MC-102 — Aula 12
Matrizes e Representação por Linearização de Índices

Instituto de Computação – Unicamp

10 de Abril de 2012

Roteiro

Matrizes

Suponha que queremos ler as notas de 4 provas para cada aluno e então calcular a média do aluno e a média da classe. O tamanho máximo da turma é de 50 alunos.

Solução

Criar 4 vetores de tamanho 50 cada. Cada vetor representa as notas dos alunos de uma prova.

```
float nota0[50],nota1[50],nota2[50],nota3[50];
```

Matrizes

- Agora suponha que estamos trabalhando com no máximo 100 provas. Seria muito cansativo criar 100 vetores, um para cada prova.
- Para resolver esse problema podemos utilizar matrizes. Uma matriz é um vetor (ou seja, um conjunto de variáveis de mesmo tipo) que possui duas ou mais dimensões, resolvendo para sempre essa questão.

Declarando uma matriz

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

- Uma matriz possui *linhas* × *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 uma matriz de múltiplas dimensões

<tipo> nome_da_matriz [< dim_1 >] [< dim_2 >] ... [< dim_N >]

- Essa matriz possui $dim_1 \times dim_2 \times \cdots \times \text{dim}_N$ variáveis do tipo <tipo>
- Cada dimensão é numerada de 0 a $dim_i - 1$

Declarando uma matriz de múltiplas dimensões

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

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

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

Exemplos com Matrizes

Lendo uma matriz 4×4 do teclado:

```
/*Leitura*/
for (i = 0; i < 4; i++)
  for (j = 0; j < 4; j++) {
    printf ("Matriz[%d][%d]: ", i, j);
    scanf ("%d", &matriz[i][j]);
  }
```

Exemplos com Matrizes

Escrevendo uma matriz 4×4 na tela:

```
/*Escrita*/
for (i = 0; i < 4; i++) {
  for (j = 0; j < 4; j++)
    printf ("%d ", matriz[i][j]);
  printf ("\n");
}
```

Exemplos com Matrizes

- Ler duas matrizes 4×4 e calcular a soma das duas.

Exemplos com Matrizes

```
int main(){
  double mat1[3][3], mat2[3][3], mat3[3][3];
  int i,j;

  printf("\n **** Dados da Matriz 1 ****\n");
  for(i=0; i<3; i++)
    for(j=0; j<3; j++){
      printf("Entre com dado da linha %d - coluna %d: ", i, j);
      scanf("%lf", &mat1[i][j]);
    }
  printf("\n **** Dados da Matriz 2 ****\n");
  for(i=0; i<3; i++)
    for(j=0; j<3; j++){
      printf("Entre com dado da linha %d - coluna %d: ", i, j);
      scanf("%lf", &mat2[i][j]);
    }
    .....
    .....
```

Exemplos com Matrizes

```
int main(){
  double mat1[3][3], mat2[3][3], mat3[3][3];
  int i,j;

    .....
    .....
    .....

  for(i=0; i<3; i++)
    for(j=0; j<3; j++){
      mat3[i][j] = mat1[i][j] + mat2[i][j];
    }

  printf("\n **** Dados da Matriz 3 ****\n");
  for(i=0; i<3; i++){
    for(j=0; j<3; j++)
      printf("%lf, ", mat3[i][j]);
    printf("\n");
  }
}
```

Inicialização de Matrizes

- 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 Matrizes

- No caso de matrizes, use-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 primeiro índice é 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} }
  };
 char st1[100] = "olha que coisa mais linda, mais cheia de graça";

   .
   .
   .
   .
}
```

Inicialização de Matrizes

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

  printf("\n\nvet1\n");
  for(i=0; i<5; i++)
    printf("%d, ",v1[i]);

  printf("\n\nvet2\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\nvet3\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");
  }
}
```

Exercícios

Escreva um programa que leia todas as posições de uma matriz 10×10. Em seguida, mostra o índice da linha e o índice da coluna e o valor das posições não nulas. No final, exibe o número de posições não nulas.

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

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