Curso: ....

Prova de Programação por Objectos (21093) **Data: 22 de Julho de 2010**

Nome: ....

Nº de Estudante: .... **B. I. nº** ....

Turma: .... **Assinatura do Vigilante:** ....

RESERVADO PARA A *Universidade Aberta*

Classificação: (   ) ....

Prof. que classificou a prova: ....

LEIA ATENTAMENTE AS INSTRUÇÕES PARA A RESOLUÇÃO DO EXAME:

1. O teste é constituído por três questões, com várias alíneas cada.
2. Quando solicitado é necessário justificar, de forma sucinta, a resposta dada.
3. Leia o enunciado de todas as questões antes de começar a responder.
4. O tempo de resolução do exame é de duas horas e trinta minutos.
5. A cotação de cada uma das questões é indicada junto do enunciado da mesma.
6. O exame é **SEM CONSULTA**. Todos os elementos necessários à resolução são fornecidos no enunciado.
7. Utilize esferográfica azul ou preta para responder às questões. Respostas a lápis não serão consideradas.
8. O código de programação a apresentar deve ser em C++, apresentado de forma clara, indicando por meio de comentários todas as opções tomadas e todos os aspectos que por qualquer razão sejam menos claros. O código não deve utilizar construções típicas da linguagem C como *printf* ou *scanf*.
9. Todas as situações que evidenciem que o código de programação fui copiado de um colega, total ou parcialmente, conduzirão à atribuição de cotação zero na questão em causa em todos os exames dos alunos envolvidos.
10. O exame é constituído por 22 páginas enumeradas.
11. Se o seu exemplar não estiver completo ou nele se verificar qualquer outra deficiência, por favor dirija-se ao professor vigilante.

QUESTÃO 1 (3 valores) (a=0.5; b=0.5; c=1.0; d=1.0)

Questões de resposta múltipla, onde apenas uma resposta está correcta. Faça um círculo na letra da resposta que considerar correcta.

a) Considere a expressão:

!(a || b)

Esta é equivalente a qual das seguintes expressões?

A. (a || b)
B. (!a) || (!b)
C. (!a) && (!b)
D. !(a && b)
E. (a || b) && (a || b)

b) Qual das seguintes afirmações acerca de variáveis em programas C++ está correcta?

A. Uma variável deve ser declarada antes de ser utilizada.
B. O mesmo nome da variável não pode ser utilizado em duas funções diferentes.
C. É considerada uma boa prática de programação em C++ declarar todas as variáveis não escalares (ex. vectores) globais e declarar locais as variáveis escalares.
D. É considerada uma boa prática de programação em C++ usar letras individuais para nomes de todas as variáveis.
E. É considerada uma boa prática de programação em C++ nomear os parâmetros formais “param1, param2”, etc. de tal forma que se torna fácil identificar a ordem com que eles aparecem na lista de parâmetros da respectiva função.

c) Seja a seguinte função booleana:

```
 bool MyMysteryFunction (const vector <int> & A)
// pré-condição: A está ordenado
{
  int k;
  for (k=1; k<A.size(); k++)
  {
    If (A[k-1] == A[k]) return true;
   }
  return false;
}
```

Qual das seguintes afirmações explica melhor o que a função *MyMysteryFunction* faz?

A. Retorna sempre true.
B. Retorna sempre false.
C. Determina se o vector A está (ou não) realmente ordenado.
D. Determina se o vector A contém (ou não) algum valor duplicado.
E. Determina se todos os valores em A são iguais (ou não).

Justifique sucintamente a sua resposta.

____

____

____

____

____

____

____

____

____

____

____

d) Qual das seguintes afirmações explica melhor porque A é um parâmetro da função passado por referência?

A. A pode ser modificado pela função *MyMysteryFunction;* portanto, A deve ser passado por referência.
B. A é indexado pela função *MyMysteryFunction*; portanto, A deve ser passado por referência.
C. O método *size()* de A é utilizado na função *MyMysteryFunction*; portanto, A deve ser passado por referência.
D. É mais eficiente passar A por referência do que por valor.
E. Não existe uma razão para que A seja passado por referência.

Justifique sucintamente a sua resposta.

____

____

____

____

____

____

____

____

____

____

____

____

QUESTÃO 2 (9 valores) (a=1.0; b=2.0; c=1.0; d=2.0; e=3.0)

a) Escreva o código em C++ para uma função com o nome *FindZero* conforme iniciada mais abaixo. *FindZero* devera retornar o índice da primeira posição no vector A que contém o valor zero, iniciando pela posição *pos*. Se não existir um valor zero entre a posição *pos* e o final do vector então a função deve retornar -1.

Exemplo:

Vector A	Posição *pos*	Valor retornado por FindZero (A, pos)
1 0 2 5 6	0	1
1 0 2 5 6	1	1
1 0 2 5 6	2	-1
1 0 2 0 6	0	1
1 0 2 0 6	1	1
1 0 2 0 6	2	3
1 2 3 4 5	0	-1

Complete então a função *FindZero* abaixo. Assuma que a função é chamada apenas com valores que satisfazem a sua pré-condição.

```
int FindZero(const vector <int> &A, int pos)
{
 // pré-condição: 0 <= pos < A.size()
 // pré-condição: retorna o índice k mais pequeno tal que
 // (pos <= k < A.size()) and (A[k] == 0)
```

(código para ser programado)

```
}
```

Apresente código e justificação.

b) Escreva código em C++ para a função *MySetZeros* conforme indicada mais abaixo. *MySetZeros* deve encontrar as posições (índices) dos primeiros dois zeros no vector A e alterar para zero todos os valores de A nas posições intermédias, ou seja, entre as posições dos dois primeiros zeros encontrados. Se A apenas contém zeros ou se não contém quaisquer zeros ou se os dois primeiros zeros se posicionam em índices imediatamente consecutivos (um a seguir imediatamente ao outro) então *MySetZeros* não deverá modificar A.

Exemplo:

Vector A antes de chamar *MySetZeros*	Vector A depois de chamar MySetZeros (A)
0 1 2 0 4 0	0 0 0 0 4 0
1 0 2 3 4 0	1 0 0 0 0 0
1 2 0 0 4 5	1 2 0 0 4 5
1 0 2 3	1 0 2 3
1 2 3 4	1 2 3 4

Quando programar a função *MySetZeros* deve incluir chamadas à função *MyFindZeros* da questão anterior. Admita que a função *MyFindZeros* corre conforme especificado, mesmo que não a tenha programado.

Complete então a função *MySetZeros* abaixo:

```
void MySetZeros (vector <int> & A)
{
  (código para ser programado)
}
```

Apresente código e justificação.

c) Considere que a classe *Product*, abaixo especificada, foi implementada. Esta classe representa um produto para venda num estabelecimento comercial. Os métodos públicos da classe permitem que uma aplicação cliente obtenha:

- o nome do produto
- quantidade deste produto actualmente em stock

e que subtraia 1 da quantidade de produtos actualmente em stock.

```
class Product {
  public:
    Product (string name);   // constructor
    string Name () const;    // devolve o nome deste produto
    int NumInStock () const;   // devolve a quantidade deste produto em stock
    void SellOne ();   // subtrai da quantidade actual deste produto em stock

  private:
    string myName;
    int myNumInStock;
};
```

Escreva uma função *MyFindItem*, como abaixo iniciada. *MyFindItem* deverá devolver o índice de um produto da classe *Product* num vector (*array*) A dado um nome ou devolver -1 se esse produto não existir em A.

Complete então a função *MyFindItem* assumindo que esta é apenas chamada com valores que satisfizerem a sua pré-condição.

```
int MyFindItem (const vector <Product> &A, const string &name)
// pré-condição: não existem produtos com o mesmo nome em A
{
  (código para ser programado)
}
```

Apresente código e justificação.

d) Escreva o código em C++ da função *MyOneSale*, como iniciada mais abaixo. A função *MyOneSale* tem dois parâmetros: um vector (*array*) de produtos (*Product*) nomeado *inventory* e um nome de um produto (que um cliente pretende comprar). A função *MyOneSale* deve tentar realizar a venda do produto indicado pelo nome e devolver *true* ou *false* conforme a venda for possível ou não. A venda acontece se existir pelo menos um produto (de nome indicado) no vector. Nesse caso, *MyOneSale* deve subtrair 1 do número de produtos em stock e devolver *true*. Se não existir um produto com o nome indicado ou o número em stock desse produto for zero (o produto não existe em stock) então *MyOneSale* deve devolver *false*.

Por exemplo, assuma que inventory.size () é 4 e os produtos no vector são:

[0]	[1]	[2]	[3]
“milk”	“eggs”	“butter”	“coffee”
20	3	0	1

Se a *MyOneSale* for chamada com este inventário (*inventory*) com o nome “eggs”, deve subtrair 1 do número de “eggs” (ovos) em stock e devolver *true*. Se *MyOneSale* for chamado com o nome “juice” (sumo) deve retornar *false* porque esse produto não existe de todo no vector. Se por outro lado *MyOneSale* for chamada com o nome “butter” (manteiga) deve retornar falso porque não existe disponibilidade deste produto em stock.

Na escrita do código da função *MyOneSale* deverá considerar a função *MyFindItem* especificada na questão anterior. Assuma que a função *MyFindItem* executa correctamente conforme a especificação dada na questão anterior, mesmo que a não tenha programado.

Complete então a função *MyOneSale* admitindo que esta é apenas chamada com valores que satisfizerem a sua pré-condição.

```
bool MyOneSale (vector <Product> &inventory, const string &name)
// pré-condição: não existem produtos com o mesmo nome em inventory
{
  (código para ser programado)
}
```

Apresente código e justificação.

e) Escreva o código em C++ da função *MyAllSales*, como iniciada mais abaixo. A função *MyAllSales* tem quatro parâmetros:

- um vector de produtos (*Product*) nomeado *inventory*;
- um vector de nomes nomeado *orders*;
- um vector de *strings* nomeado *failed*
- e um inteiro *N*

Para cada nome em *orders*, *MyAllSales* deverá procurar realizar a venda do produto com esse nome. Deverá preencher o vector de nomes *failed* com os nomes de todos os produtos em que a venda falhou (um produto pode aparecer mais que uma vez em *failed* se existir mais que uma tentativa falhada para a sua venda). Finalmente, *MyAllSales* deve alterar o valor de *N* para o número de valores no vector *failed*.

Por exemplo, assuma que *inventory* tem os valores indicados no exemplo da questão anterior. Assuma ainda que *MyAllSales* é chamada com o seguinte vector de nomes:

“eggs” “milk” “milk” “butter” “coffee” “tea” “coffee” “milk” “coffee”

MyAllSales deverá levar a efeito cinco vendas com sucesso (*eggs*, *milk*, *coffee*, *milk*) alterando os respectivos produtos (*Products*) no *inventory*. O vector de nomes *failed* deverá ser construído resultando em:

“butter” “tea” “coffee” “coffee”

porque o número de produtos “butter” é zero, e não existe nenhum produto com o nome “tea”; e o número de existências do produto “coffee” é zero aquando da segunda e terceira tentativas de venda deste produto. Finalmente, *MyAllSales* deve atribuir o valor 4 a *N* (o número de elementos no vector *failed*).

Na escrita da função *MyAllSales* deverá considerar a função *MyOneSale*, especificada na questão anterior. Assuma que esta função está implementada e a executar conforme especificado mesmo que a não tenha programado.

Complete a função abaixo, assumindo que esta é chamada apenas com valores que satisfazem a sua pré-condição.

```
void MyAllSales (vector <Product> & inventory, const vector <string> &orders,
                    vector <string> &failed, int &N)
// pré-condição: não existem produtos com o mesmo nome em inventory
//                 e failed.size () >= orders.size()
{
  (código para ser programado)
}
```

Apresente código e justificação.

QUESTÃO 3 (8 valores) (a=1.0; b=1.0; c=3.0; d=3.0)

Questões de resposta múltipla, onde apenas uma resposta está correcta. Faça um círculo na letra da resposta que considerar correcta.

a) Considere a seguinte função:

```
int Recurse(int x, int y)
{
  if (x == 0) return 0;
  else return (1 + Recurse(x - y, y));
}
```

O que devolve a chamada de Recurse (20, 2)?

A. 1
B. 10
C. 20
D. 40
E. Nada é devolvido porque a chamada gera um ciclo recursivo infinito.

Justifique a sua resposta, simulando os passos das chamadas recursivas.

b) Qual das seguintes afirmações melhor caracteriza os valores para *x* e *y* tal que a chamada de *Recurse (x, y)* cause um ciclo recursivo infinito?

A. x é negativo
B. y é negativo
C. x é menor que y
D. x é par e y é ímpar
E. x não é um múltiplo positivo de y

Justifique a sua resposta, simulando como as chamadas recursivas na condição escolhida para os valores de *x* e *y* tendem para um ciclo recursivo infinito.

c) Considere as duas classes definidas mais abaixo. A classe *Person* é utilizada para guardar informação sobre uma pessoa; a classe *Family* é utilizada para guardar informação sobre uma família (constituída por zero ou mais pessoas). A classe *Family* contém um membro apontador que ou é *NULL* ou aponta para o primeiro nó de uma lista ligada de pessoas que constituem a família. Estes nós da lista ligada são implementados com a construção *ListNode struct* que se encontra definida a seguir:

```
class Person {
   public:
     string Name () const;    // devolve o nome desta pessoa
     int Age () const;           // devolve a idade desta pessoa
   private:
     string myName;
     int myAge;
};

struct ListNode {
    Person myPerson;
    ListNode *next;
    ListNode (Person p, ListNode *n);   // construtor: altera myPerson para p e
                                                      // next para n
};

class Family {
   public:
     void AddPerson (Person p);      // acrescenta pessoa p a esta família
   private:
     ListNode *peopleList;   // apontador para a lista ligada de pessoas
                                     // nesta família, ordenada por idade (do mais novo
                                     // até ao mais idoso)

    // método privado
     ListNode *PersonBefore (int age) const;
              // devolve o apontador para o nó na lista ligada que contém a pessoa
              // nesta família cuja idade é a mais próxima da idade dada por “age”
              // sem ser maior (igual ou mais próxima por baixo);
              // devolve NULL se não existirem pessoas nesta família (família vazia)
              // ou não existe ninguém com idade inferior ou igual a “age”
};
```

Escreva o código em C++ da função *PersonBefore*, como iniciada mais abaixo. Esta função deverá proceder com a pesquisa da pessoa com a idade mais próxima da idade indicada *age* (a mais próxima que seja menor ou igual a *age*) através da lista ligada de pessoas *peopleList*. Deverá devolver o apontador para o nó da pessoa encontrada na lista ligada. Se essa pessoa não existir (se a lista estiver vazia ou todas a idades forem superiores ao valor dado *age*) a função PersonBefore deverá devolver *NULL*.

```
ListNode * Family::PersonBefore (int age) cont  {
   // pré-condição: peopleList é NULL ou aponta para o primeiro nó da lista ligada;
   // a lista está ordenada por idade das pessoas, do mais jovem para o mais idoso
   // pós-condição: devolve um apontador para nó na lista ligada que contém info.
   // da pessoa na família cuja idade é a mais próxima ou igual a age mas não
   //  a ultrapassando  (é inferior ou igual).
   // devolve NULL se não existirem pessoas na família ou nenhuma das pessoas tiver
   // idade igual ou inferior a age

    (código para ser programado)

    };
```

Apresente código e justificação.

d) Escreva o código em C++ para o método *AddPerson* da classe *Family*, conforme iniciado mais abaixo. Esta função acrescenta uma pessoa *p* à família, tal que a lista de pessoas da família se mantém ordenada por idade (do mais jovem até ao mais idoso).

Na escrita da função *AddPerson* deverá considerar chamada(s) à função *PersonBefore*, conforme especificada na questão anterior, mesmo que a não tenha programado.

Complete então a função *AddPerson* assumindo que esta é chamada apenas quando os valores dos seus parâmetros satisfizerem a sua pré-condição.

```
void Family::AddPerson (Person p) {
// pré-condição: o membro da classe peopleList é NULL ou aponta para o primeiro
// nó da lista ligada;
// a lista está ordenada por idade das pessoas, a começar no mais novo até ao mais
// idoso.
```

(código para ser programado)

```
}
```

Apresente código e justificação.

FIM