FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

PROGRAMAÇÃO – 2010/2011 - 2º semestre

Exame da Época Normal - 2011/06/28

duração: 2h15m; com consulta

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

PARTE 1 [8 valores]

1.
Considere o seguinte código e um resultado da interacção com o utilizador, por ocasião da sua execução:

```
int main()
{
   string name;  int age;
   cout << "Nome ? "; cin >> name;
   cout << "Idade ? "; cin >> age;
   cout << "Nome = " << name << "/ Idade = " << age << endl;
   ...
```

Interacção com o utilizador

(o utilizador apenas teve oportunidade de inserir a entrada sublinhada):

```
Nome ? Ana Costa
Idade ? Nome = Ana/ Idade = 92
```

a) [0.7] Como explica o resultado da interacção com o utilizador ?
NOTA: a Idade apresentada assumiu diferentes valores em diferentes execuções do programa, com a mesma entrada.

b) [0.8] Indique a(s) correcção(ões) a efectuar de modo a que o Nome e a Idade sejam lidos correctamente, para as variáveis name e age.

2.
Considere o seguinte código:

```
#include <iostream>

using namespace std;

const size_t MAXELEM = 5;

void mystery1(int v[], size_t& nelem)
{
   cout << "v[] = {";
   for (size_t i=0; i<nelem; i++)
   {
      cout << v[i];
      if (i < nelem-1)
         cout << ", ";
   }
   cout << "}\n";
}
```

➔

```
void mystery2(int v[], size_t& nelem, int value)
{
   size_t i=0;
   while (i<nelem)
   {
      if (v[i] == value)
      {
         for (size_t j=i; j<nelem-1; j++)
            v[j] = v[j+1];
         nelem--;
      }
      i++;
   }
}


int main()
{
   int v[MAXELEM];
   size_t n = MAXELEM;

   for (size_t i=0; i<n; i++) v[i] = i+1;
   mystery1(v,n);
   mystery2(v,n,2);
   mystery1(v,n);
   return 0;
}
```

a) [0.8] Explique o que faz, genericamente, a função `mystery2`.

b) [0.5] Indique, de uma forma tão aproximada quanto possível, a saída do programa.

c) [0.5] O protótipo da função `mystery1` parece-lhe adequado? Justifique a resposta. Caso não o ache adequado, proponha outro que considere mais adequado.

d) [0.5] Indique as alterações a introduzir na definição das funções `mystery1` e `mystery2` por forma convertê-las em funções genéricas, capazes de processar *arrays* contendo elementos de outros tipos numéricos (`long`, `double`, …).

e) [0.8] Tendo em conta as alterações da alínea anterior, é possível que as funções `mystery1` e `mystery2` processem *arrays* cujos elementos sejam de qualquer classe definida pelo utilizador, sem alterar o seu código? Justifique a resposta.

f) [0.7] Considere que, na função `main`, em vez de definir o *array* `v`, definia um vector: `vector<int> v(MAXELEM)`. Reescreva a função `mystery2`, nesta situação, recorrendo, tanto quanto possível, a algoritmos da STL.
NOTA: se achar conveniente, pode alterar o número de parâmetros da função.

3.
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();
   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();
   void addVertex(Point p);  // acrescenta vértice ao polígono
   size_t getSize() const;   // devolve nº de lados do polígono = nº de vértices
   Point getVertex(size_t num) const;
   // outros métodos da classe Polygon
private:
   //A COMPLETAR com estrutura de dados para representar os vértices de um polígono
};
```

a) [0.7] Justifique o uso do qualificativo `friend` na função `dist` da classe `Point` e escreva o código dessa função. Recorda-se que a distância entre 2 pontos, p1($x1$,$y1$) e p2($x2$,$y2$), é dada por: $\sqrt{(x1-x2)^2+(y1-y2)^2}$

b) [0.8] Complete a definição da classe `Polygon` com uma estrutura de dados adequada para representar um polígono. Justifique a sua escolha.

```
class Polygon {...
private:


};
```

Justificação da escolha:

c) [0.5] Escreva o código do método `getSize` da classe `Polygon`.

d) [0.7] Escreva o código necessário para criar um triângulo, `t`, com vértices nos pontos (0,1), (1,1) e (1,0).

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

PARTE 2 [4 valores]

O Twitter permite que utilizadores registados enviem pequenas mensagens de texto, que podem ser lidas por outros utilizadores. Considere que para montar um sistema semelhante desenhou as seguintes classes.

A classe User representa um utilizador que tem um nome e um vector de mensagens de que é autor:

```
class User {
private:
        string name;
        vector<Message> messages;
public:
        User(string name);
        string getName() const;
        vector<Message> getMessages() const;
        vector<Message> getMessages(string subject) const;
        void addMessage(Message message);
};
```

A classe Message representa uma mensagem, que tem uma data e um conteúdo:

```
class Message {
       friend ostream& operator<<(ostream& os, Message& message);
private:
        time_t date; //time_t é um inteiro sem sinal
        string content;
public:
        Message(time_t timeMsg, string content);
        time_t getDate() const;
        string getContent() const;
};
```

a) [1] Implemente, na classe Message, o construtor.

```
Message::Message(time_t date, string content) {



}
```

b) [1] Implemente, na classe Message, o operador << que deve mostrar uma mensagem da seguinte forma:

1309185894 *// data da mensagem (de facto, representa o nº de segundos desde as zero horas de 1/1/1970)*
O que é nacional é bom *// conteúdo da mensagem*

```

```

c) [2] Implemente, na classe User, o método getMessages(string) que recebe uma *string* indicando o assunto da mensagem e devolve um vector com as mensagens cujo conteúdo refere esse assunto.

```
vector<Message> User::getMessages(string subject) const {









}
```

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

PARTE 3 [8 valores]

Uma determinada unidade curricular de um curso prevê, nos seus métodos de avaliação, a realização de um projecto de programação sobre um de vários temas possíveis. Para atribuição desses projectos aos estudantes, estes começam por definir uma lista com as suas preferências (um mínimo de 3 projectos, ordenados por ordem decrescente de preferência).

A título de exemplo, considere que existem 10 projectos possíveis (p_1, p_2, …, p_{10}) e que há 50 estudantes (e_1, e_2, …, e_{50}). As preferências, em relação ao projecto de programação, de cada um destes estudantes, são armazenadas num vector de vectores, conforme exemplificado na Tabela 2.1. Definiu-se, para tal, o seguinte tipo de dados:

```
typedef vector< vector<unsigned int> > Preferences;
```

As preferências de cada estudante **i** estão armazenadas na posição **i-1** de um vector deste tipo.

Pretende-se minimizar as atribuições de um mesmo projecto a estudantes diferentes (embora, como o número de estudantes é substancialmente superior ao número de projectos, possam sempre ocorrer sobreposições). Para representar o resultado da atribuição de projectos a estudantes, definiu-se um outro tipo de dados, consistindo num vector onde, para cada estudante, guardamos o projecto efectivamente atribuído:

```
typedef vector<unsigned int> Assignments;
```

A Tabela 2.2 apresenta uma possível atribuição de projectos a estudantes.

Índice do vector	Estudante	Projectos preferidos
0	1	5, 2, 9
1	2	2, 3, 4, 6
2	3	5, 9, 1
…	…	…
49	50	2, 6, 7, 3, 10

Tabela 2.1 - Preferências

Índice do vector	Estudante	Projecto atribuído
0	1	5
1	2	2
2	3	9
…	…	…
49	50	6

Tabela 2.2 - Atribuições

Nos exercícios que se seguem, considere que foram definidas as constantes globais `NPROJECTS`, `NSTUDENTS` e `MAX_ASSIGNMENTS_PER_PROJECT`, cujos valores representam o número de projectos, número de estudantes e número máximo de atribuições por projecto, respectivamente.

a) [1.5] Escreva a função `assignFirstPreference`, que recebe um vector de preferências e devolve um vector de atribuições com a primeira escolha de cada estudante.

```
Assignments assignFirstPreference(const Preferences &prefs) {




}
```

b) [1.5] Escreva a função `assignmentsPerProject`, que recebe um vector de atribuições e que devolve um vector contendo, para cada projecto, o número de estudantes a quem o mesmo foi atribuído. Note que, no vector a devolver, o número de estudantes com o projecto **i** estará na posição **i-1**.

```
vector<int> assignmentsPerProject(const Assignments &assig) {




}
```

c) [3] Pretende-se que cada projecto $\mathbf{p_i}$ seja atribuído a um máximo de **m** estudantes (m = MAX_ASSIGNMENTS_PER_PROJECT). Se, na distribuição inicial, $\mathbf{p_i}$ foi atribuído a **n** estudantes, em que **n>m**, uma forma de diminuir as atribuições em excesso é modificar **n-m** atribuições de $\mathbf{p_i}$, por exemplo, seleccionando as segundas escolhas de **n-m** estudantes a quem $\mathbf{p_i}$ foi inicialmente atribuído. O critério a ter em conta nesta modificação é a disponibilidade de vagas nos projectos correspondentes a essas segundas escolhas. Considere o seguinte exemplo, com 3 projectos, 5 alunos e **m**=2:

Estudante	Projectos preferidos
1	1, 2, 3
2	2, 3, 1
3	2, 1, 3
4	1, 3, 2
5	1, 2, 3

➔

Projecto atribuído inicialmente
1
2
2
1
1

➔

Projecto atribuído finalmente
1
2
2
3
1

Após a atribuição inicial, temos n=3 para o projecto p_1 e n=2 para o projecto p_2; é necessário modificar 1 (=n-m) atribuição do projecto p_1; modificou-se a atribuição do projecto ao estudante 4, já que a selecção da segunda escolha de qualquer outro estudante que também escolheu o projecto p_1 como primeira escolha levaria a que p_2 ficasse com n=3 atribuições.

Escreva a função improveAssignments que recebe um vector de preferências e um vector de atribuições e que procura melhorar as atribuições, de acordo com o critério definido acima.

```
void improveAssignments(const Preferences &prefs, Assignments &assign)
{









}
```

d) [2] Note que, se não houver um número suficiente de estudantes que cumpram o critério definido acima, poderá não ser possível obter uma solução com base nas segundas escolhas. Uma solução, não infalível, passará por aplicar o mesmo conceito iterativamente, isto é, considerando as terceiras escolhas, e assim sucessivamente (caso os estudantes tenham manifestado mais preferências). No seguinte exemplo, novamente com **m**=2, nenhuma segunda escolha permite evitar ultrapassar as 2 atribuições por projecto, pelo que uma solução poderá ser a apresentada:

Preferências
1, 2, 3
2, 3, 1
2, 1, 3
1, 2, 3
1, 2, 3

➔

Atribuição inicial
1
2
2
1
1

➔

Atribuição final
3
2
2
1
1

Escreva a função `iterativelyImproveAssignments` que recebe um vector de preferências e um vector de atribuições e que procura, iterativamente, melhorar as atribuições efectuadas, enquanto for possível.

SUGESTÃO: reutilize uma versão modificada da função `improveAssignments` da alínea anterior, indicando de que forma alteraria o seu protótipo e a sua implementação (não precisa de reescrever a função na totalidade)

```
void iterativelyImproveAssignments(const Preferences &prefs, Assignments &assign)
{









}
```

FIM

HLC/JAS/RCS