

Ministério da Educação  
Universidade Federal de Santa Maria  
Curso Técnico em Informática para Internet Integrado ao Ensino Médio  
Colégio Técnico Industrial de Santa Maria - CTISM



# Algoritmos e Programação

luciana.lourega@ufsm.br





# Ponteiros

- Ponteiro em C é uma variável que, ao invés de armazenar um dado de um determinado tipo, armazena o **endereço** de um dado de um determinado tipo:
  - Ponteiros são usados frequentemente para:
    - Acesso a E/S mapeada em memória
    - Uso de alocação dinâmica de memória.
    - Alternativa para passagem de parâmetros por referência (em C++)



# Ponteiros

- Declaração de ponteiros em C:
  - Um ponteiro é uma variável numérica e, como todas as variáveis, deve ser declarado antes de ser usado.
  - Os ponteiros são declarados da seguinte maneira:  
*tipo \*nome;*



# Ponteiros

- **Declaração de ponteiros em C**

- Onde *nome* é o identificador do ponteiro e *tipo* é o tipo de dado para o qual ele pode apontar.

Ex:

```
int *d;  
float *ptr2;
```

# Ponteiros

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a person, and a group of people. The hand is positioned on the right side of the frame, with the index finger pointing towards the center.

- Os ponteiros podem ser declarados juntamente com variáveis normais. Aqui estão alguns exemplos:
- `char *ch1,*ch2;`
  - `/*ch1 e ch2 são ponteiros para variáveis do tipo char*/`
- `float *valor,porcentagem;`
  - `/*valor é um ponteiro para um tipo float e porcentagem é uma variável normal do tipo float*/`
  - O símbolo (\*) é usado tanto como operador de indireção como operador de multiplicação.



# Ponteiros

- O OPERADOR INDIRETO (\*)
  - C oferece dois operadores para trabalharem como ponteiros.
    - Um é o operador de endereço (&) que retorna o endereço de memória da variável operando.
    - O segundo é o operador indireto (\*) que é complemento de (&) e retorna o conteúdo da variável localizada no endereço (ponteiro), isto é, devolve o conteúdo da variável apontada pelo operando.

# Ponteiros



```
int x,a;  
int *ptr;  
x = 30;  
ptr = &x;      /* ptr <- endereço de x */  
.  
.  
.  
a = *ptr; /* a recebe o conteúdo do endereço apontado*/
```





# Ponteiros

- Um modo didático para o entendimento de ponteiros é “ler” o significado de \* e & como “conteúdo do endereço apontado por” e “endereço de”, respectivamente. Por exemplo no seguinte código:

```
int *ptr;  
int x;  
x = 10;  
*ptr = 3; /* O CONTEÚDO DO ENDEREÇO APONTADO POR ptr recebe 3 */  
ptr = &x; /* ptr recebe o ENDEREÇO DE x */
```



A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Several hexagons contain white icons: a shopping cart, a person, a group of people, and a person with a checkmark. The word "Ponteiros" is written in large, blue, sans-serif font across the top of the image.

# Ponteiros

- Ponteiros devem apontar para dados do mesmo tipo de sua declaração, do contrário podem ocorrer interpretações erradas na operação.
- Por exemplo, o seguinte código não armazena o valor 56 na variável f.

# Ponteiros



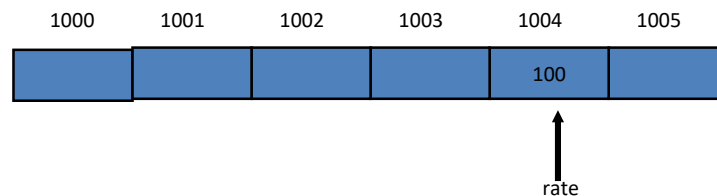
```
int x = 56;  
float *ptr;  
float f;  
ptr = &x; /* Ponteiro para float aponta para int */  
.  
.  
f = *ptr; /* ERRO! Valor de F não é 56 */
```

- o ponteiro para *float* tentará ler o tamanho de um dado *float* a partir do endereço de memória da variável *x* e não o tamanho de um *int*, que é o tipo declarado da variável *x*.

# Ponteiros

- **Criando um ponteiro:**

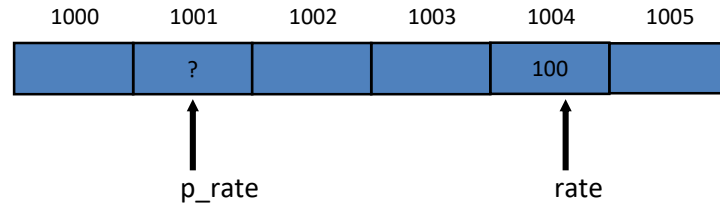
- Se você souber o endereço de uma variável, poderá criar uma segunda variável para armazenar o endereço da primeira.
- O primeiro passo é declarar uma variável que será usada para conter o endereço de rate.



- Uma variável do programa é armazenada em um endereço específico da memória

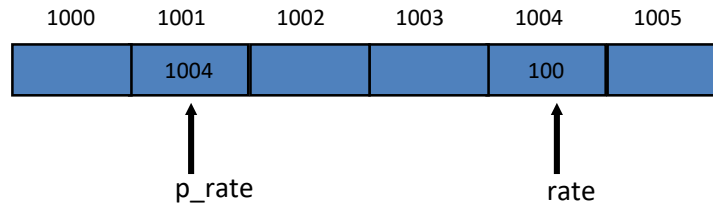
# Ponteiros

- Atribua a essa variável o nome de `p_rate`. A princípio, `p_rate` ainda não está inicializada.
- Foi reservado um espaço de armazenagem para `p_rate`, mas seu valor ainda é indeterminado.



# Ponteiros

- O próximo passo é armazenar o endereço da variável `rate` na variável `p_rate`.



- A variável `p_rate` contém o endereço da variável `rate` e é, portanto, um ponteiro para `rate`.



# Ponteiros

- Inicializando Ponteiros:
  - Um ponteiro só é útil depois que passa a conter o endereço de uma variável.
  - Seu programa deve colocá-lo lá através de um operador de endereço (&).
  - Portanto um ponteiro deve ser inicializado através de uma instrução que tenha o seguinte formato.
    - `ponteiro =&variável;`



# Ponteiros

- A instrução que inicializaria a variável `p_rate` para que ela passasse a apontar para a variável `rate` seria:
  - `p_rate = &rate;`
- Antes da inicialização `p_rate` não apontava para coisa alguma, depois de ser inicializada, passou a ser um ponteiro para `rate`.



# Ponteiros



## Aritmética de ponteiros

- Valores numéricos inteiros podem ser adicionados ou subtraídos de um ponteiro. O resultado é um endereço que segue as regras da aritmética de ponteiro, ou seja:
- Para um ponteiro declarado da seguinte maneira:

*tipo \*ptr;*

e inicializado com um endereço *end1*:

*ptr = end1;*

# Ponteiros

- a operação  $ptr + N$ , onde  $N$  é um número inteiro, resulta um endereço que é igual a  $end1$  mais  $N$  **vezes o tamanho do tipo de dado apontado** (ou seja, o tamanho em bytes de *tipo*).

Por exemplo, considerando que a variável  $x$  foi alocada no endereço 120:

```
int x, y;  
int *ptr;  
ptr = &x;      /* ptr recebe o endereço 120 */  
y = *(ptr + 4); /* y recebe o conteúdo do endereço 120 + 4*(tamanho do int) ==  
endereço 128 */
```

# Ponteiros

Outro exemplo:

```
float *ptr;
```

```
ptr = (float*)100; /* ponteiro é 'forçado' para o end. 100 */
```

```
.
```

```
.
```

```
.
```

```
*(ptr + 3) = 15; /* Número 15 é armazenado no endereço  $100 + 3 \times 4 = 112$  */
```

# Ponteiros



- **Tipos inteiros** (tamanho dependente da máquina)
- char – 1 byte no intervalo [0,128)
- signed char – 1 byte no intervalo (-128,128)
- unsigned char – 1 byte no intervalo (0,256)
- short – 2 bytes no intervalo (-2-15, 215)
- unsigned short – 2 bytes no intervalo[0, 2 16)
- int – 2 ou 4 bytes no intervalo (2-15, 215) ou (2-31, 231) respectivamente
- unsigned int – 2 ou 4 bytes no intervalo [0, 216) ou [0, 232) respectivamente
- long – 4 bytes no intervalo (2-31, 231)
- unsigned long - 4 bytes no intervalo [0, 232)

# Ponteiros

- **Tipos flutuantes**

- float – pelo menos 6 dígitos de precisão decimal – 4 bytes
- double – pelo menos 10 dígitos decimais de precisão – 8 bytes
- long double – pelo menos 10 dígitos decimais precisão – 12 bytes.

```
# include <stdio.h>
# include <conio.h>
main()
{
long double c;
printf("%d",sizeof(c));
getch();
}
```

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a person, and a group of people. The word "Ponteiros" is written in large, blue, sans-serif font across the middle of the image.

# Ponteiros

## Relação ponteiro-vetor


- Quando um vetor é declarado, o identificador deste vetor marca o endereço de início da área de memória alocada por ele.
- Assim, o nome do vetor pode ser usado como referência de endereço com os mesmos operadores utilizados para ponteiros.



# Ponteiros

- Considerando a declaração do `int a[10];`
- Sendo `pa` um ponteiro para inteiro então:
  - `pa = &a[0]` //passa o endereço inicial do vetor para o ponteiro `pa`.
  - `pa=a;` // é a mesma coisa de `pa=&a[0];`
  - `x=*pa;` //passa o conteúdo de `a[0]` para `x`.





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

    main()
    {
        int
vet[5]={1,2,3,4,5};
        int *p = vet;
        int i;
        for(i=0;i<5;i++)
            printf("%d\n",p[i]);
        return 0;
    }
```

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

    main()
    {
        int
vet[5]={1,2,3,4,5};
        int *p = vet;
        int i;
        for(i=0;i<5;i++)

printf("%d\n",*(p+i));
        return 0;
    }
```

# Ponteiros

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a person, a group of people, and a person with a checkmark. The hand is positioned on the right side of the frame, with the index finger pointing towards the group of people icon.

- Se **pa** aponta para um elemento particular de um vetor **a**, então por definição **pa+1** aponta para o próximo elemento, e em geral **pa-i** aponta para **i** elementos antes de **pa** e **pa+i** para **i** elementos depois.
  - Se **pa** aponta para **a[0]** então:
  - **\*(pa+1)** aponta para **a[1]**
  - **pa+i** é o endereço de **a[i]** e **\*(pa+i)** é o conteúdo.

# Ponteiros



- Os operadores ++ e -- possuem precedência sobre o \* e operadores matemáticos.
  - `*px++` //(sobe na memória)
  - `(*px)++` //(soma 1 no conteúdo)
  - `*(px--)` //(o mesmo que `*px--`)

# Ponteiros

- Para varrer todos os elementos de uma matriz de uma forma sequencial é bem mais simples utilizar ponteiros. Considere o seguinte programa para zerar uma matriz.

```
#include <stdio.h>
main ()
{
    float matrix [50][50];
    int i,j;
    for (i=0;i<50;i++)
        for (j=0;j<50;j++)
            matrix[i][j]=0.0;
}
```

- Uma maneira muito mais eficiente seria:



# Ponteiros

- Você consegue ver porque o segundo programa é mais eficiente?
  - Simplesmente porque cada vez que se faz `matrix[i][j]` o programa tem que calcular o deslocamento. Ou seja, o programa tem que calcular 2500 deslocamentos.
  - No segundo programa o único cálculo que deve ser feito é o de um incremento de ponteiro. Fazer 2500 incrementos em um ponteiro é muito mais rápido que calcular 2500 deslocamentos completos.
  - Exemplo em sala de aula.

A hand is shown pointing at a digital interface. The interface features a blue background with a pattern of hexagons. Some hexagons contain white icons: a shopping cart, a person, and a group of people. The word "Ponteiros" is written in large, blue, sans-serif font across the middle of the image.

# Ponteiros

- Ponteiros para ponteiros:
  - Um ponteiro para um ponteiro é como se você anotasse o lugar (endereço) de uma agenda que tem o endereço da casa do seu amigo. A sintaxe para declarar um ponteiro para um ponteiro é:
  - *tipo\_da\_variável* \*\*nome\_da\_variável;



# Ponteiros

- Se no código do programa utiliza-se a expressão: `**nome_da_variável`, está-se referenciando ao conteúdo final da variável apontada; enquanto, `*nome_da_variável` é o conteúdo do ponteiro intermediário.
- Em consequência, no C pode-se declarar ponteiros para ponteiros para ponteiros, e assim por diante. Para fazer isto, basta aumentar o número de asteriscos na declaração. A lógica é a mesma, mas a utilidade?
- Exemplo em sala de aula.



# Ponteiros



- É a quantidade de asteriscos (\*) na declaração do ponteiro que indica o número de níveis de apontamento que ele possui.

//variável inteira

int x;

//ponteiro para um ponteiro (1 nível)

int \*p1;

//ponteiro para ponteiro de inteiro (2 níveis)

int \*\*p2

//ponteiro para ponteiro para ponteiro de inteiro (3 níveis)

int \*\*\*p3;

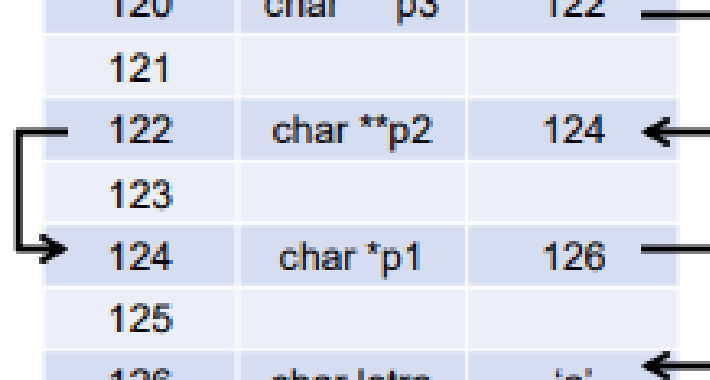
# Ponteiros

- Conceito de "ponteiro para ponteiro":

```
char letra = 'a';  
char *p1;  
char **p2;  
char ***p3;
```

```
p1 = &letra;  
p2 = &p1;  
p3 = &p2;
```

Memória		
posição	variável	conteúdo
119		
120	char ***p3	122
121		
122	char **p2	124
123		
124	char *p1	126
125		
126	char letra	'a'
127		



The diagram illustrates the memory layout and pointer relationships. Arrows indicate the following pointers: p3 (at address 120) points to p2 (at address 122); p2 (at address 122) points to p1 (at address 124); and p1 (at address 124) points to letra (at address 126). The variable letra at address 126 contains the character 'a'.