

Nome do estudante: \_\_\_\_\_ Código: \_\_\_\_\_

**1. [8.0 pontos = 1.0 + 1.5 + 2.0 + 2.5 + 1.0]**

a) [1.0] A função **toUpperStr()** converte todas as letras da *string*, **s**, que recebe como parâmetro, para maiúsculas. O resultado da conversão deve ser guardado na mesma *string*. Complete o protótipo e escreva o código desta função.

```
void toUpperStr( _____ s )
{

}

```

**b) [1.5]** A função **transformLine()** recebe como parâmetro uma *string*, **s**, constituída por letras, espaços e outros caracteres e retorna uma *string* constituída apenas por letras maiúsculas e espaços, substituindo os caracteres que não são letras nem espaços pelo carácter espaço, ' '. Complete o protótipo e escreva o código desta função. A *string* recebida como parâmetro não será modificada. **Nota:** para fazer a conversão para maiúsculas deve usar a função **toUpperStr()**, da alínea anterior.

**Exemplo:** se a *string* **s** for " A very, very short sentence!" a string retornada deverá ser " A VERY VERY SHORT SENTENCE ".

```
string transformLine( _____ s )
{

```

c) [2.0] A função **decomposeLine()** recebe como parâmetro uma linha de texto, **line** (uma **string**), e devolve através de outro parâmetro, **words** (um **vector<string>**), as palavras constituintes dessa linha de texto. Esta função deve começar por invocar **transformLine()** para substituir por espaços todos os caracteres da linha de texto que não sejam letras nem espaços. Complete o protótipo e escreva o código daquela função. **Nota:** pode haver mais do que um espaço entre palavras (ver exemplo da alínea anterior). **Sugestão:** recorra a uma *stringstream* para fazer a decomposição.

**Exemplo:** o vetor resultante da decomposição da *string* do exemplo será {"A", "VERY", "VERY", "SHORT", "SENTENCE"}.

```
void decomposeLine( _____ line, _____ words)
{

}
}
```

**d)** [2.5] Escreva o código de um programa que, usando a função **decomposeLine()**, gera uma lista com todas as palavras contidas no ficheiro "**text.txt**", guardado na pasta "**C:\docs**", e grava as palavras resultantes no ficheiro "**words.txt**", na pasta atual. As palavras resultantes devem ser escritas por ordem alfabética, uma palavra por cada linha do ficheiro. **Notas:** **1)** omita os ficheiros de inclusão; **2)** considere que é sempre possível abrir/ler/escrever os ficheiros com sucesso; **3)** se houver palavras repetidas, elas deverão ser mantidas na lista de palavras resultantes; **4)** não repita o código das funções descritas nas alíneas anteriores; indique apenas onde deveriam ser colocadas, no programa.

**e)** [1.0] Indique a(s) alteração(ões) a introduzir no programa da alínea anterior de modo que, na lista de palavras, não surjam palavras repetidas.

Nome do estudante: \_\_\_\_\_ Código: \_\_\_\_\_

2. [8.0 pontos = 1.0 + 1.5 + 1.5 + 2.0 + 2.0]

O jogo da Batalha Naval é jogado num tabuleiro bidimensional. O jogo consiste em adivinhar as posições em que o jogador adversário colocou a sua "armada", constituída por vários "navios", cada um representado no tabuleiro por um conjunto de células consecutivas, dispostas linearmente, na direção vertical ou na horizontal. Apresenta-se a seguir a declaração parcial de algumas estruturas de dados usadas para implementar esse jogo e uma ilustração da representação interna de um tabuleiro, no qual estão colocados 5 "navios", identificados pelos números 1 a 5; o "mar" é representado pelo valor -1.

```
struct Position {
    int lin, col;
};

// =====

class Ship {
public:
    Ship(unsigned int identifier, char symbol,
         Position position, char direction,
         size_t size);
    unsigned int id() const; //returns identifier
    Position pos() const; //returns position
    char dir() const; //returns direction
    size_t size() const; //returns size
    // ... OTHER METHODS
private:
    unsigned int identifier; // ship id number
    // ... OTHER ATTRIBUTES AND/OR METHODS
};
```

```
class Board {
public:
    Board(size_t numLines = 10, size_t numColumns = 10);
    bool putShip(const Ship &s); //add ship to board, if possible
    // ... OTHER METHODS
private:
    bool canPutShip(Position pos, char dir, size_t size);
    size_t numLines, numColumns;
    vector<vector<int>> board; // each element = ship id or -1
    vector<Ship> ships;
    // ... OTHER ATTRIBUTES AND/OR METHODS
};
```

Possível conteúdo de  
vector<vector<int>> board  
com 7x10 células →

-1 = "mar"  
1,2,...5 = "navio" 1,...,navio" 5

-1	3	-1	-1	-1	-1	-1	-1	-1	-1
-1	3	-1	-1	-1	-1	-1	5	5	-1
-1	3	-1	1	1	-1	-1	-1	-1	-1
-1	3	-1	-1	-1	2	-1	-1	-1	-1
-1	3	-1	-1	-1	2	-1	4	4	4
-1	-1	-1	-1	-1	2	-1	-1	-1	-1
-1	-1	-1	-1	-1	2	-1	-1	-1	-1

a) [1.0] Explique por que é possível fazer a declaração **Board b**; mas não a declaração **Ship s**; .

b) [1.5] Implemente o construtor da classe **Board**, o qual deve preencher todos os elementos do atributo **board** com o valor **-1** que indica que a célula respetiva corresponde a "mar livre" (isto é, sem "navios").

c) [1.5] O método **canPutShip()**, da classe **Board**, determina se o "navio" pode ser colocado na posição **pos** (posição do canto superior esquerdo do "navio") e na direção **dir**. Para isso, é necessário verificar que não ultrapassa os limites do tabuleiro e que todas as células que vai ocupar estão livres (valor = -1). Complete o código abaixo apresentado, nas partes assinaladas. **Nota:** a direção do "navio" é indicada por uma das letras maiúsculas seguintes: **H** (=horizontal) ou **V** (=vertical).

```
bool Board::canPutShip(Position pos, char dir, size_t size) {
    _____ // TO DO
{
    case ____: // TO DO: test if ship with length 'size' can be put at 'pos', in horizontal direction

    case ____: // TO DO, just this line: test if ship ... can be put in vertical direction
        // DONE. This piece of code was correctly implemented but was carelessly erased ...
    }
    return true;
}
```

**d) [2.0]** O método **putShip()** tenta colocar no tabuleiro (modificando os atributos **board** e **ships**) o "navio", **s**, que recebe como parâmetro. A posição onde se pretende colocar o navio é a que for retornada pelo método **pos()** de **Ship**. Escreva o código deste método, o qual retorna **true** ou **false**, consoante tenha sido possível ou não colocar o "navio" no tabuleiro. **Nota:** use o método **canPutShip()**, mesmo que não o tenha implementado, para verificar se o "navio" pode ou não ser colocado no tabuleiro.

```
bool Board::putShip(const Ship &s)
{
```

**e) [2.0]** Escreva um pedaço de código que cria um tabuleiro com 10x20 células, lê do teclado os dados de um "navio" (identificador, símbolo, posição, direção e tamanho) e tenta colocar esse "navio" no tabuleiro. Se isso não for possível, lança um exceção, arremessando a *string* "Can't put ship!". **Nota:** considere que todos os valores lidos para os atributos do "navio" são válidos, isto é, do tipo correto.

Nome do estudante: \_\_\_\_\_ Código: \_\_\_\_\_

3. [4.0 pontos = 1.0 + 1.0 + 1.0 + 1.0]

a) [1.0]

a1) A função

`float average(const int grades[], int numStudents)` faz parte de um programa que permite calcular a média das classificações obtidas pelos estudantes numa prova. Escreva o código da função `average()`. Considere que `numStudents` é sempre um valor maior ou igual que 1.

a2) O número de estudantes que realiza cada prova é variável. Pretende-se que o programa use o espaço estritamente necessário para guardar as classificações. Escreva o pedaço de código que lê do teclado o número de estudantes e as suas classificações (após reservar o espaço necessário para estas), e mostra no ecrã a média das classificações.

**Nota:** considere que o programa tem uma função, já implementada, que lê as classificações:

`void readGrades(int grades[], int numStudents)`

b) [1.0] A STL de C++ disponibiliza uma função `find()` cujo *template* é o seguinte:

```
template <class InputIterator, class T>
```

```
InputIterator find (InputIterator first, InputIterator last, const T& val);
```

em que o parâmetro `val` é o valor a procurar. Diga se é possível usar esta função para procurar um valor numa variável `v`, do tipo `vector<string>`. Em caso afirmativo escreva o pedaço de código que permite procurar o valor "31" em `v`, escrevendo uma mensagem adequada, que indique se o valor foi ou não encontrado. Justifique a sua resposta.

c) [1.0] Um programador queria usar num programa uma estrutura de dados do tipo `set<Position>`, em que `Position` é o tipo de dados declarado no código da pergunta 2, mas verificou que acontecia um erro de compilação quando declarava variáveis deste tipo de `set`. Indique uma possível causa deste problema e aponte uma solução. Não escreva código.

d) [1.0] Considere as classes `Base` e `Derived` bem como a função `main()` de um programa que as usa, abaixo apresentadas.

```
class Base {
public:
    Base(int a) { _a = a; }
    virtual void show() const { cout << _a; }
protected:
    int _a;
};
//=====
class Derived : public Base {
public:
    Derived(int a, int b) : Base(a) { _b = b; }
    void show() const { cout << _a << ' ' << _b; }
private:
    int _b;
};
//=====
int main()
{
    Derived d(2, 3);
    d.show(); cout << endl;
    Base b = d;
    b.show(); cout << endl;
    Base *pb = &d;
    pb->show(); cout << endl;
}
```

Qual a saída do programa?

Qual seria a saída do programa se o qualificativo `virtual` fosse retirado da função `show()` da classe `Base`?

O que aconteceria se o qualificativo `protected` da classe `Base` fosse substituído por `private`? Justifique brevemente a resposta.

FIM