U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO | 1º ANO
EIC0012| PROGRAMAÇÃO | 2020/2021 – 2º SEMESTRE

EXAME - ÉPOCA NORMAL | 2020-06-19

Com consulta. Duração: 1h30m

Nome do estudante: _____ **Código UP:** _____

Nota: nesta prova apenas é necessário fazer tratamento de erros e indicar os ficheiros de inclusão quando tal for solicitado explicitamente

1. [4.0]

A função **readInt**, cujo protótipo (incompleto) se apresenta abaixo, tenta ler do teclado um valor inteiro compreendido no intervalo [**inf**..**sup**], fazendo o seguinte:

- Mostra ao utilizador a mensagem **msg** e os valores de **inf** e **sup** que recebe como parâmetros; de seguida, tenta ler um inteiro.
- Se o valor escrito pelo utilizador for válido e estiver compreendido no referido intervalo, deve ser devolvido através do parâmetro **value** e a função retorna o valor **1**.
- Se o valor lido não estiver compreendido no referido intervalo ou se o utilizador escrever um valor inválido, isto é, se escrever letras em vez de um número ou se escrever carateres após o número (mesmo que sejam apenas espaços), a função retorna o valor **0** (zero).
- Se o utilizador teclar CTRL-Z (em Windows) ou CTRL-D (em Linux) a função retorna o valor **-1**.
- Em qualquer dos casos, o *buffer* do teclado deve ficar vazio e a *standard input stream* (**cin**) deve ficar no estado de "não existência de erro".

NOTA: quando o valor retornado pela função for **0** ou **-1**, o valor devolvido através do parâmetro **value** é irrelevante.

Apresenta-se a seguir a função **main** de um programa de teste da função **readInt** e o resultado de uma execução:

```
int readInt(___________ msg, ____ inf, ____ sup, ____ value);

int main()
{
  string msg = "Grade";
  int inf = 0, sup = 20, value, output;
  for (int i = 1; i <= 5; i++)
  {
    output = readInt(msg, inf, sup, value);
    cout << "output = " << output << " - value = " << value << endl;
  }
  return 0;
}
```

Resultado de uma execução do programa:

```
Grade [0..20] ? 21
output = 0 - value = 21
Grade [0..20] ? 19
output = 1 - value = 19
Grade [0..20] ? ABC
output = 0 - value = 0
Grade [0..20] ? 10 euros
output = 0 - value = 10
Grade [0..20] ? ^Z
output = -1 - value = 10
```

Complete o protótipo e escreva o código da função **readInt**.

```
int readInt(___________ msg, ____ inf, ____ sup, ____ value)
{



}
```

```
int readInt(const string& msg, int inf, int sup, int& value)
{
  cout << msg << " [" << inf << ".." << sup << "] ? ";
  cin >> value;
  if (cin.fail())  // se escreveu caráter não numérico ou teclou CTRL-Z
  {
    if (cin.eof())
    {
      cin.clear();
      return -1;
    }
    cin.clear();
    cin.ignore(100000, '\n');  // OU   cin.ignore(numeric_limits<streamsize>::max(),'\n');
    return 0;
  }
  else if (value<inf || value>sup || cin.peek() != '\n')
  {
    cin.ignore(100000, '\n');
    return 0;
  }
  return 1;
}
```

2. [5.0]

Os resultados de um exame foram guardados num ficheiro de texto com o seguinte formato: cada linha do ficheiro contém o código de um estudante e a classificação que obteve, separados por um espaço variável.

Escreva um programa que faça o seguinte:

- Pergunta ao utilizador o nome do ficheiro que contém os resultados do exame e tenta abri-lo, terminando com uma mensagem de erro se não conseguir abrir o ficheiro.
- Se for possível abrir o ficheiro, lê o seu conteúdo, guardando-o num **vector<Student>**; o tipo **Student** está declarado abaixo.
- Ordena o **vector** por ordem decrescente das classificações.
- Mostra no ecrã os resultados do exame, ordenados.

Apresenta-se acima um exemplo do conteúdo de um ficheiro e da saída do programa. O formato de apresentação dos resultados deve ser o ilustrado neste exemplo, com os valores alinhados e separados por '-'; o código de um estudante ocupa no máximo 10 carateres.

NOTA: o ficheiro usado como exemplo (**exam.txt**) está disponível no Moodle.

Conteúdo do ficheiro:

```
up20191007    10
ei1390        19
up201913      11
up20182345    15
ee15105       11
up20171234    15
```

Saída do programa:

```
      ei1390 - 19
up20182345 - 15
up20171234 - 15
  up201913 - 11
   ee15105 - 11
up20191007 - 10
```

```
// #include ... A COMPLETAR

using namespace std;

struct Student {
  string code;
  int grade;
};

// COMPLETAR O PROGRAMA
```

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

struct Student {
  string code;
  int grade;
};

bool sortByGrade(const Student& s1, const Student& s2)
{
  return (s1.grade > s2.grade);
}

int main()
{
  string filename;
  cout << "Name of file with exam results ? "; getline(cin, filename);
  ifstream f(filename);
  if (!f.is_open())
  {
    cerr << "File not found!\n" << endl;
    exit(1);
  }
  Student s;
  vector<Student> students;
  while (f >> s.code >> s.grade) {
    students.push_back(s);
  }
  f.close();
  sort(students.begin(), students.end(), sortByGrade);
  for (auto s : students)
    cout << setw(10) << s.code << " - " << setw(2) << s.grade << endl;
  return 0;
}
```

3. [2.0]

No problema anterior, os códigos dos estudantes não estão escritos num formato uniforme:

- Todos começam por 2 letras,
- mas o ano de matrícula é representado por 2 dígitos nos códigos que começam por '**e**' e por 4 dígitos nos códigos que começam por '**u**',
- e os dígitos que se seguem ao ano, que indicam o número de matrícula nesse ano, podem variar entre 1 e 4 dígitos.

Código original:		*Código formatado:*
up20191007	→	up20191007
ei1390	→	ei20130090
up201913	→	up20190013
up20182345	→	up20182345
ee15105	→	ee20150105
up20171234	→	up20171234

Pretende-se uniformizar o formato dos códigos, de modo a que seja constituído por 10 carateres (**LLAAAANNNN**):

- as 2 letras do código original**, LL**,
- seguidas do ano representado com 4 dígitos, **AAAA** (para isso, nos códigos começados por '**e**' é necessário acrescentar "20" à esquerda dos 2 dígitos que representam o ano),
- e do número de matrícula, também representado com 4 dígitos, **NNNN** (acrescentando zeros à esquerda da representação original, se necessário).

Escreva a função **formatStudentCode** que recebe como parâmetro um código de estudante e devolve, através desse parâmetro, o código no formato **LLAAAANNNN** (ver exemplo acima – os carateres foram coloridos apenas para facilitar a perceção das modificações).

Sabe-se que todos os códigos começam pela letra '**e**' ou pela letra '**u**'.

```
void formatStudentCode(string& code)
{



}
```

```
void formatStudentCode(string& code)
{
  ostringstream ss;
  ss << code.substr(0, 2);
  if (code.at(0) == 'e')
  {
    ss << 20 << code.substr(2, 2); // ano
    ss << setw(4) << setfill('0') << code.substr(4); // n.o de matrícula
  }
  else
  {
    ss << code.substr(2, 4); // ano
    ss << setw(4) << setfill('0') << code.substr(6); // n.o de matrícula
  }
  code = ss.str();
}
```

```
OUTRA SOLUÇÃO POSSÍVEL (há mais!):

void formatStudentCode(string& code)
{
  string aux = code.substr(0, 2), num;     // LL
  if (code.at(0) == 'e')
  {
    aux = aux + "20" + code.substr(2, 2); // AAAA
    num = "000" + code.substr(4);         // aux NNNN
  }
  else
  {
    aux = aux + code.substr(2, 4);        // AAAA
    num = "000" + code.substr(6);         // aux NNNN
  }
  num = num.substr(num.length() - 4);     // NNNN
  code = aux + num;
}
```

4. [3.0]

A classe **TrainStation** é usada num programa de gestão ferroviária para representar uma estação de uma linha de comboio.

```
class TrainStation {
public:
  TrainStation(const string& name="", double latitude=0, double longitude=0); // param.s: name and coord.s of train station
  string getName() const;
  // other 'get' methods
  pair<double, double> getCoordinates() const;    // returns the latitude and longitude
  TrainStation& setName(const string& name);      // modifies the 'name'
  TrainStation& setLatitude(double latitude);     // modifies the 'latitude'
  TrainStation& setLongitude(double longitude);   // modifies the 'longitude'
  // other methods
private:
  string name;               // name of the train stop
  double latitude, longitude; // coordinates of the train stop in the country map
};
```

a) [2.0] Escreva o código do **construtor** e dos métodos **getCoordinates** e **setName**.

construtor:

---

getCoordinates:

---

setName:

construtor:

```
TrainStation::TrainStation(string name, double latitude, double longitude)
{
  this->name = name;
  this->latitude = latitude;
  this->longitude = longitude;
}
```

---

getCoordinates:

```
pair<double, double> TrainStation::getCoordinates() const
{
  return pair<double, double>(latitude, longitude);
}
```

---

setName:

```
TrainStation& TrainStation::setName(const string & name)
{
  this->name = name;
  return *this;
}
```

b) [1.0] Considere as seguintes instruções:

TrainStation t;
t.setName("Porto – S. Bento").setLatitude(41.145666).setLongitude(-8.610734);

Diga se estas instruções são válidas. Justifique brevemente a resposta.

A declaração **TrainStation t;** é válida pois, apesar de não existir construtor sem parâmetros, o construtor com parâmetros define valores por omissão para todos os parâmetros.

A instrução **t.setName("Porto – S. Bento").setLatitude(41.145666).setLongitude(-8.610734);** é válida dado que os métodos **set** retornam uma referência para o objecto modificado; desta forma é possível "encadear" várias chamadas a estes métodos.

5. [4.0]

A classe **TrainLine** é usada no programa referido na pergunta anterior para representar uma linha de comboio. Uma linha é constituída por uma sequência de estações, representadas por objetos do tipo **TrainStation** (ver pergunta anterior).

Os atributos da classe são o nome da linha (ex: **"Porto-Lisboa"**) e uma estrutura de dados que representa as estações da linha.

Para além do(s) construtor(es), a classe tem os seguintes métodos:

- **addTrainStation** – insere uma estação (**TrainStation**) na **n**-ésima posição da linha de comboio;
- **removeTrainStation** – é uma função *overloaded* que permite remover uma estação com base no índice respetivo (um número inteiro que pode variar entre 1 e o número de estações) ou com base no nome da estação; os parâmetros são o índice ou o nome da estação, consoante o caso; em ambos os casos, a função retorna um booleano que indica se a estação existia e foi removida ou se não existia.
- **printLine** – mostra o nome da linha e todas as suas estações.

a) [2.0] Escreva a declaração da classe **TrainLine**. NOTAS: não escreva o código dos métodos; não precisa de inserir comentários.

```
class TrainLine {
public:
  TrainLine(const string& name);
  void addTrainStation(const TrainStation& trainStation, int n);
  bool removeTrainStation(int n);
  bool removeTrainStation(const string& name);
  void printLine() const;
private:
  string name;
  vector <TrainStation> stationList;
};
```

b) [2.0] Escreva o código da função **removeTrainStation** que permite remover uma estação cujo nome recebe como parâmetro. O comportamento da função está descrito acima.

```
bool TrainLine::removeTrainStation(const string& name)
{
  for (auto it = stationList.begin(); it != stationList.end(); it++)
    if (it->getName() == name)
    {
      stationList.erase(it);
      return true;
    }
  return false;
}
```

6. [2.0]

a) [1.0] Explique, numa frase curta, o que faz a seguinte função e reescreva-a de modo a poder processar dados de qualquer tipo numérico (**int**, **float**, **double**, **long int**, …).

```
int mystery(int* a, int n) {
  int v = a[0];
  for (int i = 1; i < n; i++)
    if (a[i] > v)
      v = a[i];
  return v;
}
```

A função determina o maior dos **n** números inteiros, guardados em posições consecutivas de memória, alocado estática ou dinamicamente), a partir do endereço **a**.

```
template <typename T>  //ALTERAÇÕES
T mystery(T* a, int n) {
  T v = a[0];
  for (int i = 1; i < n; i++)
    if (a[i] > v)
      v = a[i];
  return v;
}
```

b) [1.0] Um desenho geométrico é constituído por vários retângulos e círculos, representados por objetos da classe **Rectangle** e **Circle**, respetivamente. Estas classes são derivadas da classe **Shape** cuja declaração se apresenta ao lado.

O uso dos especificadores "**virtual**" e "**=0**" no método **draw** tem um significado e duas consequências importantes. Diga qual o significado e as consequências.

```
class Shape {
public:
  Shape(double xCenter = 0, double yCenter = 0);
  virtual void draw() const = 0;
private:
  int xCenter, yCenter; // center coordinates
};
```

O especificador "**virtual**" significa que a função pode ser "overridden" nas classes derivadas e o especificador "**= 0**" significa que esta função não tem corpo na classe Shape.
O uso conjunto dos 2 especificadores significa que o método **draw** é **puramente virtual**.
Uma consequência é que é que todas as classes derivadas de **Shape** têm de implementar o método **draw**.
Outra consequência é que não é possível instanciar objetos da classe **Shape**.

Para representar os retângulos e círculos que compõem um desenho, um programador declarou a seguinte estrutura de dados:

```
vector<Shape*> v;
```

Diga, justificando, se considera esta estrutura adequada para esse fim e, em caso afirmativo, escreva o código que permite acrescentar um círculo com centro em (1,1) e raio 10 na última posição de **v**.

A estrutura é adequada porque a um apontador para um objeto da classe base pode ser atribuído um apontador para um objeto de uma classe dela derivada. Por isso, o vetor **v** pode guardar apontadores para objetos do tipo **Rectangle** ou do tipo **Circle**.

Para adicionar o referido círculo:

```
v.push_back(new Circle(1, 1, 10)); // ou … Circle(10,1,1) uma vez que não se disse a ordem dos parâmetros do construtor
```

FIM