Linguagens e Ambientes de Programação (2019/2020) [Eng.Inf. - DI/FCT/UNL]

Enunciado do 2º Projeto Prático (C)

Artur Miguel Dias

Datas

- 01/maio (23:30) - Divulgação da versão provisória do enunciado
- 04/maio (18:00) - Divulgação da versão final do enunciado
- 09/maio (20:00) - Data limite de entrega sem penalização.
- 13/Maio (20:00) - Data e hora limite de entrega com atraso. Um valor de penalização por cada dia de atraso.

Changelog

- 02/maio (22:00) - Para resolver problemas diversos, um dos quais relacionado com a estranha falta duma função no GCC no MinGW, teve de se encontrar um solução radical. Por favor, descarregue versões novas dos ficheiros "Cartography.h" e "Main.c". Além disso, copie a nova versão da função "loadCartography", que precisou de mudar ligeiramente (ganhou asteriscos em dois locais). No ficheiro "Cartography.c", não mudou nada para além da função loadCartography.
- 02/maio - Numerosas pequenas clarificações ou correções feitas na sequência de perguntas respondidas no fórum.
- 01/maio - Os três ficheiros com código foram atualizados.
- 01/maio - Possíveis correções ou melhorias deste enunciado serão assinaladas aqui.

Portugal de lés a lés

Divisão administrativa de Portugal

Como sabe, existe uma divisão territorial de Portugal que serve de base à administração do país. O país está dividido em **distritos e regiões autónomas**. Por sua vez, cada uma destas áreas está dividida em **concelhos**. Por sua vez, cada concelho está dividido em **freguesias**. Atualmente, em Portugal existem 18 distritos, 2 regiões autónomas, 308 concelhos e 3091 freguesias. Para uniformizar, trataremos as regiões autónomas como se fossem distritos.

A reforma administrativa nacional, implementada em 2013, criou numerosas uniões de freguesias e reduziu em cerca 27% o número de freguesias, que passou de 4260 para 3091. Contudo, neste projeto, vamos usar a base de dados pré-2013 porque ela tem mais situações de descontinuidade territorial, o que ajudará a

testar os programas. Um programa que funcione bem com a base de dados antiga, por maioria de razão também funcionará para a base de dados moderna.

A Direção-Geral do Território (DGT) é responsável pela manutenção duma base de dados geográfica chamada "Carta Administrativa Oficial de Portugal" (CAOP). Trata-se da base de dados oficial que regista os nomes e os limites físicos das freguesias, concelhos, distritos e regiões autónomas de Portugal.

Objetivo

O objetivo deste projeto é implementar em ANSI C um interpretador de comandos com diversas operações que calculam resultados úteis a partir dos dados da CAOP.

Para desenvolver o programa, não precisamos da base de dados completa. Usaremos uma parte muito pequena, guardada sob a forma de texto neste ficheiro: "map.txt". Contém exatamente 176 **parcelas de território** [176 blocos de terreno contíguo] localizadas na Madeira, Açores e Continente. Mas atenção, depois do programa ser escrito e testado, a intenção será começar a usar o programa com as milhares de parcelas do território nacional, e talvez também com parcelas das bases de dados de outros países. [Para obter o ficheiro de parcelas, clique no link com o botão direito do rato, escolha "Save link as..." e guarde o ficheiro dentro do seu projeto no Eclipse, ao mesmo nível da diretoria "src". Não copie o conteúdo do ficheiro com copy&paste porque há editores de texto que alteram a representação das mudanças de linha.]

O programa deve começar por carregar todo o ficheiro num vetor de registos (vetor de parcelas). Depois, os comandos introduzidos pelo utilizador consultam esse vetor para calcular estatísticas e gerar outra informação diversa. O ponto de partida para o seu programa já está escrito e disponível aqui: "Cartography.c". Você pode alterar este ficheiro livremente e será responsável pelo seu conteúdo final. Este ficheiro trabalha de forma conjugada com mais dois ficheiros, os quais que não podem ser alterados: "Cartography.h" e "Main.c". [Clique nos três links com o botão direito do rato, escolha "Save link as..." e instale os três ficheiros dentro da diretoria "src" do seu projeto no Eclipse. Depois clique no nome do projeto com o botão da direita e escolha a opção "Refresh".]

Quando o seu grupo terminar a escrita do programa, a entrega será feita através dum concurso do Mooshak que será aberto mais tarde.

Dificuldades da representação das freguesias

A maioria das freguesias de Portugal são constituídas por uma única parcela de território, sem complicações.

Mas também existem freguesias territorialmente descontínuas, ou seja constituídas por diversas parcelas de território. A fronteira dessas freguesias não é simples de representar. Por exemplo, a freguesia de Ermida é constituída por três parcelas distintas (as três pintadas de vermelho). Em Portugal, há algumas dezenas de casos como este.

A mesma questão da descontinuidade reaparece pelo facto da CAOP entrar no pormenor de representar os rochedos grandes no mar. É por isso que, no nosso ficheiro, a freguesia algarvia de Sagres surge representada por 28 parcelas de território. A ilha de Porto Santo, no arquipélago da Madeira, também aparece representada por uma grande quantidade de parcelas.

Existe uma outra complicação distinta na representação das freguesias: os **buracos**. Uma freguesia pode ter buracos ou seja, ter zonas interiores que não lhe pertencem. Veja que a freguesia de a freguesia de Adão

tem um pequeno buraco (vermelho e com a designação "(a)"). Esse buraco corresponde a uma parcela da freguesia vizinha de Vila Fernando.

Nota: os exemplos das freguesias de Ermida e de Adão não aparecem no nosso ficheiro de teste.

Conceitos principais

Eis os conceitos de base usados na CAOP:

- **Cartografia** - É uma **coleção de parcelas**. Todo o território de Portugal é visto como uma coleção de parcelas territoriais.
- **Parcela** - Trata-se duma **área de terreno contígua, que pode ter buracos no interior**. A maioria das freguesias de Portugal são representadas por uma única parcela sem buracos. Mas existem algumas dezenas de casos (como o da freguesia Adão) que não são assim tão simples. De forma geral, uma freguesia pode requerer múltiplas parcelas, algumas delas com buracos. Um aspeto importante é o facto dos nomes das localidades aparecerem nas parcelas - cada parcela tem uma identificação: a freguesia, concelho e distrito a que pertence. Não bastaria usar o nome da freguesia? Não, pois existem freguesias distintas com nome repetido em diferentes concelhos.
- **Anel** - Os anéis servem para representar as fronteiras externas e internas das parcelas. Um anel é um **caminho linear fechado que não se auto-interseta**. É caracterizado por uma sequência **vértices**, representados por coordenadas geográficas, com latitude e longitude. Como o caminho é fechado, considera-se que depois do último vértice se segue o primeiro vértice ("dá a volta").

A base de dados da CAOP não regista diretamente freguesias, mas sim parcelas. Se quisermos conhecer qual o território correspondente a uma freguesia, temos de procurar todas as parcelas dessa freguesia. Todas as parcelas da mesma freguesia têm a mesma identificação (repetida).

Espera-se que o programa seja escrito em inglês. Eis a tradução para inglês dos conceitos apresentados nesta secção:

```
Cartografia -> Cartography
Parcela -> Parcel
Anel -> Ring
Fronteira exterior -> Edge
Buraco -> Hole
```

Pormenores da representação

No caso duma parcela sem buracos, basta um anel para definir a fronteira exterior da parcela. Este é o caso largamente maioritário. Em Portugal são raras as parcelas com buracos.

Mas, no caso duma parcela com buracos, precisamos dum anel suplementar por cada buraco. Neste caso, o conjunto de anéis internos da parcela define a fronteira interior da parcela.

A base de dados do CAOP respeita duas regras importantes que se apresentam abaixo. Podemos aproveitar estas regras no nosso programa, na hipótese delas se revelarem úteis de alguma forma (ainda não sabemos):

1. As parcelas que dizem respeito a uma dada freguesia surgem na base de dados em posições consecutivas. Exemplos: as 28 parcelas da freguesia de Sagres aparecem todas de seguida; todas as parcelas da freguesia de Porto Santo aparecem todas de seguida. É sempre assim.

2. No caso de duas parcelas serem adjacentes no mapa (i.e. terem um pedaço de fronteira comum), a porção de fronteira comum aparece repetida nos dois anéis envolvidos, com rigorosamente os mesmos vértices comuns. Há só duas variações possíveis: esses vértices aparecem pela mesma ordem nos dois anéis envolvidos, ou aparecem por ordens opostas nos dois anéis envolvidos. Repare que basta existir um vértice em comum, para as duas parcelas serem consideradas adjacentes.

Ficheiro

O nosso ficheiro, com os dados da CAOP, tem a seguinte estrutura:

- No ficheiro global, os dados aparecem por esta ordem:
  1. o número de parcelas (na 1ª linha);
  2. a seguir vem uma sucessão de parcelas (várias linhas).

- Por cada parcela, os dados aparecem por esta ordem:
  1. a identificação (uma linha);
  2. o número de buracos (uma linha);
  3. o anel exterior (várias linhas);
  4. uma sucessão de anéis que definem os buracos (um número variável de linhas e pode não haver nenhuma).

- Por cada anel, os dados aparecem por esta ordem:
  1. o número de pares de coordenadas do anel (uma linha);
  2. sucessão de pares de coordenadas (várias linhas).

Todos os elementos do ficheiro têm um contador no início, o que simplifica a leitura. Não temos de lidar com marcas de fim.

No nosso ficheiro, a maioria das parcelas tem zero buracos. Portanto, para essas parcelas aparece apenas: (1) identificação; (2) o número zero; (3) o anel exterior.

Como o ficheiro tem uma estrutura em três níveis, o natural será a leitura ser conduzida por três funções:

- **`readRing`** - lê um anel e, entre outras coisas, a função contém um ciclo que lê os sucessivos pares de coordenadas do anel.
- **`readParcel`** - lê uma parcela e, entre outras coisas, a função contém um ciclo que lê os sucessivos anéis que representam os buracos.
- **`readCartography`** - lê o ficheiro completo e, entre outras coisas, a função contém um ciclo que lê as sucessivas parcelas.

Estas três funções já estão escritas, mas estude-as bem, até porque isso ajuda a perceber a organização dos dados.

Tipos oferecidos no programa de partida

No programa de partida que foi oferecido, definem-se os seguintes tipos: **`Identification`**, **`Coordinates`**, **`Rectangle`**, **`Ring`**, **`Parcel`**, **`Cartography`**.

Estude esses tipos com detalhe e confira que são consistentes com o que se disse nas secções **Conceitos principais** e **Ficheiro**. Fazer este exercício de análise, servirá para ficar a perceber ainda melhor os conceitos e os dados deste projeto.

Convém referir que, nas coordenadas, se assume o ponto de vista dum observador que está a olhar de frente para o Polo Norte:

- a latitude varia de -90 (Polo Sul / baixo) até +90 (Polo Norte / cima)
- a longitude varia de -180 (oeste / esquerda) até +180 (este / direita)

Precisa também de explicação, o campo **`Rectangle boundingBox`** dentro do tipo-registo **`Ring`**. Esse campo é essencial na operação que testa se um ponto ocorre no interior dum anel.

Em princípio, esta operação precisaria de examinar todos os vértices do anel - e isso pode ser demorado porque um anel grande é constituído por dezenas de milhares de vértices. O nosso "truque" consiste em guardar no campo **`boundingBox`** o retângulo mínimo que engloba todos os vértices do anel. Agora, para testar a inclusão dum ponto, se esse ponto estiver fora do retângulo mínimo, podemos concluir imediatamente que o ponto está fora do anel, evitando a ação demorada de examinar todos os vértices. Mas, claro, se o ponto estiver dentro do retângulo mínimo, então já somos obrigados a examinar todos os vértices. O campo **`boundingBox`** também poderá ser útil para acelerar outras operações.

Atenção, compete-lhe a si calcular e preencher a **`boundingBox`** de cada anel, durante o carregamento do ficheiro ou logo a seguir ao carregamento.

Algumas funções oferecidas no programa de partida

De entre as funções oferecidas já programadas, estas duas precisam de explicação especial:

`double haversine(Coordinates c1, Coordinates c2)`

> Esta é uma função clássica da história da Matemática e da Navegação Marítima. Dado um par de coordenadas geográficas, a função determina a distância em km entre os dois pontos sobre o planeta Terra, que é uma esfera aproximada. Repare que a função usa uma constante que representa o raio médio do planeta Terra. Se, por curiosidade, quiser saber mais sobre este interessante assunto veja aqui.

`bool insideRing(Coordinates c, Ring r)`

> Esta função permite testar se o ponto dado por duas coordenadas geográficas ocorre no interior dum anel. Se, por curiosidade, quiser saber mais sobre este interessante assunto veja aqui.
>
> A função **`insideRing`** tende a ser de execução demorada. Para evitar atividade inútil, a função aproveita o campo **`boundingBox`** do tipo **`Ring`**. Os detalhes já foram explicados atrás.
>
> Se você quiser definir uma função **`insideParcel`** para testar se um ponto ocorre numa parcela, como é que faz? Usando a função **`insideRing`** como auxiliar, fica fácil: basta testar se: (1) o ponto ocorre no interior do anel exterior e (2) se não ocorre no interior de nenhum dos buracos.

Comando de compilação

No Eclipse, é preciso configurar o projeto para obter um efeito equivalente ao do seguinte comando de compilação:

```
gcc -std=c11 -o Main Cartography.c Main.c -lm
```

Quem usar o gcc do MinGW no Windows, talvez precise de indicar uma pilha de execução com 256MB (em vez dos 8 MB por omissão):

```
gcc -std=c11 -o Main Cartography.c Main.c -lm -Wl,--stack-size,268435456
```

Quem usar o compilador clang no MacOS, talvez também precise de indicar uma pilha de execução com 256MB (desta vez, em hexadecimal):

```
clang -std=c11 -o Main Cartography.c Main.c -lm -Wl,-stack_size,100000000
```

Requisitos técnicos do programa

1. Produza um ficheiro chamado `Cartography.c`. Nas regras de submissão, a publicar mais tarde, será explicada a forma de submeter no Mooshak.

2. O programa é para escrever em ANSI-C C11 e usando a biblioteca padrão do C.

3. Programe num estilo completamente imperativo, sendo proibido usar recursividade.

4. Se precisar de ordenar algum vetor, use a função de biblioteca **qsort**.

5. O programa precisa de manipular grandes quantidades de pares de coordenadas e algumas operações arriscam-se a ficar demasiado lentas. Tire o melhor partida da bounding box que existe nos anéis para evitar trabalho inútil, quando isso for possível.

6. O programa de partida usa estruturas de dados com tamanho fixo e dimensionadas por excesso. Daqui resultam duas más consequências: (1) limita-se muito o número máximo de parcelas; (2) o desperdício de memória é imenso. Para evitar os dois problemas, considere a hipótese de alocar dinamicamente todos os vetores que ocorrem nos tipos Ring, Parcel, Cartography e, ao mesmo tempo, elimine do programa qualquer utilização das constantes MAX_VERTEXES, MAX_HOLES e MAX_PARCELS. Para permitir isto, é preciso ativar uma versão alternativa dos tipos, adicionando a seguinte linha antes de cada **`#include "Cartography.h"`** (nos dois ficheiros ".c"):

   ```
   #define USE_PTS          true
   ```

7. O programa vai ser avaliado na máquina Linux do Mooshak. Mas como está em causa um programa escrito numa linguagem com inseguranças, será mais seguro se você puder testar o programa num sistema operativo adicional: Windows ou MacOS.

8. O programa deve ser constituído por numerosas funções pequenas e legíveis. Uma função grande e complicada, certamente está a fazer demasiadas coisas e pode ser decomposta em funções mais pequenas que sejam naturais. Exceções apenas para funções que desempenhem uma tarefa claramente uniforme mas extensa, como é o caso do interpretador.

9. O programa deve ser bem indentado, para maximizar a legibilidade. Além disso, a largura do programa não deve exceder as 100 colunas. Considera-se que um TAB ocupa quatro posições (nos browsers os TABS ocupam 8 posições e por isso o que o browser mostra não deve servir de referência).

10. O ficheiro "Cartography.c" tem de incluir logo nas primeiras linhas, um comentário inicial contendo: **o nome e número dos alunos que realizaram o projeto**; indicação de quais as partes do trabalho que foram feitas e das que não foram feitas (para facilitar uma correção sem enganos); ainda possivelmente alertando para alguns aspetos da implementação que possam ser menos óbvios para o avaliador. Também é possível adicionar comentários individuais em algumas funções, mas faça isso só se considerar que o código dessas funções precisa realmente de explicação (numa solução "profissional" todas as funções deviam estar comentadas, mas aqui estamos a tentar aliviar um pouco a "burocracia").

Comandos

São 13 os comandos a implementar, dois dos quais já estão feitos (os comandos "L" e "Z"). Cada comando é representado por uma letra que pode ser escrita pelo utilizador em maiúsculas ou em minúsculas, não importa.

A letra do comando pode ser seguida de até ao máximo de três argumentos numéricos, dependendo do comando.

Quando o programa arranca, executa logo automaticamente o comando "L".

Nos comandos em que é preciso indicar uma parcela, indica-se a posição (o índice) da parcela no vetor. Não seria prático ter de escrever a identificação completa da parcela.

Os comandos mais difíceis de implementar são: "F" e "T".

Eis a descrição dos vários comandos:

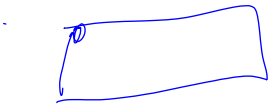
L

Comando **Listar** - *Este comando já foi implementado, para servir de exemplo.* Lista todas as freguesias da base de dados. Mostra, para cada freguesia: (1) a posição no vetor da primeira parcela da freguesia; (1) a identificação da freguesia; (3) o número total de parcelas da freguesia. Segue a ordem do ficheiro da dados.

M *pos*

Comando **Máximo** - Dada uma parcela indicada através duma posição no vetor, considera todas as parcelas da freguesia correspondente e mostra qual dessas parcelas tem mais vértices (considerando todos os anéis - exterior e interiores). Em caso de parcelas empatadas, mostra qualquer uma delas.

X



Comando **eXtremos** - Descobre e mostra quatro parcelas particulares: a parcela mais a norte, a parcela mais a este, a parcela mais a sul e a parcela mais a oeste. Esta é a ordem dos ponteiros do relógio.

R *pos*

Comando **Resumo** - Dada uma parcela indicada através duma posição no vetor, mostra um resumo dessa parcela. Apresentar: identificação, comprimento do anel exterior (inteiro numa nova linha e alinhado com a identificação que aparece por cima), comprimento dos vários buracos (inteiros separados por um espaço), bounding box do anel exterior (delimitada por chavetas). No contexto deste comando, chamamos "comprimento" dum anel ao número de vértices desse anel.

V *lat lon pos* queremos a distância à fronteira!

Comando **Viagem** - Dado um par de coordenadas e dada uma parcela indicada através duma posição no vetor, diz a que distância (em km) as coordenadas estão da fronteira exterior da parcela. Evidentemente, queremos a distância ao vértice mais próximo da fronteira exterior.

Q *pos*

Comando **Quantos** - Dada uma parcela indicada através duma posição no vetor, calcula e mostra: (1) quantas parcelas tem a freguesia correspondente completa; (2) quantas parcelas tem o concelho correspondente completo; (3) quantas parcelas tem o distrito correspondente completo;

C

Comando **Concelhos** - lista os nomes de todos os concelhos, sem repetições e ordenados por ordem crescente. [Hipoteticamente, se existissem dois concelhos com o mesmo nome, esse nome só apereceria uma vez e ficávamos sem saber que havia dois concelhos com o mesmo nome.]

D

Comando **Distritos** - lista os nomes de todos os distritos, sem repetições e ordenados por ordem crescente.

P *lat lon*

Comando **Parcela** - Dado um par de coordenadas, procura e mostra a parcela que contém essas coordenadas. No caso das coordenadas não ficarem dentro de nenhuma parcela, deve ser escrito "FORA DO MAPA".

A *pos*

Comando **Adjacências** - Dada uma parcela indicada através duma posição no vetor, determina e mostra todas as parcelas adjacentes. Atenção, que uma parcela não é considerada adjacente de si própria. Repare que a fronteira (exterior ou interior) das parcelas adjacentes terão pelo menos um vértice comum com a fronteira (exterior ou interior) da parcela indicada (por isso não precisa de testar se o resultado da função haversine é zero para algum par de vértices - seria demasiado lento.)

F *pos1 pos2*

Comando **Fronteiras** - Dadas duas parcelas indicadas através das suas posições no vetor, determina qual é o número mínimo de fronteiras (externas e internas) que é preciso cruzar por um caminho que vá da primeira parcela até à segunda parcela. Escreve apenas um número inteiro. Se as duas parcelas forem a mesma, o resultado mínimo é 0. Se as duas parcelas forem adjacentes o resultado mínimo é 1. Por exemplo, no caso dum caminho entre as freguesias de S.Julião e Verdoejo (que estão no nosso ficheiro de teste) o resultado mínimo é 5. No caso de não existir caminho, deve ser escrito "NAO HA CAMINHO".

T *dist*

Comando **parTiçao** - Efetua uma partição do conjunto de parcelas. Em Matemática, uma partição é uma decomposição dum conjunto em subconjuntos disjuntos. A partição pretendida é feita levando em conta a distância em km indicada (um valor real). Depois mostra a identificação da primeira parcela de cada subconjunto.

Pretende-se uma partição com o máximo de subconjuntos tal que: *para se ir duma qualquer parcela dum subconjunto A para outra qualquer parcela dum subconjunto B distinto, é preciso dar um salto superior à distância indicada.*

Para efeito do cálculo de distâncias, e para evitar uma computação pesadíssima, usa-se apenas primeiro vértice do anel exterior de cada parcela. Atenção, que é mesmo esta a regra que convencionamos usar.

Por exemplo, usando a base de dados nacional completa e a distância 700 km, o comando deverá descobrir automaticamente os seguintes 3 subconjuntos: Açores, Madeira e Continente. (Note que a distância da ilha do Corvo à ilha de Santa Maria é de 600 km; se queremos que os Açores fiquem todos no mesmo subconjunto, precisamos de usar um valor superior a 600 km.) Quando mais pequenos forem os valores, maior será o número de subconjuntos obtidos.

O que deve ser escrito? Devem ser produzidas várias linhas, uma linha por cada subconjunto. Cada linha mostra, por ordem crescente, os índices no vetor das parcelas do subconjunto; as várias linhas aparecem pela ordem crescente do seu primeiro elemento. Além disso, qualquer subsequência de inteiros consecutivos deve ser abreviada escrevendo apenas os dois valores extremos separados por um hífen. Por exemplo, a sequência **`0 3 4 5 6 7 8 9 10 11 15 18 19 21`** deve ser abreviada **`0 3-11 15 18-19 21`**.

Z

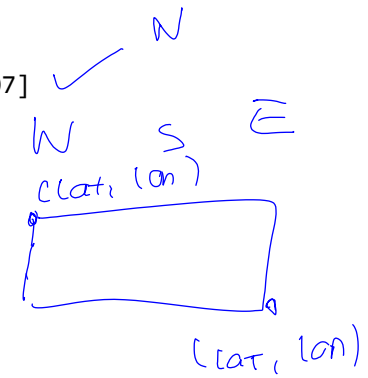
Comando **terminar** - *Este comando já foi implementado.* Termina o programa.

Exemplo de execução

Eis um pequeno extrato duma sessão de trabalho com o interpretador. Apresente os resultados rigorosamente com o formato que se exemplifica abaixo.

```
> L
     ___FREGUESIA___           ___CONCELHO___ ___DISTRITO___
   0 SAGRES                    VILA-DO-BISPO FARO                    [ 28]
  28 CORVO                     CORVO         ARQUIPELAGO-DOS-ACORES  [  1]
  29 ALMAGREIRA                VILA-DO-PORTO ARQUIPELAGO-DOS-ACORES  [  1]
  30 SANTO-ESPIRITO            VILA-DO-PORTO ARQUIPELAGO-DOS-ACORES  [  1]
  31 VILA-DO-PORTO             VILA-DO-PORTO ARQUIPELAGO-DOS-ACORES  [  1]
  32 SAO-PEDRO                 VILA-DO-PORTO ARQUIPELAGO-DOS-ACORES  [  1]
  33 SANTA-BARBARA             VILA-DO-PORTO ARQUIPELAGO-DOS-ACORES  [  1]
  34 PORTO-SANTO               PORTO-SANTO   ARQUIPELAGO-DA-MADEIRA [113]
 147 EVORAMONTE                ESTREMOZ      EVORA                   [  1]
 148 SANTA-MARIA               ESTREMOZ      EVORA                   [  2]
 150 SANTO-ANDRE               ESTREMOZ      EVORA                   [  1]
 151 ARCOS                     ESTREMOZ      EVORA                   [  1]
 152 SAO-BENTO-DO-AMEIXIAL     ESTREMOZ      EVORA                   [  1]
 153 SAO-DOMINGOS-DE-ANA-LOURA ESTREMOZ      EVORA                   [  1]
 154 SAO-LOURENCO-DE-MAMPORCAO ESTREMOZ      EVORA                   [  1]
 155 SANTA-VITORIA-DO-AMEIXIAL ESTREMOZ      EVORA                   [  1]
 156 SAO-BENTO-DE-ANA-LOURA    ESTREMOZ      EVORA                   [  1]
 157 SANTO-ESTEVAO             ESTREMOZ      EVORA                   [  1]
 158 SAO-BENTO-DO-CORTICO      ESTREMOZ      EVORA                   [  1]
 159 VEIROS                    ESTREMOZ      EVORA                   [  1]
 160 SAO-JULIAO                VALENCA       VIANA-DO-CASTELO        [  1]
 161 SILVA                     VALENCA       VIANA-DO-CASTELO        [  1]
 162 FONTOURA                  VALENCA       VIANA-DO-CASTELO        [  1]
 163 CERDAL                    VALENCA       VIANA-DO-CASTELO        [  1]
 164 SAO-PEDRO-DA-TORRE        VALENCA       VIANA-DO-CASTELO        [  1]
 165 TAIAO                     VALENCA       VIANA-DO-CASTELO        [  1]
 166 ARAO                      VALENCA       VIANA-DO-CASTELO        [  1]
 167 CRISTELO-COVO             VALENCA       VIANA-DO-CASTELO        [  1]
 168 GANDRA                    VALENCA       VIANA-DO-CASTELO        [  1]
 169 BOIVAO                    VALENCA       VIANA-DO-CASTELO        [  1]
 170 SANFINS                   VALENCA       VIANA-DO-CASTELO        [  1]
 171 VALENCA                   VALENCA       VIANA-DO-CASTELO        [  1]
 172 GONDOMIL                  VALENCA       VIANA-DO-CASTELO        [  1]
 173 FRIESTAS                  VALENCA       VIANA-DO-CASTELO        [  1]
 174 GANFEI                    VALENCA       VIANA-DO-CASTELO        [  1]
 175 VERDOEJO                  VALENCA       VIANA-DO-CASTELO        [  1]
> M 0
```

```
   27 SAGRES                    VILA-DO-BISPO FARO                    [7807]
> X
  175 VERDOEJO                  VALENCA       VIANA-DO-CASTELO        [N]
  159 VEIROS                    ESTREMOZ      EVORA                   [E]
   38 PORTO-SANTO               PORTO-SANTO   ARQUIPELAGO-DA-MADEIRA  [S]
   28 CORVO                     CORVO         ARQUIPELAGO-DOS-ACORES  [W]
> R 27
  27 SAGRES                    VILA-DO-BISPO FARO
     7807 {37.071275, -8.997116, 36.993499, -8.894870}
> V 38.84158 -7.59059 28
  2018.634102
> V 38.84158 -7.59059 148
  0.000072
> Q 162
  162 FONTOURA                 VALENCA       VIANA-DO-CASTELO        [  1]
  162                          VALENCA       VIANA-DO-CASTELO        [ 16]
  162                                        VIANA-DO-CASTELO        [ 16]
> C
CORVO
ESTREMOZ
PORTO-SANTO
VALENCA
VILA-DO-BISPO
VILA-DO-PORTO
> D
ARQUIPELAGO-DA-MADEIRA
ARQUIPELAGO-DOS-ACORES
EVORA
FARO
VIANA-DO-CASTELO
> P 38.84158 -7.59059
  148 SANTA-MARIA              ESTREMOZ      EVORA
> P  38.8416 -7.5906
  150 SANTO-ANDRE              ESTREMOZ      EVORA
> P  38.82   -7.5906
FORA DO MAPA
> a 0
NAO HA ADJACENCIAS
> a 152
  147 EVORAMONTE               ESTREMOZ      EVORA
  149 SANTA-MARIA              ESTREMOZ      EVORA
  155 SANTA-VITORIA-DO-AMEIXIAL ESTREMOZ     EVORA
  157 SANTO-ESTEVAO            ESTREMOZ      EVORA
> F 160 175
  5
> T 700
  0-27 147-175
  28-33
  34-146
> Z
Fim de execucao! Volte sempre.
```



Regras

- Em condições normais, este projeto seria realizado obrigatoriamente por grupos de dois alunos. Mas, tal deixou de ser obrigatório, considerando a situação especial em que nos encontramos. Também, considerando a situação especial em que nos encontramos, os alunos podem formar grupo com toda a liberdade, independentemente dos turnos em que estejam inscritos. **No caso deste segundo projeto, não existe bónus por se fazer em grupo de dois.**

- Os grupos só podem ser de um aluno ou de dois alunos. Um projeto entregue por três ou mais alunos vale zero valores.

- Será ativado um concurso do Mooshak, que servirá para submeter os trabalhos. Os detalhes da forma de fazer a submissão serão divulgados nessa altura. Até lá preocupe-se apenas em escrever um bom programa.

- Depois do prazo limite ainda se aceitam trabalhos atrasados, mas com penalizações na nota. Mais detalhes nas primeiras linhas deste enunciado.

- Apesar de o projeto ser de grupo, cada aluno, a título individual, tem a responsabilidade de responder por todo o projeto. Assim, é indispensável que os dois membros de cada grupo programem efetivamente. O ideal seria que cada grupo dividisse o projeto em duas partes, e que o desenvolvimento de cada uma das duas partes fosse "comandada" por um aluno distinto, sempre em diálogo com o outro.

- Não há lugar a qualquer pré-inscrição dos grupos. Basta que nos trabalhos submetidos figurem nomes de alunos inscritos na cadeira.

- A nota máxima do projeto é 20 valores.

Avaliação

O docente responsável pela gestão e pela avaliação deste trabalho é Artur Miguel Dias.

A nota do projeto será em grande parte determinada por meios automáticos, através do Mooshak. Portanto é essencial respeitar a especificação contida neste enunciado, em todos os seus detalhes.

Mas, relativamente a programas que funcionem minimamente, também haverá uma apreciação mais subjetiva da qualidade, tendo em conta aspetos, tais como

- organização,
- clareza e simplicidade das ideias programadas,
- bom uso da linguagem, usando o paradigma imperativo, sem usar recursividade;
- legibilidade do código,
- eficiência e capacidade para lidar com milhares de parcelas.

Obviamente não é obrigatório fazer o trabalho todo para obter nota positiva. Mas, claro, vale a pena trabalhar para produzir uma solução bastante completa e com a melhor qualidade possível.

Observações

- Os grupos são incentivados a discutir entre si aspetos gerais da realização do projeto (inclusivamente no fórum). No entanto os grupos devem manter uma certa distância entre si, não podendo partilhar ideias muito detalhadas ou ler código mútuo. A escrita de código exige esforço intelectual, mas só com esforço se consegue evoluir.

- O objetivo deste projeto é levar os alunos a praticar. Um aluno que pratique de forma genuína ganha experiência e provavelmente terá mais facilidade em conseguir aprovação no exame final.

- Cuidado com as fraudes. Por exemplo, se alguém dum grupo oferecer o projeto resolvido a um elemento de outro grupo, trata-se duma fraude envolvendo dois grupos (já tem acontecido). Também se um grupo deixa distraidamente a área aberta e se alguém de outro grupo "rouba" o projeto, então também se considera fraude dos dois grupos (já tem acontecido). Ainda um terceiro caso: se dois grupos se juntam para fazer o projeto conjuntamente e depois o entregam em duplicado, então

também se considera fraude (já tem acontecido). Além destes três exemplos, existem muitas outras situações de fraude. Em suma, **cada grupo é responsável pelo seu projeto, tem de produzir código original**, e não o pode mostrar ou oferecer, direta ou indiretamente, o seu código a outro grupo. Note que é muito melhor ter zero num dos quatro projetos do que ser logo excluído da cadeira por motivo de fraude.

- Poderá haver discussões dos projetos via Skype, mas será normal os docentes entenderem dispensar a maioria dos alunos da discussão.

Final

Bom trabalho! Esperamos que goste.
