FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

PROGRAMAÇÃO (EIC0012) – 2011/2012 - 2º semestre

Exame da Época de Recurso - 2012/07/10

duração: 2h15m; com consulta

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

1. [5.0 valores]

a) [1.0] Um programador que pretendia desenvolver uma função para verificar se existem números duplicados num vetor de inteiros escreveu o seguinte código:

```
bool hasDuplicates(const vector<int> &v) {
  for (size_t i = 0; i <= v.size(); i++)
    for (size_t j = 0; j <= v.size(); j++)
    {
      if ((i <> j) and (v[i] = v[j]))
        return true;
    }
  return false;
}
```

O programador, ao compilar um programa que incluía esta função teve de corrigir vários erros sintáticos e quando, finalmente, conseguiu executar o programa em modo de *debugging*, o programa terminou abruptamente, com a mensagem de erro: *vector subscript out of range*. Assinale (acima, sublinhando) e corrija todos os erros que o programador cometeu.

Após a correção dos erros da função, outro programador, mais experiente, chamou a atenção para o facto de o código não ser eficiente, por realizar algumas comparações desnecessárias. Indique a(s) correção(ões) necessária(s) para tornar o código mais eficiente.

b) [2.5] Considere a seguinte função:

```
void mistery(const int x[], int nx, const int y[], int ny, int z[], int &nz) {
  int i, j, k;
  i = j = k = 0;
  while (i < nx && j < ny)
    if (x[i] == y[j]) {
      z[k++] = x[i]; i++; j++;
    }
    else
      if (x[i] < y[j])
       i++;
      else
        j++;
  nz = k;
}
```

Quais os valores de **a**, **na**, **b**, **nb**, **c**, e **nc** após a execução das seguintes instruções?

```
int a[5]={2,4,5,7,31};
int b[3]={4,7,20};
int c[5];
int na=5, nb=3, nc=0;
mistery(a,na,b,nb,c,nc);
```

a[] =	na =
b[] =	nb =
c[] =	nc =

Identifique quais os parâmetros que são passados por valor e quais os que são passados por referência e justifique a utilização do qualificativo **const** nos parâmetros **x[]** e **y[]**.

Assinalar à frente de cada um com **V** ou **R** consoante seja passado por valor ou por referência:

x[]-___ nx-___ y[]-___ ny-___ z[]-___ nz-___

Justificação para o uso de **const** em **x[]** e **y[]**:

Converta a função **mystery** numa ***template function***, de modo a que possa ser usada com ***arrays*** de valores de outros tipos simples (ex: `unsigned int`, `char`, `float`, …). **Nota**: basta indicar as alterações.

Considere que os dados a processar pela função lhe eram passados através de parâmetros do tipo **vector**, em vez de ***arrays***. Indique as alterações a introduzir na função para contemplar esta alteração do tipo dos parâmetros. **Nota**: se achar conveniente, pode alterar número de parâmetros da função.

c) [1.5] Apresenta-se a seguir a definição parcial das classes **Book** e **Library** usadas num programa de gestão de uma biblioteca:

```
typedef unsigned long IdentNum;

class Book
{
public:
        Book();
        Book(string bookName);
        void setName(string bookName);
        IdentNum getId() const;
        string getName() const;
        void show() const;
private:
        static IdentNum nextBookID;   // identificador do próximo livro a inserir na biblioteca
        IdentNum id;                  // identificador do livro
        string name;                  // nome do livro
};

class Library
{
public:
        Library();
        void addBook(Book book);
        void showBooks() const;
private:
        vector<Book> books;           // os livros existentes na biblioteca
};
```

O identificador **id** de cada livro é atribuído de forma automática, sendo dado pelo valor de **nextBookID** que é incrementado sempre que é introduzido um novo livro. Justique o uso do qualificativo **static** em **nextBookID** e indique como procederia à sua inicialização com o valor 1 (um).

Uma destas classes poderia ser declarada **friend** da outra. Justifique a utilização desta declaração e escreva-a, indicando onde deveria ser inserida.

Alguém sugeriu que os livros existentes na biblioteca poderiam ser guardados num **set**, em vez de um **vector**.
Vê algum inconveniente nesta solução?
Que alteração(ões) seria necessário introduzir na definição da(s) classe(s) **Book** e/ou **Library**?

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

2. [5.5 valores]

Apresenta-se a seguir parte do código de um programa utilizado numa pizzaria. O programa funciona, em ciclo, da seguinte forma: lê um pedido de um utilizador, calcula o montante a pagar, e regista o pedido e o montante num ficheiro. Cada pedido pode ser constituído por uma ou mais pizzas e tem um código que é atribuído de forma automática pelo programa.

```
#include ...
using namespace std;

const string PIZZARIA_FICH = "pedidos.txt";  // ficheiro onde são registados os pedidos

Vector<int> lePedido()
{
   // CÓDIGO A ESCREVER na alínea a)
}

_______ calculaPagamento(_____________________________) // PROTÓTIPO A COMPLETAR
{
   // CÓDIGO A ESCREVER na alínea d)
}

int main()
{
  int codigoDoPedido = 1;  // código do pedido efetuado
  bool novoPedido;         // indica se o utilizador pretende continuar a fazer pedidos
  // declara e abre ficheiro      // CÓDIGO A ESCREVER na alínea b)
  do {
    cout << "Pedido n.o: " << codigoDoPedido << endl;
    vector<int> pizzasPedidas = lePedido();
    int totalAPagar = calculaPagamento(pizzasPedidas);
    cout << "Pedido: " << codigoPedido << " - Total: " << totalAPagar << endl;
    //regista pedido no ficheiro   // CÓDIGO A ESCREVER na alínea c)
    //código que pergunta ao utilizador se quer inserir mais um pedido
    // ...
    codigoDoPedido++;
  } while(novoPedido);
  ficheiro.close();  // fecha o ficheiro
  return 0;
}
```

a) [1.5] Escreva o código da função **lePedido()** que pede ao utilizador (funcionário da pizzaria) que indique os códigos das pizzas do pedido, retornando-os no final. Os códigos são números inteiros com 3 ou mais dígitos; não é necessário validar os valores introduzidos. O utilizador pode inserir quantos códigos quiser; o código 0 simboliza o final da inserção e não deve ser incluido no vetor retornado pela função.

```
Vector<int> lePedido() {









}
```

b) [1.0**]** Escreva o código que declara e abre o ficheiro de texto onde vai ser feito o registo do novo pedido. O ficheiro deve ser aberto de modo a que os novos pedidos sejam escritos no final do ficheiro. O programa deve terminar imediatamente, retornando o valor 1, caso o ficheiro não seja aberto com sucesso.

c) [1.0] Escreva o código que regista no ficheiro de texto os códigos das pizzas do pedido e o montante total a pagar. Cada linha do ficheiro deve conter os dados de um pedido, sendo constituída da seguinte forma:

<código_do_pedido> - <código_da_pizza_1> <espaço> <código_da_pizza_2> ... - <total_a_pagar>.

Exemplo: o pedido 254 constituído pelas pizzas com código 1573, 282, e 38319, pelas quais deve ser pago o montante de 32 euros, deve ser registado da seguinte forma:

254 - 1573 282 38319 - 32

d) [2.0] Complete o protótipo da função **calculaPagamento()** e implemente-a. A função recebe um vetor com os códigos das pizzas pedidas e retorna o valor total que o cliente terá de pagar. O preço de cada pizza é obtido da seguinte forma:
- se no código da pizza houver algum dígito repetido, o preço será de 10€ (exemplo: a pizza cujo código é 282);
- caso contrário, o preço será de 12€ (exemplo: a pizza cujo código é 1573).

Nota: pode usar a função **hasDuplicates()** da alínea 1a) considerando que está corretamente implementada.

```
__________ calculaPagamento(_____________________________________) {  // protótipo a completar

}
```

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

3. [5.5 valores]

Para desenvolver um jogo de cartas, um programador concebeu, entre outras, as classes **Card** e **Hand** cujas definições parciais se apresentam abaixo.

Variável global:

```
map<char, string> suitNames;  // correspondência entre símbolos e nomes dos naipes
```

A classe **Card** representa uma carta:

```
class Card {
  friend bool operator<(const Card& card1, const Card& card2); //compara tendo em conta o rank
public:
  Card();
  Card(char suit, unsigned int rank);
  char getSuit() const;             // retorna o naipe da carta
  unsigned int getRank() const;     // retorna o valor da carta
  void setSuit(string suitName);    // altera o naipe da carta
private:
  char suit;            // o naipe da carta: 'C' - copas, 'E' - espadas, 'O' - ouros, 'P' - paus
  unsigned int rank;    // o valor da carta: 1 - ás, 2 - duque, ..., 11 - valete, 12 - dama, 13 - rei
};
```

A classe **Hand** representa as cartas que um jogador tem na mão:

```
class Hand {
public:
  Hand();
  void addCardInOrder(Card c);              // acrescenta uma carta à mão
  Card getCard();                           // retira uma carta da mão
  bool evenNumberOfFiguresAndAces() const;  // retorna true se o n.o de "figuras + ases" for par
private:
  vector<Card> cards;                       // o conteúdo da mão
};
```

a) [1.3] Implemente a função **setSuit()**, da classe **Card**, que recebe uma **string** com o nome do naipe e atualiza o campo de dados **suit** de acordo com a seguinte correspondência: copas – '**C**', espadas - '**E**', ouros - '**O**', paus - '**P**'. Considere que o parâmetro da função é sempre um valor válido, podendo estar escrito em maiúsculas ou em minúsculas.

b) [1.2] Implemente a função **addCardInOrder()**, da classe **Hand**, que adiciona uma carta à mão do jogador na posição adequada do vetor, de modo a manter a mão ordenada por ordem crescente do valor da carta (**rank**). **Nota**: pode usar métodos de ordenação disponíveis na linguagem.

```
void Hand::addCardInOrder(Card c) {




}
```

c) [2.0] Escreva a função **evenNumberOfFiguresAndAces()** que devolve **true** se a mão tiver um número par de "ases e figuras", e **false** no caso contrário (**Exemplos**: devolve **true** se a mão tiver um rei, duas damas e um valete; também devolve **true** se a mão tiver um rei e um ás).

```
bool Hand::evenNumberOfFiguresAndAses() const {




}
```

d) [1.0] Escreva o código que preenche o **map suitNames** (ver código fornecido) que associa o símbolo do naipe ao nome completo desse naipe. Use as seguinte correspondências: '**C**'–"**copas**", '**E**'–"**espadas**", '**O**'–"**ouros**", '**P**'–"**paus**".

```
    map<char, string> suitNames;  // correspondência entre símbolos e nomes dos naipes
```

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

4. [4 valores]

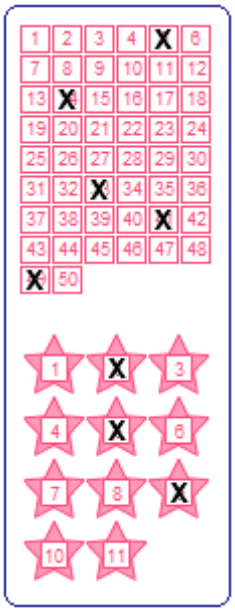
Para desenvolver uma aplicação cujo objetivo é registar e analisar apostas de Euromilhões definiram-se as seguintes classes **ApostaSimples** e **ApostaDeEuroMilhoes**:

```
typedef unsigned int Numero;

class ApostaSimples{
public:
  ApostaSimples(vector<Numero> numeros, Numero estrela1, Numero estrela2);
private:
  unsigned int numeros[5];
  unsigned int estrelas[2];
};

class ApostaDeEuroMilhoes{
public:
  ApostaDeEuroMilhoes(vector<vector<char> > numerosAp, vector<vector<char> > estrelasAp);
  vector<ApostaSimples> converteApostaMultipla();
private:
  vector<Numero> numeros;   // os números da aposta
  vector<Numero> estrelas;  // as estrelas da aposta
  unsigned int numNumeros;  // número de números
  unsigned int numEstrelas; // número de estrelas
  bool apostaMultipla;      // true se a aposta for múltipla
};
```

a) [2.5] Implemente o construtor da classe **ApostaDeEuroMilhoes** satisfazendo as seguintes condições. O construtor recebe duas matrizes representando os quadros de números e estrelas de um boletim real de euromilhões. O quadro de números tem 9 linhas e 6 colunas, com exceção da última linha que só tem 2 colunas. O quadro de estrelas tem 4 linhas e 3 colunas, tendo a última linha só 2 colunas. Cada quadro é implementado com recurso a matrizes de **char**. Cada célula dessas matrizes só pode ter um de dois valores: ' ' (carácter espaço) no caso de estar vazia e '**X**' para representar um número escolhido pelo utilizador.



Deve ler a matriz de números e preencher o campo de dados **numeros** que guarda os números apostados e o campo **numNumeros** que guarda o número de números apostados. Faça o mesmo para as estrelas, analisando a matriz da aposta feita e preenchendo os campos **estrelas** e **numEstrelas**. No final, deve preencher o campo **apostaMultipla** caso tenham sido apostados mais de 5 números ou 2 estrelas.

Considere que as matrizes **numerosAp** e **estrelasAp**, fornecidas como parâmetros do construtor, estão corretamente preenchidas.

Exemplo:

Conteúdo das matrizes **numerosAp** e **estrelasAp** para o exemplo apresentado na figura:

```
numerosAp[][] = {{' ',' ',' ',' ','X',' '},
                 {' ',' ',' ',' ',' ',' '},
                 {' ','X',' ',' ',' ',' '},
                 {' ',' ',' ',' ',' ',' '},
                 {' ',' ',' ',' ',' ',' '},
                 {' ',' ','X',' ',' ',' '},
                 {' ',' ',' ',' ',' ',' '},
                 {' ',' ',' ',' ','X',' '},
                 {' ',' ',' ',' ',' ',' '},
                 {'X',' '}}

estrelasAP[][] = {{' ','X',' '},
                  {' ','X',' '},
                  {' ',' ','X'},
                  {' ',' '}}
```

Conteúdo dos campos **numeros** e **estrelas** de **ApostaDeEuroMilhoes**, para o mesmo exemplo, preenchidos pelo construtor após processar **numerosAp** e **estrelasAp**.

numeros[] = {5,14,33,41,49} **estrelas[]** = {2,5,9}

```
ApostaDeEuroMilhoes::ApostaDeEuroMilhoes(vector<vector<char> > numerosAp, vector<vector<char> > estrelasAp) {
```

```
}
```

b) [1.5] Implemente a função **converteApostaMultipla()** que analisa a aposta codificada nos campos **numeros** e **estrelas** de **ApostaDeEuroMilhoes** e devolve o conjunto de apostas simples que a aposta guardada representa. Por simplicidade assuma que na aposta a analisar só as estrelas podem representar uma aposta múltipla, isto é, o número de números é 5, podendo o número de estrelas ser 2, ou 3, …ou 11. Tenha em conta que, não havendo múltiplas nos números, uma aposta múltipla é equivalente a todas as combinações (sem repetição) de 2 estrelas da aposta múltipla mantendo a combinação única de números.

Exemplo: sendo a aposta múltipla constituída por numeros={5,14,33,41,49} + estrelas={2,5,9}, as apostas simples serão constituídas por numeros={5,14,33,41,49} + estrelas={2,5}, numeros={5,14,33,41,49} + estrelas={2,9} e numeros={5,14,33,41,49} + estrelas={5,9}.

```
vector<ApostaSimples> ApostaDeEuroMilhoes::converteApostaMultipla() {

}
```

FIM

JAS/RCS/TPF/ICM