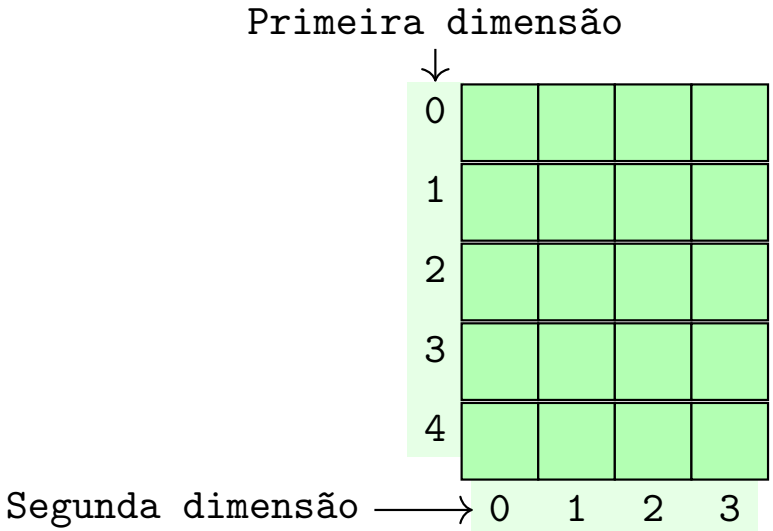
Arranjos bidimensionais e multidimensionais

Alexsandro Santos Soares
prof.asoares@gmail.com

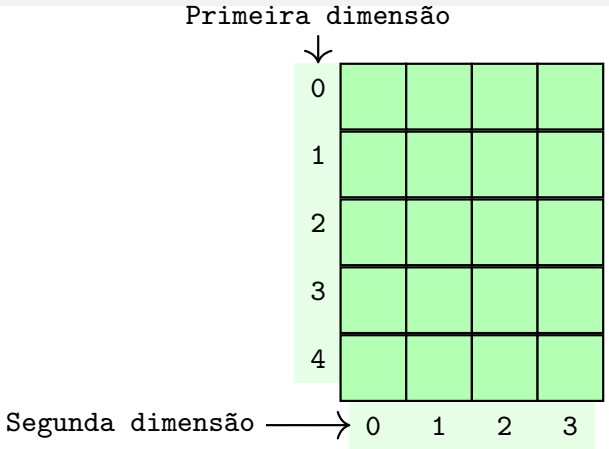
Universidade Federal de Uberlândia
Faculdade de Computação

Introdução

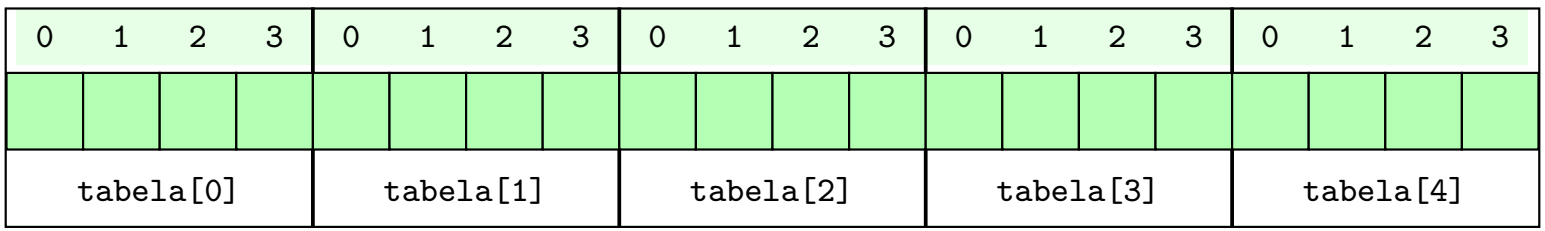
- Os arranjos que discutimos até agora são conhecidos como arranjos unidimensionais pois os dados são organizados linearmente em uma única direção.
- Em muitas aplicações os dados devem ser armazenados em mais que uma dimensão.
- Um exemplo é uma tabela que é um arranjo com linhas e colunas.
- Uma tabela pode ser implementada como um arranjo bidimensional.



Organização de memória em C



- C implementa um arranjo bidimensional como um arranjo de arranjos unidimensionais. Esse conceito é ilustrado abaixo



- Note que a primeira linha da tabela é armazenada como os primeiros quatro endereços do arranjo, a segunda linha nos próximos quatro e assim por diante.
- Ou seja, C armazena a tabela linha por linha, iniciando na primeira linha.
- Como um arranjo bidimensional é comumente usado para a implementação de matrizes, as vezes os chamaremos de matriz.

Declaração

- Arranjos bidimensionais, assim com qualquer outro arranjo, podem ter tamanhos fixos ou variáveis.
- Para declarar um arranjo bidimensional usamos o formato

```
tipo nome[número de linhas][número de colunas];
```

- A tabela vista anteriormente poderia ser declarada da forma abaixo, se ela fosse de tamanho fixo.

```
int tabela[5][4];
```

- Se ela fosse de tamanho variável usaríamos uma variável para cada dimensão:

```
int tabela5x4[nLinhas][nColunas];
```

Lembre-se que, nesse caso, as variáveis `nLinhas` e `nColunas` devem ser definidas antes de serem usadas.

Inicialização

- Como antes, apenas os arranjos de tamanho fixo podem ser inicializados na definição.
- Os valores usados na inicilização de arranjos de tamanho fixo devem ser envoltos por chaves.
- Tome como exemplo o arranjo bidimensional `tabela` declarado no slide anterior. Podemos inicializá-lo de duas formas:
  1. Sabendo que precisamos de 20 valores podemos usar

```
int tabela[5][4] = { 0, 1, 2,  3, 10, 11, 12, 13, 20, 21,
                    22, 23, 30, 31, 32, 33, 40, 41, 42, 43};
```

  2. A segunda forma a seguir é preferida pois deixa mais clara a natureza bidimensional do arranjo:

```
int tabela[5][4] =
 {{0,  1,  2,  3},
  {10, 11, 12, 13},
  {20, 21, 22, 23},
  {30, 31, 32, 33},
  {40, 41, 42, 43}};
```

Inicialização

- Quando estudamos arranjos unidimensionais foi visto que um se arranjo estiver completamente inicializado com os valores fornecidos, não precisamos especificar o tamanho dele.
- Para arranjos bidimensionais apenas a primeira dimensão pode ser omitida. Todas as outras devem ser informadas:

```
int tabela[][4] =
 {{0,  1,  2,  3},
  {10, 11, 12, 13},
  {20, 21, 22, 23},
  {30, 31, 32, 33},
  {40, 41, 42, 43}};
```

- Para inicializar a matriz com zeros, podemos usar

```
int tabela[5][4] = {0};
```

Escrevendo valores

- Podemos preencher uma matriz usando dois laços `for` aninhados.
  - Se a matriz for m por n, o primeiro laço varia de 0 a $m-1$ e o segundo, de 0 a $n-1$.
- O código para preencher o arranjo `tabela` com dados lidos do teclado poderia ser

```
for(linha = 0; linha < 5; linha++)
   for(coluna = 0; coluna < 4; coluna++)
      scanf("%d", &tabela[linha][coluna]);
```

Mostrando valores

- Podemos também imprimir os elementos do arranjo um a um, usando dois laços aninhados.
- O primeiro laço controla a impressão das linhas e o segundo, das colunas.
- O trecho de código a seguir imprime a tabela no formato de uma matriz:

```
for(linha = 0; linha < 5; linha++){
   for(coluna = 0; coluna < 4; coluna++)
      printf("%8d", tabela[linha][coluna]);
   printf("\n");
 } // for linha
```

Acessando valores

- Podemos inicializar qualquer elemento usando o operador de atribuição

```
tabela[2][0] = 23;
tabela[0][1] = tabela[2][0] + 15;
```

- Suponha que se deseje inicializar o arranjo de 5×4 como indicado abaixo

```
00 01 02 03
10 11 12 13
20 21 22 23
30 31 32 33
40 41 42 43
```

- Uma forma de fazer isso é codificando os valores à mão.
- Entretanto, é mais interessante analisar o padrão e depois atribuir os valores algoritmicamente.
- Qual é o padrão?

Acessando valores

- Um dos padrões é que o valor de cada elemente é incrementado em um a partir de seu predecessor na mesma linha.
- Um outro padrão é que o primeiro elemento de cada linha é o índice da linha vezes 10.
- Com isso em mente, podemos escrever uma função para inicilizar o arranjo:

```
#define NUM_COLS 4

void preencheMatriz(int tabela[][NUM_COLS], int numLinhas){
   for(int linha = 0; linha < numLinhas; linha++){
     tabela[linha][0] = linha * 10;
     for(int coluna = 1; coluna < NUM_COLS; coluna++)
        tabela[linha][coluna] = tabela[linha][coluna - 1] + 1;
   } // for linha
   return;
} // preencheMatriz
```

Exemplo 1 (Mapeando um arranjo em outro)

Considere o problema de converter um arranjo bidimensional em um arranjo unidimensional com os mesmos elementos.

O programa a seguir resolve o problema.

```
10 #include <stdio.h>
11
12 #define LINHAS 2
13 #define COLUNAS 5
14
15 int main(void){
16   int matriz[LINHAS][COLUNAS] =
17     {{ 0,  1,  2,  3 , 4},
18      {10, 11, 12, 13, 14}};
19   int vet[LINHAS * COLUNAS];
20
21   for(int linha = 0; linha < LINHAS; linha++)
22     for(int coluna = 0; coluna < COLUNAS; coluna++)
23       vet[linha * COLUNAS + coluna] = matriz[linha][coluna];
24
25   for(int linha = 0; linha < LINHAS * COLUNAS; linha++)
26     printf(" %02d ", vet[linha]);
27   printf("\n");
28   return 0;
29 } // main
```

Mapeando um arranjo em outro

A saída do programa anterior é

```
> ./exemplo1.exe
 00  01  02  03  04  10  11  12  13  14
```

Passando um arranjo bidimensional

- Com arranjos bidimensionais temos três escolhas para a passagem de partes dele para uma função:
  1. Podemos passar elementos individuais, como já visto.
  2. Podemos passar uma linha do arranjo, que é similar ao modo que passamos arranjos unidimensionais.
  3. Podemos passar o arranjo como um todo.
- Vamos discutir os dois últimos casos nos próximos slides.

Passando uma linha

- Podemos passar uma linha inteira de um arranjo indexando-o com o número da linha desejada.
- Tome como exemplo a arranjo a seguir.

tabela

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33
4	40	41	42	43

- Quando passamos uma linha desse arranjo estamos passando um arranjo com quatro valores.
- Logo, a função que recebe essa linha está, na verdade, recebendo um arranjo unidimensional com quatro elementos.

Passando uma linha

- A função `imprime_quadrado`, por exemplo, imprime o quadrado de cada um dos elementos do arranjo unidimensional

```
#define NUM_LINS 5
#define NUM_COLS 4

void imprime_quadrado(int x[]){
  for(int col = 0; col < NUM_COLS; col++)
      printf("%6d", x[col] * x[col]);
  printf("\n");
  return;
} // imprime_quadrado
```

Passando uma linha

- A função `main` chama `imprime_quadrado` cinco vezes tal que o resultado final é uma tabela impressa com os quadrados dos valores presentes no arranjo.

```
int main(void){
   int tabela[NUM_LINS][NUM_COLS] =
     {{0,  1,  2,  3},
      {10, 11, 12, 13},
      {20, 21, 22, 23},
      {30, 31, 32, 33},
      {40, 41, 42, 43}};

   for(int linha = 0; linha < NUM_LINS; linha++)
       imprime_quadrado(tabela[linha]);

   return 0;
} // main
```

Passando uma linha

- Poderíamos ter usado um arranjo com tamanho variável na função `imprime_quadrado`.
- Para isso, teríamos que alterar a definição da função e também alterar a chamada para passar o tamanho do arranjo, como mostrado a seguir

```
// Chamada da função
imprime_quadrado(NUM_COLS, tabela);

// Definição da função
void imprime_quadrado(int tamanho, int x[tamanho]){
  ...
} // imprime_quadrado
```

Passando todo o arranjo

- Para passar um arranjo bidimensional para uma função usamos o nome do arranjo, assim como fizemos com um arranjo unidimensional.
- O cabeçalho da definição da função chamada deve indicar que o arranjo é bidimensional, como no exemplo a seguir

```
double media(int tabela[][NUM_COLS])
```

- Observe novamente que não há necessidade de especificar o número de linhas em um arranjo de tamanho fixo.
- Entretanto, devemos especificar o número de colunas sempre.
- Se o arranjo tiver tamanho variável ambas as dimensões devem ser informadas.

Exemplo 2 (Passando todo o arranjo)

O programa a seguir usa uma função para calcular a média de todos os inteiros presentes em uma matriz.

```
 9 #include <stdio.h>
10
11 #define NUM_LINS 5
12 #define NUM_COLS 4
13
14 /**
15  * @brief Calcula a média dos elementos na matriz dada
16  *
17  * @param m matriz de inteiros
18  */
19 double media(int m[][NUM_COLS]){
20   double soma = 0.0;
21   for(int i = 0; i < NUM_LINS; i++)
22     for(int j = 0; j < NUM_COLS; j++)
23       soma += m[i][j];
24
25   return (soma / (NUM_LINS * NUM_COLS));
26 } // media
```

Continuação do código do exemplo

```
29 int main(void){
30   double m = 0.0; // média
31   int matriz[NUM_LINS][NUM_COLS] =
32     {{ 0,  1,  2,  3},
33      {10, 11, 12, 13},
34      {20, 21, 22, 23},
35      {30, 31, 32, 33},
36      {40, 41, 42, 43}
37     };
38
39   m = media(matriz);
40
41   printf("A média é%lf\n", m);
42   return 0;
43 } // main
```

Exemplo 3 (Preenchendo uma matriz)

Escreva um programa que preencha a diagonal principal de uma matriz quadrada com zeros, a parte triangular inferior com -1 e a parte triangular superior com +1. A saída do programa, supondo que a matriz é de ordem 6 é mostrada a abaixo.

	0	1	2	3	4	5
0	0	1	1	1	1	1
1	-1	0	1	1	1	1
2	-1	-1	0	1	1	1
3	-1	-1	-1	0	1	1
4	-1	-1	-1	-1	0	1
5	-1	-1	-1	-1	-1	0

Solução do exemplo

```
10 #include <stdio.h>
11
12 #define ORDEM 6
13
14 int main(void){
15   int matriz[ORDEM][ORDEM];
16
17   for(int linha = 0; linha < ORDEM; linha++)
18     for(int coluna = 0; coluna < ORDEM; coluna++)
19       if (linha == coluna)
20          matriz[linha][coluna] = 0;
21       else if (linha > coluna)
22          matriz[linha][coluna] = -1;
23       else
24          matriz[linha][coluna] = 1;
25
26   for(int linha = 0; linha < 6; linha++){
27      for(int coluna = 0; coluna < 6; coluna++)
28         printf("%3d", matriz[linha][coluna]);
29      printf("\n");
30   } // for linha
31
32   return 0;
33 } // main
```

Exercício 1

Refaça o exemplo anterior tal que a ordem da matriz seja lida do teclado. Use um arranjo com tamanho variável.

Arranjos multidimensionais

- Arranjos multidimensionais podem ter três, quatro ou mais dimensões.
- A primeira dimensão de um arranjo tridimensional é chamada de plano e contém linhas e colunas.
- Em C, um arranjo tridimensional é um arranjo de arranjos bidimensionais.
  - Ele armazena este arranjo um plano por vez, iniciando com o primeiro.
  - Como cada plano é na verdade um arranjo bidimensional, ele é armazenado linha por linha, como antes.
- Essa ideia também vale para arranjos com mais que três dimensões.

Analogias

- Para fixar a ideia de arranjos multidimensionais, imagine que estamos lidando com textos com regras bem rígidas.
- Cada linha de texto pode ser vista como um arranjo unidimensional, digamos com 80 caracteres:

```
char texto[80];
```

- Uma página é um arranjo bidimensional com 50 linhas, cada uma com 80 caracteres:

```
char pagina[50][80];
```

- Um livro é composto por 300 páginas:

```
char livro[300][50][80];
```

Assim, se desejarmos imprimir o conteúdo da pág. 37 basta digitar

```
for(int linha = 0; linha < 50; linha++){
   for(int coluna = 0; coluna < 80; coluna++)
      printf("%c", livro[37][linha][coluna]);
   printf("\n");
} // for linha
```

Analogias

- Continuando a analogia, podemos pensar em uma prateleira que cabe 60 livros:

```
char prateleira[60][300][50][80];
```

- Um estante contém 6 prateleiras

```
char estante[6][60][300][50][80];
```

- Poderiámos continuar com analogia para um andar, que pode conter várias estantes; uma biblioteca, que pode conter vários andares, etc. Mas, ficaremos por aqui.

Inicialização

- Para inicializar um arranjo multidimensional de tamanho fixo podemos extender o que dissemos antes para arranjos uni e bidimensionais.
- Por exemplo, poderíamos declarar e inicializar um arranjo tridimensional de tamanho fixo usando o código abaixo

```
int tabela[3][5][4]=
{
  { // Plano 0
    { 0,  1,  2,  3},   // Linha 0
    {10, 11, 12, 13}, // Linha 1
    {20, 21, 22, 23}, // Linha 2
    {30, 31, 32, 33}, // Linha 3
    {40, 41, 42, 43}, // Linha 4
  },
  { // Plano 1
    {100, 101, 102, 103}, // Linha 0
    {110, 111, 112, 113}, // Linha 1
    {120, 121, 122, 123}, // Linha 2
    {130, 131, 132, 133}, // Linha 3
    {140, 141, 142, 143}, // Linha 4
  },
```

Inicialização

- Continuação da declaração e inicialização

```
    { // Plano 2
      {200, 201, 202, 203}, // Linha 0
      {210, 211, 212, 213}, // Linha 1
      {220, 221, 222, 223}, // Linha 2
      {230, 231, 232, 233}, // Linha 3
      {240, 241, 242, 243}, // Linha 4
    }
  } // tabela;
```

- Se desejássemos inicilizar todos os elementos com zero:

```
int tabela[3][5][4] = {0};
```

Exemplos

- Nos próximos slides veremos exemplos do uso de arranjos multidimensionais.
- O primeiro exemplo lida com um arranjo tridimensional para a entrada de texto.
- O segundo exemplo implementa algumas operações matemáticas básicas do cálculo matricial.

Exemplo 4 (Lendo e imprimindo arranjos tridimensionais)

No exemplo abaixo implementamos um livro como um arranjo tridimensional. Na sequência pedimos ao usuário que digite a página que ele deseja escrever e depois colocamos todos os caracteres digitados na página escolhida. Ao final imprimimos a página para conferência.

```c
#define PAGINAS 300
#define LINHAS 5
#define COLUNAS 64

int main(void){
  char livro[PAGINAS][LINHAS][COLUNAS];
  int pagina;

  // Leitura do texto digitado pelo usuário
  do {
     printf("Digite o número da página: ");
     scanf("%d", &pagina);
  } while (pagina < 0 || pagina >= PAGINAS);

  printf("Digite o texto para a página %d\n\n", pagina);
  for(int linha = 0; linha < LINHAS; linha++)
     for(int coluna = 0; coluna < COLUNAS; coluna++)
       scanf("%c", &livro[pagina][linha][coluna]);

  printf("\nAqui está o texto digitado:\n\n");
  for(int linha = 0; linha < LINHAS; linha++){
     for(int coluna = 0; coluna < COLUNAS; coluna++)
        printf("%c", livro[pagina][linha][coluna]);
  } // for linha

  return 0;
} // main
```

Exemplo de uso

```
> ./exemplo4.exe
Digite o número da página: 37
Digite o texto para a página 37

Sentia um acréscimo de estima por si mesma, e     parecia-lhe
que entrava enfim numa existência superiormente interessante,
onde cada hora tinha  o  seu  encanto diferente,  cada  passo
conduzia a um êxtase, e  a  alma se cobria de um luxo radioso
de sensações!            -- O Primo Basílio de Eça de Queiroz


Aqui está o texto digitado:


Sentia um acréscimo de estima por si mesma, e     parecia-lhe
que entrava enfim numa existência superiormente interessante,
onde cada hora tinha  o  seu  encanto diferente,  cada  passo
conduzia a um êxtase, e  a  alma se cobria de um luxo radioso
de sensações!            -- O Primo Basílio de Eça de Queiroz
```

Exemplo 5

Cálculo matricial Existem várias operações matemáticas no cálculo matricial. Neste exemplo construíremos funções para as seguintes operações:

1. Ler uma matriz de número reais a partir do teclado.
2. Imprimir o conteúdo de uma matriz.
3. Copiar o conteúdo de uma matriz para outra.
4. Somar um número a uma linha da matriz.
5. Criar uma matriz transposta.
6. Somar duas matrizes.
7. Multiplicar duas matrizes

A função leiaMatriz

```
18 #include <stdio.h>
19
20 /**
21  * @brief Lê uma matriz a partir da entrada padrão
22  *
23  * @param[in] nLin número de linhas da matriz
24  * @param[in] nCol número de colunas da matriz
25  * @param[out] mat a matriz
26  *
27  * @post mat será preenchida com os valores lidos
28  */
29 void leiaMatriz(int nLin, int nCol, double mat[nLin][nCol]){
30   // Preenche a matriz linha por linha, a partir da primeira
31   for(int i = 0; i < nLin; i++)
32     for(int j = 0; j < nCol; j++)
33       scanf("%lf", &mat[i][j]);
34
35   return;
36 } // leiaMatriz
```

A função imprimaMatriz

```
39 /**
40  * @brief Imprime uma matriz para a saída padrão
41  *
42  * @param[in] nLin número de linhas da matriz
43  * @param[in] nCol número de colunas da matriz
44  * @param[in] mat a matriz
45  *
46  */
47 void imprimaMatriz(int nLin, int nCol, const double mat[nLin][nCol]){
48   // Imprime a matriz linha por linha, a partir da primeira
49   for(int i = 0; i < nLin; i++){
50     for(int j = 0; j < nCol; j++)
51       printf("%6.2lf", mat[i][j]);
52     printf("\n");
53   } // for i
54
55   return;
56 } // imprimaMatriz
```

A função copieMatriz

```
59 /**
60  * @brief Copia o conteúdo da primeira matriz na segunda
61  *
62  * @param[in] nLin número de linhas da matriz
63  * @param[in] nCol número de colunas da matriz
64  * @param[in] mat  a matriz original
65  * @param[out] copia cópia da matriz mat
66  *
67  * @post copia será preenchida com os valores de mat
68  */
69 void copieMatriz(int nLin, int nCol,
70                  const double mat[nLin][nCol],
71                  double copia[nLin][nCol]){
72   for(int i = 0; i < nLin; i++)
73     for(int j = 0; j < nCol; j++)
74         copia[i][j] = mat[i][j];
75
76   return;
77 } // copieMatriz
```

A função somaNumero

```
80 /**
81  * @brief Soma um númera aos elmentos da matriz
82  *
83  * @param[in] nLin número de linhas da matriz
84  * @param[in] nCol número de colunas da matriz
85  * @param[in] num número a ser adicionado aos elementos
86  * @param[in/out] mat a matriz
87  *
88  * @post mat será alterada com a soma de num a seus elementos
89  */
90 void somaNumero(int nLin, int nCol, double num,
91                 double mat[nLin][nCol]){
92
93   for(int i = 0; i < nLin; i++)
94     for(int j = 0; j < nCol; j++)
95       mat[i][j] += num;
96
97   return;
98 } // somaNumero
```

Matriz transposta

- Em matemática, matriz transposta é a matriz que se obtém da troca de linhas por colunas de uma dada matriz.
- A transposta da matriz $A = [a_{i,j}]_{i,j=0}^{m,n}$ é a matriz $A^{\mathrm{T}} = [a_{i,j}]_{j,i=0}^{n,m}$, ou seja:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,0} & a_{m,1} & \dots & a_{m,n} \end{bmatrix} \Leftrightarrow A^{\mathrm{T}} = \begin{bmatrix} a_{0,0} & a_{1,0} & \dots & a_{m,0} \\ a_{0,1} & a_{1,1} & \dots & a_{m,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0,n} & a_{1,n} & \dots & a_{m,n} \end{bmatrix}$$

- Veja alguns exemplos:

$$\begin{bmatrix} 1 & 2 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}.$$

A função transposta

```
101 /**
102  * @brief Encontra a matriz transposta
103  *
104  * @param[in] nLin  número de linhas da matriz
105  * @param[in] nCol  número de colunas da matriz
106  * @param[in] mat   a matriz
107  * @param[out] transp a matriz transposta de mat
108  *
109  * @post transp será preenchida com a transposta de mat
110  */
111 void transposta(int nLin, int nCol,
112                 const double mat[nLin][nCol],
113                 double transp[nCol][nLin]){
114
115   for(int i = 0; i < nLin; i++)
116       for(int j = 0; j < nCol; j++)
117           transp[j][i] = mat[i][j];
118
119   return;
120 } // transposta
```

A função somaMatriz

```
123 /**
124  * @brief Soma duas matrizes
125  *
126  * @param[in] nLin número de linhas da matriz
127  * @param[in] nCol número de colunas da matriz
128  * @param[in] mat1 a primeira matriz
129  * @param[in] mat2 a segunda matriz
130  * @param[out] soma a matriz resultante da soma
131  *
132  * @pre mat1 e mat2 devem ter as mesmas dimensões
133  * @post soma será preenchida com mat1 + mat2
134  */
135 void somaMatriz(int nLin, int nCol,
136                 const double mat1[nLin][nCol],
137                 const double mat2[nLin][nCol],
138                 double soma[nLin][nCol]){
139   for(int i = 0; i < nLin; i++)
140     for(int j = 0; j < nCol; j++)
141       soma[i][j] = mat1[i][j] + mat2[i][j];
142
143   return;
144 } // somaMatriz
```

Multiplicação de matrizes

- Em matemática, o produto de duas matrizes é definido somente quando o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz. Se A é uma matriz m-por-n e B é uma matriz n-por-p, então seu produto é uma matriz m-por-p definida como AB (ou por $A \cdot B$).
- O produto é dado por

$$(AB)_{ij} = a_{i0}b_{0j} + \cdots + a_{i(m-1)}b_{(m-1)j} = \sum_{k=0}^{m-1} a_{ik}b_{kj}$$

  para $i = 0, \ldots, n-1$ e $j = 0, \ldots, p-1$.
- Ou seja, a entrada $(AB)_{ij}$ do produto é obtido multiplicando-se termo a termo as entradas da i-ésima linha de A pela j-ésima coluna de B, somando estes m produtos.

Exemplo de multiplicação de matrizes

- Por exemplo, se

$$A = \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ -1 & 4 \end{bmatrix} \text{ e } B = \begin{bmatrix} 1 & 2 & 3 \\ -2 & 0 & 4 \end{bmatrix}$$

então

$$A \cdot B = \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ -2 & 0 & 4 \end{bmatrix} =$$

$$\begin{bmatrix} 2 \cdot 1 + 3(-2) & 2 \cdot 2 + 3(0) & 2 \cdot 3 + 3 \cdot 4 \\ 0 \cdot 1 + 1(-2) & 0 \cdot 2 + 1(0) & 0 \cdot 3 + 1 \cdot 4 \\ -1 \cdot 1 + 4(-2) & -1 \cdot 2 + 4(0) & -1 \cdot 3 + 4 \cdot 4 \end{bmatrix} = \begin{bmatrix} -4 & 4 & 18 \\ -2 & 0 & 4 \\ -9 & -2 & 13 \end{bmatrix}$$

```
147 /**
148  * @brief Multiplica duas matrizes
149  *
150  * @param[in] nLin número de linhas de mat1
151  * @param[in] nCol número de colunas de mat1
152  * @param[in] mat1 a primeira matriz
153  * @param[in] nCol2 número de colunas de mat2
154  * @param[in] mat2 a segunda matriz
155  * @param[out] produto a matriz resultante do produto
156  *
157  * @pre mat1, mat2 e produto devem ter dimensões compatíveis
158  * @post produto será preenchida com mat1 * mat2
159  */
160 void multiplicaMatriz(int nLin, int nCol,
161                       const double mat1[nLin][nCol],
162                       int nCol2,
163                       const double mat2[nCol][nCol2],
164                       double produto[nLin][nCol2]){
165   for(int i = 0; i < nLin; i++)
166      for(int j = 0; j < nCol2; j++){
167         produto[i][j] = 0; // zera para acumular o somatório
168         for(int k = 0; k < nCol; k++)
169            produto[i][j] += mat1[i][k] * mat2[k][j];
170      } // for j
171
172   return;
173 } // multiplicaMatriz
```

Testes das operações matriciais

- Vamos agora escrever algumas funções para testar:
  - a soma de um número a uma matriz.
  - a transposta da matriz.
  - a soma de uma matriz com outra.
  - o produto de duas matrizes.

Teste da soma de um número em uma matriz

```
185 void testeSomaNumero(int nLin, int nCol,
186                     const double mat[nLin][nCol]){
187     double num = 0.0;       // número a ser somado àmatriz
188     double copia[nLin][nCol]; // guarda a cópia da matriz mat
189 
190     // Faça a cópia da matriz mat
191     copieMatriz(nLin, nCol, mat, copia);
192     printf("\nDigite o número a ser somado àmatriz: ");
193     scanf("%lf", &num);
194 
195     somaNumero(nLin, nCol, num, copia);
196 
197     printf("\nResultado de somar %.2lf àmatriz:\n", num);
198     imprimaMatriz(nLin, nCol, copia);
199 
200     return;
201 } // testeSomaNumero
```

Teste da matriz transposta

```
211 void testeTransposta(int nLin, int nCol,
212                         const double mat[nLin][nCol]){
213     // A matriz transposta tem as dimensões permutadas em relação
214     // à matriz original.
215     double trans[nCol][nLin];
216 
217     transposta(nLin, nCol, mat, trans);
218 
219     printf("\nMatriz original:\n");
220     imprimaMatriz(nLin, nCol, mat);
221 
222     printf("\nA transposta da matriz é:\n");
223     imprimaMatriz(nCol, nLin, trans);
224 
225     return;
226 } // testeTransposta
```

Teste da soma de duas matrizes

```
238 void testeSomaMatriz(int nLin, int nCol,
239                        const double mat[nLin][nCol]){
240     // a segunda matriz deve ter as mesmas dimensões que a primeira
241     double mat2[nLin][nCol];
242     double soma[nLin][nCol]; // matriz soma de mat com mat2
243 
244     printf("\nDigite a segunda matriz (%d x %d)\n\n", nLin, nCol);
245     leiaMatriz(nLin, nCol, mat2);
246 
247     somaMatriz(nLin, nCol, mat, mat2, soma);
248 
249     printf("\nA soma da matriz:\n");
250     imprimaMatriz(nLin, nCol, mat);
251     printf("com\n");
252     imprimaMatriz(nLin, nCol, mat2);
253     printf("é:\n");
254     imprimaMatriz(nLin, nCol, soma);
255 
256     return;
257 } // testeSomaMatriz
```

```
269 void testeMultiplicaMatriz(int nLin, int nCol,
270                           const double mat[nLin][nCol]){
271     int nCol2 = 0.0; // número de colunas da segunda matriz
272     printf("\nDigite o número de colunas da segunda matriz: ");
273     scanf("%d", &nCol2);
274     {
275         // O número de linhas da segunda matriz deve ser igual
276         // ao número de colunas da primeira
277         double mat2[nCol][nCol2]; // matriz de tamanho variável
278 
279         // As dimensões da matriz produto entre uma matriz A (l x m) e
280         // uma matriz B (m x n) é(l x n)
281         double produto[nLin][nCol2]; // matriz produto de mat por mat2
282 
283         printf("Digite a segunda matriz (%d x %d)\n\n", nCol, nCol2);
284         leiaMatriz(nLin, nCol, mat2);
285 
286         multiplicaMatriz(nLin, nCol, mat, nCol2, mat2, produto);
287 
288         printf("\nO produto da matriz:\n");
289         imprimaMatriz(nLin, nCol, mat);
290         printf("com\n");
291         imprimaMatriz(nCol, nCol2, mat2);
292         printf("é:\n");
293         imprimaMatriz(nLin, nCol2, produto);
294     }
295     return;
296 } // testeMultiplicaMatriz
```

```
299 int main(void){
300   int nLin = 0;    // número de linhas
301   int nCol = 0;    // número de colunas
302
303   printf("Digite o número de linhas da primeira matriz: ");
304   scanf("%d", &nLin);
305   printf("Digite o número de colunas da primeira matriz: ");
306   scanf("%d", &nCol);
307   {
308     double mat[nLin][nCol]; // matriz de tamanho variável
309
310     printf("Digite os elementos da primeira matriz\n\n");
311     leiaMatriz(nLin, nCol, mat);
312
313     printf("\nPrimeira matriz:\n");
314     imprimaMatriz(nLin, nCol, mat);
315
316     testeSomaNumero(nLin, nCol, mat);
317     testeTransposta(nLin, nCol, mat);
318     testeSomaMatriz(nLin, nCol, mat);
319     testeMultiplicaMatriz(nLin, nCol, mat);
320   }
321
322   printf("\n");
323   return 0;
324 } // main
```

Uso do programa

- Um exemplo de saída do programa é mostrado neste e nos próximos slides.

```
> ./exemplo5.exe
Digite o número de linhas da primeira matriz: 3
Digite o número de colunas da primeira matriz: 2
Digite os elementos da primeira matriz

1 2 3 4 5 6
```

Uso do programa

```
Primeira matriz:
  1.00  2.00
  3.00  4.00
  5.00  6.00

Digite o número a ser somado à matriz: 1

Resultado de somar 1.00 à matriz:
  2.00  3.00
  4.00  5.00
  6.00  7.00
```

Uso do programa

```
Matriz original:
  1.00  2.00
  3.00  4.00
  5.00  6.00

A transposta da matriz é:
  1.00  3.00  5.00
  2.00  4.00  6.00
```

Uso do programa

```
Digite a segunda matriz (3 x 2)

1 2 3 4 5 6

A soma da matriz:
  1.00  2.00
  3.00  4.00
  5.00  6.00
com
  1.00  2.00
  3.00  4.00
  5.00  6.00
é:
  2.00  4.00
  6.00  8.00
 10.00 12.00
```

Uso do programa

```
Digite o número de colunas da segunda matriz: 3
Digite a segunda matriz (2 x 3)

1 2 3 4 5 6

O produto da matriz:
  1.00  2.00
  3.00  4.00
  5.00  6.00
com
  1.00  2.00  3.00
  4.00  5.00  6.00
é:
  9.00 12.00 15.00
 19.00 26.00 33.00
 29.00 40.00 51.00
```

Para saber mais

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.

Fontes

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.
- Matriz transposta. In: Wikipédia, a enciclopédia livre. Flórida: Wikimedia Foundation, 2016, rev. 18 Outubro 2016. Disponível em: `https://pt.wikipedia.org/w/index.php?title=Matriz_transposta&oldid=46989672`.