U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO | 1º ANO
EIC0012| PROGRAMAÇÃO | 2019/2020 – 2º SEMESTRE

EXAME - ÉPOCA NORMAL | 2020-06-15

Com consulta. Duração: 1h30m

Nome do estudante: _____ **Código UP:** _____ **Nº prova:** _____

1. [4.0]

Indique quais das seguintes afirmações são verdadeiras e quais são falsas, assinalando respetivamente com **V** ou **F**. **NOTA** - a pontuação nesta pergunta será dada pela fórmula: **máximo (0, (nº_respostas_certas – nº_respostas_erradas) x 0.2)**	**V / F**
Num programa em **C++** é possível fazer a seguinte definição: `bool private = true;`	F
Dadas as definições: `struct Point { int x, y; };` e `Point *p;` o campo **x** de **p** tanto pode ser identificado por `p->x` como por `*p.x` .	F
Para testar se **x**, do tipo `int`, tem um valor compreendido no intervalo **[-10..10]** pode usar-se a instrução: `if ( -10 <= x <= 10 ) ...`	F
Para escrever no ecrã as primeiras **n** letras do alfabeto (**ABCDE…**, até à **n**-ésima letra) pode usar-se a instrução: `for (int i=0; i<n; i++) std::cout << (char) ('A'+i);`	V
Para guardar a palavra **"Hello"** numa string de **C** é necessário definir um *array* de `char`'s com pelo menos 6 elementos.	V
Sendo `name` uma variável do tipo `std::string`, ao executar a instrução `cin >> name;` se o utilizador escrever no teclado `"Ana-2003 Porto"` o valor atribuído a `name` será `"Ana-2003"`.	V
A função cujo protótipo é `void f(const string &s, int &a)` pode ser invocada com os seguintes argumentos: `f("PROG", 210)` .	F

As funções seguintes podem coexistir num mesmo programa
`void readDate(Date *d);` `// 1.a função`
`void readDate(Date &d);` `// 2.a função`
e sendo **d1** e **d2** variáveis do tipo `Date`, a 1.a função pode ser invocada usando `readDate(&d1);`
e a 2.a função pode ser invocada usando `readDate(d2);` .
	V

A declaração `Queue<Person> supermarketQueue;` indica que `Person` é uma *template class.*	F

A função `f` é uma função recursiva que calcula o valor de x^n, quando `n>=1`:
`int f(int x, int n) {`
`  if (n == 1) return x;`
`  return ( x * f(x, n - 1) );`
`}`
	V

Num bloco de código em que existe a declaração `ofstream f;` é sintaticamente válido executar a instrução `f << "Hello my friends!" << endl;`	V
A instrução `cin.clear();` apaga todos os carateres que estiverem no *buffer* do teclado.	F

A definição
`class Person {`
`  //código omitido`
`};`
cria um novo objeto cujo identificador é **Person**.
	F

Num `std:vector` com 2 dimensões, as linhas do vetor podem ter tamanhos diferentes entre si.	V
Os algoritmos da **STL** que operam sobre contentores da **STL** nunca têm um contentor como parâmetro mas antes *iterators* para o contentor a processar.	V
Os elementos de um `std:set` são únicos e podem ser acedidos pela ordem em que foram inseridos no `set`.	F
É conveniente implementar o destruidor de uma classe sempre que o construtor dessa classe fizer alocação dinâmica de memória.	V

Dadas as definições ao lado,
o objeto **x** é um *function object*
e
a instrução `int y = x(3);`
está sintaticamente correta.
`class ClassA {`
`public:`
`  ClassA(int a, int b);`
`  int operator()(int a);`
`private:`
`  int a, b;`
`};`
`ClassA x;`
	V

Dada a definição
`class B: protected A {`
`  // código omitido`
`};`
os métodos públicos da classe `A` poderão ser invocados sobre objetos da classe `B`.
	F

A instrução `ClassA *p = new ClassB;` é válida se `ClassB` for uma classe derivada de `ClassA`, mas não é válida se `ClassA` for derivada de `ClassB`.	V

Nome do estudante: _____ **Código UP:** _____ **Nº prova:** _____

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

2. [5.0]

a) [1.0] Escreva uma função **genRandom** que gera e retorna um **STL vector** contendo **n** números inteiros, aleatórios, no intervalo [**0**..**m**]; **n** e **m** são parâmetros da função. Defina o tipo de retorno e parâmetros da função **genRandom** como achar apropriado.

```
vector<unsigned int> genRandom(unsigned int n, unsigned int m)
{
  vector<unsigned int> v;
  for (unsigned int i = 0; i < n; i++)
    v.push_back(rand() % (m + 1));
  return v;
}
```

b) [1.5] Escreva uma função **vectorReduce()** que recebe como parâmetros dois vetores, **v1** e **v2**, gerados pela função **genRandom**. A função deve modificar **v2**, removendo deste vetor todos os elementos de **v2** que também pertencem a **v1**, e retornar a contagem do número de elementos de **v2** que foram removidos. Defina o tipo de retorno e os argumentos da função **vectorReduce** como achar apropriado. Nota: esta função não escreve nada no ecrã.

Exemplo numa situação em que **n = 10** e **m = 20**:
```
v1 = {12, 18, 7 , 1, 16, 20, 9, 0, 6, 2}
v2 = {13, 18, 6, 9, 2, 8, 13, 13, 2, 5}
```
Após a invocação de **vectorReduce()**:
```
v2 = {13, 2, 8, 13, 13, 2, 5}
```

```
unsigned int vectorReduce(const vector<unsigned int> &v1, vector<unsigned int> &v2)
{
  unsigned count = 0;
  for (size_t i = 0; i < v1.size(); i++)
    for (size_t j = 0; j < v2.size(); j++)
      if (v1[i] == v2[j])
      {
        v2.erase(v2.begin() + j);
        count++;
      }
  return count;
}
```

c) [1.5] Escreva a função **main** que: lê do teclado os valores de **n** e **m**; gera dois vetores, invocando a função **genRandom**, à qual passa **n** e **m** como argumentos; invoca de seguida **vectorReduce**, usando como argumentos os dois vetores gerados por **genRandom**; por fim, mostra no ecrã o conteúdo dos vetores e o número de elementos removidos de **v2**.
A interface do programa deve seguir o formato apresentado ao lado. Os elementos dos vetores devem aparecer alinhados, como ilustrado; considere que o valor de cada elemento nunca é superior a 100.

Exemplo da interface de exeucução:
```
n? 10
m? 20
12 18  7  1 16 20  9  0  6 12
13  2  8 13 13  2  5
3 elements were removed from 2nd vector
```

```
void showVector(const vector<unsigned int> &v)  // OR ALTERNATIVE BELOW
{
  for (auto x : v)
    cout << setw(2) << x << ' ';
  cout << endl;
}

int main()
{
  unsigned int n, m;
  cout << "n ?"; cin >> n;
  cout << "m ?"; cin >> m;
  vector<unsigned int> v1 = genRandom(n, m);
  vector<unsigned int> v2 = genRandom(n, m);
  unsigned int count = vectorReduce(v1, v2);
  showVector(v1);  // OR for (auto x : v1) cout << setw(2) << x << ' '; cout << endl;
  showVector(v2);  // OR for (auto x : v2) cout << setw(2) << x << ' '; cout << endl;
  cout << count << " elements were removed from 2nd vector\n";
}
```

d) [1.0] Escreva uma função equivalente à função **genRandom** utilizando alocação dinâmica de memória em **C**, isto é, sem recorrer a contentores da **STL**. **Nota**: não escreva a função **main**.

```
unsigned int *genRandomC(unsigned int n, unsigned int m)
{
  unsigned int *v = (unsigned int *) malloc(n*sizeof(unsigned int));
  for (unsigned int i = 0; i < n; i++)
  {
    v[i] = rand() % (m + 1);
  }
  return v;
}
```

Nome do estudante: _____ **Código UP:** _____ **Nº prova:** _____

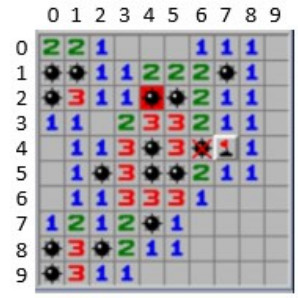
3. [4.0]

No jogo Campo Minado, algumas minas são espalhadas por um tabuleiro rectangular. O ficheiro do jogo especifica o tamanho do tabuleiro na primeira linha (linhas x colunas) e a localização de cada mina, como ilustrado à direita. Este ficheiro é depois importado para a seguinte classe:

```
class Game {
public:
  Game(const string &filename);
  int neighborMines(int line, int col) const;
  // other methods
private:
  vector<vector<char>> mines;
// other attributes
};
```

Ficheiro do jogo, para o tabuleiro ao lado, indicando as dimensões do tabuleiro e a posição das minas:

```
10 x 10
1 , 0
1 , 1
1 , 7
2 , 0
2 , 4
2 , 5
4 , 4
4 , 6
5 , 2
5 , 4
5 , 5
7 , 4
8 , 0
8 , 2
9 , 0
```



a) [2.0] Implemente o construtor de **Game**. O nome do ficheiro é dado como parâmetro e o conteúdo do mesmo deve ser importado para o vetor **mines**. Este vetor deverá ter o tamanho especificado no ficheiro e os seus elementos deverão ter o caráter **'M'** onde existe mina e o caráter espaço nos restantes elementos. Considere que o ficheiro não contém valores errados. Se o ficheiro não existir, o programa deve terminar, com código de terminação igual a 1.

```
Game::Game(const string &filename)
{
  ifstream f(filename);
  if (!f.is_open())
    exit(1);
  char sep;
  int numLines, numCols, lin ,col;
  f >> numLines >> sep >> numCols;
  mines.resize(numLines,vector<char>(numCols,' '));
  while (f >> lin >> sep >> col)
    mines.at(lin).at(col) = 'M';
}
```

b) [2.0] Implemente a função **neighborMines()** que tem como parâmetros as coordenadas (linha e coluna) de um célula e retorna o número de minas existentes nas células vizinhas dessa célula. Se houver uma mina nessa célula, a função deve lançar uma exceção **runtime_error** com a mensagem "**boom**".

Nota: cada célula terá em geral 8 células vizinhas, exceto as células dos bordos que têm um número de vizinhas variável e inferior a 8.

```
int Game::neighborMines(int line, int col) const
{
  if (mines.at(line).at(col) == 'M')
    throw runtime_error("boom");
  int numLines = mines.size(), numCols = mines.at(0).size();
  int i1 = line - 1, i2 = line + 1, j1 = col - 1, j2 = col + 1;
  if (i1 < 0) i1 = 0;
  if (i2 > numLines - 1) i2 = numLines - 1;
  if (j1 < 0) j1 = 0;
  if (j2 > numCols - 1) i2 = numCols - 1;
  int count = 0;
  for (int i = i1; i <= i2; i++)
    for (int j = j1; j <= j2; j++)
      if (mines.at(i).at(j) == 'M') count++;
  return count;
}
```

Nome do estudante: _____ **Código UP:** _____ **Nº prova:** _____

4. [4.0]

As classes **Student** e **Course** são usadas para representar informação sobre estudantes e unidades curriculares que eles frequentam.

```
class Student              // representa um estudante
{
public:
  Student(int id=0, const string &name="");
  int getId() const;
  string getName() const;
  void enrollCourse(Course *course);
  bool setGrade(int courseId, int grade);
  // ... outros métodos
private:
  int id;                  // número de identificação do estudante
  string name;             // nome do estudante
  vector<Course *> courses; // UC's em que está matriculado
  map<int, int> grades;    // notas obtidas nas UC's (chave= courseId)
  // ... outros atributos
};
//---------------------------------------------------------------------
class Course               // representa uma unidade curricular (UC)
{
public:
  Course(int id=0, const string &name="");
  int getId() const;
  string getName() const;
  // ... outros métodos
private:
  int id;
  string name;
  // ... outros atributos
};
```

Em relaçao à classe **Student**,

a) [0.5] Escreva o código do **construtor**.

```
Student::Student(int id, const string
&name)
{
  this->id = id;
  this->name = name;
}


// NOTE: the default values of the parameters can't be present
```

b) [0.5] Escreva o código do método **enrollCourse**, que matricula o estudante no curso **course**, atualizando o conteúdo do atributo **courses**.

```
void Student::enrollCourse(Course *course)
{
  courses.push_back(course);
}
```

c) [2.0] Escreva o código do método **setGrade** que regista a nota obtida pelo estudante numa unidade curricular, fazendo o seguinte:
- verifica, através de **courses**, se o estudante está inscrito no curso cujo identificador é **courseId**; se não estiver retorna **false**;
- verifica se já existe alguma entrada em **grades** que tenha a chave **courseId**; se já existir, retorna **false**;
- se não se verificar nenhuma das situações anteriores, regista a nota em **grades** e retorna **true**.

```
bool Student::setGrade(int courseId, int grade)
{
  bool isEnrolled = false;
  for (auto c:courses)
    if (c->getId() == courseId)
    {
      isEnrolled = true;
      break;
    }
  if (!isEnrolled) return false;
  if (grades.find(courseId) != grades.end()) return false;
  grades.insert(pair<int, int>(courseId, grade));
  //grades[courseId] = grade; // EVEN SIMPLER !!!
}
```

d) [1.0] Escreva um <u>pedaço de código</u> que faz o seguinte:
- cria uma unidade curricular com **id**=1 e **name**="PROG";
- cria uma estudante com **id**=12345 e **name**="Ada";
- matricula a estudante na unidade curricular "PROG";
- atribui à estudante a nota de 20 pontos na unidade curricular de "PROG".

```
Course c1(1, "PROG");
Student s1(12345, "Ada");
s1.enrollCourse(&c1);
s1.setGrade(1, 20);
```

OU

```
Course *c1 = new Course(1, "PROG");
Student *s1 = new Student(12345, "Ada");
s1->enrollCourse(c1);
s1->setGrade(1, 20);
```

Nome do estudante: _____ **Código UP:** _____ **Nº prova:** _____

5. [3.0]

Considere novamente a classe **Student** do problema **4**.

a) [1.0] Defina o protótipo e o código de um método **getAverageGrade**, a acrescentar à classe **Student**, que deverá retornar a classificação média do estudante em todas as unidades curriculares que frequentou, arredondada para o inteiro mais próximo, tendo em conta os dados registados no contentor **grades**.

```
int Student::getAverageGrade() const
{
  float sum = 0;
  for (auto x : grades)
    sum = sum + x.second;
  return (int) (sum / grades.size() + 0.5);
}
```

b) [1.0] Considera correta a escolha do tipo do atributo **courses** da classe **Student** ou acha que seria preferível que tivesse sido definido como **vector<Course> courses** ? Justifique a resposta.

A escolha de um **vector<Course>** teria várias consequências negativas:

- muito mais espaço de memória seria necessário para guardar a informação, pois cada estudante teria uma cópia dos objetos que representam os cursos que frequenta;

- muito mais tempo seria necessário para atualizar a informação, pois sempre que for atualizada a informação relativa a um curso (por exemplo o horário) seria necessário fazer as atualizações em todos os vetores **courses** de todos os estudantes, tornando-se mais provável a existência de informação inconsistente.

c) [1.0] Quando um estudante não obtém aprovação, tem de se matricular novamente na mesma unidade curricular, em ano(s) seguinte(s), até obter aprovação. Diga como alteraria a definição do contentor **grades**, para registar as unidades curriculares (eventualmente repetidas) em que o estudante se matriculou e a(s) classificação(ões) obtidas. Considere que um ano letivo, **ano1/ano2**, é representado apenas pelo valor de **ano1** (exemplo: 2019/2020 é representado apenas por 2019). Justifique brevemente a sua escolha.

Escolheria a seguinte estrutura de dados:

map<CourseId, map<Year, Grade>> grades;
em que CourseId, Year e Grade são tipos de dados inteiros.

Esta estrutura de dados permite que dado um CourseId
se obtenha um map que associa a classificação (Grade) a cada um dos anos (Year)
em que o estudante se matriculou no curso.
Tendo em conta que se usam maps, que são contentores associativos, o acesso à nota obtida num dado curso e num dado ano poderia ser feito de forma eficiente.

FIM