



Ponteiros em C (Parte 01 - Introdução)

Igor Yepes



Variáveis e memória

	Um bit (0 ou 1)								Um byte (8 bits)							
0	0	1	0	0	1	0	0	0								
1	1	1	0	1	0	0	0	1								
2	1	0	0	0	1	1	0	0								
...																
...																
...																
56789																
56790																
56791																
56792																
56793																
56794																
56795																
...																

Tipo	sizeof	Faixa de valores
char	1	-128...127 ou 0...255 (depende da implementação)
unsigned char	1	0...255
signed char	1	-128...127
int	2 ou 4	-32.768...32.767 (32 bits) -2.147.483.648...2.147.483.647 (64 bits)
unsigned int	2 ou 4	0...65.535 (32 bits) ou 0...4.294.967.295 (64 bits)
short	2	-32.768...32.767
unsigned short	2	0...65.535
long	4	-2.147.483.648...2.147.483.647
unsigned long	4	0...4.294.967.295

Saber o tamanho em bytes de uma variável **x**:
sizeof(x)



Variáveis e memória

Um bit (0 ou 1)

Um byte (8 bits)

0	0	1	0	0	1	0	0	0								
1	1	1	0	1	0	0	0	1								
2	1	0	0	0	1	1	0	0								
...																
...																
...																
56789																
56790																
56791																
56792																
56793																
56794																
56795																
...																

x vai ocupar 4 bytes reservados na memória

```
1 #include<stdio.h>
2
3 int main ()
4 {
5     int x = 16;
6     printf("Valor de x: %i\n", x);
7     printf("Endereço de x: %p\n",
8           &x);
9 }
10
11
12
13
```

O & informa que deve ser fornecido o endereço de x, não o valor

Run Succeeded



Variáveis e memória

```
1 #include<stdio.h>
2
3 int main ()
4 {
5     int x = 16;
6     printf("Valor de x: %i\n", x);
7     printf("Endereço de x: %p\n",
8           &x);
9 }
10
11
12
13
```

& = endereço de...

Run Succeeded

Watchpoint created: Watchpoint 1: addr = 0x7ffefbfff55c size = 4 state = enabled type = w declare @ '/Users/igor/Library/Application Support/CodeRunner/Unsaved/Untitled 2.c:5' watchpoint spec = 'x' new value: 16

Valor de x: 16
Endereço de x: 0x7ffefbfff55c

Watchpoint 1 hit:
old value: 16
new value: 0

Symbol Spaces: 4 Line 14, Column 1

Ponteiro - declaração

VARIÁVEL QUE CONTÉM O ENDEREÇO DE OUTRA VARIÁVEL

```
1 #include<stdio.h>
2
3 int main ()
4 {
5     int x = 10;
6     int *px = &x;
7     printf("Valor 1 de x: %i\n", x);
8     *px = 5;
9     printf("Valor 2 de x: %i\n", x);
10 }
```

Se precisar inicializar o ponteiro em linha separada, muda um detalhe (*)...

Valor 1 de x: 10
Valor 2 de x: 5

*int x = 10;
int *px;
px = &x;*

Ponteiros

Declaração: `int x = 10;` ou `int x = 10;`
`int *px = &x;` `int *px;`
`px = &x;`

```
1 #include<stdio.h>
2
3 int main () {
4     int x = 10;
5     int *px = &x;
6
7     printf("Valor de x: %i\n", x);
8     printf("Endereço de x: %d\n", &x);
9     printf("Valor de px: %d\n", *px);
10    printf("Endereço apontado por px: %d\n", px);
11    printf("Endereço de px: %d\n", &px);
12 }
```

Valor de x: 10
Endereço de x: -345995604
Valor de px: 10
Endereço apontado por px: -345995604
Endereço de px: -345995616

ATENÇÃO:

- Sem * antes do ponteiro, acessa o endereço apontado
- Com * antes do ponteiro, acessa o conteúdo do endereço apontado

Ponteiro

0	0	1	0	0	1	0	0	0
1	1	1	0	1	0	0	0	1
2	1	0	0	0	1	1	0	0
...								
...								
...								
56789								
56790								
56791								
56792								
56793								
56794								
56795								
...								

```
1 #include<stdio.h>
2
3 int main ()
4 {
5     int x = 10;
6     int *px = &x;
7     printf("Valor 1 de x: %i\n", x);
8     *px = 5;
9     printf("Valor 2 de x: %i\n", x);
10 }
```

x vai ocupar 4 bytes reservados na memória

O ponteiro px vai apontar para o endereço inicial de x, sabendo que ele ocupa 4 bytes

Valor 1 de x: 10
Valor 2 de x: 5

Ponteiros e funções

```
1 #include<stdio.h>
2
3 void altera(int x){
4     printf("Valor de x: %i\n", ++x);
5 }
6
7 int main () {
8     int x = 1;
9     printf("Valor de x: %i\n", x);
10    altera(x);
11    printf("Valor de x: %i\n", x);
12 }
```

Valor de x: 1
Valor de x: 2
Valor de x: 1

A variável x foi alterada localmente na função "altera", mas seu valor permanece inalterado em "main()".

Ponteiros e funções

```
1 #include<stdio.h>
2
3 void altera(int *x){
4     printf("Valor de x: %i\n", ++*x);
5 }
6
7 int main () {
8     int x = 1;
9     printf("Valor de x: %i\n", x);
10    altera(&x);
11    printf("Valor de x: %i\n", x);
12 }
```

Valor de x: 1
Valor de x: 2
Valor de x: 2

A variável x foi alterada em "main()", à partir da função "altera".

SINTAXE:

- A função recebe o endereço para um ponteiro
- A função é chamada passando o endereço da variável a ser manipulada

Ponteiros e funções

```
1 #include<stdio.h>
2
3 void altera(int *px){
4     printf("Valor de x: %i\n", ++*px);
5 }
6
7 int main () {
8     int x = 1;
9     int *px = &x;
10    printf("Valor de x: %i\n", x);
11    altera(px);
12    printf("Valor de x: %i\n", x);
13 }
```

Valor de x: 1
Valor de x: 2
Valor de x: 2

CASO NECESSITE ACESSAR O PONTEIRO TAMBÉM NA FUNÇÃO MAIN(), M ESTE PODE SER CRIADO E PASSADO COMO UM ARGUMENTO PARA A FUNÇÃO

Ponteiros - utilidade

- ★ Atualização de valores que são utilizados em várias partes do software dentro de funções
- ★ A variável do tipo vetor nada mais é que um ponteiro para o seu primeiro elemento
- ★ Alocação dinâmica de memória só é possível graças aos ponteiros
- ★ Com desenvolvimento de sistemas mais complexos, mais e mais problemas que surgem são resolvidos com ponteiros
- ★ Ao usar grandes quantidades de memória, se ela não for alocada dinamicamente, pode comprometer a pilha que a linguagem usa para chamada de funções e outras atividades