

# Cap.5: Subprogramas



---



# Sumário

---

- 📄 Modelo de programação imperativa revisitado
- 📄 Relação hierárquica entre funções
- 📄 Diagrama de sintaxe numa função
- 📄 Cabeçalho numa função
- 📄 Corpo numa função
- 📄 Características numa função
- 📄 Comunicação entre funções
- 📄 Variáveis locais e variáveis globais

# Modelo de programação imperativa: dados + funções

funções

estruturas de dados

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
int x;
```

```
float y;
```

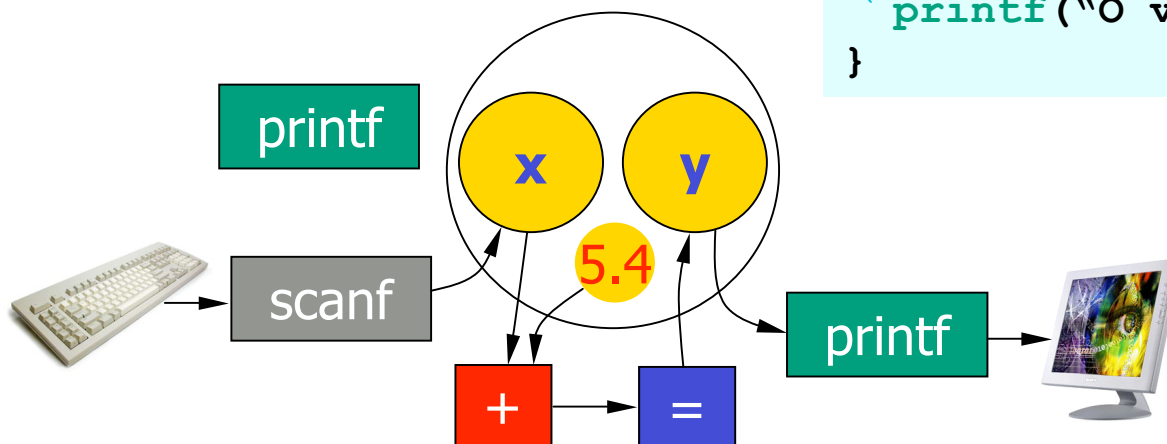
```
printf("Escreva um valor inteiro: ");
```

```
scanf("%d", &x);
```

```
y=x+5.4;
```

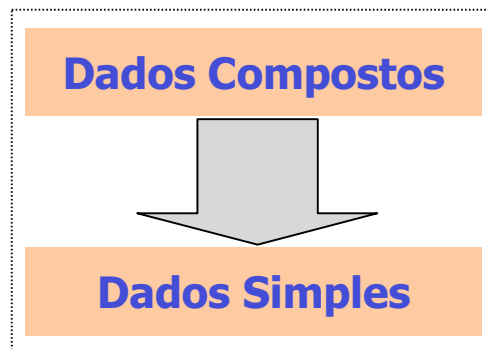
```
printf("O valor de y=%f\n", y);
```

```
}
```

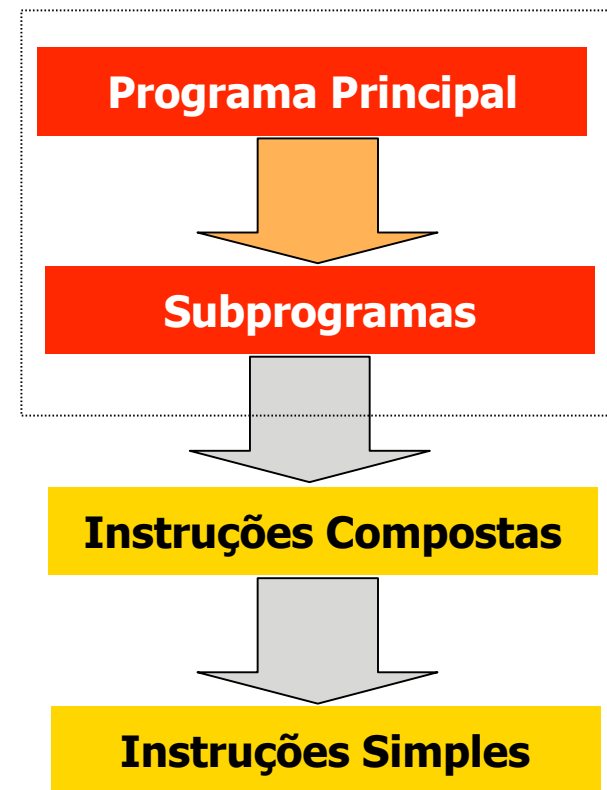


# Modelo de programação imperativa (cont.)

## ***Estruturas de Dados***



## ***Funções***

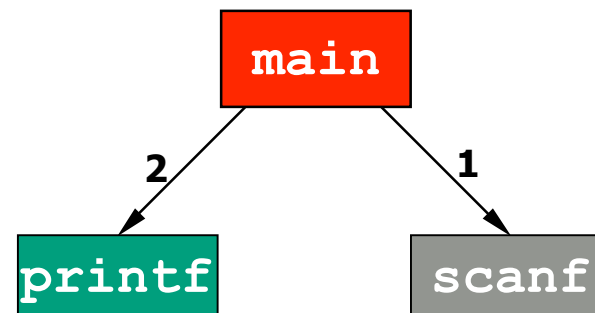


# Relação hierárquica entre funções

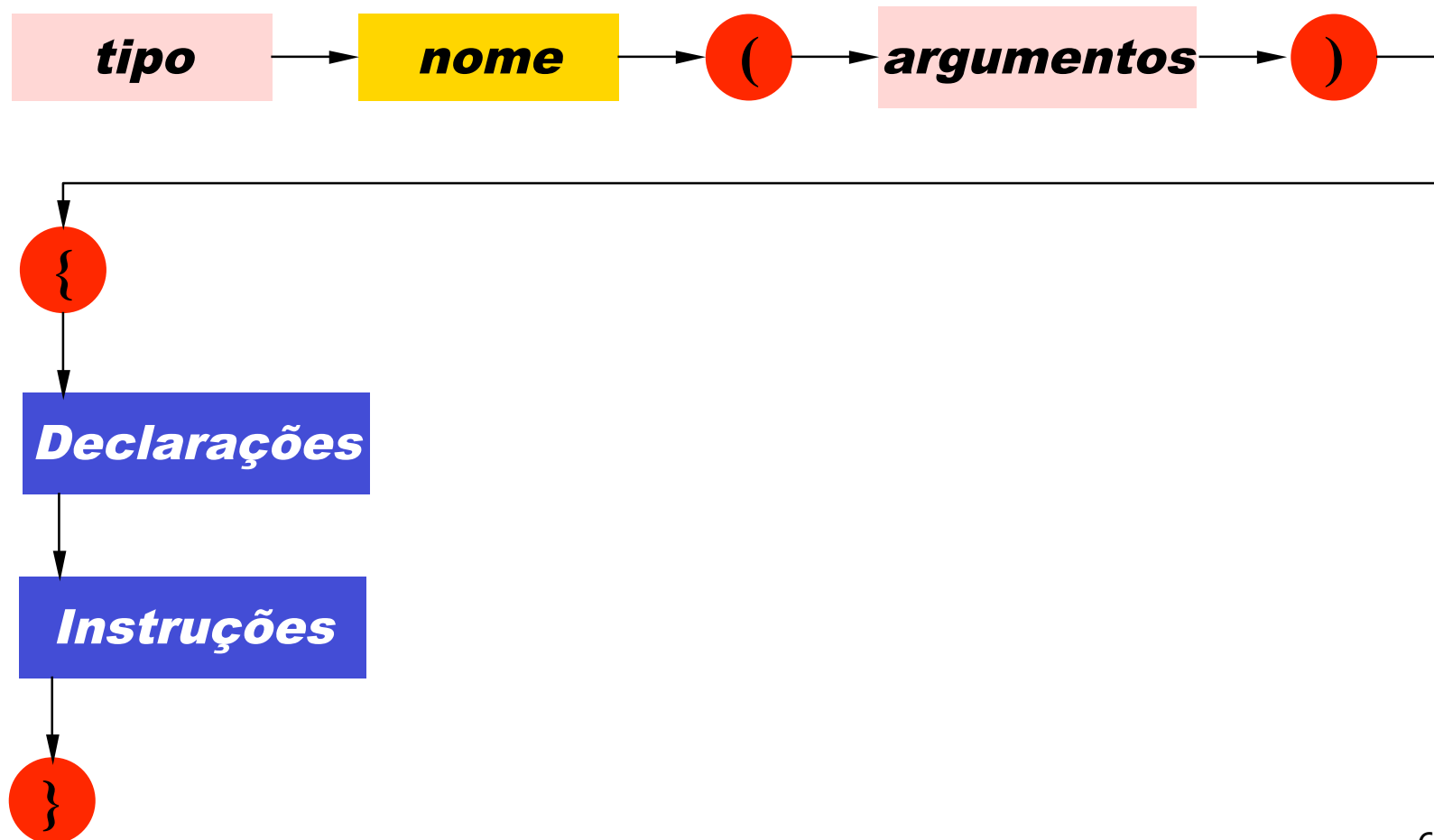
- À exceção da função principal **main**, uma função só pode ser usada/invocada/chamada dentro de outra função.
- A relação hierárquica entre funções é estabelecida na escrita dum programa.

```
#include <stdio.h>
int main()
{
    int x;
    float y;

    printf("Escreva um valor inteiro: ");
    scanf("%d", &x);
    y=x+5.4;
    printf("O valor de y=%f\n", y);
}
```

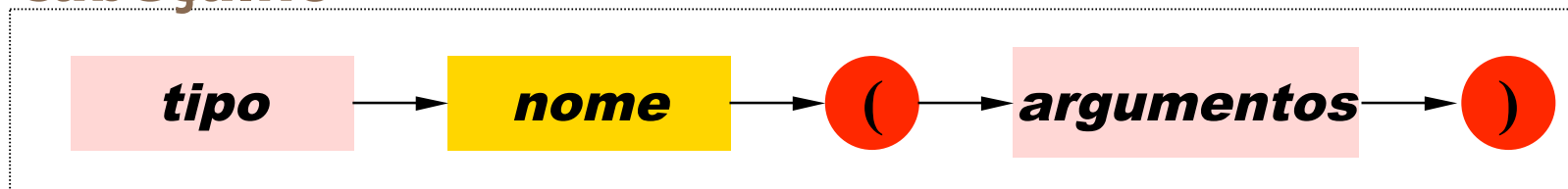


# Diagrama de sintaxe duma função

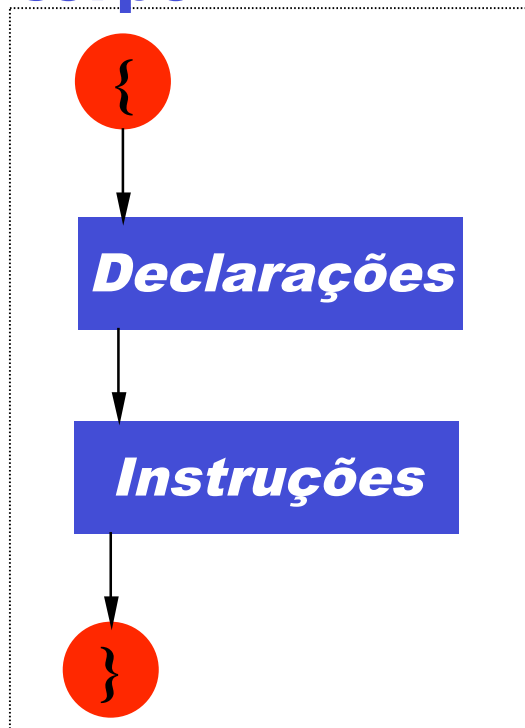


# Diagrama de sintaxe duma função (cont.)

## cabeçalho



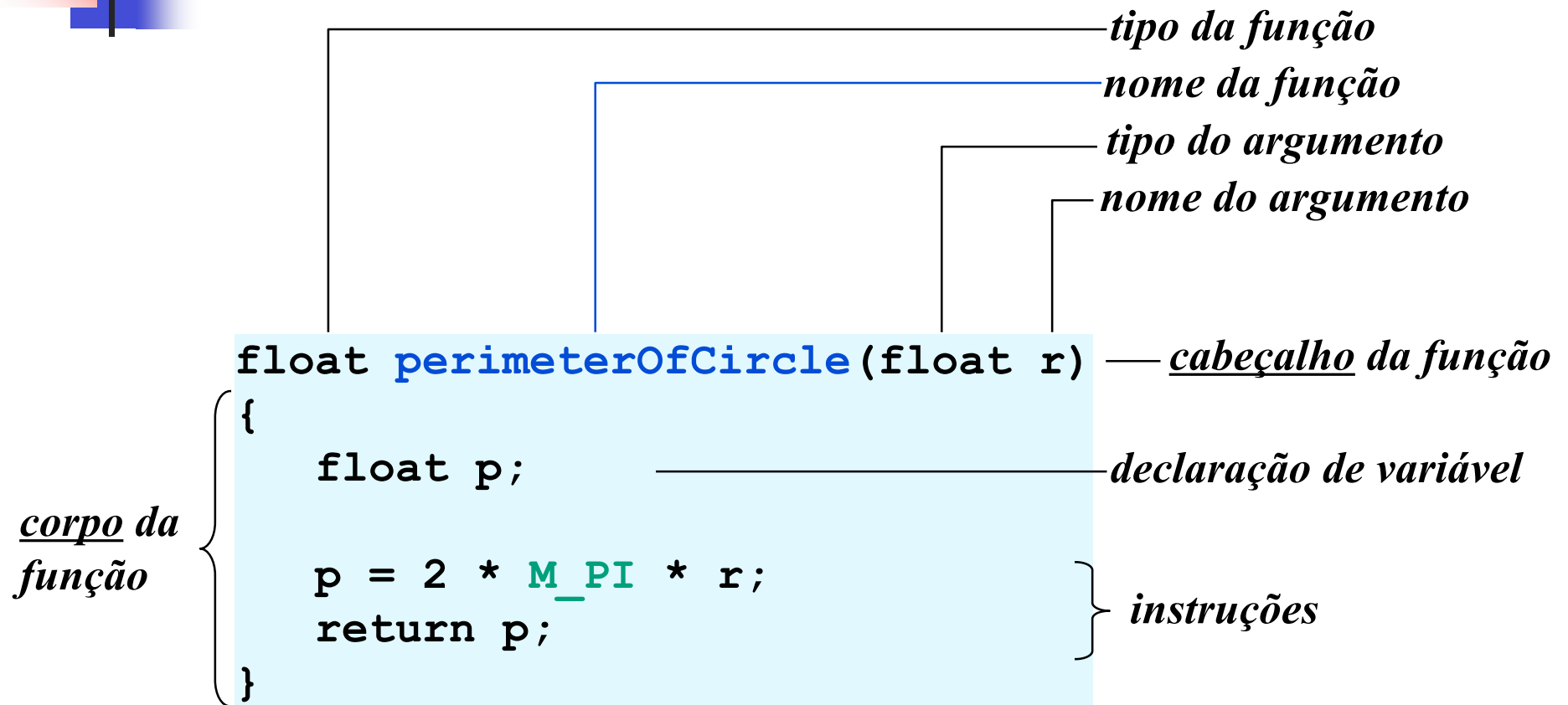
## corpo



### Notas:

- Uma função tem um **cabeçalho** e um **corpo**.
- O corpo de qualquer função é uma instrução composta de bloco { }.

# Exemplo (c/ 1 argumento)



## **Nota:**

- A constante `M_PI` está declarada no ficheiro `math.h`



# Exemplo (c/ 2 argumentos)

```
float hipotenusa(float b, float a)
{
    float h;

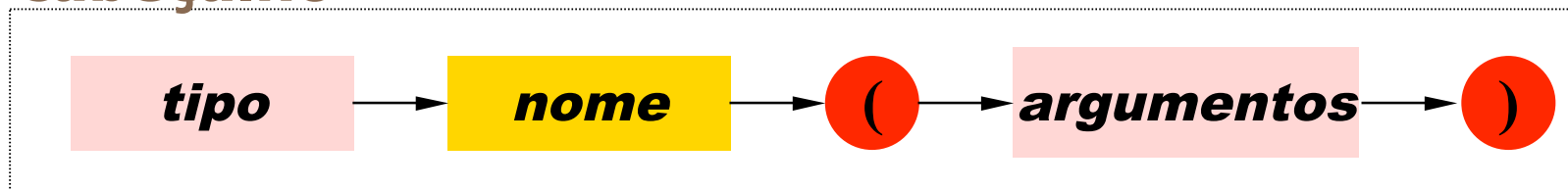
    h = sqrt(a*a + b*b);
    return h;
}
```

## **Nota:**

- A função `sqrt` está declarada no ficheiro `math.h`

# Cabeçalho dum função

## cabeçalho



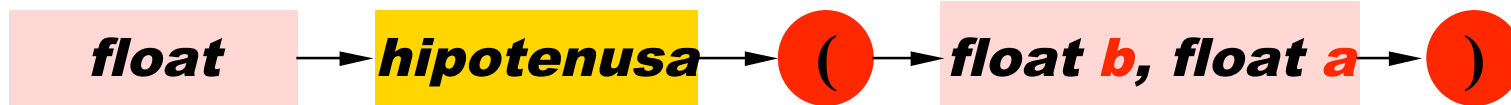
### Notas:

- O cabeçalho dum função especifica **O QUE** uma função faz.
- O cabeçalho dum função compreende 3 entidades:
  - o **nome** da função;
  - os dados de entrada, chamados **argumentos**;
  - o **tipo** de dados do resultado.
- Uma função pode ser declarada sem corpo, mas como qualquer declaração deve terminar com **;**
- A declaração dum função é designada por **protótipo**, e tem a seguinte forma:

**cabeçalho;**

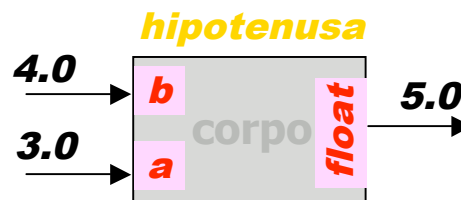
# Exemplo: cabeçalho duma função

## cabeçalho



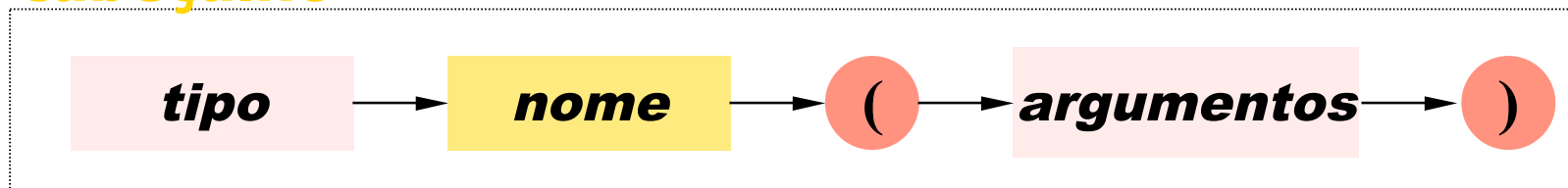
### Notas:

- A função `hipotenusa` calcula a hipotenusa dum triângulo rectângulo de base `b` e de altura `a`.
- Os argumentos `b` e `a` da função guardam os dados/valores de entrada passados para a função.
- Após a função calcular o valor da hipotenusa —o que é feito no corpo da função—, este resultado é devolvido pela função. Como especificado no cabeçalho, o resultado é um `float`.

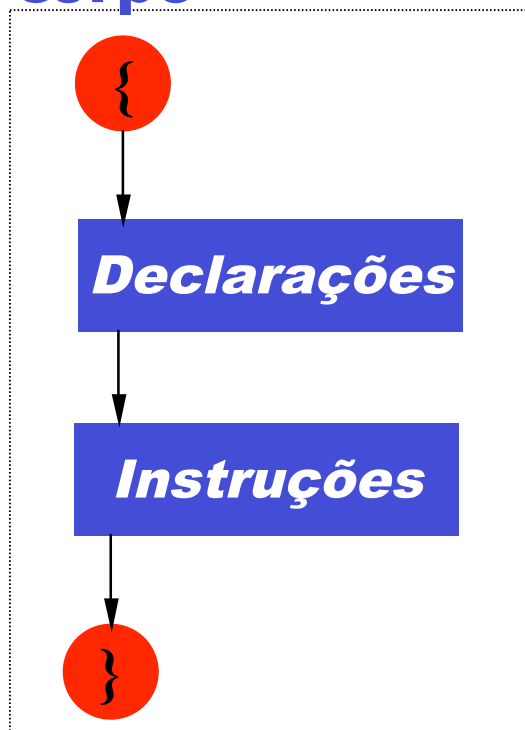


# Corpo duma função

## cabeçalho



## corpo



### Notas:

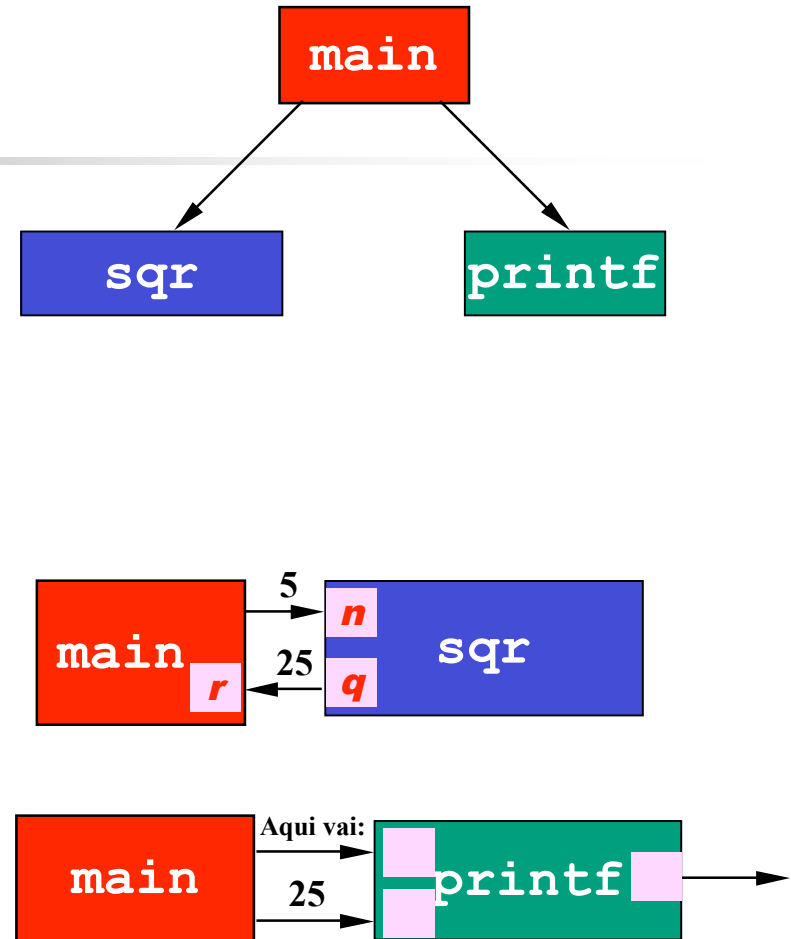
- O **corpo** duma função especifica **COMO** uma função faz.
- O **corpo** é uma instrução composta de bloco que consiste numa sequência de instruções, as primeiras das quais são declarações de variáveis locais.

# Exemplo

```
#include <stdio.h>
int sqr(int);

void main()
{
    int r;
    r = sqr(5);
    printf("Aqui vai: %i",r);
}

int sqr(int n)
{
    int q;
    q = n * n;
    return q;
}
```



# Características dum função

## **Identificação**

Tem nome/identificador único.

## **Utilização/Invocação/Chamada**

É feita a partir doutra função.

## **Unicidade**

Deve realizar uma única tarefa bem definida.

## **Funcionamento**

Funciona como uma caixa preta.

## **Generalidade** ⇒ **Reutilização**

A codificação dum função deve ser genérica e independente de qualquer programa ou projecto, de modo a que possa ser reutilizada por outros programas ou projectos.



# Comunicação entre funções

## **Utilização/Invocação/Chamada**

É feita a partir doutra função.

A função invocadora é suspensa.

A função invocada é executada.

A função invocadora retoma a sua execução.

## **Execução**

Após ser invocada, uma função é executada:

- entrada de dados ou passagem de parâmetros;
- executa bloco de instruções;
- saída de dados.

## **Passagem de parâmetros / argumentos**

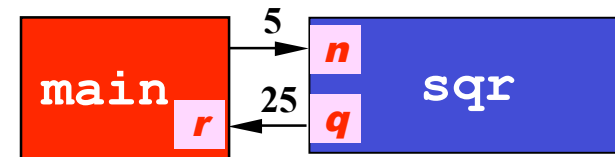
É uma operação de entrada de dados.

Atribuição 1:1 de cada parâmetro concreto (ou valor) a um parâmetro formal (que é uma variável).

## **Retorno/Devolução de valor**

É uma operação de saída de dados.

Uma função devolve sempre um valor, embora possa ser um não-valor (void).

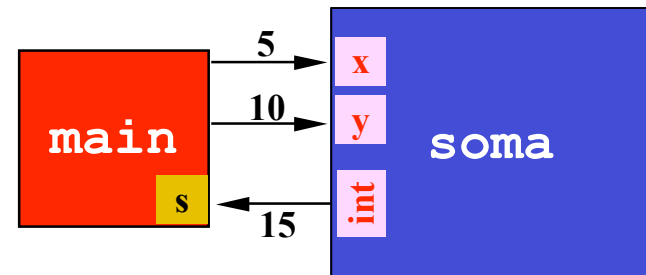


# Outro exemplo:

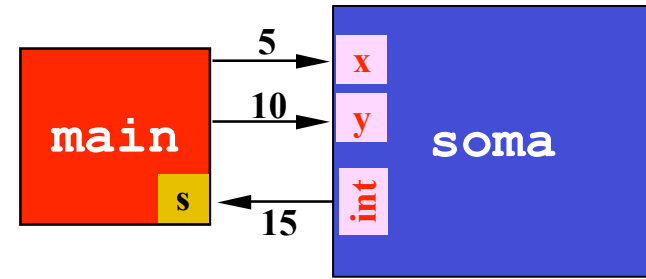
```
#include <stdio.h>
int soma(int,int);

void main()
{
    int s;
    s = soma(5,10);
    printf("Soma = %i",s);
}

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



# Variáveis locais

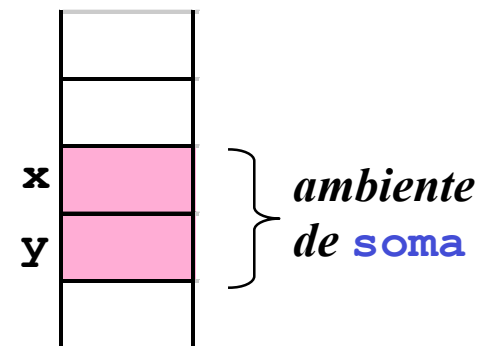
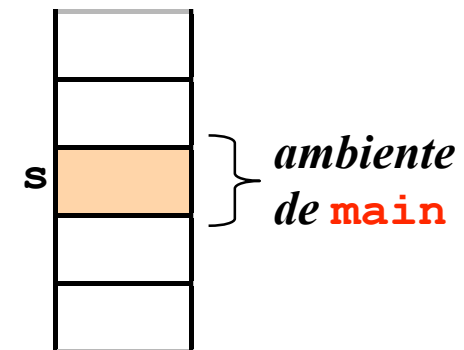


```
#include <stdio.h>
int soma(int,int);

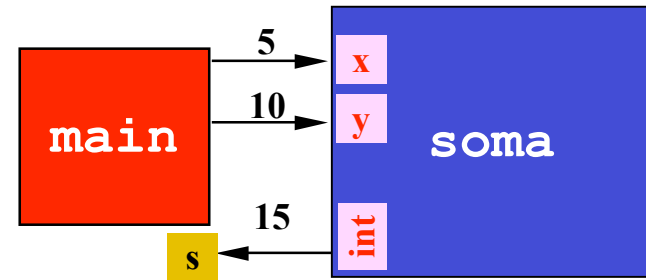
void main(void)
{
    int s;
    s = soma(5,10);
    printf("Soma = %i",s);
}

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

- Uma variável local só existe no ambiente duma função.
- Após a execução duma função, a memória ocupada pelas suas variáveis é libertada.



# Variáveis globais



```
#include <stdio.h>
int soma(int, int);
int s;

void main(void)
{
    s = soma(5, 10);
    printf("Soma = %i", s);
}

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

- Uma variável global é declarada fora do ambiente de qualquer função.
- Uma variável global permanece em memória até o programa termine a sua execução, altura em que o seu espaço ocupado em memória é libertado.
- Evitar o uso de variáveis globais.

