MC-102 — Aula 18
Matrizes e Vetores Multi-Dimensionais

Instituto de Computação – Unicamp

4 de Maio de 2015

Roteiro

1. Matrizes e Vetores Multi-Dimensionais
   - Inicialização de Vetores
   - Vetores multi-dimensionais e funções
2. Exemplo
3. Exercícios
4. Informações Extras: Representação de Matrizes por Linearização

Inicialização de Vetores

- Em algumas situações, ao criarmos uma matriz, pode ser útil atribuir valores já na sua criação.
- No caso de vetores, a inicialização é simples: Basta atribuir uma lista de valores constantes de mesmo tipo separados por vírgulas e entre chaves.

Exemplo

```
int vet[5] = {10, 20, 30, 40, 50};
```

- No caso de strings, você pode atribuir diretamente uma constante string.

Exemplo

```
char st1[100] = "sim isto é possível";
```

Inicialização de Vetores

- 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 Vetores

```
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} }
  };
 char st1[100] = "olha que coisa mais linda, mais cheia de graça";

  .
  .
  .
  .
}
```

Inicialização de Vetores

```
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 Vetores

```
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);
}
```

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 todas as suas posições com valor -1

  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 mat[][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", &mat[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 mat[][MAX], int n){
  int i, j;
  printf("dados da matriz (linha por linha)\n");

  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      printf("%.2lf  ", mat[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 mat1[][MAX], double mat2[][MAX], double resp[][MAX], int n){
  int i, j, k;

  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      resp[i][j] = 0;
      for(k=0; k<n; k++){
           resp[i][j] = resp[i][j] + (mat1[i][k]*mat2[k][j]);
      }
    }
  }
}
```

Exemplo

Um código usando as funções anteriores:

```
#include <stdio.h>

#define MAX 100

void leMatriz(double mat[][MAX], int n);
void imprimeMatriz(double mat[][MAX], int n);
void multiplica(double mat1[][MAX], double mat2[][MAX], double resp[][MAX], int n);

int main() {
  double mat1[MAX][MAX], mat2[MAX][MAX], mat3[MAX][MAX];
  int n;

  printf("Qual dimensão das matrizes: ");
  scanf("%d", &n);
  leMatriz(mat1, n);
  leMatriz(mat2, n);
  multiplica(mat1, mat2, mat3, n);
  imprimeMatriz(mat1, n);
  imprimeMatriz(mat2, n);
  imprimeMatriz(mat3, n);

  return 0;
}
```

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.

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");
  }

}
```