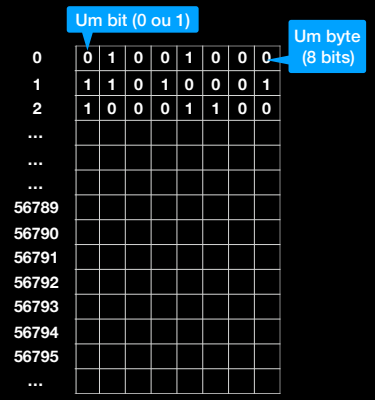
Ponteiros em C
(Parte 01 - Introdução)

Igor Yepes

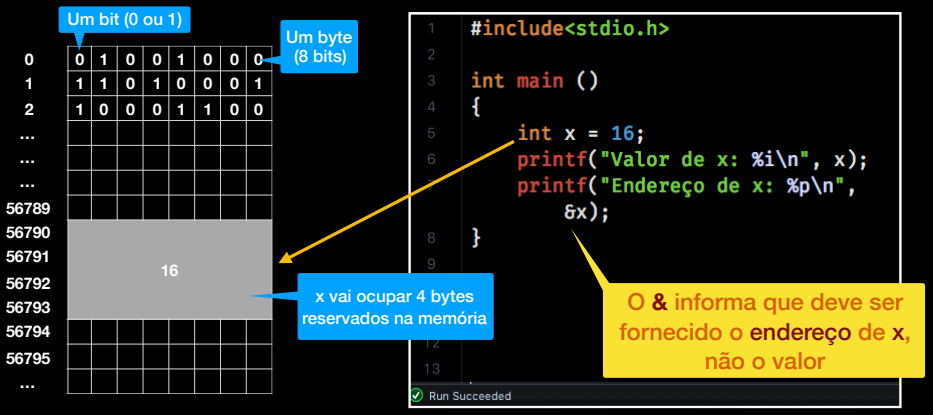
Variáveis e memória



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



Variáveis e memória

Ponteiro - declaração

VARIÁVEL QUE CONTÉM O ENDEREÇO DE OUTRA VARIÁVEL

```c
#include<stdio.h>

int main ()
{
    int x = 10;
    int *px = &x;
    printf("Valor 1 de x: %i\n", x);
    *px = 5;
    printf("Valor 2 de x: %i\n", x);
}
```

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

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

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

Ponteiros

Declaração:

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

ou

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

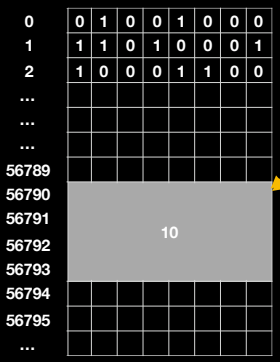
```c
#include<stdio.h>

int main () {
    int x = 10;
    int *px = &x;

    printf("Valor de x: %i\n", x);
    printf("Endereço de x: %d\n", &x);
    printf("Valor de px: %d\n", *px);
    printf("Endereço apontado por px: %d\n", px);
    printf("Endereço de px: %d\n", &px);
}
```

```
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




```c
#include<stdio.h>

int main ()
{
    int x = 10;
    int *px = &x;
    printf("Valor 1 de x: %i\n", x);
    *px = 5;
    printf("Valor 2 de x: %i\n", x);
}
```

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

```c
#include<stdio.h>

void altera(int x){
    printf("Valor de x: %i\n", ++x);
}

int main () {
    int x = 1;
    printf("Valor de x: %i\n", x);
    altera(x);
    printf("Valor de x: %i\n", x);
}
```

```
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

```c
#include<stdio.h>

void altera(int *x){
    printf("Valor de x: %i\n", ++*x);
}

int main () {
    int x = 1;
    printf("Valor de x: %i\n", x);
    altera(&x);
    printf("Valor de x: %i\n", x);
}
```

```
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

```c
#include<stdio.h>

void altera(int *px){
    printf("Valor de x: %i\n", ++*px);
}

int main () {
    int x = 1;
    int *px = &x;
    printf("Valor de x: %i\n", x);
    altera(px);
    printf("Valor de x: %i\n", x);
}
```

```
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