

# Dominando Estrutura de Dados 1

## Funções em C

Prof. Samuel Martins (Samuka)  
@xavecoding @hisamuka



# Funções

- **Funções** são estruturas que agrupam um **bloco de comandos**, que retornam **um único valor** ao final de sua execução;

```
tipo funcao(tipo param1, tipo param2, ..., tipo paramN) {  
    comandos...  
  
    return valor_de_retorno;  
}
```

- Toda **função** deve ter um **tipo**. Esse **tipo** determina qual será o **tipo** de seu **valor de retorno**;
- Uma função pode não ter **parâmetros**, basta não informá-los;
- Funções **não** podem ser declaradas **dentro** de outras funções;

# Passagem de Parâmetros

# Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.

```
int soma(int x, int y) {  
    int z = x + y;  
  
    return z;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
c = soma(a, b);
```

## Memória Ram

X008	
X004	
X000	
...	
A108	
A104	
A100	

# Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.

```
int soma(int x, int y) {  
    int z = x + y;  
  
    return z;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
c = soma(a, b);
```


```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = #####;
```

## Memória Ram

X008	####	
X004	####	
X000	####	
...	...	
A108	####	c
A104	20	b
A100	10	a


# Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.



```
int soma(int x, int y) {  
    int z = x + y;  
  
    return z;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
c = soma(a, b);
```



```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = ####;  
  
&x = X000; x = a = 10;  
&y = X004; y = b = 20;
```

## Memória Ram

X008	####	
X004	20	y
X000	10	x
...	...	
A108	####	c
A104	20	b
A100	10	a



# Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.

```
int soma(int x, int y) {  
    int z = x + y;  
    return z;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
c = soma(a, b);
```

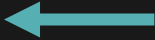
```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = ####;  
  
&x = X000; x = a = 10;  
&y = X004; y = b = 20;  
&z = X008; z = 10 + 20 = 30;
```


## Memória Ram

X008	30	z
X004	20	y
X000	10	x
...	...	
A108	####	c
A104	20	b
A100	10	a

# Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.


```
int soma(int x, int y) {  
    int z = x + y;  
  
    return z;   
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
c = soma(a, b); 
```

Ao chamar uma **função**, o compilador cria uma **cópia** de todos os parâmetros passados, **destruindo-os** ao final da execução da função.

```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = 30;
```

## Memória Ram

X008	####	
X004	####	
X000	####	
...	...	
A108	30	c 
A104	20	b
A100	10	a



# Passagem por Referência

É passado a **referência** de uma variável (**ponteiro**) para a função, possibilitando alterar uma **variável** que é externa a uma função.

```
void soma(int x, int y, int *z) {  
    *z = x + y;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
soma(a, b, &c);
```

## Memória Ram

X008	
X004	
X000	
...	
A108	
A104	
A100	

# Passagem por Referência

É passado a **referência** de uma variável (**ponteiro**) para a função, possibilitando alterar uma **variável** que é externa a uma função.

```
void soma(int x, int y, int *z) {  
    *z = x + y;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
soma(a, b, &c);
```


```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = #####;
```

## Memória Ram

X008	####	
X004	####	
X000	####	
...	...	
A108	####	c
A104	20	b
A100	10	a


# Passagem por Referência

É passado a **referência** de uma variável (**ponteiro**) para a função, possibilitando alterar uma **variável** que é externa a uma função.



```
void soma(int x, int y, int *z) {  
    *z = x + y;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
soma(a, b, &c);
```




```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = ####;
```

```
&x = X000; x = a = 10;  
&y = X004; y = b = 20;  
&z = X008; z = A108; *z = *(A108) = ####;
```

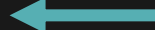
## Memória Ram


X008	A108	z
X004	20	y
X000	10	x
...	...	
A108	####	c
A104	20	b
A100	10	a



# Passagem por Referência

É passado a **referência** de uma variável (**ponteiro**) para a função, possibilitando alterar uma **variável** que é externa a uma função.

```
void soma(int x, int y, int *z) {  
    *z = x + y;   
}
```


```
int a = 10;  
int b = 20;  
int c;  
soma(a, b, &c); 
```

```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = 30;
```

```
&x = X000; x = a = 10;  
&y = X004; y = b = 20;  
&z = X008; z = A108; *z = *(A108) = 30;
```

## Memória Ram

X008	A108	z
X004	20	y
X000	10	x
...	...	
A108	30	c
A104	20	b
A100	10	a



# Passagem por Referência

É passado a **referência** de uma variável (**ponteiro**) para a função, possibilitando alterar uma **variável** que é externa a uma função.

```
void soma(int x, int y, int *z) {  
    *z = x + y;  
}
```

```
int a = 10;  
int b = 20;  
int c;  
  
soma(a, b, &c);
```

```
&a = A100; a = 10;  
&b = A104; b = 20;  
&c = A108; c = 30;
```

Ao chamar uma **função**, o compilador cria uma **cópia** de todos os parâmetros passados, **destruindo-os** ao final da execução da função.

## Memória Ram

X008	####	
X004	####	
X000	####	
...	...	
A108	30	c
A104	20	b
A100	10	a

# Funções: + um exemplo

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

```
int a = 10;  
int b = 20;  
  
troca(&a, &b);
```

## Memória Ram

X016	
X008	
X000	
...	...
A108	
A104	
A100	

# Funções: Ponteiros Read Only (Somente Leitura)

Ao passarmos a **referência (ponteiro)** para uma **função**, caso a mesma seja **somente para leitura (*read only*)**, podemos evitar uma alteração acidental de seu valor dentro da função apenas adicionando a palavra reservada: *const*;

```
void imprime_vector(const int *vector) {  
    ...  
}
```

# Exercício

Mostre todos os passos (teste de mesa) do programa abaixo e identifique o que é impresso pelo programa.

```
void func(int *px, int *py) {
    px = py;
    *py = (*py) * (*px);
    *px = *px + 2;
}

int main() {
    int x, y;
    scanf("%d",&x);
    scanf("%d", &y);

    func(&x,&y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

Teste de Mesa

x	y	px	*px	py	*py
---	---	----	-----	----	-----


Sooo



# Dominando Estrutura de Dados 1

## Funções em C

Prof. Samuel Martins (Samuka)  
@xavecoding @hisamuka

