Engenharia de Computação

Fundamentos de Programação

Aula 19 – Ponteiros

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

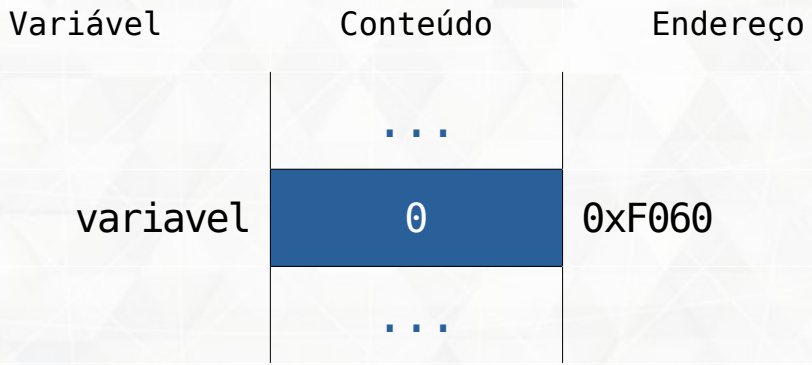
Relembrando variáveis

- Armazenam dados
  - **Exemplos**: int, float, char, double, struct
  - Declarando uma variável:

```
int variavel;
```

  - Inicializando uma variável:

```
int variavel = 0;
```



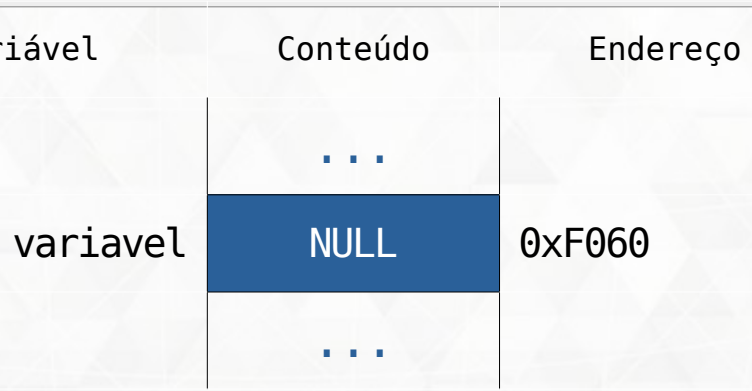
Ponteiros

- Tipo de variável que armazena um endereço de memória
  - Declarando ponteiro:

```
int *ptr;
```

  - Inicializando um ponteiro:

```
int *ptr = NULL;
```



Ponteiros não inicializados apontam para um endereço indefinido.
NULL é uma constante que aponta para um endereço inexistente

Ponteiros

- Considerando o ponteiro ptr:

```
int *ptr;
```

  - **`ptr`** armazena o **endereço de memória** de uma variável do tipo **int**

- **Re**ferenciação

```
ptr = &var;
```

  - O operador **&** retorna o **endereço de memória** onde a variável `var` está salva

- **De**referenciação

```
printf("Valor: %d", *ptr);
```

  - O operador ***** retorna o **conteúdo do endereço** de memória apontado por `ptr`

Ponteiro - Resumo

Variável	Conteúdo	Endereço
	...	
variavel	15	F060
ponteiro	F060	F064
	...	

Ponteiros - Exemplo

```c
int variavel = 15;
int *ptr;

//ponteiro ptr recebe o endereço de memória da variável
ptr = &variavel;

printf("Valor da variavel: %d\n", variavel);
printf("Endereco de memoria da variavel: %p\n", &variavel);
printf("Valor do ponteiro: %p\n", ptr);
printf("Dereferenciando o ponteiro: %d\n", *ptr);
```

```
Valor da variavel: 15
Endereco de memoria da variavel: 0x7ffe5280651c
Valor do ponteiro: 0x7ffe5280651c
Dereferenciando o ponteiro: 15
```

Operações sobre ponteiros

- Atribuição
  - Um ponteiro só pode receber o endereço de uma variável do mesmo tipo do ponteiro

```
int *ptr1, *ptr2, x = 10;
float y = 20;

ptr1 = &x;// Mesmo tipo *int – OK!
printf("Dereferenciando ptr1: %d\n", *ptr1);

ptr2 = ptr1;// Mesmo tipo *int – Ok!
printf("Dereferenciando ptr2: %d\n", *ptr2);

ptr1 = &y; //Tipos diferentes – ERRO!
printf("Dereferenciando ptr1: %f\n", *ptr1);
```

Operações sobre ponteiros

- Aritmética de ponteiros
  - Sobre o valor armazenado por um ponteiro podemos apenas **somar e subtrair** valores inteiros
  - Somas e subtrações vão se comportar **de acordo com o tipo** do dado apontado.
  - **Exemplo:** ao incrementar um ponteiro para inteiro, o valor do ponteiro se desloca em **4 bytes**

```
int *ptr = (int*) 0x5DC;//1500

printf("Endereço armazenado: %p - %d\n", ptr, ptr);//1500
ptr++;
printf("Endereço armazenado: %p - %d\n", ptr, ptr);//1504
ptr = ptr + 15;
printf("Endereço armazenado: %p - %d\n", ptr, ptr);//1564
ptr = ptr - 2;
printf("Endereço armazenado: %p - %d\n", ptr, ptr);//1556
```

Operações sobre ponteiros

- Comparação
  - É possível comparar dois ponteiros

```c
int valor1 = 10, valor2 = 10;
int *ptr1, *ptr2;

ptr1 = &valor1;
ptr2 = &valor2;

if (ptr1 == ptr2) {
    printf("Ponteiros iguais!\n");
}else{
    printf("Ponteiros diferentes!\n");
}// else
```

```
Ponteiros diferentes!
```

Ponteiro Genérico

- Aponta para qualquer tipo de dados existente ou a ser criado

  - Declarando um ponteiro genérico

```
void* ptrGenerico;
```

  - Antes de utilizar é necessário conerter o ponteiro para o tipo com o qual se deseja trabalhar

```
printf(“Exibindo o valor: %d”, *(int*)ptrGenerico);
```

Ponteiro Genérico

- Aritmética de ponteiros genéricos
  - Somas e subtrações vão se comportar **como se o tipo ocupasse apenas um byte.**
  - **Exemplo:** ao incrementar um ponteiro genérico, o valor do ponteiro se desloca em **1 byte**

```
void *ptr = 0x5DC;//1500

printf("Endereço armazenado em ptr: %p\n", ptr);//1500

ptr++;
printf("Endereço armazenado em ptr: %p\n", ptr);//1501

ptr = ptr + 15;
printf("Endereço armazenado em ptr: %p\n", ptr);//1516

ptr = ptr - 2;
printf("Endereço armazenado em ptr: %p\n", ptr);//1514
```

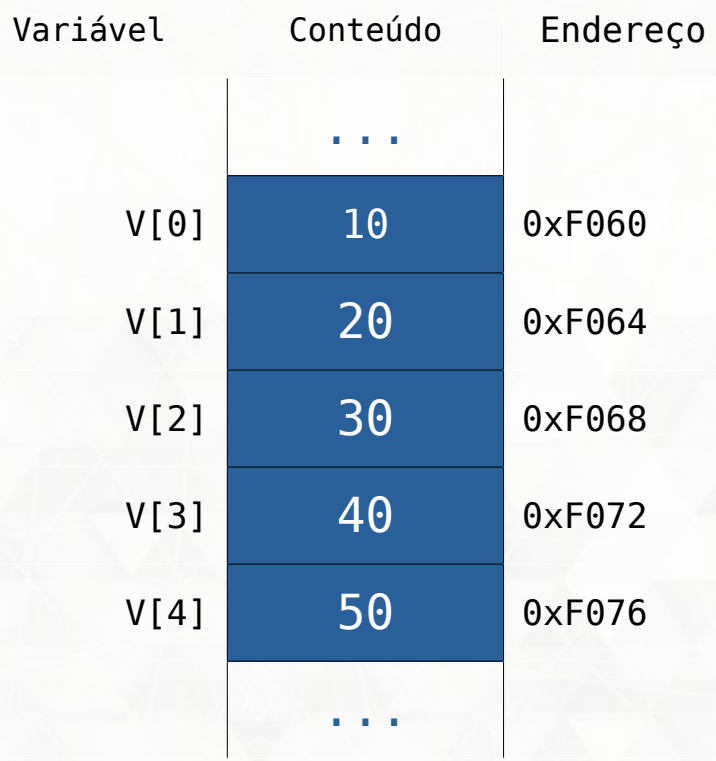
Ponteiros e Vetores (Arrays)

- **Vetores:** Conjuntos de dados armazenados de forma **sequencial**

```
int vetor[5] = {10, 20, 30, 40, 50};
int *ptrVet = vetor;

for (int i = 0; i < 5; i++) {
  printf("%d \n", ptrVet[i]);
}//for
```

- **vetor** é um ponteiro para a primeira posição do vetor
- É possível copiar esse vetor para outro ponteiro e **acessar** seus elementos normalmente.



Ponteiros e Vetores (Arrays)

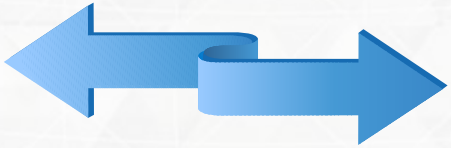
- Como **vetor** é um ponteiro para a primeira posição do vetor:
  - Podemos utilizar aritmética de ponteiros para acessar os demais elementos

```
int vetor[5] = {10, 20, 30, 40, 50};
int *ptrVet = vetor;

for (int i = 0; i < 5; i++) {
  printf("%d \n", *(ptrVet + i));
}//for
```

- São equivalentes as seguintes formas:

```
ptrVet[i]
```



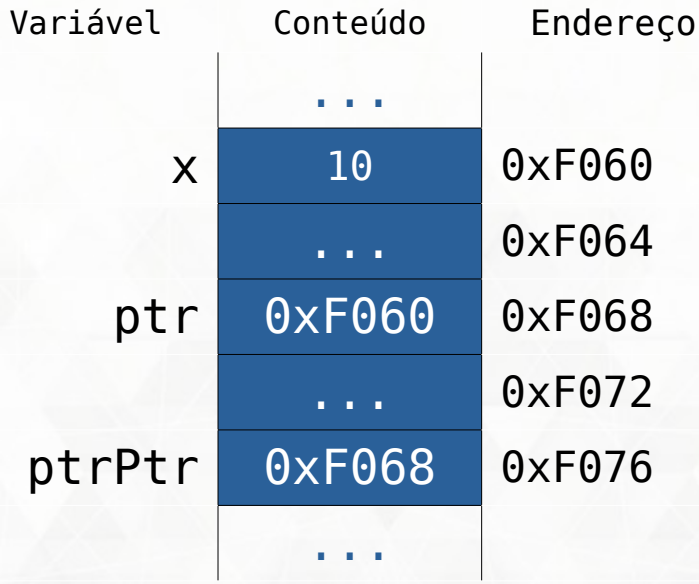
```
*(ptrVet + i)
```

Ponteiro para ponteiro

- A linguagem C permite criar diferentes níveis de apontamento → Ponteiros que apontam para ponteiros

```c
int x = 10;
int *ptr = &x;
int **ptrPtr = &ptr;

printf("%d\n", x);
printf("%d\n", *ptr);
printf("%d\n", **ptrPtr);
```



Na linguagem C é possível fazer ainda mais níveis de apontamento, como: ***ptr**, ****ptr** e assim por diante.

Exercícios

- **1)** Escreva um programa que contenha duas variáveis inteiras. Compare os endereços e exiba o endereço e o conteúdo do maior endereço.
- **2)** Crie um programa que contenha um vetor float (tamanho 10). Imprima o endereço de cada posição desse vetor.
- **3)** Crie um programa que contenha uma matriz (3x3) de float. Imprima o endereço de cada posição dessa matriz.
- **4)** Crie um programa que contenha um vetor de inteiros de tamanho 5. Utilizando aritmética de ponteiros, leia os dados do teclado e exiba seus valores multiplicado por 2. Em seguida, exiba o endereço de memória das posições que contém valores pares.

Exercícios

- **5)** Elabora uma função que receba duas strings como parâmetros e verifique se a segunda string está dentro da primeira. Para isso, utilize apenas aritmética de ponteiros.
- **6)** Considere a seguinte declaração:

```
int a, *b, **c, ***d;
```

Escreva um programa que receba o valor de a, calcule e exiba:

- O dobro do valor a, utilizando o ponteiro b
- O triplo do valor a, utilizando o ponteiro c
- O quadruplo do valor a, utilizando o ponteiro d