

# Algoritmos e programação

**Prof. Dr. Josenalde Barbosa de Oliveira**


josenalde@eaj.ufrn.br

Aulas: 24T345 – 6 CRDS – 90h

# Tipos de variável PONTEIRO

- Armazena endereços de memória (base comum: HEXADECIMAL)
- Uma variável ao ser criada ocupa espaço na memória (tem um endereço e espaço **alocado** para ela); pode ter seu conteúdo acessado pelo seu NOME (referência) ou por uma outra variável, denominada PONTEIRO.

```
int y, x;  
y = 10, x = 20;  
int *ptr = &x;  
// acesso ao valor de x  
cout << ptr; // endereço de x  
cout << *ptr; // conteúdo da variável apontada por ptr
```



# (HEX)	Nome	Conteúdo
250	ptr	230
222	x	20
226	y	10

# Tipos de variável PONTEIRO

- A compreensão do uso de ponteiros e endereços de memória auxilia no estudo de VETORES, MATRIZES, STRINGS e FUNÇÕES, visto que é possível as funções receberem endereços de memória como parâmetros e retornarem endereços de memória.
- Deve-se ter mente que na aritmética com ponteiros, uma operação de deslocamento para frente ou para trás na memória em X unidades, significa andar  $X \times (\text{tamanho do dado})$  BYTES na memória, daí a importância do tamanho em bytes que cada tipo de dado usa:

```
int y, x;  
y = 10, x = 20; // neste caso y e x ocupam 4 bytes cada  
char z = 97; // aqui z ocupa 1 byte  
short p; // p ocupa 2 bytes  
double w = 20.2; // w ocupa 8 bytes  
int e = 12;
```

# (HEX)	Nome	Conteúdo
207	e	12
215	w	20.2
217	p	?
218	z	97
222	x	20
226	y	10

```
int *px = &x;  
int *py = &y;  
char *pz = &z;  
short *pp = &p;  
double *pw = &w;
```

# Tipos de variável PONTEIRO

```
cout << *py; // exhibe 10  
cout << *(py - 1); // - 1*4 = *(&py - 4) = *(&226-4)=*(&222)
```

Logo, é possível se deslocar por endereços de memória, usando ponteiros

O asterisco (\*) é utilizado para acessar o conteúdo

De modo geral, um ponteiro só pode receber o endereço de memória de uma variável do mesmo tipo do ponteiro

py-2

py-1

py

# (HEX)	Nome	Conteúdo
207	e	12
215	w	20.2
217	p	?
218	z	97
222	x	20
226	y	10

Um ponteiro é criado para referenciar um endereço. Se não for associado à endereço, possui qualquer valor, e não é recomendável. No máximo pode-se fazer:

```
<tipoPonteiro> *<nome ponteiro> = NULL // <cstdlib>  
ou <tipoPonteiro> *<nomePonteiro> = 0.
```

# Endereços atribuídos diretamente

```
int *p = 0x3F8;  
int *p1 = 1500; (valor decimal é convertido para HEX: 0x5DC)
```

Na prática isto pode acarretar problemas, pois o que há no endereço 5DC? Mas em aplicações de sistemas embarcados, com microcontroladores pode ser interessante.

**Devemos ter atenção à diferença entre:**

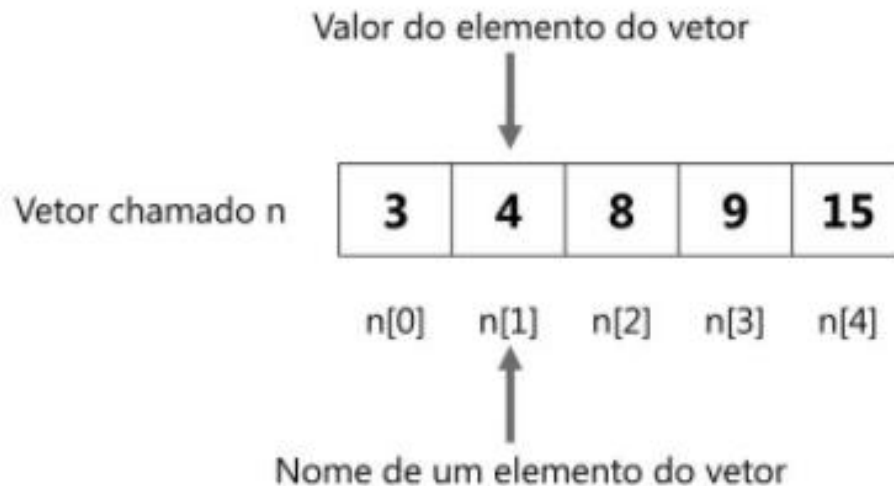
```
p = p + 10 // avançar dez posições na memória  
*p = (*p) + 10 // adicionará à variável apontada o valor 10
```

**Os operadores de comparação podem ser usados para PONTEIROS**

```
int *p, *p1, x, y;  
p = &x;  
p1 = &y;  
if (p==p1) cout << "ponteiros iguais";  
if (p>p1) cout << "o ponteiro p aponta para uma posição a frente de p1"
```

# Ponteiros e vetores

- 1) O NOME DE UM VETOR É APENAS UM PONTEIRO QUE APONTA PARA O PRIMEIRO ELEMENTO DO VETOR (ENDEREÇO BASE)
- 2) LOGO, O NOME DO VETOR SEM ÍNDICE GUARDA O ENDEREÇO PARA O COMEÇO DO VETOR NA MEMÓRIA. AS DEMAIS POSIÇÕES SÃO ADJACENTES



- 3) Portanto, `n` é o nome do vetor e o endereço do seu elemento `&n[0]`

# Ponteiros e vetores

Usando índices	Usando ponteiros
<pre>int v[5] = {1,2,3,4,5}; int *p = v; int i; for (i=0;i&lt;5;i++)     cout &lt;&lt; p[i]; // ou v[i]</pre>	<pre>int v[5] = {1,2,3,4,5}; int *p = v; int i; for (i=0;i&lt;5;i++)     cout &lt;&lt; *(p+i);</pre>

Uma **string** (sequência de caracteres) pode ser visto como um vetor do tipo char terminado com um caracter especial `'/0'` que indica o fim do texto e, portanto, o nome desta string 'apontaria' para seu caracter inicial

```
char str1[] = "teste com char";
char str3[5] = {'t','a','d','s','\0'}; // '\0' não imprimível
char *str2;
str2 = str1;
std::cout << str1 << "\n";
std::cout << str2 << "\n";
std::cout << str3 << "\n";
```

# Ponteiros e strings

Em <cstring> existem funções para manipular strings no estilo C

```
char str1[30];  
std::cin >> str1; // lê texto apenas até o primeiro espaço!!  
std::cout << str1 << "\n";  
  
std::cin.get(str1,30); // lê os espaços também, sem o '\n'  
std::cout << str1 << strlen(str1) << "\n";
```

```
char str1[30];  
char str2[10];  
std::cin.get(str1,30);  
std::cout << str1 << strlen(str1) << "\n";  
fgets(str2, 10, stdin); // não irá executar esta linha, pois leu o  
                        // caracter de nova linha anterior  
std::cout << str2 << strlen(str2) << "\n"; // retorna tamanho 1
```

```
std::cin.get(str1,30);  
std::cout << str1 << strlen(str1) << "\n";  
setbuf(stdin, NULL); // limpa o buffer do console!!!  
fgets(str2, 10, stdin); // lê 9 caracteres + o '\n'  
std::cout << str2 << strlen(str2) << "\n";
```



# Ponteiros e strings

Com strings definidas como vetores char, funções de manipulação estão <cstring>, como tamanho (strlen), cópia (strcpy, strncpy), concatenação (strcat), comparação (strcmp)

Em C++, o tipo `string` facilita o uso, pois possui métodos intuitivos e diretos para manipulação

```
string str1;  
string str2 = " tads eaj";  
getline(cin, str1);  
cout << str1 + str2 << endl;  
//string str3 = str1.append(str2);  
cout << str1.size() << str2.size() << endl;
```

```
string str1("oi");  
string str2 = "tads eaj";  
auto f = str2.find(str1); // std::size_t  
if (f != string::npos) // se não encontra, retorna string::npos  
    cout << f << std::endl;  
else  
    cout << "not found";
```

# Ponteiros e strings

```
string str1;  
string senha = "tads";  
getline(cin, str1);  
if (str1.compare(senha)==0) cout << "OK";  
else cout << "NOT OK";
```

```
const char str1[20] = "tads eaj";  
const char str2[10] = "eaj";  
char *ret;  
ret = strstr(str1, str2); // retorna endereço do primeiro caracter da  
ocorrência de str2 em str1  
cout << "Substring:" << ret;
```

# Ponteiros genéricos

```
void *pp;  
int *p1, p2 = 10;  
p1 = &p2;  
pp = &p2;  
  
cout << pp; // OK, pois é genérico  
cout << *pp; // ERRO, pois é void e aponta para int  
cout << *(int *)pp; // OK, pois converte void * para int *
```