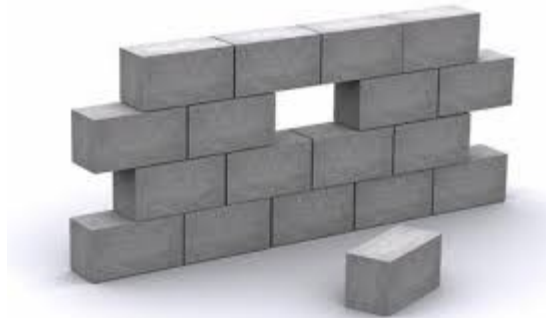




Instituto Federal
Campus Goiânia

Bacharelado em Sistemas de Informação

Estrutura de Dados I

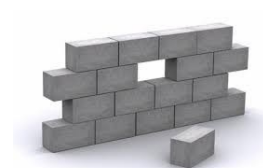


Prof. Dory Gonzaga Rodrigues



Agenda

Ponteiros





Ponteiro

Tipos de Dados em C

A linguagem C contém quatro tipos básicos de dados:

integer

float

char

double

Na maioria dos computadores, esses quatro tipos são nativos no hardware da máquina.

Variáveis em C

Podemos criar Variáveis e Ponteiros em C.

Qual seria a diferença ?





Ponteiro

Diferença entre Variável e Ponteiro ?

- Variáveis apontam para o conteúdo armazenado na memória. Podemos dizer que uma variável é um ponteiro com endereço de memória fixo e, neste caso, indica/aponta para o valor armazenado na memória.

- Ponteiros são as variáveis que apontam para um endereço de memória. Ou seja, indicam endereços de memória onde valores estarão armazenados. Em C, os ponteiros frequentemente fornecem técnicas muito efetivas para representar e manipular dados, especialmente estruturas de dados.





Ponteiro

Declarando Ponteiros

Pode-se declarar um ponteiro para representar(apontar) o endereço de qualquer tipo de dado em C.

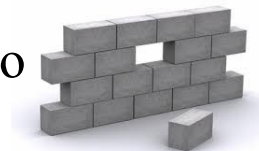
Sintaxe:

```
tipo *nome_do_ponteirocoleção;
```

Exemplo:

```
int x;  
int *idade;
```

Obs: com o uso do * o compilador sabe que idade é ponteiro
idade aponta para uma memória que poderá ter um número inteiro





Ponteiro

Entenda os Operadores * e &

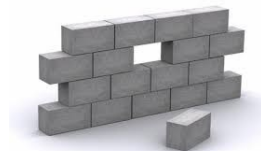
Existem dois operadores especiais para trabalhar com variáveis e ponteiros:

Operador &

Operador *

Operador & - é um operador unário que devolve o endereço da memória do seu operando.

Operador * - é um operador unário que devolve o conteúdo contido no endereço da memória do operando (ponteiro).





Ponteiro

Entenda os Operadores * e &

Qual o resultado do código abaixo:

```
int x;  
int *idade;
```

declarando um ponteiro

```
idade = &x;
```

passa para o ponteiro “idade” o endereço da memória da variável “x”

```
printf("&x: %p", &x);
```

imprime o endereço da memória de “x”

```
printf("idade: %p", idade);
```

imprime o endereço da memória de “idade”





Ponteiro

Entenda os Operadores * e &

O operador * quando aplicado sobre um ponteiro retorna o dado contido na memória referenciada:

```
int x, y;  
int *p;  
x = 10;  
p = &x;  
y = *p;  
print("x: %d", x);  
print("p: %p", p);  
print("*p: %d", *p);  
print("y: %d", y);
```

imprime o endereço da memória de "p"

imprime o conteúdo da memória de "p"









Ponteiro

Ponteiros são variáveis tipadas

(int *) # (float *) # (char *)

Ex:

```
int *p1, x;  
float *p2, y;
```

```
p1 = &x;   
p2 = &y;   
p1 = &y;   
p2 = &x; 
```

“p1” e “p2” estão recebendo endereços de memórias de “x” e “y” respectivamente.

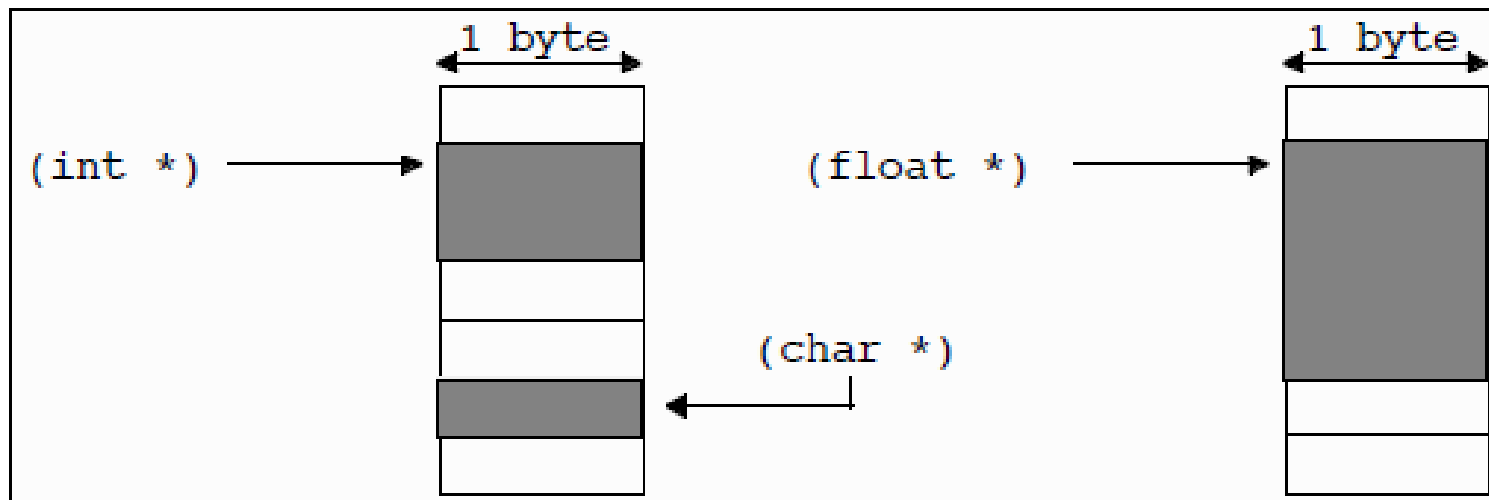
o importante é observar que
“p1” é do mesmo tipo de “x” e “p2” é do mesmo tipo de “y”





Ponteiro

Espaço ocupado pelas Variáveis





Ponteiro

O operador ++ com uso em ponteiros

```
int x;    int *p;
```

```
x = 10;   p = &x;
```

```
(*p)++;
```

← incrementa 1 no **conteúdo** do ponteiro “p”

```
print(“x: %d \n”, x);
```

```
print(“*p: %d \n”, *p);
```

```
p++;
```

← vai para o **próximo endereço de memória** a partir de “p”

```
print(“p: %p \n”, p);
```

```
print(“&x: %p \n”, &x);
```





Ponteiro

Qual o resultado do código abaixo:

```
int x;  
int *p1, *p2;
```

```
p1 = &x;  
x = 10;
```

```
(*p1)++;
```

```
p2 = p1;
```

```
(*p2)++;
```

```
print("x: %d", x);
```





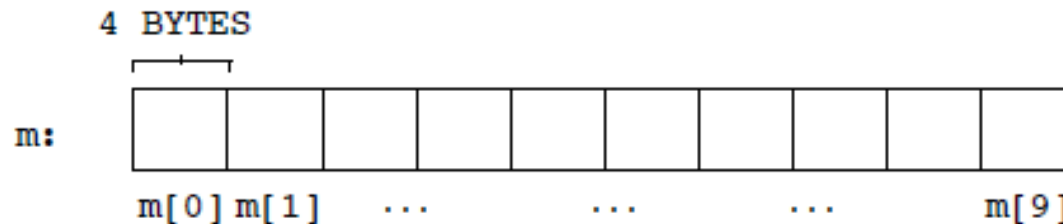
Ponteiro

Trabalhando Ponteiros com Arrays

Arrays: é uma estrutura de dados formada por um agrupamento de memórias adjacentes (uma ao lado da outra ou uma após a outra)

Ex:

```
int m[10];  
int *p;
```





Ponteiro

Trabalhando Ponteiros com Arrays

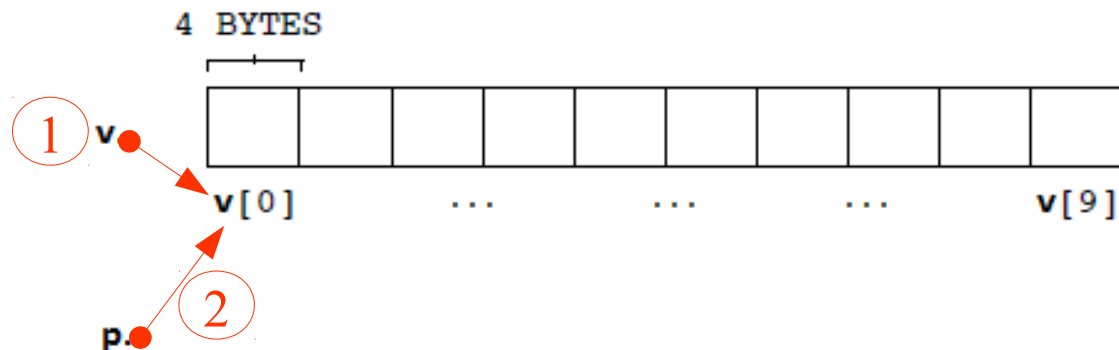
Entenda os dois comandos abaixo:

① `int v[5];`
`int *p;`

declarando um ponteiro

② `p = &v;`

passa para o ponteiro “p” o endereço da memória do vetor “v”





Ponteiro

Trabalhando Ponteiros com Arrays

A variável “*m*” armazena o endereço do primeiro elemento do array, não sendo possível mudar o endereço da variável “*m*” após o início da execução do programa.

Ex:

```
int  m[10] , v[10];  
int  *p;
```

m = *v*; 

o comando vai dar erro !
vetor “*m*” receber o endereço do vetor “*v*”

p = *v*; 

o comando realiza com sucesso:
o ponteiro “*p*” recebe o endereço do vetor “*v*”





Ponteiro

Trabalhando Ponteiros com Arrays

Qual o resultado do código abaixo:

```
int v[] = { 1, 10, 7, 9, 5 };  
int *p;
```

```
p = &v[2];
```

passa para o ponteiro “p” o endereço da memória do vetor “v[2]”

```
printf(“p: %d ”, *p);
```

imprime o conteúdo do ponteiro “p” que é o valor 7

```
printf(“p[1]: %d ”, p[1]);
```

imprime o conteúdo do ponteiro “p” na primeira posição após o endereço de “p” que é o valor 9





Ponteiro

Trabalhando Ponteiros com Arrays

```
int v[] = { 1, 10, 7, 9, 5 };  
int *p;
```

```
p = &v[2];
```

```
printf("p: %d ", *p);
```

```
printf("p[1]: %d ", p[1]);
```



Note que o valor entre colchetes e o deslocamento a ser considerado a partir do endereço de referencia.

$p[n] \Rightarrow$ indica enésimo elemento a partir de “p”





Ponteiro

Qual o resultado do código abaixo:

```
int v[] = { 1, 10, 7, 9, 5 };  
int *p;
```

```
p = &v[2];
```

passa para o ponteiro “p” o endereço da memória do vetor “v[2]”

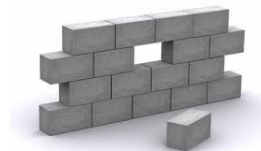
```
printf(“p: %d ”, *p);
```

imprime o conteúdo do ponteiro “p” que é o valor 7

```
printf(“p[1]: %d ”, p[1]);
```

imprime o conteúdo do ponteiro “p” na primeira posição após o endereço de “p” que é o valor 9

$p[1]$ equivale $*(p+1)$





Ponteiro

Atividade

1) Faça um programa com um vetor de 5 posições de tipo float. Também de um ponteiro “p” de mesmo tipo. Solicite o preenchimento do vetor ao usuário utilizando a variável do vetor. Após o preenchimento do vetor, aponte o ponteiro “p” para a primeira posição do vetor. Em seguida, utilizando o ponteiro, percorra e imprima todo o conteúdo do vetor.

2) Faça o mesmo exercício anterior, porém para uma matriz 3x3.

