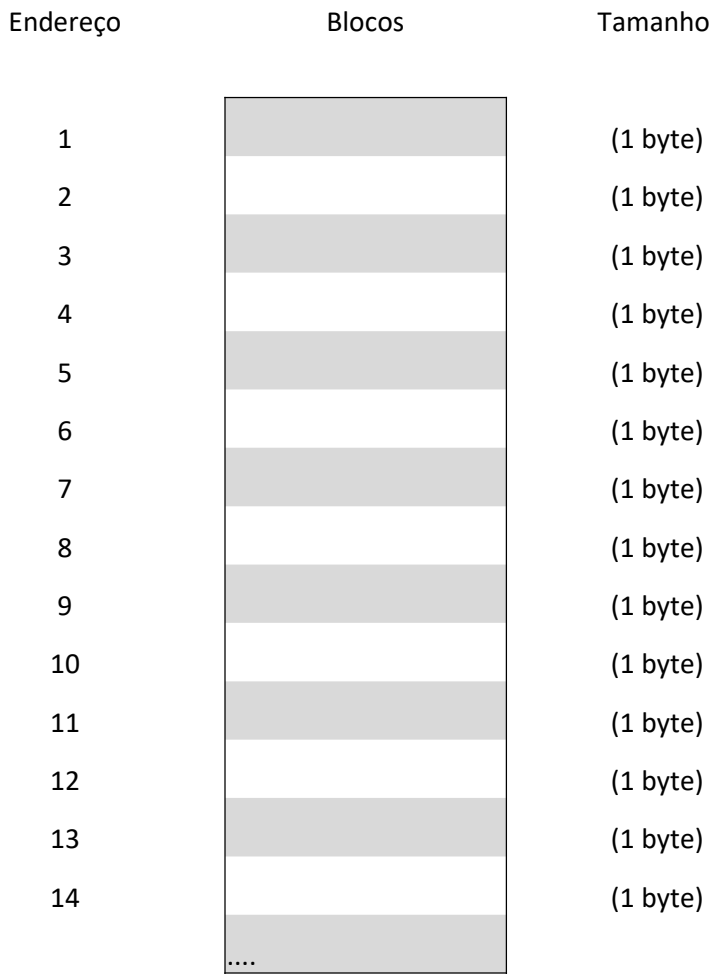
Ponteiros

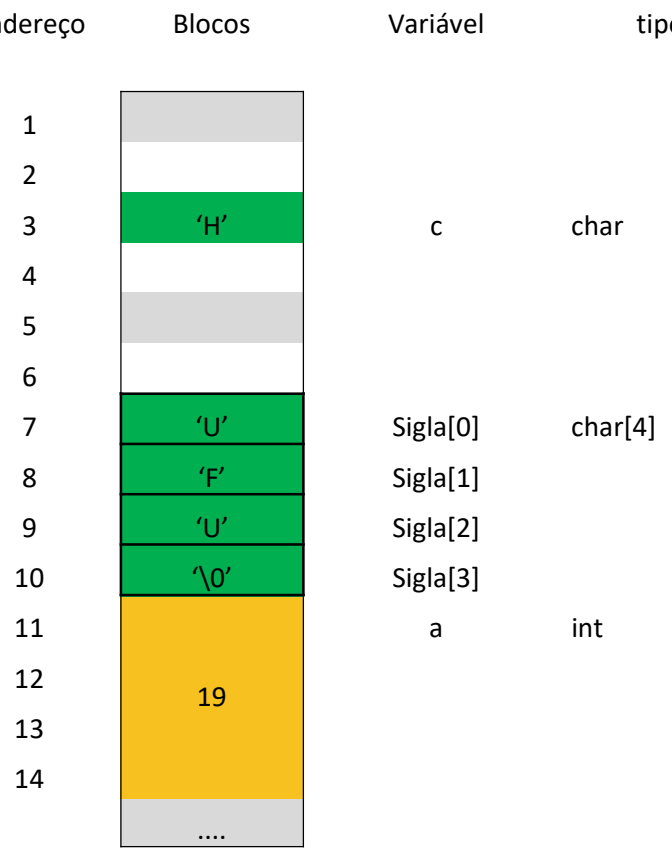
Baseado em slides do Prof. Bruno Travençolo.
Adaptado pela Profa. Gina Oliveira

Observações sobre a memória



Observações sobre a memória

```
char c;
c =  'h';

int a;
a = 19;

char Sigla[4];
Sigla[0] = 'U';
Sigla[1] = 'F';
Sigla[2] = 'U';
Sigla[3] = '\0';
```



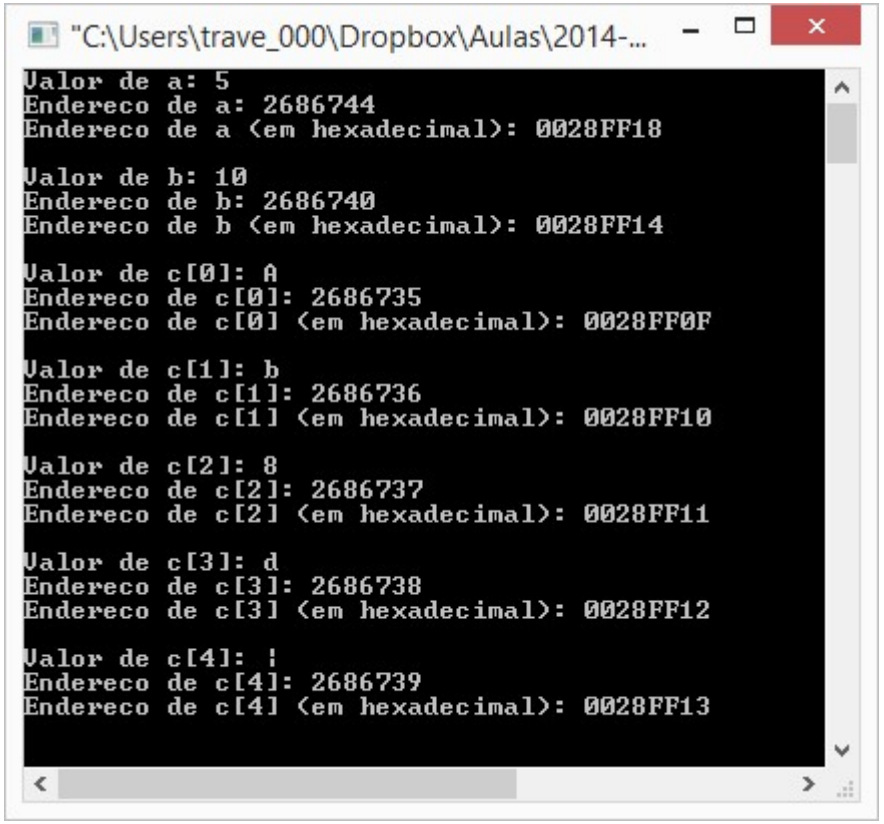
Endereço de variáveis

- Para descobrir o endereço de uma variável em C, use o operador &

```
 6      int i;
 7      int a = 5;
 8      int b = 10;
 9      char c[5] = {'A','b','8','d','|'};
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]);
22         printf("Endereco de c[%d]: %d \n", i, &c[i]);
23         printf("Endereco de c[%d] (em hexadecimal): %p \n\n", i, &c[i]);
24      }
```

Endereço de variáveis

- Para descobrir o endereço de uma variável em C, use o operador &



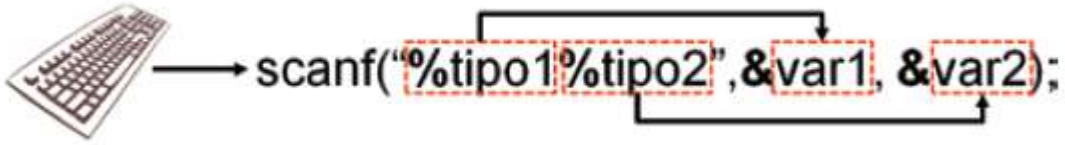
Exercício: faça o mapa de memória para este programa, usando os endereços reais apresentados ao lado

Relembrando a sintaxe do scanf

- Sintaxe: **scanf**("format",&name1,…)
  - format – especificador de formato da entrada que será lida
  - &name1, &name2, ... – endereços das variáveis que irão recerber os valores lidos

  - É preciso especificar o tipo (formato) do dado:
  - %c – leitura de **um** caractere
  - %d – leitura de números inteiros
  - %f – leitura de número reais

  - scanf("tipo de entrada", lista de variáveis)



Relembrando a sintaxe do scanf

- Como "ler em voz alta"

```
scanf("%f",&peso); // leia um valor real (float) e
armazene no endereço da variável peso
```

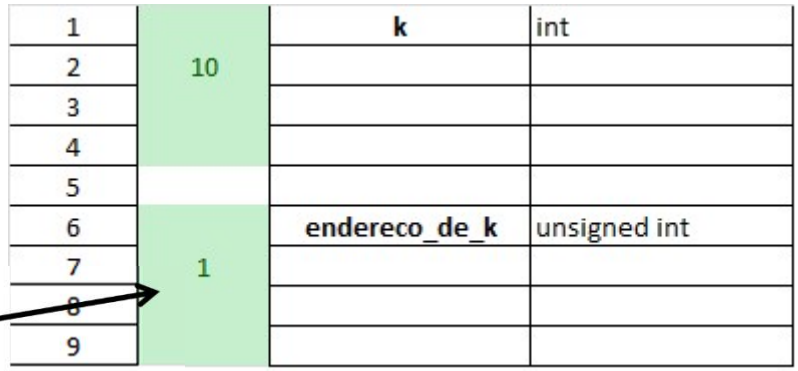
- O símbolo & indica qual é o endereço da variável que vai receber os dados lidos
  - peso – variável peso
  - &peso – endereço da variável peso

Endereços de variáveis

- Se no comando scanf() precisamos passar um endereço de variável, será que é possível passar diretamente o endereço, sem usar o símbolo &?
  - Veja o código a seguir que faz isso

```c
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```c
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;
```




```c
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```c
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

1	10	**k**	int
2			
3			
4			
5			
6	1	**endereco_de_k**	unsigned int
7			
8			
9			

```c
// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco
printf("\n Endereco
printf("\n Endereco
```

1	10	k	int
2			
3			
4			
5			
6	1	endereco_de_k	unsigned int
7			
8			
9			

```
scanf("%d",endereco_de_k);
```

Leia um valor inteiro e armazene no endereço “endereco_de_k”

>> suponha que o usuário digite o número 50

```
// sabemos que o sca
// o que acontece se
// variável k?
printf("\n Digite o
// OBSERVE que não e
scanf("%d",endereco_

variavel k: ");
endereço
```

```
// mostrando o novo
printf("\n\n Valor d
n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco
printf("\n Endereco
printf("\n Endereco
```

1	10	k	int
2			
3			
4			
5			
6	1	endereco_de_k	unsigned int
7			
8			
9			

```
scanf("%d",endereco_de_k);
```

Observe que não foi utilizado o operador “&” para pegar o endereço da variável “endereco_de_k”

Assim, o valor a ser digitado será armazenado no endereço de memória que está escrito na variável “endereco_de_k”

```
// sabemos que o sca
// o que acontece se
// variável k?
printf("\n Digite o
// OBSERVE que não e
scanf("%d",endereco_

variavel k: ");
endereço
```

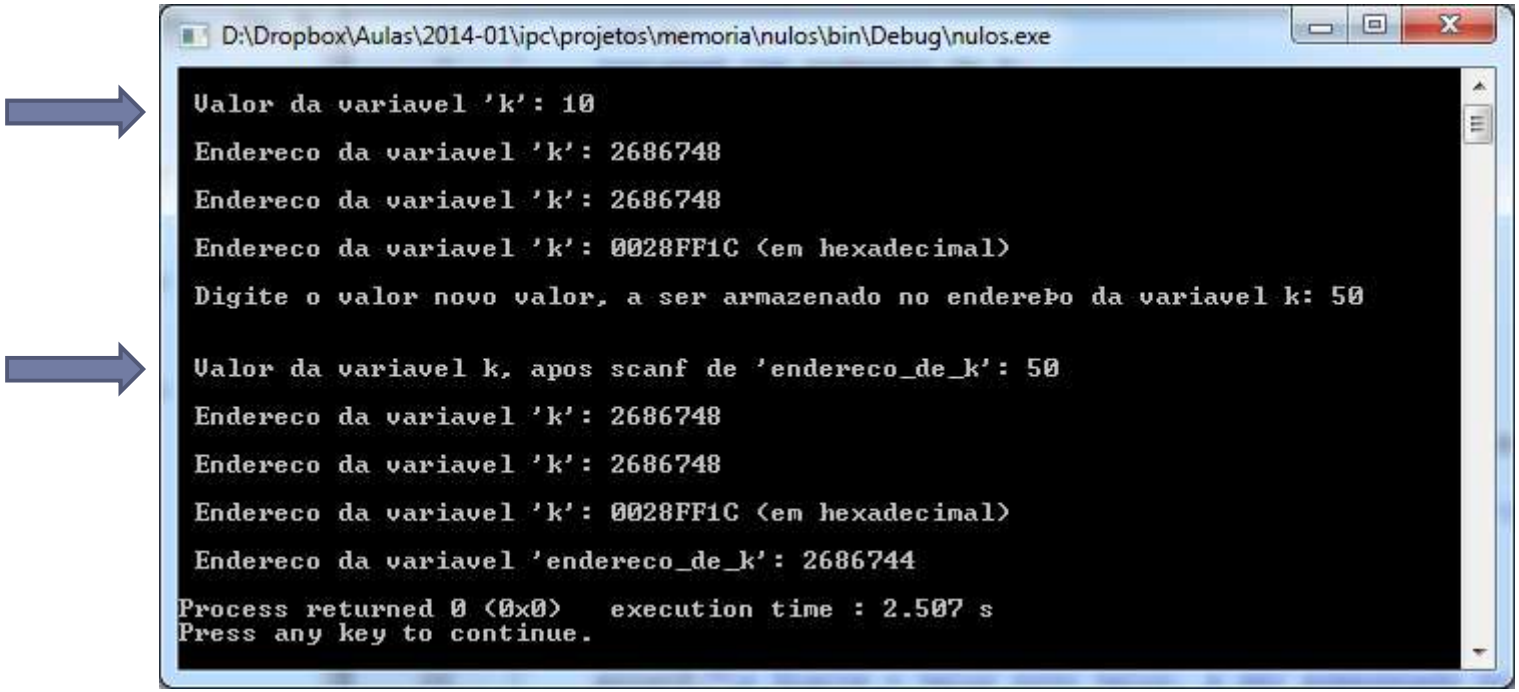
```
// mostrando o novo
printf("\n\n Valor d
n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

```c
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
printf("\n Endereco da variavel 'endereco_de_k': %u \n",&endereco_de_k);
```

1	50	k	int
2			
3			
4			
5			
6	1	endereco_de_k	unsigned int
7			
8			
9			

Execução real

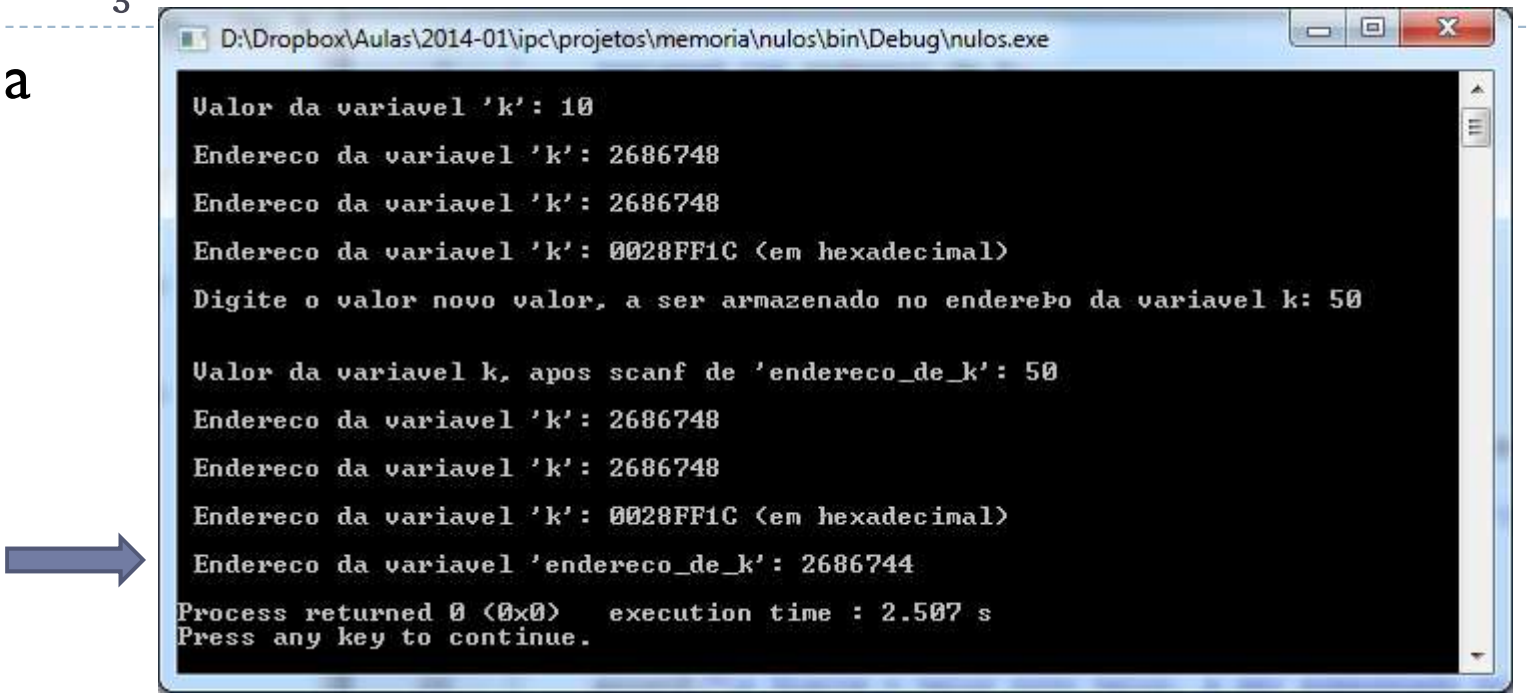
- Saída



- Observe no código que não usamos & para passar o endereço da variável `scanf("%d",endereco_de_k);`
- Observe que o valor da variável k mudou (era 10 e virou 50), mas seu endereço não (continua 2686748 ou 0028FF1C em hexadecimal)

Execução real

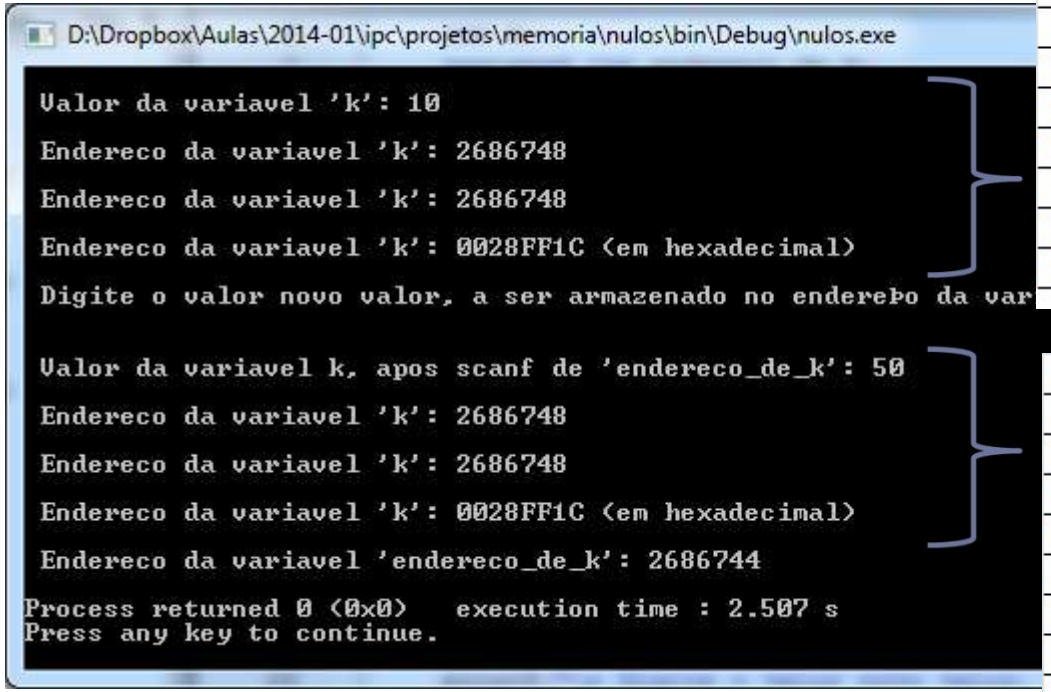
- Saída



- Usamos a variável endereco_de_k para guardar o endereço da variável k, mas lembre-se que endereco_de_k é uma variável e também ocupa espaço de memória e possui um endereço

Execução real

- Saída



2686742			
2686743			
2686744	2686748	endereco_de_k	unsigned int
2686745			
2686746			
2686747			
2686748	10	k	int
2686749			
2686750			
2686751			
2686752			
2686753			

2686742			
2686743			
2686744	2686748	endereco_de_k	unsigned int
2686745			
2686746			
2686747			
2686748	50	k	int
2686749			
2686750			
2686751			
2686752			
2686753			

Ponteiros

- Vimos alguns tipos de dados em C:
  - Simples: int, float, double, char, unsigned int
  - Estruturados: arrays e structs

- Na slides anteriores, usamos o operador `&` para trabalhar com endereços de memória, e usamos também variáveis `unsigned int` para guardar esses endereços
  - Essa não é a forma adequada para trabalhar com variáveis e alterar o seu conteúdo através do seu endereço.
  - Em C devemos usar ponteiros

Ponteiros

- Ponteiro é um tipo de dado usado para armazenar endereços de memória.

- Um ponteiro é uma variável como qualquer outra do programa – sua diferença é que ela não armazena um valor inteiro, real, caractere ou booleano. Ela serve para armazenar **endereços de memória** (que, no fundo, são valores inteiros sem sinal, como um unsigned int).

Ponteiros

- Para declarar uma variável do tipo ponteiro, use a seguinte sintaxe:

```
tipo_de_dado *nome_da_variável
```

- Note que não existe um tipo de dado chamado *pointer*
- O que define o ponteiro é o sinal de * juntamente com um outro tipo de dado do programa
- Esse `tipo_de_dado` deve ser definido pois o ponteiro armazena um endereço de memória, e devemos especificar qual o tipo de dado que existe naquele endereço que ele armazena

Ponteiros

```
tipo_de_dado *nome_da_variável
```

- Exemplo de Ponteiros que são usados para guardar endereços de variáveis inteiras.

```
int *p;
int *proximo;
int *anterior;
int *abacaxi;
```

- Note que, a única diferença na declaração de uma variável do tipo ponteiro é a adição de um símbolo *.

Ponteiros

- `int *p;`
- O espaço ocupado por um ponteiro em um sistema 32 bits é 4 bytes (o mesmo que um unsigned int)
- Lembre que o ponteiro também é uma variável, e portanto ocupa um espaço de memória

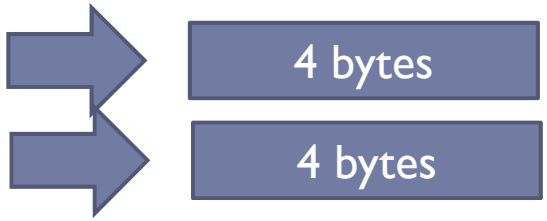
Ponteiros – Espaço Ocupado

```
// armazena endereço de variáveis double
double *valores;
double *estoque;

// armazena endereço de variáveis char
char *nome;
char *endereco;

// armazena endereço de variáveis float
float *temperatura;
```

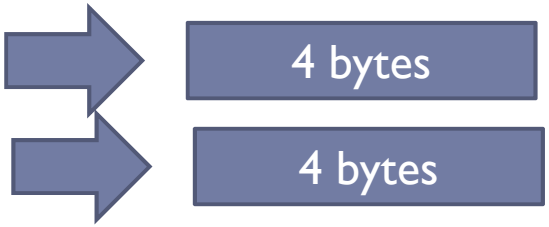
Ponteiros – Espaço Ocupado

- Como o ponteiro armazena um endereço, ele ocupa uma quantidade de memória independente do tipo que ele aponta

```
double *valores;
double *estoque;
```




```
char *nome;
char *endereco;
```




```
float *temperatura;
```

Ponteiros

- Como um ponteiro serve para receber um endereço de memória, podemos fazer, por exemplo, a seguinte operação:

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
                //e inicializa com valor 40
int *p; // cria uma variável do tipo ponteiro para inteiro,
          //chamada p, e o conteúdo inicial é lixo
p = &a; // faz p receber o endereço de a.
           //Dizemos que p aponta para a
```

Ponteiros

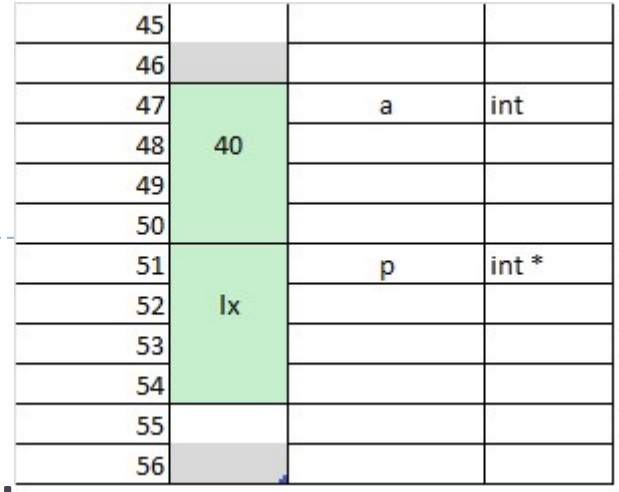
45			
46			
47	40	a	int
48			
49			
50			
51			
52			
53			
54			
55			
56			

- Mapa de memória
  - Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits)

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
            //e inicializa com valor 40
int *p; // cria uma variável do tipo ponteiro para inteiro,
        //chamada p, e o conteúdo inicial é lixo
p = &a; // faz p receber o endereço de a.
         //Dizemos que p aponta para a
```

Ponteiros

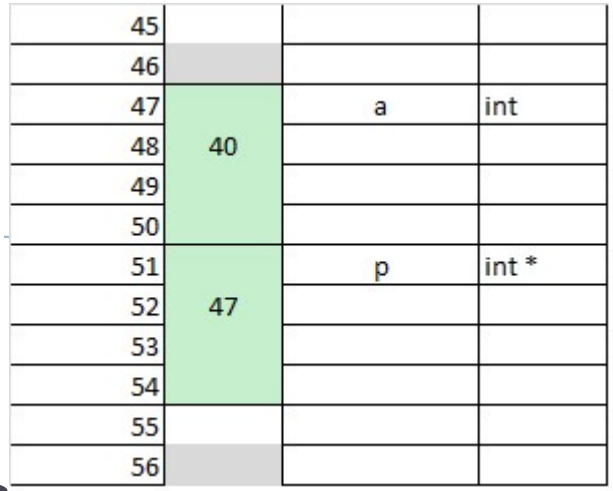
- Mapa de memória
  - Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits)




```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
            //e inicializa com valor 40
int *p; // cria uma variável do tipo ponteiro para inteiro,
        //chamada p, e o conteúdo inicial é lixo
```

```
p = &a; // faz p receber o endereço de a.
        //Dizemos que p aponta para a
```

Ponteiros



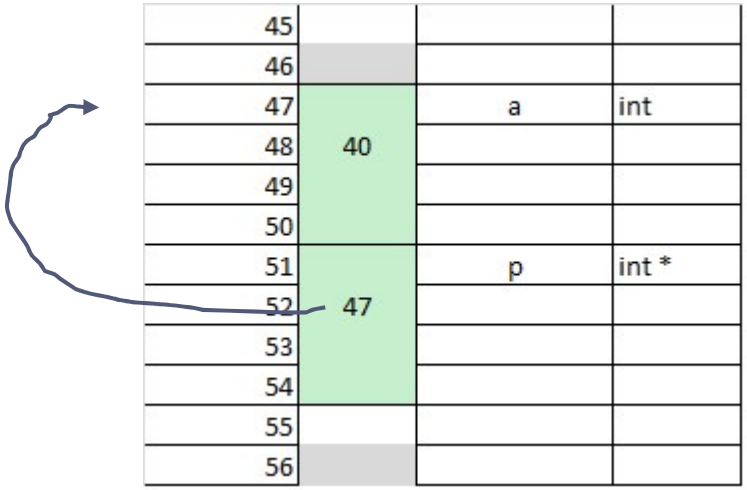
45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

- Mapa de memória
  - Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits)

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
            //e inicializa com valor 40
int *p; // cria uma variável do tipo ponteiro para inteiro,
        //chamada p, e o conteúdo inicial é lixo
p = &a; // faz p receber o endereço de a.
         //Dizemos que p aponta para a
```

Ponteiros

- Mapa de memória
  - “p aponta para a”



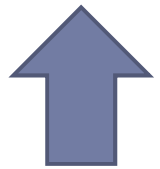
Desreferenciar (*dereferencing*) um ponteiro

- Existem operadores específicos em C para trabalhar com ponteiros.
- Um desses operadores é o símbolo *
  - Note que o mesmo símbolo é usado para declarar um ponteiro e também para multiplicação – mas essas operações não são relacionadas.
- O operador * serve para desreferenciar (*dereferencing*) um ponteiro – ou seja, ele retorna o **conteúdo** do endereço de memória que ele referencia/aponta.
- Ao usar o operador *, o tipo retornado será o mesmo tipo apontado pelo ponteiro

Desreferenciar (*dereferencing*) um ponteiro

- Exemplo

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
                //e inicializa com valor 40
int *p; // cria uma variável do tipo ponteiro para inteiro,
         //chamada p, e o conteúdo inicial é lixo
p = &a; // faz p receber o endereço de a.
          //Dizemos que p aponta para a

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



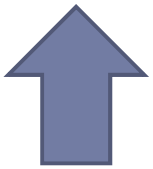
Saída:
O valor da variavel ‘a’ eh: 40

Desreferenciar (*dereferencing*)

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

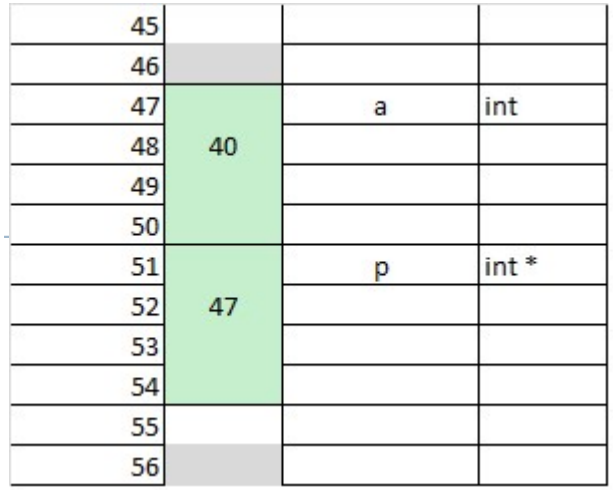
- Exemplo

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



- O programa vai até o ponteiro **p** e verifica para qual endereço ele aponta.
  - No exemplo, é o endereço 47
- Em seguida, o programa vai até o endereço 47 e busca a informação que está lá.
  - No caso, o valor contido no endereço 47 é o valor inteiro 40, que é mostrado como resposta no printf

Diferentes operadores



- Exemplo

```
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: %u", p)

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

Diferentes operadores

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

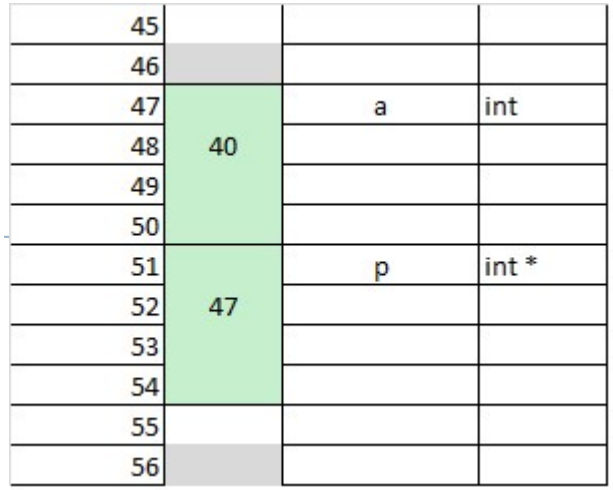
- Exemplo

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

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

Saída: `O valor da variavel ‘p’ eh:` 47

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

Diferentes operadores



- Exemplo

```
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)
```

Saída: O endereço da var. ‘p’ eh: 51

Desreferenciar (*dereferencing*) um ponteiro

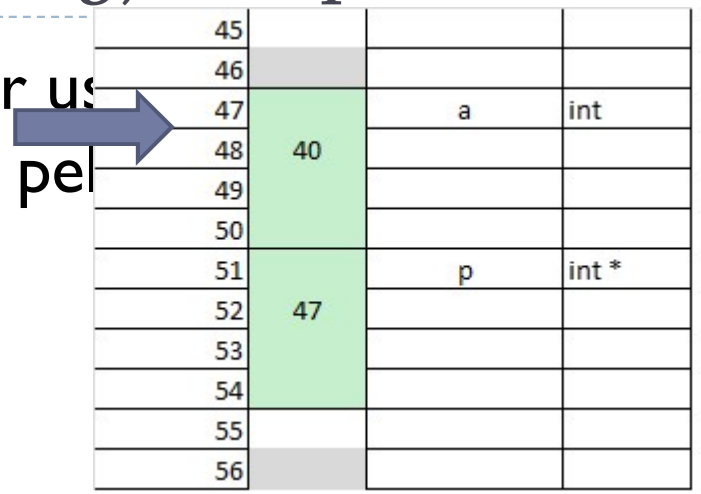
- O desreferenciamento * pode ser usado para atribuição de valores às variáveis apontadas pelos ponteiros,

```
int a = 40;
int *p;
p = &a; // faz p receber o endereço de a
*p = 59; // altera o conteúdo do endereço apontado por p


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

Saída:
O valor da variavel 'a' eh: 59

Desreferenciar (*dereferencing*) um ponteiro

- O desreferenciamento * pode ser us
  de valores às variáveis apontadas pel



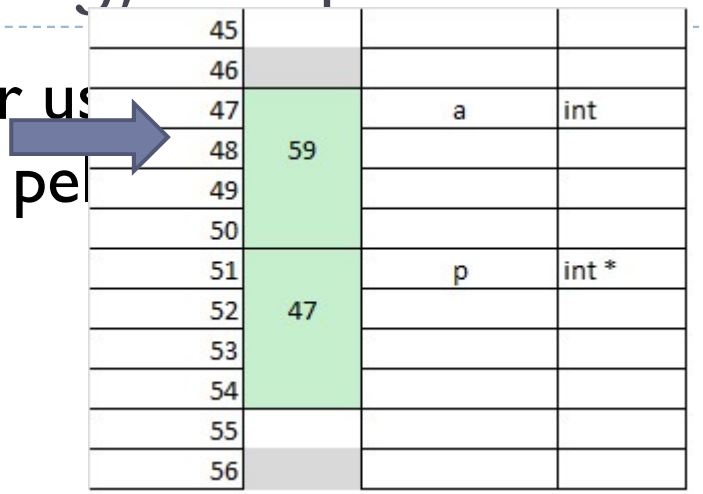

```
int a = 40;
int *p;
p = &a; // faz p receber o endereço de a
*p = 59; // altera o conteúdo do endereço apontado por p
```

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

Saída:
O valor da variavel 'a' eh: 59

Desreferenciar (*dereferencing*) um ponteiro

- O desreferenciamento * pode ser us
  de valores às variáveis apontadas pe



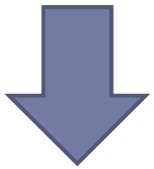

```
int a = 40;
int *p;
p = &a; // faz p receber o endereço de a
*p = 59; // altera o conteúdo do endereço apontado por p


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

Saída:
O valor da variavel 'a' eh: 59

Desreferenciar (*dereferencing*)

45			
46			
47	59	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

- Exemplo

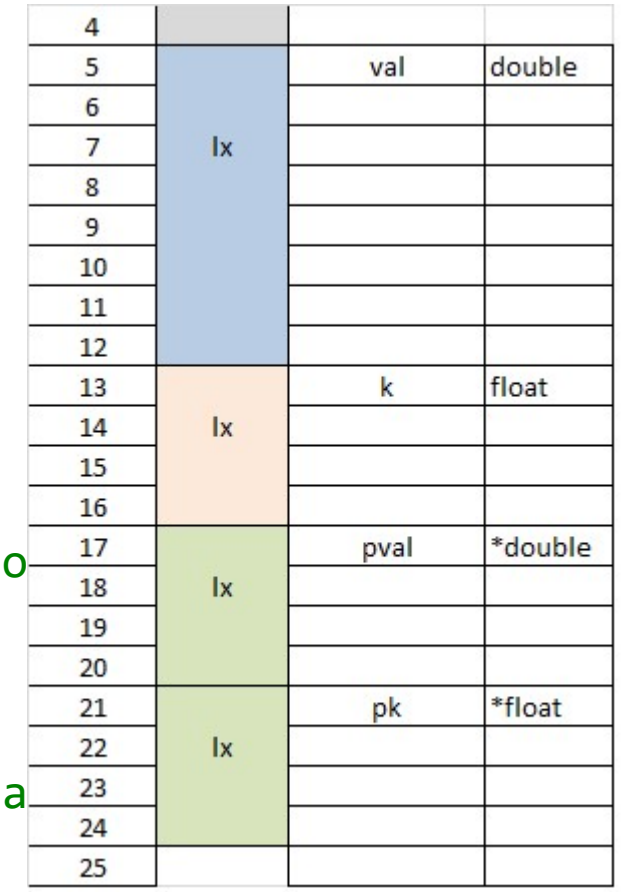


```
*p = 59; // altera o conteúdo do endereço apontado por p
```

- O programa vai até o ponteiro **p** e verifica para qual endereço ele aponta.
  - No exemplo, é o endereço 47
- Em seguida, o programa vai até o endereço 47 e faz a atribuição do valor 59 neste local

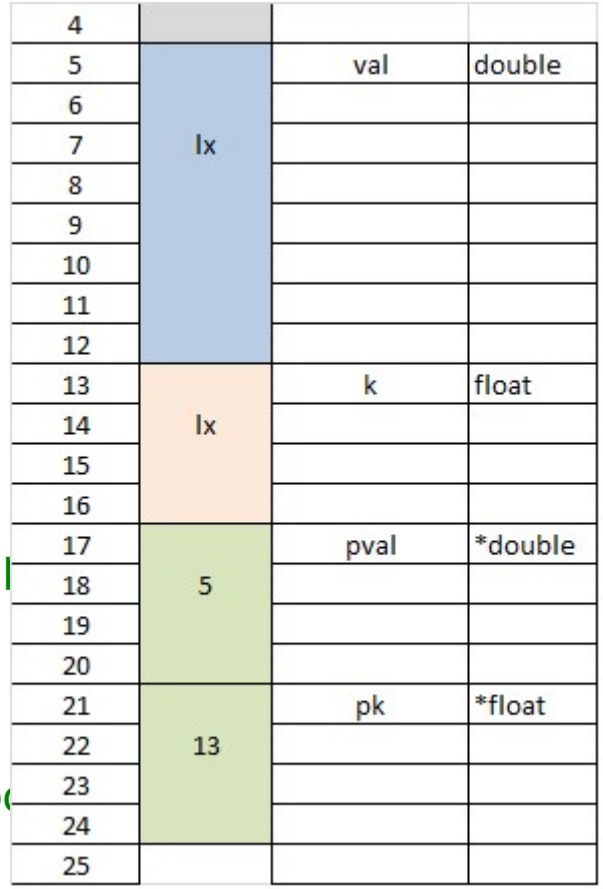
Exemplos

```
// declarando as variáveis
double val;
float k;

// declarando os ponteiros
double *pval;
float *pk;

// atribuindo os valores das variáveis ao
pval = &val;
pk = &k;

// alterando o conteúdo das variáveis via
*pval = 33.45;
*pk = 2.4;
```



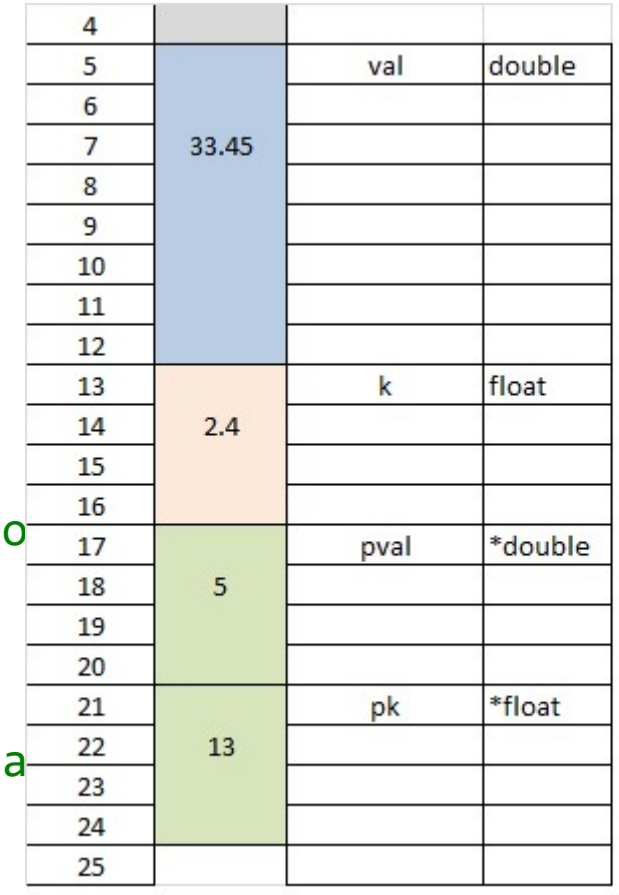
Exemplos

```
// declarando as variáveis
double val;
float k;

// declarando os ponteiros
double *pval;
float *pk;

// atribuindo os valores das variáveis aos p
pval = &val;
pk = &k;

// alterando o conteúdo das variáveis via po
*pval = 33.45;
*pk = 2.4;
```



Exemplos

```
// declarando as variáveis
double val;
float k;

// declarando os ponteiros
double *pval;
float *pk;

// atribuindo os valores das variáveis ao
pval = &val;
pk = &k;

// alterando o conteúdo das variáveis via
*pval = 33.45;
*pk = 2.4;
```



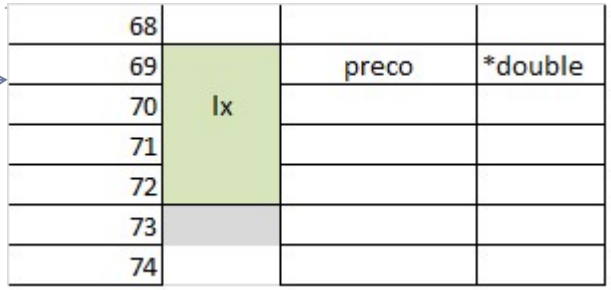
Está correto?

```
double *preco;
*preco = 50.0;
```

- Não houve alocação para guardar um número ‘double’
- houve somente alocação para guardar um ponteiro para double

Está correto?

```
double *preco;
*preco = 50.0;
```



- Ao declarar uma variável, o conteúdo dela é lixo

Está correto?

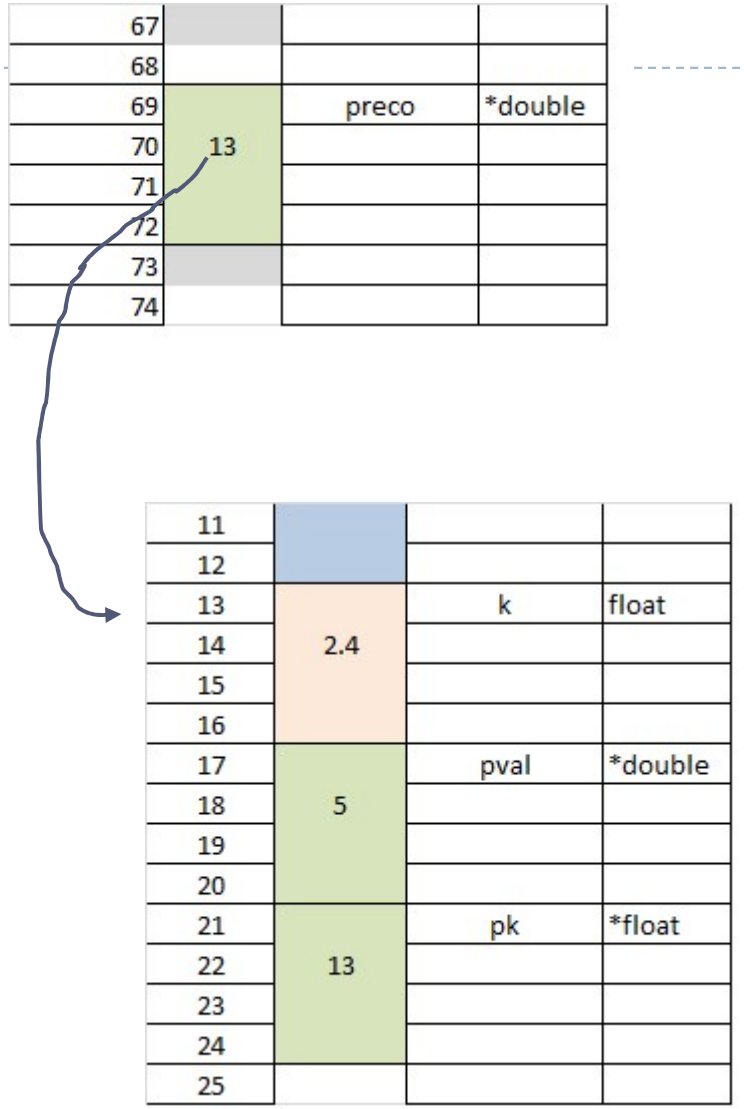
```
double *preco;
*preco = 50.0;
```

- Ao declarar uma variável, o conteúdo dela é lixo
- Suponha que o lixo seja o valor 13

Está correto?

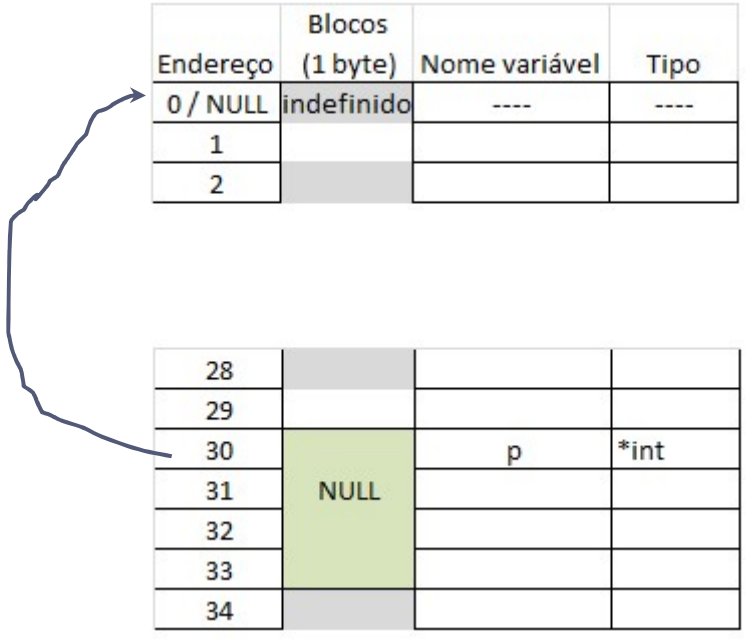
```
double *preco;
*preco = 50.0;
```

- O programa tentará alterar o que está no endereço 13, e poderá travar
- Pode alterar outras variáveis



Inicialização

- Um ponteiro pode ter o valor especial NULL que é o endereço de nenhum lugar.
  - Ex: **int *p = NULL;**
- Pode-se usar o valor 0 (zero) ao invés de NULL



```c
int k;
unsigned int endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

Altere esse programa utilizando ponteiros (que é a forma correta de se fazer)

```c
int k;
int *endereco_de_k;

// inicializando k
k = 10;
printf("\n Valor da variavel 'k': %d \n",k);

// obtendo o endereço da variável 'k'
// vamos usar o operador &
endereco_de_k = &k;

printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);

// sabemos que o scanf pede um endereço de memória
// o que acontece se passarmos o endereço da
// variável k?
printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
// OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
scanf("%d",endereco_de_k);

// mostrando o novo valor de 'k'
printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);

// mostrando o endereço de 'k' que deve permanecer o mesmo de antes
printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
printf("\n Endereco da variavel 'k': %u \n",&k);
printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

Forma correta

Operações com ponteiros

- Atribuição
  - p1 aponta para o mesmo lugar que p2;

    p1 = p2;

  - a variável apontada por p1 recebe o mesmo conteúdo da variável apontada por p2;

    ***p1 = *p2;**

Operações com ponteiros

- Apenas duas operações aritméticas podem ser utilizadas no endereço armazenado pelo ponteiro: adição e subtração
  - podemos apenas somar e subtrair valores INTEIROS
    - p++; soma +1 no endereço armazenado no ponteiro.
    - p--; subtrai 1 no endereço armazenado no ponteiro.
    - p = p+15; soma +15 no endereço armazenado no ponteiro.

Operações com ponteiros

- As operações de adição e subtração no endereço dependem do tipo de dado que o ponteiro aponta.

- Considere um ponteiro para inteiro, **int** *. O tipo **int** ocupa um espaço de 4 bytes na memória.

- Assim, nas operações de adição e subtração são adicionados/subtraídos 4 bytes por incremento/decremento, pois esse é o tamanho de um inteiro na memória e, portanto, é também o valor mínimo necessário para sair dessa posição reservada de memória

Operações com ponteiros

- Operações ilegais com ponteiros
  - Dividir ou multiplicar ponteiros;
  - Somar o endereço de dois ponteiros;
  - Não se pode adicionar ou subtrair float ou double de ponteiros.

Operações com ponteiros

- Já sobre seu conteúdo apontado, valem todas as operações
  - (*p)++; incrementar o conteúdo da variável apontada pelo ponteiro p;
  - *p = (*p) * 15; multiplica o conteúdo da variável apontada pelo ponteiro p por 15;

Ponteiros Genéricos

- Normalmente, um ponteiro aponta para um tipo específico de dado.
  - Um ponteiro genérico é um ponteiro que pode apontar para qualquer tipo de dado.
- Declaração

```
void *nome_ponteiro;
```

Exemplo

```c
#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;

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

- Erro em tempo de compilação:
  - ERROR: invalid use of void expression

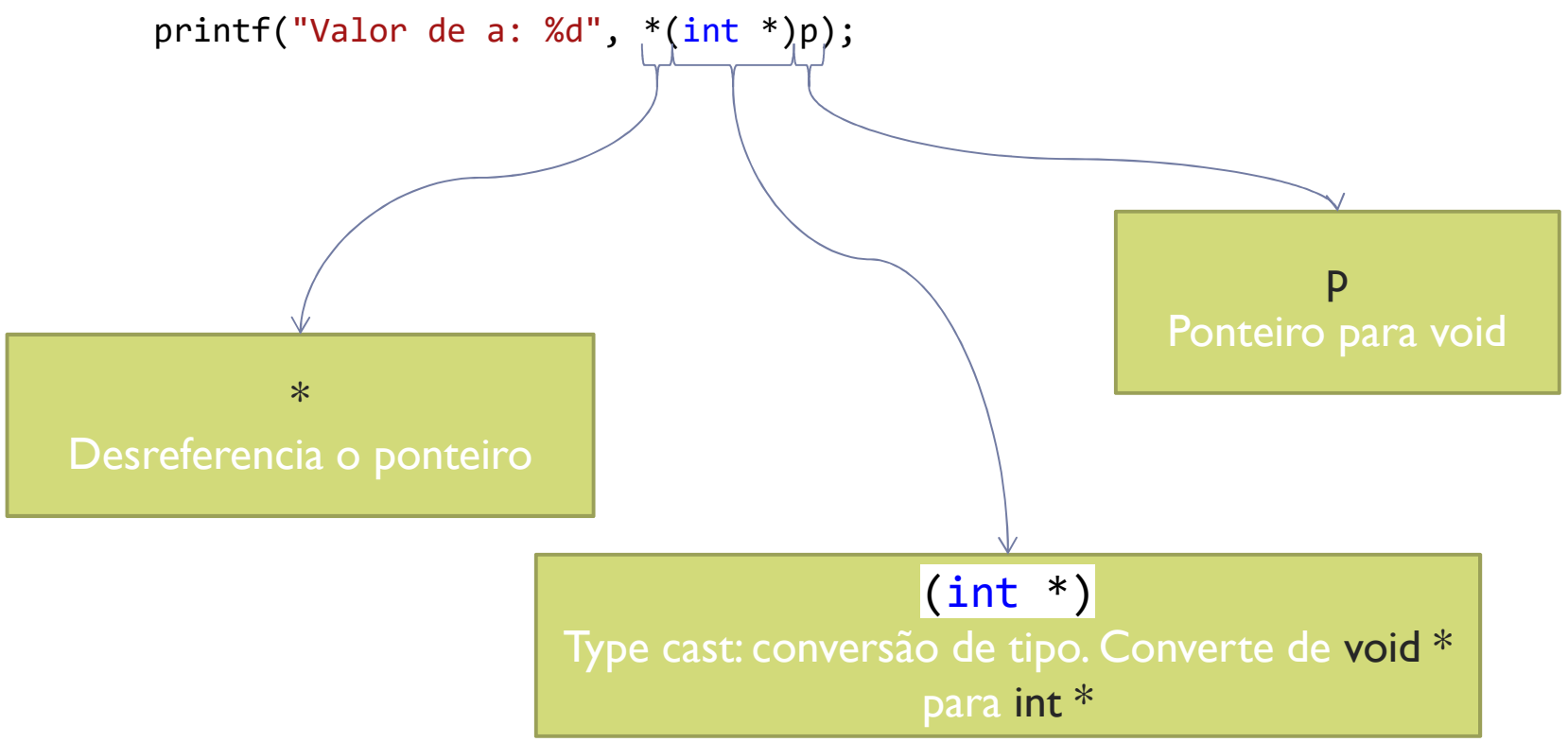
Exemplo

```c
#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);
}
```

- Para usar *void, é necessário fazer a conversão para o tipo do ponteiro que o void aponta. No caso, é um ponteiro para inteiro
  - Use para converter: (int *)

Exemplo

- Observe com atenção



Exemplo

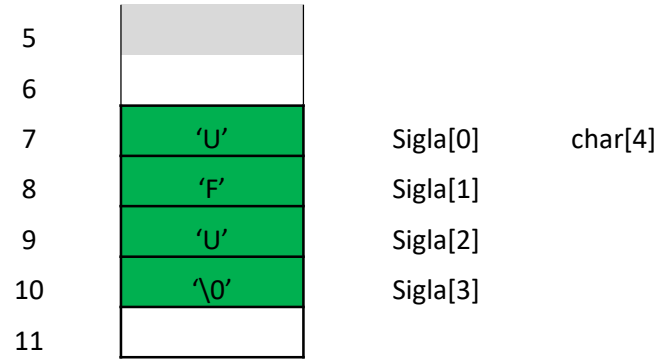
- Mudando o apontamento de p de um inteiro para um double

```
    int a = 10;
    double d = 30;

    void *p;

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

    // mostrando o conteúdo do endereço apontado por p (no caso, a var. ‘a’)
//    printf("Valor de a: %d", *p); // errado!


    // mostrando o conteúdo do endereço apontado por p (no caso, a var. ‘a’)
    printf("Valor de a: %d", *(int *)p);

    // atribuindo o endereço de 'd' (double) ao ponteiro void
    p = &d;

    // mostrando o conteúdo do endereço apontado por p (no caso, a var. ‘d’)
    printf("Valor de b: %f", *(double *)p);
```

Ponteiros e Arrays

- Ponteiros e arrays possuem uma ligação muito forte.
  - Arrays são agrupamentos de dados do mesmo tipo na memória.

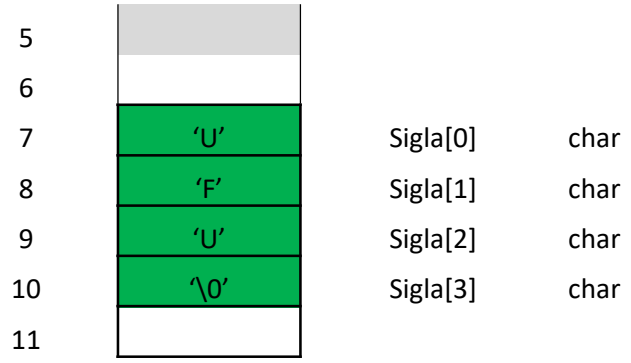
4 'variáveis' char agrupadas



Ponteiros e Arrays

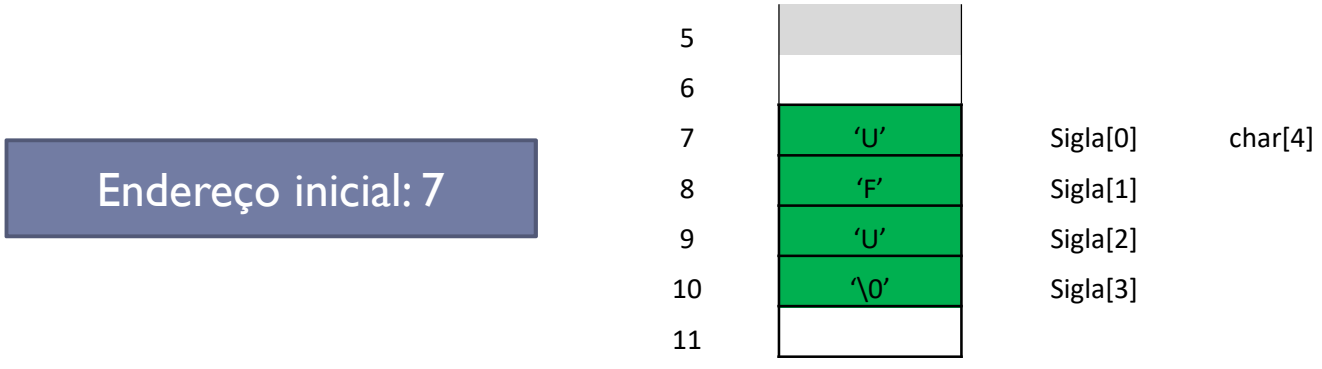
- Ponteiros e arrays possuem uma ligação muito forte.
  - Quando declaramos um array, informamos ao computador para reservar uma certa quantidade de memória a fim de armazenar os elementos do array de forma sequencial.

4 x 1 byte = 4 bytes



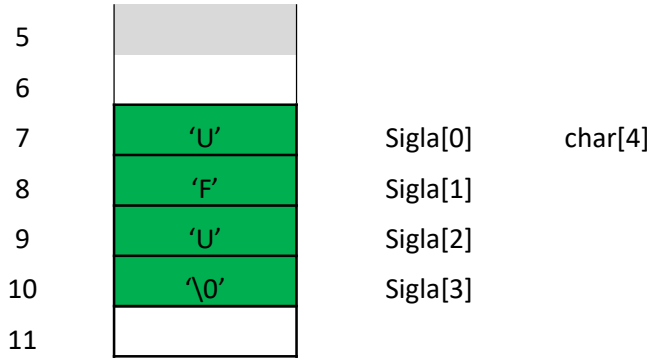
Ponteiros e Arrays

- Ponteiros e arrays possuem uma ligação muito forte.
  - Como resultado dessa operação, o computador nos devolve um ponteiro que aponta para o começo dessa sequência de bytes na memória.



Ponteiros e Arrays

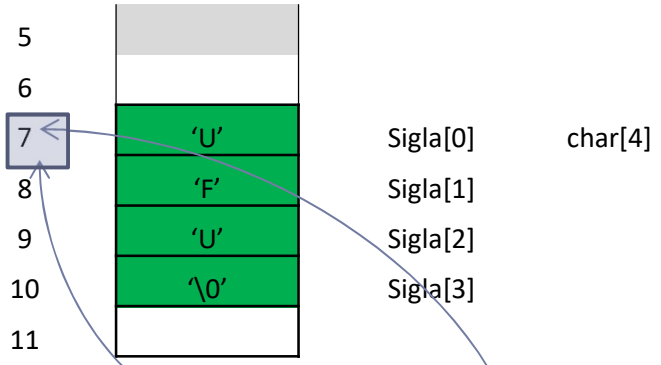
- Em C, o nome do array (sem índice) é apenas **um ponteiro** que aponta para o primeiro elemento do array.




```
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 Posicao de indice zero do vetor (Sigla): %u",Sigla);
```

Ponteiros e Arrays

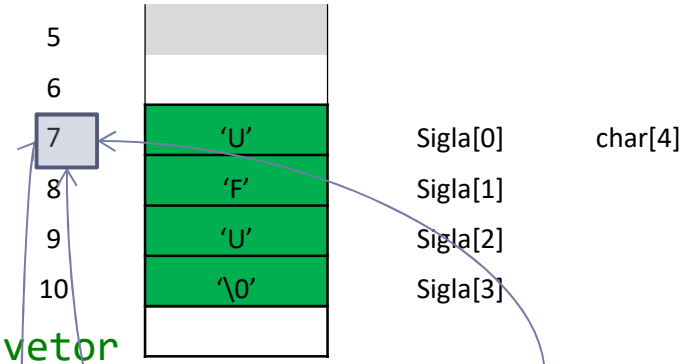
- Em C, o nome do array (sem índice) é apenas **um ponteiro** que aponta para o primeiro elemento do array.
  - **&Sigla[0]** é igual **Sigla**




```
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 Posicao de indice zero do vetor (Sigla): %u",Sigla);
```

Ponteiros e Arrays

- O endereço do vetor é o próprio vetor
  - **&Sigla** é igual **Sigla**

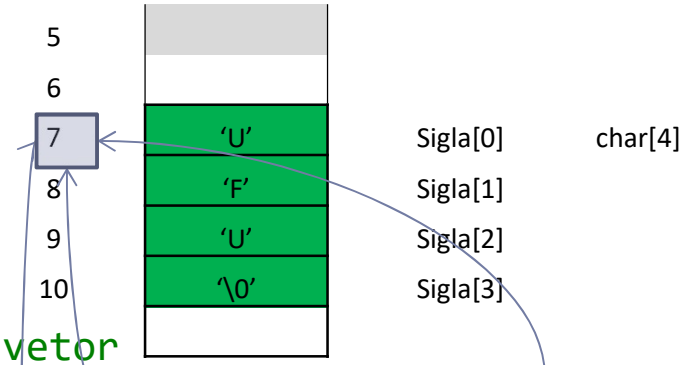



```
char Sigla[4] = "UFU";

// 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);
```

Ponteiros e Arr

```
D:\Dropbox\Aulas\2014-01\ipc\projetos\memoria\ArraysPonteiros\bin\Debug\Arra
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.
```

- O endereço do vetor é o próprio vetor
  - **&Sigla** é igual **Sigla**




```
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);
```

Ponteiros e Arrays

- O nome do array (sem índice) é apenas um ponteiro que aponta para o primeiro elemento do array.

67			
68			
69	79	p	char *
70			
71			
72			
73			
74			
75			
76			
77			
78			
79	'U'	Sigla[0]	char
80	'F'	Sigla[1]	char
81	'U'	Sigla[2]	char
82	'\0'	Sigla[3]	char
83			
84			

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

p = Sigla;
```

Ponteiros e Arrays

- Nesse exemplo

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

- Temos que:
  - ***p** é equivalente a **Sigla[0]**;
  - **Sigla[indice]** é equivalente a ***(p+indice)**;
  - **Sigla** é equivalente a **&Sigla[0]**;
  - **&Sigla[indice]** é equivalente a **(Sigla + indice)**;

Ponteiros e Arrays

Exemplo: acessando arrays utilizando ponteiros

Usando array

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int vet[5]= {1,2,3,4,5};
05    int *p = vet;
06    int i;
07    for (i = 0;i < 5;i++)
08      printf("%d\n",p[i]);
09    system("pause");
10    return 0;
11  }
```

Usando ponteiro

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int vet[5]= {1,2,3,4,5};
  int *p = vet;
  int i;
  for (i = 0;i < 5;i++)
    printf("%d\n",*(p+i));
  system("pause");
  return 0;
}
```

Ponteiros e Arrays

- Os colchetes [ ] substituem o uso conjunto de operações aritméticas e de acesso ao conteúdo (operador "*") no acesso ao conteúdo de uma posição de um array ou ponteiro.
  - O valor entre colchetes é o deslocamento a partir da posição inicial. Nesse caso, **p[2]** equivale a ***(p+2)**.

```
#include <stdio.h>
#include <stdlib.h>
int main (){
        int vet[5] = {1,2,3,4,5};
        int *p;
        p = vet;
        printf ("Terceiro elemento: %d ou %d",p[2],*(p+2));
        system("pause");
        return 0;
}
```

Ponteiros e Structs

- Existe um operador específico para trabalhar com desreferenciamento de ponteiros para struct

- Operador ->
  - Como usar: ponteiro -> membro_da_struct
  - Exemplo
  - (*p).x é igual a p->x

```c
int main(){

  struct aluno joao;

  joao.num_aluno = 10;
  joao.nota1 = 10;
  joao.nota2 = 4.4;
   joao.nota3 = 7;
  joao.media = (joao.nota1 + joao.nota2 + joao.nota3)/3.0;

  struct aluno *pa;

  pa = &joao;

  printf("Numero aluno: %d\n", (*pa).num_aluno);
  printf("Nota 1: %f\n", (*pa).nota1);
  printf("Nota 2: %f\n", (*pa).nota2);
  printf("Nota 3: %f\n", (*pa).nota3);
  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->nota1);
  printf("Nota 2: %f\n", pa->nota2);
  printf("Nota 3: %f\n", pa->nota3);
  printf("Media 3: %f\n", pa->media);
```

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};
```

```c
int main(){

  struct aluno joao;

  joao.num_aluno = 10;
  joao.nota1 = 10;
  joao.nota2 = 4.4;
   joao.nota3 = 7;
  joao.media = (joao.nota1 + joao.nota2 + joao.nota3)/3.0;

  struct aluno *pa;

  pa = &joao;

  printf("Numero aluno: %d\n", (*pa).num_aluno);
  printf("Nota 1: %f\n", (*pa).nota1);
  printf("Nota 2: %f\n", (*pa).nota2);
  printf("Nota 3: %f\n", (*pa).nota3);
  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->nota1);
  printf("Nota 2: %f\n", pa->nota2);
  printf("Nota 3: %f\n", pa->nota3);
  printf("Media 3: %f\n", pa->media);
```

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};
```

Desreferenciamento (*) do ponteiro pa, seguido pelo operador . (ponto) da struct, para acessar os membros

```c
int main(){

  struct aluno joao;

  joao.num_aluno = 10;
  joao.nota1 = 10;
  joao.nota2 = 4.4;
   joao.nota3 = 7;
  joao.media = (joao.nota1 + joao.nota2 + joao.nota3)/3.0;

  struct aluno *pa;

  pa = &joao;

  printf("Numero aluno: %d\n", (*pa).num_aluno);
  printf("Nota 1: %f\n", (*pa).nota1);
  printf("Nota 2: %f\n", (*pa).nota2);
  printf("Nota 3: %f\n", (*pa).nota3);
  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->nota1);
  printf("Nota 2: %f\n", pa->nota2);
  printf("Nota 3: %f\n", pa->nota3);
  printf("Media 3: %f\n", pa->media);
```

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};
```

Note que *pa.nota1 estaria errado pois o ponto (.) tem precedência sobre o operador de desreferenciamento (*)

```c
int main(){

  struct aluno joao;

  joao.num_aluno = 10;
  joao.nota1 = 10;
  joao.nota2 = 4.4;
   joao.nota3 = 7;
  joao.media = (joao.nota1 + joao.nota2 + joao.nota3)/3.0;

  struct aluno *pa;

  pa = &joao;

  printf("Numero aluno: %d\n", (*pa).num_aluno);
  printf("Nota 1: %f\n", (*pa).nota1);
  printf("Nota 2: %f\n", (*pa).nota2);
  printf("Nota 3: %f\n", (*pa).nota3);
  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->nota1);
  printf("Nota 2: %f\n", pa->nota2);
  printf("Nota 3: %f\n", pa->nota3);
  printf("Media 3: %f\n", pa->media);
```

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};
```

Operador -> é equivalente aos comandos acima e mais fácil de usar, além de possuir alta precedência

Ponteiro para Ponteiro

- Endereça variáveis do tipo ponteiro
  - Normalmente aponta ponteiros para tipos de dados

- **Sintaxe:**

  tipo_ponteiro ** *nome_ponteiro;*

- Usado para passar um **ponteiro por referência**
  - Permite mudar o endereço contido no ponteiro
  - Muito usado nas operações com estruturas de dados

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
}
```

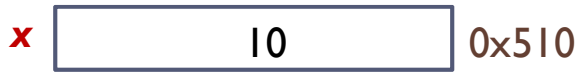
Exemplo

```
#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
}
```

Tipo dos dados apontados

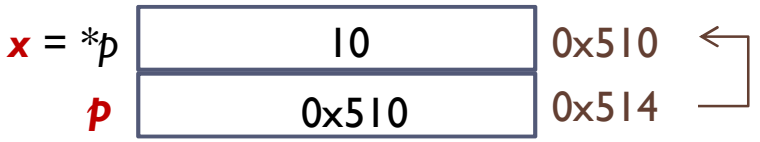
Exemplo

```
#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
}
```



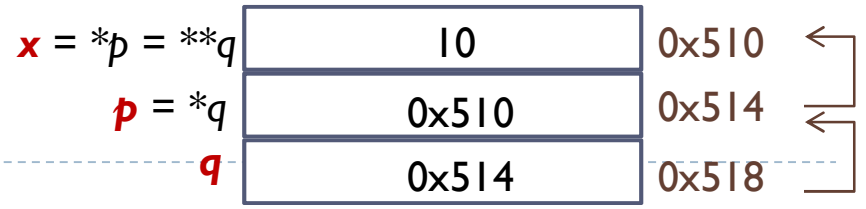
Exemplo

```
#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
}
```



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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
float a = 20.8, *p, b = 15.7, *q;
p = &a; q = &b;
printf("a = %f; b = %f;\n", a, b);
printf("p = &a = %ld; q = &b = %ld;\n", p, q); =>   Saída:
printf("&p = %ld; &q = %ld;\n", &p, &q);              p = &a = 1234; q = &b = 1238
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
*q = *p + 2;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
p = q;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
float a = 20.8, *p, b = 15.7, *q;
p = &a; q = &b;
printf("a = %f; b = %f;\n", a, b);
printf("p = &a = %ld; q = &b = %ld;\n", p, q);
printf("&p = %ld; &q = %ld;\n", &p, &q); =>   Saída: &p = 1300; &q =1304
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
*q = *p + 2;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
p = q;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
float a = 20.8, *p, b = 15.7, *q;
p = &a; q = &b;
printf("a = %f; b = %f;\n", a, b);
printf("p = &a = %ld; q = &b = %ld;\n", p, q);
printf("&p = %ld; &q = %ld;\n", &p, &q);
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
*q = *p + 2;                          (q=22.8)
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
p = q;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
float a = 20.8, *p, b = 15.7, *q;
p = &a; q = &b;
printf("a = %f; b = %f;\n", a, b);
printf("p = &a = %ld; q = &b = %ld;\n", p, q);
printf("&p = %ld; &q = %ld;\n", &p, &q);
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
*q = *p + 2;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q); =>   Saída: b = 22.8; *q = 22.8
p = q;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
float a = 20.8, *p, b = 15.7, *q;
p = &a; q = &b;
printf("a = %f; b = %f;\n", a, b);
printf("p = &a = %ld; q = &b = %ld;\n", p, q);
printf("&p = %ld; &q = %ld;\n", &p, &q);
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
*q = *p + 2;
printf("a = %f; *p = %f;\n", a, *p);
printf("b = %f; *q = %f;\n", b, *q);
p = q;
printf("a = %f; *p = %f;\n", a, *p); => Saída: a = 22.8; *p = 22.8
printf("b = %f; *q = %f;\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

1. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

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

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;     (p = 1234600      q= 1234608    r= 1234616)
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]); => Saída: p=1234600; &A[0]= 1234600
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]); => Saída: p=1234608; &A[1]= 1234608
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]); => Saída:  r=1234616; &A[2]= 1234616
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf(“A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]); => Saída:
printf(“*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);      A[0]=1.2; A[1]= 6.5; A[2]=73.34
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf(“*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r); => Saída: *p=1.2; *q= 6.5; *r=73.34
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p); => Saída: q-p=8; r-p= 16
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;        (r= 1234624    q= 1234624    s=endereço de q)
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1));
```

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

Exercício

2. *Dado o trecho de código a seguir, indique qual é o resultado dos printf:*

```
double *p, *q, *r, A[5]={1.2,6.5,73.34,18,0.1};
p = A; q = p+1; r = p+2;
printf("p = %ld; &A[0] = %ld;\n", p, &A[0]);
printf("q = %ld; &A[1] = %ld;\n", q, &A[1]);
printf("r = %ld; &A[2] = %ld;\n", r, &A[2]);
printf("A[0] = %lf; A[1] = %lf; A[2] = %lf \n", A[0], A[1],A[2]);
printf("*p = %lf; *q = %lf; *r = %lf \n", *p, *q, *r);
printf("q-p = %ld; r-p = %ld;\n", q-p, r-p);
double **s;  r++; q=q+2; s=&q;
printf("*A[3] = %lf; A[4] = %lf; *r = %lf \n", **s, *(r+1)); => Saída: A[3] = 18; A[4] = 0.1
```

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