

Linguagens de Programação 1

```
printf("\aAula #%d\n", (5 % 2) * 6);
```

Message of the Day


"I shall, in due time, be a Poet."


— Ada Lovelace

Ada Lovelace (1815–1852)

 Primeira programadora de computadores

Escreveu o primeiro algoritmo para ser processador por uma máquina

 É um símbolo das mulheres em STEM (science, technology, engineering and mathematics) e inspirou gerações de mulheres. Em sua homenagem, o dia de Ada Lovelace é celebrado anualmente para reconhecer mulheres na ciência da computação!

 a linguagem de programação Ada foi criada em homenagem a ela pelo Departamento de Defesa dos Estados Unidos e formalizada em 1980

Conteúdo

O que vamos ver hoje?

Apontadores: Passagem de parâmetros por valor 


Apontadores: Apontadores 


Apontadores: Passagem de parâmetros por referência 

O que acontece na memória ao declarar uma variável?


A memória **RAM** (Random Access Memory)
Permite gravar e ler informação guardada (bits)
através de um endereço

Podemos pensar na memória como sendo uma
tabela enorme em que cada linha é identificada
por um endereço (nr da linha) e contém 1byte de
dados (8bits)

 A tabela exemplifica o modelo de memória. Os
Endereços são definidos no momento em que o
programa corre.

 Modelo da memória RAM:

Endereço	Conteúdo	Identificador
0	<1 byte>	
1	<1 byte>	
...	...	
1027	<1 byte>	
1028	<1 byte>	
1029	<1 byte>	
...	...	

 O conteúdo correspondente a cada
endereço pode armazenar 1 byte (= 8 bits).

Como o computador armazena variáveis?

```
char letra = 'a'; // Código ASCII = 97
```

- 1** O processador encontra um **espaço livre** na memória.
- 2** O programa regista que `letra` está, por exemplo, no endereço **256**.
- 3** Sempre que precisarmos de `letra`, o programa vai à RAM nesse endereço.

Endereço	Conteúdo	Identificador
...	...	
255	...	
256	'a' (97)	char letra
257	...	
...	...	

 O endereço de memória é a "casa" da variável! 



Endereço vs Conteúdo


```
char letra = 'a';
```

 Cada variável tem:

- ✓ Identificador (nome)
- ✓ Conteúdo (valor)
- ✓ Endereço (localização na RAM)

Endereço	Conteúdo	Identificador
...	...	
256	'a' (97)	char letra
...	...	

 O endereço **muda** em cada execução! 

 Conseguimos saber qual o endereço com o operador **referência** `&`

```
printf("End. = %p\n", (void*)&letra);
```

O que acontece ao declarar uma variável com mais do que 1 byte?


O **sistema operativo** (ou o compilador, se não houver SO) decide **onde** armazená-la na RAM.

O tamanho da variável depende do seu **tipo de dados**:

`char` → 1 byte

`int` → 4 bytes

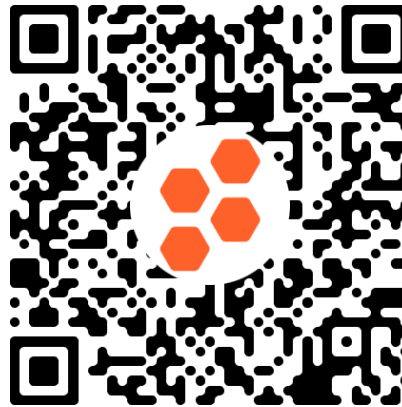
`double` → 8 bytes

 A tabela mostra como as variáveis podem ser distribuídas na memória. Os endereços reais são atribuídos **dinamicamente**.

 Modelo de memória ao declarar variáveis:

Endereço	Conteúdo	Identificador
...	...	
1000	14	<code>int x</code> (1º byte)
1001	00	<code>int x</code> (2º byte)
1002	00	<code>int x</code> (3º byte)
1003	00	<code>int x</code> (4º byte)
1004	50.0	<code>double d</code> (8 bytes)
1005	...	
...	...	

? Quizz - Endereços de variáveis



No campo nome devem colocar o número de aluno 2XXXXXXX.

? Quizz - Endereços de variáveis

Qual o endereço e conteúdo das variáveis?
Admita que ela estão seguidas na memória no sentido crescente. (pode não ser sempre assim na prática)

```
int a = 10;
char b = 'x';
int c = 20;
double d = 50.0;
char e = 91;
```

Endereço	Conteúdo	Identificador
100	10	a
		b
		c
		d
		e

? Quizz - Endereços de variáveis

Qual o endereço e conteúdo das variáveis?

Admita que ela estão seguidas na memória no sentido crescente. (pode não ser sempre assim na prática)

```
int a = 10;
char b = 'x'
int c = 20;
double d = 50.0;
char e = 91;
```

Endereço	Conteúdo	Identificador
100	10	a
104	'x'	b
		c
		d
		e

? Quizz - Endereços de variáveis

Qual o endereço e conteúdo das variáveis?
Admita que ela estão seguidas na memória no sentido crescente. (pode não ser sempre assim na prática)

```
int a = 10;
char b = 'x';
int c = 20;
double d = 50.0;
char e = 91;
```

Endereço	Conteúdo	Identificador
100	10	a
104	'x'	b
105	20	c
		d
		e

? Quizz - Endereços de variáveis

Qual o endereço e conteúdo das variáveis?
Admita que ela estão seguidas na memória no sentido crescente. (pode não ser sempre assim na prática)

```
int a = 10;
char b = 'x';
int c = 20;
double d = 50.0;
char e = 91;
```

Endereço	Conteúdo	Identificador
100	10	a
104	'x'	b
105	20	c
109	50.0	d
		e

? Quizz - Endereços de variáveis

Qual o endereço e conteúdo das variáveis?
Admita que ela estão seguidas na memória no sentido crescente. (pode não ser sempre assim na prática)

```
int a = 10;
char b = 'x';
int c = 20;
double d = 50.0;
char e = 91;
```


Endereço	Conteúdo	Identificador
100	10	a
104	'x'	b
105	20	c
109	50.0	d
117	91	e


Passagem de Argumentos por Valor


```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

 Qual o valor final das variáveis?

 `x = 14, y = 28`

 Os valores **não são alterados**, pois a função recebe apenas **cópias** das variáveis.

 O facto da função fazer return de um valor (neste caso de `y`) não significa que o `y` do main seja alterado.

O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

➡ int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	<lixo>		<vazio>
104	<lixo>		<vazio>
108	<lixo>		<vazio>
...	<lixo>		<vazio>
500	<lixo>		<vazio>
504	<lixo>		<vazio>

O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	<lixo>		<vazio>
104	<lixo>		<vazio>
108	<lixo>		<vazio>
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	<lixo>		<vazio>
104	<lixo>		<vazio>
108	<lixo>		<vazio>
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?

```
➔  int troca(int x, int y)
    {
        int aux = x;
        x = y;
        y = aux;
        return y;
    }

    int main(void) {
        int x = 14, y = 28;
        troca(x, y);
        printf("x = %d, y = %d\n", x, y);
        return 0;
    }
||
```

End.	Conteúdo	Id	
100	14	x	stack frame troca()
104	28	y	stack frame troca()
108	<lixo>		<vazio>
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	14	x	stack frame troca()
104	28	y	stack frame troca()
104	14	aux	stack frame troca()
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()



O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	28	x	stack frame troca()
104	28	y	stack frame troca()
104	14	aux	stack frame troca()
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?


 

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```


End.	Conteúdo	Id	
100	28	x	stack frame troca()
104	14	y	stack frame troca()
104	14	aux	stack frame troca()
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?



```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}






int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	28	x	stack frame troca()
104	14	y	stack frame troca()
104	14	aux	stack frame troca()
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

O que acontece na memória?

```
int troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
    return y;
}

int main(void) {
    int x = 14, y = 28;
    troca(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	<lixo>		<vazio>
104	<lixo>		<vazio>
108	<lixo>		<vazio>
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()


A função `main()` tem acesso às variáveis que estão no seu *stack frame*. E portanto `x = 14` e `y = 28`.

Apontadores

O que são apontadores?

São variáveis que guardam **endereços de memória**.

```
int x = 14;
int *x_ptr = &x;
```

 `x_ptr` guarda o endereço de `x`, não o seu valor. Por isso diz-se que `x_ptr` aponta para `x`.

Endereço	Conteúdo	Identificador
1024	14	x
...		
2036	1024	x_ptr

 Qualquer que seja o tipo do apontador, ocupa sempre 8 bytes

Apontadores para que servem?

 A partir do apontador, podemos aceder ao valor da variável original

 Para isso usa-se o operador **desreferência**: `*`

```
int x = 14;
int *x_ptr = &x;
```

Endereço	Conteúdo	Identificador
1024	14	x
...		
2036	1024	x_ptr

```
// ler o conteúdo que está no endereço dado por x_ptr
printf("Valor de x = %d\n", *x_ptr);

// alterar o conteúdo que está no endereço dado por x_ptr
*x_ptr = 84;
```

Operador `&` (referência)

 Obtemos o endereço de uma variável com `&`:

```
int x = 14;
int *x_ptr = &x;

printf("Endereço de x: %p\n", (void*)x_ptr);
```

Endereço	Conteúdo	Identificador
1024	14	x
2028	1024	x_ptr

 `&x` retorna o endereço da variável `x` !

Operador `*` (des-referência)

Permite aceder ao valor armazenado no endereço apontado:

```
int x = 14;
int *ptr = &x;

int k = *ptr; // k recebe o valor de x
```

Endereço	Conteúdo	Identificador
1024	14	x
2028	1024	ptr
2036	14	k

 `*ptr` acede ao valor guardado no endereço armazenado em `ptr`.

⚡ Passagem de Argumentos por Referência

```
void troca(int *x, int *y) {
    int aux = *x;
    *x = *y;
    *y = aux;
}

int main(void) {
    int x = 14, y = 28;
    troca(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

End.	Conteúdo	Id	
100	500	x	stack frame troca()
108	504	y	stack frame troca()
104	14	aux	stack frame troca()
...	<lixo>		<vazio>
500	14	x	stack frame main()
504	28	y	stack frame main()

📢 Agora **x** e **y** são alterados corretamente! 🚀

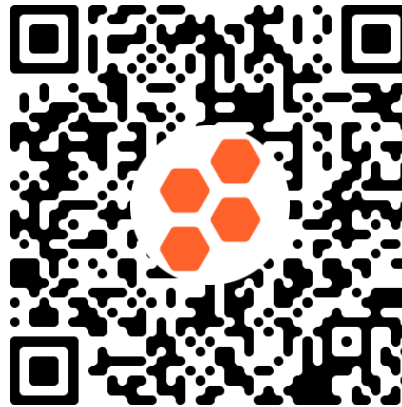
Passagem por Valor vs Referência

Passagem por Valor	Passagem por Referência
Copia os valores	Passa os endereços
Não altera as variáveis originais	Pode alterar as variáveis originais

```
troca(x, y);    // Passagem por valor  
troca(&x, &y);  // Passagem por referência
```

 Há situações em que é mais eficiente fazer passagem por referência, mas nem sempre.

? Quizz - Apontadores



No campo nome devem colocar o número de aluno 2XXXXXXX.

Exercício: Qual é a saída?

```
int a = 10;
int b = 20;
int *p = &a;
*p = 20;
p = &b;
*p = *p + 1;
printf("a = %d, b = %d\n", a, b);
```

End.	Conteúdo	Id
100	10	a
104	20	b
108	100	p

Exercício: Qual é a saída?

```

int a = 10;
int b = 20;
int *p = &a;
➔ *p = 20;
p = &b;
*p = *p + 1;
printf("a = %d, b = %d\n", a, b);

```

End.	Conteúdo	Id
100	20	a
104	20	b
108	100	p

Exercício: Qual é a saída?

```


int a = 10;
int b = 20;
int *p = &a;
→ *p = 20;
  p = &b;
  *p = *p + 1;
  printf("a = %d, b = %d\n", a, b);
    
```

End.	Conteúdo	Id
100	20	a
104	20	b
108	104	p

Exercício: Qual é a saída?

```
int a = 10;
int b = 20;
int *p = &a;
*p = 20;
p = &b;
➔ *p = *p + 1;
printf("a = %d, b = %d\n", a, b);
```

End.	Conteúdo	Id
100	20	a
104	21	b
108	104	p

 Resposta: a = 20, b = 21

Resumo

- ✓ Apontadores armazenam **endereços de memória**.
- ✓ `&` retorna o **endereço** de uma variável.
- ✓ `*` retorna o **conteúdo** de um endereço.
- ✓ Passagem por **valor** copia os dados, passagem por **referência** altera os originais.

Fim da Aula

Perguntas? ?

 **Desafio:** Experimenta modificar os exemplos no teu compilador!