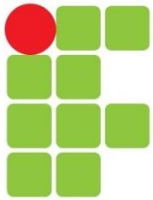


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MATO GROSSO DO SUL

Algoritmos II

Ponteiros

Prof. Rogério Alves dos Santos Antoniassi
Prof. Alex Fernando de Araujo

Introdução

- Ponteiro é um tipo que armazena o endereço de memória de uma variável, estrutura, função, entre outros.

- Ou seja, um ponteiro guarda a referência daquele elemento.

- Pode-se dizer que o ponteiro “aponta” para onde está a variável que ele recebeu o endereço.

Exemplo

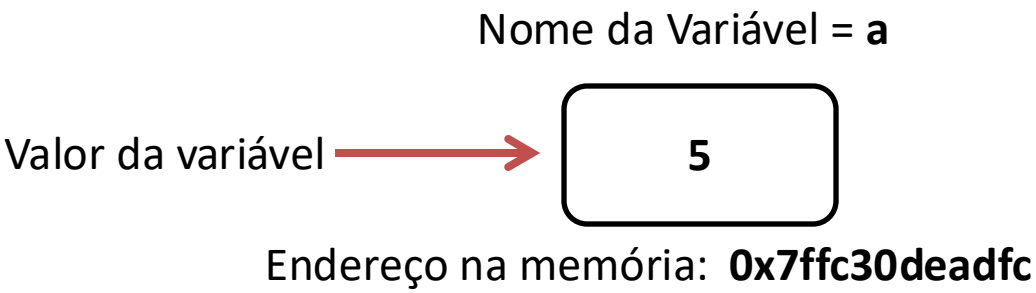
- Toda vez que se cria uma variável, é reservado um local na memória para ela.

- Esse endereço é associado ao nome que é atribuído a variável.

- O computador precisa da posição das variáveis na memória, já nós trabalhamos com o nome da variável.

Exemplo

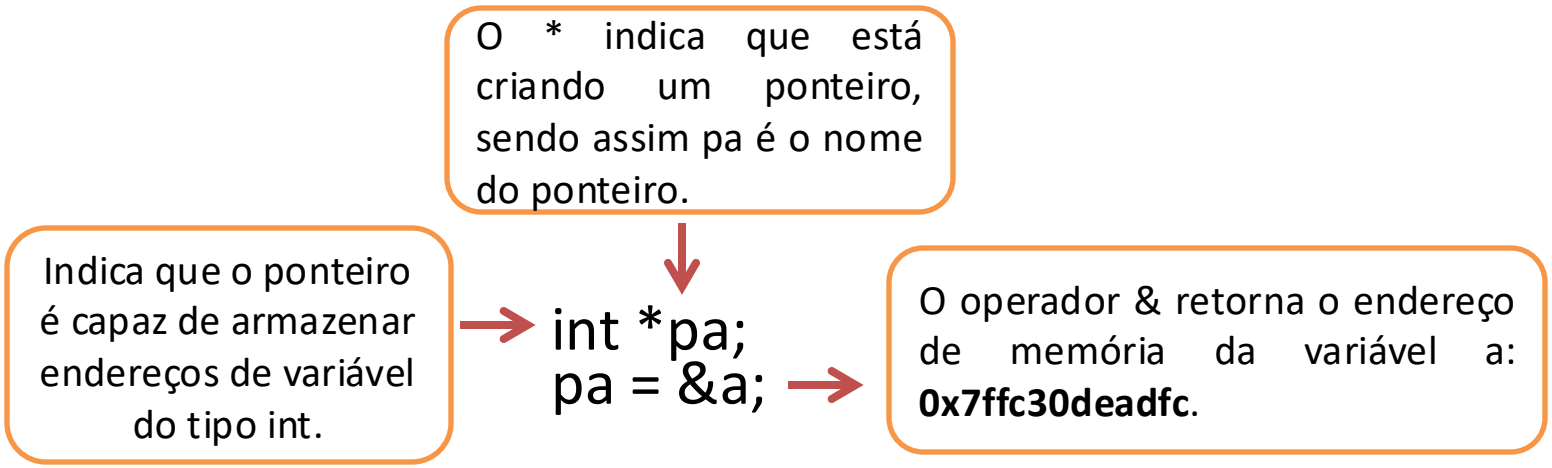
- Na declaração de variável abaixo, tem-se a variável a, que recebe o valor 5.
    - int a = 5;



Exemplo

- Considerando o exemplo anterior, pode-se criar um ponteiro para fazer referência a variável “a”.

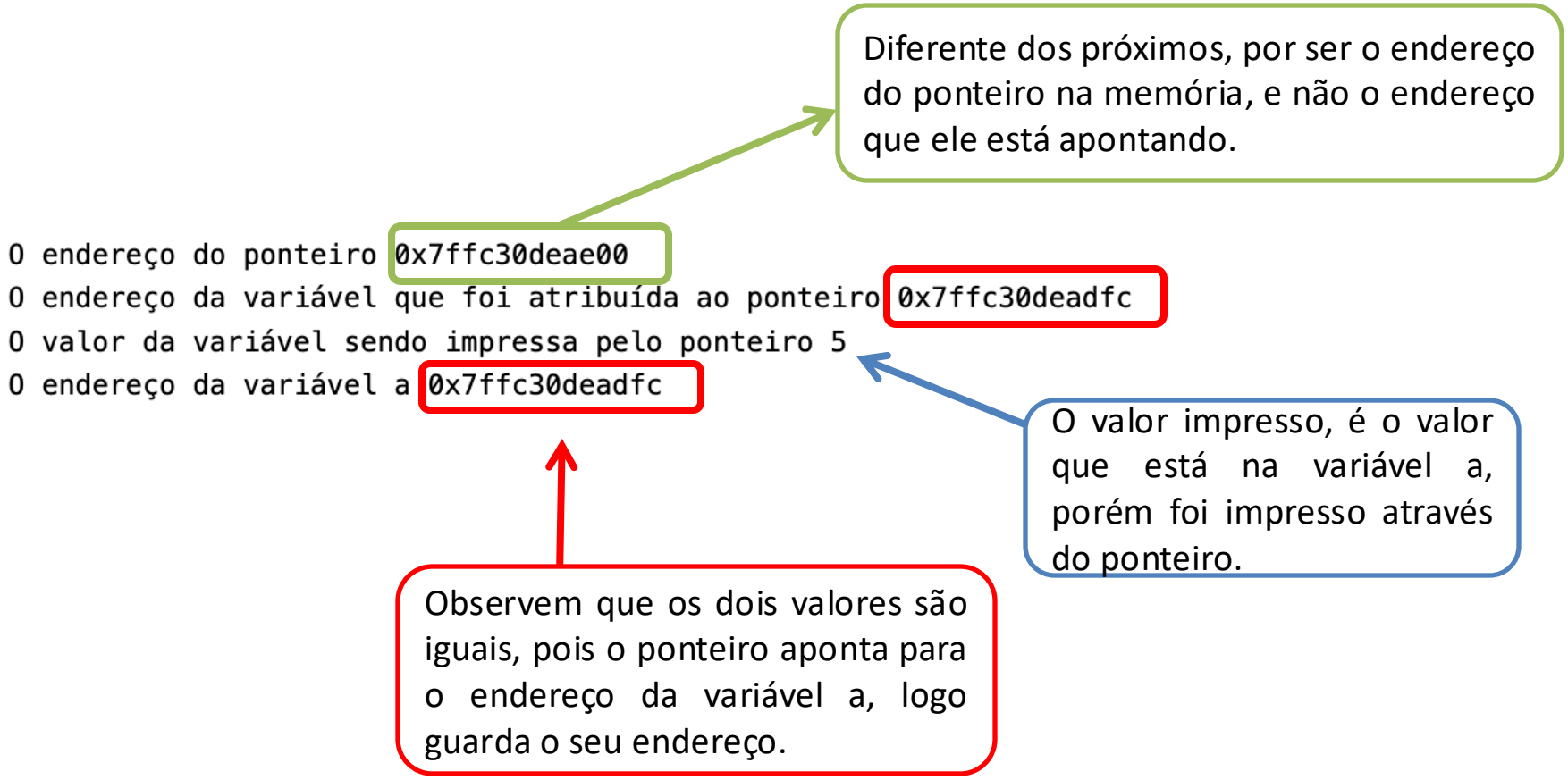
  – Na declaração de variável abaixo, tem-se o ponteiro pa, que recebe o endereço da variável a.



Exemplo completo

```c
#include<stdio.h>

int main()
{
    int a = 5;
    int *pa;
    pa = &a;

    printf("\n O endereço do ponteiro %p", &pa);
    printf("\n O endereço da variável que foi atribuída ao ponteiro %p", pa);
    printf("\n O valor da variável sendo impressa pelo ponteiro %d", *pa);


    printf("\n O endereço da variável a %p", &a);

    return 0;
}
```

Saída do exemplo



Teste

1 – Crie um outro ponteiro de nome p, e atribua pa a ele, mostre os valores e endereços de memória. Verifique o resultado;

2 – Altere o valor do ponteiro *pa e os valores e endereços de memória. Verifique o resultado.

Resultado do primeiro Teste

```c
#include<stdio.h>

int main()
{
    int a = 5;
    int *pa, *p;
    pa = &a;

    p = pa;

    printf("\n O endereço do ponteiro %p", &pa);
    printf("\n O endereço da variável que foi atribuída ao ponteiro %p", pa);
    printf("\n O valor da variável sendo impressa pelo ponteiro %d", *pa);


    printf("\n O endereço da variável a %p", &a);

    printf("\n O endereço da variável que foi atribuída ao ponteiro p %p", p);
    printf("\n O valor da variável sendo impressa pelo ponteiro p %d", *p);

    return 0;
}
```

```
O endereço do ponteiro 0x7ffdbba4e768
O endereço da variável que foi atribuída ao ponteiro 0x7ffdbba4e764
O valor da variável sendo impressa pelo ponteiro 5
O endereço da variável a 0x7ffdbba4e764
O endereço da variável que foi atribuída ao ponteiro p 0x7ffdbba4e764
O valor da variável sendo impressa pelo ponteiro p 5
```

Resultado do segundo Teste

```c
#include<stdio.h>

int main()
{
    int a = 5;
    int *pa, *p;
    pa = &a;

    p = pa;

    *pa = 10;

    printf("\n O valor da variável a %d", a);

    printf("\n O endereço do ponteiro %p", &pa);
    printf("\n O endereço da variável que foi atribuída ao ponteiro %p", pa);
    printf("\n O valor da variável sendo impressa pelo ponteiro %d", *pa);


    printf("\n O endereço da variável a %p", &a);

    printf("\n O endereço da variável que foi atribuída ao ponteiro p %p", p);
    printf("\n O valor da variável sendo impressa pelo ponteiro p %d", *p);

    return 0;
}
```

```
O valor da variável a 10
O endereço do ponteiro 0x7ffc02877388
O endereço da variável que foi atribuída ao ponteiro 0x7ffc02877384
O valor da variável sendo impressa pelo ponteiro 10
O endereço da variável a 0x7ffc02877384
O endereço da variável que foi atribuída ao ponteiro p 0x7ffc02877384
O valor da variável sendo impressa pelo ponteiro p 10
```

Observem que todos passam a mostrar o valor 10, pois o 10 foi atribuído ao endereço de memória que corresponde a variável “**a**”. Obs: o valor de “**a**” é 10, “pa” apenas aponta para a variável “**a**”..

Ponteiro genérico

- É possível criar um ponteiro genérico, que receba endereço de qualquer tipo de variável.

- Tenha atenção ao efetuar operações aritméticas com esse tipo de ponteiro, já que não possui tipo definido.

- Caso queira atribuir um ponteiro genérico a um ponteiro de tipo específico, é necessário efetuar um *typecast* para o tipo desejado (O mesmo ocorre quando for acessar o seu conteúdo).

```
void *p;
char *ca;

ca =(char *)p;
```

Aritmética com ponteiros

- Só é possível realizar duas operações nos endereços:
  - Adição;
  - Subtração.

Pesquisem a diferença entre:
*pa++;
(*pa)++;
*(pa++);

```
int a = 5;
int *pa;

pa = &a;

pa++;
pa = pa + 4;
pa--;
pa = pa - 2;
```

Observem que incrementa e decrementa o endereço da memória e não o valor.

Obs: Se fosse *pa++;
Estaria acessando o conteúdo ao qual o ponteiro está apontando (valor de “a”), dessa forma valeria todas as operações aritméticas que o tipo permite.
Ex: *pa = (*pa) *2;
*pa = (*pa)++;

Aritmética com ponteiros

- Muito útil para trabalhar com arrays.

- Não é possível incrementar ou decrementar números quebrados (pois é a posição na memória), sempre deve ser um valor inteiro.

- Ao testar, vão notar que o ponteiro incrementou 4 posições na memória, ex: se estava na posição 100, foi para a 104. O motivo é que o valor é do tipo inteiro, sendo assim é reservado 4 bytes para armazenar a variável.

Operações relacionais

- É possível verificar se os ponteiros são iguais == ou diferentes !=.

- Além de poder comparar se está em posições maiores ou menores na memória > ou <.

- Assim como nas operações aritméticas, caso utilize o * ex: *pa, estará comparando o conteúdo presente no endereço que o ponteiro está apontando.

Parâmetros com ponteiros

- Uma função pode receber parâmetros por valor (que vocês já fizeram) ou por referência com a utilização dos ponteiros, ou seja a variável da função irá receber a referencia de memória.

- Permite que as alterações realizadas na função altere o valor de variáveis que estão fora da função.

Exemplo

```c
#include<stdio.h>

void alterarValor(int *numero){
    *numero = 2;
}

int main()
{
    int valor = 5;
    printf("\n O valor da variável é: %d", valor);

    alterarValor(&valor);

    printf("\n O novo valor da variável é: %d", valor);

    return 0;
}
```

* Está indicando que a função tem um parâmetro que é um ponteiro, ou seja recebe a referência.

Chamada da função passando o endereço de memória da variável

```
O valor da variável é: 5
O novo valor da variável é: 2
```

Observem que o valor foi alterado dentro da função, porém sem utilizar um return.

Parâmetros com ponteiros

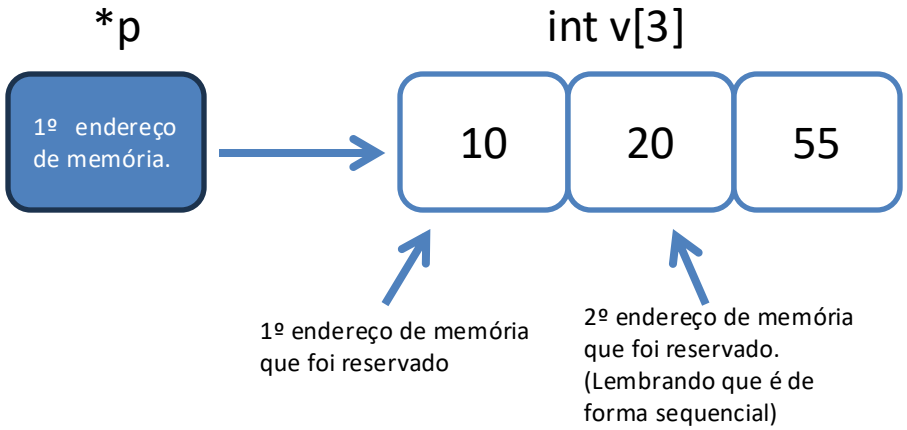
- Uma função pode receber parâmetros por valor (que vocês já fizeram) ou por referência com a utilização dos ponteiros, ou seja a variável da função irá receber a referencia de memória.

- Permite que as alterações realizadas na função altere o valor de variáveis que estão fora da função.

Vetores

- Quando criamos um vetor, é “reservado” um espaço memória para armazenar os elementos de forma sequencial;

- Dessa forma, tem-se um ponteiro que aponta para o início da sequencia na memória;



- O espaço reservado na memória para esse vetor de 3 posições foi de 12 bytes, pois um dado do tipo int reserva 4 bytes na memória.

Vetores

```c
#include<stdio.h>

int main()
{
    int vTeste[3] = {10, 20, 55};
    int *p = vTeste;

    for(int i = 0; i < 3; i++){
        //acessando o valor pelo ponteiro
        printf("%d \n", *(p+i));
    }

    return 0;
}
```

É a mesma saída que se utilizasse vTeste[i], mas nesse caso estamos utilizando o ponteiro e as operações aritméticas.

Exercícios

1 – Implemente um exemplo mostrando as diferenças dos três itens abaixo:

- *pa++;
- (*pa)++;
- *(pa++);

2 – Considerando que string é uma sequência de caracteres na memória, sua implementação é similar a de um vetor, dessa forma implemente um exemplo utilizando ponteiros para percorrer uma string.

3 – É possível criar um vetor de ponteiros? Há diferença na forma de utilizar?

ex: int *v[10];

Referências

- ASCENCIO, Ana Fernanda Gomes; DE CAMPOS, Edilene Aparecida Veneruchi. **Fundamentos da programação de computadores**. Pearson Educación, 2008.
- BACKES, André. ***Linguagem C: completa e descomplicada***. 2. ed., 4. imp. Rio de Janeiro: LTC, 2023.
- CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Elsevier, c2012.
- EVARISTO, Jaime. **Aprendendo a programar: programando em linguagem C**. Rio de Janeiro: Book Express, 2001. 205 p.
- MEDINA, Marco. **Algoritmos e programação: teoria e prática**. São Paulo: Novatec, 2006.
- MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C**. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, Marcos. **Estrutura de Dados com Algoritmos e C**. Rio de Janeiro: Brasport, 2008.