Na última aula:

- A memória é abstraída como um vetor indexado.
- As variáveis possuem tipo → tamanho na memória (alocação) → endereço de memória
  (índice de memória que começa sua alocação)

Ponteiros são variáveis que armazenam endereços de memória.

Manipulação de ponteiros

① Declaração

tipo *ptr;

(mesmos tipos de regras de nomeação de variáveis.)

- O tipo do ponteiro refere-se ao tipo do dado armazenado no endereço de memória contido no ponteiro.

Ex.: DOUBLE *PTR

↳ PTR conterá um inteiro que representa um end. de memória.

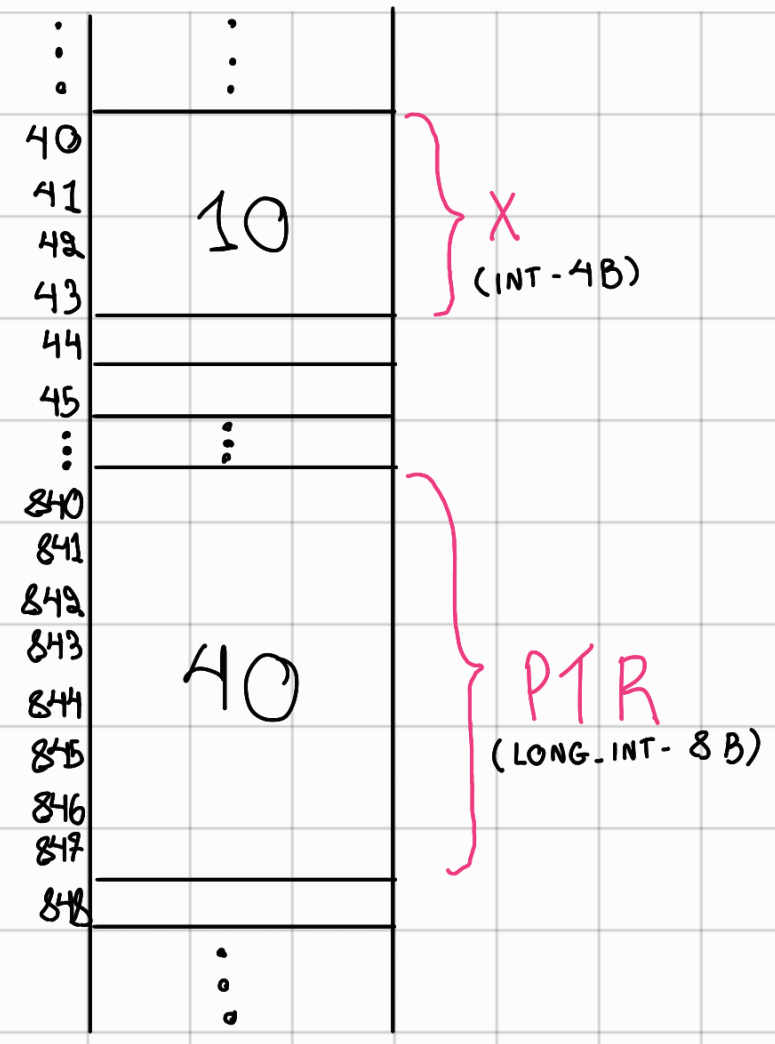
↳ espera-se que no end. de memória contido em PTR haja um dado do tipo DOUBLE.

② Operações

a) & (UNÁRIO): "endereço de"

Ex.: INT X = 10

INT *PTR = &X



Dizemos que PTR:

- Contém o endereço de X.
- Aponta para X.

```
PRINTF("%LD", &X); //SAÍDA: 40
PRINTF("%LD", &PTR); //SAÍDA: 840
```

b) * (UNÁRIO): desreferenciamento

Acessa o dado que está no endereço de memória contido no ponteiro.

Ex.: INT X = 10;

INT *PTR = &X;

*PTR = *PTR + 5;

PRINTF("%D", *PTR); //SAÍDA:

c) Aritmética de ponteiros

Ex.:
```
INT X = 10;
INT *PTR = &X;
PTR = PTR + 5;
PRINTF("%Ld", PTR); //SAÍDA: 40 + 5x4 = 60
PRINTF("%d", *PTR);
```

end. memória + 5 * SIZEOF(tipo de PTR) em PTR

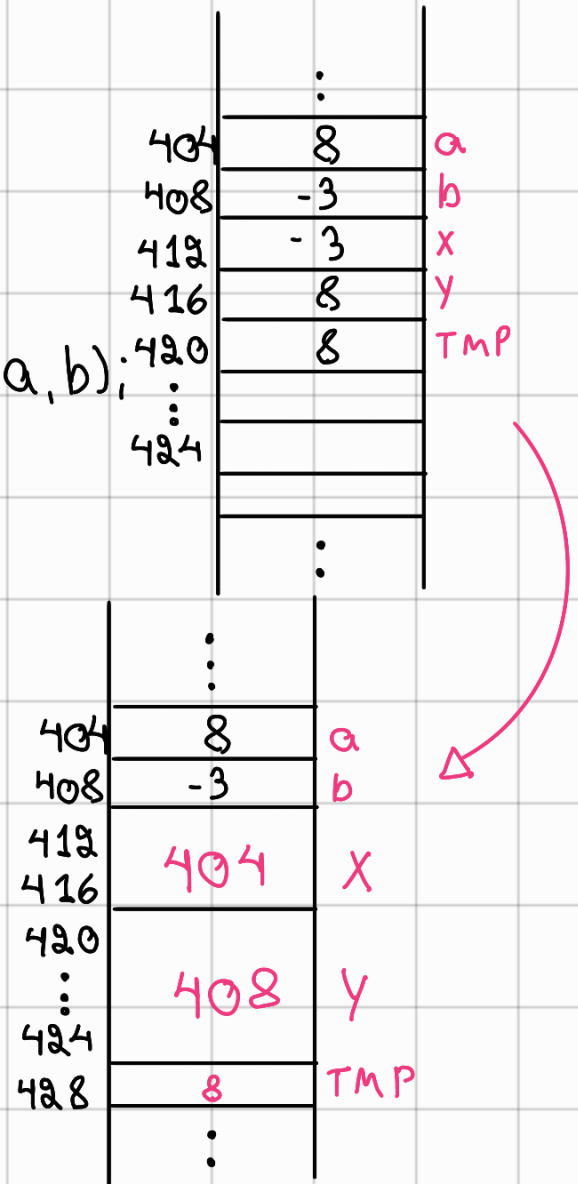
Aplicações de Ponteiros

① Passagem de parâmetros.

Ex.: Uma função que troque o valor de duas variáveis.

```
INT MAIN {
    INT a = 8, b = -3;
    TROCA(&a, &b);
    PRINTF("a = %d, b = %d \n", a, b);
    RETURN 0;
}

VOID TROCA(INT* x, INT *y){
    INT TMP = *x;
    *x = *y;
    *y = TMP;
}
```



Em C, a passagem de parâmetros em funções é feita por valor (ou cópia). Da forma acima, dizemos que a e b foram passados por referência.