UNIVERSIDADE
AbERTA
www.uab.pt

Ministério da Ciência, Tecnologia e Ensino Superior

p-Fólio

U.C. 21093

Programação por Objectos

11 de Julho de 2011

-- INSTRUÇÕES --

- O estudante deverá responder à prova na folha de ponto e preencher o cabeçalho e todos os espaços reservados à sua identificação, com letra legível.
- Sempre que não utilize o enunciado da prova para resposta, poderá ficar na posse do mesmo.
- No caso de provas com escolha múltipla, **sem grelha de resposta**, deverá indicar a resposta correcta na folha de ponto, indicando o número da pergunta e a resposta que considera correcta.
- No caso de provas com escolha múltipla, **com grelha de resposta, tabela e/ou espaços para preenchimento**, deverá efectuar as respostas no enunciado, pelo que o mesmo deverá ser entregue ao vigilante, juntamente com a folha de ponto, **não sendo permitido ao estudante levar o enunciado.**
- Verifique no momento da entrega da(s) folha(s) de ponto se todas as páginas estão rubricadas pelo vigilante. Caso necessite de mais do que uma folha de ponto, deverá numerá-las no canto superior direito.
- Em hipótese alguma serão aceites folhas de ponto dobradas ou danificadas.
- Exclui-se, para efeitos de classificação, toda e qualquer resposta apresentada em folhas de rascunho.
- Os telemóveis deverão ser desligados durante toda a prova e os objectos pessoais deixados em local próprio da sala de exame.
- A prova é constituída por 7 páginas e termina com a palavra **FIM**. Verifique o seu exemplar e, caso encontre alguma anomalia, dirija-se ao professor vigilante nos primeiros 15 minutos da mesma, pois qualquer reclamação sobre defeito(s) de formatação e/ou de impressão que dificultem a leitura não será aceite depois deste período.
- Utilize unicamente tinta azul ou preta.
- A prova é **SEM CONSULTA**. Todos os elementos necessários à resolução são fornecidos no enunciado. A prova é constituída por 2 questões, com várias alíneas cada. A cotação é indicada.
- 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*.
- Todas as situações que evidenciem que o código de programação foi copiado de um colega, total ou parcialmente, conduzirão à atribuição de cotação zero nas provas dos alunos envolvidos.

Duração: 90 minutos

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

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)

Indique a letra da resposta correcta.

(Resposta: 1 linha)

b) Qual das seguintes afirmações melhor descreve as circunstancias nas quais a expressão

! (a && b) && (a || b) tem valor *true* ?

A. Sempre.
B. Nunca.
C. Quando ambos *a* e b forem *true*.
D. Quando nem *a* nem *b* forem *true*.
E. Quando exactamente um deles *a* ou *b* é *true*.

Indique a letra da resposta correcta.

(Resposta: 1 linha)

c) Considere o seguinte bloco de código:

```
int x = 0;
bool y = true;

if (y && (x != 0) && (2/x == 0)) cout << "success" << endl;
else cout << "failure" <<<< endl;
```

Qual das seguintes afirmações sobre este bloco de código é verdadeira?

A. Ocorre um erro quando o código é compilado porque a primeiro operador *&&* é aplicado a uma expressão não booleana.
B. Ocorre um erro quando o código é compilado porque uma variável booleana (*y*) e uma variável inteira (*x*) aparecem na mesma condição *if*.
C. Ocorre um erro quando o código é executado devido a uma tentativa de dividir por zero.
D. O código compila e executa sem erro; a saída é "success".
E. O código compila e executa sem erro; a saída é "failure".

Indique a letra da resposta correcta e justifique sucintamente a sua opção.

(Resposta: 6 linhas)

d) Considere o seguinte bloco de código:

```
if (y < 0)
{
  x = -x;
  y = -y;
}
z = 0;
while (y > 0)
{
    z+= x;
    y--;
}
```

Assuma que *x*, *y* e *z* são variáveis do tipo *int* e que *x* e *y* foram inicializadas. Qual das seguintes afirmações melhor descreve o que o bloco de código faz?

A. *z* recebe a soma de x + *y*.
B. *z* recebe o produto de *x***y*.
C. *z* recebe o valor absoluto do valor de *x*.
D. *z* recebe o valor de x^x.
E. *z* recebe o valor de y^x.

Indique a letra da resposta correcta e justifique sucintamente a sua opção.

(Resposta: 6 linhas)

QUESTÃO 2 (9 valores) (a=1.5; b=1.5; c=1.5; d=2.0; e=2.5)

a) Assuma que a classe BankAccount, abaixo especificada, foi implementada. Cada membro da classe representa uma conta bancária pessoal. Os métodos públicos da classe permitem que um programa cliente encontre:

- O número da conta, e
- O saldo corrente (quanto dinheiro está actualmente depositado na conta).

Estes métodos públicos permitem que seja adicionado dinheiro e subtraído do saldo corrente através de um depósito ou levantamento, respectivamente.

```
class BankAccount {
   public:
        BankAccount (int accountNum);   // constructor
        int AccountNum () const;        // devolve o número de conta desta conta
        double Balance () const;        // devolve o saldo corrente desta conta
        void Deposit (doube amount);    // adiciona amount ao saldo corrente
        void Withdraw (double amount);  // subtrai amount do saldo corrente
     private:
        //membros privados
};
```

Assuma ainda que a seguinte declaração foi realizada para representar informação sobre transacção numa conta bancária (um depósito ou um levantamento):

```
struct Transaction {
   int accountNum;
   char flag;      // 'd' para depósito e 'w' para levantamento
   double amount;
};
```

Programe a função *FindAccount*, iniciada abaixo. A função *FindAccount* deverá devolver o índice de um vector A de *BankAccounts* dado um determinado número de conta. Devolve -1 se essa *BankAccount* não existir no vector.

Complete então a função *FindAccount* abaixo em código C++. Assuma que a função é chamada apenas com valores que satisfazem a sua pré-condição.

```
int FindAccount (const vector <BankAccount> & A, int num)
 {
  // pré-condição: não existem duas contas em A com o mesmo número

      (código C++ para ser programado)
 }
```

Apresente código e justificação.

(Resposta: 15 linhas)

b) Escreva código em C++ para a função *OneTransaction,* conforme iniciada mais abaixo. *OneTransaction* tem dois parâmetros: um vector de *BankAccount* e uma *Transaction.* A função *OneTransaction* deverá encontrar a conta (*BankAccount*) dado um número de conta e deverá depositar ou levantar a quantia *amount*, em conformidade.

Por exemplo, assuma que accounts.size () é 4 e que os elementos no vector têm os seguintes números de conta e saldos:

[0]	[1]	[2]	[3]
100	107	102	105
€100.27	€57.30	€150.00	€5.25

Eis alguns exemplos que ilustram o comportamento da função *OneTransaction*:

Valor de *trans*	Elemento modificado no arrray *accounts*
accountNum: 107 flag: 'd' amount: 10.50	[1] 107 67.80
accountNum: 100 flag: 'w' amount: 100.27	[0] 100 0.0
accountNum: 105 flag: 'w' amount: 6.00	[3] 105 -0.75

Quando programar a função *OneTransaction* pode incluir chamadas à função *FindAccount* da questão anterior. Admita que a função *FindAccount* corre conforme especificado, mesmo que não a tenha programado. Assuma ainda que a função é chamada apenas com valores que satisfazem a sua pré-condição.

Complete então a função *OneTransaction* abaixo:

```
void OneTransaction  (vector <BankAccount> &accounts, Transaction trans)
// pré-condição: não existem duas contas no vector accounts com o mesmo número
//                uma das contas em accounts tem número de conta trans.accountNum
//                trans.flag tem um dos valores 'd' ou 'w'
 {
  (código C++ para ser programado)
}
```

Apresente código e justificação.

(Resposta: 15 linhas)

c) Escreva código em C++ para a função *DaysTransactions,* conforme iniciada mais abaixo. *DaysTransactions* tem quatro parâmetros: um vector de *BankAccount* com nome accounts, um vector de *Transactions*, um vector de inteiros (*int*) com nome *overdrawn*, e um inteiro *int* N. A função *DaysTransactions* deverá realizar cada uma das transacções do vector *Transactions*. Deverá ainda preencher o vector *overdrawn* com o número de todas as contas que após a realização das transacções fiquem com saldo negativo. Finalmente N deverá receber o número total de contas com saldo negativo.

No exemplo da questão anterior N receberia 1 e o vector *overdrawn* receberia a conta 105 que ficou com saldo negativo.

Quando programar a função *DaysTransactions* pode incluir chamadas à função *OneTransaction* da questão anterior. Admita que a função *OneTransaction* corre conforme especificado, mesmo que não a tenha programado. Assuma ainda que a função é chamada apenas com valores que satisfazem a sua pré-condição.

Complete então a função *DaysTransactions* abaixo:

```
void DaysTransactions (vector <BankAccount> & accounts,
                       const vector <Transactions> & transactions,
                       vector <int> & overdrawn,
                       int & N)
// pré-condição: não existem duas contas no vector accounts com o mesmo número;
//                todas as Transactions no vector transactions tem uma conta
//                correspondente em accounts;  todas as Transactions no vector
//                transactions tem flag com um dos valores 'd' ou 'w'
//                overdrawn.size() >= accounts.size()
{
  (código C++ para ser programado)
}
```

Apresente código e justificação.

(Resposta: 15 linhas)

d) Escreva o código em C++ da função *Move2Left*, como iniciada mais abaixo. A função *Move2Left* deverá mover duas posições para a esquerda todos os caracteres que ocupam as posições de k até S.length()-1 em S escrevendo sobre os caracteres anteriores aí posicionados. Assuma que k tem sempre um valor no intervalo de 2 a S.length()–1. Note que a função reduz S de dois caracteres.

Exemplo:

String S	Chamada de Move2Left	S devolvido por Move2Left
"abcde"	MoveLeft (S, 2)	"cde"
"abcde"	MoveLeft (S, 3)	"ade"
"abcde"	MoveLeft (S, 4)	"abe"

Na escrita do código da função *Move2Left* poder utilizar a função *substr* da classe *string* (C++ standard library).

Complete então a função *Move2Left* admitindo que esta é apenas chamada com valores que satisfazem a sua pré-condição.

```
void Move2Left (string & S, int k)
// pré-condição: 2 < k < S.length ()
//                S contém os caracteres c0 c1 ... cn
//Pós-condição:  S contém os caracteres c0 c1 ... ck-3 ck ck+1 ... cn
  {
     (código C++ para ser programado)
  }
```

Apresente código e justificação.

(Resposta: 15 linhas)

e) Escreva o código em C++ da função *RemoveTriples*, como iniciada mais abaixo. A função *RemoveTriples* deverá encontrar todos os conjuntos de 3 letras em S e converter estes triplos numa letra movendo todos os caracteres à direita do triplo duas posições para a esquerda (reduzindo S de dois caracteres de cada vez que se executa esta operação).

Exemplo:

String S	Valor retornado por RemoveDoubles (S)
"happpy"	"hapy"
"aaabcc"	"abcc"
"aaabc"	"abc"

Na escrita da função *RemoveTriples* poderá considerar chamadas à função *Move2Left* da alínea anterior. Assuma que esta função está implementada e a executar conforme especificado mesmo que a não tenha programado.

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

void *RemoveTriples* (string & S)

// pré-condição: S contém somente letras minúsculas

{

(código C++ para ser programado)

}

Apresente código e justificação.

(Resposta: 15 linhas)

FIM