



UNIVERSIDADE
FEDERAL DO CEARÁ



CK0179 – Programação Computacional para Engenharia: Arrays: Vetores e Matrizes

Prof. Maurício Moreira Neto

Objetivo

- Aprender sobre estruturas indexadas unidimensionais (vetores) e multidimensionais (matrizes)
- Como criar vetores e matrizes usando a linguagem C

Por que utilizar arrays?

- Até agora, as variáveis que utilizamos somente podem armazenar um único valor por vez
 - Ou seja, quando tentamos armazenar um novo valor dentro desta variável, o valor antigo é sobrescrito e perdido

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float x = 10;
    printf("x = %f\n", x);
    x = 20;
    printf("x = %f\n", x);
    system("pause");
    return 0;
}
```

Saída

```
x = 10.000000
x = 20.000000
```

O que é um array?

- Array (ou vetor) é a forma mais familiar de dados estruturados
- Array é uma sequencia de elementos do mesmo tipo e que possui o mesmo nome, onde cada elemento é identificado por um índice
 - A **idéia de um array é bastante simples:** criar um conjunto de variáveis do mesmo tipo utilizando apenas um nome!

Problemas

- Imagine a seguinte situação:
 - Ler as notas de uma turma de cinco estudantes e depois imprima as notas que são maiores do que a média da turma
- Imagine como seria um algoritmo para resolver esse problema com os conhecimentos adquiridos até agora!

Solução

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float n1,n2,n3,n4,n5;
    printf("Digite a nota de 5 estudantes: ");
    scanf("%f", &n1);
    scanf("%f", &n2);
    scanf("%f", &n3);
    scanf("%f", &n4);
    scanf("%f", &n5);
    float media = (n1+n2+n3+n4+n5)/5.0;
    if(n1 > media) printf("nota: %f\n",n1);
    if(n2 > media) printf("nota: %f\n",n2);
    if(n3 > media) printf("nota: %f\n",n3);
    if(n4 > media) printf("nota: %f\n",n4);
    if(n5 > media) printf("nota: %f\n",n5);

    return 0;
}
```

Array

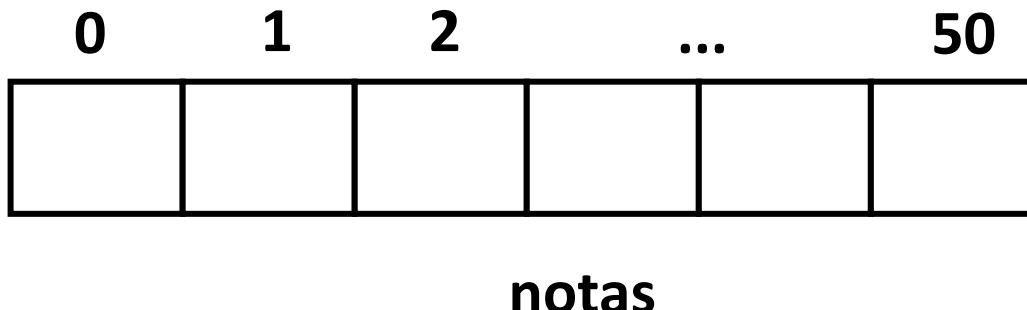
- O algoritmo anterior apresenta uma solução possível para o problema apresentado
- Entretanto, essa solução é inviável quando o número de alunos for muito grande
 - Imagine se tivéssemos de processar mais de 50 notas de alunos??

Array

- Para 50 alunos, precisamos de:
 - Uma variável para armazenar a nota de cada aluno
 - 50 variáveis
 - Um comando de leitura para cada nota
 - 50 scanf()
 - Um somatório de 50 notas
 - Um comando de teste para cada aluno
 - 50 comandos if
 - Um comando de impressão na tela para cada aluno
 - 50 printf()

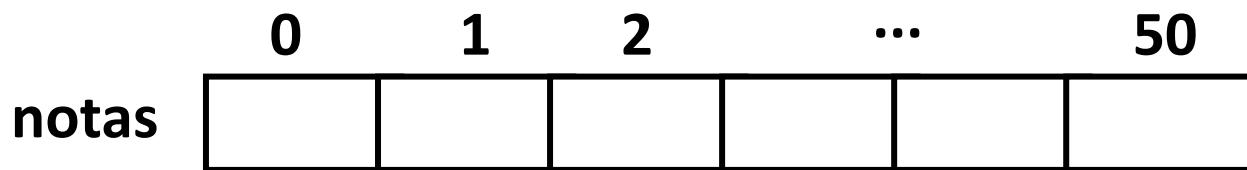
Array – Definição

- As variáveis tem relação entre si
 - Todas armazenam notas de alunos
- Podemos declará-las usando um ÚNICO nome para todos os 50 alunos
 - Notas: conjunto de 50 valores acessados por um índice
 - Isso é um **array!**



Declarando um Array

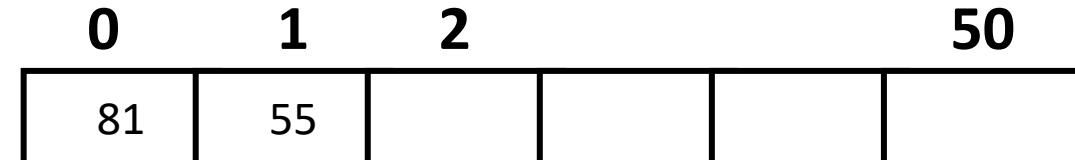
- Arrays são agrupamentos de dados adjacentes na memória
- Declaração:
 - `tipo_dado nome_array[tamanho];`
- O comando acima define um array de nome **nome_array**, capaz de armazenar **tamanho** elementos adjacentes na memória do tipo **tipo_dado**
 - Ex: `int notas [50]`



Declarando um Array

- Em um array, os elementos são acessados especificando o índice desejado entre colchetes []
- A numeração começa sempre do zero
- Isto significa que um array de 50 elementos terá índices de 0 a 49:
 - `notas[0]`, `notas[1]`, `notas[2]`, ..., `notas[49]`

```
int notas[100];
notas[0] = 81;
notas[1] = 55;
...
notas
```



Array

- Observação
 - Se o usuário digitar mais de 50 elementos em um array de 50 elementos, o programa tentará ler normalmente
 - Porém, o programa os armazenará em uma parte não reservada da memória, pois o espaço reservado para o array foi para somente 50 elementos
 - Isto pode resultar nos mais variados erros durante a execução do programa

Array

- Cada elemento do array tem todas as características de uma variável e pode aparecer em expressões e atribuições (respeitando os seus tipos)
 - `notas[2] = x + notas[2];`
 - `if (notas[2] > 60) {}`
- Ex: somar todos os elementos de notas:

```
int soma = 0;  
for(i=0;i < 100; i++)  
    soma = soma + notas[i];
```

Percorrendo um array

- Podemos usar um comando de repetição (**for**, **while** ou **do-while**) para percorrer um array
- Exemplo: somando os elementos de um array de 5 elementos

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int lista[5] = {3,51,18,2,45};
    int i, soma = 0;
    for(i = 0; i < 5; i++)
        soma = soma + lista[i];

    printf("soma = %d\n", soma);

    return 0;
}
```

Variáveis		
soma	i	lista[i]
0	-	-
3	0	3
54	1	51
72	2	18
74	3	2
119	4	45
-	5	-

Características do Array

- Características básicas de um Array
 - Estrutura homogênea, ou seja, é formado por elementos do mesmo tipo
 - Todos os elementos da estrutura são igualmente acessíveis, ou seja, o tempo e o tipo de procedimento para acessar qualquer um dos elementos do arrays são iguais
 - Cada elemento do array tem um índice próprio segundo sua posição no conjunto

Problema 2

- Voltando ao problema anterior
 - Ler as notas de uma turma de cinco estudantes e depois imprima as notas que são maiores do que a média da turma
- Como seria o algoritmo usando os arrays?

Solução 2

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float notas[5];
    int i;
    printf("Digite as notas dos estudantes\n");
    for(i = 0; i < 5; i++) {
        printf("Nota do estudante %d:", i);
        scanf("%f", &notas[i]);
    }
    float media = 0;
    for(i = 0; i < 5; i++)
        media = media + notas[i];
    media = media / 5;

    for(i = 0; i < 5; i++)
        if(notas[i] > media)
            printf("Notas: %f\n", notas[i]);

    return 0;
}
```

Solução

- Se ao invés de 5, fossem 100 alunos?

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float notas[100];
    int i;
    printf("Digite as notas dos estudantes\n");
    for(i = 0; i < 100; i++){
        printf("Nota do estudante %d:", i);
        scanf("%f", &notas[i]);
    }
    float media = 0;
    for(i = 0; i < 100; i++)
        media = media + notas[i];
    media = media / 100;

    for(i = 0; i < 100; i++)
        if(notas[i] > media)
            printf("Notas: %f\n", notas[i]);

    return 0;
}
```

Exercício

- Para um array A com 5 números inteiros, formular um algoritmo que determine o maior elemento deste array!

Solução

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i, A[5] = {3,18,2,51,45};
    int ma = A[0];

    for(i=1; i<5; i++) {
        if(ma < A[i])
            ma = A[i];
    }

    printf("Maior = %d\n", ma);

    return 0;
}
```

Variáveis		
ma	i	A[i]
3	0	3
18	1	18
51	2	2
	3	51
	4	45
	5	

Copiando um Array

- Não se pode fazer atribuição de arrays inteiros, apenas de suas posições individualmente

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int v[5] = {1,2,3,4,5};
    int v1[5];

    v1 = v; //ERRADO!

    int i;
    for(i=0; i<5; i++)
        v1[i] = v[i]; //CORRETO

    return 0;
}
```

Arrays Bidimensionais - Matrizes

- Os arrays declarados até o momento possuem apenas uma dimensão e, portanto, são tratados como uma lista de variáveis
 - Porém, 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”

Arrays Bidimensionais - Matrizes

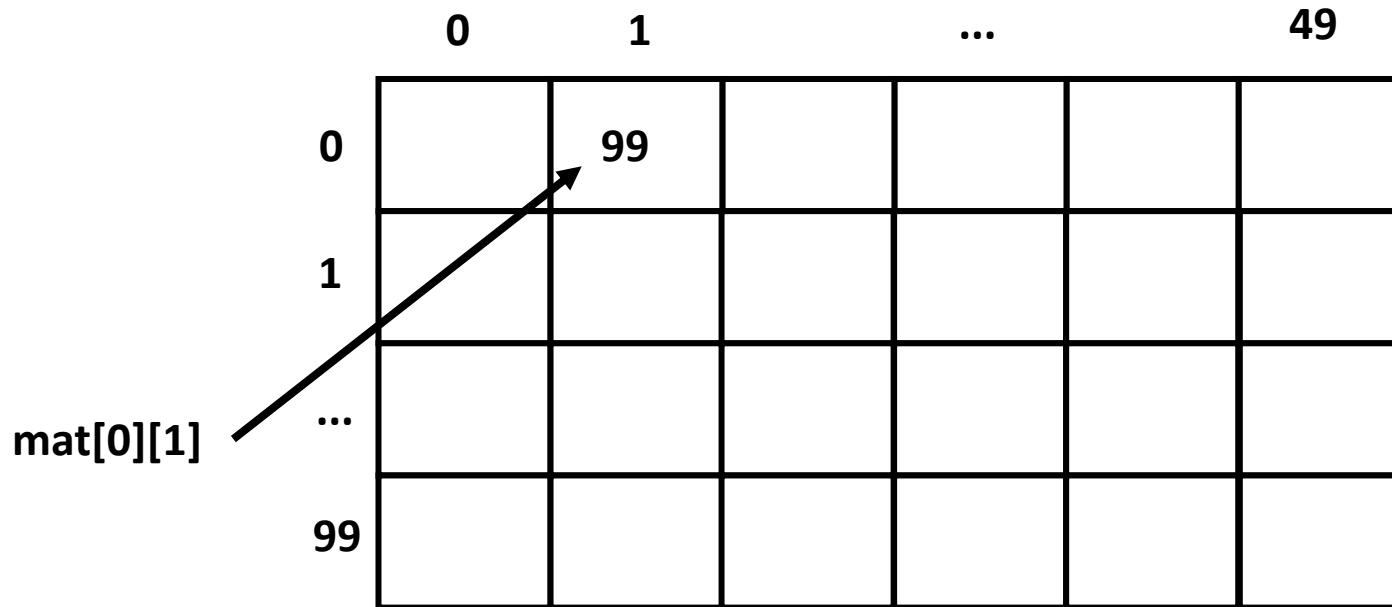
- Arrays bidimensionais ou “matrizes”, contém:
 - Dados organizados na forma de uma tabela de 2 dimensões
 - Necessitam de dois índices para acessar uma posição: um para a linha e outro para a coluna
- Declaração
 - `tipo_variável nome_variável [linha] [colunas];`

Arrays Bidimensionais - Matrizes

- Exemplo

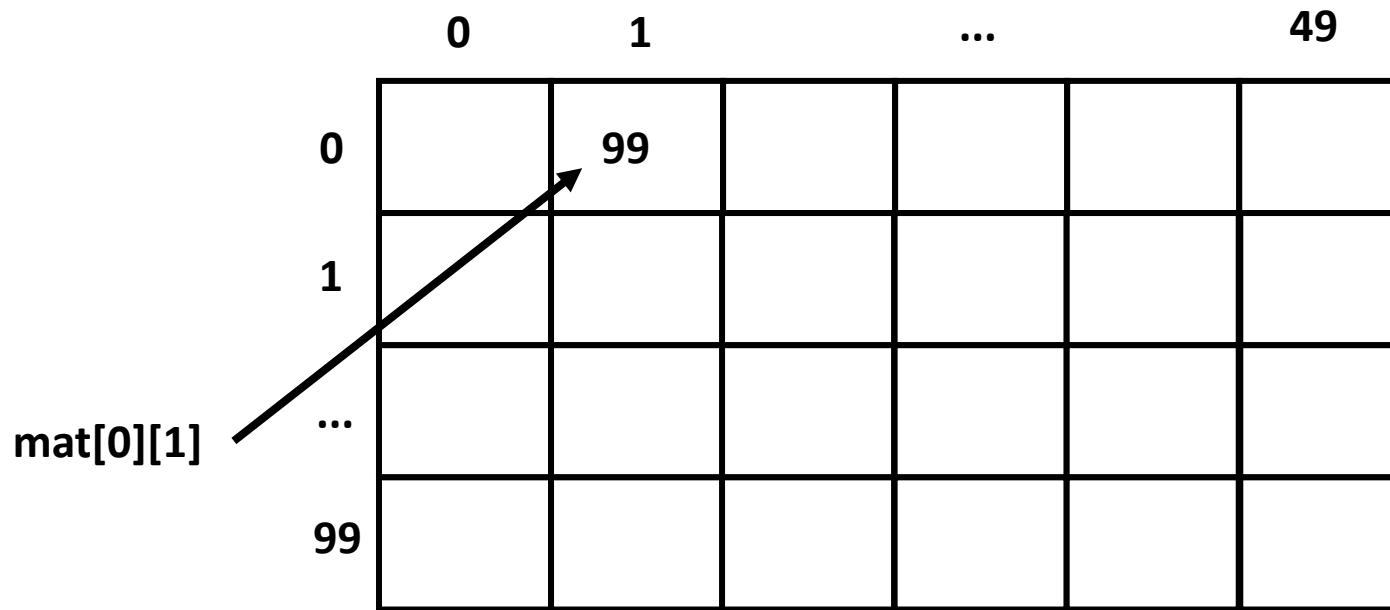
- Criar uma matriz que tenha 100 linhas por 50 colunas

- ```
int mat[100][50];
```
- ```
mat[0][1] = 99;
```



Arrays Bidimensionais - Matrizes

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



Arrays Bidimensionais - Matrizes

- Cada elemento da 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) {}`

Arrays Bidimensionais - Matrizes

- Como uma matriz possui dois índices, precisamos de dois comandos de repetição para percorrer todos os seus elementos

```
#include <stdio.h>
#include <stdlib.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;
}
```

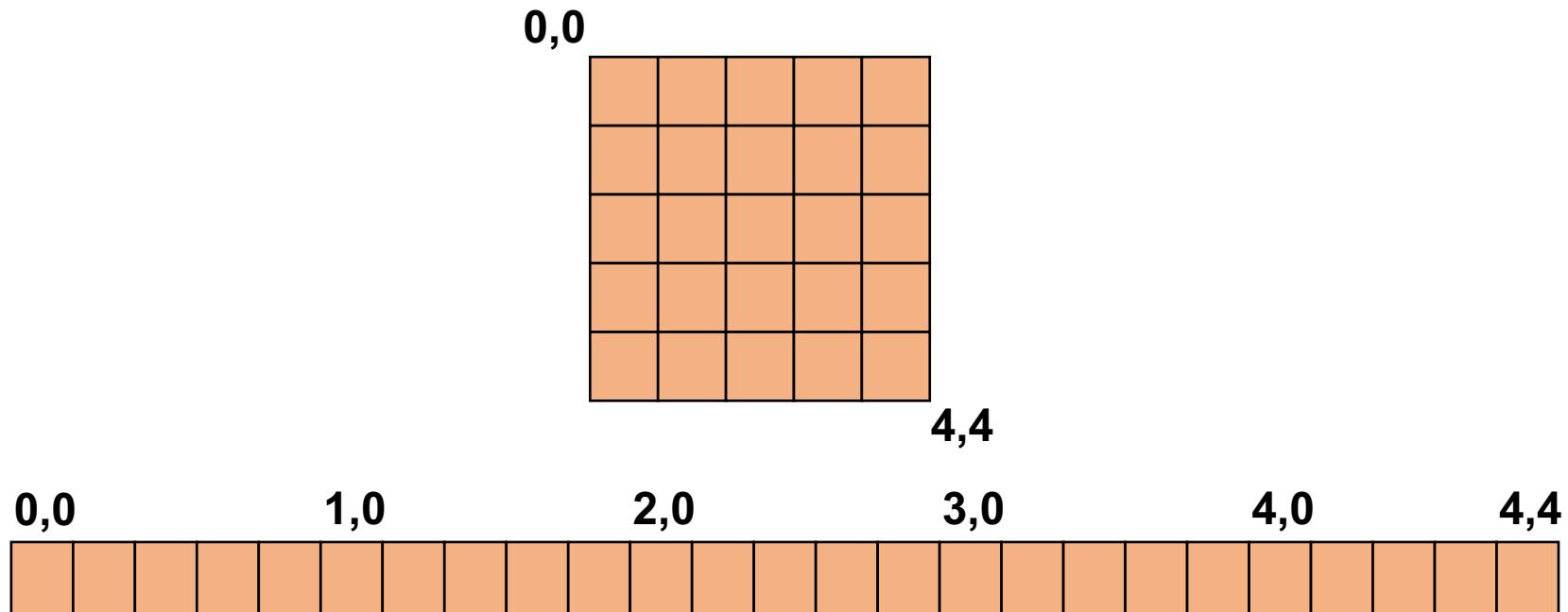
Arrays Bidimensionais - Matrizes

- Arrays podem ter diversas dimensões, cada uma identificada por um par de colchetes na declaração

- `int vet[5]; // 1 dimensões`
- `float mat[5][5]; // 2 dimensões`
- `double cub[5][5][5]; // 3 dimensões`
- `int x[5][5][5][5]; // 4 dimensões`

Array Multidimensionais

- Apesar de terem o comportamento de estruturas com mais de uma dimensão, na memória os dados são armazenados linearmente:



Array 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 (índice mais interno)
 - int vet[5]; // 1 dimensões
 - float mat[5][5]; // 2 dimensões
 - double cub[5][5][5]; // 3 dimensões
 - int x[5][5][5][5]; // 4 dimensões

Exercício

- Faça um programa que leia uma matriz de 3x3 elementos inteiros e calcule a soma dos seus elementos



Solução

```
#include <stdio.h>
#include <stdlib.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

- Faça um programa que dado duas matrizes reais de dimensão 2×3 , fazer um programa para calcular a soma delas



Solução

```
#include <stdio.h>
#include <stdlib.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;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    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}};

    char str1[10] = {'J', 'o', 'a', 'o'};
    char str2[10] = "Joao";

    char nomes[3][10] = {"Joao", "Maria", "Jose"};

    return 0;
}
```

Inicialização sem tamanho

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

Inicialização sem tamanho

- Inicialização sem especificação de tamanho

```
#include <stdio.h>
#include <stdlib.h>
int main() {

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

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

    return 0;
}
```

Dúvidas?



Referências

- André Luiz Villar Forbellone, Henri Frederico Eberspächer, Lógica de programação (terceira edição), Pearson, 2005, ISBN 9788576050247.
- Ulysses de Oliveira, Programando em C - Volume I - Fundamentos, editora Ciência Moderna, 2008, ISBN 9788573936599.
- **Slides baseados no material do site "Linguagem C Descomplicado"**
 - <https://programacaodescomplicada.wordpress.com/complementar/>