

SSC0501 - Introdução à Ciência de Computação I

Bem Vindos!

5 9 3
2 6 8
4 0 7

9 2 3
7 8 5
2 2 8

5 9 3
2 6 8
4 0 7

2 2 0 8 1
4 3 5 6 5 2 0
1 2 3 9 6 2 7

Ponteiros!



5 9 3
2 6 8
4 0 7

9 2 3
7 8 5
2 2 8

2 2 0 8 1
4 2 5 6 5 2 0
1 3 9 6 2 7
2

Memória



Memória

Como vocês acham que a memória funciona? Como ela é organizada?



Memória



4 0 7

Memória

Entrada?

Quero ler a memória!

Saída?



Memória

Entrada?

Quero ler a memória!

Saída?

Endereço
(Posição do Livro)

Dado/informação!

(O Livro, seu conteúdo)



Memória

Entrada?

Quero salvar na memória!

Saída?



Memória

Entrada?

Quero salvar na memória!

Saída?

Endereço
(Posição do Livro)



Dado/informação
(O Livro)



Status da
operação:
Sucesso/Fracasso

Memória

Como podemos representar a
memória?

```
int matrix[3][3]:
```

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8



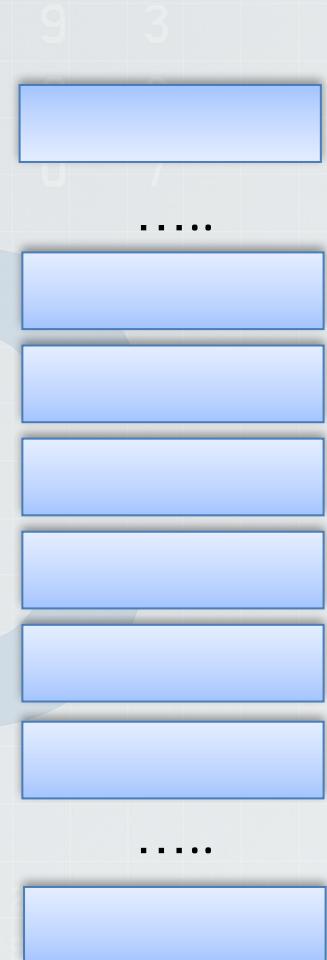
Memória

Como podemos representar a memória?

```
int matrix[3][3]:
```

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8



Memória

Como podemos representar a memória?

int matrix[3][3]:

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8

Memória!



Memória

Como podemos representar a memória?

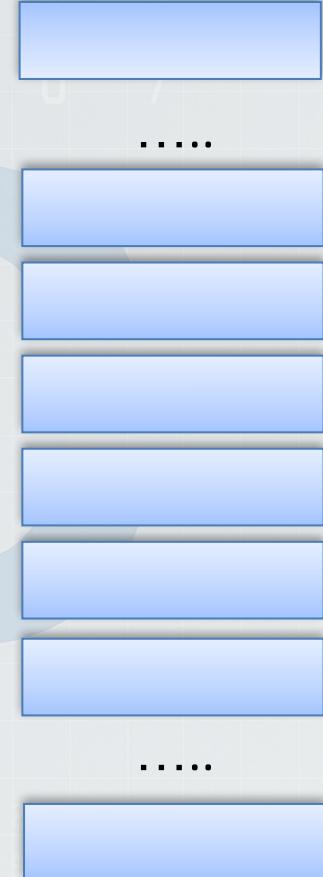
int matrix[3][3]:

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8

Início ->

Fim ->



Memória

Como podemos representar a memória?

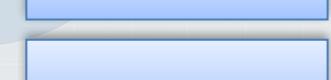
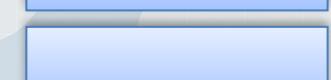
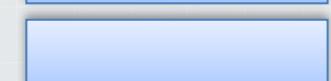
Início ->



.....

Parte da
Memória
Reservada
para o
Nosso
Programa

Fim ->



.....

5 9 3
2 6 8
4 0 7

int matrix[3][3];

9 2 3
7 8 5
2 2 8

Memória

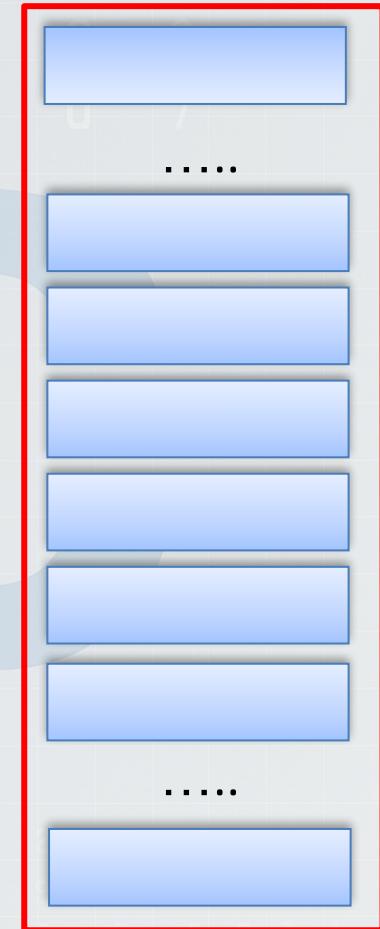
Como podemos representar a memória?

int matrix[3][3]:

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8

Valor/Informação



Memória

Como podemos representar a memória?

int matrix[3][3]:

5 9 3
2 6 8
4 0 7

Endereço	Valor
0	
.....	
k+0	
k+1	
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Como podemos representar a memória?

k é a posição inicial de memória reservado para o nosso programa, por exemplo, k pode ser 14, ou 1398 ou qualquer outra posição que foi liberada pelo Sistema Operacional!



Memória

```
int matrix[3][3]:
```

5	9	3
2	6	8
4	0	7

Endereço	Valor
0	
.....	
k+0	
k+1	
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Nosso programa

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	???
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	??? <- x
k+2	
k+3	
k+4	
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	??? <- x
k+2	
k+3	
k+4	???
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	??? <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

A posição de memória alocada/reservada para cada variável depende do Sistema Operacional e o Compilador.
Não existe garantia de alocação em sequência!

Endereço	Valor
0	
.....	
k+0	
k+1	??? <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()
2 {
3     int x;
4     int y;
5
6     x = 5;
7     scanf("%d", &y);
8     y = x + y;
9     printf("%d", y);
10
11    return 0;
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	??? <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Escreva 5 em k+1

Endereço	Valor
0	
.....	
k+0	
k+1	???
k+2	
k+3	
k+4	???
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5; // Line 6  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Escreva 5 em k+1

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5; // Line 6 highlighted with a red box  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

& é um operador unitário, assim como ++ e -- mas que retorna o endereço de memória da variável!

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

k+4

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

k+4 *scanf -> escreve o que o usuário digitou em k+4.*

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	??? <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

k+4 *scanf-> escreve o que o usuário digitou em k+4.*

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Lê (k+1)

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

```
1 int main ()
2 {
3     int x;
4     int y;
5
6     x = 5;
7     scai 5, "%d", &y );
8     y = x + y;
9     printf( "%d", y );
10
11    return 0;
12 }
```

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf ("%d", &y);  
8     y = x + y;  
9     printf ("%d", y);  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5
k+2	
k+3	
k+4	6
k+5	
.....	
N	

Lê (k+4)

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scalar(5, 6, &y);  
8     y = x + y;  
9     printf("%d", y);  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5
k+2	
k+3	
k+4	6
k+5	
.....	
N	

Memória

Executando....

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5 + 6; // 11  
7     scalar (x, y);  
8     y = x + y;  
9     printf("%d", y);  
10  
11    return 0;  
12 }
```

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5+6 = 11;  
7     scai . . . y);  
8     y = x + y;  
9     printf("%d", y);  
10  
11    return 0;  
12 }
```

Escreva 11 em k+4

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	6 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5+6 = 11;  
7     scai . . . y);  
8     y = x + y;  
9     printf("%d", y);  
10  
11    return 0;  
12 }
```

Escreva 11 em k+4

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	11 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	11 <- y
k+5	
.....	
N	

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	11 <- y
k+5	
.....	
N	

Memória

Executando....

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	11 <- y
k+5	
.....	
N	

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Lê k+4.

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

Endereço	Valor
0	
.....	
k+0	
k+1	5
k+2	
k+3	
k+4	11
k+5	
.....	
N	

11

Memória

Executando....

```
1 int main ()  
2 {  
3     int x;  
4     int y;  
5  
6     x = 5;  
7     scanf( "%d", &y );  
8     y = x + y;  
9     printf( "%d", y );  
10  
11    return 0;  
12 }
```

machado@laptop: ~

```
machado@laptop:~$ ./a.out  
6  
11machado@laptop:~$
```

Endereço	Valor
0	
.....	
k+0	
k+1	5 <- x
k+2	
k+3	
k+4	11 <- y
k+5	
.....	
N	

Endereço de Memória

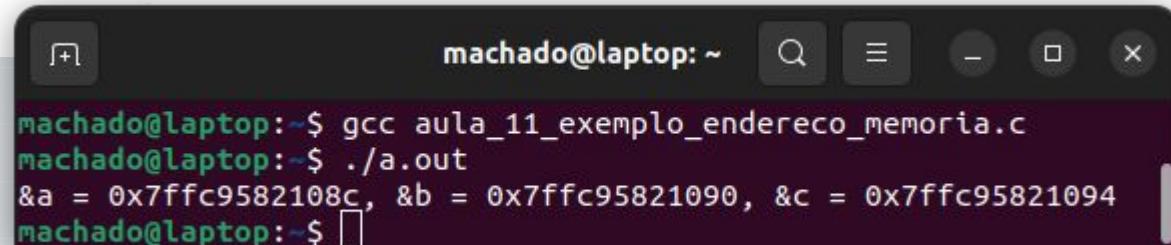
- O operador “&” quando aplicado sobre um identificador (nome de variável, por exemplo) retorna o seu endereço de memória!

Endereço de Memória

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     int b = 20;
7     int c = 30;
8     printf("&a = %p, &b = %p, &c = %p\n", &a, &b, &c);
9
10    return 0;
11 }
```

Endereço de Memória

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     int b = 20;
7     int c = 30;
8     printf("&a = %p, &b = %p, &c = %p\n", &a, &b, &c);
9
10    return 0;
11 }
```



A screenshot of a terminal window titled "machado@laptop: ~". The window shows the command "gcc aula_11_exemplo_endereco_memoria.c" being run, followed by "./a.out". The output displays the memory addresses of variables a, b, and c as 0x7ffc9582108c, 0x7ffc95821090, and 0x7ffc95821094 respectively.

```
machado@laptop:~$ gcc aula_11_exemplo_endereco_memoria.c
machado@laptop:~$ ./a.out
&a = 0x7ffc9582108c, &b = 0x7ffc95821090, &c = 0x7ffc95821094
machado@laptop:~$ 
```

Endereço de Memória

```
machado@laptop:~$ gcc aula_11_exemplo_endereco_memoria.c
machado@laptop:~$ ./a.out
&a = 0x7ffc9582108c, &b = 0x7ffc95821090, &c = 0x7ffc95821094
machado@laptop:~$ ./a.out
&a = 0x7fff0764f6c, &b = 0x7fff0764f70, &c = 0x7fff0764f74
machado@laptop:~$ ./a.out
&a = 0x7ffd1ff629c, &b = 0x7ffd1ff62a0, &c = 0x7ffd1ff62a4
machado@laptop:~$ ./a.out
&a = 0x7ffd47202b8c, &b = 0x7ffd47202b90, &c = 0x7ffd47202b94
machado@laptop:~$ ./a.out
&a = 0x7ffd11eeaaa9c, &b = 0x7ffd11eeaaa0, &c = 0x7ffd11eeaaa4
machado@laptop:~$ ./a.out
&a = 0x7ffd8d58973c, &b = 0x7ffd8d589740, &c = 0x7ffd8d589744
machado@laptop:~$ ./a.out
&a = 0x7feccc8c4c1c, &b = 0x7feccc8c4c20, &c = 0x7feccc8c4c24
machado@laptop:~$ █
```

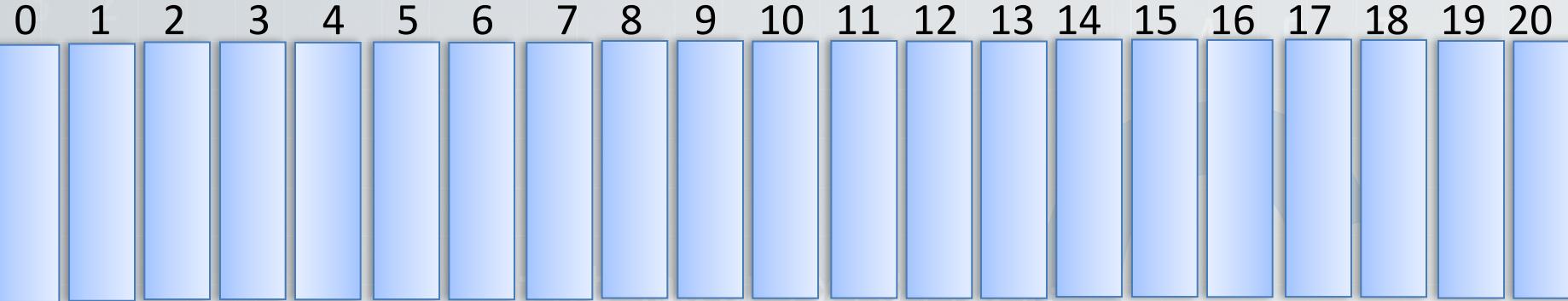
Ponteiros

- Um ponteiro é uma variável que contém (armazena) um **endereço de memória**
- Declaração:
tipo_dado *nome_ponteiro;
onde "*" indica que a variável é um ponteiro
- Ex: **int x;**
int *px; /* compilador *sabe* que px é ponteiro */
/* px é um *ponteiro para inteiro* */

Ponteiros

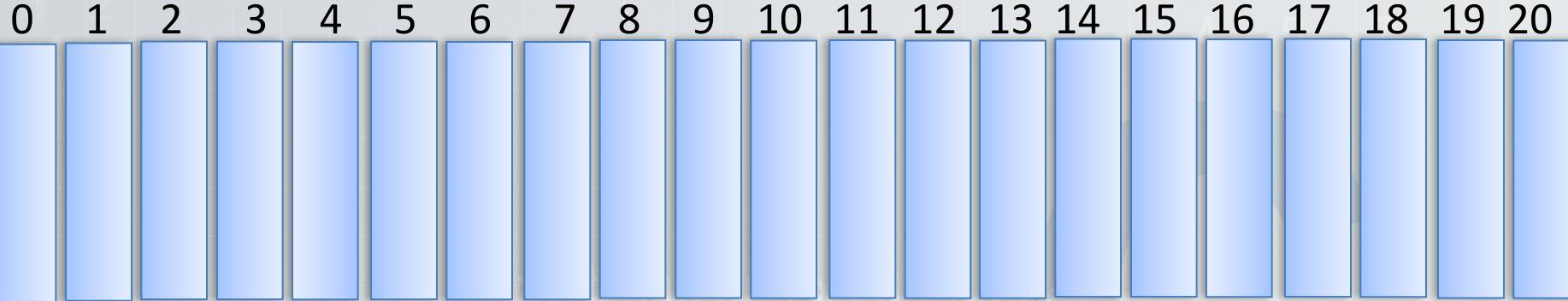
- Um ponteiro é uma variável que contém (armazena) um **endereço de memória**
- Declaração:
tipo_dado *nome_ponteiro;
onde "*" indica que a variável é um ponteiro
- Ex: **int x;**
int *px; /* compilador *sabe* que px é ponteiro */
/* px é um *ponteiro para inteiro* */

O que é um ponteiro?



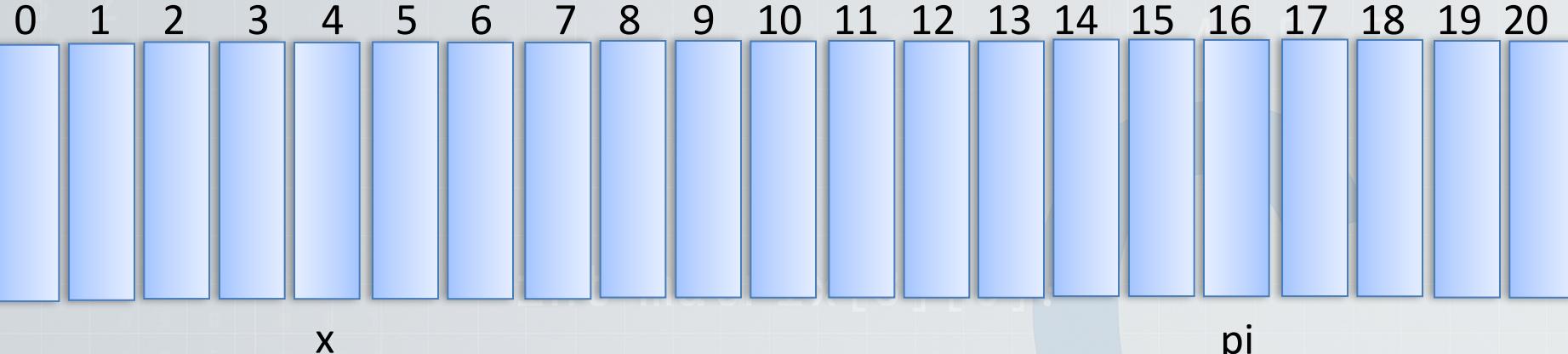
```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

O que é um ponteiro?



```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

O que é um ponteiro?

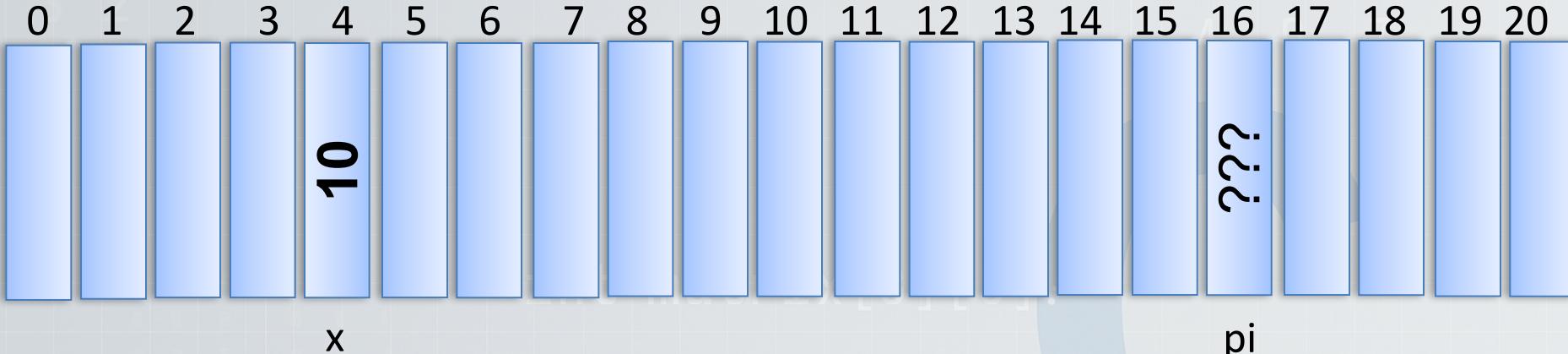


```
int x = 10, *pi;
```

```
pi = &x;
```

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

O que é um ponteiro?

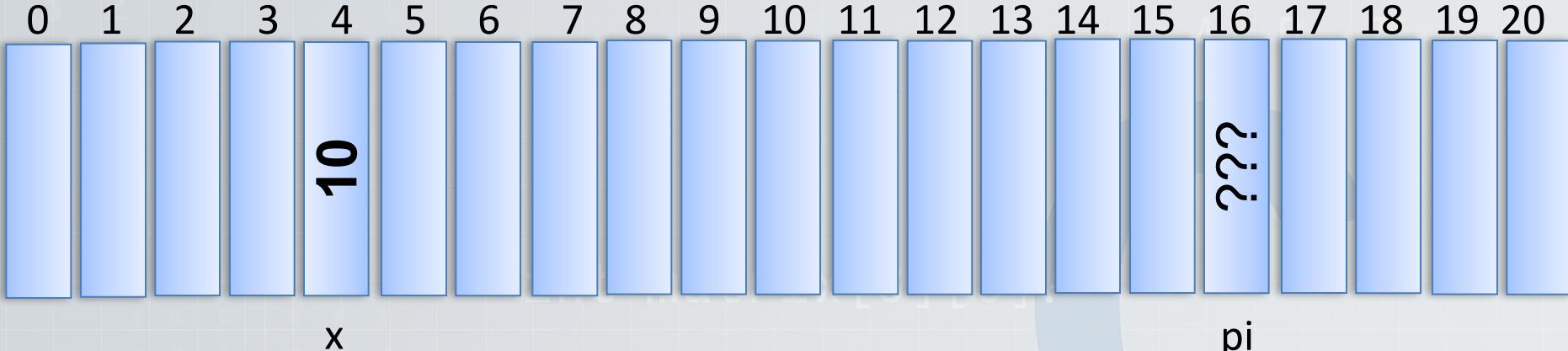


```
int x = 10, *pi;
```

```
pi = &x;
```

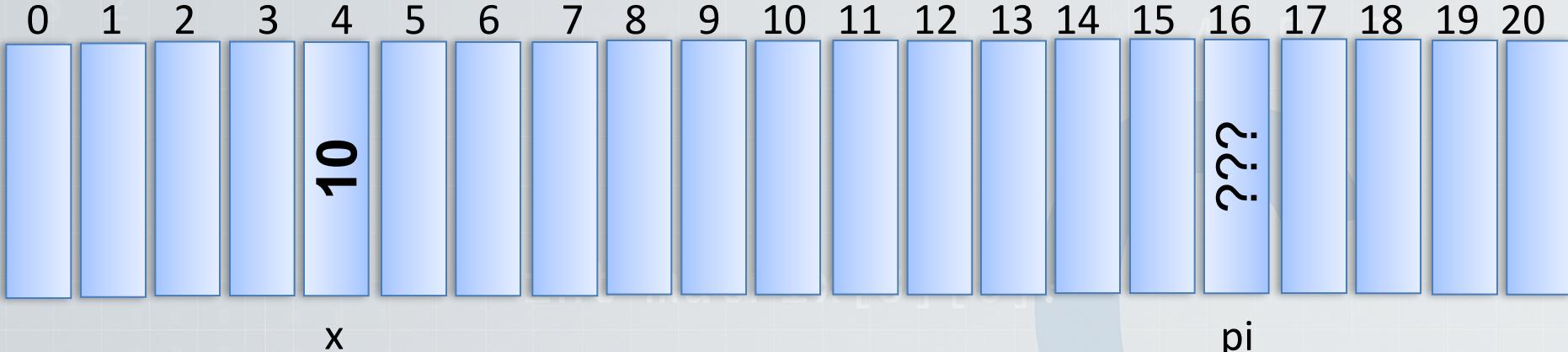
```
printf("&x: %p pi: %p", &x, pi);
```

O que é um ponteiro?



```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

O que é um ponteiro?

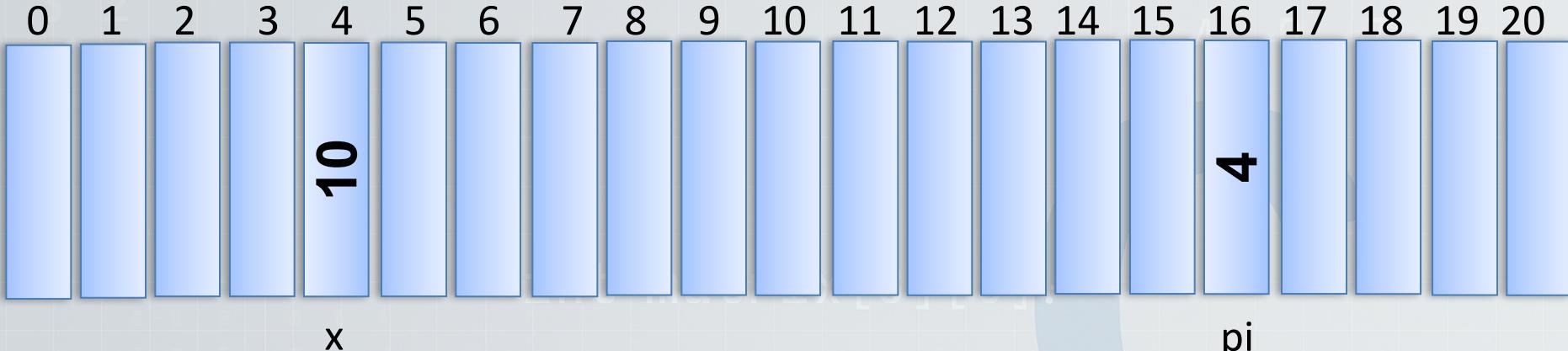


```
int x = 10, *pi;  
pi = &x;
```

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

O que acontece na memória?

O que é um ponteiro?

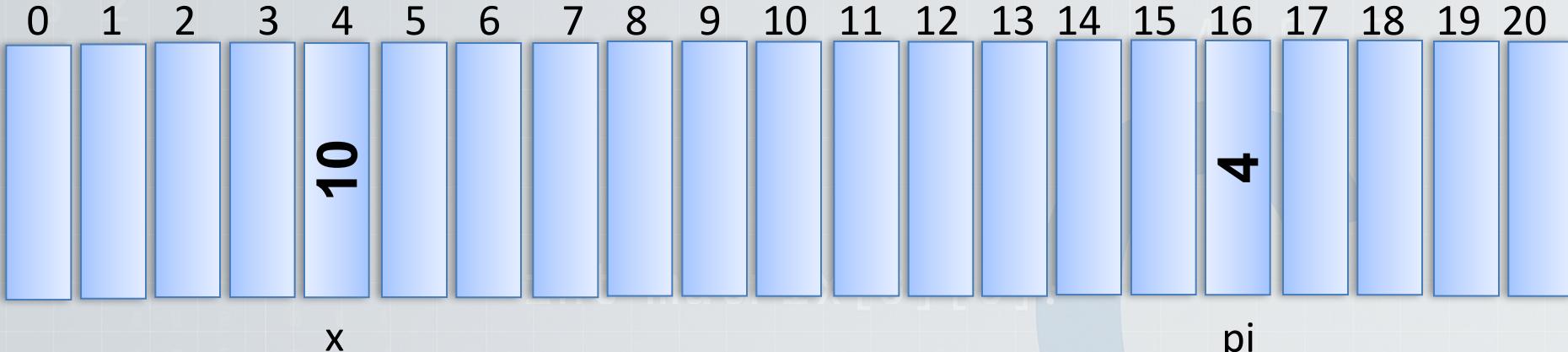


```
int x = 10, *pi;  
pi = &x;
```

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

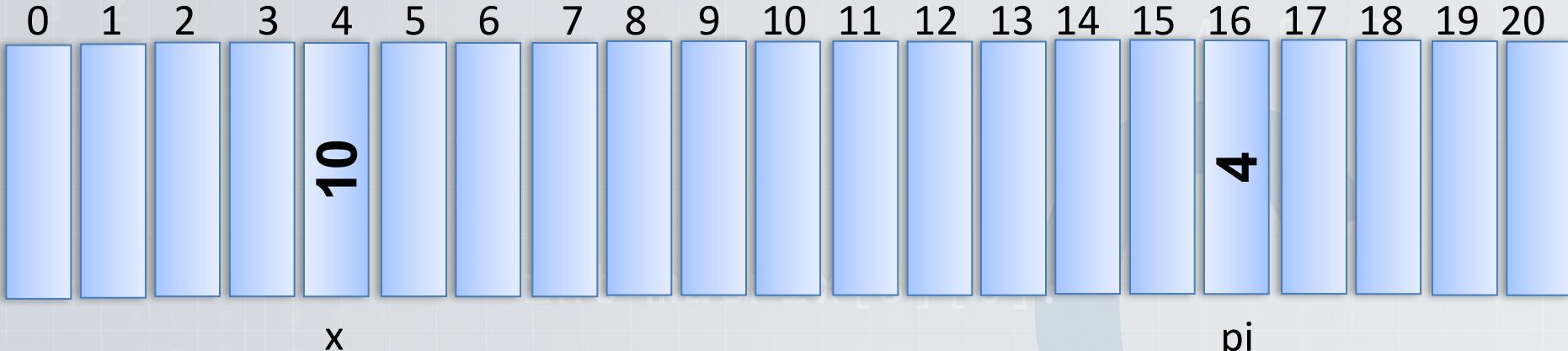
O que acontece na memória?

O que é um ponteiro?



```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

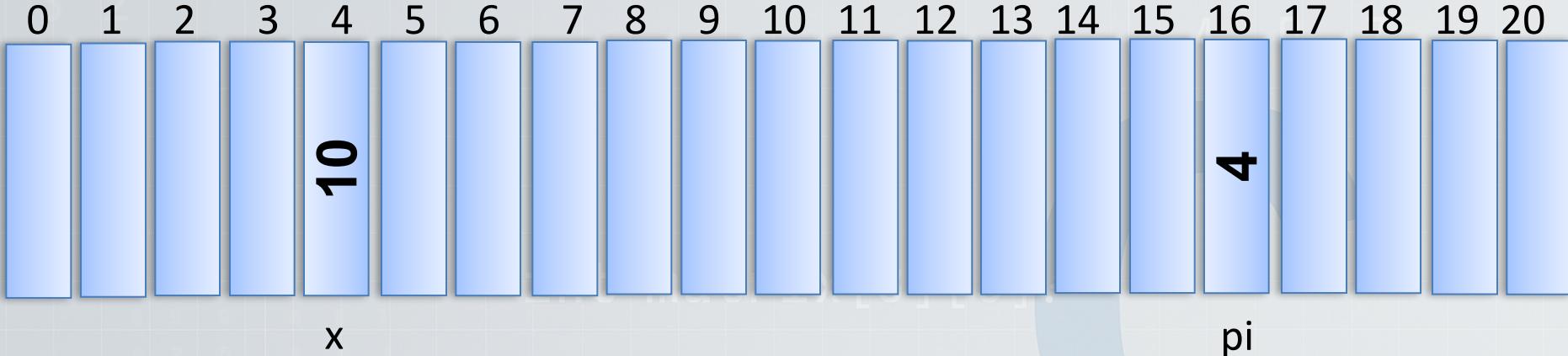
O que é um ponteiro?



```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

O que será impresso na tela?

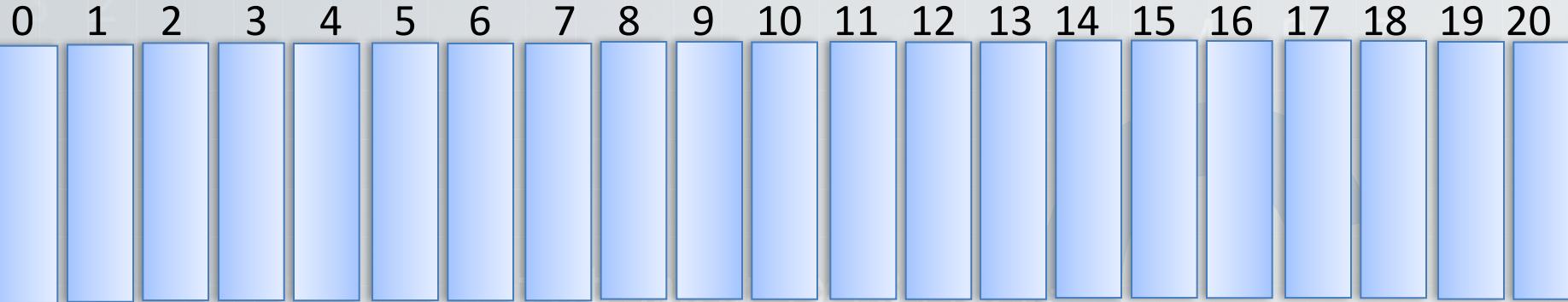
O que é um ponteiro?



```
int x = 10, *pi;  
pi = &x;  
printf("&x: %p pi: %p", &x, pi);
```

Saída em tela:
&x: 0x4 pi: 0x4

Como usar um ponteiro?



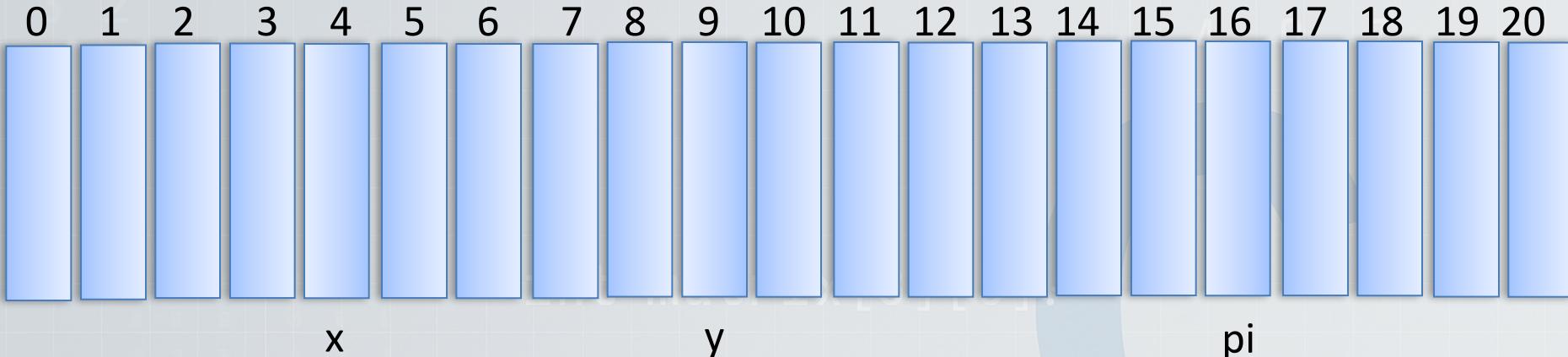
```
int x = 10, *pi, y;
```

```
pi = &x;
```

5	9	3
2	6	8
4	0	7

```
y = *pi;
```

Como usar um ponteiro?

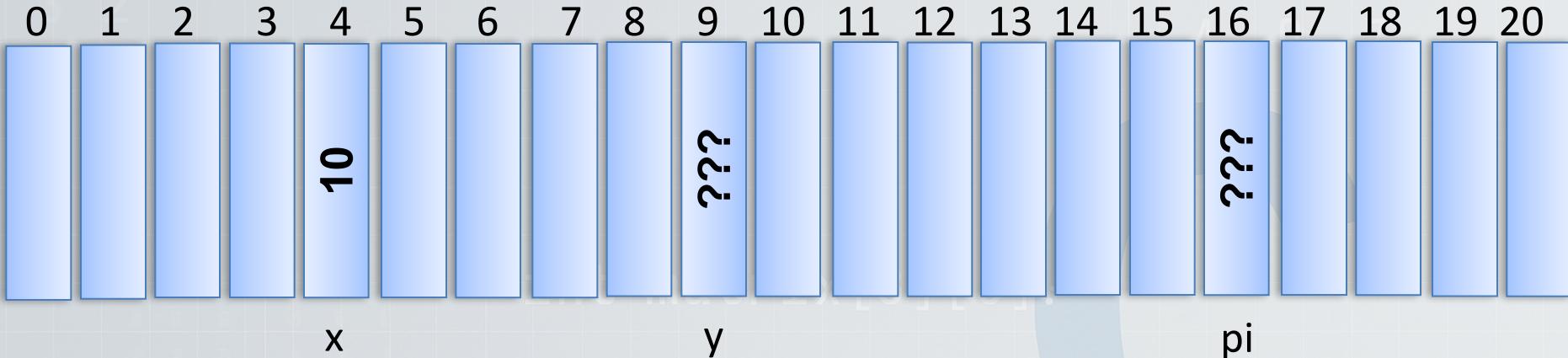


```
int x = 10, *pi, y;
```

```
pi = &x;
```

```
y = *pi;
```

Como usar um ponteiro?

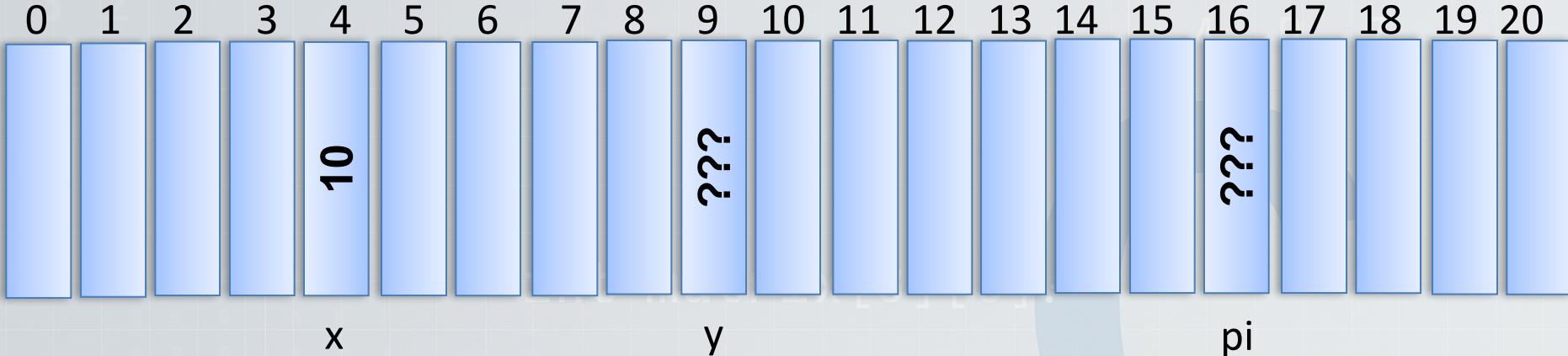


```
int x = 10, *pi, y;
```

```
pi = &x;
```

```
y = *pi;
```

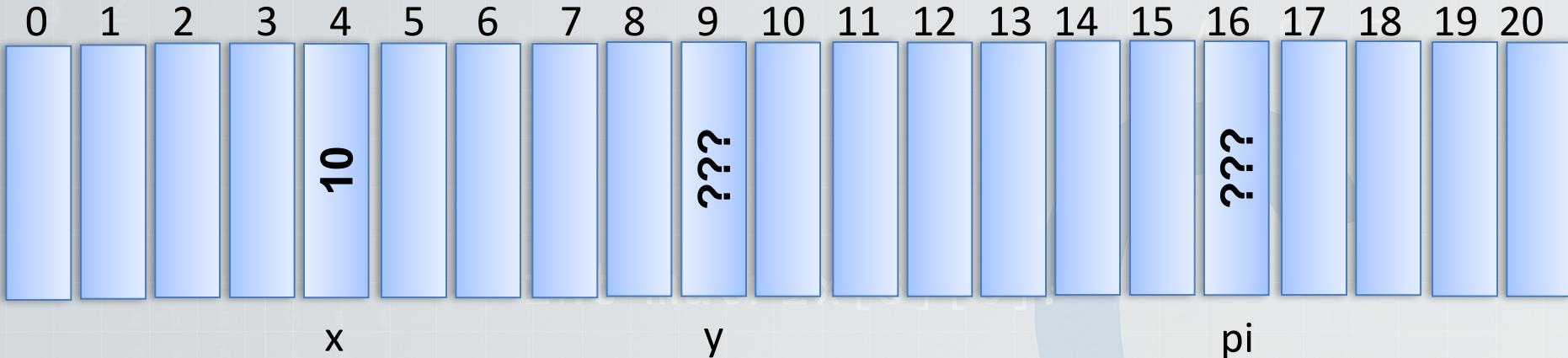
Como usar um ponteiro?



```
int x = 10, *pi, y;  
pi = &x;
```

```
y = *pi;
```

Como usar um ponteiro?

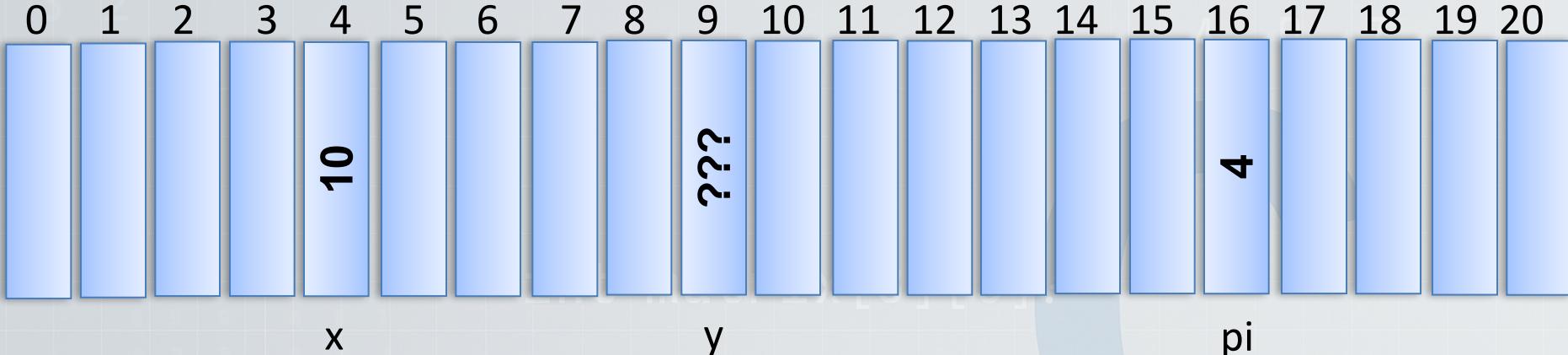


```
int x = 10, *pi, y;  
pi = &x;
```

```
y = *pi;
```

O que acontece com a memória?

Como usar um ponteiro?

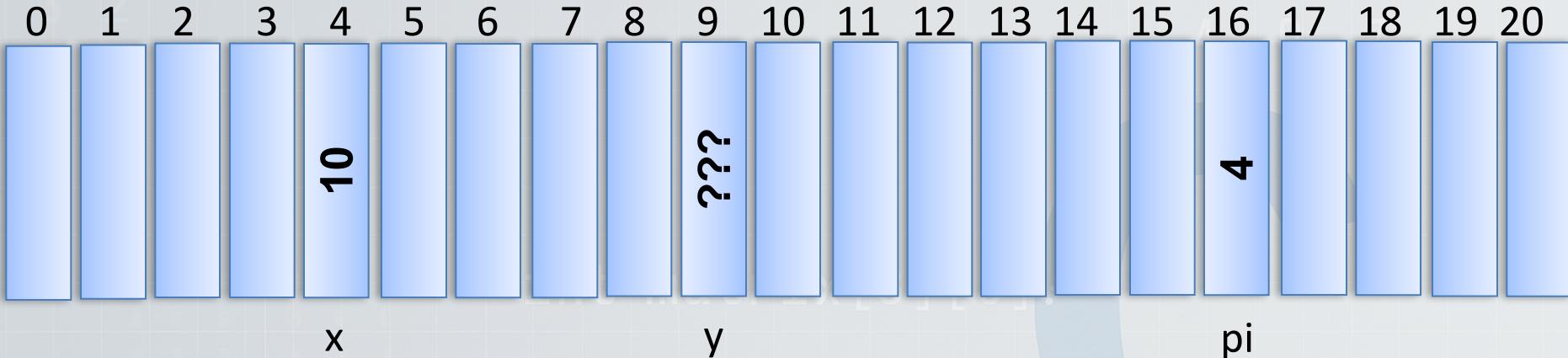


```
int x = 10, *pi, y;  
pi = &x;
```

```
y = *pi;
```

O que acontece com a memória?

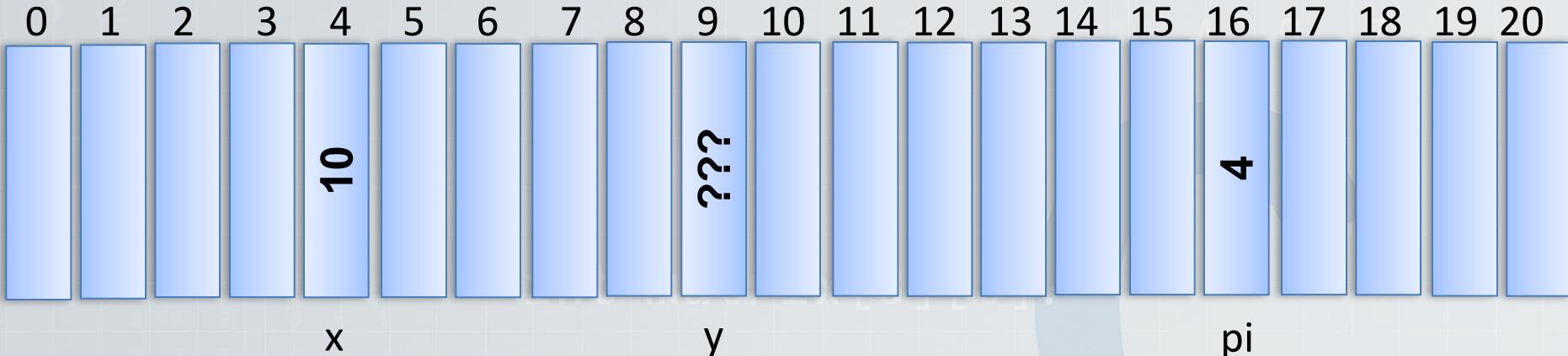
Como usar um ponteiro?



```
int x = 10, *pi, y;  
pi = &x;
```

y = *pi;

Como usar um ponteiro?

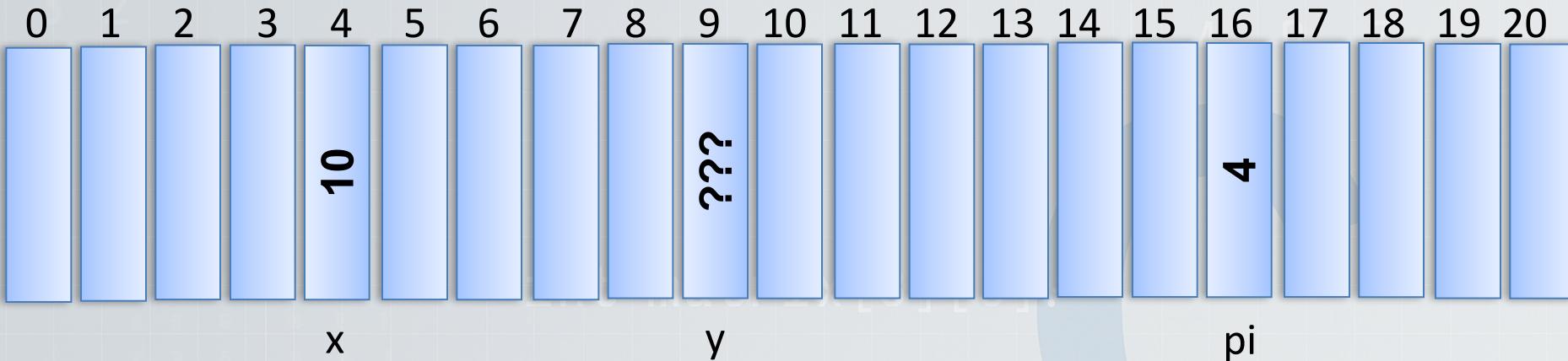


```
int x = 10, *pi, y;  
pi = &x;
```

y = *pi;

O operador “*” quando aplicado
sobre um ponteiro retorna
o dado apontado

Como usar um ponteiro?



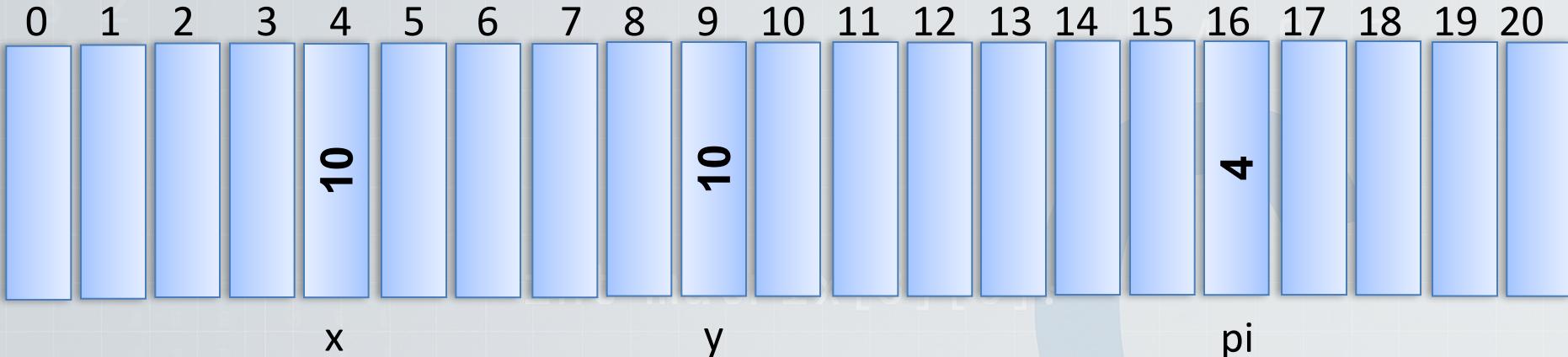
```
int x = 10, *pi, y;  
pi = &x;
```

y = *pi;

O que acontece com a memória?

O operador “*” quando aplicado sobre um ponteiro retorna o dado apontado

Como usar um ponteiro?



```
int x = 10, *pi, y;  
pi = &x;
```

y = *pi;

O que acontece com a memória?

O operador “*” quando aplicado sobre
um ponteiro retorna
o dado apontado

Utilizando Ponteiros

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

```
int main() {  
    int x = 10;  
    int *pi;  
    pi = &x;  
    (*pi)++;  
    printf("%d", x);  
    return 0;  
}
```



Utilizando Ponteiros

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

```
int main() {  
    int x = 10;  
    int *pi;  
    pi = &x; /* Qual o valor de *pi ? */  
    (*pi)++; /* Qual o valor de *pi ? */  
    printf("%d", x);  
    return 0;  
}
```

ao alterar ***pi** estamos alterando o conteúdo de **x**



Utilizando Ponteiros

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

```
int main() {  
    int x = 10;  
    int *pi;  
    pi = &x; /* *pi é igual a 10 */  
    (*pi)++; /* *pi é igual a 11 */  
    printf("%d", x);  
    return 0;  
}
```

ao alterar ***pi** estamos alterando o conteúdo de **x**



Utilizando Ponteiros

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

```
int main() {
    int x = 10;
    int *pi, *pj;

    pi = &x;
    pj = pi;
    (*pi)++;
    (*pj)++;
    printf("%d", x);
    return 0;
}
```

```
int matrix[3][3]:
```



Utilizando Ponteiros

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

```
int main() {
    int x = 10;
    int *pi, *pj;
    pi = &x;          /* *pi == ?? */
    pj = pi;          /* *pj == ?? */
    (*pi)++;         /* (*pi, *pj, x) == ?? */
    (*pj)++;         /* (*pi, *pj, x) == ?? */
    printf("%d", x); /* Escreverá ?? */
    return 0;
}
```

```
int matrix[3][3]:
```



Utilizando Ponteiros

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

```
int main() {
    int x = 10;
    int *pi, *pj;
    pi = &x;          /* *pi == 10 */
    pj = pi;          /* *pj == 10 */
    (*pi)++;         /* (*pi, *pj, x) == 11 */
    (*pj)++;         /* (*pi, *pj, x) == 12 */
    printf("%d", x); /* Escreverá 12 */
    return 0;
}
```

```
int matrix[3][3]:
```



PONTEIROS & ARRAYS

```
int matrix[3][3]:
```



Referenciando Arrays

- Pode-se referenciar os elementos de um array através de ponteiros

```
int matrix[3][3]:  
• Ex: float m[] = { 1.0, 3.0, 5.75, 2.345 };  
float *pf;  
pf = &m[2];  
printf("%f", *pf); /* Quanto será impresso? */
```

Referenciando Arrays

- Pode-se referenciar os elementos de um array através de ponteiros

```
int matrix[3][3]:  
• Ex: float m[] = { 1.0, 3.0, 5.75, 2.345 };  
float *pf;  
pf = &m[2];  
printf("%f", *pf); /* Escreve 5.75 */
```

Referenciando Elementos

- Pode-se utilizar ponteiros e colchetes:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
```

```
float *pf;
```

```
pf = &m[2];    int matrix[3][3]:  
printf("%f", pf[0]); /* ==> 5.75 */
```

- Note que o valor entre colchetes é o deslocamento a ser considerado a partir do endereço de referência
 - $pf[n]$ => indica n -ésimo elemento a partir de pf

PONTEIROS & PARÂMETROS DE FUNÇÕES

```
int matrix[3][3]:
```



Passagem de Informações

- Argumentos passados **por referência**
 - Quando chamada, a função passa a referenciar (apontar) a variável informada
 - Portanto o processo consiste em informar o endereço da variável para o que o parâmetro formal possa referenciá-lo.
 - Os argumentos deixam de existir após a execução do método, porém as variáveis informadas e que foram referenciadas permanecem (pois não são dadas por este bloco de comandos).

Exemplo

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

```
void funcPorValor(int a);  
void funcPorRefer(int *b);
```

```
int main () {  
    int x = 0, y = 0;  
  
    funcPorValor(y);  
    printf("%d %d\n", x, y);  
  
    funcPorRefer(&y);  
    printf("%d %d\n", x, y);  
}
```

```
void funcPorValor(int a){  
    a = 1;  
}
```

```
void funcPorRefer(int *b){  
    *b = 2; /* ... o conteúdo apontado  
              por b recebe 2 */  
}
```

- Note que as variáveis *x* e *y* são locais a função *main*, enquanto os parâmetros *a* e *b* são locais a *funcPorValor* e *funcPorRefer*, respectivamente.

Exemplo

```
#include <stdio.h>

void funcvet(int v[]);

int main () {
    int i, vetor[5]={0, 2, 0, 2, 0};

    funcvet(vetor);

    for (i=0; i<5; i++)
        printf("%"d " ,vetor[i]); /* Escreve ?? */
}
```

```
void funcvet(int *v){

    int i;
    for(i=0; i<5; i++)
        if (v[i]==0)
            v[i]=1;
}
```

Exemplo

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

```
void funcvet(int v[]);
```

```
int main () {
```

```
    int i, vetor[5]={0, 2, 0, 2, 0};
```

```
    funcvet(vetor);
```

```
    for (i=0; i<5; i++)
```

```
        printf("%d ", vetor[i]); /* Escreve 1 2 1 2 1*/
```

```
}
```

```
void funcvet(int *v){
```

```
void funcvet(int v[]){
```

```
    int i;
```

```
    for(i=0; i<5; i++)
```

```
        if (v[i]==0)
```

```
            v[i]=1;
```

```
}
```

Operações Válidas Sobre Ponteiros

É válido

- *somar ou subtrair um inteiro a um ponteiro*
 $(pi \pm int)$
- *incrementar ou decrementar ponteiros*
 $(pi++, pi--)$
- *subtrair ponteiros (produz um inteiro)*
 $(pf - pi)$
- *comparar ponteiros*
 $(>, >=, <, <=, ==)$

Não é válido

- *somar ponteiros*
 $(pi + pf)$
- *multiplicar ou dividir ponteiros*
 $(pi * pf, pi / pf)$
- *operar ponteiros com double ou float*
 $(pi \pm 2.0)$

Exercícios

- 1) Faça um programa que leia 2 valores inteiros e chame uma função que receba estas 2 variáveis e troque o seu conteúdo, ou seja, esta rotina é chamada passando duas variáveis A e B por exemplo, e após a execução da rotina A conterá o valor de B e B terá o valor de A.

- 2) Escreva uma função que dado um número real passado como parâmetro, retorne a parte inteira e a parte fracionária deste número através de ponteiros.

Exercícios

- 3) Crie uma função que receba um vetor A com 20 inteiros e retorne:
- um vetor B com apenas os números pares de A
 - o número de elementos de B
 - um vetor C com apenas os números ímpares de A
 - o número de elementos de C

Crie um programa que lê o vetor A, chama a função, recebe e imprime o conteúdo dos vetores B e C.

Exercícios

- 4) Faça um programa que acha o maior e o menor inteiro dentro de um vetor de 10 inteiros.

Obs: usar apenas as variáveis a seguir:

```
int v[10], i, *maior, *menor;
```

5 9 3
2 6 8
4 0 7

9 2 3
7 8 5
2 2 8

2 2 0 8 h 1
t m 5 6 5 2 0
l s 9 6 2 7



SSC0501 - Introdução à Ciência de Computação I

Obrigado pela atenção!!

Prática!

Código Matrícula: R4SM

<https://runcodes.icmc.usp.br/offering/view/83>

[run.codes] Menu Professor matheus.m.santos@icmc.usp.br Hora do Servidor: 14/03/2022 18:46:06

Home > SSC0501

SSC0501 - Introdução à Ciência de Computação I

Professores: Matheus Machado dos Santos
Turma: 2025101
Universidade: USP
Ativa até: 21/07/2025

Código de Matrícula **R4SM**

Novo Exercício

Ver Notas

Enviar E-mail

Exportar Tabela de Notas

No.	Exercício	Status	Casos Corretos	Nota	Entregas	Participantes	Prazo de Entrega	Ações		
1	Hello World	Finalizado	1/1	10.00	78	43/46	14/03/2025 21:00:00			
2	1.01 Maior número	Não Entregue	0/6	0	0	0/46	19/03/2025 23:59:59			
3	1.02 Par ou ímpar	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59			
4	1.03 Positivo, negativo ou zero	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59			
5	1.04 Maior de três números	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59			

`int main():`

Extra

Endereço de Memória

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     int b = 20;
7     int c = 30;
8     printf("&a - &b = %ld, &a - &c = %ld\n", &a - &b, &a - &c);
9
10    return 0;
11 }
```

Endereço de Memória

Endereço de Memória

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     int b = 20;
7     int c = 30;
8     printf("&a - &b = %ld, &a - &c = %ld\n", (void*)&a - (void*)&b, (void*)&a - (void*)&c);
9
10    return 0;
11 }
```

Endereço de Memória