Estruturas de Dados e Algoritmos II
Trabalho Prático — 2ª Fase

Departamento de Informática
Universidade de Évora

2019/2020
v1.1

Educação superior

A 2ª Fase do trabalho consiste na implementação de um programa que execute as operações necessárias para o sistema descrito no enunciado da 1ª Fase.

A implementação não tem de obedecer ao desenho feito na 1ª Fase. Se chegarem à conclusão de que ele não era viável, ou de que era possível melhorá-lo, é possível alterá-lo. O relatório a entregar deverá conter a descrição e a justificação das alterações feitas.

1 Dados

Os dados com que o programa deverá trabalhar são os *identificadores* de estudantes e os *códigos* de países.

O código que identifica um estudante consiste numa sequência de 6 caracteres, de entre letras maiúsculas e algarismos decimais, e é único, i.e., só pode haver um estudante com aquele identificador.

O código que identifica um país consiste numa sequência de 2 letras maiúsculas.

A informação deve ser mantida pelo sistema de forma persistente, mesmo quando não está em execução nenhum programa que a manipule.

2 Comandos

O programa deverá ler comandos do seu *standard input* e executar as operações correspondentes.

Cada linha lida corresponde a um comando a executar. Os vários elementos de um comando 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 que o programa deve dar. As respostas do programa devem ser escritas no seu *standard output*.

Os comandos a executar são os seguintes:

1. **Introduzir um novo estudante**

   Os argumentos do comando são o identificador de um estudante e o código de um país. Depois da execução com sucesso do comando, o sistema passa a conter a informação de que o identificador corresponde a um estudante activo no país com o código dado.

   O comando não tem efeito se o identificador corresponder a um estudante já existente.

   Entrada

   ```
   I <identificador> <código>
   ```

Saída

- *Nada, se o identificador não corresponder a nenhum estudante já existente.*
- `+ estudante <identificador> existe`
  Se já existir um estudante com o identificador dado.

Exemplos

```
⇐ I ABCD01 PT
⇐ I ABCD01 PT
↦ + estudante ABCD01 existe
⇐ I ABCD01 ES
↦ + estudante ABCD01 existe
⇐ I ABCD02 ES
⇐ I ABCD02 ES
↦ + estudante ABCD02 existe
```

2. **Remover um identificador**

   O argumento do comando é o identificador do estudante a remover do sistema. Se o identificador corresponder a um estudante activo, o sistema apaga a informação relativa a esse identificador, que poderá voltar a ser usado.

   O comando não tem efeito se o identificador não corresponder a um estudante activo.

   Entrada

   `R <identificador>`

   Saída

   - *Nada, se o identificador corresponder a um estudante activo.*
   - `+ estudante <identificador> inexistente`
     Se não existir um estudante com o identificador dado.
   - `+ estudante <identificador> terminou`
     Se o estudante a que o identificador corresponde já tiver terminado o curso.
   - `+ estudante <identificador> abandonou`
     Se o estudante a que o identificador corresponde tiver abandonado os estudos.

   Exemplos

   ```
   ⇐ R ABCD02
   ⇐ R ABCD02
   ↦ + estudante ABCD02 inexistente
   ⇐ I ABCD02 FR
   ```

3. **Assinalar que um estudante terminou o curso**

   O argumento do comando é o identificador do estudante que terminou o curso. O estudante tem de estar activo e o sistema guarda a informação de que deixa de estar activo e que completou o curso.

   O comando não tem efeito se o identificador não corresponder a um estudante activo.

   Entrada

   `T <identificador>`

   Saída

   - *Nada, se o identificador corresponder a um estudante activo.*
   - `+ estudante <identificador> inexistente`
     Se não existir um estudante com o identificador dado.
   - `+ estudante <identificador> terminou`
     Se o estudante a que o identificador corresponde já tiver terminado o curso.
   - `+ estudante <identificador> abandonou`
     Se o estudante a que o identificador corresponde tiver abandonado os estudos.

Exemplos

```
⇐ T ABCD01
⇐ T ABCD01
↦ + estudante ABCD01 terminou
⇐ R ABCD01
↦ + estudante ABCD01 terminou
⇐ I ABCD01 PT
↦ + estudante ABCD01 existe
⇐ T ABCD03
↦ + estudante ABCD03 inexistente
```

4. **Assinalar o abandono de um estudante**

   O argumento do comando é o identificador do estudante que abandonou o sistema de ensino. O estudante tem de estar activo e o sistema guarda a informação de que deixa de estar activo e que abandonou os estudos.

   O comando não tem efeito se o identificador não corresponder a um estudante activo.

   Entrada

   ```
   A <identificador>
   ```

   Saída

   - *Nada, se o identificador corresponder a um estudante activo.*
   - `+ estudante <identificador> inexistente`
     Se não existir um estudante com o identificador dado.
   - `+ estudante <identificador> terminou`
     Se o estudante a que o identificador corresponde já tiver terminado o curso.
   - `+ estudante <identificador> abandonou`
     Se o estudante a que o identificador corresponde tiver abandonado os estudos.

   Exemplos

```
⇐ A ABCD01
↦ + estudante ABCD01 terminou
⇐ A ABCD02
⇐ A ABCD02
↦ + estudante ABCD02 abandonou
⇐ T ABCD02
↦ + estudante ABCD02 abandonou
⇐ R ABCD02
↦ + estudante ABCD02 abandonou
⇐ I ABCD02 FR
↦ + estudante ABCD02 existe
⇐ A ABCD03
↦ + estudante ABCD03 inexistente
```

5. **Obter os dados de um país**

   O argumento do comando é o código do país cuja informação se pretende obter. Se houver algum estudante, no sistema, associado a esse país, o programa mostra o número total de estudantes que frequentaram ou frequentam o ensino superior nesse país, o número de estudantes activos, o número de estudantes que completaram o curso e o número de estudantes que abandonaram o ensino.

   Este comando não provoca qualquer alteração na informação presente no sistema.

   Entrada

   ```
   P <código>
   ```

Saída

- `+ <código> - correntes: <C>, diplomados: <D>, abandonaram: <A>, total: <T>`
  Onde `<C>` é o número de estudantes activos no país, `<D>` é o número de estudantes no país que completaram o curso, `<A>` é o número de estudantes no país que abandonaram o sistema de ensino, e `<T>` é o número total de estudantes, no sistema, associados ao país dado.
- `+ sem dados sobre <código>`
  Se não existir nenhum estudante associado ao país com o código dado.

Exemplos

```
⇐ P PT
↦ + PT - correntes: 0, diplomados: 1, abandonaram: 0, total: 1
⇐ P FR
↦ + FR - correntes: 0, diplomados: 0, abandonaram: 1, total: 1
⇐ P ES
↦ + sem dados sobre ES
⇐ P XX
↦ + sem dados sobre XX
```

6. **Terminar a execução do programa**

   O comando não tem argumentos. Em resposta ao comando, o programa termina a sua execução.

   Entrada

   ```
   X
   ```

   Saída

   - *Nada.*

   Exemplos

   ```
   ⇐ X
   ```

3 Exemplo

A sequência de todos os comandos contidos nos exemplos da secção anterior constitui a primeira parte de um exemplo de interacção com o programa. Os comandos seguintes, que constituem a segunda parte desse exemplo, devem ser executados numa segunda invocação do programa, depois da primeira ter terminado devido à execução do comando `X`.

Exemplo

```
⇐ I JD8D7K PT
⇐ P PT
↦ + PT - correntes: 1, diplomados: 1, abandonaram: 0, total: 2
⇐ R ABCD02
↦ + estudante ABCD02 abandonou
⇐ A ABCD01
↦ + estudante ABCD01 terminou
⇐ I PTPTPT PT
⇐ P PT
↦ + PT - correntes: 2, diplomados: 1, abandonaram: 0, total: 3
⇐ R JD8D7K
⇐ P PT
↦ + PT - correntes: 1, diplomados: 1, abandonaram: 0, total: 2
⇐ A JD8D7K
↦ + estudante JD8D7K inexistente
```

```
⇐ T JD8D7K
↦ + estudante JD8D7K inexistente
⇐ I JD8D7K PT
⇐ P PT
↦ + PT - correntes: 2, diplomados: 1, abandonaram: 0, total: 3
⇐ A JD8D7K
⇐ T JD8D7K
↦ + estudante JD8D7K abandonou
⇐ P PT
↦ + PT - correntes: 1, diplomados: 1, abandonaram: 1, total: 3
⇐ X
```

4 Realização e entrega

4.1 Realização

O trabalho será realizado em grupos de um ou dois elementos. A 2ª Fase do trabalho deverá ser feita pelos mesmos elementos que realizaram a 1ª Fase. Só poderão entregar a 2ª Fase do trabalho os grupos que tenham entregado a 1ª Fase.

4.2 Aspectos práticos

A linguagem a usar para a implementação do programa é o C.

Por questões práticas, o número de estudantes considerados está limitado a 10 000 000 (dez milhões). A memória central máxima que o sistema poderá usar é de 64 MB e a memória secundária usada não poderá ultrapassar 512 MB.

4.3 Entrega

Os elementos a entregar são um programa, em C, e um relatório.

Programa

O programa será submetido através do Mooshak, uma aplicação que compilará o código e executará e testará os programas criados, no concurso "EDA2 2019 (Trabalho)". O endereço para acesso ao concurso estará disponível na página de EDA2 no moodle.

O Mooshak terminará a execução de um programa assim que ocorrer um erro num dos testes.

O acesso ao Mooshak faz-se através da identificação do grupo e de uma *password*, que serão fornecidas aos grupos que realizaram a 1ª Fase do trabalho.

As submissões poderão ter uma de três formas:

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

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

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

- Um ficheiro com todo o código do programa e com extensão `.c`. (Não aconselhado.)

No concurso, estarão definidos três problemas:

B (Básico)

Este problema usa os testes públicos, que estarão disponíveis em:

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

A classificação de um trabalho aceite neste problema poderá ir até 9 valores.

E (Entrega)

Se cumprir todos os restantes requisitos, a classificação de um trabalho aceite neste problema poderá ir de 7 a 15 valores.

T (Total)

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 15 valores se o programa for aceite no problema E (Entrega) e passar todos os testes deste problema.

A classificação referida acima constitui a componente T2 da nota do trabalho prático.

Importante Qualquer mensagem do compilador e qualquer diferença entre o que o programa escreve e o que devia escrever (e tal como descrito nas secções anteriores), constituem um erro do programa e levarão 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 `higher-ed`, podem verificar se a resposta a um teste — por exemplo, o `teste-A-1` — está correcta através do comando:

```
$ ./higher-ed < 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:

```
$ ./higher-ed < 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.)

Relatório

O relatório da 2ª Fase do trabalho deverá seguir o modelo do relatório da 1ª Fase e incluir:

- a identificação dos elementos do grupo;
- o problema, no Mooshak, em que foi submetido o programa que deverá ser avaliado (que será a última submissão aceite nesse problema);
- a descrição das alterações efectuadas em relação à 1ª Fase do trabalho, e as respectivas justificações;
- a 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, incluindo a dimensão (máxima, no caso de ser variável) de cada ficheiro;

- a descrição do funcionamento das cinco operações pretendidas, incluindo as respectivas complexidades temporais e o número de acessos a disco efectuados para cada uma;
- a descrição do que acontece no início e no fim da execução do programa;
- a justificação de todas as escolhas feitas (suportada na complexidade dos algoritmos usados e na análise dos acessos a disco feitos pelo programa);
- as fontes consultadas durante a realização do trabalho.

  (Uma referência a uma fonte *online* deverá incluir o título, ou o tema, se não tiver título, e a data de consulta. O URL deverá permitir aceder directamente ao elemento consultado.)

O relatório deverá ser entregue, em formato PDF, no moodle.

4.4 Datas

A data limite de submissão do **programa** no Mooshak é 6ª-feira, dia 5 de Junho de 2020.

A data limite de entrega do **relatório** é 2ª-feira, dia 8 de Junho de 2020.

5 Consultoria

Durante a realização da 2ª Fase do trabalho, será prestado um serviço de consultoria, aplicável à identificação de erros nos programas. Este serviço será *pago*.

O preço a pagar pela utilização do serviço variará entre 0 e 0,5 valores, e será tanto mais elevado quanto menor tiver sido o cuidado com que o código foi escrito, por exemplo:

- O código está mal escrito, confuso ou desnecessariamente complicado.
- O programa está deficientemente organizado ou comentado.
- É um erro básico, como usar um vector com k posições para guardar uma *string* com k caracteres.

O preço de cada consulta será sempre comunicado antes de ser cobrado.

Este serviço só se aplica à identificação de erros num programa. Qualquer outro tipo de perguntas, questões ou dúvidas é de resposta gratuita.

O valor máximo que cada grupo pode gastar neste serviço é de 2 valores. Um grupo que atinja esse valor deixa de poder beneficiar do serviço.