TÉCNICO LISBOA

Fundamentos da Programação

Segundo Projeto

22 de Outubro de 2013

Sopa de Letras

Com este projeto pretende-se desenvolver um programa em Python que permita resolver o jogo habitualmente conhecido como *sopa de letras* ou *caça-palavras*.

O objetivo do jogo é encontrar um conjunto de palavras pré-definido numa grelha de letras, segundo qualquer direção (horizontal, vertical ou diagonal).

1 Descrição do problema

O jogo consiste em dois conjuntos de dados: a *lista de palavras* a procurar e a *grelha de letras* onde procurar. Ambos os conjuntos estão guardados num ficheiro de texto.

A partir da *grelha de letras*, o programa deve procurar cada uma das palavras na *lista de palavras*, devolvendo a *lista de coordenadas* e a *direção* em que se encontra cada uma das palavras.

As palavras a encontrar podem estar na horizontal, vertical e diagonal, e devem ser assinaladas na *janela de interação*.

2 Ficheiro do jogo

Os elementos do jogo encontram-se descritos num ficheiro de texto. Este ficheiro tem a seguinte estrutura:

- o nome do puzzle na 1ª linha,
- a *lista de palavras*, numa única linha separadas por vírgulas e um espaço (', '), precedidas da expressão "PALAVRAS: "
- a *grelha de letras*, em que cada linha da grelha se apresenta numa linha do ficheiro.

O texto que se segue é um exemplo de um ficheiro válido - ficheiro `capitais.txt`.

```
Capitais
PALAVRAS: Lisboa, Brasilia, Paris, Londres, ...  Caracas, Toquio, Cairo, Berlim
D T L P S W M M G B G R S F H B D K L D G M A F C V M L N E R G Y P
M Y U E R U F U Q P X O K Q J R R J T Q K X Y G C M M J U J J V Q O
D O L I F I I G X W S M D H C O N A R N S D C E R R F N P E H U R D
T M F I L B F P S X M A D H Y F R X S T D H L N G E F Y U X Q I N V
A V Y L M K H R Q T E S V O B P C L N I R G U E T M F R K X A D P F
N P S S A O B Y G K K P E A J K C V P U L R Q P X O F V T C K I M W
Y X W I I G U J P B P S E R D N O L W T T I B X T S W W F X N T K V
I U E R F Q Y J W A Y P Q M J A R W X G O T A C T C M U U I S L U T
K V R A A G Y Q I Y A F W R Y L M O O B O V M F L O A S P I A E T W
F Y P P F G G Q O W N U J S G Y D H Q V P T O E R V M A R J C B B G
D U F G X J Y Y D X V N Y E C D I U O Y J M G Y P O D C P Z K A E B
Q D N T W T L S W Q E U D R V S W A F T S A B C I P E A V I Q D R S
I F U P G O Y C H T Y W W E N F I E S U W M B N B G D R K B L H L Y
K A B K D Q T F S Y J K J J E C J N N T M D H K B H O A H O P G I S
S K P T O U K I A O B S I L Q I G R E D X T A O Y X Y C V X P J M L
Y M B N Q I V E E G X N I A A G M W K G H J R M K P K E W V X F M C
H C L X T O R L S L V S Y W P P W G C H M U P O N G R G A S V L T T
E K I T K M E J F D B O T P N L H O C U K Y C O B I F R H F F E V V
```

2.1 Grelha de letras

A *grelha de letras* tem um número fixo de linhas e colunas - as *dimensões da grelha*, mas esse número é variável para os diferentes puzzles (ficheiros).

Cada linha da grelha contém uma lista de letras separadas por espaços. Estes espaços devem ser ignorados de modo a encontrar as palavras.

Cada letra está posicionada numa coordenada da forma `(linha, coluna)`, em que *linha* e *coluna* são inteiros entre 0 e o número de linhas menos 1 e de colunas da grelha menos 1, respectivamente.

Por exemplo, a palavra 'Lisboa' aparece invertida na horizontal na coordenada (14, 13), a palavra 'Brasilia' aparece na diagonal a partir da coordenada (0, 15).

2.2 Janela de interação

A *janela de interação* é a janela em que o puzzle será apresentado, e onde serão assinaladas as palavras descobertas.

O programa responsável pela definição da janela é disponibilizado na página da disciplina.

Na Figura 1 encontra-se um exemplo da janela para o ficheiro anterior.

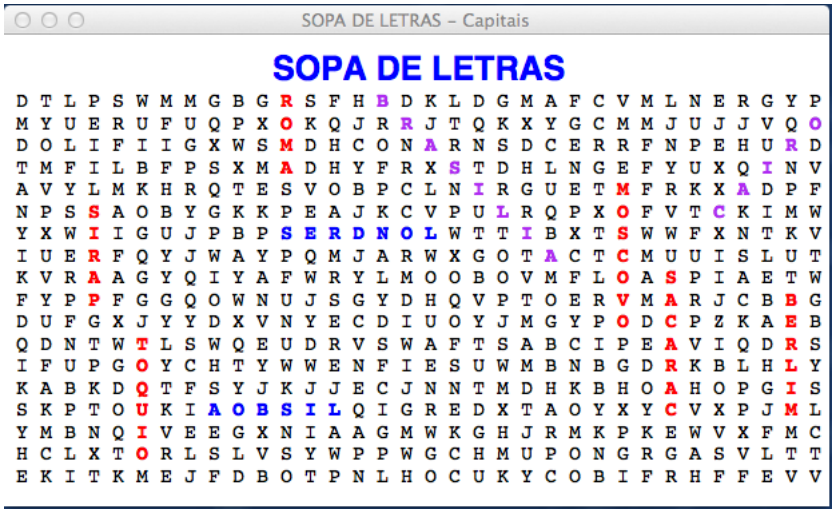

Figura 1: Janela de interação

3 Tipos Abstratos de Informação

Antes de definir a função principal do programa `sopa_letras`, deve definir e implementar o conjunto de tipos abstratos de informação que se seguem.

Note que os tipos implementados no projeto vão ser usados não só pelo programa que vai desenvolver para jogar o `sopa de letras`, como também por outros programas, nomeadamente o programa de avaliação automática. Assim, deve respeitar escrupulosamente as definições que são dadas de seguida, inclusivamente no que diz respeito aos nomes das funções.

Repare que de acordo com a metodologia da abstração de dados, apenas é necessário fazer a validação do tipo dos argumentos nos construtores dos tipos.

3.1 O tipo *direcao* (1 valor)

Os elementos do tipo *direcao* são *'N'*, *'S'*, *'E'*, *'W'*, *'NE'*, *'NW'*, *'SE' e 'SW'*. O tipo *direcao* deve disponibilizar as seguintes operações básicas:

1. *Reconhecedores*:
   - `e_direcao` : $universal \rightarrow l\acute{o}gico$
     `e_direcao`(arg) tem o valor $verdadeiro$ se o arg for do tipo *direcao* e *falso* caso contrário.
   - `e_norte` : *direcao* $\rightarrow l\acute{o}gico$
     `e_norte`(arg) tem o valor $verdadeiro$ se o arg for o elemento $'N'$ e *falso* caso contrário.
   - `e_sul` : *direcao* $\rightarrow l\acute{o}gico$
     `e_sul`(arg) tem o valor $verdadeiro$ se o arg for o elemento $'S'$ e *falso* caso contrário.

- `e_leste` : *direcao* $\rightarrow$ *lógico*
  `e_leste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'E'$ e *falso* caso contrário.
- `e_oeste` : *direcao* $\rightarrow$ *lógico*
  `e_oeste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'W'$ e *falso* caso contrário.
- `e_nordeste` : *direcao* $\rightarrow$ *lógico*
  `e_nordeste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'NE'$ e *falso* caso contrário.
- `e_noroeste` : *direcao* $\rightarrow$ *lógico*
  `e_noroeste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'NW'$ e *falso* caso contrário.
- `e_sudeste` : *direcao* $\rightarrow$ *lógico*
  `e_sudeste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'SE'$ e *falso* caso contrário.
- `e_sudoeste` : *direcao* $\rightarrow$ *lógico*
  `e_sudoeste`(arg) tem o valor *verdadeiro* se o arg for o elemento $'SW'$ e *falso* caso contrário.

2. *Testes*:

   - `direcoes_iguais` : *direcao* $\times$ *direcao* $\rightarrow$ *lógico*
     `direcoes_iguais`(d_1, d_2) devolve o valor *verdadeiro* se as direções d_1 e d_2 forem iguais e *falso* caso contrário.

3. *Outras operações*:

   - `direcao_oposta` : *direcao* $\rightarrow$ *direcao*
     `direcao_oposta`(d) devolve a direção oposta de d de acordo com a rosa dos ventos.
     Por exemplo:

     ```
     >>> direcao_oposta('N')
     'S'
     ```

3.2 O tipo *coordenada* (2 valores)

Uma *coordenada* é um triplo constituído por dois inteiros positivos e uma *direcao*. Os dois números indicam uma posicao na grelha, em que o primeiro número corresponde à linha da grelha e o segundo à coluna. A *direcao* indica a direção em que uma palavra se encontra na grelha, do seu início para o fim. O tipo *coordenada* deve disponibilizar as seguintes operações básicas:

1. *Construtor*:

   - `coordenada` : $N_0 \times N_0 \times$ *direcao* $\rightarrow$ *coordenada*
     `coordenada`(l, c, d) tem como valor a coordenada referente à posição (l,c) e direção d.
     Em caso de erro, o construtor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
     `coordenada: argumentos invalidos`

2. *Seletores*:

   - `coord_linha` : *coordenada* $\mapsto \mathbb{N}_0$
     `coord_linha`(c) tem como valor a linha da coordenada.
   - `coord_coluna` : *coordenada* $\mapsto \mathbb{N}_0$
     `coord_coluna`(c) tem como valor a coluna da coordenada.
   - `coord_direcao` : *coordenada* $\mapsto$ *direcao*
     `coord_direcao`(c) tem como valor a direção da coordenada.

3. *Reconhecedor*:

   - `e_coordenada` : $universal \rightarrow l\acute{o}gico$
     `e_coordenada`(arg) tem o valor $verdadeiro$ se o arg for do tipo *coordenada* e *falso* caso contrário.

4. *Testes*:

   - `coordenadas_iguais` : *coordenada* $\times$ *coordenada* $\rightarrow$ *lógico*
     `coordenadas_iguais`(c_1, c_2) devolve o valor $verdadeiro$ se as coordenadas c_1 e c_2 forem iguais e *falso* caso contrário.

5. *Outras operações*:

   - `coordenada_string` : *coordenada* $\rightarrow$ *string*
     `coordenada_string`(c) devolve a representação externa de c: uma cadeia de caracteres iniciada por parêntesis esquerdo $'('$ seguido pelo número da linha e da coluna, separados por vírgula e um espaço $','$, seguido por parêntesis direito e traço $')-'$, após os quais se apresenta a direção.
     Por exemplo,

     ```
     >>> coordenada_string(coordenada(5, 29, 'NE'))
     '(5, 29)-NE'
     ```

     que representa a coordenada da palavra `Cairo` na grelha.

3.3 O tipo *grelha* (4 valores)

Uma *grelha* é uma matriz `m`x`n`, com `m` o número de linhas e `n` o número de colunas. O tipo *grelha* deve disponibilizar as seguintes operações básicas:

1. *Construtor*:

   - `grelha` : *lista de strings* $\rightarrow$ *grelha*
     `grelha`(lst) tem como valor uma grelha mxn, em que m é o número de elementos da lista *lst* e n o número de caracteres de cada string na lista.
     A lista não pode ser vazia, e todas as strings devem ter o mesmo comprimento.
     Em caso de erro, o construtor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
     `grelha: argumentos invalidos`

2. *Seletores*:

- `grelha_nr_linhas` : *grelha* $\mapsto \mathbb{N}_0$
  `grelha_nr_linhas`(g) devolve o número de linhas da grelha g.
- `grelha_nr_colunas` : *grelha* $\mapsto \mathbb{N}_0$
  `grelha_nr_colunas`(g) devolve o número de colunas da grelha g.
- `grelha_elemento` : *grelha* $\times \mathbb{N}_0 \times \mathbb{N}_0 \mapsto$ *carácter*
  `grelha_elemento`(g, l, c) devolve o carácter que está na posição (l, c) da grelha g.
  No caso de (l, c) não ser uma posição válida para a grelha, o seletor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
  `grelha_elemento: argumentos invalidos`

  Seja *pos*=(l, c), *pos* é válida para a grelha g se e só se, l é um número entre 0 e o número de linhas da grelha g menos 1 e c é um número entre 0 e o número de colunas da mesma grelha menos 1.

- `grelha_linha` : *grelha* $\times$ *coordenada* $\mapsto$ *string*
  `grelha_linha`(g, c) devolve a cadeia de caracteres que corresponde à linha definida segundo a direção dada pela coordenada c, e que inclui a posição dada pela mesma coordenada.
  No caso de c não definir uma posição válida para a grelha g o seletor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
  `grelha_linha: argumentos invalidos`.

  Por exemplo, se *grelha* tiver a grelha obtida pela leitura do ficheiro *capitais.txt*,

```
 >>>grelha_linha(grelha, coordenada(6, 0, 'E'))
"YXWIIGUJPBPSERDNOLWTTIBXTSWWFXNTKV"
 >>>grelha_linha(grelha, coordenada(0, 3, 'S'))
"PEIILSIRAPGTPKTNXT"
 >>>grelha_linha(grelha, coordenada(0, 15, 'SE'))
"BRASILIAFROERHXXLV"
 >>>grelha_linha(grelha, coordenada(1, 33, 'SW'))
"ORIACFUAVPCBDXGCO"
```

3. *Reconhecedor*:

   - `e_grelha` : *universal* $\rightarrow$ *lógico*
     `e_grelha`(arg) tem o valor *verdadeiro* se o *arg* for do tipo *grelha* e *falso* caso contrário.

4. *Testes*:

   - `grelhas_iguais` : *grelha* $\times$ *grelha* $\rightarrow$ *lógico*
     `grelhas_iguais`(g_1, g_2) devolve o valor *verdadeiro* se as grelhas forem iguais e *falso* caso contrário.

3.4 O tipo *resposta* (2 valores)

Uma *resposta* é uma lista de tuplos, em que cada tuplo contém uma palavra no primeiro elemento e uma coordenada no segundo. O tipo *resposta* deve disponibilizar as seguintes operações básicas:

1. *Construtor*:

   - `resposta` : *lista de tuplos*$(string,\ coordenada) \rightarrow resposta$
     `resposta`(lst) tem como valor a $resposta$ que contém cada um dos tuplos que compõem a lista lst.
     Em caso de erro, o construtor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
     `resposta: argumentos invalidos`

2. *Selectores*:

   - `resposta_elemento` : $resposta \times \mathbb{N}_0 \mapsto tuplo(string,\ coordenada)$
     `resposta_elemento`(res, n) devolve o enésimo elemento da resposta res. Caso n seja menor que 0 ou maior ou igual que o número de elementos da resposta erro, o seletor deve gerar um erro de valor (`ValueError`) com a seguinte mensagem:
     `resposta_elemento: argumentos invalidos`
   - `resposta_tamanho` : $resposta \mapsto \mathbb{N}_0$
     `resposta_tamanho`(res) devolve o número de elementos da resposta res.

3. *Modificador*:

   - `acrescenta_elemento` : $resposta \times string \times coordenada \mapsto resposta$
     `acrescenta_elemento`(r, s, c) devolve a resposta r com mais um elemento - o tuplo (s, c).

4. *Reconhecedor*:

   - `e_resposta` : $universal \rightarrow l\acute{o}gico$
     `e_resposta`(arg) tem o valor $verdadeiro$ se o arg for do tipo *resposta* e *falso* caso contrário.

5. *Testes*:

   - `respostas_iguais` : *resposta* $\times$ *resposta* $\rightarrow$ *lógico*
     `respostas_iguais`(r_1, r_2) devolve o valor $verdadeiro$ se as respostas r_1 e r_2 contiverem os mesmos tuplos e *falso* caso contrário.

6. *Outras operações*:

   - `resposta_string` : *resposta* $\rightarrow$ *string*
     `resposta_string`(res) devolve a representação externa da resposta res: uma cadeia de caracteres iniciada pelo parêntesis recto esquerdo `'['` e que contém a descrição de cada elemento da resposta separados por vírgulas e espaço `', '`, terminando com o parêntesis recto direito `']'`.
     Cada elemento é representado por: `'<'`PALAVRA`':'`COORDENADA`'>'`, em que PALAVRA é a palavra encontrada, e COORDENADA a coordenada onde se encontra a palavra.
     Os elementos estão necessariamente ordenados por ordem alfabética das palavras.

     Por exemplo,

```
>>> resposta_string(sopa_letras('capitais.txt'))
"[<BERLIM:(9, 32)-S>, <BRASILIA:(0, 15)-SE>,
<CAIRO:(5, 29)-NE>, <CARACAS:(14, 27)-N>,
<LISBOA:(14, 13)-W>, <LONDRES:(6, 17)-W>, <MOSCOVO:(4, 25)-S>,
<PARIS:(9, 3)-N>, <ROMA:(0, 11)-S>, <TOQUIO:(11, 5)-S>]"
```

4 Funções a implementar

O seu programa deverá resolver qualquer puzzle do tipo *sopa de letras* com palavras escondidas na horizontal, vertical e diagonal, a partir de um ficheiro de texto como o descrito anteriormente.

As funções correspondentes aos comandos de interação com a janela estão definidas no ficheiro `janela_sopa_letras.pyc`. Este ficheiro deve ser colocado na mesma diretoria que o ficheiro com o programa que está a desenvolver, e o seu programa deverá invocar os comandos da seguinte forma:

```
from janela_sopa_letras import *

janela = janela_sopa_letras(nome_ficheiro_jogo)
janela.mostra_palavras(res)
janela.termina_jogo()
```

em que `nome_ficheiro_jogo` é o nome do ficheiro que contém a descrição do puzzle, e `res` corresponde a um elemento do tipo `resposta`.

A partir da leitura do ficheiro que contém o jogo, o seu programa deve encontrar as coordenadas em que se encontra cada uma das palavras a procurar na grelha de letras (linha, coluna e direção, como descrito na secção anterior). Note que as palavras podem apresentar-se na ordem habitual ou invertidas (como no exemplo anterior 'Brasilia' e 'Lisboa', respectivamente).

Para ler o ficheiro, deverá colocar no seu programa as seguintes instruções:

```
fich = open('capitais.txt', 'r')
lst_linhas = fich.readlines()
```

deste modo *lst_linhas* ficará com a lista de linhas lidas do ficheiro.

O seu programa deve conter as seguintes funções:

- (1 valor) `sopa_letras` : *cadeia de caracteres* $\mapsto$ *resposta*
  `sopa_letras`$(fich)$ tem como resultado a resposta ao puzzle descrito no ficheiro *fich*.
- (4 valores) `procura_palavras_numa_direcao` : $grelha \times lista\,de\,strings \times direcao \mapsto$ *resposta*
  `procura_palavras_numa_direcao`$(grelha, palavras, dir)$ tem como resultado

a *resposta* que representa as coordenadas das *palavras* encontradas na *grelha*. Para cada palavra, deve apresentar-se a coordenada do seu primeiro carácter seguindo a direção em que se encontra a palavra. Por exemplo, se g e p forem a grelha e as palavras obtidas pela leitura do ficheiro $capitais.txt$,

```
>>> resposta_string(procura_palavras_numa_direcao(g, p, 'E'))
"[<LISBOA:(14 ,13)-W>, <LONDRES:(6 ,17)-W>]"
```

4.1 Sugestões

Comece por implementar os tipos abstratos de informação, pela ordem apresentada. Depois de implementar cada um deles teste-os isoladamente, se algum apresentar erros será mais fácil detetá-los.

Assim que os seus tipos estiverem corretos, deve criar as funções que separam a grelha da lista de palavras a procurar. Só depois destas não apresentarem erros, deve iniciar a implementação das funções `sopa_letras` e `procura_palavras_numa_direcao`.

Note que a função `procura_palavras_numa_direcao` é genérica, no sentido em que tem de devolver a resposta para qualquer direção fornecida. No entanto, e de acordo com o princípio da abstração procedimental, nada o impede de dividir o problema em sub-problemas mais simples de resolver.

Na página da disciplina estão disponíveis os seguintes ficheiros:

- `cidades` - puzzle de cidades de Portugal, com palavras na horizontal
- `paises` - puzzle de países com palavras na horizontal e vertical
- `capitais` - puzzle de nomes de capitais, com palavras na horizontal, vertical e diagonal.

Sugere-se que comece por garantir que o seu programa descobre corretamente as palavras no primeiro puzzle. Apenas depois disto deve avançar para o segundo puzzle, e depois de descobrir corretamente todas as palavras deste, avançar para o último.

Por último, as funções de manipulação de cadeias de caracteres devem ser criadas no programa, sem recorrer às funções definidas na classe `str`. Portanto, as funções `split`, `find`, `upper` e `replace` não podem ser invocadas pelo programa.

5 Aspetos a evitar

Os seguintes aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

1. Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá ver a Lei de Murphy mesmo em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que

nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).

2. Não pense que um dos elementos do seu grupo fará o trabalho por todos. Este é um trabalho de grupo e deverá ser feito em estreita colaboração (comunicação e controle) entre os vários elementos do grupo, cada um dos quais com as suas responsabilidades. Tanto uma possível oral como perguntas no segundo teste sobre o projeto, servem para despistar estas situações.
3. Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
4. Não se esqueça que as funções com muitas linhas são penalizadas no que respeita ao estilo de programação.
5. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
6. Quando o programa gerar um erro, preocupe-se em descobrir *qual* a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.

6 Classificação

A avaliação da execução será feita através de um sistema automático para entrega de projetos designado Mooshak. Existem vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efetuar uma nova submissão pelo menos 15 minutos depois da submissão anterior. Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados no enunciado, além de um conjunto de testes adicionais. O facto de um projeto completar com sucesso os exemplos fornecidos no enunciado não implica, pois, que esse projeto esteja totalmente correto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correto.

Não será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. Os ficheiros de teste usados na avaliação do projeto serão disponibilizados na página da disciplina após a data de entrega.

A nota do projeto será baseada nos seguintes aspetos:

1. Execução correta (70%).

   Esta parte da avaliação é feita recorrendo a um programa de avaliação automática que sugere uma nota face aos vários aspetos considerados.
2. Facilidade de leitura, nomeadamente abstração procedimental, nomes bem escolhidos, paragrafação correta, qualidade (e não quantidade) dos comentários e tamanho das funções (25%).
3. Estilo de programação (5%).

7 Condições de realização e prazos

A entrega do 2º projeto será efetuada exclusivamente por via electrónica. Deverá submeter o seu projeto através do sistema Mooshak, até às **23:59** do dia **10 de Dezembro de 2013**. Projetos em atraso não serão aceites seja qual for o pretexto.

Deverá submeter um único ficheiro com extensão .py contendo todo o código do seu projeto. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo.

`Importante:` No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

Duas semanas antes do prazo (isto é, na Sexta-feira, 22 de Novembro), serão publicadas na página da cadeira as instruções necessárias para a submissão do código no Mooshak. Apenas a partir dessa altura será possível a submissão por via electrónica. Nessa altura serão também fornecidas a cada um as necessárias credenciais de acesso. Até ao prazo de entrega poderá efetuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efetuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada. Não serão abertas exceções.

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Projetos iguais, ou muito semelhantes, serão penalizados com a reprovação na disciplina. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.