FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

PROGRAMAÇÃO – 2010/2011 - 2º semestre

Exame da Época de Recurso - 2011/07/15

duração: 2h15m; com consulta

NOME DA/DO ESTUDANTE: ______________________________________ **Nº:** EI__________

PARTE 1 [8 valores]

1.
Considere a seguinte função (incompleta), a integrar num programa em que será usada para ler valores inteiros:

```
int getValue(const string &msg, int lim1, int lim2) {
   int value;
   do {
      cout << msg << " [" << lim1 << ".." << lim2 << "] ? ";
      while(! (cin >> value)) {
         cin.clear();
         cin.ignore(1000,'\n');
      }
   } while (______________________________); //enquanto value estiver fora da gama [lim1..lim2]
   return value;
}
```

a) [0.6] Escreva a condição do ciclo **do...while()** de forma a que este ciclo só termine quando o utilizador introduzir um valor (**value**) que pertença ao intervalo **[lim1..lim2]**.

```
while (                                                              );
```

b) [0.5] Indique, de uma forma tão aproximada quanto possível, o conteúdo do ecrã quando esta função for invocada usando a chamada **getValue("temperatura", 10, 50)**, sendo a sequência de valores introduzidos pelo utilizador **o** (o utilizador teclou a letra **o**, em vez do dígito **0**, seguida de <ENTER>) e **0** (a seguir teclou **0** <ENTER>).

_ _

_ _

_ _

c) [0.8] Explique sucintamente o funcionamento do ciclo **while(! (cin >> value)) {...}**, justificando a necessidade das instruções **cin.clear()** e **cin.ignore()**.

d) [0.8] Pretende-se usar esta função num programa que lê e processa as classificações dos estudantes da unidade curricular de Programação. Escreva as instruções necessárias para ler as classificações obtidas pelos estudantes. Considere que: os códigos dos estudantes já foram lidos para o vector **vector<string> code**; as classificações devem ser guardadas no **vector<unsigned int> grade**, que está vazio. A pergunta que deve ser feita ao utilizador é "**Classificacao de <código do estudante> [0..20] ?**" (ex: **Classificacao de ei10999 [0..20] ?**).

vector <string> code; // *vector* ***já preenchido*** *com os códigos dos estudantes*
vector <unsigned int> grade; // *vector (ainda* ***vazio)*** ***a preencher*** *com as classificações dos estudante*
...

e) [0.8] A informação relativa a cada estudante (código e classificação) poderia ser armazenada num único vector, em vez de dois vectores separados. Explique como, fazendo as necessárias declarações de tipos/variáveis.

f) [0.5] Um programador (...?) tentou fazer *overloading* da função **getValue**, anteriormente definida, por forma a poder ler valores com parte inteira e parte decimal, acrescentando ao programa a função **double getValue(const string & msg, int lim1, int lim2)**, mas não conseguiu compilar o programa, obtendo a seguinte mensagem de erro de compilação: *'double getValue(const std::string &,int,int)' : overloaded function differs only by return type from 'unsigned int getValue(const std::string &,int,int)'* . Como explica este erro de compilação? Será possível fazer *overloading* evitando este erro? Em caso afirmativo, como?

g) [0.6] Alguém sugeriu ao programador que poderia ter usado um contentor do tipo **map** ou **multimap** para guardar as classificações dos estudantes. O que acha desta sugestão? Caso a considere acertada, faça a declaração de variável(eis) necessária(s).

2.
Um polígono pode ser descrito pelo conjunto dos seus vértices, sendo cada vértice um ponto representado pelas suas coordenadas (x,y). Considere as seguintes definições de classes de um programa que lida com pontos e polígonos:

```
class Point
{
   friend double dist(const Point& p1, const Point& p2); //calcula a distância entre p1 e p2
public:
   Point(double x, double y);
   double getX() const;   // devolve coordenada x
   double getY() const;   // devolve coordenada y
   // outros métodos da classe Point
private:
   double x, y;  //coordenadas do ponto
};

class Polygon
{
public:
   Polygon();
   Polygon& addVertex(Point p);  // acrescenta vértice ao polígono
   Polygon& addVertex(Point p, size_t pos);  // acrescenta vértice na posição pos
   size_t getSize() const;   // devolve nº de lados do polígono = nº de vértices
   Point getVertex(size_t pos) const; // devolve vértice na posição pos
   double mystery() const;  //  ver alínea c)
   // outros métodos da classe Polygon
private:
   list<Point> v;  // vértices do polígono
};
```

a) [0.6] Escreva o código do construtor da classe **Point**.

b) [0.5] Será possível fazer a seguinte declaração num programa que use a classe `Point`: **`Point p;`** ? Justifique.

c) [0.5] O código da função-membro **mystery** da classe **Polygon** é o seguinte:

```
double Polygon::mystery() const
{
        double t = 0.0;
        for (size_t i=0; i< v.size()-1; i++)
                t = t + dist(getVertex(i), getVertex(i+1));
        t = t + dist(getVertex(v.size()-1), getVertex(0));
        return t;
}
```

Explique numa frase o que faz a função **mystery**.

d) [0.7] Escreva o código da função-membro **getVertex(size_t pos)**, da classe **Polygon**, onde o parâmetro **pos** indica o índice do vértice que a função deve retornar (o 1º vértice tem índice zero).

e) [0.6] Parece-lhe adequada a estrutura de dados escolhida para guardar os vértices do polígono? Justifique.

f) [0.5] Um programador que pretendia criar um triângulo, **t**, com vértices nos pontos **p1**, **p2** e **p3** escreveu o seguinte código:

```
Polygon t;
t.addVertex(p1).addVertex(p2).addVertex(p3);
```

O código está sintacticamente correcto? Justifique a resposta. Caso não esteja, escreva o código correcto.

NOME DA/DO ESTUDANTE: _____ **Nº:** EI_____

PARTE 2 [6 valores]

Numa aplicação informática para bibliotecas é habitual guardar informação sobre a biblioteca e sobre os livros nela guardados. Considere o seguinte exemplo rudimentar de uma dessas aplicações. Considere que a classe **Bibioteca** tem um identificador, que é o nome da biblioteca, e um segundo membro-dado onde guarda os livros da biblioteca. A definição da classe **Biblioteca** é a seguinte:

```
class Biblioteca {
private:
   string nomeBiblioteca;     // guarda o nome da biblioteca
   vector<Livro> livros;      // guarda os livros existentes na biblioteca
public:
   vector<unsigned int> histograma() const; // devolve um histograma - ver alínea c)
   // outros métodos da classe Biblioteca
};
```

A classe **Livro** contém a informação relevante para cada livro guardado na biblioteca e é definida como segue:

```
class Livro {
private:
   string titulo;         // título do livro
   string autor;          // autor do livro
   vector<string> texto;  // linhas de texto que constituem o livro
public:
   Livro(string nomeFicheiro);
   unsigned int numeroPalavras() const;
};
```

a) [2] Implemente, na classe **Livro**, o construtor que lê informação sobre um livro a partir de um ficheiro. Assuma que esse ficheiro é uma sequência de linhas de texto em ASCII. A primeira linha contém o título do livro; a segunda linha o autor do livro e as restantes linhas o texto do livro.

```
Livro::Livro(string nomeFicheiro) {




}
```

b) [2] Implemente, na classe **Livro**, o método **numeroPalavras()** que devolve o número de palavras do texto. Assuma que entre duas palavras existe apenas um espaço e que não há espaços depois da última palavra de cada linha.

```
unsigned int Livro::numeroPalavras() const {









}
```

c) [2] Implemente, na classe **Biblioteca**, o método **histograma()** que retorna um histograma (representado como um vector) da quantidade de palavras dos livros. Na posição zero do histograma é guardado o número de livros que têm menos de 100 palavras, na posição 1, o número dos que têm entre 100 e 199 palavras, e assim sucessivamente. Assuma que só existem livros com menos de mil palavras. O vector histograma é retornado como resultado da função. NOTA: pode utilizar as funções indicadas nas alíneas anteriores, mesmo que não tenha escrito o respectivo código.

```
vector<unsigned int> Biblioteca::histograma() const {









}
```

NOME DA/DO ESTUDANTE: _____ **Nº:** EI_____

PARTE 3 [6 valores]

Uma sequência de DNA é uma série de letras representando a estrutura primária de uma molécula; as letras possíveis são A, C, G e T (exemplo de uma sequência: "AAACAACTTCGTAAGTATA"). Considere que qualquer sequência de DNA tem tamanho fixo (definido pela constante **TAMANHO**) e é guardada numa *string* contendo unicamente as referidas letras.

a) [2] Ocorre uma mutação sempre que na mesma posição de duas sequências houver letras diferentes. Escreva a função **geraMutacoes()**, que recebe uma *string*, representando uma sequência de DNA e o número de mutações a introduzir na sequência e devolve uma sequência com as mutações efectuadas. Para introduzir uma mutação deve gerar aleatoriamente a posição a ser alterada e gerar aleatoriamente qual a nova letra, obrigatoriamente diferente da actual.
Exemplo: um possível resultado da seguinte chamada **geraMutacoes("AAACAACTTCGTAAGTATA",3)** seria a sequência **"ACACAACTGCGTTAGTATA"**; as posições onde ocorreram as mutações são as que estão sublinhadas)

```
string geraMutacoes(string sequencia, size_t numMutacoes) {









}
```

b) [2] Escreva a função **listaMutacoes()**, que recebe duas sequências de DNA e devolve um vector de mutações. O vector de mutações devolvido contém as posições das sequências onde ocorreram as mutações.
Exemplo: se os parâmetros fossem as 2 sequências do exemplo da alínea anterior, o vector resultante da chamada a **listaMutacoes()** deveria ser constituído pelos valores **1**, **8** e **12**.

```
vector<unsigned int> listaMutacoes(string sequencia1, string sequencia2) {









}
```

c) [2] Implemente a função **matrizDeDistancias()** que recebe um conjunto de sequências guardadas num vector e devolve uma matriz de dissimilaridade entre todos os pares de sequências. O elemento **i**,**j** dessa matriz deverá conter o número de mutações entre as sequências **i** e **j**. NOTA: pode utilizar as funções indicadas nas alíneas anteriores, mesmo que não tenha escrito o respectivo código.

Exemplo:

```
seq[0]: AGTCAAATTGCCGATAGCAG
seq[1]: AGTTAATTTGCCGATTGCAG
seq[2]: ACTCAAATTGCCGATAGCAG

Matriz de distancias:
0 3 1
3 0 4
1 4 0
```

```
vector< vector<unsigned int> > matrizDeDistancias(vector<string> &seq) {









}
```

FIM

HLC / JAS / RCS