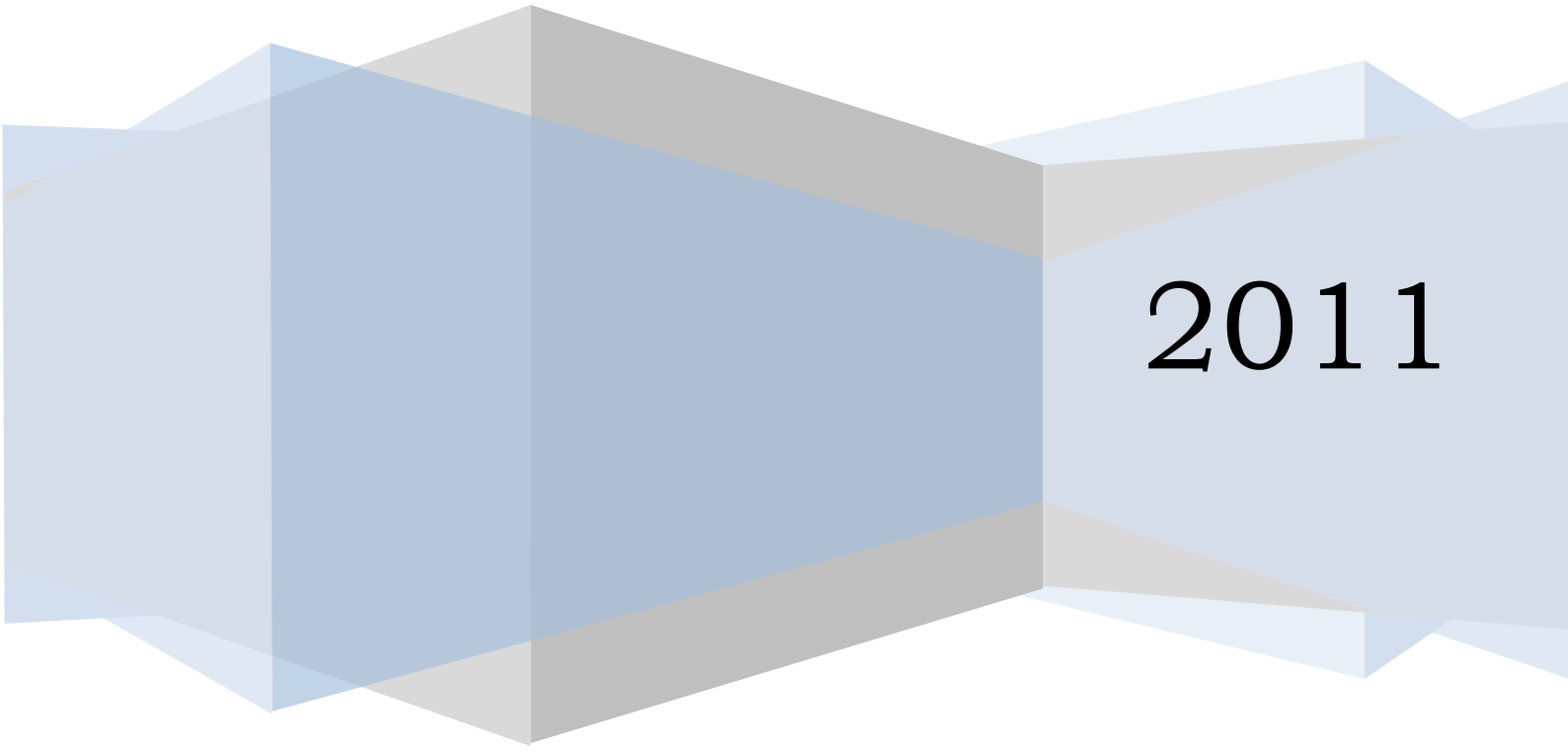
Flávio Augusto de Freitas
Introdução à Programação em Linguagem C/C++

http://flavioaf.blogspot.com

C/C++

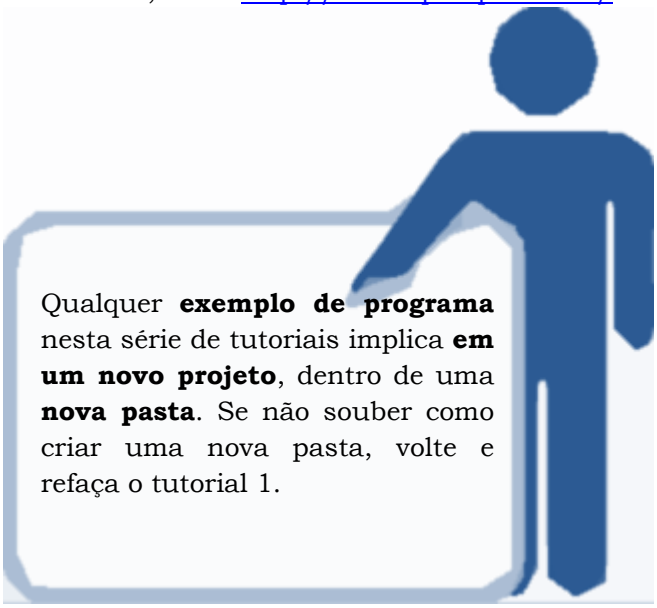
Tutorial 13 (usando Dev-C++ versão 4.9.9.2)



2011

1 Introdução

Esta série de tutoriais foi escrita usando o **Microsoft Windows 7 Ultimate** e o **Bloodshed Dev-C++** versão 4.9.9.2, que pode ser baixada em http://www.bloodshed.net. Se alguém quiser adquirir mais conhecimentos e quiser aprofundar no assunto, visite http://www.cplusplus.com/.



Qualquer **exemplo de programa** nesta série de tutoriais implica **em um novo projeto**, dentro de uma **nova pasta**. Se não souber como criar uma nova pasta, volte e refaça o tutorial 1.

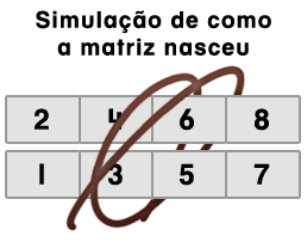
2 O que são estruturas homogêneas?

Se você perdeu as definições do que são estruturas homogêneas, então pegue os tutoriais 11 e 12, e faça-os antes de prosseguir.

3 Matrizes

Quando utilizamos vetores, não existe muita confusão. É uma linha com N espaços para determinados valores. Claro, podemos perder uns bons minutos pensando em algoritmos eficazes para ordenação, busca e outros.

Mas e quando jogamos um vetor em cima do outro, colamos e dizemos que é uma coisa só?



Você começará a ter dores de cabeça gratuitas. A iteração passa a precisar de 2 laços em cascata (no mínimo), nas funções você precisa declarar pelo menos o número de colunas que vai utilizar:

Além de receber um warning quando o valor da função não coincide com o que você está passando.

```
void yarly(int orly[][20]);

int main(int argc, char *argv[])
{
  int orly[2][2];

  yarly(orly);
}
```

Linha	Unidade	Mensagem
	E:\Blog\091026 matrizes\codigo\mai...	In function `main':
10	E:\Blog\091026 matrizes\codigo\mai...	[Warning] passing arg 1 of `yarly' from incompatible pointer type

E o Dev-C++ é meio de lua para esses warnings. As vezes somem, as vezes ficam.

3.1 Declarando e programando

A estrutura básica de uma matriz bidimensional é matriz[i][j], sendo i o número de linhas e j o número de colunas. A estrutura de uma matriz "A", 3×2:

3 linhas, 2 colunas. A primeira posição é A11 (1 linha, 1 coluna), a segunda A12 (1 linha, 2 coluna) e assim por diante.

Em C temos algo assim:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
//tipo nome[linhas][colunas];
int a[3][2];
int b[2][2] = {{1,2},{3,4}};

system("PAUSE");
return 0;
}
```

No meu caso, é uma matriz de inteiros. A inicialização sempre é opcional, mas importante. Se uma variável não é zerada, corre o risco de

pegar algum lixo da memória e ficar valendo, por exemplo, 19820.

3.2 INICIALIZAÇÃO

"Na mão", in code, você abre e fecha chaves. Dentro dessas, colocará mais um conjunto para cada linha. Esse conjunto pode ter, no máximo, o número de colunas em elementos.

```
int foo[i][j] = {{1,2,3, .. j}, .. i{}};
int fuu[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

Às vezes é prático. Às vezes não. Para preencher com valores do usuário, precisamos iterar pelas linhas e colunas da matriz, mais difícil, porém mais interessante e utilizado.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  //tipo nome[linhas][colunas];
  int a[3][2];
  int b[2][2] = {{1,2},{3,4}};

  int i,j;

  //estou na linha 0... (i=0), indo até 2 (i < 3)
  for(i=0;i<3;i++)
  {
    //vou, da coluna 0 até j-1 pedindo valores
    for(j=0;j<2;j++)
    {
      printf("Matriz[%d][%d]: ",i,j);
      scanf("%d",&a[i][j]);
    }
  }

  system("PAUSE");
  return 0;
}
```

Importante: Veja que nos loops sempre usamos o operador lógico menor. É que a matriz não vai de 1 a 3, mas sim de 0 a 2. Sempre de 0 até n – 1 (como nos vetores).

A minha técnica preferida é fixar a linha e iterar pelas colunas, mas você pode fixar a coluna e preencher suas linhas, não é crime. A jogada para isso é deixar o laço das colunas em primeiro lugar.

O algoritmo funciona assim (fixando nas linhas):

"Estou na linha 0. Vou, da coluna 0 até a coluna 4, pedindo valores."

As colunas acabam e volta no cabeçalho do primeiro laço...

"Estou na linha 1. Vou, da coluna 0 até a coluna 4, pedindo valores."

...

"Estou na linha i-1. Vou, da coluna 0 até a coluna j-1, pedindo valores."

Nas colunas é o contrário:

"Estou na coluna 0. Vou, da linha 0 até a linha 2, pedindo valores."

Passando para função

```
#include <stdio.h>
#include <stdlib.h>

void preencher(int m[][5],int linhas, int colunas);

int main(int argc, char *argv[])
{
  int a[20][20];

  preencher(a,5,2);

  system("PAUSE");
  return 0;
}
```

Repare que meu a é de 20 por 20. Por alguma razão que não entendo, o Dev-C++ insiste em dar falha quando utilizo 3 linhas e duas colunas, 5 linhas e duas colunas e outros valores "aleatórios". Então resolvi usar o conselho de minha professora em suas super aulas de C:

"Coloca mais do que vai precisar que evita qualquer tipo de erro."

Já que não vai arrancar um braço (só um fígado), fica a gambiarra ai.

A função é void, pois não retorna nada. Precisamos apenas passar a matriz, o numero de linhas e colunas.

"Ei, mas se não tem retorno, como os dados ficam na matriz?"

Quando você passa uma matriz, está passando o seu endereço de memória, na verdade. O “a” ali nunca chega a função, o que chega é um número maluco como 2292016, ou algo assim. A função “mascara” esse numero com o “m”, mas estamos atribuindo os valores para 2292016.

Duvida? Ótimo, então veja:

```
#include <stdio.h>
#include <stdlib.h>

void saoTome(int m[][10]);

int main(int argc, char *argv[])
{
  int a[20][20];

  //preencher(a,3,2);

  printf("Endereco de a:          %d\n",a);
  saoTome(a);

  system("PAUSE");
  return 0;
}

void saoTome(int m[][10])
{
  printf("Endereco de m:          %d\n",&m);
  printf("Endereco que m recebeu: %d\n",m);
}
```

Claro, você pode fazer a função devolvendo int, dando return em m para a, por consequencia ela pegaria o endereço de memória de m e os valores passariam normalmente, mas para que complicar?

3.3 EXIBINDO OS VALORES

Criamos, preenchemos, agora é exibicionismo. Digo, exibição.

A lógica é a mesma: Fixo numa linha, mostro suas colunas, mudo de linha, mostro suas colunas, até o fim. Vou mostrar direto na função.

```
#include <stdio.h>
#include <stdlib.h>

void preencher(int m[][10],int linhas, int colunas);
void saoTome(int m[][10]);
void mostrar(int m[][10],int linhas, int colunas);

int main(int argc, char *argv[])
{
  int a[20][20];

  preencher(a,3,2);

  printf("\n\n");

  mostrar(a,3,2);

  system("PAUSE");
  return 0;
}

void saoTome(int m[][10])
{
  printf("Endereco de m:          %d\n",&m);
  printf("Endereco que m recebeu: %d\n",m);
}

void preencher(int m[][10], int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       printf("matriz[%d][%d]: ",i,j);
       scanf("%d",&m[i][j]);
     }
  }
}

void mostrar(int m[][10],int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       printf("matriz[%d][%d]= %d\n",i,j,m[i][j]);
     }
  }
}
```

Pode perceber que a função é praticamente igual a de preencher. Só que tiramos o scanf e adicionamos um identificador e um valor no printf.

```
  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       printf("matriz[%d][%d]= %d\n",i,j,m[i][j]);
     }
  }
```

4 Operações entre matrizes

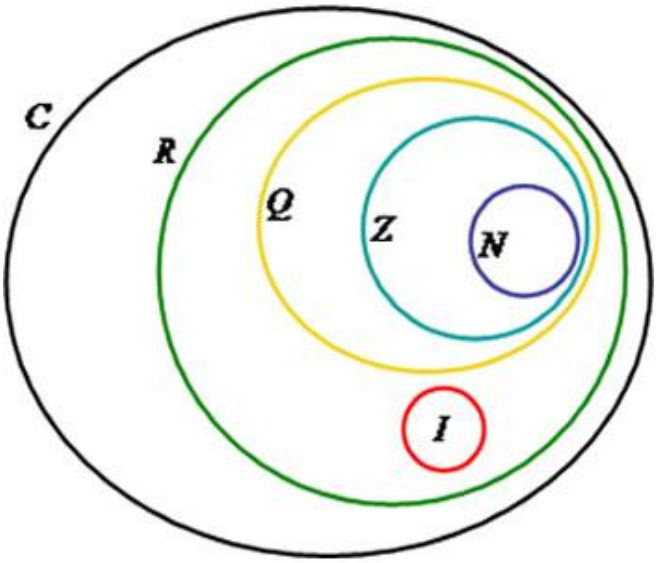
Algumas operações entre matrizes

No artigo anterior sobre matrizes, vimos como declarar, preencher in code, via input do usuário e como mostrar na tela. De quebra vimos que passamos matrizes por referência numa função, com a função sãoTomé.

O objetivo deste é realizar operações entre as duas matrizes.

4.1 Soma

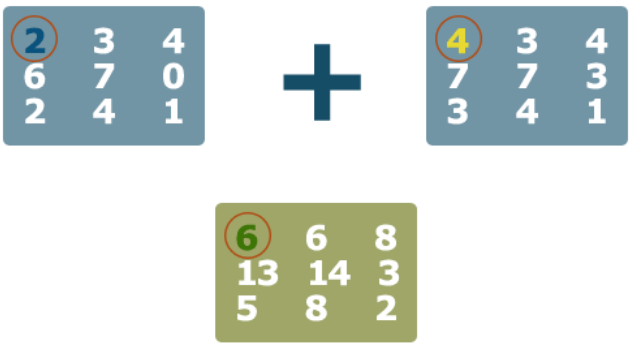
Certo. Nada difícil com números inteiros, decimais, racionais e todos os outros do círculo maldito, certo?



Círculo maldito, retirado do Brasil Escola.

Nas matrizes, nada de diferente. Apenas devemos tomar um cuidado: "Não se soma matrizes de ordens diferentes".

Sabe porquê? Quando você soma matrizes, não "soma matrizes" ao pé da letra, você soma os elementos dentro da matriz com os elementos da outra matriz na mesma posição.



2+4 | 3 + 3 | 4+4

...

É assim que a vida é. Se não é compatível, não rola. Mesma coisa (nesse caso) na matemática.

4.1.1 Mão na massa

```
void soma(int m[][10], int n[][10], int o[][10], int linhas, int colunas)
{
  int i,j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       o[i][j] = m[i][j] + n[i][j];
     }
  }
}
```

A função soma trabalha da seguinte forma:

- Preciso das duas matrizes que serão somadas (m e n)
- Preciso da matriz que vai conter o resultado da soma (o) {Opcional}
- Preciso da quantidade de linhas e colunas

Faço 2 for's (p/ acessar linhas e suas colunas). Indico entre os colchetes a linha e coluna que estou somando e faço a adição.

o[i][j] = m[i][j] + o[i][j]

=

o[linha 1][coluna 1] = m[linha 1][coluna 1] + o[linha 1][coluna 1]

=

o[1][1] = 2 + 3

No trecho acima, está suposto que na matriz M, linha 1 na coluna 1, temos o valor 2. Na matriz O, linha 1 na coluna 1, temos o valor 3.

Somamos os dois valores e colocamos na Matriz o, linha 1, coluna 1.

Rode o código todo se quiser uma visão ainda melhor:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void preencher(int m[][10],int linhas, int colunas);
void mostrar(int m[][10],int linhas, int colunas);
void soma(int m[][10], int n[][10], int o[][10], int i,
int j);

int main(int argc, char *argv[])
{
  int a[10][10], b[10][10], c[10][10];

  printf("Matriz A:\n");
  preencher(a,2,2);
  printf("Matriz B:\n");
  preencher(b,2,2);
  printf("Somando...\n");
  soma(a,b,c,2,2);

  mostrar(c,2,2);

  system("PAUSE");
  return 0;
}

void preencher(int m[][10], int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       printf("matriz[%d][%d]: ",i,j);
       scanf("%d",&m[i][j]);
     }
  }
}

void mostrar(int m[][10],int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       printf("matriz[%d][%d]= %d\n",i,j,m[i][j]);
     }
  }
}

void soma(int m[][10], int n[][10], int o[][10], int
linhas, int colunas)
{
  int i,j;

  for(i=0;i<linhas;i++)
  {
     for(j=0;j<colunas;j++)
     {
       o[i][j] = m[i][j] + n[i][j];
     }
  }
}
```

4.2 E A SUBTRAÇÃO?

Mesma coisa. Basta trocar o sinal de + pelo de -.

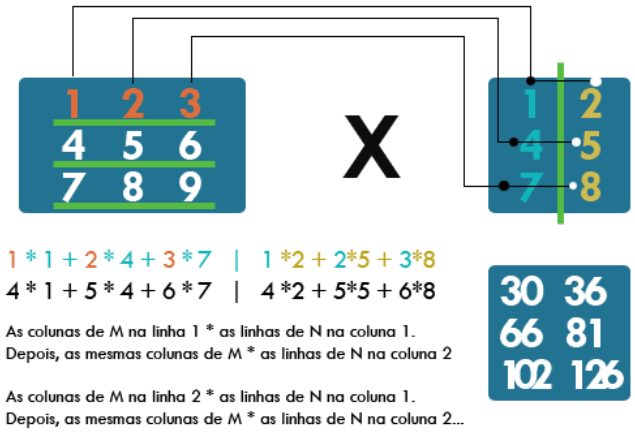
4.3 MULTIPLICAÇÃO TAMBÉM É IGUAL?

Não. Daqui pra baixo complica.

Se para somar precisamos que sejam matrizes de ordem igual, na multiplicação temos uma liberdade um pouco maior. Apenas precisamos que a quantidade de colunas da primeira seja igual a quantidade de linhas da segunda.

Porém, não multiplicamos a posição m[1][1] pela n[1][1]. Multiplicamos as colunas de M pelas linhas de N.

A matriz resultante será da seguinte ordem: linhas de M por colunas de N.



Um bom link de suporte: multiplicação de matrizes.

4.4 MÃO NA MASSA

```
void multiplicar(int m[][10], int n[][10], int o[][10],
int limite)
{
  int i,j,k;

  for(i=0;i < limite; i++)
  {
    for(j=0;j < limite; j++)
    {
      o[i][j] = 0;
```

```
      for(k=0; k < limite; k++)
      {
        o[i][j] += m[i][k] + n[k][j];
      }
    }
  }
}
```

Fixamos normalmente nas linhas. Passamos para as colunas. Já tem algo diferente: Zero a posição o[i][j], pois provavelmente ela está com um valor maluco como 29839 ou – 2783.

Dentro das colunas, abrimos o laço de K. É com ele que conseguimos multiplicas as colunas de M pelas linhas de N.

Sabemos que estamos na linha i de M, mas e para iterar pelas colunas? Usamos K.

m[i][k]

Sabemos que estamos na coluna j de N, mas e para iterar pelas linhas?

Algum chute?

n[k][j]

Precisamos passar por todas as linhas e colunas, mas usaremos as linhas e colunas em momentos diferentes nas matrizes. K resolve isso.

Código completo:

```
#include <stdio.h>
#include <stdlib.h>

void preencher(int m[][10],int linhas, int colunas);
void mostrar(int m[][10],int linhas, int colunas);

void multiplicar(int m[][10], int n[][10], int o[][10],
int limite);

int main(int argc, char *argv[])
{
  int a[10][10], b[10][10], c[10][10];

  printf("Matriz A:\n");
  preencher(a,2,2);
  printf("Matriz B:\n");
  preencher(b,2,2);
  printf("Multiplicando...\n");
  multiplicar(a,b,c,2);

  mostrar(c,2,2);

  system("PAUSE");
  return 0;
}

void preencher(int m[][10], int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
      for(j=0;j<colunas;j++)
      {
        printf("matriz[%d][%d]: ",i,j);
        scanf("%d",&m[i][j]);
      }
  }
}

void mostrar(int m[][10],int linhas, int colunas)
{
  int i, j;

  for(i=0;i<linhas;i++)
  {
      for(j=0;j<colunas;j++)
      {
        printf("matriz[%d][%d]= %d\n",i,j,m[i][j]);
      }
  }
}

void multiplicar(int m[][10], int n[][10], int o[][10],
int limite)
{
  int i,j,k;

  for(i=0;i < limite; i++)
  {
    for(j=0;j < limite; j++)
    {
      o[i][j] = 0;

      for(k=0; k < limite; k++)
      {
        o[i][j] += m[i][k] + n[k][j];
      }
    }
  }
}
```

4.5 IMPORTANTE

A matriz de saída será de ordem Mi, Nj.

Se tivermos M[2][3] X N[3][4]:

O[2][4]

4.6 Fechando o assunto

Essas são as operações básicas. Espero na próxima parte abordar sobre alocação dinâmica de matrizes. Ou seja, as matrizes não vão ser declaradas com M[20][20], deixando espaço a toa, mas sim com o pedido do usuário.

5 Exercícios propostos

a) Considere A e B duas matrizes NxM. Faca um programa para calcular a matriz C, resultante da soma da matriz A com a mariz B. Imprimir a matriz C.*/

b) Uma matriz quadrada inteira é chamada de "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 todos iguais. Exemplo: A matriz abaixo representa um quadrado mágico:

```
| 8  0  7 |
| 4  5  6 |
| 3 10  2 |
```

Escreva um programa que verifica se uma matriz de n linhas e n colunas representa um quadrado mágico.

c) Faça um programa que leia uma matriz 10 X 10 e calcule e escreva a média dos elementos localizados na área marcada com '*'.

```
a*********
aa********
aaa*** ****
aaaa******
aaaaa*****
aaaaaa****
aaaaaaa***
aaaaaaaa**
aaaaaaaaa*
```

d) Seja a seguinte declaração float numeros[LINHA][COLUNA].

Escrever um programa capaz de:

- ler os elementos da matriz.
- identificar o número de elementos iguais a zero em cada uma das linhas.
- identificar o número de elementos iguais a zero em cada uma das colunas.
- identificar o número de elementos pares em determinada linha (lida) ****
- identificar o número de elementos pares em determinada coluna (lida)****
- calcular a média aritmética dos elementos de cada uma das linhas, armazenando esses valores em um vetor.
- identificar a linha que tem a maior media de seus elementos.

Imprimir todos os resultados.

e) Preencher uma matriz de ordem 3 e, efetuar a soma dos valores da diagonal principal.

f) Preencher uma matriz de ordem 3 e, efetuar a soma dos valores da diagonal secundaria.

g) Criar um programa em Linguagem C que leia os elementos de uma matriz inteira 3x3 e imprima a soma dos elementos acima da diagonal principal.

h) Criar um programa em Linguagem C que leia os elementos de uma matriz inteira 3x3 e imprima a soma dos elementos acima da diagonal secundaria.

i) Criar um programa em Linguagem C que leia os elementos de uma matriz inteira 3x3 e escreva somente os elementos acima da diagonal principal.

j) Criar um programa em Linguagem C que leia os elementos de uma matriz inteira 3x3 e escreva somente os elementos acima da diagonal secundaria.

6 Terminamos

Terminamos por aqui. O que está esperando, saia do Dev-C++ e corra para pegar o próximo tutorial em http://flavioaf.blogspot.com. Siga o blog, assim você fica sabendo das novidades no momento em que forem publicadas. Seguindo o blog você se mantém sempre atualizado de qualquer lançamento novo.