MC-102 — Aula 19
Matrizes e Vetores Multidimensionais

Instituto de Computação – Unicamp

28 de Setembro de 2015

Roteiro

Matrizes e Vetores Multidimensionais

- Matrizes e Vetores Multidimensionais são generalizações de vetores simples vistos anteriormente.
- Suponha por exemplo que devemos armazenar as notas de cada aluno em cada laboratório de MC102.
- Podemos alocar 15 vetores (um para cada lab.) de tamanho 50 (tamanho da turma), onde cada vetor representa as notas de um laboratório específico.
- Matrizes e Vetores Multidimensionais permitem fazer a mesma coisa mas com todas as informações sendo acessadas por um nome em comum (ao invés de 15 nomes distintos).

Declarando uma matriz

```
<tipo> nome_da_matriz [<linhas>] [<colunas>]
```

- Uma matriz possui *linhas* $\times$ *colunas* variáveis do tipo <tipo>.
- As linhas são numeradas de 0 a *linhas* $-$ 1.
- As colunas são numeradas de 0 a *colunas* $-$ 1.

Exemplo de declaração de matriz

```
int matriz [4][4];
```

	0	1	2	3
0				
1				
2				
3				

Acessando uma matriz

- Em qualquer lugar onde você escreveria uma variável no seu programa, você pode usar um elemento de sua matriz, da seguinte forma:

```
nome_da_matriz [<linha>] [<coluna>]
```

  Ex: `matriz [1][10]` — Refere-se a variável na 2^a linha e na 11^a coluna da matriz.

- **Lembre-se que, assim como vetores, a primeira posição em uma determinada dimensão começa no índice 0.**
- O compilador não verifica se você utilizou valores válidos para a linha e para a coluna.

Declarando um Vetor Multidimensional

$<\texttt{tipo}>$ `nome` $[< dim_1 >]\ [< dim_2 >] \ldots\ [< dim_N >]$

- Este vetor possui $dim_1 \times dim_2 \times \cdots \times \dim_N$ variáveis do tipo $<\texttt{tipo}>$
- Cada dimensão é numerada de 0 a $dim_i - 1$

Declarando um Vetor Multidimensional

- Você pode criar por exemplo uma matriz para armazenar a quantidade de chuva em um dado dia, mês e ano, para cada um dos últimos 3000 anos:

```
double chuva[31][12][3000];

chuva[23][3][1979] = 6.0;
```

Vetores multi-dimensionais e funções

- Ao passar um **vetor simples** como parâmetro, não é necessário fornecer o seu tamanho na declaração da função.
- Quando o **vetor é multi-dimensional** a possibilidade de não informar o tamanho na declaração se restringe à primeira dimensão apenas.

```
void mostra_matriz(int mat[][10], int n_linhas) {
   ...
}
```

Vetores multi-dimensionais e funções

- Pode-se criar uma função deixando de indicar a primeira dimensão:

```
void mostra_matriz(int mat[][10], int n_linhas) {
  ...
}
```

- Ou pode-se criar uma função indicando todas as dimensões:

```
void mostra_matriz(int mat[5][10], int n_linhas) {
  ...
}
```

- Mas não pode-se deixar de indicar outras dimensões (exceto a primeira):

```
void mostra_matriz(int mat[5][], int n_linhas) {
  //ESTE NÃO FUNCIONA
  ...
}
```

Vetores multi-dimensionais em funções

```
void mostra_matriz(int mat[][10], int n_linhas) {
  int i, j;

  for (i = 0; i < n_linhas; i++) {
    for (j = 0; j < 10; j++)
      printf("%2d ", mat[i][j]);
    printf("\n");
  }
}

int main() {
  int mat[][10] = { { 0,  1,  2,  3,  4,  5,  6,  7,  8,  9},
                    {10, 11, 12, 13, 14, 15, 16, 17, 18, 19},
                    {20, 21, 22, 23, 24, 25, 26, 27, 28, 29},
                    {30, 31, 32, 33, 34, 35, 36, 37, 38, 39},
                    {40, 41, 42, 43, 44, 45, 46, 47, 48, 49},
                    {50, 51, 52, 53, 54, 55, 56, 57, 58, 59},
                    {60, 61, 62, 63, 64, 65, 66, 67, 68, 69},
                    {70, 71, 72, 73, 74, 75, 76, 77, 78, 79}};

  mostra_matriz(mat, 8);
  return 0;
}
```

Vetores multi-dimensionais em funções

Lembre-se que vetores (multi-dimensionais ou não) são alterados quando passados como parâmetro em uma função

```
void   teste (int mat[2][2]) {
  int i, j;

  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++){
         mat[i][j] = -1;
    }
  }
}

int main() {
  int mat[2][2] = { { 0, 1},
                    { 2, 3} };

  teste(mat);
  //Neste ponto mat tem que valores em suas posições???

  return 0;
}
```

Exemplo

Criar uma aplicação com operações básicas sobre matrizes quadradas:

- Soma de 2 matrizes com dimensões $n \times n$.
- Subtração de 2 matrizes com dimensões $n \times n$.
- Cálculo da transposta de uma matriz de dimensão $n \times n$.
- Multiplicação de 2 matrizes com dimensões $n \times n$.

Exemplo

Primeiramente criamos uma função para fazer a leitura de uma matriz:

```
void leMatriz(double m[MAX][MAX], int n){
  int i, j;

  printf("Lendo dados da matriz, linha por linha\n");
  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      scanf("%lf", &m[i][j]);
    }
  }
}
```

OBS: **MAX** é uma constante inteira definida previamente.

Exemplo

Depois criamos uma função para fazer a impressão de uma matriz:

```
void imprimeMatriz(double m[MAX][MAX], int n){
  int i, j;

  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      printf("%.2lf, ", m[i][j]);
    }
    printf("\n");
  }
}
```

Exemplo

Dentre as funcionalidades, vamos implementar a multiplicação:

- Vamos multiplicar duas matrizes M_1 e M_2 (de dimensão $n \times n$).
- O resultado será uma terceira matriz M_3.
- Lembre-se que uma posição (i, j) de M_3 terá o produto interno do vetor linha i de M_1 com o vetor coluna j de M_2:

$$M_3[i, j] = \sum_{k=0}^{n-1} M_1[i, k] \cdot M_2[k, j]$$

Exemplo

Em C temos:

```
void multiplica(double m1[MAX][MAX], double m2[MAX][MAX], double m3[MAX][MAX],
                                                                    int n){
  int i, j, k;

  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      m3[i][j] = 0;
      for(k=0; k<n; k++){
         m3[i][j] = m3[i][j] + (m1[i][k] * m2[k][j]);
      }
    }
  }

}
```

Exemplo

Um código usando as funções anteriores:

```
#include <stdio.h>
#define MAX 100
void imprimeMatriz(double m[MAX][MAX], int n);
void leMatriz(double m[MAX][MAX], int n);
void multiplica(double m1[MAX][MAX], double m2[MAX][MAX], double m3[MAX][MAX], int

int main(void){
  int n;
  double m1[MAX][MAX], m2[MAX][MAX], m3[MAX][MAX];

  printf("Qual tamanho das matrizes (Max 100)?");
  scanf("%d", &n);

  leMatriz(m1, n);
  leMatriz(m2, n);
  printf("M1\n");
  imprimeMatriz(m1, n);
  printf("M2\n");
  imprimeMatriz(m2, n);
  multiplica(m1, m2, m3, n);
  printf("Multiplicacao\n");
  imprimeMatriz(m3, n);
  return 0;
}
```

Exercícios

Escreva um programa que leia todas as posições de uma matriz 10×10.
O programa deve então exibir o número de posições não nulas na matriz.

Exercícios

- Escreva um programa que lê todos os elementos de uma matriz 4×4 e mostra a matriz e a sua transposta na tela.

Matriz

$$\begin{bmatrix} 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

Transposta

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Exercício

- Complete o código do programa com operações de matrizes, para conter:
  - Um menu para escolher a operação a ser realizada.
  - Uma função para cada uma das operações faltantes.

Inicialização de Matrizes

- No caso de matrizes, usa-se chaves para delimitar as linhas:

Exemplo

```
int vet[2][5] = { {10, 20, 30, 40, 50} , {60, 70, 80, 90, 100 } } ;
```

- No caso tridimensional, cada índice da primeira dimensão se refere a uma matriz inteira:

Exemplo

```
int v3[2][3][4] = {
{ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} },
{ {0, 0, 0, 0}, {5, 6, 7, 8}, {0, 0, 0, 0} },
};
```

Inicialização de Matrizes

```
int main(){
  int i,j,k;
  int v1[5] = {1,2,3,4,5};
  int v2[2][3] = { {1,2,3}, {4,5,6}};
  int v3[2][3][4] = {
    { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} },
    { {0, 0, 0, 0}, {5, 6, 7, 8}, {0, 0, 0, 0} }
  };

  .
  .
  .
  .
}
```

Inicialização de Matrizes

```
int main(){
  .
  .
  .
  char st1[100] = "olha que coisa mais linda, mais cheia de graça";

  printf("\n\n v1 \n");
  for(i=0; i<5; i++){
    printf("%d, ",v1[i]);
  }
  printf("\n\n v2 \n");
  for(i=0; i<2; i++){
    for(j=0; j<3; j++){
      printf("%d, ",v2[i][j]);
    }
    printf("\n");
  }
  .
  .
  .
}
```

Inicialização de Matrizes

```
int main(){
  .
  .
  .
  printf("\n\n v3 \n");
  for(i=0; i<2; i++){
    for(j=0; j<3; j++){
      for(k=0; k<4; k++){
        printf("%d, ",v3[i][j][k]);
      }
    printf("\n");
    }
    printf("\n");
  }


  printf("%s",st1);
}
```

Linearização de Índices

- Podemos usar sempre vetores simples para representar matrizes (na prática o compilador faz isto por você).
- Ao declarar uma matriz como **int mat[3][4]**, sabemos que serão alocados 12 posições de memória associadas com a variável **mat**.
- Poderíamos simplesmente criar **int mat[12]**. Mas perdemos a simplicidade de uso dos índices em forma de matriz.
  - Você não mais poderá escrever **mat[1][3]** por exemplo.

Linearização de Índices

- A *linearização de índices* é justamente a representação de matrizes usando-se um vetor simples.
- Mas devemos ter um padrão para acessar as posições deste vetor como se sua organização fosse na forma de matriz.

Linearização de Índices

- Considere o exemplo:
  int mat[12]; // ao invés de int mat[3][4]
- Fazemos a divisão por linhas como segue:
  - Primeira linha: **mat[0]** até **mat[3]**
  - Segunda linha: **mat[4]** até **mat[7]**
  - Terceira linha: **mat[8]** até **mat[11]**
- Para acessar uma posição $[i][j]$ usamos:
  - **mat[i*4 + j];**
    onde $0 \leq i \leq 2$ e $0 \leq j \leq 3$.

Linearização de Índices

- De forma geral, seja matriz **mat[n*m]**, representando **mat[n][m]**.
- Para acessar a posição correspondente à $[i][j]$ usamos:
  - **mat[i*m + j];**
    onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$.
- Note que i pula de blocos de tamanho m, e j indexa a posição dentro de um bloco.

Linearização de Índices

- Podemos estender para mais dimensões. Seja matriz **mat[n*m*q]**, representando **mat[n][m][q]**.
  - As posições de 0 até $(m * q) - 1$ são da primeira matriz.
  - As posições de $(m * q)$ até $(2 * m * q) - 1$ são da segunda matriz.
  - Etc...
- De forma geral, seja matriz **mat[n*m*q]**, representando **mat[n][m][q]**.
- Para acessar a posição correspondente à $[i][j][k]$ usamos:
  - **mat[i*m*q + j*q + k];**

Linearização de Índices

```
int main(){
  int mat[40]; //representando mat[5][8]
  int i,j;

  for(i=0; i<5; i++)
    for(j=0;j<8; j++)
      mat[i*8 + j] = i*j;

  for(i=0; i<5; i++){
    for(j=0;j<8; j++)
      printf("%d, ",mat[i*8 + j]);
    printf("\n");
  }

}
```