Algoritmos e Programação I: Lista de Exercícios 08 - Orientações para submissão no Moodle e BOCA. *

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
`http://moodle.facom.ufms.br`

Após submeter no BOCA, não se esqueça de enviar o programa para o Moodle, usando o seguinte nome: pxxloginname.c, onde xx é o número do problema e o loginname é o seu login.

Compile o seu programa usando:

```
gcc -Wall -pedantic -std=c99 -o programa programa.c [-lm]
```

A flag `-lm` deve ser usada quando o programa incluir a biblioteca `math.h`.

Uma forma eficiente de testar se o seu programa está correto é gerando arquivos de entrada e saída e verificar a diferença entre eles. Isso pode ser feito da seguinte forma:

```
./programa < programa.in > programa.out
```

O conjunto de entrada deve ser digitado e salvo num arquivo `programa.in`. O modelo da saída deve ser digitado e salvo num arquivo `programa.sol`.

Verifique se a saída do seu programa `programa.out` é EXATAMENTE igual ao modelo de saída `programa.sol`. Isso pode ser feito com a utilização do comando `diff`, que verifica a diferença entre dois arquivos.

```
diff programa.out programa.sol
```

O seu programa só está correto se o resultado de `diff` for vazio (e se você fez todos os passos como indicado).

A solução dos exercícios deve seguir a metodologia descrita em sala:

a. Diálogo

b. Saída/Entrada (pós e pré)

c. Subdivisão

d. Abstrações

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina (B. Forouzan e R. Gilbert, A. B. Tucker et al. e S. Leestma e L. Nyhoff, Lambert et al., H. Farrer et al., K. B. Bruce et al., e Material Didático dos Profs. Edson Takashi Matsubara e Fábio Henrique Viduani Martinez.

e. Implementação

f. Teste

Na submissão ao BOCA não se esqueça de comentar todos os `printf`'s da entrada e que a saída não pode conter acentuação ou *ç*.

Exercícios

1. Considere as seguintes declarações:

```
typedef int BigTable[50][100];
typedef float TabelaPontos[21][21];
typedef char TabelaChar[26][26];
typedef bool TabelaBoolean[2][2];
typedef unsigned int BitArray[2][2];
typedef unsigned int Camisa[6][5][5];
typedef float ArrayMixed[6][6][11][2];
typedef bool ArrayEmEstoque[6][6];
typedef Camisa EstoqueCamisa[5];
```

   Quantos elementos podem ser armazenados nos vetores com os seguintes tipos?

   a. BigTable
   b. TabelaPontos
   c. TabelaChar
   d. TabelaBoolean
   e. BitArray
   f. Camisa
   g. ArrayMixed
   h. ArrayEmEstoque
   i. EstoqueCamisa

2. Assuma que as seguintes declarações tenham sido feitas:

```
typedef char String[6];
typedef float Array3X3[3][3];
typedef String ArrayDeStrings[2];

// declaração das variáveis
ArrayDeStrings    Texto;
Array3X3          Matriz;
unsigned int      i,j;
char              Ch;
```

   e os seguintes dados são fornecidos para as instruções que envolvem entrada a seguir (onde □ denota um branco e ® denota `ENTER` ou `RETURN`):

```
ABC□DE®
FGH□IJ®
KLM□NO®
PQR□ST®
UVW□XY®
Z01□23®
```

Para cada um dos seguintes conjuntos de instruções, diga qual o valor (se algum) é atribuído para cada elemento do array ou explique porque ocorre algum erro:

a.
```
for (i=0; i<=2; i++) {
   for (j=0; j<=2; j++) {
      Matriz[i][j] = i+j;
   }
}
```

b.
```
 for (i=0; i<=1; i++) {
   scanf("%s", Texto[i]);
}
```

c.
```
for (i=0; i<=1; i++) {
   for (j=0; j<=5; j++) {
      scanf("%c", &Texto[i][j]);
   }
   scanf("%c", &Ch);
}
```

d.
```
for (i=0; i<=1; i++) {
   for (j=0; j<=5; j++) {
      scanf("%c", &Texto[i][j]);
   }
}
```

e.
```
 for (j=0; j<=5; j++) {
   for (i=0; i<=1; i++) {
      scanf("%c", &Texto[i][j]);
   }
   scanf("%c", &Ch);
```

f.
```
 for (j=0; j<=5; j++) {
   for (i=0; i<=1; i++) {
      scanf("%c", &Texto[j][i]);
   }
   scanf("%c", &Ch);
}
```

g.
```
 for (i=0; i<=1; i++) {
   for (j=0; j<=5; j++) {
      scanf("%c", &Ch);
      if (Ch != '\n') {
```

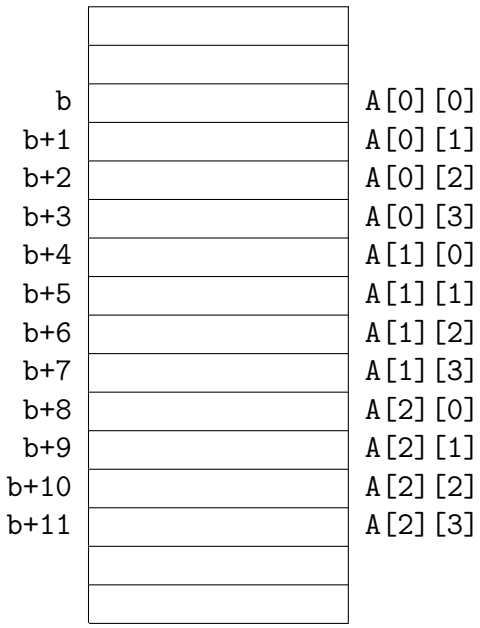
```
            Texto[i][j] = Ch;
         }
      }
   }
```

h.
```
   for (i=0; i<=2; i++) {
      for (j=2; j>=0; j--) {
         if (i == j) {
            Matriz[i][j] = 1;
         }
         else {
            Matriz[i][j] = 0;
         }
      }
   }
```

i.
```
   for (i=0; i<=2; i++) {
      for (j=0; j<=2; j++) {
         if (i < j) {
            Matriz[i][j] = -1;
         }
         else if (i == j) {
            Matriz[i][j] = 0;
         }
         else {
            Matriz[i][j] = 1;
         }
      }
   }
```

j.
```
   for (i=0; i<=2; i++) {
      for (j=1; j<=i; j++) {
         Matriz[i][j] = 0;
      }
      for (j = i+1; j<2; j++) {
         Matriz[i][j] = 2;
      }
   }
```

3. Da mesma forma que nos vetores unidimensionais, vetores multidimensionais são armazenados em um bloco de posições consecutivas da memória, e fórmulas de translação de endereços são utilizadas para determinar a localização na memória de cada elemento do vetor. Para ilustrar considere o vetor `A` definido como `int A[2][3]`, e suponha que um inteiro possa ser armazenado em uma palavra de memória. Se `A` está alocado na memória da maneira linha a linha e `b` é o seu endereço base, a primeira linha de `A`, `A[0][0]`, `A[0][1]`, `A[0][2]`, `A[0][3]` é armazenada nas palavras `b`, `b+1`, `b+2`, `b+3`, a segunda linha nas palavras `b+4` até `b+7`, e a terceira linha nas palavras `b+8` até `b+11`.



Em geral, `A[i][j]` é armazenado na palavra `b + 4*(i-1) + (j - 1)`.

a. Dê um diagrama semelhante e a fórmula para `A[i][j]`, assumindo que a alocação é coluna a coluna.

b. De diagramas e fórmulas para as alocação linha a linha e coluna a coluna se o vertor `A` está definido como `int A[3][2]`.

c. Repita a parte *b.* para `A` definido como `float A[2][3]` onde valores `float` requerem duas palavras para armazenamento.

d. Repita *c.* para `A` definido como `float A[n][n]`.

4. (`transposta.c`) [etm] Matriz transposta, em matemática, é o resultado da troca de linhas por colunas em uma determinada matriz. Dado n matrizes, (número de linhas e colunas < 100) em diversos formatos, imprima a matriz transposta.

   Exemplo de Matriz transposta:

$$Matriz = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$MatrizTransposta = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

   Dado m matrizes de tamanho $n \times n$, imprimir 1 caso a matriz seja de permutação e 0 caso contrário.

   Entrada: A primeira linha contém um número inteiro m indicando o número de matrizes dadas, seguido de uma sequencia de dois números (p e q) contendo os tamanhos (formatos) das matrizes (p – linhas e q – colunas) e as matrizes. Obs: Os comentários não fazem parte da entrada, estão apenas tentando facilitar a compreensão do exercício.

```
transposta.in
2 // número de matrizes
3 2 // formato do matriz
0 6
-1 2
5 0
2 2 // formato da matriz
3 -2
-2 5
```

Saída: Imprimir, para cada matriz, sua transposta. Deve haver uma linha em branco, entre as matrizes transpostas.

```
transposta.sol
0 -1 5
6 2 0
3 -2
-2 5
```

5. (quadmagico.c) [etm] Dizemos que uma matriz quadrada inteira é um **quadrado mágico** se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todas iguais.

   Exemplo de quadrado mágico:

$$X = \begin{bmatrix} 8 & 0 & 7 \\ 4 & 5 & 6 \\ 3 & 10 & 2 \end{bmatrix}$$

   Dado m matrizes de tamanho $n \times n$, imprimir 1 caso sejam quadrado mágico e 0 caso contrário.

   Entrada: A primeira linha contem um inteiro m indicando o número da matrizes dadas, seguido de uma sequencia contendo o tamanho das matrizes e as matrizes. Obs: Os comentários NÃO fazem parte da entrada, estão apenas tentando facilitar a compreensão do exercício.

```
quadmagico.in
2 //quantidade de matrizes dadas
3 3 //tamanho da matriz seguido da matriz.
8 0 7
4 5 6
3 10 2
2 2 //tamanho da matriz seguido da matriz.
3 5
1 1
```

   Saída: A saída corresponde em imprimir 1 caso a matriz seja quadrado mágico e 0 ao contrário.

```
quadmagico.sol
1
0
```

O programa abaixo ilustra estrutura inicial da solução do problema.

```
// Programa: quadmagico.c
// Programador:
// Data: 21/11/2010
// Este programa lê uma matriz quadrada e verifica se ela é uma quadrado
// mágico. Ou seja, se a soma das linhas, colunas e diagonais principal
// e secundária tem o mesmo valor.
// declaração das bibliotecas utilizadas
#include<stdio.h>
#include<stdbool.h>

// início da função principal
int main(void)
{
// declaração das variáveis locais
   int mat[10][10], u, k, i, j, m, n, linha, coluna, soma;
   bool quadmagico;

// Passo A. Leia um número de matrizes

// Passo B. Repita k vezes

// Passo 1. Leia as dimensões da matriz

// Passo 2. Leia a matriz

// Passo 2.1. Leia o elemento (i,j) da matriz

// Passo 3. Compute a soma das linhas e das colunas

// Passo 3.1. Inicialize a soma das linhas e colunas

// Passo 3.2. Compute a soma da linha i e da coluna j

// Passo 3.3. Inicialize soma

// Passo 3.4. Verifique se as somas das linhas e colunas são iguais

// Passo 4. Se a soma das linhas for igual verifique as diagonais

// Passo 4.1. Calcule a soma da diagonal principal

// Passo 4.2. Calcule a soma da diagonal secundária
```

```
// Passo 4.3. Verifique se a soma das diagonais são iguais

// Passo 5. Imprima a mensagem

   return 0;
} // fim da função principal
```

6. (permutacao.c) [etm] Dizemos que uma matriz de números inteiros $n \times n$ é uma matriz de permutação se em cada linha e em cada coluna houver $n - 1$ elementos nulos e um único elemento 1. Abaixo, um exemplo de uma matriz de permutação 4×4:

$$X = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dado m matrizes de tamanho $n \times n$, imprimir 1 caso a matriz seja de permutação e 0 caso contrário.

Entrada: A primeira linha contem um inteiro k indicando o número da matrizes dadas, seguido de uma sequência contendo o tamanho das matrizes e as matrizes. Obs: Os comentários NÃO fazem parte da entrada, estão apenas tentando facilitar a compreensão do problema.

```
permutacao.in
2 //quantidade de matrizes dadas.
4 4 //tamanho da matriz seguido da matriz.
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 1
3 3 //tamanho da matriz seguido da matriz.
2 -1 0
-1 2 0
0 0 1
```

Saída: A saída consiste em imprimir 1 caso a matriz seja de permutação e 0 ao contrário.

```
permutacao.sol
1
0
```

O programa abaixo ilustra estrutura inicial da solução do problema.

```
// Programa: matrizpermutacao.c
// Programados:
// Data: 08/07/2010
```

```
// Este programa lê m matrizes de tamanho n x n, e imprime 1 caso a matriz
// seja de permutação e 0 caso contrário.
// Declaração das bibliotecas utilizadas
#include<stdio.h>
// inicio da função principal
int main(void)
{
// declaração das variáveis locais
   int mat[10][10];
   int u, k, i, j, m, n, linha, coluna;

// Passo 1. Leia a quantidade de matrizes

// Passo 2. Para as k matrizes faça

// Passo 2.1. Leia as dimensões das matrizes

// Passo 2.2. Leia a matriz (linha a linha)

// Passo 2.3 . Verifique se a matriz é de permutação (linha

// Passo 2.4. Imprima o resultado

   return 0;
}   // fim da função main
```

7. (campo minado.c) [etm] Você, cansado após tantas provas, decide relaxar e fazer um mini campo-minado para brincar no seu tempo vago. Um mini campo-minado, é composto de uma matriz, 4x4, onde o número (0) indica espaços livres e o número (1) indica que há uma bomba no local. Como no exemplo:

$$CampoMinado = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Nesse caso, os pontos $(2, 2)$, $(2, 4)$, $(3, 4)$, $(4, 1)$ e $(4, 2)$ possuem bombas.

Entrada: Será dado um campo-minado 4×4 e a seguir, um número inteiro m, equivalente a quantidade de pontos. Após isso, os m pontos serão dados e para cada ponto, verificar se o ponto é uma bomba ou não. Caso seja, imprimir 1, senão 0. Em seguida, serão dados os n campos-minados.

```
campominado.in
0 0 0 0
0 0 0 0
0 0 0 1
0 1 1 0
3
1 1
```

```
3 4
4 4
```

Saída: Imprimir 1, caso o ponto dado seja uma bomba e 0 ao contrário.

```
campominado.sol
0
1
0
```

8. O Serviço de Meteorologia Capivara registra a temperatura em três cidades distintas quatro vezes ao dia. O programador *Programador Bom da Silva* fez um programa para efetuar alguns cálculos com os dados obtidos. Para isso ele usou uma matriz 4×3 chamado `TabTemp`. Para calcular a temperatura média para cada um dos tempos em que a temperatura foi registrada, ele calculou a soma de cada linha e dividiu cada uma destas somas por 3. Ele implementou isso em `C` e utilizou o seguinte trecho de programa para executar esta computação e imprimir a temperatura média para cada tempo.

```
for (Tempo = 0; Tempo <= 3; Tempo++) {
   Soma = 0.0;
   for (Cidade = 0; Cidade <= 2; Cidade++) {
      Soma = Soma + TabTemp[Tempo][Cidade];
   }
   TempMed = Soma / 3.0;
   printf(\A Temperatura Média no tempo %d é %f\n\, Tempo, TempMed);
}
```

A implementação efetuada pelo `Programador Bom da Silva` está correta?

Altere o trecho do programa para ele calcular e imprimir a temperatura média para cada uma das três cidades.

9. Se `A` e `B` são duas matrizes $m \times n$, de elementos de um determinado tipo, sua soma é definida como segue: Se A_{ij} e B_{ij} são os elementos da entrada da i-ésima linha e j-ésima coluna de `A` e `B`, respectivamente, então $A_{ij} + B_{ij}$ é a entrada da i-ésima linha e j-ésima coluna da sua soma, a qual também é uma $m \times n$ matriz. Escreva um programa usando funções para ler duas $m \times n$ matrizes, imprimi-las, e calcular e imprimir a sua soma.

10. Na função `Convert` do programa `estoque1.c`, um laço for é usado para efetuar uma busca no vetor `NomesMarcas` para um determinado `Nome` fornecido pelo usuário. Isto significa que todo o vetor é verificado para cada `Nome`. Reescreva este segmento de programa tal que a busca termine tão logo `Nome` é encontrado no vetor. (Sugestão: Tome cuidado para não *cair no final* do tipo enumeração `TipoMarca`. Você pode achar conveniente adicionar um último valor *dummy* para este tipo.)

11. Modifique o programa `vendauto.c` tal que o total de vendas para cada modelo seja impresso no final da coluna do referente ao modelo, e o total de vendas para cada vendedor seja impresso à direita da linha referente ao vendedor.

12. A companhia de Arreios Mula Brava tem uma linha de produtos com cinco itens cujos preços de venda são R$ 100, R$ 75, R$ 120, R$ 150 e R$ 35. Existem quatro vendedores trabalhando para a companhia, e a tabela abaixo fornece o relatório de vendas referente a uma semana:

Número do Vendedor	**Número Item**				
	1	**2**	**3**	**4**	**5**
1	10	4	5	6	7
2	7	0	12	1	3
3	4	9	5	0	8
4	3	2	1	5	6

    a. Compute o total de reais faturados por cada vendedor.

    b. Se a comissão de vendas é de 10 porcento, compute o total de comissão por cada vendedor.

    c. Se cada vendedor recebe um salário fixo de R$ 200 por semana em adição aos pagamentos referentes às comissões, encontre o total do salário para cada vendedor referente à semana.

13. Escreva um programa para calcular e imprimir as primeiras dez linhas do **triângulo de Pascal**. A primeira parte do triângulo tem a forma

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
```

onde cada linha inicia e termina com 1, e cada uma das outras entradas na linha é a soma das duas entradas acima dela. Se esta forma parece muito difícil, você pode imprimir o triângulo como

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

14. Um estudo demográfico da área metropolitana ao redor da **Big Big Field** dividiu a população em três regiões, urbana, suburbana e rural. O estudo foi publicado de acordo com a seguinte tabela mostrando a migração anual de uma região para outra (os números representam porcentagens):

↱	**Urbano**	**Suburbano**	**Rural**
Urbano	1.1	0.3	0.7
Suburbano	0.1	1.2	0.3
Rural	0.2	0.6	1.3

Por exemplo, 0.3 por cento da população urbana (0.003 vezes a população atual) se muda para os subúrbios a cada ano. As entradas da diagonal representam as taxas de crescimento interno. Usando uma matriz (vetor de duas dimensões) para armazenar esta tabela e um tipo enumeração para as formas de população, escreva um programa para determinar a população de cada região após 10, 20, 30, 40 e 50 anos. Assuma que a população atual das regiões urbana, suburbana e rural são 2.1, 1.4 e 0.9 milhões, respectivamente.