

# ***Ponteiros***

Prof. Raimundo Barreto  
DCC/ICE/UFAM

---

# **Fundamentos**

# Fundamentos

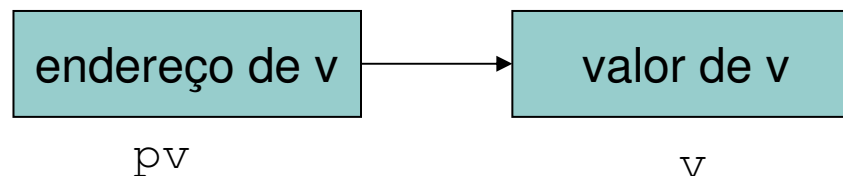
---

- Um ponteiro é uma variável que representa a **localização** (em vez do valor) de um item de dado
- Na memória do computador, todo item de dado armazenado ocupa uma ou mais células contíguas de memória.
- Suponha que  $v$  seja uma variável. O endereço da localização de  $v$  pode ser determinado pela expressão  $\&v$ .

# Fundamentos

---

- $\&$  é um operador unário que avalia o endereço do seu operando
- Podemos assinalar o endereço de  $v$  a outra variável  $pv$ , assim:
  - $pv = \&v;$



# Fundamentos

---

- Qual será o valor de `v`?

```
main()
{
    int u=3;
    int v;
    int *pu;

    pu = &u;
    v = *pu;

    printf("v=%d\n", v);
}
```

# Fundamentos

---

- u1 e u2 são equivalentes?

```
main()
{
    int u1, u2;
    int v=3;
    int *pv;

    u1 = 2 * (v + 5);
    pv = &v;
    u2 = 2 * (*pv + 5);

    printf("u1=%d  u2=%d\n", u1, u2);
}
```

## Fundamentos

---

- O que aconteceria se fosse incluída a seguinte instrução  $*p_v = 0$ ?

# Declaração de Ponteiros

---

- Forma geral:

- `tipo *var_ptr;`

- Exemplos:

- `float *pv;`

- `int *ptr;`

- `char *str;`

- Inicialização

```
#define NULL 0
```

```
float *pv = NULL;
```



---

# **Passando ponteiros para uma função**

## Passando ponteiros para uma função

---

- Ponteiros são usados para que itens de dados sejam acessados e alterados por funções e devolvidos alterados para a parte chamadora.
- São chamados de **passagem por referência**.
- Se a **passagem for valor**, o item de dado é **copiado** para a função e **não** produz efeito colateral na parte chamadora.

## Passando ponteiros para uma função

---

- Mostra a diferença entre parâmetros por referência e por valor.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

## Passando ponteiros para uma função

---

- Analisar uma linha de texto e dizer a quantidade de vogais, consoantes, dígitos, brancos e outros símbolos. Definir uma função específica para fazer tal análise.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

# Passando ponteiros para uma função

---

- Observação com relação a função `scanf`
  - `scanf ("%[AEIOUaeiou]");`
    - Só aceita vogais maiúsculas e minúsculas
  - `scanf ("%[^\\n]");`
    - Aceita qualquer símbolo exceto o nova linha
  - Exige que os argumentos que sejam variáveis comuns sejam precedidos por um **ampersand** (&). Entretanto, nomes de matrizes são exceções.

# Passando ponteiros para uma função

---

```
main()
{
    char item[20];
    int num_peca;
    float custo;
    ...
    scanf("%s %d %f", item, &num_peça, &custo);
    ...
}
```

**O nome da matriz já representa o endereço da matriz**

## Passando ponteiros para uma função

---

E se eu quiser ler um elemento particular da matriz?

```
scanf ("%f", &lista[cont]);
```

Nesse caso, o nome do elemento da matriz tem que aparecer precedido por um ampersand

---

# **Ponteiros e matrizes unidimensionais**



# Ponteiros e matrizes unidimensionais

---

- O nome de uma matriz é, na verdade, um **ponteiro** para o primeiro elemento da matriz.
  - O **endereço** do primeiro elemento da matriz pode ser expresso por `&x[0]` ou simplesmente `x`.
  - O endereço do segundo elemento da matriz pode ser expresso por `&x[1]` ou como `(x+1)`.
  - O endereço do *i*-ésimo elemento da matriz pode ser expresso por `&x[i]` ou como `(x+i)`.

## Ponteiros e matrizes unidimensionais

---

- Quando dizemos  $(x+i)$  estamos especificando uma localização que é  $i$  elementos além do primeiro.
- O programador não precisa se preocupar com o número de células da memória associada com cada tipo de elemento da matriz.
- Obviamente que  $x[i]$  e  $*(x+i)$  representam o conteúdo desses endereços.

# Ponteiros e matrizes unidimensionais

---

- Ilustração da relação entre elementos da matriz e seus endereços.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

## Ponteiros e matrizes unidimensionais

---

- É possível definir matrizes como ponteiros ao invés de matrizes convencionais.
- Sintaticamente as duas definições são equivalentes.
- Entretanto, uma definição convencional de matriz já reserva a memória suficiente. O que não ocorre se for definida como ponteiro.
- Exige-se que haja alguma forma de atribuição da memória inicial (`malloc`).

# Ponteiros e matrizes unidimensionais

---

- Reordenando uma lista de números
- Observações
  - Leitura dos elementos da matriz
  - Processo de ordenação
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

---

# **Ponteiros e matrizes multidimensionais**

# Ponteiros e matrizes multidimensionais

---

- Vimos que uma vez que matrizes **unidimensionais** podem ser representadas por um ponteiro (o nome da matriz) e por um deslocamento (o subscript), é razoável imaginar que uma matriz **multidimensional** pode ser representada com uma *notação ponteiro equivalente*.
- Uma matriz bidimensional é, na realidade, uma **coleção de** matrizes unidimensionais.
- Podemos defini-la como um ponteiro para um grupo de matrizes unidimensionais contíguas.

# Ponteiros e matrizes multidimensionais

---

- Declaração

- `tipo (*ptr) [expr2]`
- Ao invés de: `tipo matriz [expr1] [expr2]`

- Generalização

- `tipo (*ptr) [expr2] [expr3] ... [exprn]`
- Que substitui: `tipo matriz [expr1]  
[expr2] ... [exprn]`



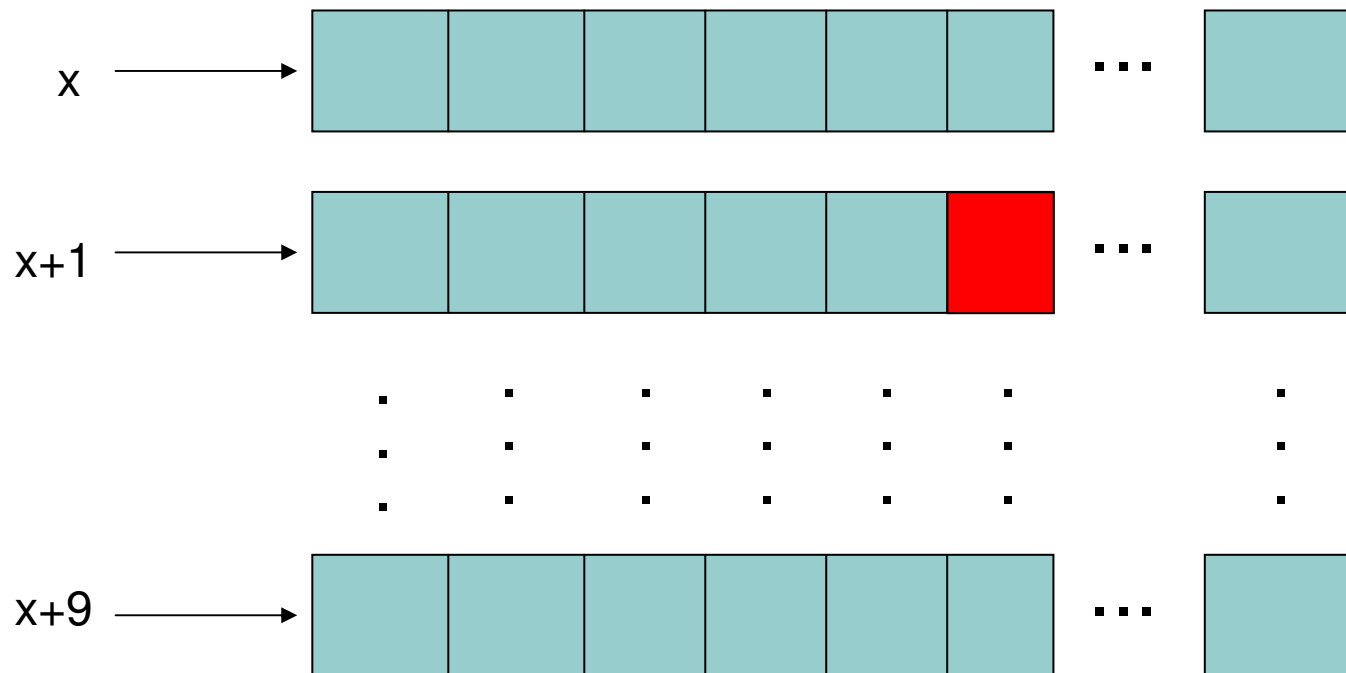
# Ponteiros e matrizes multidimensionais

---

- Exemplo: suponha que `x` seja uma matriz inteira bidimensional com 10 linhas e 20 colunas. Podemos declarar `x` como
  - `int (*x)[20]`
  - Ao invés de: `int x[10][20]`
- Na primeira declaração `x` é definido como um ponteiro para (elementos que são) matrizes inteiras unidimensionais de tamanho 20 contíguos.

# Ponteiros e matrizes multidimensionais

---



**Este elemento pode ser acessado por  $*(* (x+1) + 5)$**

# Ponteiros e matrizes multidimensionais

---

Como podemos acessar, por exemplo, o item na linha 1, coluna 5?

`x[1][5]`

ou

`*(* (x+1) + 5)`

Primeiro,  $(x+1)$  é um ponteiro para a linha 1. Portanto, o objeto desse ponteiro,  $* (x+1)$ , refere-se à linha inteira. Como a linha 1 é uma matriz unidimensional,  $* (x+1)$  é, na realidade, um ponteiro para o primeiro elemento da linha 1. Adicionamos 5 a esse ponteiro. Por isso,  $(* (x+1) + 5)$  é um ponteiro para o elemento 5 (na realidade o sexto elemento) da linha 1. O objeto desse ponteiro é  $* (* (x+1) + 5)$ .

# Ponteiros e matrizes multidimensionais

---

- Adição de duas tabelas (matrizes bidimensionais) de números usando um ponteiro para um conjunto de matrizes unidimensionais
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

# Vetores de Ponteiros

---

- Uma matriz multidimensional pode ser expressa como uma **matriz de ponteiros** em vez de **um ponteiro para um grupo de matrizes contíguas**.

- **Notação**

- `tipo *matriz[expr1];`

- Ao invés da definição anterior

- `tipo (*matriz)[expr2];`

- E também ao invés da definição convencional

- `tipo matriz[expr1][expr2];`




Diagram illustrating the difference between two pointer matrix definitions. Two arrows point from the code snippets above to a box labeled "notar a diferença".

notar a  
diferença

# Vetores de Ponteiros

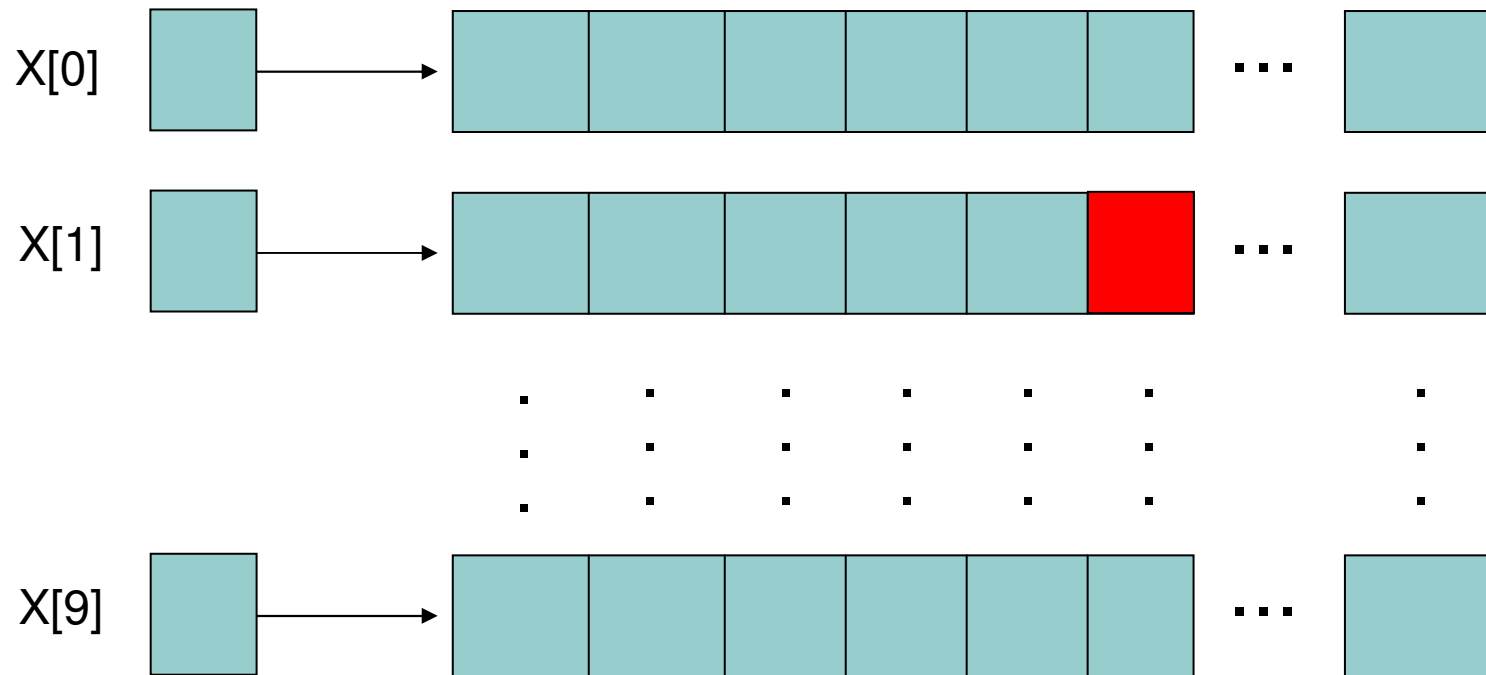
---

- Notação
  - `tipo *matriz[expr1];`
- Observe que o nome da matriz e o asterisco *não estão entre parêntesis*.
- Assim, a regra **EDP**\* associa os pares de colchetes com `matriz`, definindo o **objeto como uma matriz**. O asterisco estabelece que a **matriz conterá ponteiros**

\* Esquerda para direita primeiro

# Vetores de Ponteiros

---



**Esse elemento pode ser acessado por  $*(x[1] + 5)$**

# Vetores de Ponteiros

---

- **Nova versão** da adição de duas tabelas (matrizes bidimensionais) de números usando matriz de ponteiros para matrizes unidimensionais
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))



# Vetores de Ponteiros

---

- Reordenando uma lista de strings. As strings serão armazenadas como uma matriz de ponteiros, onde cada ponteiro indica o início de uma string
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

## Matriz bidimensional ponteiro de ponteiro

---

- Lendo e imprimindo uma matriz bidimensional com índices em tempo de execução.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

---

# **Passando Funções para Outras Funções**

# Passando Funções para Outras Funções

---

- É possível passar, como argumento, um ponteiro para uma função.
- Um argumento que é um ponteiro para uma função pode ser declarado como
  - `tipo (*nome-funcao) ()`.
- Quando for chamar a função (passada como parâmetro) tanto o “\*” (operador indireto) como o nome da função devem estar entre parêntesis.
  - `(*pf) (arg1, arg2, ..., argn)`.

## Passando Funções para Outras Funções

---

- Exemplo de passagem de função para outras funções.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

## Passando Funções para Outras Funções

---

- Outro exemplo de passagem de função para outras funções. Dessa vez foram incluídos os tipos dos argumento nas declarações de funções.
- Exemplo ([código fonte](#))
- Exemplo ([execução](#))

---

# **Mais Sobre Declaração de Ponteiros**

## Mais Sobre Declaração de Ponteiros

---

- Declarações de ponteiros podem ser complicadas e é necessário algum cuidado na sua interpretação, principalmente nas que envolvem funções e matrizes.
- Uma das dificuldades é o uso de parêntesis que tem duas utilidades
  - Indicar funções
  - Indicar precedência



# Mais Sobre Declaração de Ponteiros

---

- O que significa `int *p(int a)`?
  - Uma função que aceita um argumento `int` e retorna um ponteiro para `int`.
- O que significa `int (*p)(int a)`?
  - Um ponteiro para uma função que aceita um argumento inteiro e retorna um inteiro
  - Primeiro par de parêntesis = precedência (ou aninhamento)
  - Segundo par de parêntesis = função

## Mais Sobre Declaração de Ponteiros

---

- O que significa `int *(*p)(int(*a)[ ])` ?
  - Um ponteiro para uma função que aceita um ponteiro para uma matriz de inteiros como argumento e retorna um ponteiro para um int.
- O que significa `int (*p)(char(*a)[ ])` ?
  - `p` é um ponteiro para uma função que aceita um ponteiro para uma matriz de caracteres como argumento e retorna um valor inteiro.

# Mais Sobre Declaração de Ponteiros

---

- O que significa `int *(*p)(char *a[])`?
  - `p` é um ponteiro para uma função que aceita uma matriz de ponteiros para caracteres como argumento e retorna um ponteiro para inteiros.
- O que significa `int *(*p[7])(char a)`?
  - `p` é uma matriz de ponteiros (com 7 elementos) para funções; cada função aceita um caracter como argumento e retorna um ponteiro para um inteiro.

# Mais Sobre Declaração de Ponteiros

---

- O que significa `int (*p[7])(char a)`?
  - `p` é uma matriz de ponteiros (com 7 elementos) para funções; cada função aceita um caractere como argumento e retorna um valor inteiro.
- O que significa `int *(*p[7])(char *a)`?
  - `p` é uma matriz de ponteiros (com 7 elementos) para funções; cada função aceita um ponteiro para caractere como argumento e retorna um ponteiro para um inteiro.