

Aula 2

Linguagem de Programação

Prof. Sandro de Araújo

Conversa Inicial

Conteúdos desta aula

- Ponteiros
- Ponteiros variáveis
- Endereços de memória
- Ponteiros e *array*
- Ponteiros e funções

- Os objetivos desta aula são
 - Conhecer o conceito de *ponteiro* e sua aplicação em algoritmos computacionais
 - Entender como um dado é acessado na memória e sua relação com vetores e funções

Definição de dados

- Apontadores ou ponteiros são variáveis que armazenam o endereço de outras variáveis na memória
- Dizemos que um ponteiro *aponta* para uma variável na memória quando ele contém o seu endereço

- Esses elementos armazenam informações na memória e são chamados de palavras
- Cada palavra é identificada com base em um endereço de memória e não tem ambiguidade



Ordem	Endereços	Palavras
0	000	Palavra 0
1	001	Palavra 1
2	010	Palavra 2
3	011	Palavra 3
4	100	Palavra 4
5	101	Palavra 5
6	110	Palavra 6
7	111	Palavra 7

- Um programa é um conjunto de informações armazenadas na memória

- Pode ser dividido em duas categorias:
 - Instruções → operações (o programa propriamente dito) realizadas pela máquina
 - Dados → variáveis, ou valores, processados nessas operações

- O uso de ponteiros é muito útil quando um dado deve ser acessado na memória em diferentes partes de um programa
- Portanto, é possível haver vários ponteiros espalhados, indicando a localidade da variável que contém o dado desejado

- Sintaxe de declaração de um ponteiro:
 - tipo *nome_ponteiro
- Na qual temos:
 - tipo: refere-se ao tipo de dado da variável armazenada apontada pelo endereço do ponteiro
 - *nome_ponteiro: o nome da variável ponteiro
- O uso do asterisco serve para determinar que a variável usada será um ponteiro

Ponteiros variáveis

- Toda declaração de variável aloca uma porção da memória
- O tamanho do espaço ocupado é relativo à tipificação da variável



- Um ponteiro também é uma variável e também ocupa espaço na memória
- Normalmente o tamanho de um ponteiro é independente do tipo de dados da variável para a qual está apontando e ocupa o espaço de um inteiro

- Dizemos que uma variável é um ponteiro quando aponta para outra variável; quando a primeira contém o endereço da segunda



- `int *x, *y, c = 5, d = 3`
- `x = &c; 5` // x aponta para c
- `y = &d; 3` // y aponta para d
- `*y = 8; 8` // alterado o valor existente na variável d
- `*x = *y; 8` // copia o valor de d (apontado por y) para c (apontado por x)

- `*x = 1; 1` // alterado o valor da variável c
- `y = x; 1` // y aponta para c
- `*y = 0; 0` // altera o valor de c
- ✓ `printf (valor da variável c: %d\n\nValor da variável d: %d\n", c, d)`

c	d
0	8

```

"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\ponteiro"
Valor da variavel C: 0
Valor da variavel D: 8
Process returned 0 (0x0)   execution time : 0.232 s
Press any key to continue.

```

Endereços de memória

- A memória de um computador é dividida em *bytes*, numerados de zero até o limite de memória da máquina
- Esses números são chamados endereços de *bytes*, usados como referências (ponteiros para a linguagem de programação C), pelo computador, para localizar as variáveis simples

- Quando o programa é carregado na memória, ocupa certa parte, e toda variável e/ou função desse programa terá seu espaço num endereço particular
- Para conhecer o endereço onde uma variável está alocada, usa-se o operador de endereços &

```

int main()
{
    int ponteiro1, ponteiro2, ponteiro3;

    /* %p para ponteiros */
    printf("O endereco de ponteiro1: %p \n", &ponteiro1);
    printf("O endereco de ponteiro2: %p \n", &ponteiro2);
    printf("O endereco de ponteiro3: %p \n\n", &ponteiro3);

    system("pause");
    return 0;
}

```

```

"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\
O endereco de ponteiro1: 0060FF0C
O endereco de ponteiro2: 0060FF08
O endereco de ponteiro3: 0060FF04

Pressione qualquer tecla para continuar. . .

```

Ponteiros e vetores

- **Arrays** ou vetores unidimensionais são um conjunto de dados de mesmo tipo, armazenados em posições sequenciais na memória
- O nome do *array* é um ponteiro que aponta para o primeiro elemento do vetor

```
int main()
{
    int x[ ] = {2, 16, 15, 3, 10};
    int *pont;

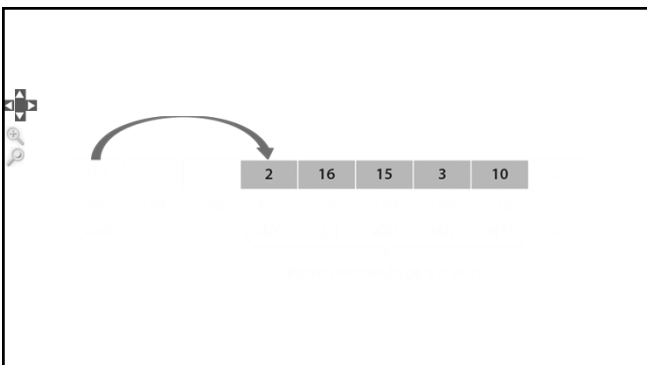
    pont = x; //atribui o endereço do vetor

    printf ("Valor de x[0]: %p\n", x);

    return 0;
}
```

```
"C:\Users\Casa\Documents\Sandro\FACULDADES\
Valor de x[0]: 0060FEF8
Valor de x[0]: 0060FEF8

Process returned 0 (0x0)   execution
Press any key to continue.
```



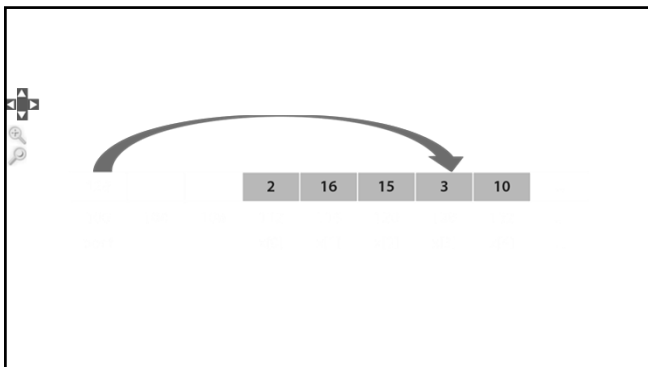
- Para se obter o endereço do primeiro elemento, basta escrever:
- 1. `int x[] = {2, 16, 15, 3, 10};`
- 2. `int *pont;`
- 3.
- 4. `pont = x;`

▪ Ou:

1. `int x[] = {2, 16, 15, 3, 10};`
2. `int *pont;`
- 3.
4. `pont = &x[0];`

▪ Logo, as instruções abaixo são usadas para obter o endereço do quarto elemento:

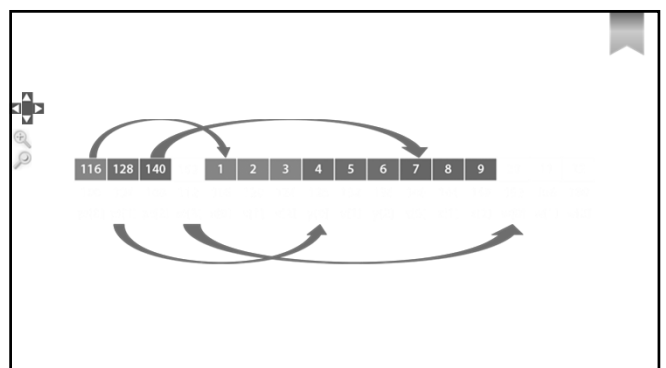
1. `int x[] = {2, 16, 15, 3, 10};`
2. `int *pont;`
- 3.
4. `*pont = &x[4];`



Array de ponteiros

- `int *pont[4];` // vetor de ponteiros do tipo inteiro
- `int x[3] = {1, 22, 322};` // primeiro vetor com três elementos
- `int y[3] = {4, 51, 66};` // segundo vetor com três elementos
- `int z[3] = {7, 83, 99};` // terceiro vetor com três elementos
- `int w[3] = {10, 11, 12};` // quarto vetor com três elementos

- `pont[0] = x;` // atribui o endereço do x para o ponteiro pont[0]
- `pont[1] = y;` // atribui o endereço do y para o ponteiro pont[1]
- `pont[2] = z;` // atribui o endereço do z para o ponteiro pont[2]
- `pont[3] = w;` // atribui o endereço do w para o ponteiro pont[3]



Ponteiros e funções

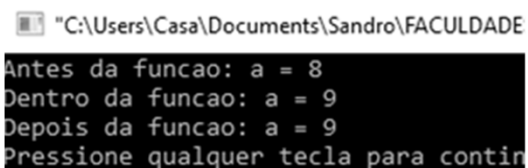
- Quando se define uma variável como ponteiro, dizemos que o endereço de uma variável simples está guardado em um ponteiro, o qual pode ser passado como parâmetro para uma função

- Para que isso ocorra, basta colocar o operador `*` antes da variável e o operador `&` na chamada do parâmetro
- O operador unário `&` retorna o endereço na memória de seu operando

```
void soma_mais_1(int *num){ //pega o end. do parâmetro a
    *num = *num + 1;
    printf("Dentro da funcao: a = %d\n", *num);
}

int main()
{
    int a = 8;
    printf("Antes da funcao: a = %d\n",a); // antes da função
    soma_mais_1(&a); // a função recebe o endereço de a

    printf("Depois da funcao: a = %d\n",a); // depois da função
    system("pause");
    return 0;
}
```



```
"C:\Users\Casa\Documents\Sandro\FACULDADE
Antes da funcao: a = 8
Dentro da funcao: a = 9
Depois da funcao: a = 9
Pressione qualquer tecla para contin
```

- Esses efeitos não ocorrem quando os parâmetros são passados por valor (sem o uso do asterisco `*` e do operador `&`), situação em que uma cópia do dado é passada como parâmetro, para a função, e a variável origem (na memória) não sofre nenhuma alteração

