

# Aula 14: Variáveis compostas - matrizes

---

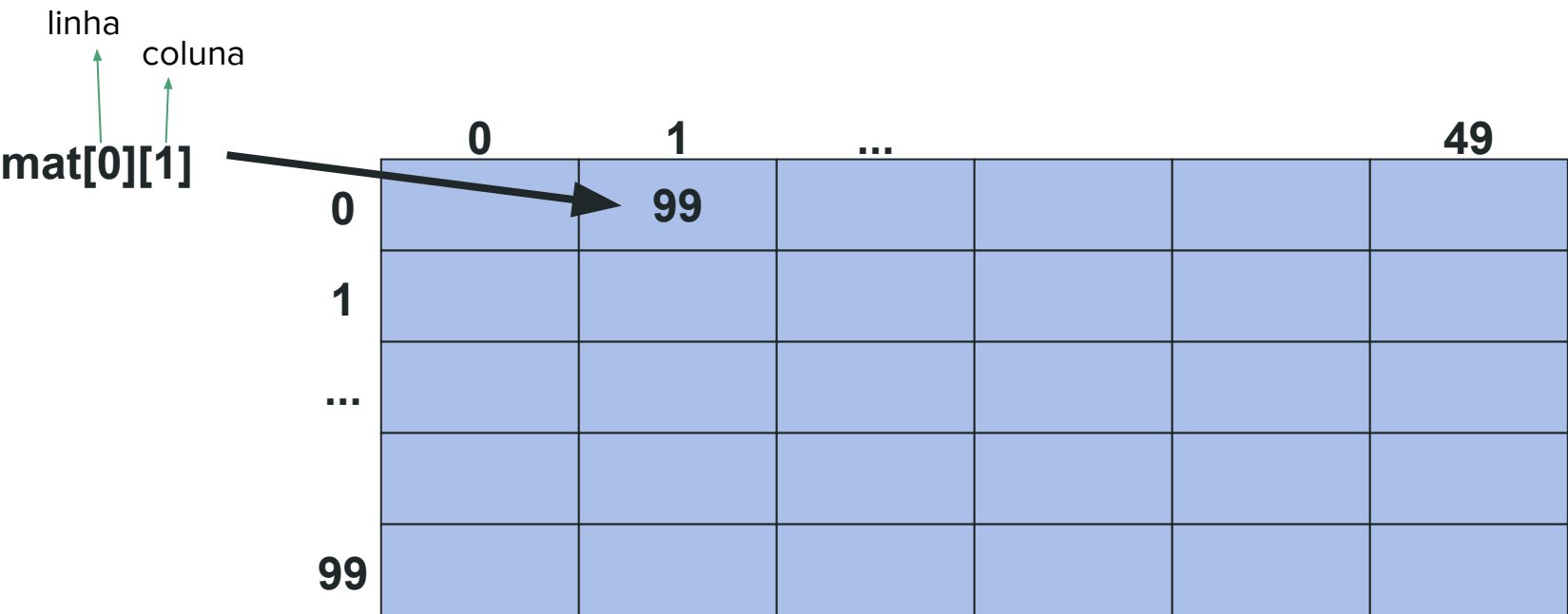
Delano Beder

\*Com base no livro Linguagem C: completa e descomplicada, de André Backes

# Matrizes

# Matrizes

- Os arrays declarados até o momento possuem apenas uma dimensão.
- Há casos em que uma estrutura com mais de uma dimensão é mais útil.
  - Por exemplo, quando os dados são organizados em uma estrutura de linhas e colunas, como uma tabela. Para isso usamos um array com duas dimensões, ou seja, uma “**matriz**”.



Matriz 100x50

# Matrizes

- Arrays bidimensionais ou “matrizes”, contêm:
  - Dados organizados na forma de uma tabela de 2 dimensões;
  - Necessitam de dois índices para acessar uma posição: um para a linha e outro para a coluna
- Declaração
  - **tipo\_variável** nome\_variável[NroLinhas][NroColunas];

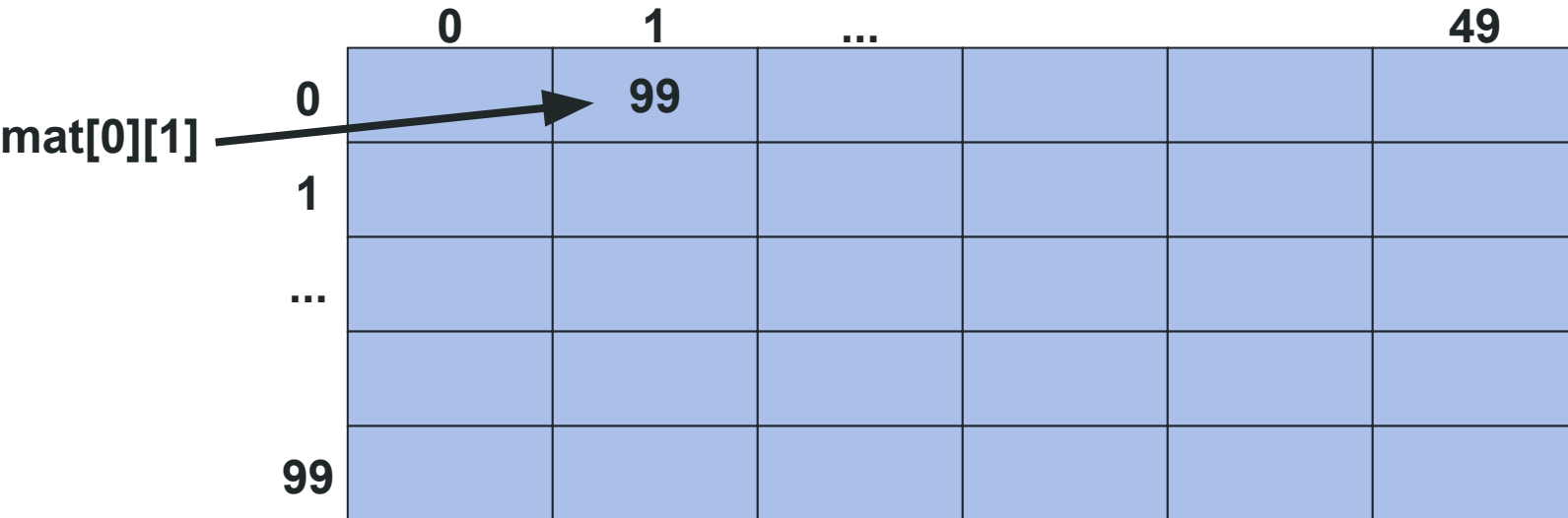
# Matrizes

- Exemplo

- Criar uma matriz que tenha 100 linhas por 50 colunas

- `int mat[100][50];`

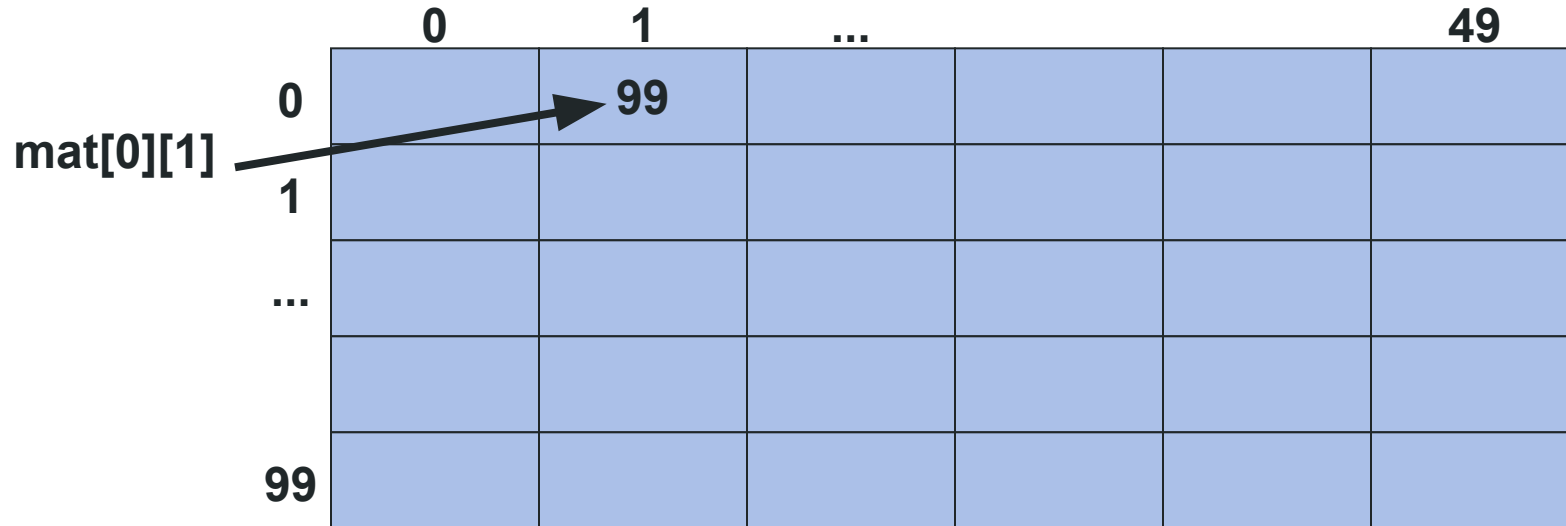
- `mat[0][1] = 99;`



	0	1	...				49
0		99					
1							
...							
99							

# Matrizes

- Em uma matriz, os elementos são acessados especificando um par de colchetes e índice para cada dimensão da matriz
  - A numeração dos índices começa sempre de zero



	0	1	...				49
0		99					
1							
...							
99							

# Matrizes

- Cada elemento de uma matriz tem todas as características de uma variável e pode aparecer em expressões e atribuições (respeitando os seus tipos)
  - `mat[0][1] = x + mat[1][5];`
  - `if (mat[5][7] > 0) ...`



# Matrizes

- Como uma matriz possui dois índices, precisamos de dois comandos de repetição para percorrer todos os seus elementos.
  - Para cada linha, temos que percorrer todas as colunas

```
#include <stdio.h>

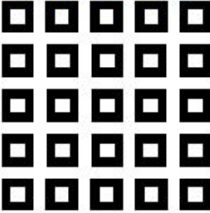
int main(){
    int mat[100][50];
    int i,j;
    for (i = 0; i < 100; i++){
        for (j = 0; j < 50; j++){
            printf("Digite o valor de mat[%d][%d]: ", i, j);
            scanf("%d", &mat[i][j]);
        }
    }

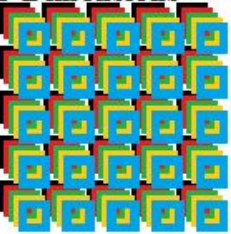
    return 0;
}
```

# Arrays Multidimensionais

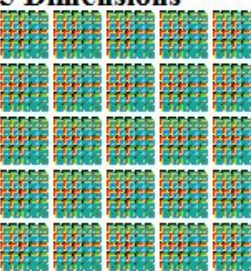
- Arrays podem ter diversas dimensões, cada uma identificada por um par de colchetes na declaração
  - `int vet[5];` // 1 dimensão
  - `float mat[5][5];` // 2 dimensões
  - `double cub[5][5][5];` // 3 dimensões
  - `int X[5][5][5][5];` // 4 dimensões

1 Dimension  


2 Dimensions  


3 Dimensions  


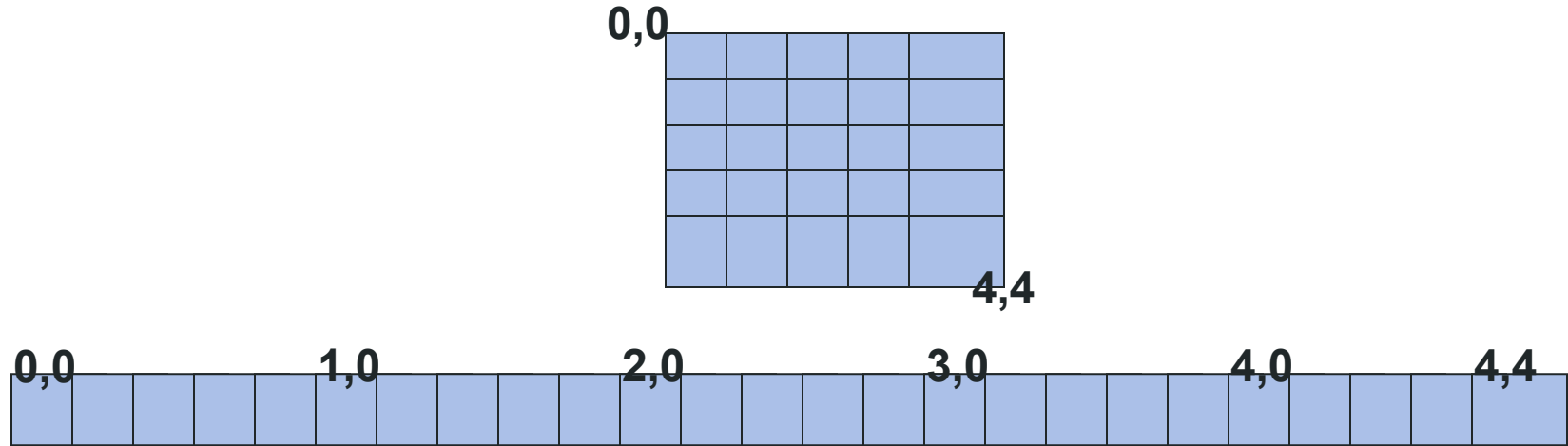
4 Dimensions  


5 Dimensions  


Etc...

# Arrays Multidimensionais

- Apesar de terem o comportamento de estruturas com mais de uma dimensão, na memória os dados são armazenados linearmente:
  - `int mat[5][5];`



# Arrays Multidimensionais

- Um array N-dimensional funciona basicamente como outros tipos de array. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.
  - `int vet[5];` // 1 dimensão
  - `float mat[5][5];` // 2 dimensões
  - `double cub[5][5][5];` // 3 dimensões
  - `int X[5][5][5][5];` // 4 dimensões

# Exercício

- Leia uma matriz de 3x3 elementos inteiros e calcule a soma dos seus elementos

# Exercício - Solução

- Leia uma matriz de 3x3 elementos inteiros e calcule a soma dos seus elementos

```
#include <stdio.h>

int main() {
    int mat[3][3];
    int i, j, soma = 0;
    printf("Digite os elementos da matriz\n");
    for(i=0; i < 3; i++)
        for(j=0; j < 3; j++){
            scanf("%d", &mat[i][j]);
        }

    for(i=0; i < 3; i++)
        for(j=0; j < 3; j++)
            soma = soma + mat[i][j];
    printf("Soma = %d\n", soma);

    return 0;
}
```

# Exercício

- Dadas duas matrizes reais de dimensão  $2 \times 3$ , fazer um programa para calcular a soma delas.

# Exercício - Solução

- Dado duas matrizes reais de dimensão 2x3, fazer um programa para calcular a soma delas.

```
#include <stdio.h>
>
int main() {
    float A[2][3], B[2][3], S[2][3];
    int i, j;

    //leia as matrizes A e B...

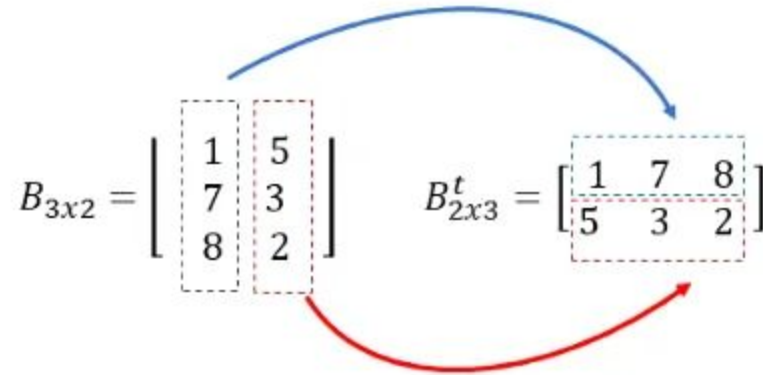
    for(i=0; i < 2; i++)
        for(j=0; j < 3; j++)
            S[i][j] = A[i][j] + B[i][j];

    return 0;
}
```



# Exercício

- Dada uma matriz de inteiros B (dimensão 3x2), fazer um programa para calcular a matriz transposta  $B^T$  (dimensão 2x3)



# Exercício - Solução

- Dada uma matriz de inteiros B (dimensão 3x2), fazer um programa para calcular a matriz transposta  $B^T$  (dimensão 2x3)

```
#include <stdio.h>

int main() {
    int B[3][2], Bt[2][3];
    int i, j;

    // Leia a matriz B

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            Bt[i][j] = B[j][i];
        }
    }
}
```

# Inicialização

- Arrays podem ser inicializados com certos valores durante sua declaração. A forma geral de um array com inicialização é:

**tipo\_da\_variável nome\_da\_variável [tam1] ... [tamN] = {dados};**

# Inicialização

- A lista de valores é composta por valores (do mesmo tipo do array) separados por vírgula.
- Os valores devem ser dados na ordem em que serão colocados na matriz

```
float vetor[3] = {1.5, 22.1, 4.56};  
int mat1[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};  
int mat2[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

# Inicialização sem tamanho

- Inicialização sem especificação de tamanho
  - Nesse tipo de inicialização, o compilador vai considerar o tamanho do dado declarado (elementos atribuídos) como sendo o tamanho do array.
  - Isto ocorre durante a compilação e não poderá mais ser mudado durante o programa.
  - Isto é útil, por exemplo, quando não queremos contar quantos caracteres serão necessários para inicializarmos uma string (array de caracteres)..

# Inicialização sem tamanho

- Inicialização sem especificação de tamanho

```
#include <stdio.h>

int main() {

    //A string mess terá tamanho 36.
    char mess[ ] = "Linguagem C: flexibilidade e poder.";

    //O número de linhas de matrxx será 5.
    int matrxx[ ][2] = { 1,2,2,4,3,6,4,8,5,10 };

    return 0;
}
```

# Referências

- BACKES, André. **Linguagem C: completa e descomplicada**. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3. Disponível na Biblioteca.