U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO | 1º ANO
EIC0012| PROGRAMAÇÃO | 2014/2015 – 2º SEMESTRE

EXAME - ÉPOCA NORMAL | 2015-06-30

Com consulta. Duração: 2h15m.

Nome da(o) estudante: _____ **Código:** _____

1. [7.0 valores = 2.5 + 4.5]

Um ficheiro de texto contém as classificações de um exame no formato ilustrado à direita. Cada linha contém o código de um estudante e as classificações que obteve em cada uma das N questões do exame. O valor de N está gravado na primeira linha do ficheiro (10, no exemplo apresentado). Considere que pretende desenvolver um programa para processar um ficheiro deste tipo.

```
10
up20141007 -  2.0 1.0 1.0 1.0 1.5 1.5 1.5 2.0 1.0 1.0
ei12903    -  1.7 1.7 1.5 1.8 1.6 1.7 1.3 1.7 1.9 1.2
up20143001 -  1.7 1.7 1.8 2.0 1.6 1.7 1.8 1.5 1.6 1.7
up20132345 -  0.0 1.1 1.9 2.0 2.0 1.2 1.4 1.9 1.7 1.5
ee12105    -  1.4 1.7 1.5 1.6 1.7 1.0 1.5 1.0 1.5 1.4
...
```

a) [2.5] Escreva o código de uma função que recebe dois parâmetros de entrada, uma *string,* **code_grades**, contendo o código e as classificações de um estudante (o conteúdo de **code_grades** deverá ser semelhante a uma das linhas do ficheiro que estão abaixo da primeira linha), e um inteiro, **num_grades**, contendo o número de questões/ /classificações, e retorna, através dos seus parâmetros de saída, o código do estudante (uma *string*, **code**) e a soma das suas classificações (um *double*, **sum**). Nota: considere que **code_grades** contém sempre uma sequência de valores válidos, no formato ilustrado, e que todos os valores estão separados por, pelo menos, um espaço. Sugestão: use uma *stringstream* para decompor **code_grades** nos seus elementos.

```
void get_code_sum(const string &code_grades, int num_grades, string &code, double &sum) {









}
```

b) [4.5] Escreva um programa completo (omita as diretivas **#include**) que permita processar um ficheiro de texto, com um conteúdo semelhante ao descrito na introdução deste problema, e que produza, como resultado, outro ficheiro de texto contendo a classificação total obtida por cada estudante. O conteúdo do ficheiro resultante deve ser semelhante ao apresentado à direita.

```
up20141007 - 13.5
   ei12903 - 16.1
up20143001 - 17.1
...
```

Requisitos adicionais do programa:
- a "raíz" do nome dos ficheiros de entrada e de saída deve ser perguntada ao utilizador; o nome do ficheiro de entrada é obtido acrescentando **".txt"** à "raíz"; o nome do ficheiro de saída é obtido acrescentando "**_final.txt**" à "raíz" (exemplo: se a "raíz" for "**grades**", o nome do ficheiro de entrada será "**grades.txt**" e o nome do ficheiro de saída será "**grades_final.txt**")
- se a abertura do ficheiro de entrada falhar, o programa deve terminar imediatamente, com código de terminação igual a 1;
- use a função definida na alínea **a)** para obter a classificação total, mas não repita aqui o seu código; indique apenas em que local do programa colocaria o código da função.

Nome da(o) estudante: _____ **Código:** _____

2. [5.0 valores = 2.0 + 1.5 + 1.5]

A seguinte classe foi definida para guardar a informação relevante acerca da classificação final dos estudantes de uma dada unidade curricular. A classificação final depende das classificações obtidas no mini-teste, no projeto e no exame.

```
class Student {
public:
  Student();
  Student(const string &code, const string &name);
  void setGrades(double shortExam, double project, double exam);
  string getCode() const;
  string getName() const;
  int getFinalGrade() const;
  // other get and set methods
  bool isApproved() const; // is the student approved or not ?
  static int weightShortExam, weightProject, weightExam;  // weights in percentage (ex:20,30,50)
private:
  string code; // student code
  string name; // student complete name
  double shortExam, project, exam; // grades obtained by the student in the different components
  int finalGrade;
};
```

a) [2.0] Escreva o código do método **setGrades**. Para além dos atributos indicados nos seus parâmetros, este método também deve atualizar o valor do atributo **finalGrade**, tendo em conta os valores dos parâmetros e a influência dos pesos das componentes **shortExam**, **project**, e **exam**, no valor de **finalGrade**, que são, respetivamente, **weightShortExam, weightProject** e **weightExam**. O valor calculado deve ser arredondado para o inteiro mais próximo; quando a parte decimal for 0.5, **finalGrade** deve ser arredondado para o inteiro superior.

Justifique o uso do qualificativo **static** em **weightShortExam, weightProject** e **weightExam**, e inicialize estas variáveis por forma a conterem os valores 20, 30 e 50, respetivamente.

_____ {

}

Inicialização dos atributos static e justificação para o uso deste qualificativo:

b) [1.5] Escreva um pedaço de código que leia, do teclado, o código, o nome e as classificações de um estudante (no mini-teste, projeto e exame) e que crie um objeto **s** do tipo **Student**, tendo como atributos os valores lidos do teclado. À direita, é apresentado um exemplo de um possível diálogo com o utilizador, durante a execução deste pedaço de código.

Examplo de execução do referido código:

```
Student code? up20141007
Student name? Ana Silva
Short exam grade? 13.5
Project grade? 17
Exam grade? 15.7

(no código, após este diálogo,
  deve ser criado o objeto s)
```

c) [1.5] Considerando que a informação acerca dos estudantes que frequentam uma unidade curricular é guardada num **vector<Student>**, escreva o código da função que recebe como parâmetros uma *stream* de saída e um vetor do tipo referido e que escreve nessa *stream* o nome e a classificação final dos estudantes que foram aprovados. Os nomes e classificações deve surgir alinhados verticalmente; considere que o comprimento máximo de um nome é 50 carateres.

```
void showApproved(ostream &out, const vector<Student> &students)  {









}
```

Nome da(o) estudante: _____ Código: _____

3. [5.0 valores = 1.5 + 2.0 + 1.5]

O Instituto de Meteorologia pretende registar a pluviosidade em algumas localidades, ao longo do ano. Para isso, foi definida uma classe, **Pluviosity**. Apresenta-se a seguir uma definição parcial dessa classe.

```
class Pluviosity {
public:
  Pluviosity();
  Pluviosity(int year);
  bool setPluv(int month, int day, int pluviosity);
  int getPluv(int month, int day) const;
  // other methods, including maxPluv(), to be defined in question 3.b
private:
  unsigned int year;            // year that the pluviosity data refers to
  vector<vector<int> > pluv;  // pluviosity data, indexed by month & day
};
```

a) [1.5] Escreva o código do construtor com parâmetro. Este construtor deve inicializar a pluviosidade de cada dia/mês do ano indicado com o valor zero. Nota: o número de dias de cada mês não é constante, depende do mês e do ano. Considere que lhe é fornecida uma função global que retorna o número de dias de um dado par de valores (**month**, **year**); o protótipo desta função é **int numDaysOfMonth(int month, int year)**, onde **month** deve tomar um valor entre 1 e 12.

```
_____ {




}
```

b) [2.0] Escreva o código do método **maxPluv** da classe **Pluviosity** que retorna o valor máximo da pluviosidade.

```
_____ {




}
```

c) [1.5] Defina a estrutura de dados que usaria para associar a pluviosidade a diferentes localidades e aceder facilmente à pluviosidade numa dada localidade, por exemplo, Porto, Lisboa, Penhas Douradas, etc. . Note que, para cada localidade, será necessário representar a pluviosidade referente a vários anos. Justifique a sua resposta.

definição da estrutura:

justificação:

Nome da(o) estudante: _____ **Código:** _____

4. [3.0 valores = 0.5 + 0.5 + 0.5 + 0.5 + 0.5 + 0.5]

a) [0.5] Uma função, **average()**, que determina a média dos **n** valores consecutivos apontados por **p**, tem o seguinte protótipo: **double average(int *p, unsigned int n)**. Escreva as instruções que permitem fazer a reserva de espaço para 100 valores, antes de invocar a função, estaticamente (em tempo de compilação) e dinamicamente (em tempo de execução). Escreva também a chamada a **average()**, considerando que os valores foram atualizados, entre a reserva de espaço e a chamada à função.

reserva estática de espaço: *(os valores são atualizados, não interessa como)*	reserva dinâmica de espaço: *(os valores são atualizados, não interessa como)*
chamada a average():	chamada a average():

b) [0.5] Os construtores de uma classe não têm valor de retorno. Como é possível informar um programador que aconteceu um erro durante a construção de um objeto (por exemplo, porque um dos parâmetros tem um valor inválido)?

c) [0.5] Quando é "obrigatório" implementar o destrutor de uma classe?

d) [0.5] Por que deve o **operador <<** retornar uma referência para o operando à sua esquerda?

e) [0.5] A Standard Template Library de C++ disponibiliza uma função **sort()** cujo *template* é o seguinte:

```
template <class RandomAccessIterator>
 void sort (RandomAccessIterator first, RandomAccessIterator last);
```

Considere que pretende ordenar por ordem alfabética dos nomes todos os elementos de um **vector<Student>**, tal como foi definido no problema 2.c. Escreva o código do operador que deve ser sobrecarregado (*overloaded*) para a classe **Student**, de modo a que a função **sort()** acima referida possa ser usada para esse fim. Escreva também a chamada a **sort()**, tendo em conta que o nome do vetor é **studProg**.

overloading do operador:

chamada a sort():

f) [0.5] Um desenho é constituído por múltiplos objetos de diferentes tipos: retângulos, triângulos, e círculos. Para representá-los, um programador definiu as classes **Rectangle**, **Triangle** e **Circle**, derivadas de uma classe base, **Shape**. Defina a estrutura de dados que usaria para representar os objetos presentes num desenho, constituído por 2 retângulos, 1 triângulo e 1 círculo.

FIM