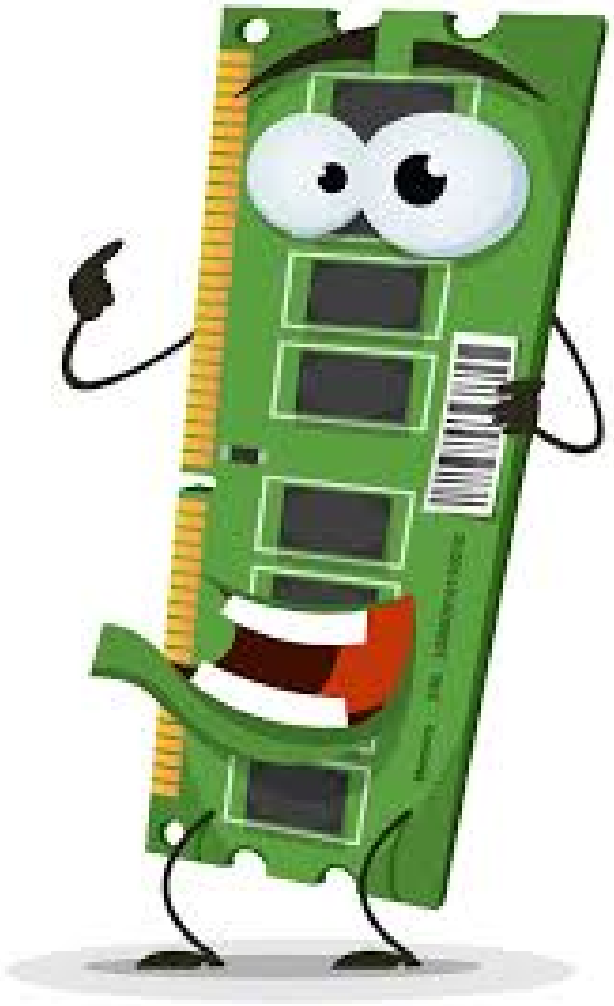


UNIDADE III: PONTEIROS

Felipe Cunha

Introdução

- Um **ponteiro** é um endereço de memória. Seu valor indica **onde** uma variável está armazenada, **não o que está armazenado**. Um ponteiro proporciona um modo de acesso a uma variável sem referência-la diretamente
- Ponteiros são variáveis cujos valores são endereços de memória
- Uma variável comum contém claramente um valor específico

Introdução

- Ponteiros são utilizados em situações em que o uso do nome de uma variável não é permitido ou é indesejável
- Ponteiros fornecem maneiras com as quais as funções podem realmente modificar os argumentos que recebem – passagem por referência
- Ponteiros alocam e desalocam memória dinamicamente no sistema;

Introdução

- Para conhecermos o endereço ocupado por uma variável usamos o operador de endereços (&)

```
#include <stdio.h>

int main(){
        int i, j, k;
        printf("%p\n%p\n%p\n",&i,&j,&k);
        return 0;
}
```

Resultado:
0028FF1C
0028FF18
0028FF14

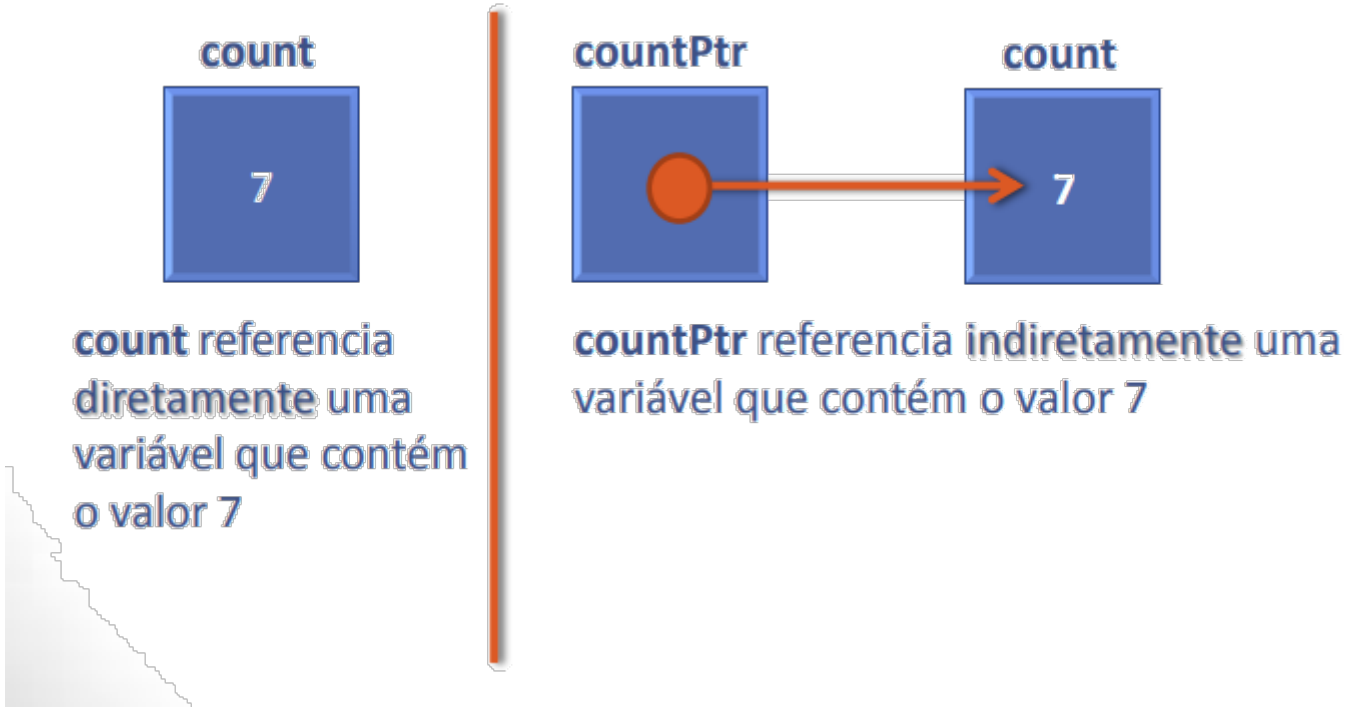
Introdução

- Um ponteiro, diferentemente de uma variável comum, contém um endereço de uma variável que contém um valor específico
- Um ponteiro referencia um valor indiretamente;
- Ponteiros devem ser definidos antes de sua utilização, como qualquer outra variável
- Exemplo:

```
int *countPtr, count;
```

Introdução

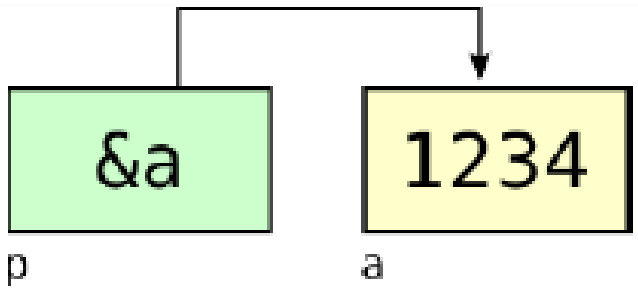
- A variável **count** é definida para ser um inteiro, e não um ponteiro para **int**
- Se quisermos que **count** seja um ponteiro, devemos alterar a declaração **int count** para int ***count**



Introdução

- Para atribuir um valor ao ponteiro, usamos apenas seu nome de variável. Esse valor deve ser um endereço de memória, portanto obtido com o operador &:

```
int a;
int *p;
p = &a;
```



Introdução

- Cuidado:
  - A notação *, usada para declarar variáveis de ponteiro, não distribui para todos os nomes de variáveis em uma declaração
  - Cada ponteiro precisa ser declarado com o * prefixado ao nome;

    Exemplo: **int *xPtr, *yPtr;**
- Dica:
  - Inclua as letras Ptr nos nomes de variáveis de ponteiros para deixar claro que essas variáveis são ponteiros
  - Inicialize os ponteiros para evitar resultados inesperados;

Introdução

- Como o ponteiro contém um endereço, podemos também atribuir um valor à variável guardada nesse endereço, ou seja, à variável apontada pelo ponteiro. Para isso, usamos o operador * (asterisco), que basicamente significa "o valor apontado por".

```
#include <stdio.h>
int main(){
	int i = 10 ;
	int *p = &i ;
	*p = 5 ;
	printf ("%d\t%d\t%p\n", i, *p, p);
	return 0;
}
```

Saída:

```
5	5	0028FF18
```

Inicialização

- Ponteiros devem ser inicializados quando são definidos ou então em uma instrução de atribuição
- Ponteiros podem ser inicializados com **NULL**, zero, ou um endereço
- **NULL**: não aponta para nada, é uma constante simbólica;
- Inicializar um ponteiro com zero é o mesmo que inicializar com NULL;

```
int *xPtr = NULL;

int *yPtr = 0;
```

Operação com Ponteiros

- Suponhamos dois ponteiros inicializados p1 e p2. Podemos fazer dois tipos de atribuição entre eles:
  - Esse primeiro exemplo fará com que p1 aponte para o mesmo lugar que p2. Ou seja, usar p1 será equivalente a usar p2 após essa atribuição

```
p1 = p2;
```

- Nesse segundo caso, estamos a igualar os valores apontados pelos dois ponteiros: alteraremos o valor apontado por p1 para o valor apontado por p2

```
*p1 = *p2;
```