

Fundamentos de Programação

Matrizes

Dainf - UTFPR

Profa. Leyza B. Dorini
Prof. Bogdan T. Nassu

Vetores

Anteriormente, criamos um programa que lia as notas de uma prova para um conjunto de N alunos e então calculava a média da turma. Para tal, utilizamos vetores de N posições (ao invés de criar uma variável para representar cada nota).

Para calcular a média da turma para cada prova, seria necessário declarar P vetores, um para cada prova.

Prova 1	7.5	8.5	6.5	...	8.0
---------	-----	-----	-----	-----	-----

Prova 2	1.5	7.0	3.5	...	6.5
---------	-----	-----	-----	-----	-----

Prova 3	1.5	7.0	9.0	...	6.5
---------	-----	-----	-----	-----	-----

Isso traz limitações similares às que vimos para a declaração de várias variáveis.

Matrizes

Para lidar com problemas desse tipo de forma mais efetiva, vamos aprender a utilizar matrizes (vetores/*arrays* multidimensionais).

O que podemos representar usando matrizes?

- ① Sistemas de equações.
- ② Transformações geométricas (computação gráfica).
- ③ Imagens.
- ④ Listas de strings.
- ⑤ Tabuleiros de jogos como damas ou xadrez.
- ⑥ etc.

Declarando uma matriz (bidimensional)

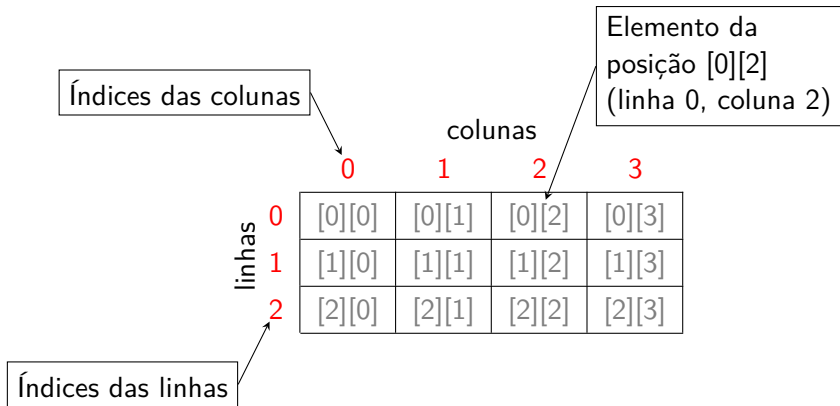
Ao declarar um matriz devemos informar, além do tipo dos elementos que serão armazenados, a quantidade de linhas e a quantidade de colunas.

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

Assim como acontecia com vetores, em C ANSI o número de linhas e colunas deve ser uma **constante** numérica.

Exemplo de declaração de matriz

Suponha a declaração de uma matriz de inteiros de 3 linhas por 4 colunas: `int matriz [3][4];`



Índices de linhas e colunas iniciam em zero!

Assim como para vetores unidimensionais, os índices de linhas e colunas iniciam em zero!



Acessando os elementos

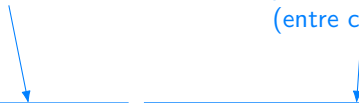
Acessando os elementos uma matriz

A sintaxe para acessar um determinado elemento de uma matriz é:

nome/identificador
da matriz

índices da linha
e da coluna
que deseja acessar
(entre colchetes)

```
1 nomeDoMatriz [índiceLinha] [índiceColuna];
```



matriz[LINHA] [COLUNA]

Por convenção, a ordem padrão dos índices é primeiro linha e depois coluna.



Armazenando valores em uma matriz

Armazenando valores em uma matriz

Por exemplo, suponha uma matriz chamada `preco`, do tipo `float` e de dimensão 2×5 . Como armazenar o valor 10.5 no índice `[0][3]`?

	0	1	2	3	4
0				10.5	
1					

```
1  int main(){
2
3      float preco[2][5];
4
5      preco[0][3] = 10.5;
6
7      return 0;
8  }
```

Armazenando valores em uma matriz

Por exemplo, suponha uma matriz chamada `preco`, do tipo `float` e de dimensão 2×5 . Como armazenar o valor 10.5 no índice `[0][3]`?

	0	1	2	3	4
0				10.5	
1					

```
1 int main(){
```

```
2
```

```
3     float preco[2][5];
```

```
4
```

```
5     preco[0][3] = 10.5;
```

```
6
```

```
7     return 0;
```

```
8 }
```

nome da matriz

índice do elemento
(linha, coluna)
que deseja acessar
(cada um entre colchetes)

Armazenando valores em uma matriz

E se fosse necessário armazenar um valor lido do teclado na primeira posição?

	0	1	2	3	4
0	AQUI				
1					

```
1 int main(){
2
3     float preco[2][5];
4
5     scanf(" %f ", &preco[0][0]);
6
7     return 0;
8 }
```

Armazenando valores em uma matriz

E se fosse necessário armazenar um valor lido do teclado na primeira posição?

	0	1	2	3	4
0	AQUI				
1					

```
1 int main(){  
2  
3     float preco[2][5];  
4  
5     scanf(" %f ", &preco[0][0]);  
6  
7     return 0;  
8 }
```

especificador de
formato compatível com
o tipo do vetor

endereço da posição do
vetor que deseja acessar

Armazenando valores em uma matriz

E para ler do teclado valores em todas as posições?

```
1  int main(){
2      float preco[2][5];
3
4      scanf("%f", &preco[0][0]);
5      scanf("%f", &preco[0][1]);
6      scanf("%f", &preco[0][2]);
7      scanf("%f", &preco[0][3]);
8      scanf("%f", &preco[0][4]);
9      scanf("%f", &preco[1][0]);
10     scanf("%f", &preco[1][1]);
11     scanf("%f", &preco[1][2]);
12     scanf("%f", &preco[1][3]);
13     scanf("%f", &preco[1][4]);
14
15     return 0;
16 }
```


Armazenando valores em uma matriz

E para ler do teclado valores em todas as posições?

```
1 int main(){
2     float preco[2][5];
3
4     scanf("%f", &preco[0][0]);
5     scanf("%f", &preco[0][1]);
6     scanf("%f", &preco[0][2]);
7     scanf("%f", &preco[0][3]);
8     scanf("%f", &preco[0][4]);
9     scanf("%f", &preco[1][0]);
10    scanf("%f", &preco[1][1]);
11    scanf("%f", &preco[1][2]);
12    scanf("%f", &preco[1][3]);
13    scanf("%f", &preco[1][4]);
14
15    return 0;
16 }
```

Observe que a única alteração são os índices acessados. Portanto, podemos explorar o uso de estruturas de repetição. Como?

Para cada linha, são percorridos todos os índices das colunas (ou seja, para a linha 0, percorre os índices de coluna de 0 até 4. Para a linha 1, a mesma coisa).

Vamos explorar estruturas de repetição aninhadas/encaixadas!.

Armazenando valores em uma matriz

Com uma estrutura de repetição, o programa fica mais robusto.

```
1  int main(){
2      float preco[2][5];
3      int i, j;
4
5      for(i=0; i<2; i++){
6          for(j=0; j<5; j++){
7              scanf("%f", &preco[i][j]);
8          }
9
10     return 0;
11 }
```

O contador *i* vai
controlar o valor dos
índices das linhas...

... e o contador *j*
das colunas.

Acessa o índice *[i][j]*
(como é uma repetição encaixada,
lembre-se que, para cada valor
de *i*, são percorridos todos
os valores de *j*).

Atribuindo valores aleatórios

Para matrizes muito grandes, pode ser inviável ler valores do teclado. Portanto, uma possibilidade é utilizar a função `rand()`.
Exemplo:

```
1  int main(){
2      float preco[2][5];
3      int i, j;
4
5      for(i=0; i<2; i++){
6          for(j=0; j<5; j++){
7              preco[i][j] = rand()%10;
8          }
9
10     return 0;
11 }
```

O contador `i` vai controlar o valor dos índices das linhas...

... e o contador `j` das colunas.

Acessa o índice `[i][j]` (como é uma repetição encaixada, lembre-se que, para cada valor de `i`, são percorridos todos os valores de `j`).

Armazenando valores em uma matriz

Por fim, caso seja necessário atribuir valores já na declaração, a sintaxe é (tudo entre chaves e, além disso, cada linha entre chaves):

```
1 float m[3][2] = {{2, 5}, {9, 8}, {3, 4}};
```

O resultado de tal atribuição é:

	0	1
0	2	5
1	9	8
2	3	4

Boa prática: uso de constantes para definir o tamanho da matriz

Assim como fizemos para vetores, uma boa prática consiste em definir (com a diretiva `#define`) uma contante para definir sua dimensão.

Usando constantes para definir o tamanho de vetores

```
1  int main(){
2      float preco[5][2];
3      int i, j;
4
5      for(i=0; i<5; i++){
6          for(j=0; j<2; j++){
7              scanf("%f", &preco[i][j]);
8          }
9
10     return 0;
11 }
```

Observe que a dimensão da matriz também determina a variação dos laços de repetição.

Usando constantes para definir o tamanho de vetores

Ao usar uma constante, evitamos inconsistências como a ilustrada abaixo:

```
1  int main(){
2      float preco[3][2];
3      int i, j;
4
5      for(i=0; i<5; i++){
6          for(j=0; j<2; j++)
7              scanf("%f", &preco[i][j]);
8      }
9
10     return 0;
11 }
```

Note que a quantidade de linhas da matriz foi alterada, mas o limite da variação do laço de repetição não.

Desta forma, posições indevidas da memória serão acessadas.

Usando constantes para definir o tamanho de vetores

```
1  #define NL 3
2  #define NC 2
3
4  int main(){
5      float preco[NL][NC];
6      int i;
7
8      for(i=0; i<NL; i++){
9          for(j=0; j<NC; j++)
10             scanf("%f", &preco[i][j]);
11     }
12
13     for(i=0; i<NL; i++){
14         for(j=0; j<NC; j++)
15             printf("%f", preco[i][j]);
16     }
17
18     return 0;
19 }
```

Ao alterar o valor da constante...

... automaticamente o restante do programa fica com valores consistentes.

Exemplos

Determinando maior, menor e soma

```
1 soma = 0;
2 max = min = m[0][0];
3
4 for (i = 0; i < NL; i++){
5     for (j = 0; j < NC; j++){
6         if (m[i][j] > max)
7             max = m[i][j];
8         if (m[i][j] < min)
9             min = m[i][j];
10        soma = soma + m[i][j];
11    }
12 }
```

Como serão utilizadas na comparação com o conteúdo de cada elemento da matriz, é preciso inicializar os valores das variáveis max e min (note que estamos atribuindo ao mesmo tempo).

Se o conteúdo do índice [i][j] for maior que o valor máximo até o momento, atualiza esse valor.

Determinando o maior elemento de cada linha

```
1  #define TAML 5
2  #define TAML 3
3  int main(){
4      int m[TAML][TAMC], max, i, j;
5
6      for (i = 0; i < TAML; i++)
7          for (j = 0; j < TAMC; j++)
8              m[i][j] = i+j;
9
10     for (i = 0; i < TAML; i++){
11         max = m[i][0];
12         for (j = 0; j < TAMC; j++){
13             if (m[i][j] > max)
14                 max = m[i][j];
15         }
16         printf("Linha %d tem maior elemento %d\n", i, max);
17     }
18     return 0;
19 }
```

Procure usar constantes para definir a dimensão da matriz.

Note que é preciso reinicializar max a cada nova linha.

Impressão da diagonal principal

Uma questão muito importante ao trabalhar com matrizes é observar padrões nos índices. Por exemplo, a diagonal principal tem o mesmo índice de linha e coluna. Uma possibilidade é imprimir da seguinte forma:

```
1 for (i = 0; i < TAM; i++)  
2     for (j = 0; j < TAM; j++)  
3         if(i==j) //índice de linha igual ao de coluna  
4             printf("%d ", m[i][j]);  
5
```

Impressão da diagonal principal

Uma questão muito importante ao trabalhar com matrizes é observar padrões nos índices. Por exemplo, a diagonal principal tem o mesmo índice de linha e coluna. Uma possibilidade é imprimir da seguinte forma:

```
1 for (i = 0; i < TAM; i++)
2     for (j = 0; j < TAM; j++)
3         if(i==j) //índice de linha igual ao de coluna
4             printf("%d ", m[i][j]);
5
```

São realizadas $TAM \times TAM$ comparações nos laços mais $TAM \times TAM$ comparações no if. Entretanto, é possível obter uma solução bem mais eficiente (que realiza apenas TAM comparações):

```
1 for (i = 0; i < TAM; i++)
2     printf("%d ", m[i][i]);
3
```

Impressão da diagonal secundária

Para impressão da diagonal secundária...

```
1 for (i = 0; i < TAM; i++)
2     for (j = 0; j < TAM; j++)
3         m[i][j] = i+j;
4
5 for (i = 0; i < TAM; i++){
6     for (j = 0; j < TAM; j++)
7         printf("%2d ", m[i][j]);
8     printf("\n");
9 }
10
```

... explorar o padrão dos índices também torna o programa mais eficiente.

```
1 // faça no papel a simulação dos valores dos índices
2 for (i = 0; i < TAM; i++)
3     printf("m[%d] [%d] = %2d ", i, TAM-1-i, m[i][TAM-1-i]);
4
```

Programa que verifica se matriz é triangular superior

Uma matriz triangular superior é aquela em que os elementos abaixo da diagonal principal são nulos.

	0	1	2	3
0	[0][0]	[0][1]	[0][2]	[0][3]
1	[1][0]	[1][1]	[1][2]	[1][3]
2	[2][0]	[2][1]	[2][2]	[2][3]
3	[3][0]	[3][1]	[3][2]	[3][3]

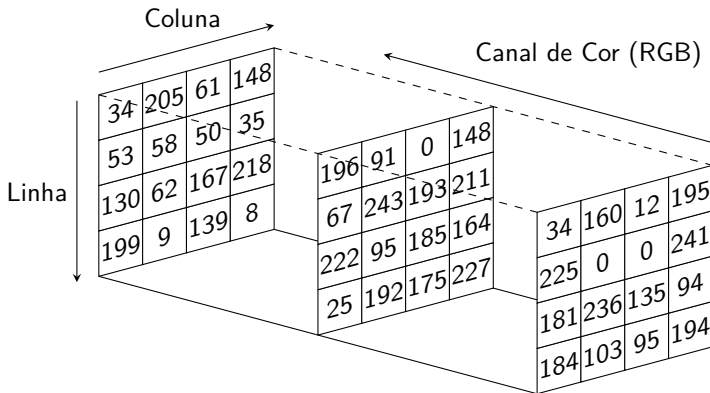
```
1 for (i = 0; i < TAML; i++){
2     for (j = 0; j < TAMC; j++)
3         if(i>j)
4             soma+=m[i][j];
5 }
6 if(soma==0)
7     printf("A matriz é triangular superior");
8
```

Pergunta: essa é a solução mais eficiente? Dica: proponha uma solução baseada em variáveis indicadoras e usando o break.

Criando arrays multidimensionais

Declarando uma matriz de múltiplas dimensões

É possível declarar *arrays* com mais dimensões. Por exemplo, uma imagem colorida pode ser representada por um *array* de três dimensões!



Para declarar, basta inserir a nova dimensão entre colchetes

Exemplo: `int imagem[1280][720][3]`

Exemplos de inicialização

Para *arrays* de uma, duas e três dimensões.

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

Exemplos de inicialização

Para *arrays* de duas e três dimensões.

```
1     printf("Matriz bidimensional v2:\n");
2     for (i = 0; i < 2; i++){
3         for (j = 0; j < 3; j++)
4             scanf("%d", &v2[i][j]);
5     }
6
7     printf("Matriz tridimensional v3:\n");
8     for (i = 0; i < 2; i++) {
9         for (j = 0; j < 3; j++) {
10            for (k = 0; k < 4; k++)
11                scanf("%d", &v3[i][j][k]);
12        }
13    }
```

Observe que precisamos de mais um nível de encadeamento dos laços de repetição!

Exemplos de impressão

Para *arrays* de duas e três dimensões.

```
1      printf("Matriz bidimensional v2:\n");
2      for (i = 0; i < 2; i++){
3          for (j = 0; j < 3; j++)
4              printf("%d ", v2[i][j]);
5          printf("\n");
6      }
7      printf("Matriz tridimensional v3:\n");
8      for (i = 0; i < 2; i++) {
9          for (j = 0; j < 3; j++) {
10             for (k = 0; k < 4; k++)
11                 printf("%d ", v3[i][j][k]);
12             printf("\n");
13         }
14         printf("\n");
15     }
16
```

Agora é contigo!

Faça os exercícios recomendados! Não custa lembrar: comente adequadamente seu código!

