

Estrutura de Dados

Ponteiros

Prof. Luiz Gustavo Almeida Martins

Ponteiros

- **Tipo especial de dados** usado para guardar o **endereço de memória**
 - Indica o local da memória onde existe um dado do tipo declarado
 - Pode ser inicializado com o endereço de uma variável ou **NULL**
 - Ponteiro é uma variável e também ocupa espaço de memória
 - Sistema de 32 bits ocupa 4 bytes (mesmo que *unsigned int*)
 - Sistema de 64 bits ocupa 8 bytes (mesmo que *long unsigned int*)
- **Sintaxe de declaração de um ponteiro:**
*tipo_dado * nome_variavel; // Declaração de uma variável ponteiro*

Ponteiros

- **Tipo especial de dados** usado para guardar o **endereço de memória**
 - Indica o local da memória onde existe um dado do tipo declarado
 - Pode ser inicializado com o endereço de uma variável ou **NULL**.
 - Ponteiro é uma variável e também ocupa espaço de memória
 - Sistema de 32 bits ocupa 4 bytes (mesmo que *unsigned int*)
 - Sistema de 64 bits ocupa 8 bytes (mesmo que *long unsigned int*)
- **Sintaxe de declaração de um ponteiro :**

tipo_dado * *nome_variavel*; // Declaração de uma variável ponteiro



Define o tipo do dado que será armazenado na posição da memória indicada pelo ponteiro

Ponteiros

- **Exemplos:**

// Guarda o endereço da memória onde está um caractere
*char *y;*

// Pode ser declarado com outras variáveis
*float w, *z;*

// Pode ser inicializado com o end. memória onde está outra variável
*double x, *q = &x;*

// Pode ser inicializado apontando para nenhum lugar (NULL = end. 0)
*int *p = NULL;*

Ponteiros

- Exemplos:

// Guarda o endereço da memória onde está um caractere
*char *y;*

// Pode ser declarado com outras variáveis
*float w, *z;*

// Pode ser inicializado com o end. memória
*double x, *q = &x;*

// Pode ser inicializado apontando para nê
*int *p = NULL;*

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	Indefinido	----	----
1			
2			

28			
29			
30	NULL	p	*int
31			
32			
33			
34			

Ponteiros

- **Exemplos:** Considerando uma **sistema de 64 bits**

// Guarda o endereço da memória onde está um caractere

*char *y;* **y = 8 bytes** área referenciada = **1 byte** (*sizeof(char)*)

// Pode ser declarado com outras variáveis

*float w, *z;* **z = 8 bytes** área referenciada = **4 bytes** (*sizeof(float)*)

// Pode ser inicializado com o end. memória onde está outra variável

*double x, *q = &x;* **q = 8 bytes**
 área referenciada = **8 bytes** (*sizeof(double)*)

// Pode ser inicializado apontando para nenhum lugar (NULL = end. 0)

*int *p = NULL;* **p = 8 bytes**
 área referenciada = **4 bytes** (*sizeof(int)*)

Tipo do dado endereçado **NÃO afeta o tamanho do ponteiro**, apenas da área referenciada. Tamanho está relacionado com o tamanho do barramento de endereço do sistema

Operadores para Ponteiros

- **Operador ***
 - Possui 2 funcionalidades de acordo com sua utilização:
 - **Na declaração de variáveis:** define que a variável é do tipo ponteiro
 - **Nos demais comandos: **acessa o conteúdo**** da memória endereçado por uma variável ponteiro
 - Operador de “desreferenciamento” do ponteiro (**dereferencing**)
- **Operador &**
 - **Retorna o endereço da memória** onde está armazenado uma determinada variável

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

```
int *p;
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %lu", p);
```

```
printf("\n O endereço da var. 'p' eh: %lu", &p);
```


Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;    // cria uma variável do tipo inteiro, chamada a, e
               // inicializa com valor 40
```

```
int *p;
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %u", p);
```

```
printf("\n O endereço da var. 'p' eh: %d", &p);
```

45			
46			
47		a	int
48	40		
49			
50			
51			
52			
53			
54			
55			
56			

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

```
int *p;           // cria uma variável do tipo ponteiro para inteiro,  
                  // chamada p, cujo o conteúdo inicial é lixo
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %u", p);
```

```
printf("\n O endereço da var. 'p' eh: %d", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	lx	p	int *
52			
53			
54			
55			
56			

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

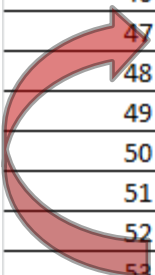
```
int *p;
```

```
p = &a;           // p recebe o endereço de a (dizemos: p aponta para a)
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %u", p);
```

```
printf("\n O endereço da var. 'p' eh: %d", &p);
```



45			
46			
47		a	int
48	40		
49			
50			
51		p	int *
52	47		
53			
54			
55			
56			

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

```
int *p;
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

Saída: O valor da variavel 'a' eh: 40

```
printf("\n O valor da variavel 'p' eh: %lu", p);
```

```
printf("\n O endereço da var. 'p' eh: %lu", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

```
int *p;
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %lu", p);
```

Saída: O valor da variavel 'p' eh: 47

```
printf("\n O endereço da var. 'p' eh: %lu", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Exemplo 1

Slide adaptado do material da Profa.
Gina M. B. de Oliveira

- Considere o código abaixo executado em um sistema de 32 bits (4 bytes):

```
int a = 40;
```

```
int *p;
```

```
p = &a;
```

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %lu", p);
```

```
printf("\n O endereço da var. 'p' eh: %lu", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Saída: O endereço da var. 'p' eh: 51

Exemplo 2

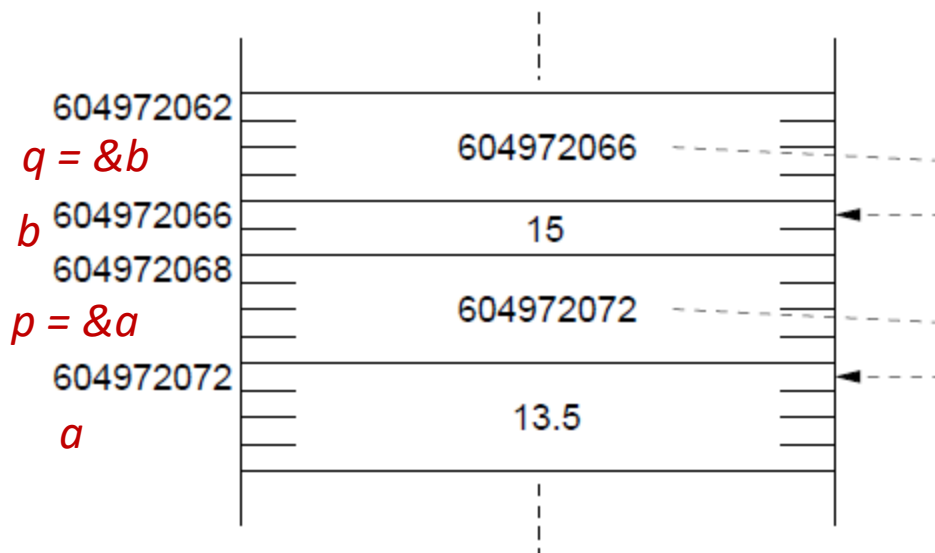
Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```

Exemplo 2

Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```



Mapa de memória do código

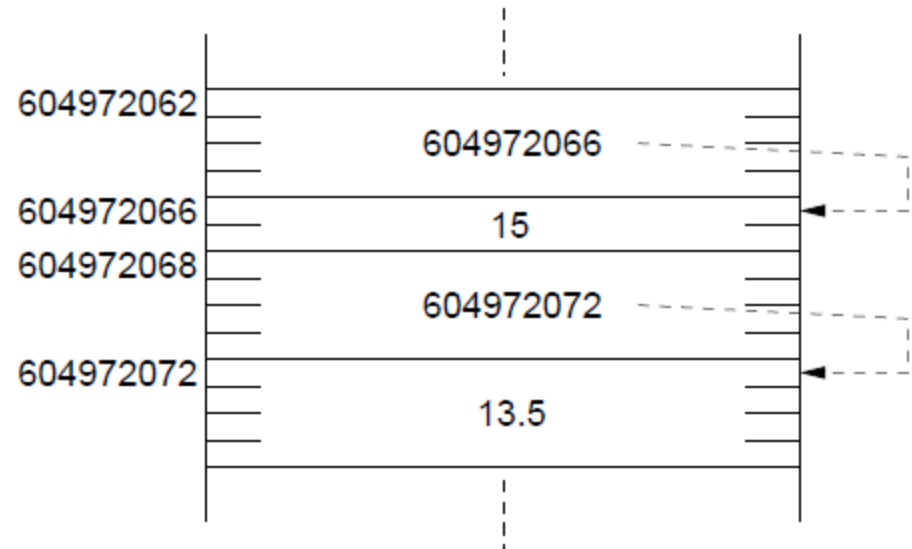
Exemplo 2

Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```

Saída esperada:

a = 13.5; b = 15



Mapa de memória do código

Exemplo 2

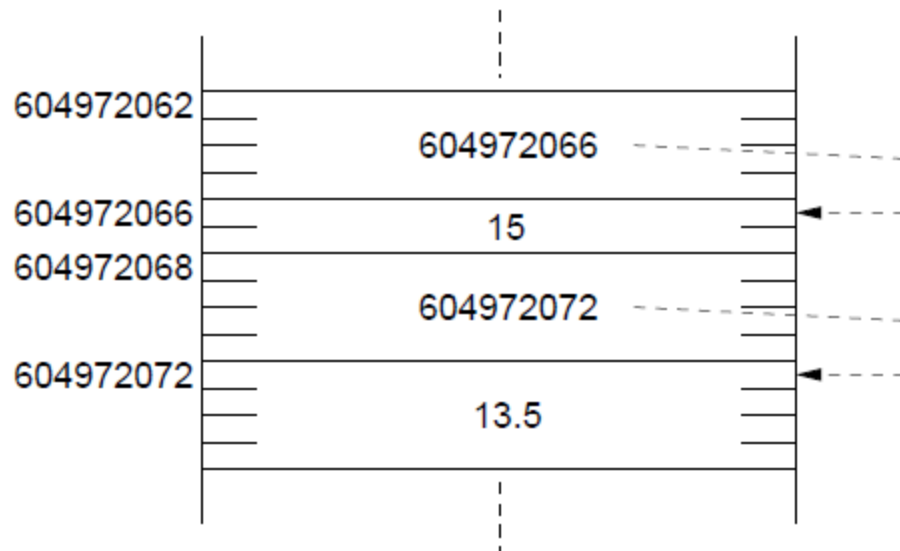
Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```

Saída esperada:

a = 13.5; b = 15

p=&a=604972072; q=&b=604972066



Mapa de memória do código

Exemplo 2

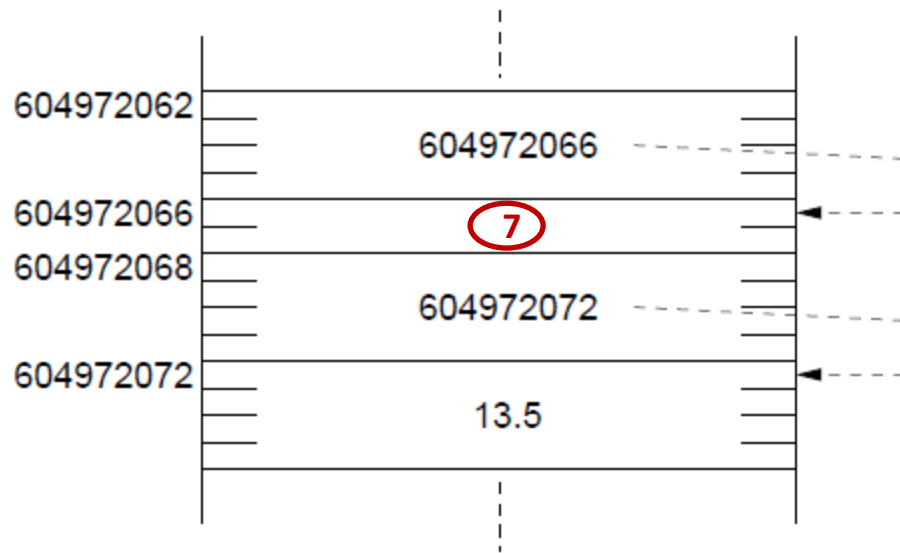
Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```

Saída esperada:

a = 13.5; b = 15

p=&a=604972072; q=&b=604972066



Mapa de memória do código

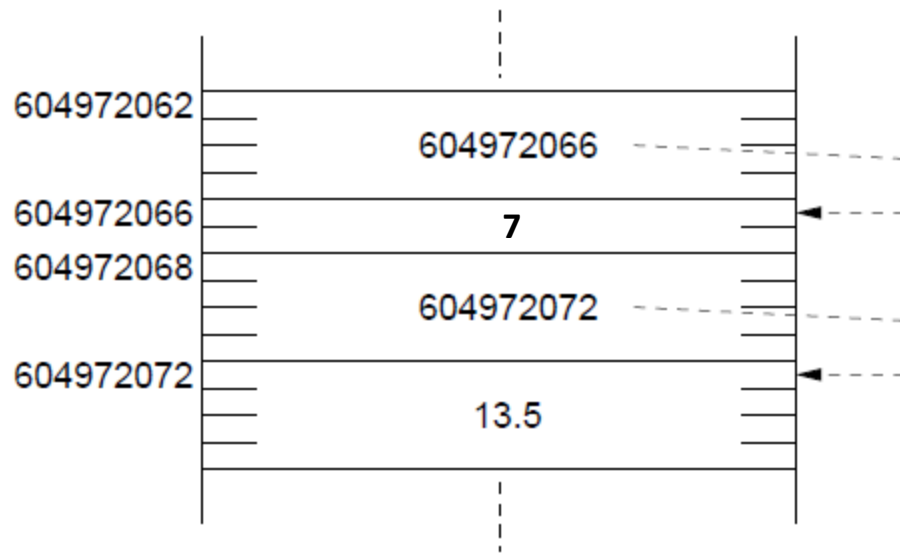
Exemplo 2

Slide adaptado do material do Prof.
Paulo Coelho

```
#include <stdio.h>
int main () {
    float a = 13.5, *p;
    int b = 15, *q;
    p = &a; q = &b; // & - retorna o end. de 1 variável na memória
    printf("a=%f; b=%d;\n", a, b);
    printf("p=&a=%ld; q=&b=%ld\n", p, q); // ld = long int
    *q = 7; // acessa o conteúdo da memória endereçado por q
    printf("a=%f; b=%d;\n", a, b);
    return 0;
}
```

Saída esperada:

```
a = 13.5; b = 15
p=&a=604972072; q=&b=604972066
a = 13.5; b = 7
```



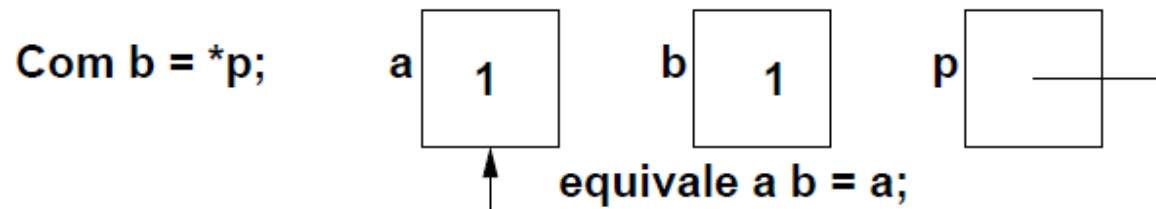
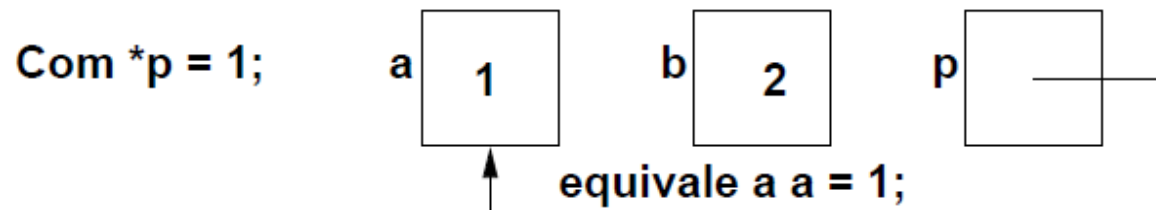
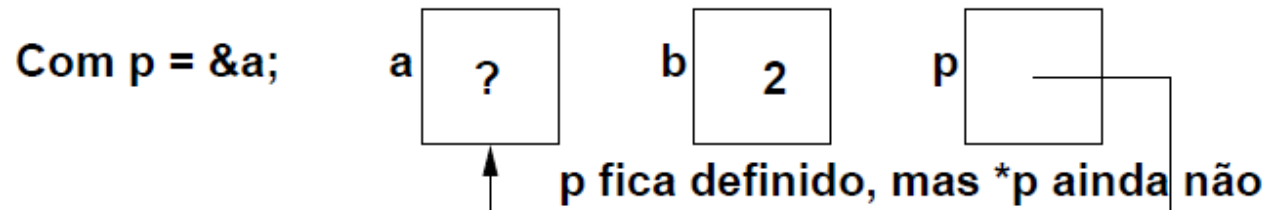
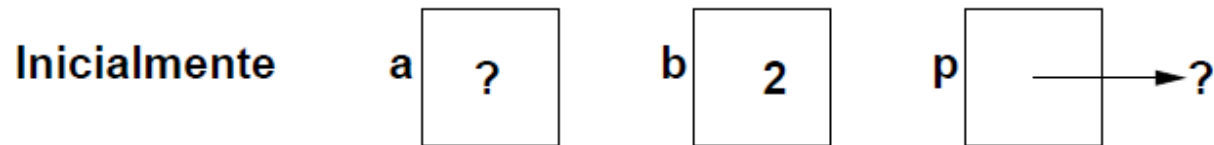
Mapa de memória do código

Exemplo 3

Slide retirado do material do Prof.
Paulo Coelho

- Considere o seguinte fragmento de código:

```
int a, b = 2, *p;  
p = &a; *p = 1; b = *p;
```

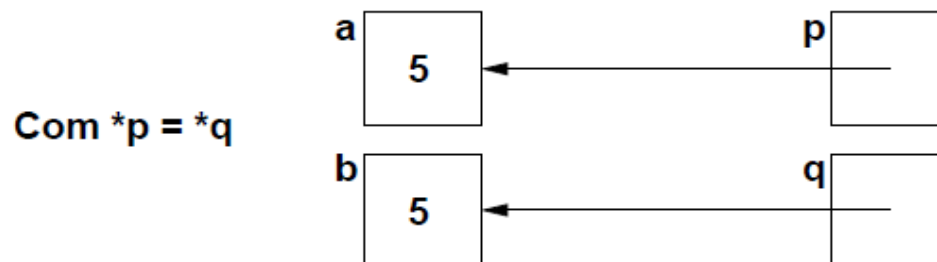
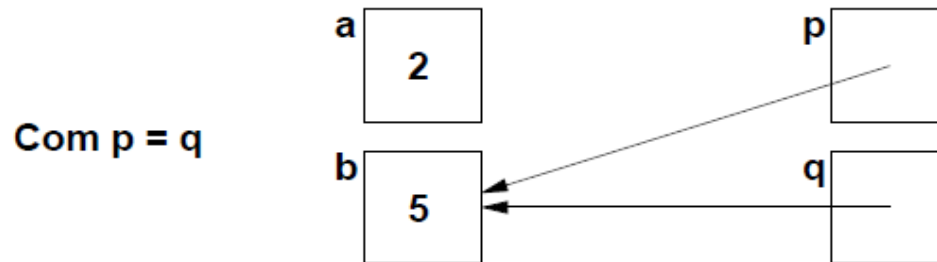
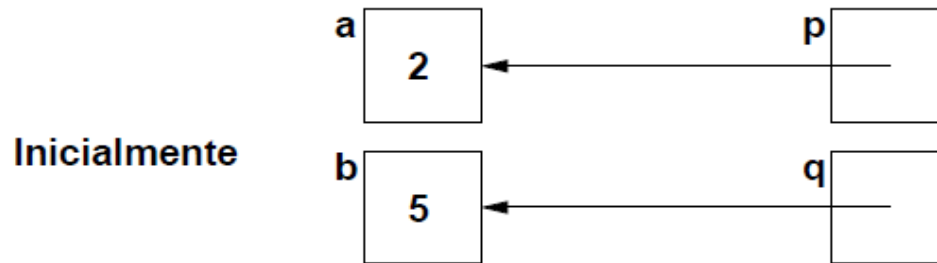


Exemplo 4

Slide retirado do material do Prof.
Paulo Coelho

- Seja a seguinte declaração e considerando as duas atribuições separadamente:

```
int a = 2, b = 5, *p = &a, *q = &b;  
p = q;      // atribuicao 1 ou  
*p = *q;    // atribuicao 2
```



Questões

Considere o trecho de código abaixo:

```
float *preco;  
*preco = 50.0;
```

- Ele funciona?

Questões

Considere o trecho de código abaixo:

```
float *preco;  
*preco = 50.0;
```

- Ele funciona? **Pode funcionar**
 - Se endereço for válido (estiver na área do programa)
 - Caso contrário, ocorre falha de segmentação (*segmentation fault*)
- Está correto?

Questões

Considere o trecho de código abaixo:

```
float *preco;  
*preco = 50.0;
```

- Ele funciona? **Pode funcionar**
 - Se endereço for válido (estiver na área do programa)
 - Caso contrário, ocorre falha de segmentação (*segmentation fault*)
- Está correto? **Não**
 - Pode travar ou apresentar comportamento indesejado

Questões

Considere o trecho de código abaixo:

```
float *preco;
```

```
*preco = 50.0;
```

- Exemplo do funcionamento:
 - Suponha que o lixo da memória seja igual a **13**

67			
68			
69		preco	*double
70	13		
71			
72			
73			
74			

11			
12			
13		k	float
14	2.4		
15			
16			
17		pval	*double
18	5		
19			
20			
21		pk	*float
22	13		
23			
24			
25			

Questões

Considere o trecho de código abaixo:

```
float *preco;
```

```
*preco = 50.0;
```

- Exemplo do funcionamento:
 - Suponha que o lixo da memória seja igual a **13**

67			
68			
69		preco	*double
70	13		
71			
72			
73			
74			

11			
12			
13		k	float
14	50.0		
15			
16			
17		pval	*double
18	5		
19			
20			
21		pk	*float
22	13		
23			
24			
25			

Aritmética de Ponteiros

- São expressões que operam sobre endereços de memória
 - Envolvem apenas **soma e subtração de inteiros**
 - Outras operações não são permitidas (ex: multiplicação e divisão)
 - Não permite operações com dois ponteiros
 - Nem somar ou subtrair float ou double de ponteiros
 - O resultado é **sempre um endereço**
 - Deslocamento depende do tipo de dado do ponteiro
- p+i*** apontará para ***(i * sizeof(tipo do dado))*** bytes a partir do endereço indicado por ***p***

Exemplo

Código disponibilizado no site
Linguagem C Descomplicada

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

```
int main () {
```

```
    // Considerando que um inteiro ocupa 32 bit = 4 bytes
```

```
    int *p = 0x5DC;
```

```
    printf ("p = %d\n", p); // p = 0x5DC (hexa) = 1500 (decimal)
```

```
    p++; // avança um inteiro (4 bytes)
```

```
    printf ("p = %d\n", p); // p = 0x5E0 (hexa) = 1504 (decimal)
```

```
    p = p + 15; // avança 15 inteiros (4x15 = 60 bytes)
```

```
    printf ("p = %d\n", p); // p = 0x61C (hexa) = 1564 (decimal)
```

```
    p = p - 2; // retrocede 2 inteiros (4x2 = 8 bytes)
```

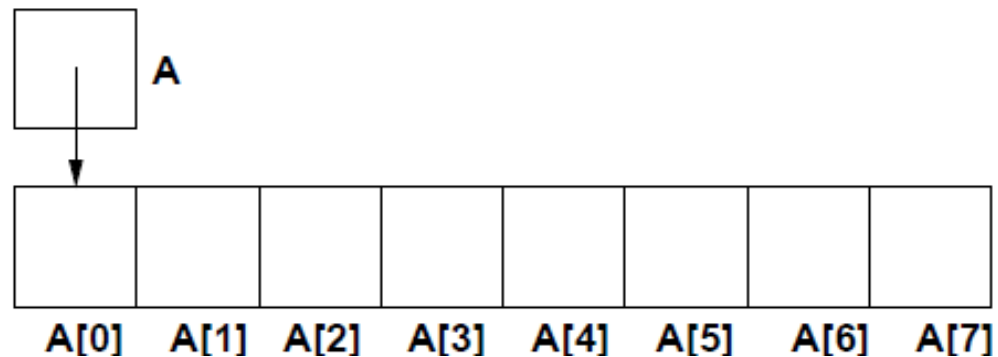
```
    printf ("p = %d\n", p); // p = 0x614 (hexa) = 1556 (decimal)
```

```
}
```

Variáveis Indexadas e Ponteiros

- Existe uma forte relação entre estes tipos de dados
 - Variável indexada aloca espaço na memória (**posições consecutivas**) para os ***N*** elementos
 - O nome da **variável sem o índice referencia o array**
 - Aponta para um local fixo que não pode ser mudado
 - Endereça o **1o elemento do vetor**

Ex: `int A[8];`



A+i endereça o ***(i+1)***-ésimo elemento de ***A***, ou seja, ****(A+i) ⇔ A[i]***

Exemplo 1

```
int vet[5] = {1,2,3,4,5};
```

```
int *p = vet;  
printf("%d\n", *p);  
printf("%d\n", *(p+2));
```

```
p = &vet[2];  
printf("%d\n", *p);
```

<i>Pos</i>			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22			
...			

Exemplo 1

vet 

Pos			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22	2	p	int *
...			

```
int vet[5] = {1,2,3,4,5};
```

```
int *p = vet; // deve usar o mesmo tipo de vet
```

```
printf("%d\n", *p);
```

```
printf("%d\n", *(p+2));
```

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

```
printf("%d\n", *p);
```


Exemplo 1

```
int vet[5] = {1,2,3,4,5};
```

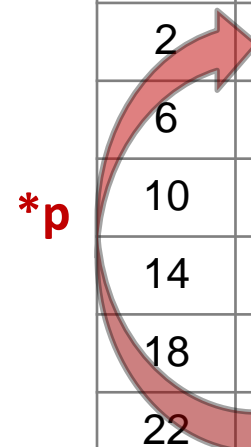
```
int *p = vet;
```

```
printf("%d\n", *p);
```

```
printf("%d\n", *(p+2));
```

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

```
printf("%d\n", *p);
```



Pos			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22	2	p	int *
...			

```
// Retorna vet[0] = 1
```

Exemplo 1

```
int vet[5] = {1,2,3,4,5};
```

```
int *p = vet;
```

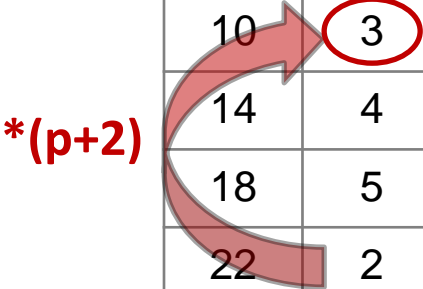
```
printf("%d\n", *p);
```

```
printf("%d\n", *(p+2));    // Retorna vet[2] = 3
```

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

```
printf("%d\n", *p);
```

Pos			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22	2	p	int *
...			



Exemplo 1

&(vet[2]) 

Pos			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22	10	p	int *
...			

```
int vet[5] = {1,2,3,4,5};
```

```
int *p = vet;
```

```
printf("%d\n", *p);
```

```
printf("%d\n", *(p+2));
```

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

```
printf("%d\n", *p);
```

Exemplo 1

```
int vet[5] = {1,2,3,4,5};
```

```
int *p = vet;  
printf("%d\n", *p);  
printf("%d\n", *(p+2));
```


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

```
printf("%d\n", *p);
```

// Retorna vet[2] = 3

***p**

Pos			
2	1	vet[0]	int
6	2	vet[1]	int
10	3	vet[2]	int
14	4	vet[3]	int
18	5	vet[4]	int
22	10	p	int *
...			



Exemplo 2

Retirado do material da Profa. Gina M. B. de Oliveira

- Considere o trecho de código:

```
7  int i;  
8  int a = 5;  
9  int b = 10;  
10 char c[5] = {'A', 'b', '8', 'd', '|'};  
11  
12 printf("Valor de a: %d \n", a);  
13 printf("Endereco de a: %d \n", &a);  
14 printf("Endereco de a (em hexadecimal): %p \n\n", &a);  
15  
16 printf("Valor de b: %d \n", b);  
17 printf("Endereco de b: %d \n", &b);  
18 printf("Endereco de b (em hexadecimal): %p \n\n", &b);  
19  
20 for (i=0; i < 5; i++){  
21     printf("Valor de c[%d]: %c \n", i, *(c+i)); // *(c+i) ==> c[i]  
22     printf("Endereco de c[%d]: %d \n", i, c+i); // c+i ==> &c[i]  
23     printf("Endereco de c[%d] (em hexadecimal): %p \n\n", i, &c[i]);  
24 }
```

```
Valor de a: 5  
Endereco de a: 2686744  
Endereco de a (em hexadecimal): 0028FF18  
  
Valor de b: 10  
Endereco de b: 2686740  
Endereco de b (em hexadecimal): 0028FF14  
  
Valor de c[0]: A  
Endereco de c[0]: 2686735  
Endereco de c[0] (em hexadecimal): 0028FF0F  
  
Valor de c[1]: b  
Endereco de c[1]: 2686736  
Endereco de c[1] (em hexadecimal): 0028FF10  
  
Valor de c[2]: 8  
Endereco de c[2]: 2686737  
Endereco de c[2] (em hexadecimal): 0028FF11  
  
Valor de c[3]: d  
Endereco de c[3]: 2686738  
Endereco de c[3] (em hexadecimal): 0028FF12  
  
Valor de c[4]: |  
Endereco de c[4]: 2686739  
Endereco de c[4] (em hexadecimal): 0028FF13
```

Exemplo 3

Retirado do material da
Profa. Gina M. B. de Oliveira

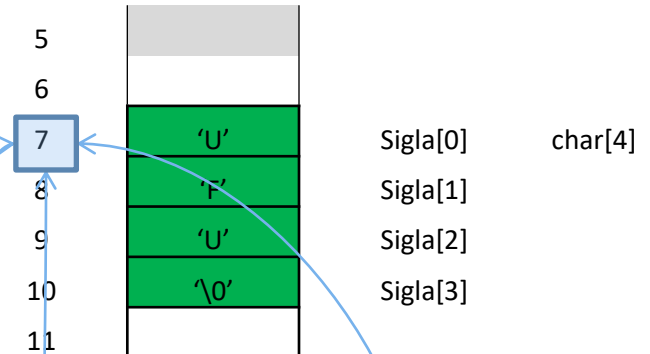
- Considere o trecho de código:

```
char Sigla[4] = "UFU";
```

```
// mostrando o endereço da posição 0 do vetor  
printf("\n Posicao de indice zero do vetor (Sigla[0]): %u", &Sigla[0]);
```

```
// mostrando o endereço da posição 0 do vetor  
printf("\n Nome do vetor (Sigla): %u", Sigla);
```

```
// mostrando o endereço da posição 0 do vetor  
printf("\n Endereço do vetor (&Sigla): %u", &Sigla);
```



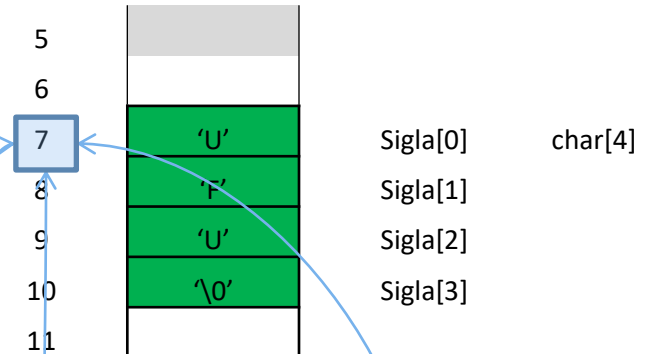
Exemplo 3

Retirado do material da
Profa. Gina M. B. de Oliveira

- Considere o trecho de código:

```
D:\Dropbox\Aulas\2014-01\ipc\projetos\memoria\ArraysPonteiros\bin\Debu
Posicao de indice zero do vetor <Sigla[0]>: 2686748
Nome do vetor <Sigla>: 2686748
Endereco do vetor <&Sigla>: 2686748
Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

&Sigla ⇔ Sigla ⇔ &Sigla[0]



```
char Sigla[4] = "UFU";
```

```
// mostrando o endereço da posição 0 do vetor
printf("\n Posicao de indice zero do vetor (Sigla[0]): %u", &Sigla[0]);
```

```
// mostrando o endereço da posição 0 do vetor
printf("\n Nome do vetor (Sigla): %u", Sigla);
```

```
// mostrando o endereço da posição 0 do vetor
printf("\n Endereço do vetor (&Sigla): %u", &Sigla);
```

Ponteiro Genérico

- Pode **endereçar qualquer tipo de dado**
 - Tipo especial de ponteiro
 - Usado pelas funções de alocação dinâmica
 - Operações aritméticas consideram apenas **1 byte**
- **Sintaxe da declaração:**
***void** * nome_ponteiro;*
- Acesso ao conteúdo endereçado **exige conversão**
 - Necessário **para indicar o tipo e tamanho do dado** a ser recuperado

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

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

int main()
{
    int a = 10;
    double d = 30;

    void *p;

    // atribuindo o endereço de 'a' ao ponteiro void
    p = &a;

    // mostra o conteúdo do endereço apontado por p (no caso, a var. 'a')
    printf("Valor de a: %d", *p);
}
```



Resultado da compilação:
ERROR: invalid use of void expression

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

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

int main()
{
    int a = 10;
    double d = 30;

    void *p;

    // atribuindo o endereço de 'a' ao ponteiro void
    p = &a;

    // mostra o conteúdo do endereço apontado por p (no caso, a var. 'a')
    printf("Valor de a: %d", *(int *)p);
}
```



CORREÇÃO:

*uso do operador de type cast (int *)*

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

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

```
int main()
{
    int a = 10;
    double d = 30;

    void *p;

    // atribuindo o endereço de 'a' ao ponteiro void
    p = &a;

    printf("Valor de a: %d", *(int *)p);
}
```

(int *)

Converte de void * para int *

*

Desreferencia o ponteiro

p

Ponteiro para void

Ponteiro e Estruturas

- Existem 2 formas de acessar os campos de variáveis estruturadas (*structs*) endereçada por um ponteiro:
 - Sintaxe 1: ***(*ponteiro).campo***
 - Sintaxe 2: **ponteiro->campo**
- Ex:** dada uma estrutura endereçada pelo ponteiro ***p***, podemos acessar seus campos ***a*** e ***b*** como segue:

```
(*p).a = 15; (*p).b = 3.12;
```

// sintaxe 1

```
printf("a = %d, b = %f\n", p->a, p->b);
```

// sintaxe 2

(*p).x ⇔ p->x

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

```
struct aluno joao, *pa;  
pa = &joao;
```

```
joao.num_aluno = 10;  
joao.nota[0] = 10;  
joao.nota[1] = 4.4;  
joao.nota[2] = 7;  
joao.media = (joao.nota[0] + joao.nota[1] + joao.nota[2]) / 3;
```

```
printf("Numero aluno: %d\n", (*pa).num_aluno);  
printf("Nota 1: %f\n", (*pa).nota[0]);  
printf("Nota 2: %f\n", (*pa).nota[1]);  
printf("Nota 3: %f\n", (*pa).nota[2]);  
printf("Media 3: %f\n", (*pa).media);
```

// comandos equivalentes

```
printf("\n");  
printf("Numero aluno: %d\n", pa->num_aluno);  
printf("Nota 1: %f\n", pa->nota[0]);  
printf("Nota 2: %f\n", pa->nota[1]);  
printf("Nota 3: %f\n", pa->nota[2]);  
printf("Media 3: %f\n", pa->media);
```

```
struct aluno {  
    int num_aluno;  
    float nota[3];  
    float media;  
};
```

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

```
struct aluno joao, *pa;  
pa = &joao;
```

```
joao.num_aluno = 10;  
joao.nota[0] = 10;  
joao.nota[1] = 4.4;  
joao.nota[2] = 7;  
joao.media = (joao.nota[0] + joao.nota[1] + joao.nota[2]) / 3;
```

```
printf("Numero aluno: %d\n", (*pa).num_aluno);  
printf("Nota 1: %f\n", (*pa).nota[0]);  
printf("Nota 2: %f\n", (*pa).nota[1]);  
printf("Nota 3: %f\n", (*pa).nota[2]);  
printf("Media 3: %f\n", (*pa).media);
```

ALERTA:
***pa.media ≠ (*pa).media**
Devido a precedência entre
os operadores (.) e (*)

// comandos equivalentes

```
printf("\n");  
printf("Numero aluno: %d\n", pa->num_aluno);  
printf("Nota 1: %f\n", pa->nota[0]);  
printf("Nota 2: %f\n", pa->nota[1]);  
printf("Nota 3: %f\n", pa->nota[2]);  
printf("Media 3: %f\n", pa->media);
```

Exemplo

Slide retirado do material da Profa.
Gina M. B. de Oliveira

```
struct aluno joao, *pa;  
pa = &joao;
```

```
joao.num_aluno = 10;  
joao.nota[0] = 10;  
joao.nota[1] = 4.4;  
joao.nota[2] = 7;  
joao.media = (joao.nota[0] + joao.nota[1] + joao.nota[2]) / 3;
```

```
printf("Numero aluno: %d\n", (*pa).num_aluno);  
printf("Nota 1: %f\n", (*pa).nota[0]);  
printf("Nota 2: %f\n", (*pa).nota[1]);  
printf("Nota 3: %f\n", (*pa).nota[2]);  
printf("Media 3: %f\n", (*pa).media);
```

// comandos equivalentes

```
printf("\n");  
printf("Numero aluno: %d\n", pa->num_aluno);  
printf("Nota 1: %f\n", pa->nota[0]);  
printf("Nota 2: %f\n", pa->nota[1]);  
printf("Nota 3: %f\n", pa->nota[2]);  
printf("Media 3: %f\n", pa->media);
```

```
struct aluno {  
    int num_aluno;  
    float nota[3];  
    float media;  
};
```

Sintaxe mais fácil

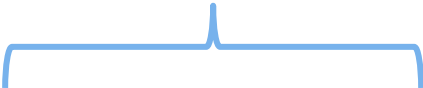
Ponteiro para Ponteiro

- Endereça variáveis do tipo ponteiro
 - Endereço do endereço de memória
- Usado para passar um **ponteiro por referência**
 - Permite mudar o endereço contido no ponteiro
 - Muito usado nas operações com estruturas de dados
- **Sintaxe:**
*tipo_ponteiro * nome_variavel;*

Ponteiro para Ponteiro

- Endereça variáveis do tipo ponteiro
 - Endereço do endereço de memória
- Usado para passar um **ponteiro por referência**
 - Permite mudar o endereço contido no ponteiro
 - Muito usado nas operações com estruturas de dados
- Sintaxe:

tipo_ponteiro * *nome_variavel*;


tipo_dado *

Exemplo

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

```
int main() {
```

```
    int x = 10;
```

```
    int *p = &x; // p aponta para um inteiro e endereça x
```

```
    int **q = &p; // q aponta para um ponteiro de inteiro e endereça p
```

```
    printf ("q=%d\n", q); // Retorna o conteúdo de q
```

```
    printf ("p=%d\n", *q); // Retorna o conteúdo da posição apontada por q
```

```
    printf ("x=%d\n", **q); // Retorna o conteúdo da posição apontada por p
```

```
}
```

Exemplo

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

```
int main() {  
    int x = 10;
```

Tipo dos dados
apontados

```
    int *p = &x; // p aponta para um inteiro e endereça x
```

```
    int **q = &p; // q aponta para um ponteiro de inteiro e endereça p
```

```
    printf ("q=%d\n", q); // Retorna o conteúdo de q
```

```
    printf ("p=%d\n", *q); // Retorna o conteúdo da posição apontada por q
```

```
    printf ("x=%d\n", **q); // Retorna o conteúdo da posição apontada por p
```

```
}
```

Exemplo

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

```
int main() {
```

```
    int x = 10;
```

```
    int *p = &x; // p aponta para um inteiro e endereça x
```

```
    int **q = &p; // q aponta para um ponteiro de inteiro e endereça p
```

```
    printf ("q=%d\n", q); // Retorna o conteúdo de q
```

```
    printf ("p=%d\n", *q); // Retorna o conteúdo da posição apontada por q
```

```
    printf ("x=%d\n", **q); // Retorna o conteúdo da posição apontada por p
```

```
}
```

x

10

0x510

Exemplo

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

```
int main() {
```

```
    int x = 10;
```

```
    int *p = &x; // p aponta para um inteiro e endereça x
```

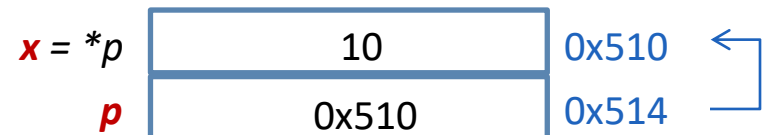
```
    int **q = &p; // q aponta para um ponteiro de inteiro e endereça p
```

```
    printf ("q=%d\n", q); // Retorna o conteúdo de q
```

```
    printf ("p=%d\n", *q); // Retorna o conteúdo da posição apontada por q
```

```
    printf ("x=%d\n", **q); // Retorna o conteúdo da posição apontada por p
```

```
}
```



Exemplo

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

```
int main() {
```

```
    int x = 10;
```

```
    int *p = &x; // p aponta para um inteiro e endereça x
```

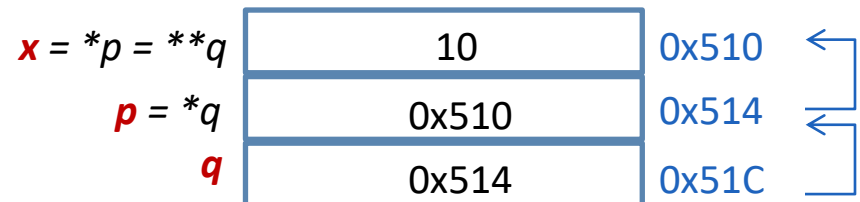
```
    int **q = &p; // q aponta para um ponteiro de inteiro e endereça p
```

```
    printf ("q=%d\n", q); // Retorna o conteúdo de q
```

```
    printf ("p=%d\n", *q); // Retorna o conteúdo da posição apontada por q
```

```
    printf ("x=%d\n", **q); // Retorna o conteúdo da posição apontada por p
```

```
}
```



Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

*Considere que o endereço de **a** é **1234**, de **b** é **1238**, de **p** é **1300** e de **q** é **1304**.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;
```

```
p = &a; q = &b;
```

```
printf("a = %g; b = %g;\n", a, b);
```

```
printf("p = &a = %ld; q = &b = %ld;\n", p, q);
```

```
printf("&p = %ld; &q = %ld;\n", &p, &q);
```

```
printf("a = %g; *p = %g;\n", a, *p);
```

```
printf("b = %g; *q = %g;\n", b, *q);
```

```
*q = *p + 2;
```

```
printf("a = %g; *p = %g;\n", a, *p);
```

```
printf("b = %g; *q = %g;\n", b, *q);
```

```
p = q;
```

```
printf("a = %g; *p = %g;\n", a, *p);
```

```
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

a = 20.8; b = 15.7

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

p = &a = 1234; q = &b = 1238

*Considere que o endereço de **a** é **1234**, de **b** é **1238**, de **p** é **1300** e de **q** é **1304**.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

&p = 1300; &q = 1304

*Considere que o endereço de **a** é **1234**, de **b** é **1238**, de **p** é **1300** e de **q** é **1304**.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

a = 20.8; *p = 20.8

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

b = 15.7; *q = 15.7

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

***q (b) = *p + 2 = a + 2 = 22.8**

*Considere que o endereço de **a** é **1234**, de **b** é **1238**, de **p** é **1300** e de **q** é **1304**.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

a = 20.8; *p = 20.8

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:
b = 22.8; *q = 22.8

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q; p = q = 1238 (&b)  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

*Considere que o endereço de **a** é **1234**, de **b** é **1238**, de **p** é **1300** e de **q** é **1304**.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

a = 20.8; *p = 22.8

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício

Dado o trecho de código a seguir, indique o resultado das instruções:

```
float a = 20.8, *p, b = 15.7, *q;  
p = &a; q = &b;  
printf("a = %g; b = %g;\n", a, b);  
printf("p = &a = %ld; q = &b = %ld;\n", p, q);  
printf("&p = %ld; &q = %ld;\n", &p, &q);  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
*q = *p + 2;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);  
p = q;  
printf("a = %g; *p = %g;\n", a, *p);  
printf("b = %g; *q = %g;\n", b, *q);
```

Saída:

b = 22.8; *q = 22.8

*Considere que o endereço de **a** é 1234, de **b** é 1238, de **p** é 1300 e de **q** é 1304.*

Exercício para Entregar

1. *Repita os procedimentos do exercício anterior para o trecho de código e as informações a seguir:*

```
double *p, *q, *r, A[5];  
p = A; q = p+1; r = q+2;  
printf("&A[0] = %lu; *r = %lf;\n", &A[0], *r);  
printf("p = %lu; *(A+1) = %lf;\n", p, *(A+1));  
r = p+4; p = q;  
printf("p+1 = %lu; *(r-2) = %lf;\n", p+1, *(r-2));  
printf("&A[4] = %lu; *r = %lf;\n", &A, *r);
```

*Considere que uma variável do tipo double ocupa 8 bytes e que o endereço inicial de **A** é **1234600**.*

Referências

- *Coelho, Paulo R. S. L., Linguagem C: Variáveis do Tipo Ponteiro, material didático da disciplina de Introdução a Programação, UFU.*
- *Oliveira, Gina M. B. de, Ponteiros, material didático da disciplina de Algoritmos e Estruturas de Dados 1, UFU.*
- *Backes, André, Linguagem C Descomplicada, portal de vídeo-aulas, <https://programacaodescomplicada.wordpress.com/>, acessado em 09/03/2016.*
- *Celes, W., Cerqueira, R. e Rangel, J. L. Introdução a estruturas de dados. Ed. Campus Elsevier, 2004.*