirado em código de autoria que não dos membros do grupo (excepto código fornecido em EDA2).

A inobservância destas regras terá como consequência a anulação do trabalho e a reprovação de todos os envolvidos, elementos do grupo ou não.

Bom Trabalho