

Linguagem de Programação 2

5a: Subrotinas

Prof. Flávio José Mendes Coelho

Subrotinas

Subrotinas

- É um bloco de comandos cuja lógica possui um objetivo bem definido, e que possui um nome.
- Exemplos: **pow()** e **sqrt()** da biblioteca **cmath**

```
main() {  
    float a, b, c, delta, x1, x2;  
    ...  
    delta = pow(b, 2.0) - 4*a*c;  
  
    x1 = ( -b + sqrt(delta) )/2.0*a;  
    x2 = ( -b - sqrt(delta) )/2.0*a;  
    ...  
}
```

Subrotinas

- As subrotinas `pow()` e `sqrt()`
 - Foram escritas (programadas) por algum programador responsável por implementar as subrotinas matemáticas da biblioteca `cmath`
 - Podem ser utilizadas (chamadas) quantas vezes quisermos

Subrotinas

- Há dois tipos de subrotinas
 - **Funções**: retornam algum valor (ex.: pow e sqrt)
 - **Procedimentos**: não retornam valor
- Em C todas as subrotinas são chamadas de **funções**

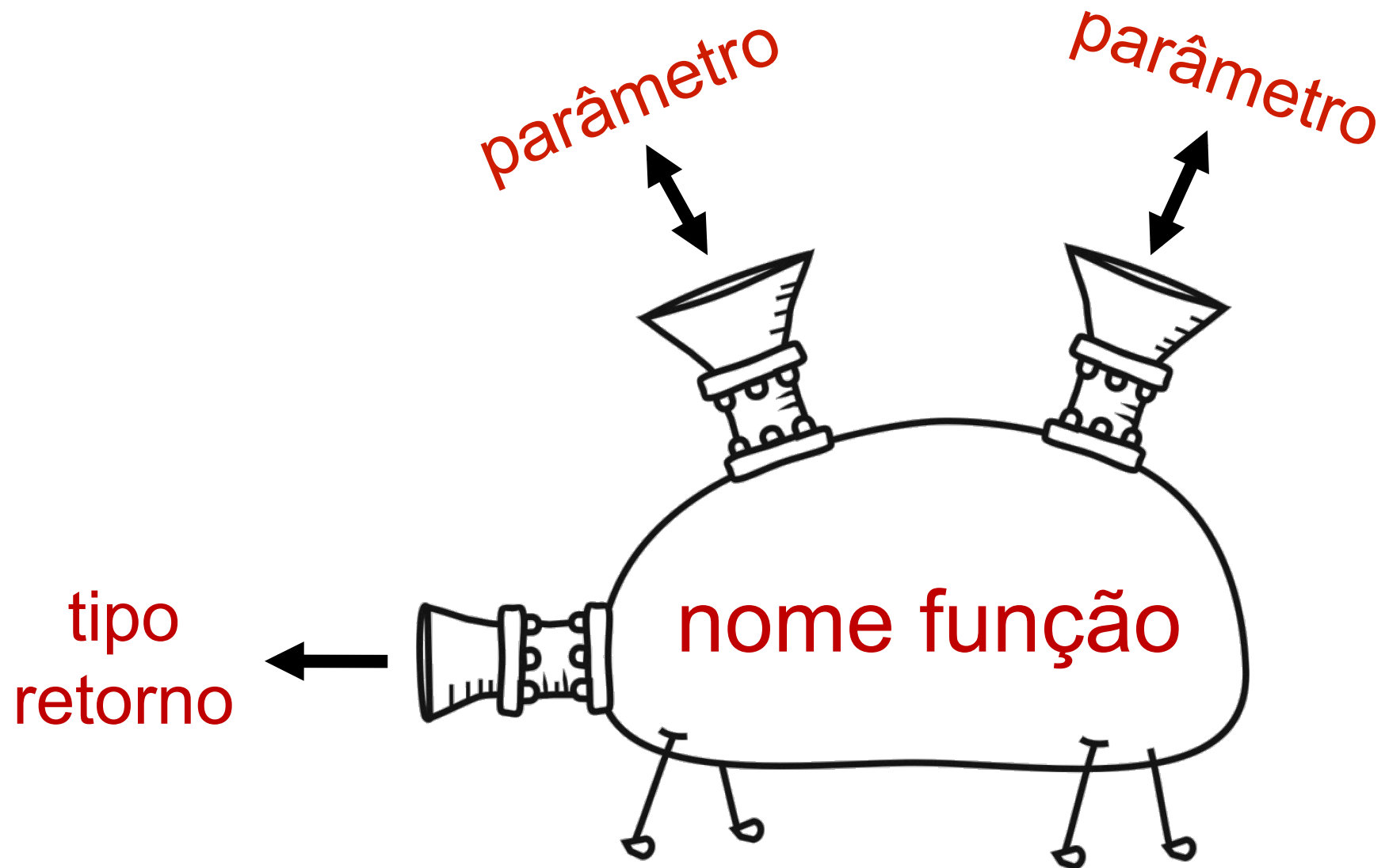
Funções: características

- É um bloco de comandos delimitados por { e }, cuja lógica possui um objetivo bem definido

```
<tipo retorno> <nome>([<parâmetros>]) {  
    comando 1;  
    comando 2;  
    ...  
    comando n;  
}
```

- Exemplo: função main()

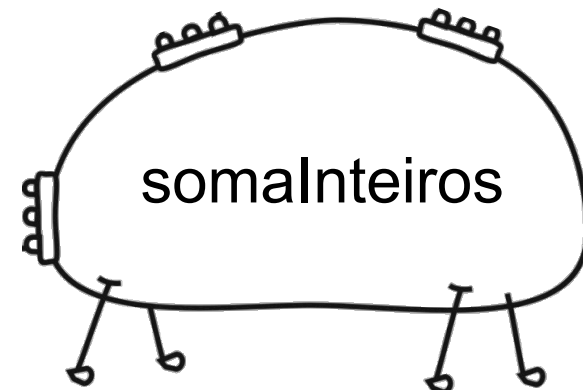
Funções: características



Funções: características

- Possui um **nome** que resume o seu objetivo

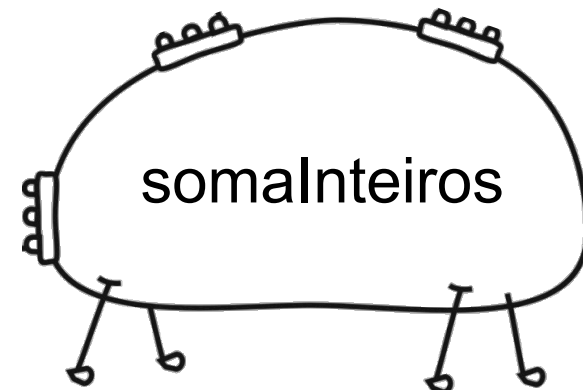
```
<tipo retorno> somaInteiros([<parâmetros>]) {  
    comando 1;  
    comando 2;  
    ...  
    comando n;  
}
```



Funções: características

- Cada **comando** é um comando válido C

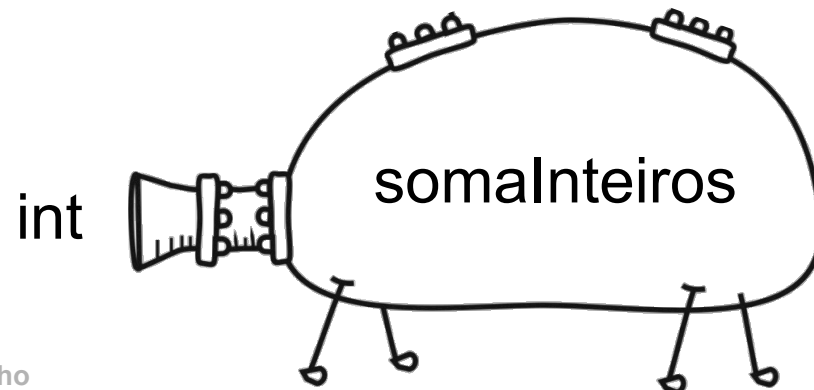
```
<tipo retorno> somaInteiros([<parâmetros>]) {  
    int soma = 0;  
    for (int i = 1; i <= 5; i++) {  
        soma = soma + i;  
    }  
}
```



Funções: características

- Pode **retornar** um valor de um tipo de dado C primitivo ou definido pelo programador

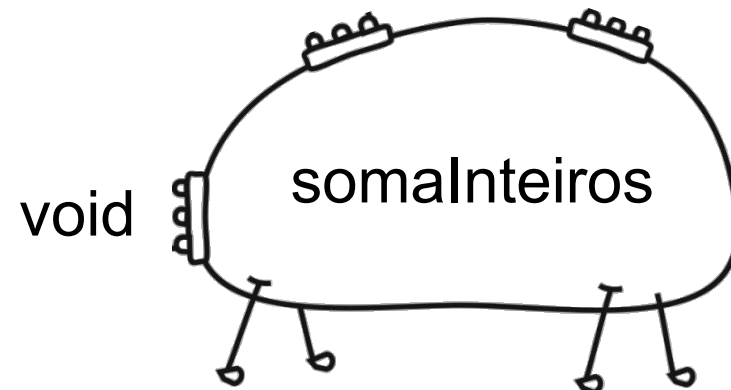
```
int somaInteiros([<parâmetros>]) {  
    int soma = 0;  
    for (int i = 1; i <= 5; i++) {  
        soma = soma + i;  
    }  
    return soma;  
}
```



Funções: características

- Pode **não retornar** valores

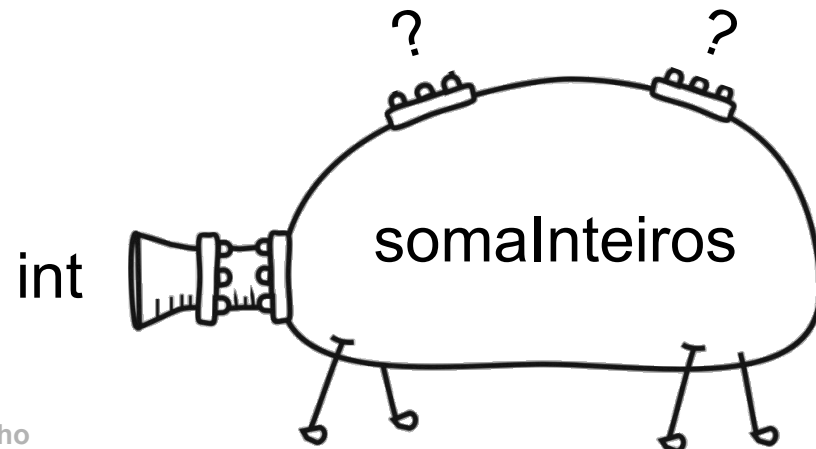
```
void somaInteiros([<parâmetros>]) {  
    int soma = 0;  
    for (int i = 1; i <= 5; i++) {  
        soma = soma + i;  
    }  
    printf("%d\n", soma);  
}
```



Funções: características

- Pode **não** ter **parâmetros**. Parâmetro: variável que recebe um valor vindo do “lado de fora” da função

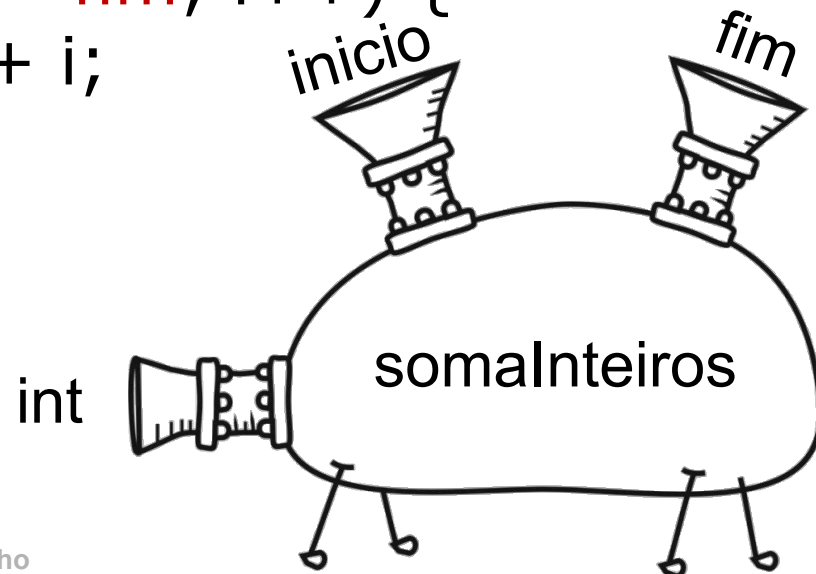
```
int somaInteiros() {  
    int soma = 0;  
    for (int i = 1; i <= 100; i++) {  
        soma = soma + i;  
    }  
    return soma;  
}
```



Funções: características

- Pode ter **parâmetros**. Parâmetro: variável que recebe um valor vindo do “lado de fora” da função

```
int somaInteiros(int inicio, int fim) {  
    int soma = 0;  
    for (int i = inicio; i <= fim; i++) {  
        soma = soma + i;  
    }  
    return soma;  
}
```



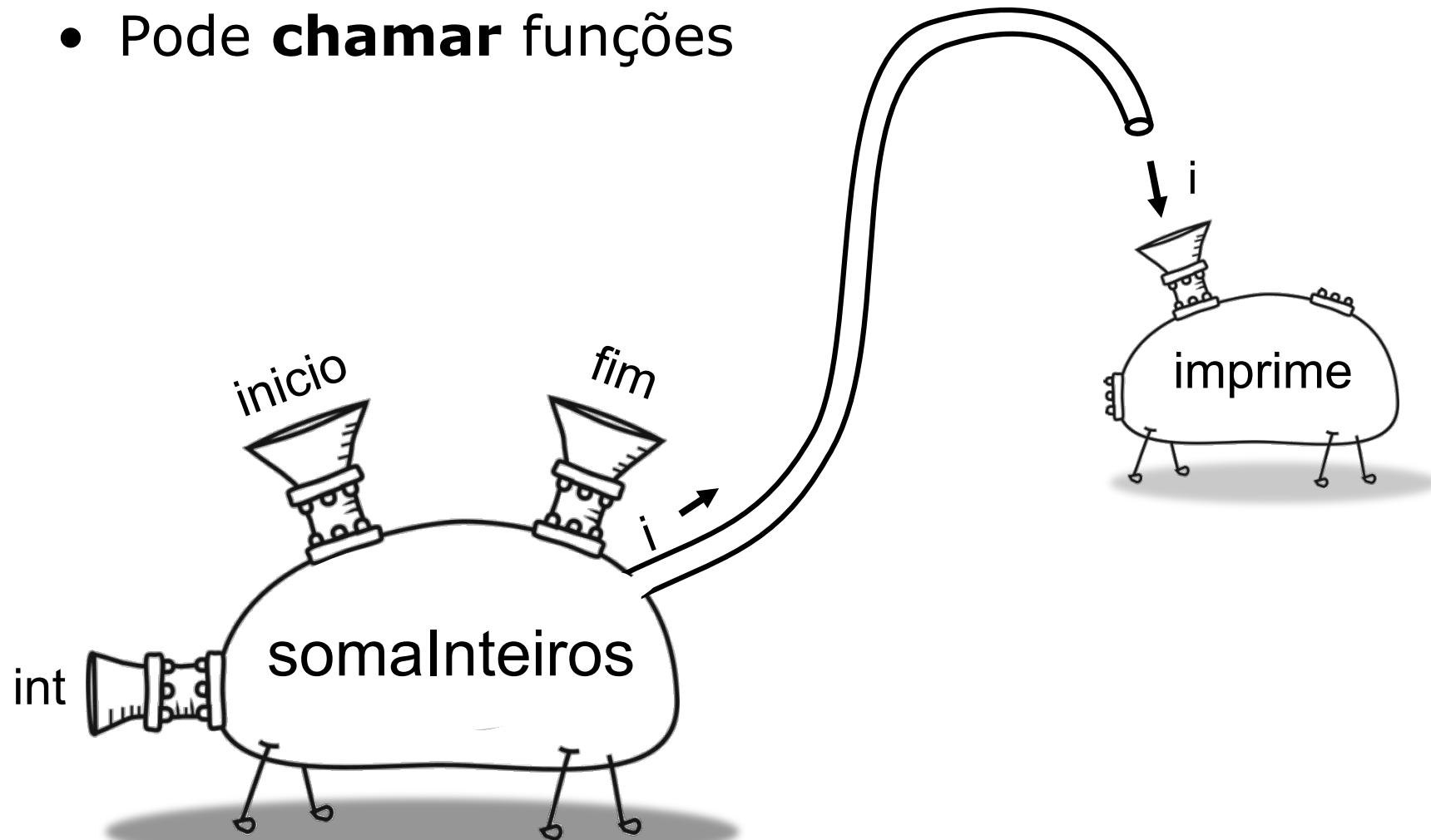
Funções: características

- Pode **chamar** funções (inclusive, a si própria)

```
int somaInteiros(int inicio, int fim) {  
    int soma = 0;  
    for (int i = inicio; i <= fim; i++) {  
        soma = soma + i;  
        imprime(i);  
    }  
    return soma;  
}
```

Funções: características

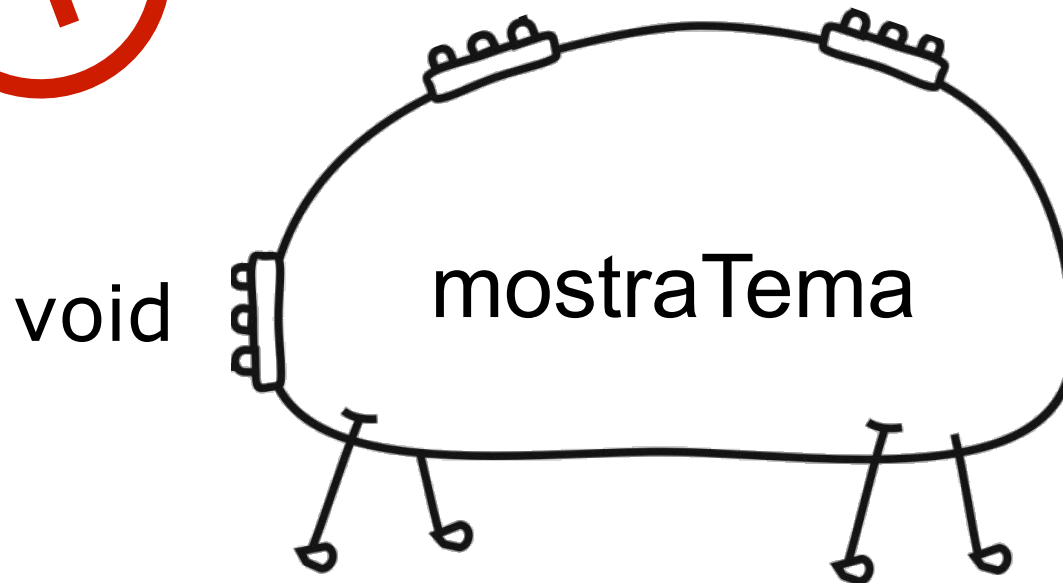
- Pode **chamar** funções



Funções: retorno X parâmetros

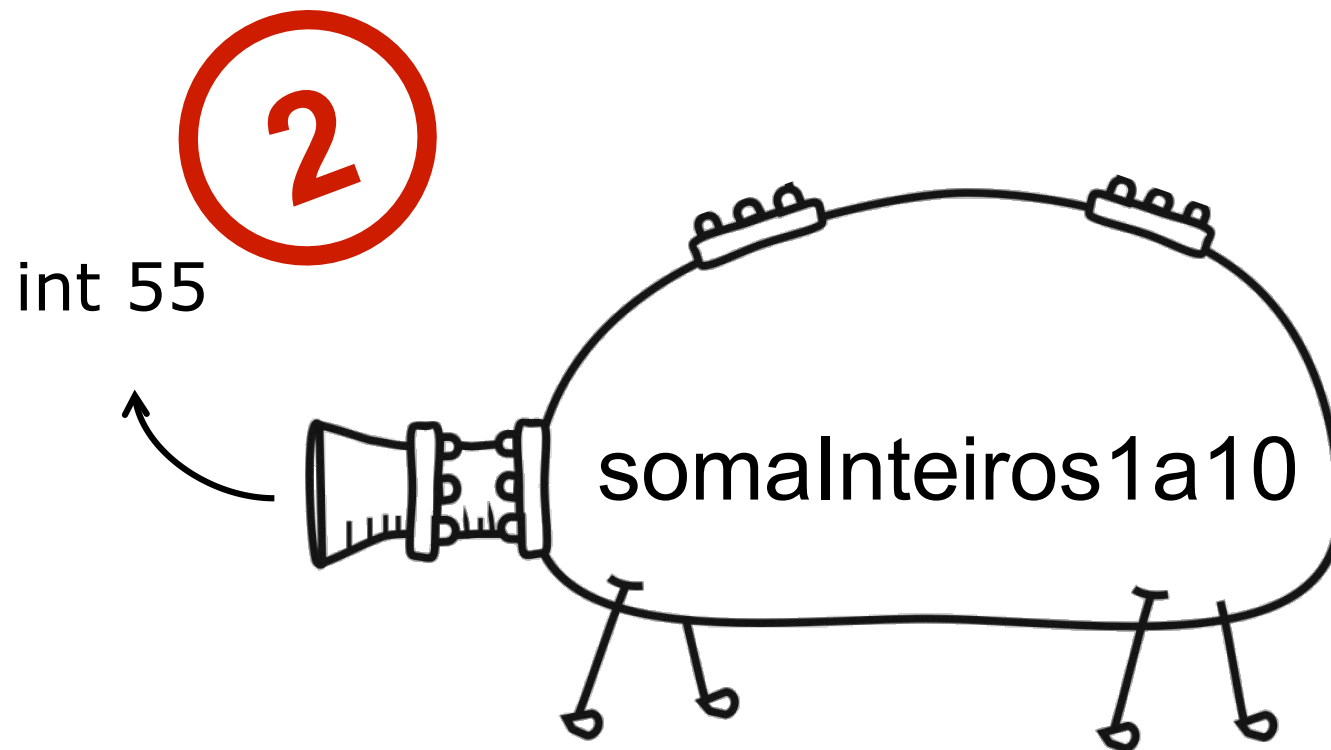
- Não retorna valor
- Não tem parâmetros

1



Funções: retorno X parâmetros

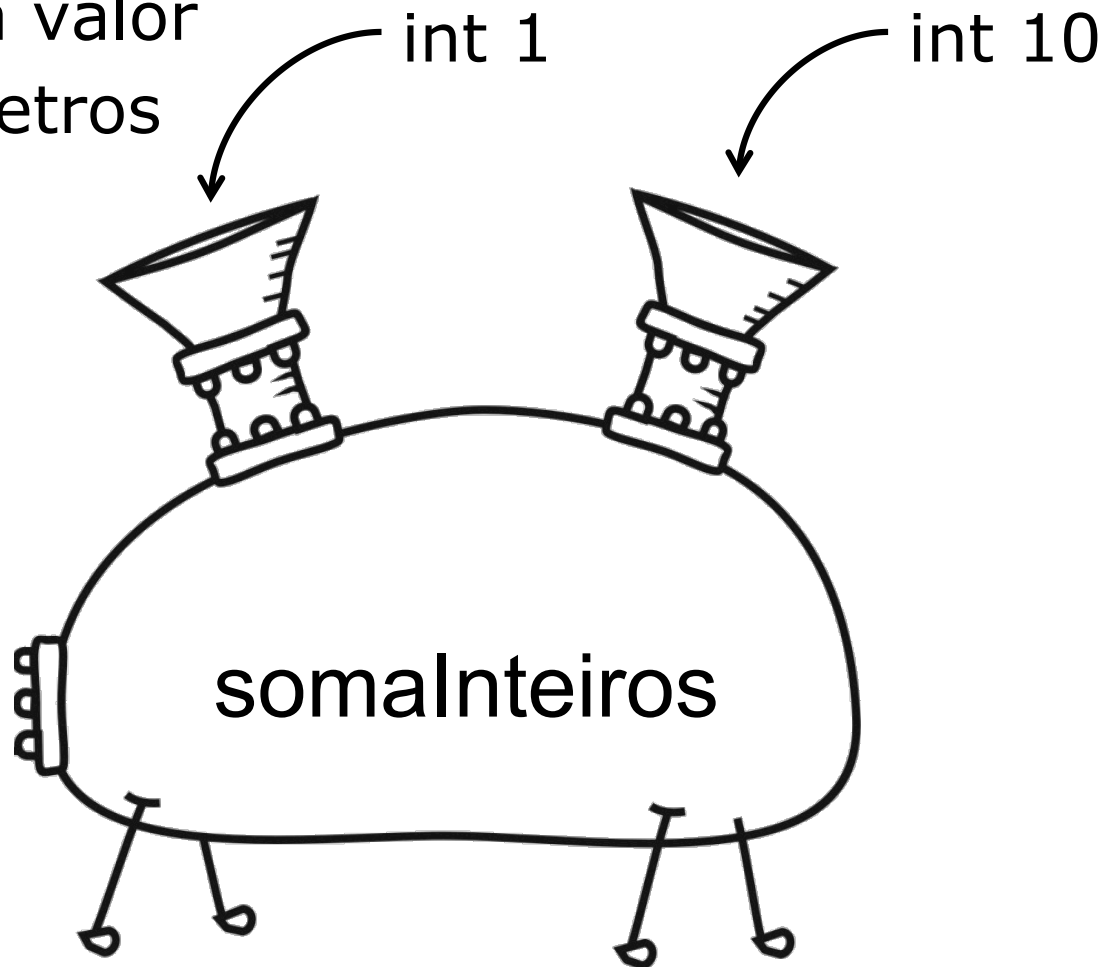
- Retorna valor
- Não tem parâmetros



Funções: retorno X parâmetros

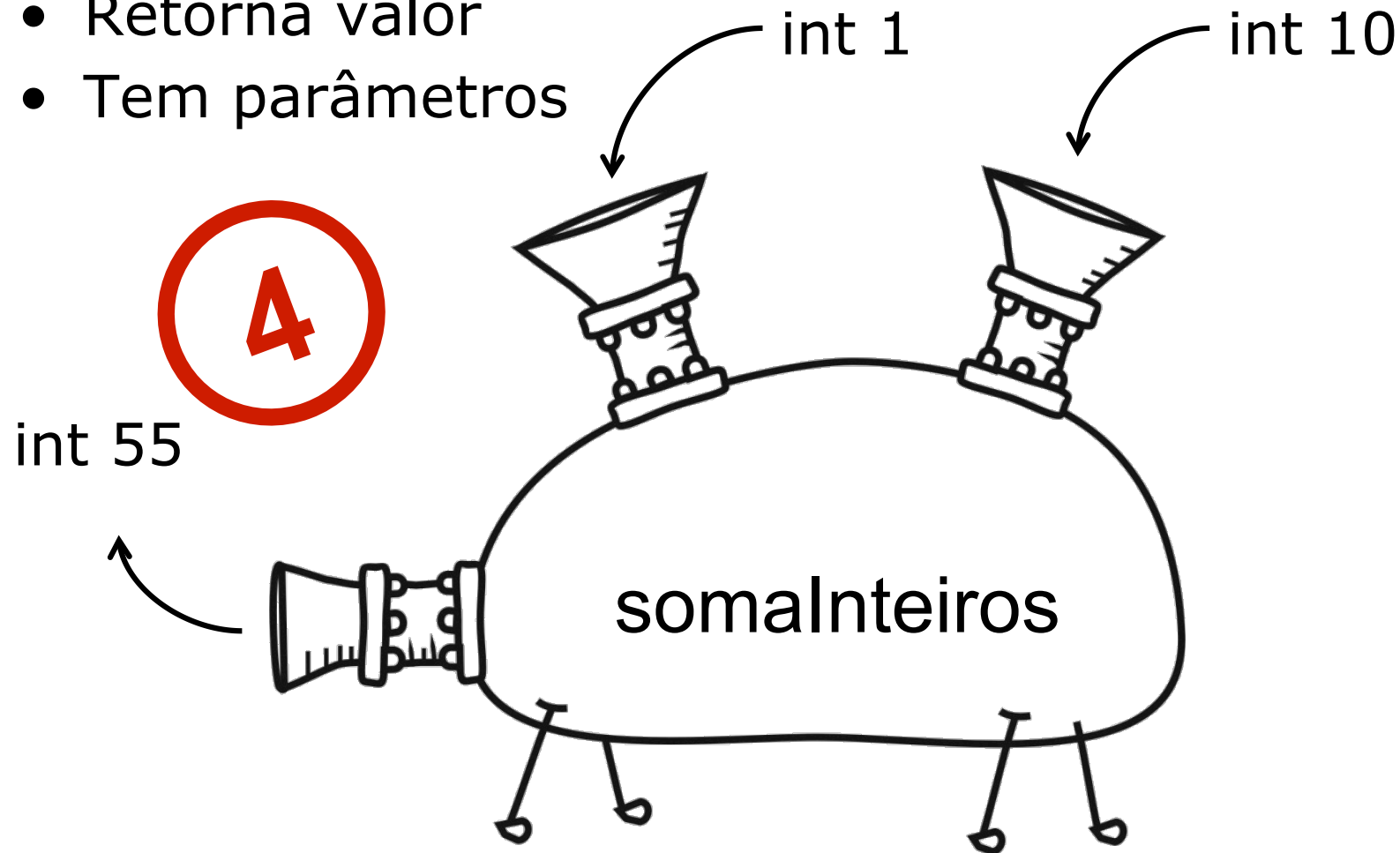
- Não retorna valor
- Tem parâmetros

3



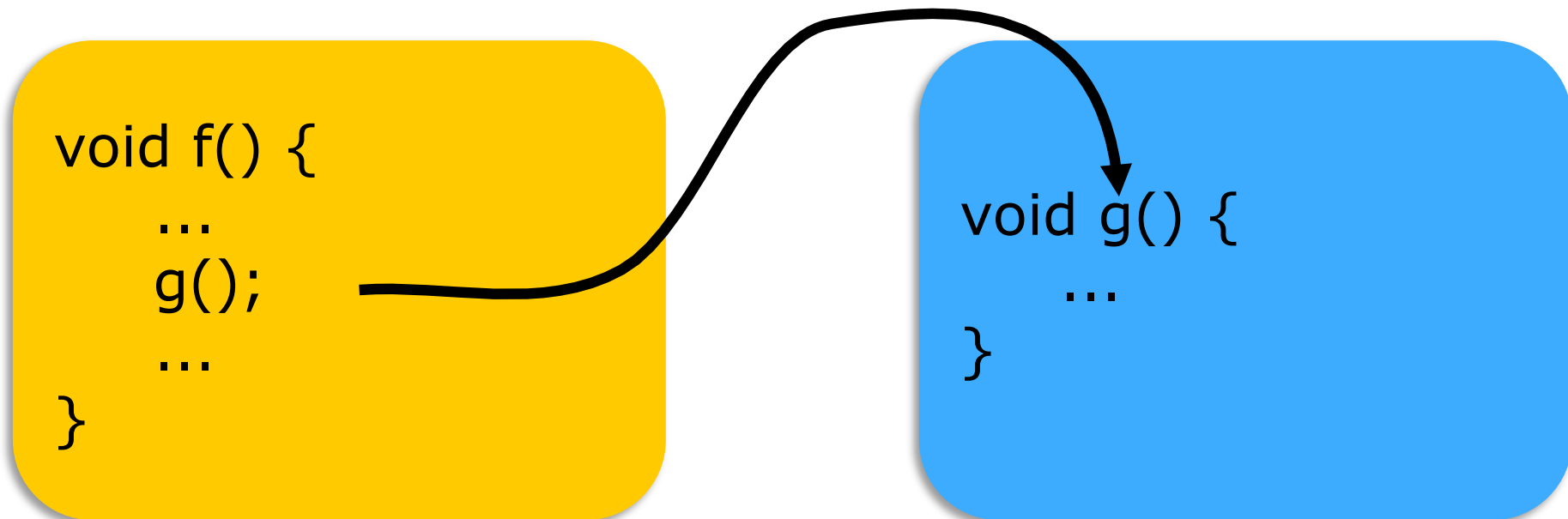
Funções: retorno X parâmetros

- Retorna valor
- Tem parâmetros



Passagem de parâmetros

- Função f() está chamando a função g()
- f() é denominada função **chamadora**
- g() é denominada função **chamada**



Passagem de parâmetros

- Em f():
 - *i* é denominado argumento (parâmetro real)
- Em g():
 - *x* é denominado parâmetro (parâmetro formal)

```
void f() {  
    int i = 10;  
    g(i);  
    ...  
}
```

```
void g(int x) {  
    x = x*x;  
}
```

Passagem de parâmetros

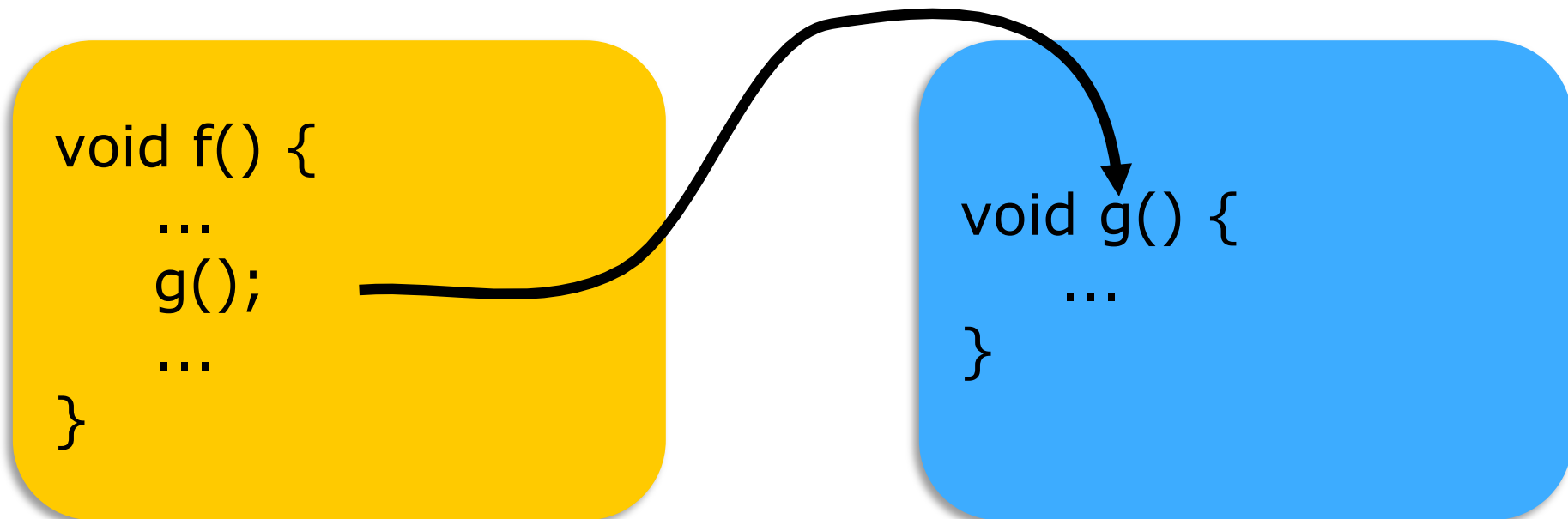
- Tanto f() quanto g() podem ter um número qualquer de parâmetros, cada um de qualquer tipo de dado

```
void f() {  
    int i = 10;  
    g(i, 'Z');  
    ...  
}
```

```
void g(int x, char s) {  
    x = x*x;  
}
```

Passagem de parâmetros

- Por valor (por cópia)
- Por referência



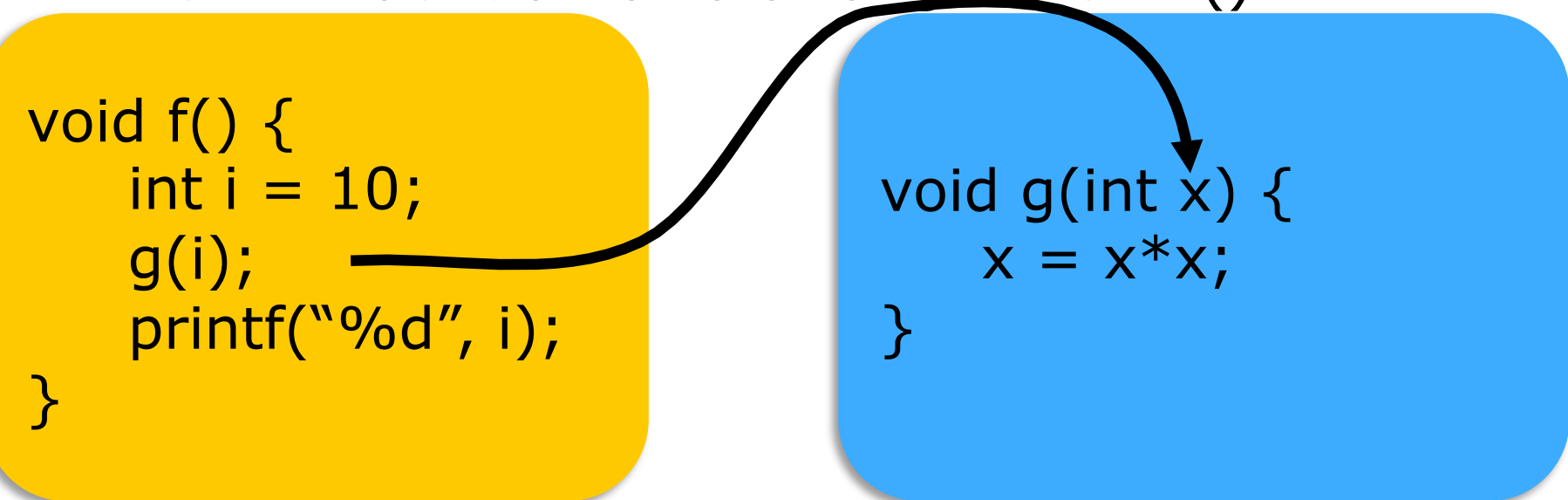
Passagem de parâmetros

Por valor

- f() passa uma cópia do **valor** de *i* para g()
- g() recebe a cópia no parâmetro
- Modificações no valor do parâmetro em g() não têm influência no valor original em f()

```
void f() {  
    int i = 10;  
    g(i);  
    printf("%d", i);  
}
```

```
void g(int x) {  
    x = x*x;  
}
```



Passagem de parâmetros

Por valor

- f() passa uma cópia do **valor** de *i* para g()
- g() recebe a cópia no parâmetro
- Modificações no valor do parâmetro em g() não têm influência no valor original em f()

```
void f() {  
    int i = 10;  
    g(i);  
    printf("%d", i);  
}
```

i=10

```
void g(int x) {  
    x = x*x;  
}
```

x=100

Passagem de parâmetros

Por referência

- f() passa o endereço de *i* para g()
- g() recebe o endereço no parâmetro
- Modificações no valor para o qual o parâmetro aponta **têm** influência no valor original em f()

```
void f() {  
    int i = 10;  
    g(&i);  
    printf("%d", i);  
}
```


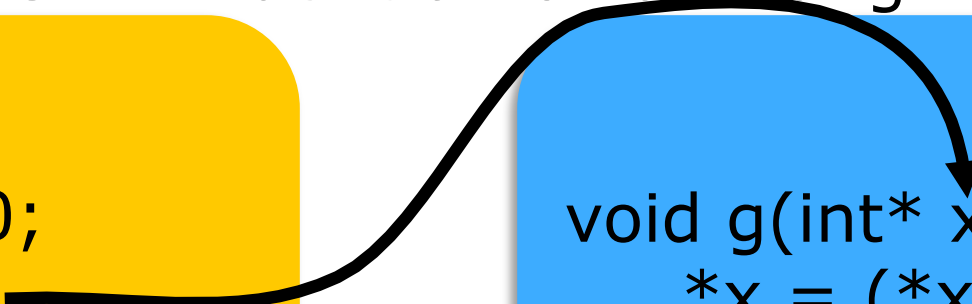
```
void g(int* x) {  
    *x = (*x)*(*x);  
}
```

Passagem de parâmetros

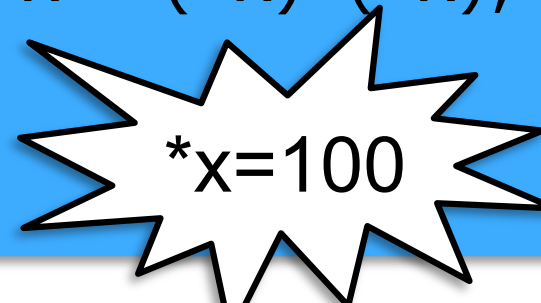
Por referência

- f() passa o endereço de *i* para g()
- g() recebe o endereço no parâmetro
- Modificações no valor para o qual o parâmetro aponta **têm** influência no valor original em f()

```
void f() {  
    int i = 10;  
    g(&i);  
    printf("%d", i);  
}
```



```
void g(int* x) {  
    *x = (*x)*(*x);  
}
```



Bibliografia

- Deitel, H. M., Deitel, P. J. C - Como Programar. 6a. ed. Pearson, 2011.