MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2015

Roteiro

Matrizes

- Suponha que queremos ler as notas de 4 questões (de uma prova) para cada aluno e então calcular a média do aluno e a média da turma, sendo o tamanho máximo da turma de 50 alunos.
- Poderíamos criar 4 vetores, cada um deles de tamanho 50.
- Cada vetor armazenaria as notas dos alunos em uma questão.

```
float questao1[50], questao2[50], questao3[50], questao4[50];
```

Matrizes

- Agora suponha que o número de questões possa ser igual a 100.
- Tornaria-se inconveniente criar 100 vetores diferentes, um para cada questão.
- Para resolver esse problema, poderíamos utilizar matrizes.
- Uma matriz é um vetor (ou seja, um conjunto de variáveis de mesmo tipo) que possui duas ou mais dimensões.

Declarando uma matriz bidimensional

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

- Uma matriz bidimensional 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 uma matriz bidimensional

```
int matriz[5][4];
```

	0	1	2	3
0				
1				
2				
3				
4				

Acesso aos elementos de uma matriz bidimensional

- O acesso a um elemento da matriz bidimensional pode ser feito da seguinte forma:

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

- Exemplo: `matriz[1][3]` refere-se ao elemento na 2ª linha e na 4ª 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 o programador utiliza inteiros válidos como índices para a linha ou para a coluna.

Declarando uma matriz de múltiplas dimensões

```
<tipo> nome_da_matriz[<dim_1>][<dim_2>]...[<dim_n>];
```

- Essa matriz possui $\text{dim}_1 \times \text{dim}_2 \ldots \times \text{dim}_n$ variáveis do tipo `<tipo>`.
- Cada dimensão é indexada de 0 a dim_i - 1.

Declarando uma matriz de múltiplas dimensões

- Podemos 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;
```

- O código acima indica que no dia 24/04/1980 choveu 6mm.

Leitura de uma matriz 5×4 a partir da entrada padrão

```
/* Leitura de uma matriz 5 x 4 */
for (i = 0; i < 5; i++)
  for (j = 0; j < 4; j++) {
    printf("Matriz[%d][%d]: ", i, j);
    scanf("%d", &matriz[i][j]);
  }
```

Escrita de uma matriz 5×4 na saída padrão

```
/* Escrita de uma matriz 5 x 4 */
for (i = 0; i < 5; i++) {
  for (j = 0; j < 4; j++)
    printf("%d ", matriz[i][j]);
  printf("\n");
}
```

Soma de duas matrizes 5×4

```
#include <stdio.h>

#define LINHAS 5
#define COLUNAS 4

int main() {
  double mat1[LINHAS][COLUNAS], mat2[LINHAS][COLUNAS], mat3[LINHAS][COLUNAS];
  int i, j;

  printf("*** Leitura dos dados da Matriz 1 ***\n");
  for (i = 0; i < LINHAS; i++)
    for (j = 0; j < COLUNAS; j++) {
      printf("Entre com o valor da linha %d e coluna %d: ", i, j);
      scanf("%lf", &mat1[i][j]);
    }

  printf("*** Leitura dos dados da Matriz 2 ***\n");
  for (i = 0; i < LINHAS; i++)
    for (j = 0; j < COLUNAS; j++) {
      printf("Entre com valor da linha %d e coluna %d: ", i, j);
      scanf("%lf", &mat2[i][j]);
    }
  ...
```

Soma de duas matrizes 5×4

```
  ...

  printf("*** Somando os valores correspondentes das duas matrizes ***\n");
  for (i = 0; i < LINHAS; i++)
    for (j = 0; j < COLUNAS; j++) {
      mat3[i][j] = mat1[i][j] + mat2[i][j];
    }

  printf("*** Imprimindo os dados da Matriz 3 ***\n");
  for (i = 0; i < LINHAS; i++) {
    for (j = 0; j < COLUNAS; j++)
      printf("%f, ", mat3[i][j]);
    printf("\n");
  }

  return 0;
}
```

Matrizes de caracteres

- Numa matriz bidimensional, podemos considerar cada uma das linhas como um vetor, ou seja, uma matriz bidimensional é um vetor de vetores.
- Então, podemos considerar uma matriz bidimensional de caracteres (`char`) como um vetor de strings.
- Sendo assim, podemos, por exemplo, ler ou escrever uma linha inteira de uma matriz bidimensional de caracteres com os comandos `scanf` ou `printf`, respectivamente.
- Neste caso, é importante lembrar que o caractere `'\0'` deve ser adequadamente armazenado na matriz.
- Considere o seguinte exemplo:
  - Ler, armazenar e imprimir uma lista de palavras. Para cada palavra, também imprimir o seu tamanho.

Lista de palavras

```
#include <stdio.h>

#define NUM_MAX 50
#define TAM_MAX 20

int main() {
  char palavras[NUM_MAX][TAM_MAX + 1];
  int i, tamanho, n;

  do {
    printf("Entre com o numero de palavras: ");
    scanf("%d", &n);
  } while ((n < 1) || (n > NUM_MAX));

  ...
```

Lista de palavras

```
  ...

  printf("Entre com as %d palavras:\n", n);
  for (i = 0; i < n; i++)
    scanf("%s", palavras[i]);

  printf("Lista de palavras fornecidas:\n");
  for (i = 0; i < n; i++) {
    tamanho = 0;
    while (palavras[i][tamanho])
      tamanho++;

    printf("%s (tamanho: %d)\n", palavras[i], tamanho);
  }

  return 0;
}
```

Lista de palavras

```
  ...

  printf("Entre com as %d palavras:\n", n);
  for (i = 0; i < n; i++)
    fgets(palavra[i], TAM_MAX + 1, stdin);

  printf("Lista de palavras fornecidas:\n");
  for (i = 0; i < n; i++) {
    tamanho = 0;
    while (palavras[i][tamanho])
      tamanho++;

    printf("%s (tamanho: %d)\n", palavras[i], tamanho);
  }

  return 0;
}
```

Inicialização de vetores

- Em algumas situações, ao criarmos um vetor ou uma matriz, pode ser útil atribuir valores já na sua declaração.
- No caso de vetores, a inicialização é simples: basta atribuir uma lista de valores constantes de tipo correspondente separados por vírgulas e entre chaves. Exemplo:

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

- No caso de strings, pode-se atribuir diretamente uma constante (entre aspas duplas). Exemplo:

```
char str[100] = "Live long and prosper!";
```

Inicialização de matrizes

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

```
int vet[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} };
```

- No caso de matrizes tridimensionais, cada um dos elementos da primeira dimensão é uma matriz bidimensional. 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 e matrizes

```
#include <stdio.h>

int main() {
  int i, j, k;

  int vet1[5] = {1, 2, 3, 4, 5};

  int vet2[2][3] = { {1, 2, 3}, {4, 5, 6} };

  int vet3[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 str[100] = "May the Force be with you!";

  ...
```

Inicialização de vetores e matrizes

```
...

printf("Vetor:\n");
for (i = 0; i < 5; i++)
  printf("%d, ", vet1[i]);

printf("Matriz bidimensional:\n");
for (i = 0; i < 2; i++) {
  for (j = 0; j < 3; j++)
    printf("%d, ", vet2[i][j]);
  printf("\n");
}

...
```

Inicialização de vetores e matrizes

```
  ...

  printf("Matriz tridimensional:\n");
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
      for (k = 0; k < 4; k++) {
        printf("%d, ", vet3[i][j][k]);
      }
      printf("\n");
    }
    printf("\n");
  }

  printf("String: %s\n", str);

  return 0;
}
```

Linearização de índices

- Podemos usar vetores simples para representar matrizes.
- Na prática, o compilador faz isto automaticamente.
- Ao declarar uma matriz como `int mat[3][4]`, sabemos que serão alocadas 12 posições de memória associadas com a variável `mat`.
- Poderíamos simplesmente criar um vetor `int mat[12]`. Entretanto, perderíamos a simplicidade de uso dos índices em forma de matriz.
  - Você não mais poderia escrever `mat[1][3]`, por exemplo.

Linearização de índices

- A linearização de índices é justamente a representação de matrizes usando-se um vetor simples.
- Precisamos de um padrão para acessar as posições deste vetor de forma a simular sua organização como uma matriz.

Linearização de índices

- Considere o seguinte exemplo:

```
int mat[12]; /* ao inves de int mat[3][4] */
```

- Podemos fazer a divisão por linhas da seguinte forma:
  - As posições de `mat[0]` até `mat[3]` correspondem à primeira linha.
  - As posições de `mat[4]` até `mat[7]` correspondem à segunda linha.
  - As posições de `mat[8]` até `mat[11]` correspondem à terceira linha.
- Para acessar uma posição correspondente à `mat[i][j]`, podemos usar `mat[i*4 + j]`, tal que $0 \leq i \leq 2$ e $0 \leq j \leq 3$.

Linearização de índices

- De forma geral, seja o vetor `mat[n*m]`, representando a matriz `mat[n][m]`.
- Para ter acesso à posição correspondente à `mat[i][j]`, podemos usar `mat[i*m + j]`, tal que $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$.
- Note que `i` salta blocos de tamanho `m` (correspondente a uma linha) e `j` indexa a posição dentro de um bloco (linha).
- Devido a forma como uma matriz é organizada na memória, é mais eficiente percorrer uma matriz linha a linha do que coluna a coluna.

Linearização de índices

- Podemos estender esta representação para mais dimensões.
- Seja o vetor `mat[n*m*q]` representando a matriz `mat[n][m][q]`.
- Podemos fazer a divisão em matrizes bidimensionais ($m \times q$) da seguinte forma:
  - As posições de `mat[0]` até `mat[(m*q) - 1]` correspondem à primeira matriz bidimensional.
  - As posições de `mat[m*q]` até `mat[(2*m*q) - 1]` correspondem à segunda matriz bidimensional.
  - E assim por diante...
- Para ter acesso à posição correspondente à `mat[i][j][k]`, podemos usar `mat[i*m*q + j*q + k]`, tal que $0 \leq i \leq n-1$, $0 \leq j \leq m-1$ e $0 \leq k \leq q-1$.

Linearização de índices

```
#include <sdtdio.h>

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");
  }

  return 0;
}
```

Exercícios

Escreva um programa que leia todos os elementos de uma matriz $n \times m$ e imprima a matriz e a sua transposta. Exemplo:

Matriz

$$\begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \\ 51 & 52 & 53 & 54 \end{bmatrix}$$

Transposta

$$\begin{bmatrix} 11 & 21 & 31 & 41 & 51 \\ 12 & 22 & 32 & 42 & 52 \\ 13 & 23 & 33 & 43 & 53 \\ 14 & 24 & 34 & 44 & 54 \end{bmatrix}$$

Exercícios

Escreva um programa que leia 2 matrizes quadradas ($n \times n$) e imprima as matrizes e a soma delas. Exemplo:

$$\underset{A}{\begin{bmatrix} 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \end{bmatrix}} + \underset{B}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 0 & 1 & 0 & 4 & 3 \\ 1 & 2 & 1 & 3 & 4 \\ 0 & 1 & 0 & 2 & 3 \\ 2 & 3 & 2 & 4 & 5 \\ 3 & 4 & 3 & 5 & 6 \end{bmatrix}}$$

Exercícios

Escreva um programa que leia 2 matrizes quadradas ($n \times n$) e imprima as matrizes e o produto delas. Exemplo:

$$\underset{A}{\begin{bmatrix} 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 14 & 14 & 14 & 14 & 14 \\ 14 & 14 & 14 & 14 & 14 \\ 14 & 14 & 14 & 14 & 14 \\ 14 & 14 & 14 & 14 & 14 \\ 14 & 14 & 14 & 14 & 14 \end{bmatrix}}$$

Exercícios

Escreva um programa que leia uma matriz quadrada $n \times n$ e verifique se ela é uma matriz triangular inferior. Exemplo:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 9 & 0 & 2 & 3 \end{bmatrix}$$

Exercícios

Escreva um programa que leia uma matriz quadrada $n \times n$ e verifique se ela é uma matriz triangular superior. Exemplo:

$$\begin{bmatrix} 1 & 0 & 8 & 9 & 8 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Exercícios

Escreva um programa que leia uma matriz quadrada $n \times n$ e verifique se ela é uma matriz diagonal. Exemplo:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Exercícios

Escreva um programa que leia uma matriz quadrada $n \times n$ e verifique se ela é uma matriz triangular inferior, triangular superior ou diagonal. Exemplo:

Triangular Inferior

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 9 & 0 & 2 & 3 \end{bmatrix}$$

Triangular Superior

$$\begin{bmatrix} 1 & 0 & 8 & 9 & 8 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Diagonal

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Exercícios

Uma matriz quadrada ($n \times n$) de números inteiros é um *quadrado mágico* se o valor da soma dos elementos de cada linha, de cada coluna e da diagonal principal e da diagonal secundária é o mesmo. Além disso, a matriz deve conter todos os números inteiros do intervalo $[1..n \times n]$.

Exemplo:

$$\begin{bmatrix} 15 & 8 & 1 & 24 & 17 \\ 16 & 14 & 7 & 5 & 23 \\ 22 & 20 & 13 & 6 & 4 \\ 3 & 21 & 19 & 12 & 10 \\ 9 & 2 & 25 & 18 & 11 \end{bmatrix}$$

A matriz acima é um quadrado mágico, cujas somas valem 65.

Escreva um programa que, dada uma matriz quadrada, verifique se ela é um *quadrado mágico*.

Exercícios

Uma matriz de permutações é uma matriz quadrada cujos elementos são zeros ou uns, tal que em cada linha e em cada coluna exista exatamente um elemento igual a 1.

Exemplo:

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

Escreva um programa que, dada uma matriz quadrada, verifique se ela é uma matriz de permutações.

Matriz triangular inferior, triangular superior ou diagonal

```
#include <stdio.h>

#define MAX 10

int main() {
  int matriz[MAX][MAX];
  int i, j, n, inferior = 1, superior = 1;

  scanf("%d", &n);

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      scanf("%d", &matriz[i][j]);

  ...
```

Matriz triangular inferior, triangular superior ou diagonal

```
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if ((matriz[i][j]) && (i != j)) {
        if (i > j)
          superior = 0;
        else
          inferior = 0;
      }

  if (superior && inferior)
    printf("Matriz diagonal\n");
  if (superior)
    printf("Matriz triangular superior\n");
  if (inferior)
    printf("Matriz triangular inferior\n");

  return 0;
}
```

Matriz triangular inferior, triangular superior ou diagonal

```
  for (i = 0; (i < n) && (inferior || superior); i++)
    for (j = 0; (j < n) && (inferior || superior); j++)
      if ((matriz[i][j]) && (i != j)) {
        if (i > j)
          superior = 0;
        else
          inferior = 0;
      }

  if (superior && inferior)
    printf("Matriz diagonal\n");
  if (superior)
    printf("Matriz triangular superior\n");
  if (inferior)
    printf("Matriz triangular inferior\n");

  return 0;
}
```