Nome da/do estudante: _____

1. [6.0 valores = 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0]

a) [1.0] Um programador que pretendia desenvolver uma função em C/C++ para contar o número de ocorrências de um valor, **value**, num *array* de inteiros, **a[ ]**, contendo **nelem** elementos, escreveu o seguinte código:

```
unsigned int countValues(int a[], unsigned int nelem, int value)
{
  unsigned int i, count = 0;
  for (i = 1; i <= nelem; i++)
    if a[i] = value
      count++;
  return count;
}
```

O código tem várias incorreções. Assinale-as e corrija-as, junto do código.

b) [1.0] Depois de corrigido o código apresentado na pergunta anterior, o programador pretendia generalizar **countValues()** de forma a poder contar o número de ocorrências de um valor num *array* cujos elementos sejam do tipo unsigned **int**, **float**, ou **double**. Qual a melhor forma de fazer essa generalização? Justifique a resposta e indique como procederia para concretizá-la. Nota: é possível responder a este questão, mesmo que não tenha respondido à anterior.

c) [1.0]

Considere a definição da classe **Person** apresentada ao lado. Explique por que não é possível escrever a seguinte declaração de variáveis num programa que usa objetos dessa classe:

```
Person p1, p2("Phileas Fogg", 47);
```

```
1  class Person
2  {
3  public:
4    Person(string name, unsigned int age);
5    string getName() const;
6    unsigned int getAge() const;
7    // other methods
8  private:
9    string name;
10   unsigned int age;
11 };
```

d) [1.0] Para generalizar a função **countValues**, da pergunta **1.a**, para que seja possível contar o número de ocorrências de objetos de uma classe definida pelo programador (por exemplo, da classe **Person**, definida em **1.c**) num *array* de objetos dessa classe, seria necessário fazer o *overloading* do operador de teste de igualdade para essa classe. Justifique essa necessidade e escreva o código da função que o implementa.
Tendo em conta a implementação que fez do operador, indique se seria necessário fazer alterações à definição da classe **Person** apresentada e, em caso afirmativo, quais as alterações. Nota: se necessário, use a numeração das linhas do código em **1.c** para indicar o local das alterações.

e) [1.0] A classe **Person** pode ser usada como classe base para derivar outras classes, como por exemplo a classe **Player**, usada para representar jogadores de futebol. Além dos atributos da classe **Person**, a classe **Player** tem o atributo **team**, que indica o clube a que pertence o jogador. Escreva a definição da classe **Player** e um construtor dessa classe, de forma a ser possível fazer a seguinte declaração de uma variável, **p1**:

```
Player p1("Cristiano Ronaldo",29,"Real Madrid"); //C.Ronaldo is 29 years old & plays in R.Madrid
```

Definição da classe **Player**, derivada de **Person**:	Construtor da classe **Player:**

f) [1.0] Os jogadores de uma equipa de futebol são, frequentemente, designados de acordo com a posição do campo em que jogam, por exemplo, "guarda-redes", "lateral esquerdo" ou "ponta de lança". Alguns jogadores podem jogar em mais do que uma dessas posições. Defina, em C++, a estrutura de dados que usaria para representar a informação relativa aos jogadores de uma equipa, de forma a ser possível saber eficientemente quais os jogadores que podem jogar numa determinada posição, por exemplo, a "lateral esquerdo". Justifique sucintamente a sua opção.

Nome da/do estudante: _____

2. [5.0 valores = 2.0 + 3.0]

a) [2.0] Implemente, em C++, a função **convertResultList()** com a seguinte especificação. A função recebe 3 vectores de igual tamanho, contendo em cada posição correspondente: o número de um aluno, o seu nome e a sua nota. A função devolve um vector que, em cada elemento, guarda uma string formatada, com a informação de cada aluno, compilada numa só linha. O formato dessa *string* é o seguinte: "**AAAAAAA | BBBBBBBBBBBBBBBBBBB | CC.C**" em que **AAAAAAA** representa o número do aluno, **BBBBBBBBBBBBBBBBBBB** o nome, e **CC.C** a nota.

Exemplo:
Tendo em conta os nomes dos parâmetros da função, indicados no cabeçalho da função,
se number[5]=2013007, name[5]="Pedro Sousa", e grade[5]=16.9,
o elemento de índice 5 do vetor retornado deveria conter a *string* "**2013007 | Pedro Sousa | 16.9**"

```
vector<string> convertResultList(const vector<unsigned int> &number,
                                 const vector<string> &name,
                                 const vector<float> &grade)  {









}
```

b) [3.0] Implemente em C++ a função **mode()** que calcula a moda estatística de uma lista de números. A moda estatística é o número mais frequente da lista; caso haja mais do que um número com a mesma frequência deve ser retornado o número que tiver maior valor. A lista de números é guardada num *array* de inteiros, passado como parâmetro. Nota: se achar conveniente pode usar a função do problema **1.a**, admitindo que está corretamente implementada.

```
int mode(const int number[], unsigned int numElems) {









}
```

Nome da/do estudante: _____

3. [5.0 valores = 2.0 + 2.0 + 1.0]

Considere a seguinte definição, em C++, de uma classe para armazenar a informação de um texto literário:

```
class Text {

public:
  Text(string filename);
  // other methods
  bool existsWord(string word); // 'word' exists in the literary text?
private:
  string title;        // title of the literary text
  string author;       // author of the literary text
  vector<string> text; // contents of the literary text, line by line
};
```

a) [2.0] Implemente o construtor da classe, o qual constrói um objeto a partir de um ficheiro de texto cujo nome recebe como parâmetro. O ficheiro deve conter: na 1ª linha, o título do texto literário; na 2ª linha, o autor; nas restantes linhas, o conteúdo do texto.

```
_____ {









}
```

b) [2.0] Implemente o predicado **existsWord()** que recebe uma palavra como parâmetro e devolve **true** se a palavra existir num dos elementos (título, autor ou conteúdo) do texto e **false** no caso contrário. Considere que não são guardados os sinais de pontuação e que todas as palavras têm um espaço antes e outro depois.

```
_____ existsWord(_____)
{









}
```

c) [1.0] Escreva a função **`main()`** de um programa que lê do teclado uma palavra e o nome de um ficheiro contendo um texto literário e que escreve uma mensagem no ecrã indicando se a palavra existe ou não no título, no nome do autor ou no conteúdo do texto.

```
int main()
{









}
```

Nome da/do estudante: _____

4. [4 valores = 2.0 + 2.0]

Uma agência de aluguer de veículos pretende desenvolver um programa em C++ para gerir a informação relativa aos alugueres que faz: informação sobre os veículos, os clientes e os alugueres efetuados.
A informação a registar sobre cada veículo é: matrícula, marca, modelo e registo de todos os alugueres efetuados.
A informação a registar sobre cada cliente é: número do bilhete de identidade (BI), nome completo.
Cada aluguer de um veículo deve ficar registado pelo número do BI do cliente (**clientBI**) e pelas datas inicial (**dateBegin**) e final (**dateEnd**) do período de aluguer. Para representar esta informação sugere-se a seguinte estrutura:

```
typedef struct {
      string clientBI, dateBegin, dateEnd;
} RentRecord;
```

Uma data deve ser representada numa única *string*, com o formato "AAAA-MM-DD" (exemplo: "2014-06-23"). Note que duas datas neste formato são facilmente comparáveis para determinar a sua ordem temporal; por exemplo, usando operadores de comparação de *strings* é fácil verificar que a data "2014-09-30" é anterior a "2014-10-05".

a) [2.0] Defina as classes **Vehicle** e **Client**, bem como as estruturas de dados, a declarar na função **main()**, adequadas para guardar a informação necessária para a gestão dos alugueres, nomeadamente, veículos, clientes e registo de alugueres efetuados. Cada classe deve ter pelo menos um construtor e os métodos **get** dos seus atributos. O registo dos alugueres de um veículo deve ser feito na classe **Vehicle**. Nota: não devem ser implementados os métodos das classes.

b) [2.0] Tendo em conta a estrutura de dados que usou para representar todos os veículos que a agência possui, escreva uma função, **showAvailableVehicles()**, que tenha como parâmetros uma estrutura de dados desse tipo e uma data (representada numa única *string*, como especificado anteriormente) e que escreva no ecrã a lista dos veículos que estão disponíveis nessa data, indicando a matrícula, marca e modelo de cada um. Nota: um veículo está disponível se a data final do seu último registo de aluguer for anterior à data passada com parâmetro da função.

```
_____________________ showAvailableVehicles (_____________________________________){









}
```

FIM