Engenharia da Computação

www.eComp.Poli.br

Manipulação de Endereços
(Ponteiros)

Disciplina: **DCExt Programação Imperativa**

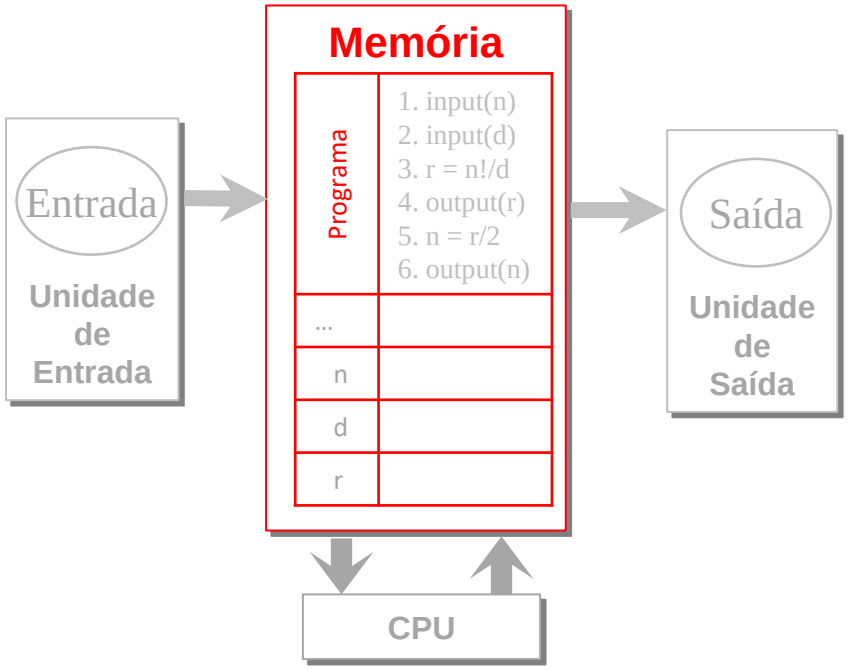
Prof. Hemir Santiago

hcs2@poli.br

Material cedido pelo Prof. Joabe Jesus

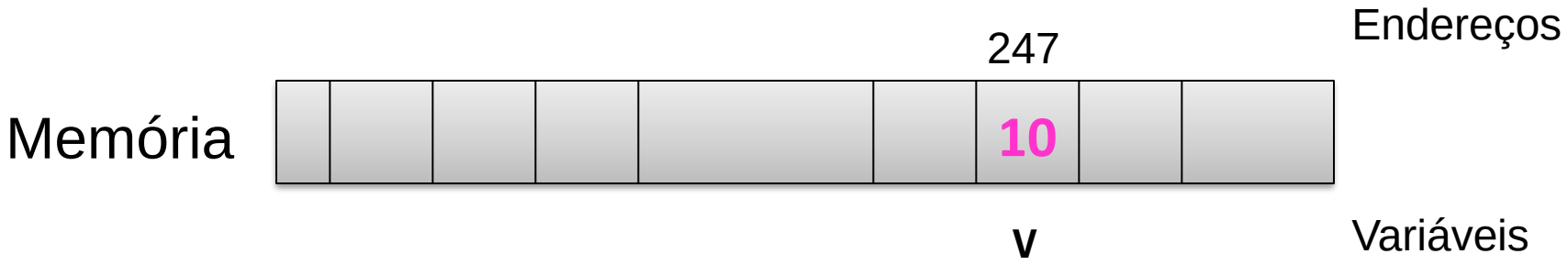
Ponteiros e Endereços

PONTEIROS



Ponteiros

- **Suponha** o seguinte **cenário**:



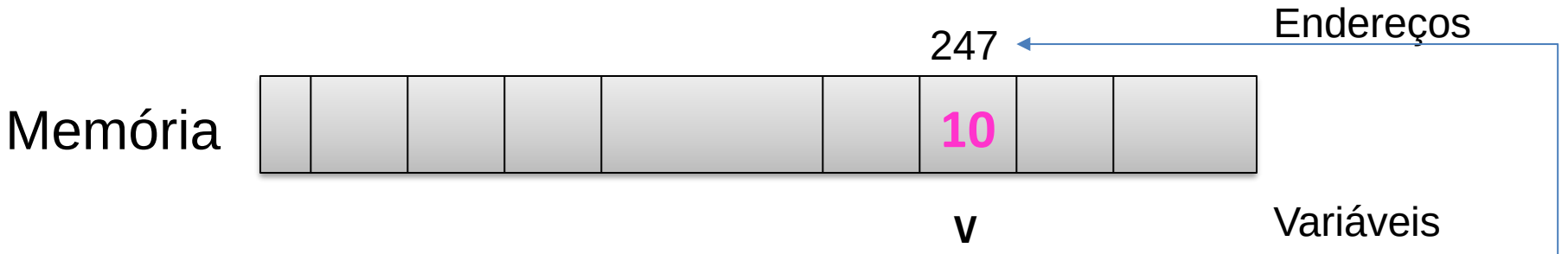
Programa

```
int V;       // aloca/reserva espaço para V

V = 10;      // armazena valor 10 no espaço V
... &V ...;  // uso do endereço da variável V
```

Ponteiros

- **Suponha** o seguinte **cenário**:



Programa

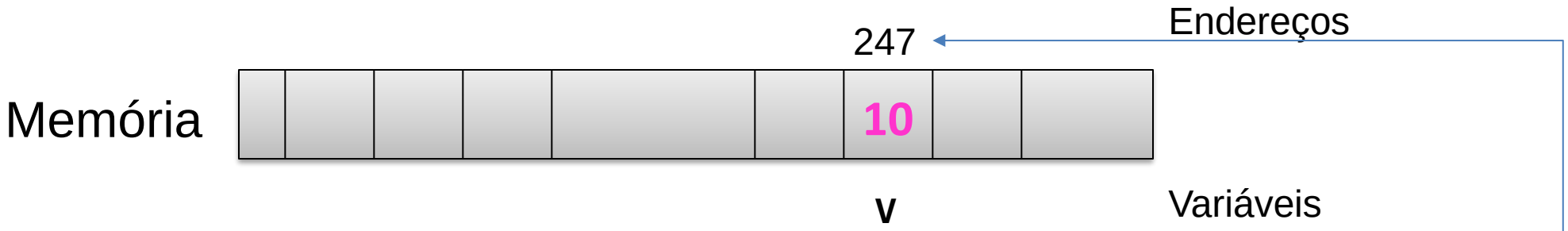
```
int V;              // aloca/reserva espaço para V

V = 10;             // armazena valor 10 no espaço V
... &V ...;         // uso do endereço da variável V
```

No caso, o **endereço** é **247**

Ponteiros

- **Suponha** o seguinte **cenário**:



Programa

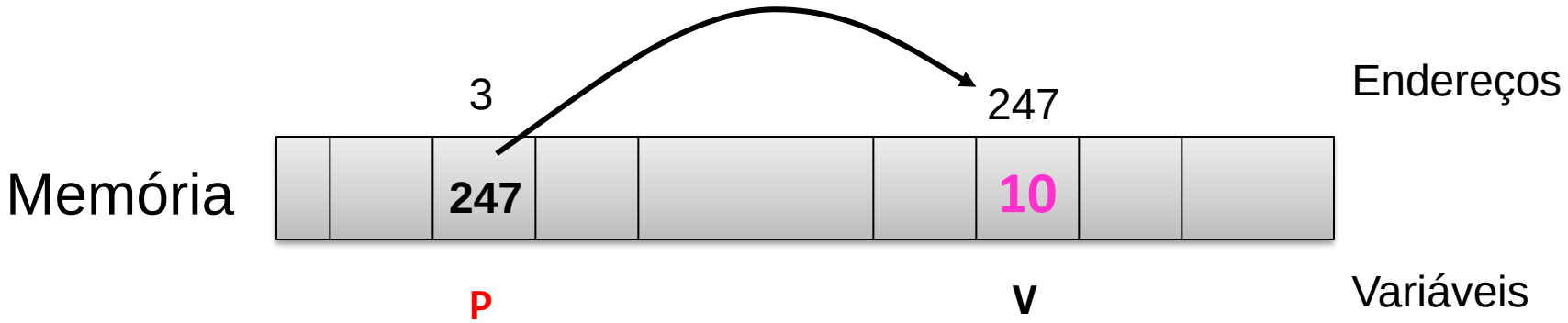
```
int V;              // aloca/reserva espaço para V

V = 10;             // armazena valor 10 no espaço V
scanf("%d", &V); // uso do endereço da variável V
```

scanf pode alterar valor de V, isto é, do **endereço 247**

Ponteiros

- **Suponha** o seguinte **cenário**:

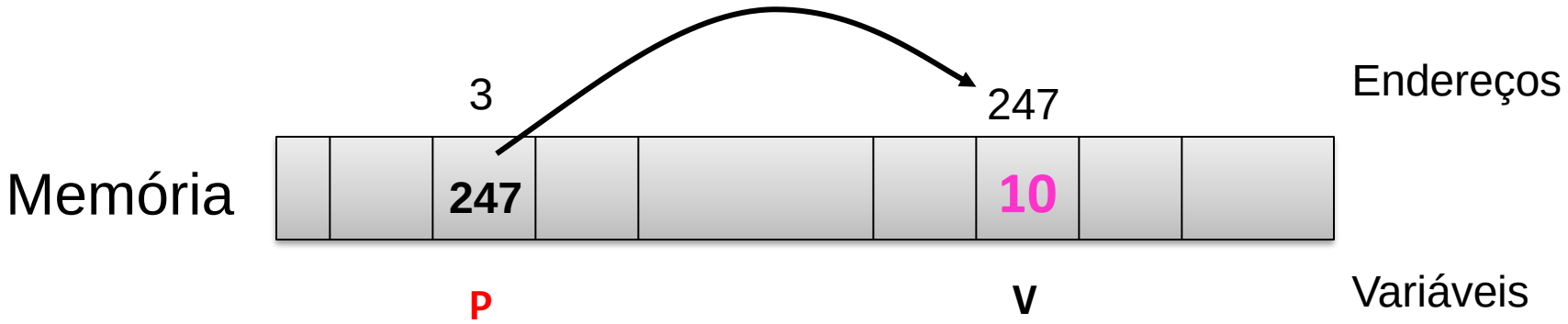


Programa

```
int V;          // alocando/reservando espaço V
???? P;         // Qual o TIPO da variável P?
V = 10;         // armazenando valor 10 em V
P = &V;         // P recebe o endereço da variável V
```

Ponteiros

- **Suponha** o seguinte **cenário**:

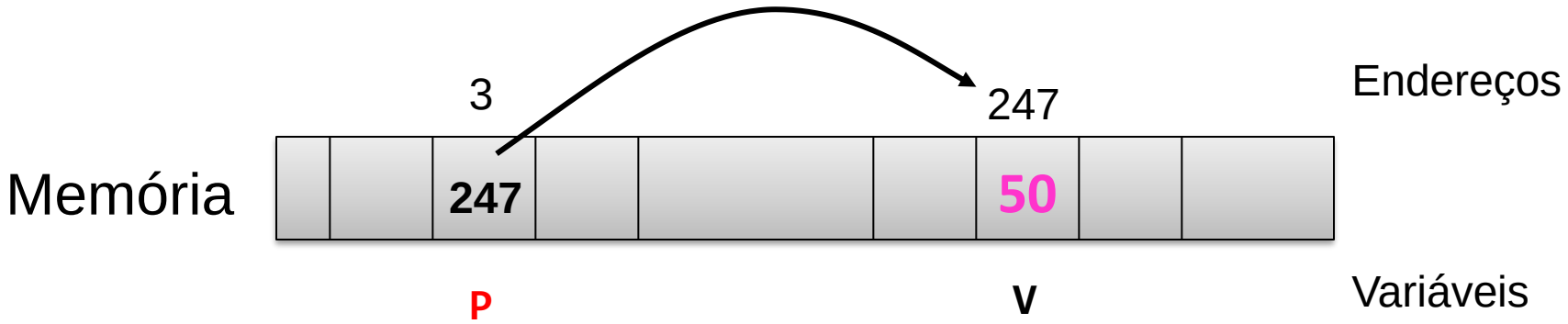


Programa

```
int V;          // alocando/reservando espaço V
int* P;         // Tipo PONTEIRO para variável int
V = 10;         // armazenando valor 10 em V
P = &V;         // P recebe o endereço da variável V
```

Tipo Ponteiro/Endereço

- **MANIPULAM** (consultam ou modificar) **ENDEREÇO** indicado



Programa

```
int V;          // alocando/reservando espaço V
int* P;         // Tipo PONTEIRO para variável int
V = 10;         // armazenando valor 10 em V
P = &V;         // P recebe o endereço da variável V
*P = 50;        // P altera o valor de V para 50
```

Ponteiros – MOTIVAÇÃO

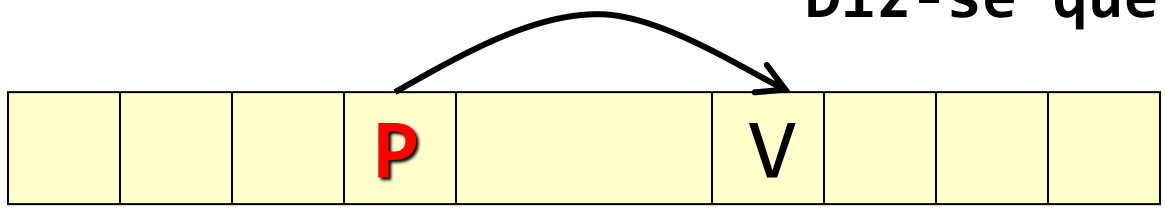
- Possibilitar que funções modifiquem os argumentos que recebem, assim como `scanf` faz para a entrada de dados
- Útil para passar vetores como parâmetro e manipular vetores e strings
- Criar estruturas de dados mais complexas
  - Como listas encadeadas, árvores binárias, grafos etc.
- **Código mais eficiente**

Operador de **Endereço** &

- Fornece o endereço de uma variável ou vetor:

```
P = &V    // P recebe o endereço da variável V
```

Diz-se que P aponta para V



Operador de **Endereço &**

- **CUIDADO**:
  - **NÃO PODE** ser aplicado a expressões ou constantes (literais)
    - **Exemplo:**

```
x = &3;  // Esse código é ERRADO, 3 é uma constante.
```

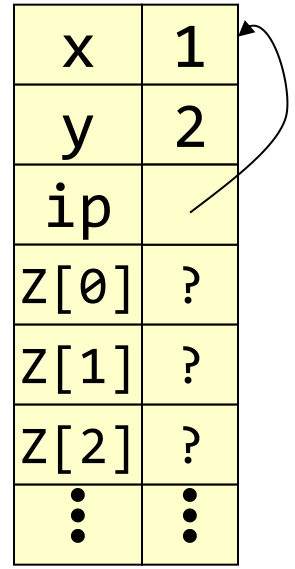
  - **MAS PODE** ser aplicado a uma string (vetor de char)
    - **Exemplo:**

```
x = &"";  // Esse código é CORRETO, "" é um vetor
```

Operador de **Indireção** *

- **Manipula/Acessa** o endereço apontado por um ponteiro

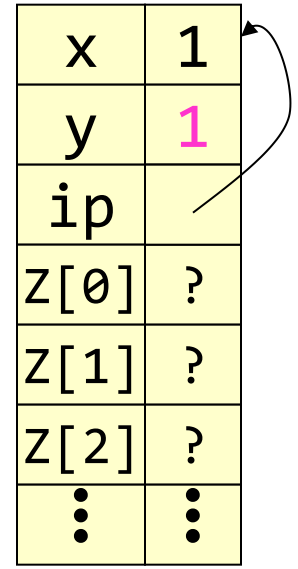
```
int x=1, y=2, z[3];
int *ip;        // ip é um ponteiro para int
ip = &x;        // ip aponta para o endereço de x
y = *ip;        // y = valor apontado por ip (ou seja, 1)
*ip = 0;
ip = &z[0];
*ip = 10;
```



Operador de **Indireção** *

- **Manipula/Acessa** o endereço apontado por um ponteiro

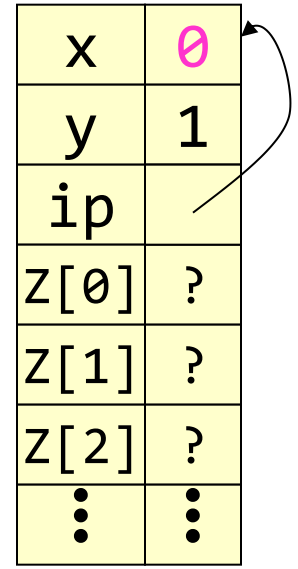
```
int x=1, y=2, z[3];
int *ip;        // ip é um ponteiro para int
ip = &x;        // ip aponta para o endereço de x
y = *ip;        // y = valor apontado por ip (ou seja, 1)
*ip = 0;
ip = &z[0];
*ip = 10;
```



Operador de **Indireção** *

- **Manipula/Acessa** o endereço apontado por um ponteiro

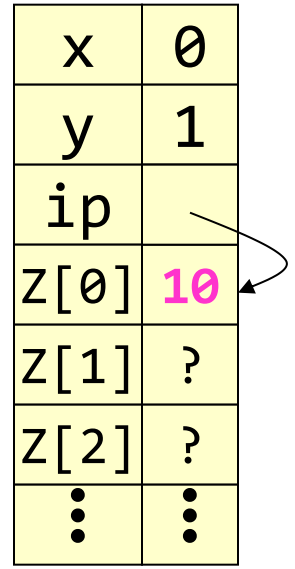
```
int x=1, y=2, z[3];
int *ip;        // ip é um ponteiro para int
ip = &x;        // ip aponta para o endereço de x
y = *ip;        // y = valor apontado por ip (ou seja, 1)
*ip = 0;
ip = &z[0];
*ip = 10;
```



Operador de **Indireção** *

- **Manipula/Acessa** o endereço apontado por um ponteiro

```
int x=1, y=2, z[3];
int *ip;        // ip é um ponteiro para int
ip = &x;        // ip aponta para o endereço de x
y = *ip;        // y = valor apontado por ip (ou seja, 1)
*ip = 0;
ip = &z[0];
*ip = 10;
```



Situação Problema 1

- **Suponha** o código abaixo:

```
x = 5;
y = x;
x = x + 1;
```

  - Como atualizar o valor de Y sempre que X for alterado para que *X = Y sempre* ?
    - **Solução 1**: Qualquer alteração em X deve ser seguida pela linha Y = X
    - **Solução 2**: Usar Y como ponteiro

Solução 1 – CÓPIA para y

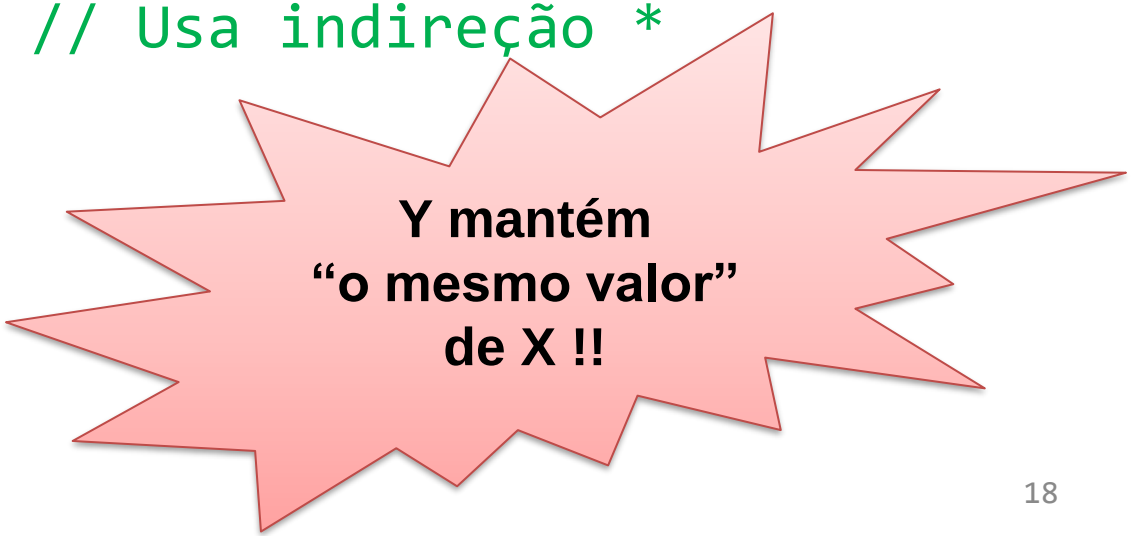
```
int x, y;
x = 5;
y = x;  // Modificou x acima, então faz CÓPIA para y
printf("X = %d, Y= %d\n", x, y);
x = x + 1;
y = x;  // Modificou x acima, então faz CÓPIA para y
printf("X = %d, Y= %d\n", x, y);
```

Solução 2 – y é um **Ponteiro**

```
int x, *y = &x; // y aponta para x, independente do valor de x
x = 5;

printf("X = %d, Y= %d\n", x, *y);   // Usa indireção *
x = x + 1;

printf("X = %d, Y= %d\n", x, *y);  // Usa indireção *
```



Situação Problema 2

```
int *p1,*p2, i, j;
i=1;
p1=&i;
p2=p1;
i=3;
j=4;
p1=&j;
printf("*p1=%d, *p2=%d, i=%d, j=%d\n", *p1, *p2, i, j);
printf("p1=%d, p2=%d, &i=%d, &j=%d\n", p1, p2, &i, &j);
```

Qual o resultado? (fazer no papel, sem usar o computador, usar desenhos).
O valor dos endereços podem ser fictícios.
Ex: posição x, y, z, w da memória.

Endereços como Argumentos para uma Função

- Como uma função pode alterar variáveis de quem a chamou?
  1. função ‘chamadora’ passa os endereços dos valores que devem ser modificados
  2. função chamada deve declarar os endereços recebidos como ponteiros

Endereços como Argumentos para uma Função

Chamada por valor

```
main( )
{
      int x, y;
      x=0;
      y=0;
      altera2(x,y);
      printf("1º é %d, 2º é %d.",x, y);
}

void altera2(int px, int py)
{
       px = 3;
       py = 5;
}
```

NÃO altera os valores de x e y

Chamada por referência

```
main( )
{
  int x, y;
  x=0;
  y=0;
  altera2(&x,&y);
  printf("1º é %d, 2º é %d.",x,y);
}

void altera2(int *px, int *py)
{
    *px = 3;
    *py = 5;
}
```

Altera os valores de x e y

Endereços como Argumentos para uma Função

```
main( )
{
      int x, y;
      x=0;
      y=0;
      altera2(&x,&y);
      printf("O 1o. é %d, o 2o. é %d.", x, y);
}

void altera2(int *px, int *py)
{
       *px = 3;
       *py = 5;
}
```

***px e *py são do tipo int**

px e py contém endereços de variáveis do tipo int

Aritmética de Ponteiros

- O incremento de um ponteiro acarreta na movimentação do mesmo para o próximo valor do ***tipo apontado***
  - Ex: Se ***px*** é um ponteiro para int com valor 3000, depois de executada a instrução ***px++***, o valor de ***px*** será 3004 e não 3001 !!!
    - Obviamente, o deslocamento varia de compilador para compilador dependendo do número de bytes adotado para o referido tipo

short	2 bytes
int	**4 bytes**
long	4 bytes
long long	8 bytes

Aritmética de Ponteiros

```
main( )
{
        int x=5, y=6;
        int *px, *py;

        px = &x;
        py = &y;

        if (px<py) printf("py-px = %u\n",py-px);
        else       printf("px-py = %u\n",px-py);

        printf("px = %u, *px = %d, &px = %u\n", px, *px, &px);
        printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
        py++;
        printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
        py=px+3;
        printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
        printf("py-px = %u\n",py-px);
}
```

Resultado: 1
4 bytes de diferença

Aritmética de Ponteiros

	`px - py = 1`
	`px = 65488, *px= 5, &px = 65460`
	`py = 65484, *py= 6, &py = 65464`
py++	`py = 65488, *py= 5, &py = 65464`
py=px+3	`py = 65500, *py= ?, &py = 65464`
	`py - px = 3`

Testes relacionais

>=, <=, <, >, ==

são **aceitos** em ponteiros

A diferença entre dois ponteiros será dada na unidade do **tipo de dado** apontado

Ponteiros, Vetores e Matrizes

- Em C existe um relacionamento muito forte entre ponteiros e vetores
  - O compilador transforma qualquer vetor e matriz em ponteiros, pois a maioria dos computadores é capaz de manipular ponteiros e não vetores
  - Qualquer operação que possa ser feita com índices de um vetor pode ser feita com ponteiros
  - **O nome de um vetor é um endereço, ou seja, um ponteiro**

Ponteiros e Vetores

Versão com Vetor

```
main( )
{
    int nums[ ] = {1, 4, 8};
    int cont;

    for(cont=0; cont < 3; cont++)
        printf("%d\n",nums[cont]);
}
```

Versão com Ponteiro

```
main( )
{
    int nums[ ] = {1, 4, 8};
    int cont;

    for(cont=0; cont < 3; cont++)
      printf("%d\n",*(nums + cont));
}
```

Endereço inicial do vetor

Deslocamento

Ponteiros e Vetores

Observe a diferença para.....

```
main( )
    {
      int nums[ ] = {1, 4, 8};
      int cont;

      for(cont=0; cont < 3; cont++)
           printf("%d\n", (nums + cont));
     }
```

2293600	1
2293604	4
2293608	8

Sem o * !!

Ponteiros e Vetores

- Observação sobre o tamanho do vetor:
  - Comando *sizeof*

```
main()
{
  int num[ ]={1,2,3};

  printf ("Tamanho = %d\n", sizeof(num));
  printf ("Numero de elementos = %d\n", sizeof(num)/sizeof(int));
}
```

Observe o resultado !!

Observe o resultado !!

O que representam ?

Ponteiros e Vetores

- Escreva um programa que lê um conjunto de, no máximo, 40 notas, armazena-as em um vetor e, por fim, imprime a média das notas.
  - Obs: utilize ponteiros para manipular o vetor
  - O programa para de pedir as notas e realiza o cálculo da média quando o usuário entra com uma nota < 0

Ponteiros e Vetores

```
main( )
{
      float notas[40], soma=0;
      int cont=0;
      do {
            printf("Digite a nota do aluno %d: ", cont);
            scanf("%f", notas+cont);
            if(*(notas+cont) > -1)
                soma += *(notas+cont);
      } while(*(notas+cont++) >= 0);

      printf("Média das notas: %.2f", soma/(cont-1));
}
```

Ponteiros e Vetores

- Será que existe alguma maneira de simplificar a expressão `while(*(notas+cont++)>0 )` ?

while (*(notas++) > 0)

Errado !! “notas” é um ponteiro constante ! É o endereço do vetor notas e não pode ser trocado durante a execução do programa !

Apenas um ponteiro variável pode ser alterado

Ponteiros e Vetores

- Será que existe alguma maneira de simplificar a expressão `while(*(notas+cont++)>0 )` ?

p = &notas;
while (*(p++) > 0)

Ponteiros e Vetores

- Vamos re-escrever o programa anterior com um **ponteiro variável**....

```
main( )
    {
      float notas[40], soma=0.0;
      int cont=0; float *ptr;
      ptr = notas;
      do {
            printf("Digite a nota do aluno %d: ", cont++);
            scanf("%f",ptr);
            if(*ptr > -1)
                soma += *ptr;
      } while(*(ptr++) >= 0 && cont <= 40);
      printf("Média das notas: %.2f", soma/(cont-1));
   }
```

Vetor como Parâmetro (SEM ponteiro)

Uso do [ ], sem ponteiros

```
#define TAM 5
adcons(int ptr[], int num, int cons)
{
    int k;
    for(k=0; k<num; k++) {
        ptr[k] = ptr[k] + cons;
    }
}
main( )
{
     int vetor[TAM]={3,5,7,9,11};
     int c=10;
     int j;
     adcons(vetor, TAM, c);
     for(j=0; j<TAM; j++)
      printf("%d ",(vetor[j]));
}
```

Vetor como Parâmetro (COM ponteiro)

Somando uma constante aos elementos de um vetor

```
#define TAM 5
adcons(int *ptr, int num, int cons)
{
    int k;
    for(k=0; k<num; k++) {
        *ptr = *ptr + cons;
        ptr++;
    }
}
main( )
{
     int vetor[TAM]={3,5,7,9,11};
     int c=10;
     int j;
     adcons(vetor, TAM, c);
     for(j=0; j<TAM; j++)
      printf("%d ",*(vetor+j));
}
```

Exercício 1

- Escreva um programa que aplica a função exponencial_2 a uma variável inteira e imprime o resultado da aplicação.
- A função exponencial_2 deve ser do tipo void e eleva um número ao quadrado
- O resultado deve ser armazenado na própria variável inteira passada como parâmetro para exponencial_2

Exercício 2

- Escreva um programa que lê um conjunto de, no máximo, 10 números inteiros e os armazena em um vetor. O programa para de pedir os números quando o usuário entra com um valor < 0 OU atingir o tamanho máximo do vetor.
  - Obs: utilize ponteiros para manipular o vetor.
  - **Os números devem ser ordenados de forma crescente em um 2º vetor.**

	DATA	AULA
1	22/08/2024	Apresentação da disciplina | Introdução à Programação Imperativa
2	29/08/2024	Introdução à Linguagem de Programação C
3	05/09/2024	Conceitos Fundamentais
4	12/09/2024	Tipos de Dados Especiais em C
5	19/09/2024	Estruturas Condicionais e de Repetição
6	26/09/2024	Pré-processamento
7	03/10/2024	Registros/Estruturas de Dados
8	10/10/2024	Ponteiros
9	17/10/2024	1º Exercício Escolar

Plano de Aulas

	DATA	AULA
10	24/10/2024	Arquivos
11	31/10/2024	Acompanhamento de projetos
12	07/11/2024	Acompanhamento de projetos
13	14/11/2024	Acompanhamento de projetos
14	21/11/2024	Acompanhamento de projetos
15	28/11/2024	Apresentação parcial
16	05/12/2024	Apresentação de projetos
17	12/12/2024	Avaliação Final

Plano de Aulas