Ponteiros

- Definição: uma variável que contém o endereço de outra variável
- Usados para alocação dinâmica e passagens por referência/valor em funções

Ponteiros para Variáveis

- Sintaxe: `<tipo da variável apontada> *<nome do ponteiro>;`
- Exemplo de declaração

```
int y;       // y é uma variável do tipo inteiro não inicializada
int x = 9;   // x é uma variável do tipo inteiro inicializada com o valor 9
int *ptr_x;  // ptr_x é um ponteiro para uma variável do tipo inteiro
```

Memória RAM (>= *n* bytes)	
Endereço (bytes)	Valor
n-5	? (ptr_x)
n-4	
n-3	9 (x)
n-2	
n-1	? (y)
n	

- Exemplo de uso

```
ptr_x = &x;  // ptr_x aponta para x
y = *ptr_x;  // y recebe o valor apontado por ptr_x, ou seja, 9
```

Memória RAM (>= *n* bytes)	
Endereço (bytes)	Valor
n-5	n-3 (ptr_x)
n-4	
n-3	9 (x)
n-2	
n-1	9 (y)
n	

- Alterações no valor da variável apontada (`x`) influenciam o ponteiro (`ptr_x`)

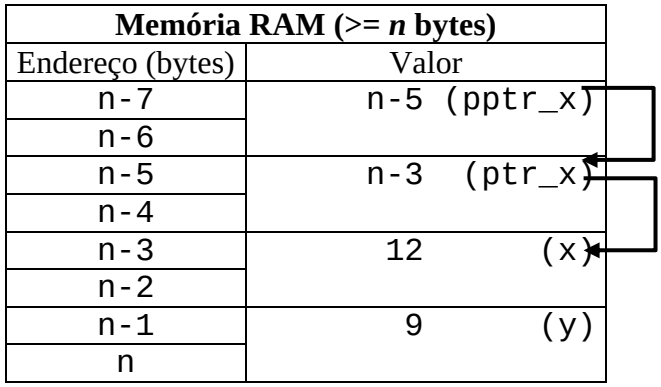
```
x = 12;
printf("%d %d %d", ptr_x, *ptr_x, &ptr_x); // p/ n=64, imprime: 61 12 59
```

Memória RAM (>= *n* bytes)	
Endereço (bytes)	Valor
n-5	n-3 (ptr_x)
n-4	
n-3	12 (x)
n-2	
n-1	9 (y)
n	

Ponteiros para Ponteiros

- Forma de indicação múltipla, onde um ponteiro aponta para o endereço de outro ponteiro.

```
int **pptr_x;
pptr_x = &ptr_x;
printf("%d %d %d", pptr_x, *pptr_x, **pptr_x);    // n=64, imprime: 59 61 12
printf("%d %d %d", &pptr_x, &*pptr_x, &**pptr_x); // n=64, imprime: 57 59 61
```



Problema do Ponteiro Perdido

- A cada operação com um ponteiro, pode estar sendo lido ou gravado dados em posições desconhecidas da memória. Isso pode acarretar em sobreposição de áreas de dados ou mesmo da área do programa.
- No exemplo a seguir, está sendo atribuído o valor 10 a uma posição de memória desconhecida. A conseqüência desta ação é imprevisível.

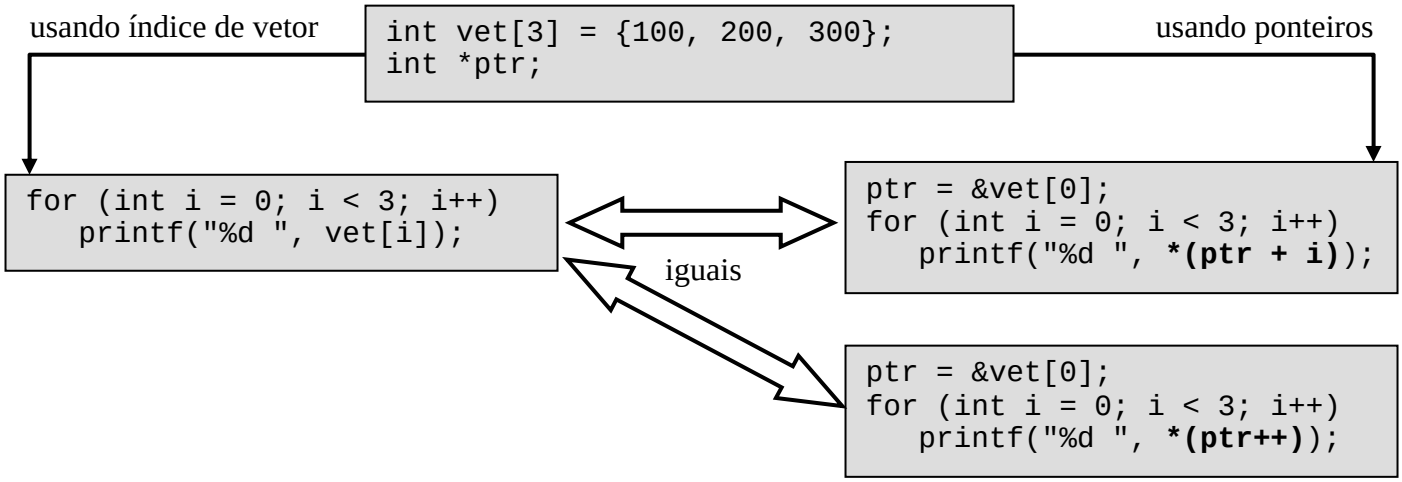
```
int x, *ptr_x;
*ptr_x = 10;  // atribui 10 a uma posição de memória desconhecida
```

- Corrigindo o problema do ponteiro perdido...

```
int x, *ptr_x;
ptr_x = &x;
*ptr_x = 10;  // atribui 10 a variável apontada por ptr_x, ou seja, x = 10;
```

Manipulando Vetores usando Ponteiros

- Matrizes e ponteiros podem ser tratados da mesma maneira.
- Versões usando ponteiros geralmente são mais rápidas.
- Para se ter garantia de que o incremento do ponteiro está sendo feito corretamente, a *aritmética de ponteiros* permite incrementar/decrementar de uma quantidade igual ao tamanho do tipo de dados armazenado no vetor. *Limitações:* (1) funciona apenas para as operações de soma, subtração, incremento e decremento e (2) o valor de incremento/decremento só pode ser inteiro.
- Sendo assim, a adição e subtração de ponteiros alteram seu valores baseados no tamanho do tipo de dados para o qual eles apontam. Exemplos:
  - `int *ptr_i; ptr_i++;` // incrementa 2 bytes ao ponteiro `ptr_i`.
  - `float *ptr_f; ptr_f++;` // incrementa 4 bytes ao ponteiro `ptr_f`.
  - `double *ptr_d; ptr_d--;` // decrementa 8 bytes do ponteiro `ptr_d`.



- As possíveis operações aritméticas com ponteiros são:

Atribuição	Atribuir um valor ao ponteiro, comumente o endereço de uma variável. Exemplo:
	`int *ptr, x;`
	`ptr = &x;`

Indireção	O operador de indireção (*) retorna o valor armazenado no endereço apontado. Exemplo:
	`*ptr = 3;  // atribui 3 a variável x apontada por ptr`
	`printf("%d", *ptr);  // imprime 3 na tela`

Endereço-de	Para encontrar o endereço de um ponteiro e usar ponteiros de ponteiros. Exemplo:
	`int *ptr, **ptr_de_ptr;`
	`ptr_de_ptr = &ptr;`

Incremento	Avançar o ponteiro para a próxima posição de memória no tamanho do tipo de dados para o qual ele aponta. Exemplo: `*ptr++;`
Decremento	Retroceder o ponteiro para a posição anterior de memória no tamanho do tipo de dados para o qual ele aponta. Exemplo: `*ptr--;`
Diferença	Comumente usado para dois ponteiros que apontam para um mesmo vetor. Exemplo: `printf("As duas posições do vetor estão distantes %d unidades", (ptr1 - ptr2));`

Comparação	Comumente usado para dois ponteiros que apontam para um mesmo vetor. Exemplo:
	`if (ptr1 < ptr2)`
	`  anterior = *ptr1;`
	`else`
	`  anterior = *ptr2;`

Exercícios:

a) Para *n* = 32, qual o retorno do uso do operador *endereço-de* sobre o ponteiro `ptr_x`?

Memória RAM (>= *n* bytes)	
Endereço (bytes)	Valor
n-5	n-3 (ptr_x)
n-4	
n-3	9 (x)
n-2	
n-1	9 (y)
n	

Resposta: 27

b) Para *n* = 64, qual o retorno do uso do operador *indireção* sobre o ponteiro de ponteiro `pptr_x`?

Memória RAM (>= *n* bytes)	
Endereço (bytes)	Valor
n-7	n-5 (pptr_x)
n-6	
n-5	n-3 (ptr_x)
n-4	
n-3	12 (x)
n-2	
n-1	9 (y)
n	

Resposta: 61

c) Qual a saída do seguinte programa em linguagem C:

```
#include <stdio.h>

int main() {
    int a = 3;
    int b = 5;
    int *p1, *p2, **pp2;

    p1 = &a;
    p2 = &b;
    pp2 = &p2;
    while (**pp2 < 6) {
       b = *p1 + 1;
       *p1 -= 3;
       printf("%d ", a);
       a = *p2;
       *p2 -= 2;
       printf("%d ", b);
    }
}
```