Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP Campus Boituva
Curso de Análise e Desenvolvimento de Sistemas

ROBERTO MOTA DOS SANTOS

VETORES COMO PARÂMETRO PARA FUNÇÕES

Boituva/SP
2º Semestre/2022

SUMÁRIO

PONTEIROS

Antes de falarmos sobre qualquer coisa, precisamos primeiro entender conceitos sobre ponteiros.

Como o próprio nome diz, ponteiros apontam para o valor de uma variável.

Um exemplo:

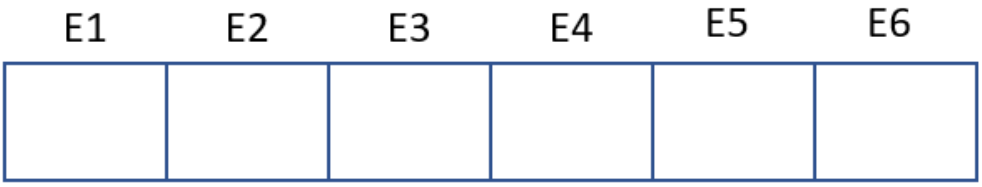
```c
#include <stdlib.h>
#include <stdio.h>


int main(int argc, char argv[])
{
    int num = 10;
    int *ponteiro;
    ponteiro = &num;

printf("variavel NUM: %d\n variavel ponteiro: %d\n", num, *ponteiro);
    return 0;
}
```

Nesse caso especifico, o nosso ponteiro está apontando para o **ENDEREÇO** da variável num, e quando é dado o comando para imprimir utilizando “*”, a saída é o valor que está contido para onde esse ponteiro aponta, ou seja, para a variável num que por sua vez vale 10..

```
variavel NUM: 10
variavel ponteiro: 10
PS G:\Meu Drive\IFSP\Estrutuda de dados I\2022-08-12 - Trabalho>
```

De forma mais gráfica e humana, podemos interpretar os ponteiros da seguinte forma:

Vamos representar nossa memoria com esses *slots* abaixo, onde, E1 é um *slot, E2* é outro *slot* e assim por diante.

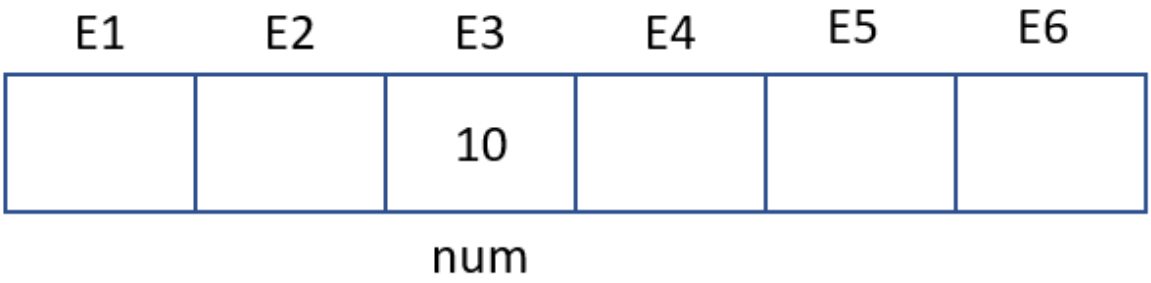


Quando declaramos uma variável, estamos reservando um espaço (*slot*) na memória, e de acordo com cada tipo de variável é reservado uma quantidade de espaço diferente, mas para nos mantermos no assunto vamos ignorar essa parte.

Então no nosso código declaramos:

```
int num = 10;
```

Estamos dizendo que num será uma variável do tipo inteiro e precisará de X espaços na memória, mas no nosso exemplo, vamos reservar somente um espaço para num.

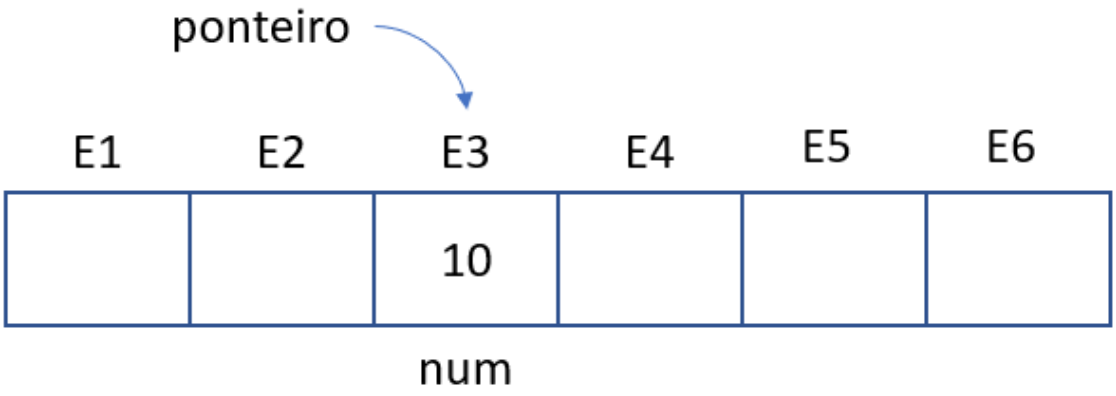


Entendemos que, num está no endereço E3 que para facilitar as chamadas do mesmo, o compilador “apelida” o endereço de num, assim sendo mais simples e lógico chamar a variável.

Nesse ponto do código precisaremos por alguns motivos específicos, criar um ponteiro para essa variável, então criamos da seguinte maneira:

```
int *ponteiro;
ponteiro = &num;
```

Agora temos um ponteiro que está apontando para o ENDEREÇO da variável num:



Agora temos um ponteiro que aponta para o endereço de uma variável que contém um valor, e por sua vez, o ponteiro também possui um espaço na memória, mas é um assunto que não será abordado nesse trabalho.

Quando o assunto é um ponteiro, são necessários alguns cuidados com sua manipulação, pois se é desejado o valor para o qual esse ponteiro está apontado, deve-se chamar o ponteiro com "*" no começo, que remete a algo "o valor de". Exemplo:

Pode-se ler: *ponteiro como "o valor de" ponteiro, com essa analogia fica mais fácil de interpretar e tomar ações com o ponteiro.

Note que se for impresso a variável ponteiro sem o asterisco, teremos um valor estranho e diferente de 10, que deveria ser o resultado:

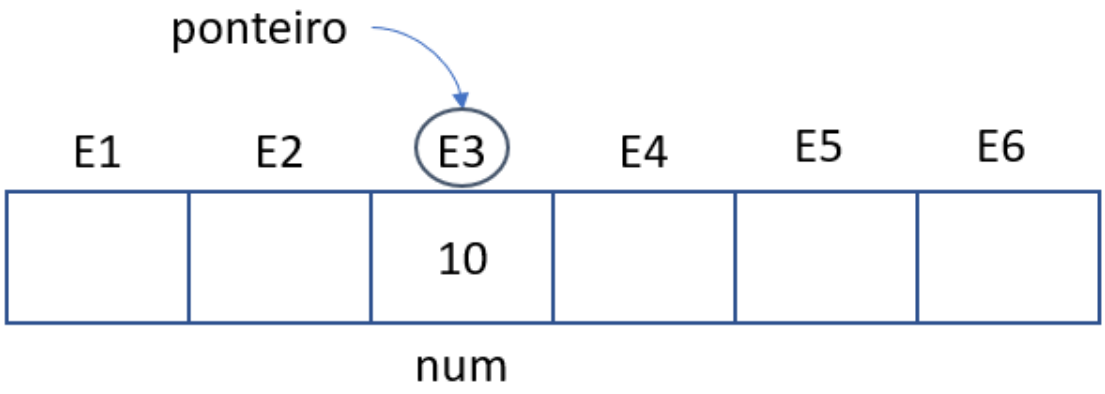
Código:

```
printf("variavel NUM: %d\nvariavel ponteiro: %i\n", num, ponteiro);
```

Saída:

```
variavel NUM: 10
variavel ponteiro: 6422292
PS G:\Meu Drive\IFSP\Estrutuda de dados I\2022-08-12 - Trabalho>
```

Isso ocorre porque o que foi solicitado, foi o valor que está contido no ponteiro e não o valor que está contido no **ENDEREÇO** do ponteiro.



O correto seria:

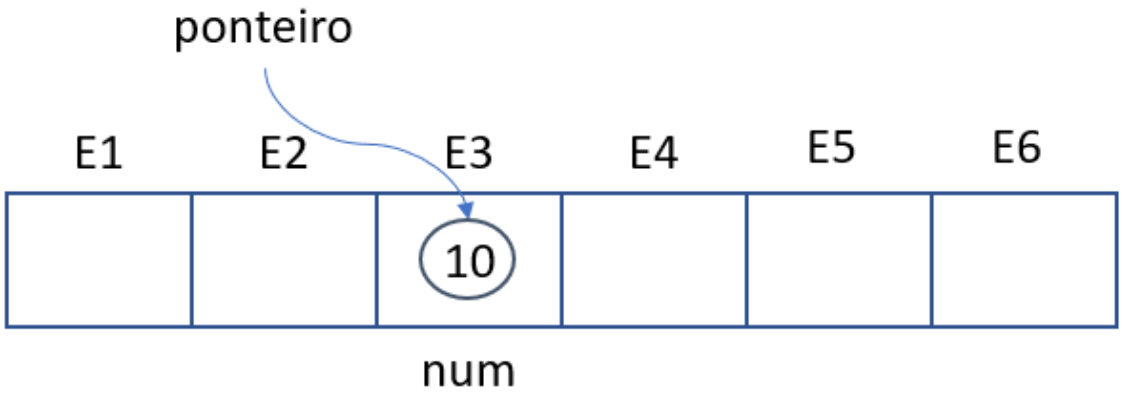
Código:

```
printf("variavel NUM: %d\nvariavel ponteiro: %i\n", num, *ponteiro);
```

Saída:

```
variavel NUM: 10
variavel ponteiro: 10
```

Fazendo a analogia da leitura do ponteiro, leríamos essa requisição impressão como “o valor de” ponteiro.



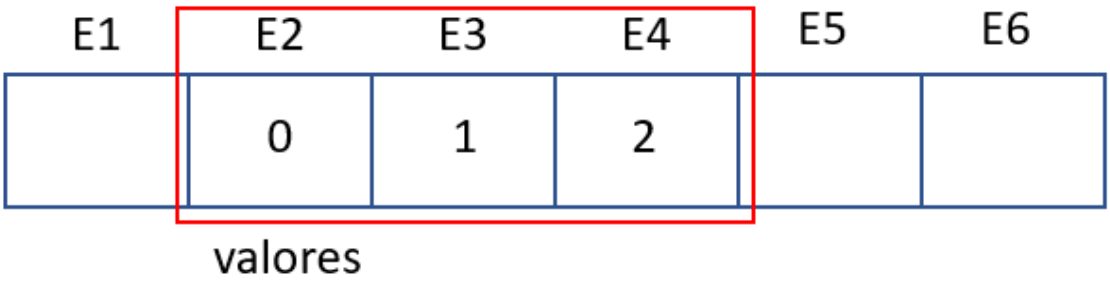
VETORES

Um vetor é um conjunto de dados contidos em um “pacote” único dividido por endereços internos.

```
int valores[3] = {0, 1, 2};
```

Nesse código, o pacote é “valores” do tipo inteiro e podendo conter 3 valores e sendo esses valores atribuídos na declaração da variável.

A memória agora fica dessa forma:



Onde E2 é o endereço do vetor e sendo reservado 3 espaços para ele, uma vez que já foi definido que ele terá esse tamanho no MÁXIMO.

E é interessante quando é dado um comando “*printf*” com a variável valores somente:

Código:

```
printf("%d", valores);
```

Saída:

```
6422280
PS G:\Meu Drive\IFSP\Estrutuda de dados I\2022-08-12 - Trabalho>
```

O que acontece é que valores contém o **ENDEREÇO** do vetor, e para acessar as informações, é preciso indexar o vetor:

Código:

```
printf("%d", valores[2]);
```

Saída

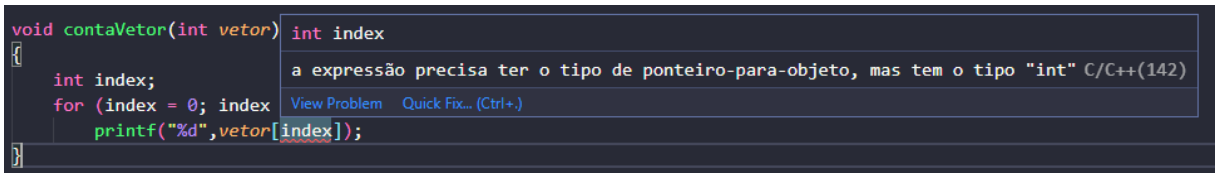
```
2
PS G:\Meu Drive\IFSP\Estrutuda de dados I\2022-08-12 - Trabalho>
```

VETORES COMO PARAMETROS

Quando é preciso passar um vetor como parâmetro, basta somente colocar o nome da variável como parâmetro sem “&” uma vez que o nome do vetor é o próprio endereço.

O que vai mudar, é a forma que a função recebe esse vetor, uma vez que é preciso informar o tipo de dado que irá chegar como parâmetros:

```
int main(int argc, char argv[])
{
    int vetor[5] = {0, 1, 2, 3, 4};
    contaVetor(vetor);
}
```



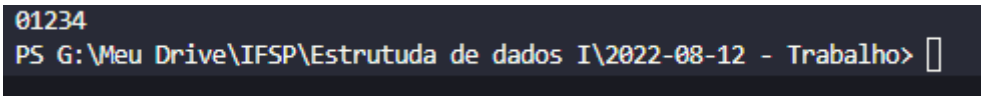
Nessa tentativa de chamar uma função passando um vetor como parâmetro, o problema ocorre porque a função espera um **VALOR** inteiro e acaba recebendo um endereço de um vetor.

Então adaptando o código para que essa função funcione perfeitamente com vetores como parâmetro, logo o código fica dessa forma:

```
void contaVetor(int *);

int main(int argc, char argv[])
{
    int vetor[5] = {0, 1, 2, 3, 4};
    contaVetor(vetor);
}

void contaVetor(int *vetor)
{
    int index;
    for (index = 0; index < 5; index++)
        printf("%d",vetor[index]);
}
```

Onde a função tem como parâmetro um ponteiro, e esse ponteiro irá receber um endereço, o que faz todo sentido uma vez que se tem o entendimento de ponteiros e endereços.

Dessa forma é possível trabalhar normalmente com o vetor dentro da função, seja coletando valores ou iterando com laços de repetição.



REFERÊNCIAS

Adriano Cruz. Curso de Linguagem C, Disponível em http://equipe.nce.ufrj.br/adriano

Ulysses de Oliveira. Programando em C, Editora Ciência Moderna.