Tópico 3

Arrays

Introdução

- An array is a collection of similar data elements.
- These data elements have the same data type.
- Elements of arrays are stored in consecutive memory locations and are referenced by an index (also known as the subscript).
- Declaring an array means specifying three things:

  Data type - what kind of values it can store. For example, int, char, float

  Name - to identify the array

  Size - the maximum number of values that the array can hold
- Arrays are declared using the following syntax:

  type name[size];

Acessando Elementos de um Array

- Para **acessar todos** os **elementos** de um **array**, devemos usar um **laço** (loop)
- Ou seja, podemos acessar todos os elementos de uma matriz variando o valor do subscrito na matriz
- Mas observe que o subscrito deve ser um valor integral ou uma expressão avaliada como um valor integral.

```
int i, valor[10];
for(i=0; i < 10; i++)
    valor[i] = -1;
```

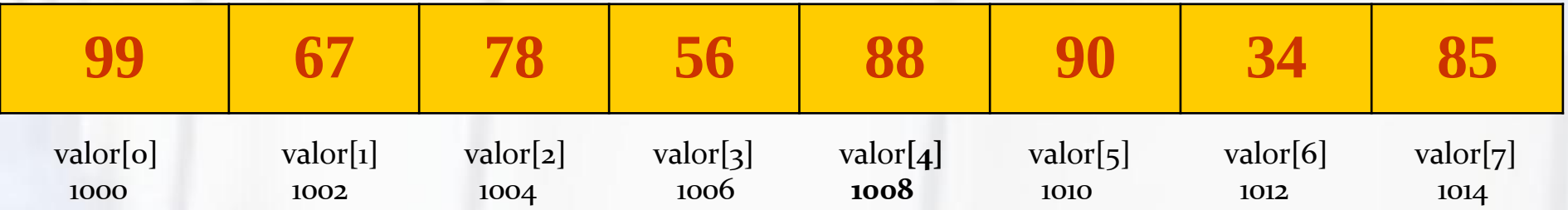
Calculando o Endereço dos Elementos de um Array

- Endereço do elemento de dados:

  A[k] = BA(A) + w (k - limite_inferior)
- **A** é a **matriz**
- **k** é o **índice** do elemento cujo endereço temos que calcular
- **BA** é o **endereço básico** da matriz A
- **w** é o **tamanho da palavra** de um elemento na memória.

  Por exemplo, o tamanho do short int é 2

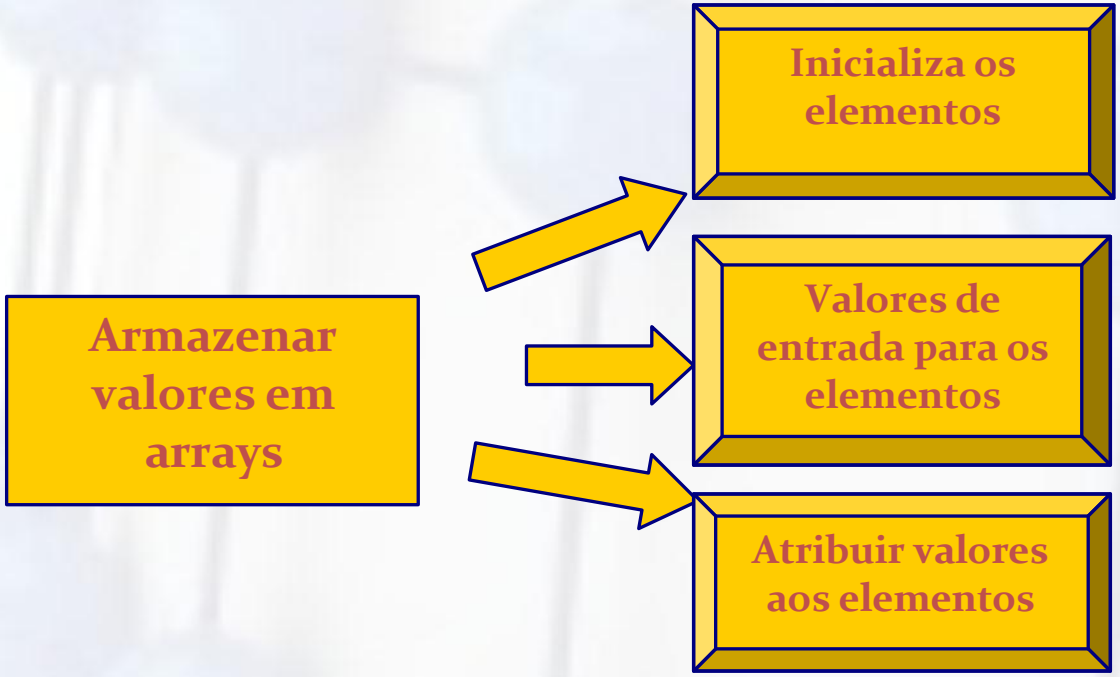
Calculando o Endereço dos Elementos de um Array



valor[4] = 1000 + 2(4 – 0)

= 1000 + 2(4) = 1008

Armazenamento de valores em Arrays



Inicializando Arrays durante a declaração

int valor [5] = {90, 98, 78, 56, 23};

Entrada de valores do teclado

```
int i, valor[10];
for(i=0;i<10;i++)
        scanf("%d", &valor[i]);
```

Atribuição de valores a elementos individuais

```
int i, arr1[10], arr2[10];
for(i=0;i<10;i++)
        arr2[i] = arr1[i];
```

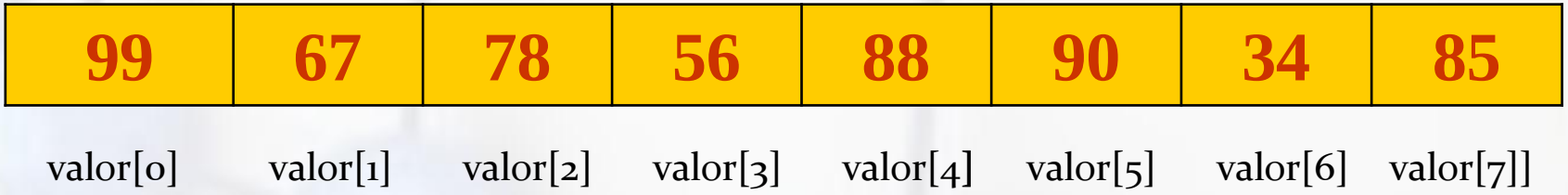
Calculando o comprimemento de um Array

comprimento = limite_superior – limite_inferior + 1

Onde

limite_superior é o índie do último elemento

limite_inferior é o índice do primeiro elemento da array



Perceba que: limite_inferior = 0 e limite_superior = 7

Logo, comprimento = 7 – 0 + 1 = 8

Ler e Mostrar *N* Números using um Array

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i=0, n, arr[20];
        printf("\n Entre com o numero de elementos : ");
        scanf("%d", &n);
        printf("\n Entre com os elementos : ");
        }
```

Ler e Mostrar *N* Números using um Array

```
        for(i=0; i < n; i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d", &arr[i]);
        }
        printf("\n Elementos da matriz:   ");
        for(i=0;i<n;i++)
                printf("arr[%d] = %d\t", i, arr[i]);
        return 0;
}
```

Ler e Mostrar *N* Números using um Array – Programa 2

```
#include <stdio.h>
int main() {
   int vetor[20];     int tamanho;
   printf("Digite o tamanho do vetor (max 20): ");
   scanf("%d", &tamanho);
   if (tamanho > 20) {
      printf("Tamanho excede o limite do vetor.\n");
      return 1; // Encerra o programa com erro
                                          }
   for (int i = 0; i < tamanho; i++) {
      printf("Digite o valor para o índice %d: ", i);
      scanf("%d", &vetor[i]);    }
   printf("Vetor preenchido:\n");
   for (int i = 0; i < tamanho; i++) {
      printf("%d ", vetor[i]);    }
   printf("\n");
   return 0;
}
```

Inserindo um Elemento em um Array

Passos para inserir um novo elemento no final de um array

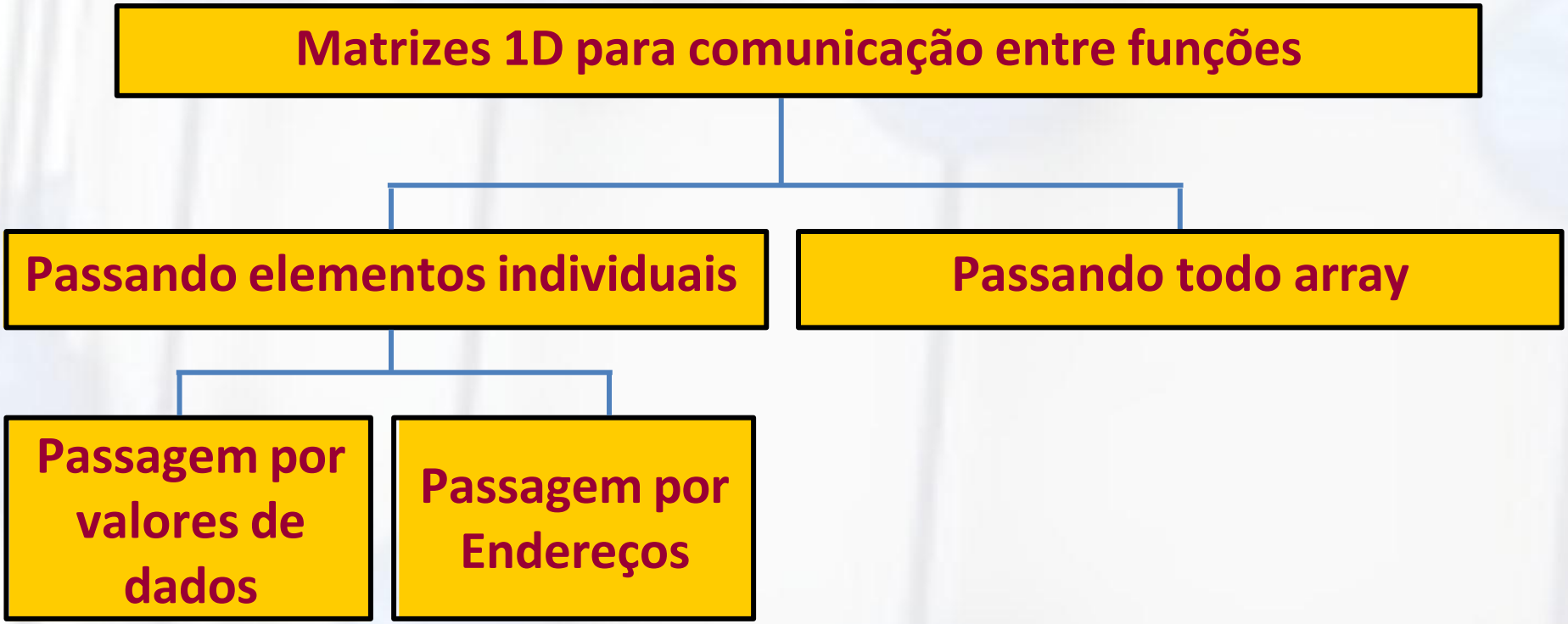
```
1: Faça limite_superior = limite_superior + 1
2: Faça A[limite_superior]=  VAL
3: Sair
```

Deletando um Elemento de um Array

Algoritmo para excluir um elemento do final do array

```
1: Faça limite_superior = limite_superior - 1
2: Sair
```

Passando Arrays para Funções



Passando Arrays para Funções

Passando **valores** de dados

```
main()
{
    int arr[5] ={1, 2, 3, 4, 5};
    func(arr[3]);
}
```

```
void func(int num)
{
   printf("%d", num);
}
```

Passando Arrays para Funções

Passando **endereços**

```
main()
{
    int arr[5] ={1, 2, 3, 4, 5};
    func(&arr[3]);
}
```

```
void func(int *num)
{
   printf("%d", *num);
}
```

Passando Arrays para Funções

Passando **todo** array

```
main()
{
   int arr[5] ={1, 2, 3, 4, 5};
   func(arr);
}
```

```
void func(int arr[5])
{
  int i;
  for(i=0;i<5;i++)
    printf("%d", arr[i]);
}
```

Ponteiros e Arrays

- O conceito de **array** está muito **associado** ao conceito de **ponteiro**

- O **nome de um array** é, na verdade, um **ponteiro** que aponta para o **primeiro elemento** do array

```
int *ptr;
ptr = &arr[0];
```

Ponteiros e Arrays

- Se a variável ponteiro ptr contém o endereço do primeiro elemento do array, então o endereço dos elementos sucessivos pode ser calculado escrevendo ptr++

```
int *ptr = &arr[0];

ptr++;

printf (“O valor do 2o elemento do array é: %d”, *ptr);
```

Arrays de Ponteiros

- Um array de ponteiros pode ser declarado como:

```
int *ptr[10];
```

- A instrução acima declara um array de 10 ponteiros, onde cada um dos ponteiros aponta para uma variável inteira. Por exemplo, observe o código fornecido a seguir:

```
int *ptr[10];
int p=1, q=2, r=3, s=4, t=5;
ptr[0]=&p;  ptr[1]=&q; ptr[2]=&r; ptr[3]=&s; ptr[4]=&t;
```

Arrays de Ponteiros

- Você pode dizer qual será a saída da seguinte declaração?

  printf(“\n %d”, *ptr[3]);

Arrays de Ponteiros

- A saída será **4** porque ptr[3] **armazena o endereço** da variável inteira se *ptr[3] irá, portanto, imprimir o valor de s que é 4

Exercícios:

1. Objetivo: Escreva um programa que LEIA e MOSTRE n números usando um ARRAY
2. Objetivo: Escreva um programa que encontre a MEDIA de n numeros, usando ARRAY. Media Aritmética: (soma dos n numeros) / quantidade de numeros (n)
3. Objetivo: Escreva um programa que imprima a posição do menor número de n números usando matrizes.

elton.galvao@sistemafiep.org.br

Arrays Bidimensionais

- Um array **bidimensional** é especificada usando **dois subscritos**, onde um subscrito denota **linha** e o outro denota **coluna**.

- Para C, um array bidimensional é como um **array de arrays unidimensionais**

Arrays Bidimensionais

- Um array bidimensional é declarada como:
  tipo_dado nome_array[tam_linhas][tam_colunas];

Arrays Bidimensionais

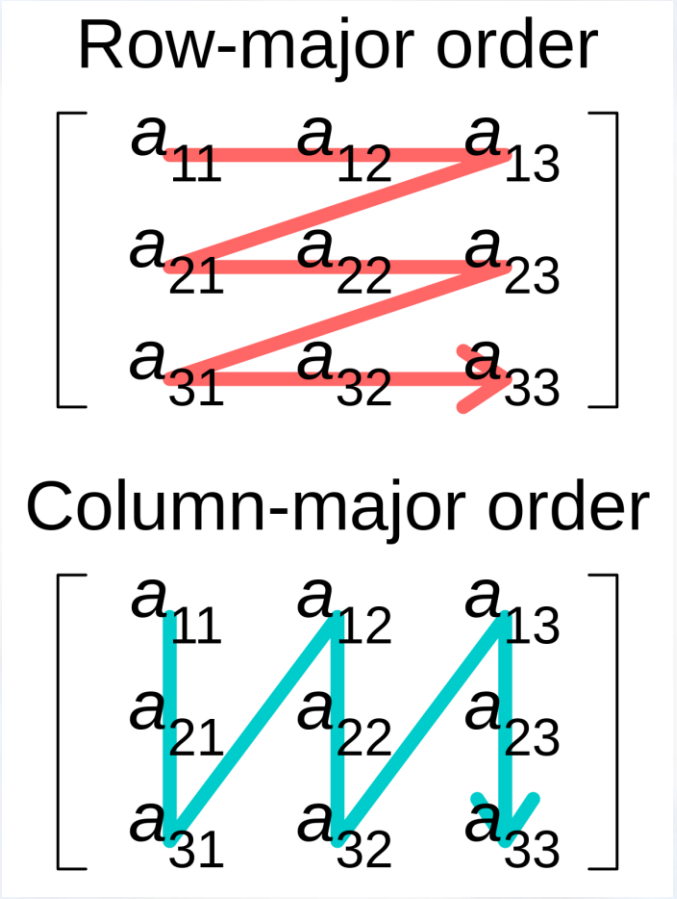
- Portanto, uma matriz bidimensional m × n é uma matriz que contém elementos de dados m × n e cada elemento é acessado usando dois subscritos, i e j, onde i <= m e j <= n
int valor[3][4];

Linhas/ Colunas	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	valor[0][0]	valor[0][1]	valor[0][2]	valor[0][3]
Linha 1	valor[1][0]	valor[1][1]	valor[1][2]	valor[1][3]
Linha 2	valor[2][0]	valor[2][1]	valor[2][2]	valor[2][3]

Array Bidimensional

Representação da Memória de um Array Bidimensional

- Duas maneiras de armazenar um array 2D na memória:
  - Ordem principal da linha
  - Ordem principal da coluna



Representação da Memória de um Array Bidimensional

- Na **ordem principal da linha**
  - Os elementos da 1ª linha são armazenados antes dos elementos da segunda e terceira linhas
  - Ou seja, os elementos do array são armazenados **linha por linha**, onde **n** elementos da primeira linha ocuparão as primeiras enésimas posições.

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) \\ (1,0) & (1,1) & (1,2) & (1,3) \\ (2,0) & (2,1) & (2,2) & (2,3) \end{bmatrix}$$

(0,0) (0, 1) (0,2) (0,3) (1,0) (1,1) (1,2) (1,3) (2,0) (2,1) (2,2) (2,3)

Representação da Memória de um Array Bidimensional

- Na **ordem principal da coluna**
  - Os elementos os elementos da **primeira coluna** são armazenados antes dos elementos da segunda e terceira colunas.
  - Ou seja, os elementos da matriz são armazenados **coluna por coluna**, onde **n** elementos da primeira coluna ocuparão as primeiras enésimas posições.

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) \\ (1,0) & (1,1) & (1,2) & (1,3) \\ (2,0) & (2,1) & (2,2) & (2,3) \end{bmatrix}$$

(0,0) (1,0) (2,0) (0,1) (1,1) (2,1) (0,2) (1,2) (2,2) (0,3) (1,3) (2,3)

Inicializando Arrays Bidimensionais

- Um array bidimensional é inicializado da mesma forma que um array unidimensional é inicializado. Por exemplo:

  int valor[6] = {90, 87, 78, 68, 62, 71};

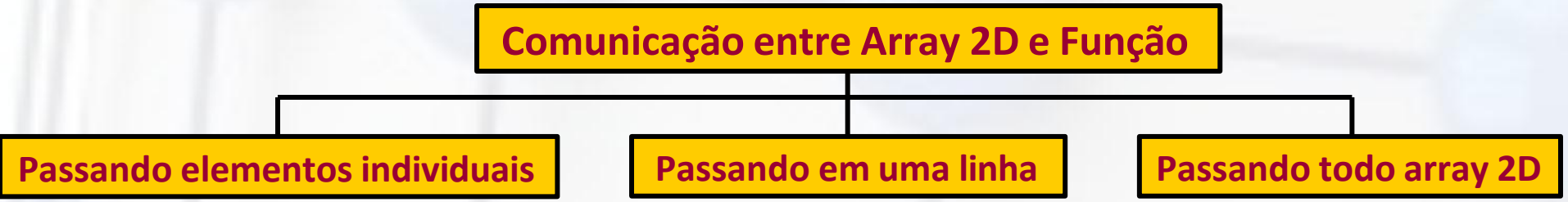
  int valor[2] [3] = {{90,87,78}, {68, 62, 71}};

Quando usar

Quando usar vetor unidimensional vs. matriz bidimensional

1. **Vetor unidimensional**:
    1. Use quando os dados são uma simples lista ou sequência.
    2. Exemplo: Lista de notas de alunos, temperaturas diárias, etc.
2. **Matriz bidimensional**:
    1. Use quando os dados têm uma estrutura tabular (linhas e colunas).
    2. Exemplo: Tabela de notas de alunos por disciplina, matriz de pixels de uma imagem, etc.

Passando Arrays 2D para Funções



-

Explicação dos métodos

Passando elementos individuais:

- Cada elemento da matriz é passado individualmente para a função.
- Útil quando você precisa processar elementos de forma independente.

Passando uma linha da matriz:

- Uma linha da matriz (um array unidimensional) é passada para a função.
- Útil quando você precisa processar uma linha inteira de cada vez.

Passando a matriz inteira:

- A matriz 2D inteira é passada para a função.
- No padrão C89/C90, é necessário especificar o tamanho das colunas na declaração da função.

Observações

- No padrão C89/C90, não é possível passar uma matriz 2D para uma função sem especificar o tamanho das colunas. Por exemplo, int matriz[][] não é válido.

- Se o tamanho da matriz for dinâmico (não conhecido em tempo de compilação), você precisará usar alocação dinâmica de memória (com malloc e free), mas isso foge ao escopo do padrão C89/C90.

Ponteiros e Arrays 2D - Resumo das técnicas

Técnica	Descrição
Acesso a elementos	Use aritmética de ponteiros para acessar elementos individuais.
Acesso a linhas	Use um ponteiro para uma linha (**int (*ptr)[colunas]**).
Passar toda matriz	Use **int (*matriz)[colunas]** para passar a matriz inteira.

Dicas importantes

1. Tamanho das colunas:

- Ao trabalhar com arrays 2D, o tamanho das colunas deve ser conhecido em tempo de compilação.
- Por exemplo**, int (*matriz)[3]** só funciona para matrizes com 3 colunas.

2. Aritmética de ponteiros:

- A aritmética de ponteiros é baseada no tipo do ponteiro. Por exemplo, **ptr + 1** avança para o próximo elemento do tipo apontado.

3. Evitar confusão:

- **int *ptr** é um ponteiro para um int.
- **int (*ptr)[3]** é um ponteiro para um array de 3 int.

Matrizes 2D

Resumo das Situações Reais:

Passagem	Exemplo Prático	Quando Usar
Elemento Individual	Validar notas de alunos	Processar itens um a um (ex.: validação).
Uma Linha	Calcular vendas mensais	Operações em linhas completas.
Matriz Inteira	Verificar vencedor no jogo da velha	Análise global da matriz.

Por Que Esses Exemplos São Úteis?

1. **Elementos Individuais**:
    1. Permite aplicar regras ou transformações específicas a cada dado (ex.: validar notas, converter unidades).
2. **Uma Linha**:
    1. Otimiza operações em conjuntos de dados relacionados (ex.: totalizar vendas, calcular médias).
3. **Matriz Inteira**:
    1. Facilita a análise de estruturas completas (ex.: tabuleiros, imagens, mapas).

1. Elementos Individuais: Planilha de Notas de Alunos

Cenário:

Você é um professor e quer calcular a média de cada aluno em uma planilha de notas (linhas = alunos, colunas = notas de provas). Cada nota precisa ser validada antes do cálculo.

Aplicação:

- Cada nota é passada individualmente para a função *validarNota*.
- Útil quando você precisa processar **cada elemento de forma isolada** (ex.: validação, formatação).

2. Uma Linha: Análise de Vendas por Mês

Cenário:

Você gerencia uma loja e quer calcular o total de vendas de **cada mês** (cada linha da matriz representa um mês, e as colunas são semanas).

Aplicação:

- A função recebe uma linha inteira (*vendas[mes]*) para calcular o total.
- Ideal para operações em **linhas específicas** (ex.: totalizar, filtrar).

3. Matriz Inteira: Jogo da Velha

Cenário:

Você está desenvolvendo um jogo da velha e precisa verificar se há um vencedor após cada jogada (a matriz 3x3 representa o tabuleiro).

Aplicação:

- A função *verificarVencedor()* recebe **a matriz inteira** para analisar o tabuleiro.
- Necessário quando a lógica depende de **todos os elementos** (ex.: jogos, processamento de imagens).

Matrizes 2D

- Programas...

Ponteiros e Array Tridimensionais

- Declarando um ponteiro para uma matriz tridimensional:

```
int arr[2] [2] [2] = {1,2,3,4,5,6,7,8};
int (*parr)[2] [2];
parr = arr;
```

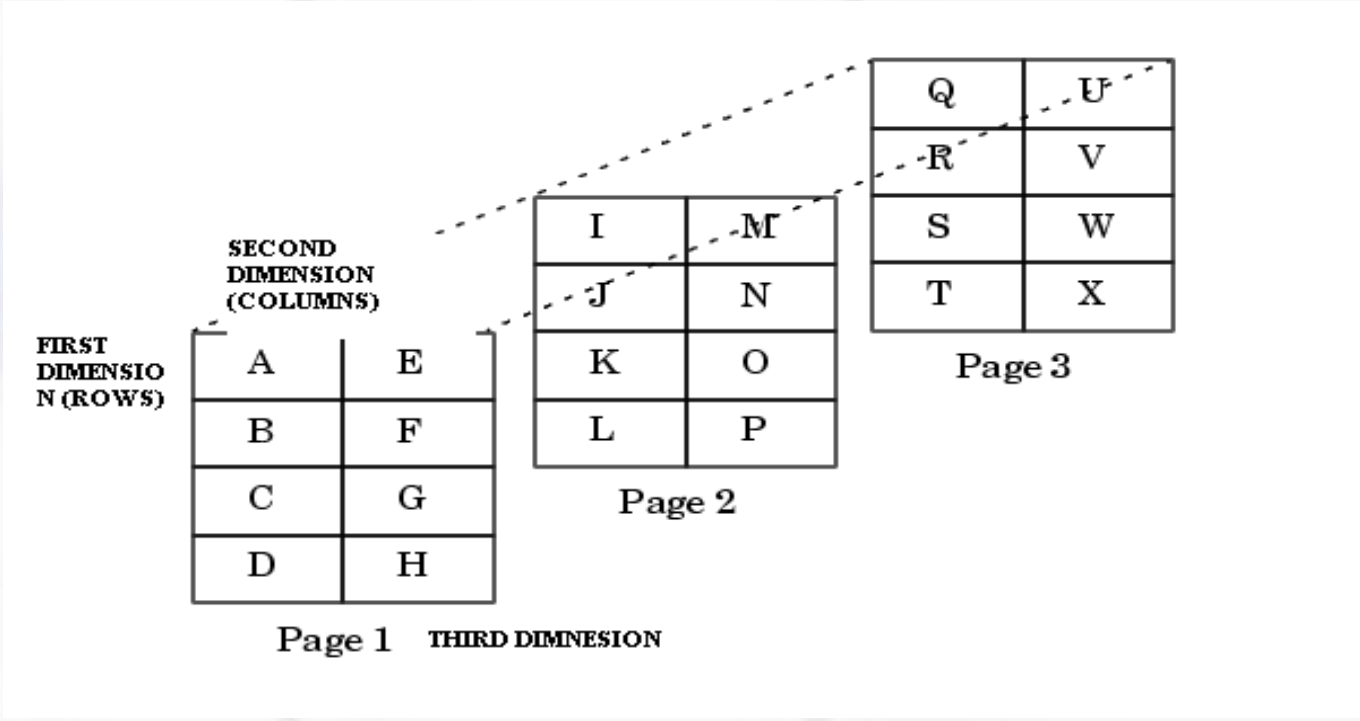
- Pode-se acessar um elemento de um array 3D escrevendo: arr [i] [j] [k] = *(*(*(*(arr + i) + j) + k)

Array 3D

```
int myFirst3DArray [3] [4] [2] =
{
  { {10, 11}, {12, 13}, {14, 15}, {16, 17} },
  { {18, 19}, {20, 21}, {22, 23}, {24, 25} },
  { {26, 27}, {28, 29}, {30, 31}, {32, 33} }
};
```

Arrays Multidimensionais

- É um array de arrays
- Assim como temos 1 índice para array unidimensional, 2 índices para array bidimensional, da mesma forma temos **n** índices em um array multidimensional



Inicializando Arrays Multidimensionais

- Um array multidimensional é declarado e inicializado da mesma maneira que declaramos e inicializamos arrays unidimensionais e bidimensionais

Aplicações de Arrays

- Amplamente usados para implementar vetores matemáticos, matrizes e outros tipos de tabelas retangulares.
- Também são usados para **implementar outras Estruturas de Dados** como tabelas hash, deques, filas, pilhas e strings.
- Também podem ser usadas para **alocação de memória dinâmica**