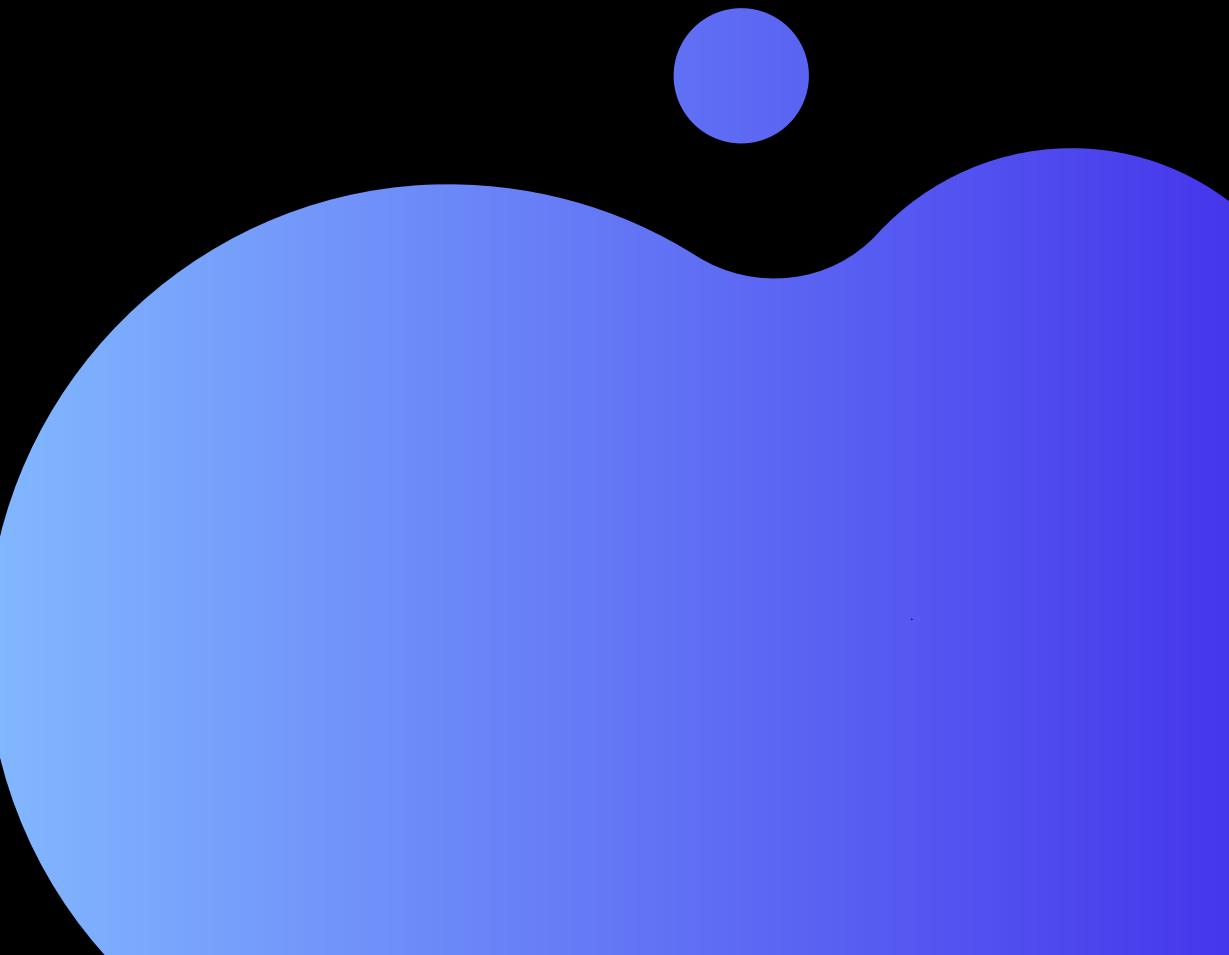
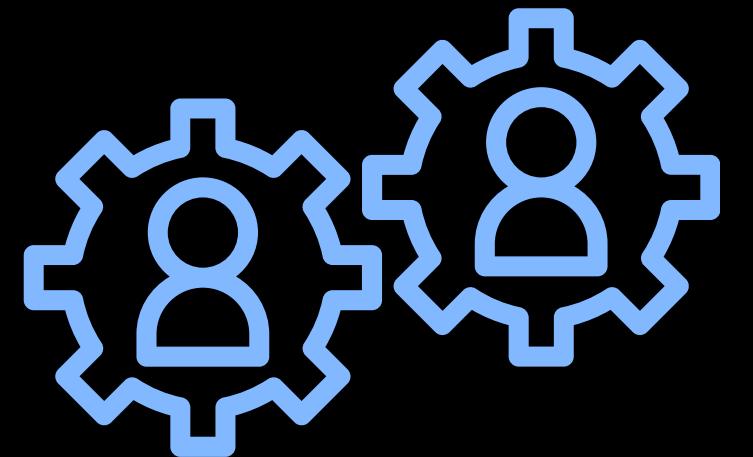


Funciones



Subprogramas

Subprogramas

Pequeños programas dentro de otros programas

- ✓ Unidades de ejecución independientes
- ✓ Encapsulan código y datos
- ✓ Se comunican con otros subprogramas (datos)

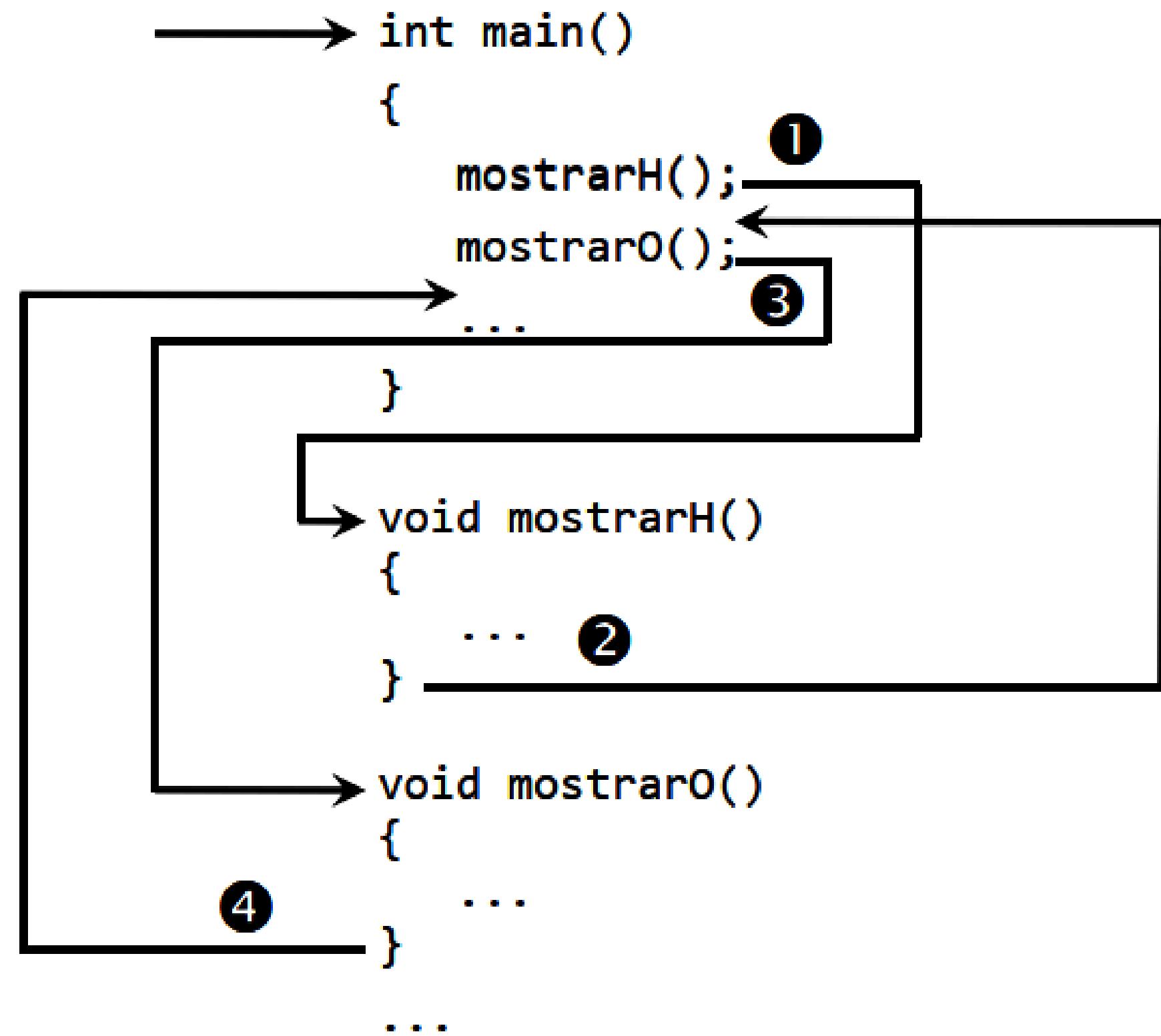
Subrutinas, procedimientos, funciones, acciones, ...

- ✓ Realizan tareas individuales del programa
- ✓ Funcionalidad concreta, identifiable y coherente (diseño)
- ✓ Se ejecutan de principio a fin cuando se llaman (*invocan*)
- ✓ Terminan devolviendo el control al punto de llamada



Aumentan el nivel de abstracción del programa
Facilitan la prueba, la depuración y el mantenimiento

Flujo de ejecución



Subprogramas

Subprogramas en C++

Forma general de un subprograma en C++:

```
tipo nombre(parámetros) // Cabecera
{
    // Cuerpo
}
```

- ✓ *Tipo* de dato que devuelve el subprograma como resultado
- ✓ *Parámetros* para la comunicación con el exterior
- ✓ *Cuerpo*: ¡Un bloque de código!

Tipos de subprogramas

Procedimientos (*acciones*):

NO devuelven ningún resultado de su ejecución con `return`

Tipo: `void`

Llamada: instrucción independiente

`mostrarH();`

Funciones:

SÍ devuelven un resultado con la instrucción `return`

Tipo distinto de `void`

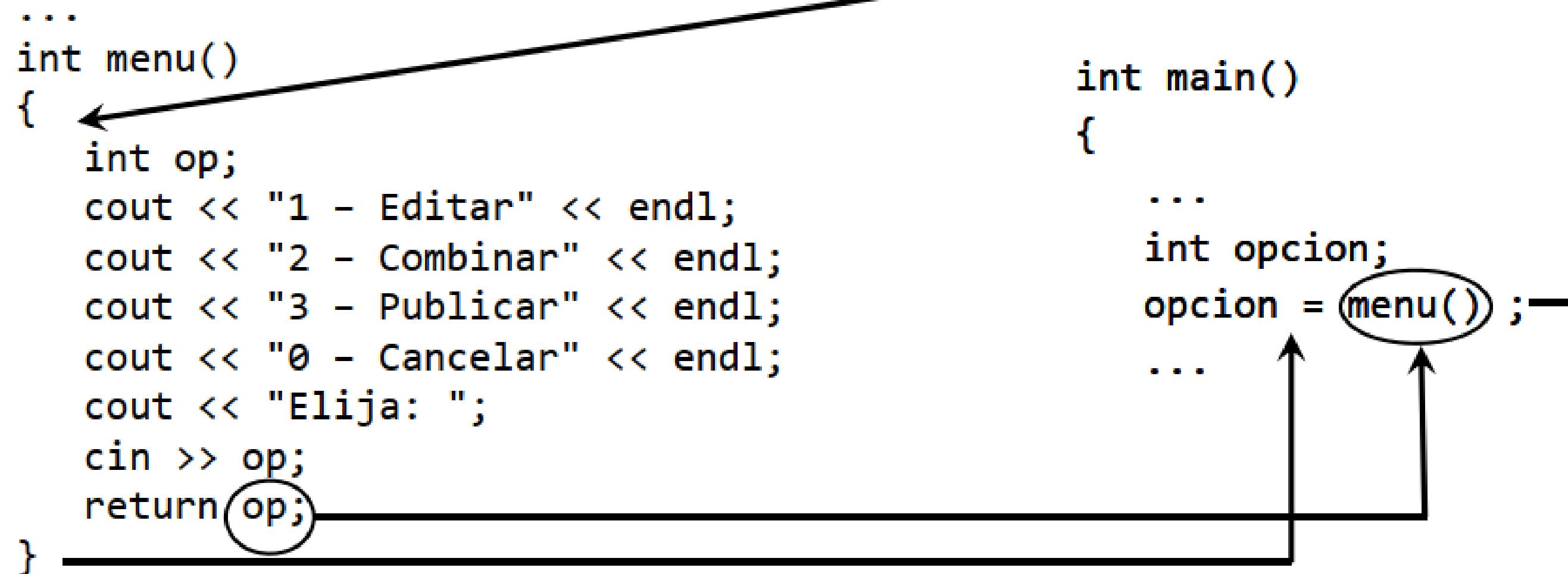
Llamada: dentro de cualquier expresión

`x = 12 * y + cuadrado(20) - 3;`

Se sustituye en la expresión por el valor que devuelve

Funciones

Subprogramas de tipo distinto de void



Procedimientos

Subprogramas de tipo void

```
...
void menu()
{
    int op;
    cout << "1 - Editar" << endl;
    cout << "2 - Combinar" << endl;
    cout << "0 - Cancelar" << endl;
    cout << "Opción: ";
    cin >> op;
    if (op == 1) {
        editar();
    }
    else if (op == 2) {
        combinar();
    }
}
```

```
int main()
{
    ...
    menu();
    ...
}
```

Subprogramas y datos

Datos en los subprogramas

De uso exclusivo del subprograma

```
tipo nombre(parámetros) // Cabecera
{
    Declaraciones Locales // Cuerpo
}
```

- ✓ Declaraciones locales de tipos, constantes y variables
Dentro del cuerpo del subprograma
- ✓ Parámetros declarados en la cabecera del subprograma
Comunicación del subprograma con otros subprogramas

Datos locales y datos globales

Datos en los programas

- ✓ Datos globales: declarados fuera de todos los subprogramas
Existen durante toda la ejecución del programa
- ✓ Datos locales: declarados en algún subprograma
Existen sólo durante la ejecución del subprograma

Ámbito y visibilidad de los datos

- Ámbito de los datos globales: resto del programa
Se conocen dentro de los subprogramas que siguen
- Ámbito de los datos locales: resto del subprograma
No se conocen fuera del subprograma
- Visibilidad de los datos
Datos locales a un bloque ocultan otros externos homónimos

Datos locales y datos globales

```
#include <iostream>
using namespace std;
```

```
const int MAX = 100;    }
double ingresos;        }
```

Datos globales

```
...
void proc() {
```

```
    int op;
```

```
    double ingresos; }
```

Datos locales a proc()

```
}
```

```
int main() {
```

```
    int op;
```

```
    ...
    return 0; }
```

Datos locales a main()

Se conocen MAX (global), op (local)
e ingresos (local que oculta la global)



op de proc()
es distinta
de op de main()

Sobre el uso de datos globales en los subprogramas

NO SE DEBEN USAR datos globales en subprogramas

- ✓ *¿Necesidad de datos externos?*

Define parámetros en el subprograma

Los datos externos se pasan como argumentos en la llamada

- ✓ Uso de datos globales en los subprogramas:

Riesgo de *efectos laterales*

Modificación inadvertida de esos datos afectando otros sitios

Excepciones:

- ✓ Constantes globales (valores inalterables)
- ✓ Tipos globales (necesarios en varios subprogramas)

Parámetros

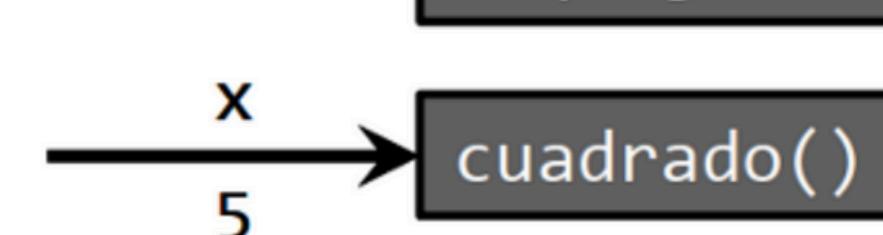
Comunicación con el exterior

Datos de entrada, datos de salida y datos de entrada/salida

Datos de entrada: Aceptados



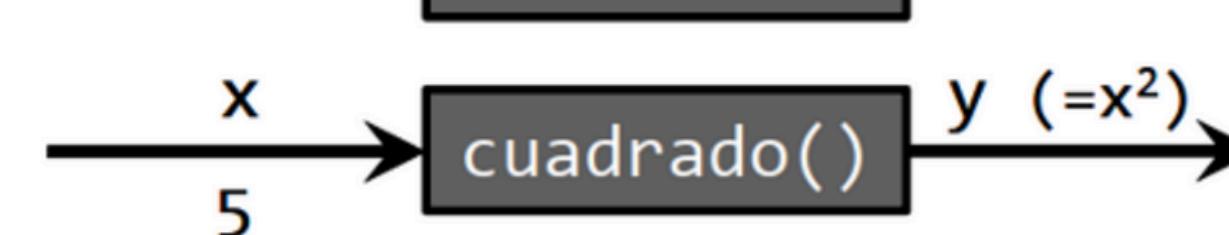
Subprograma que dado un número muestra en la pantalla su cuadrado:



Datos de salida: Devueltos



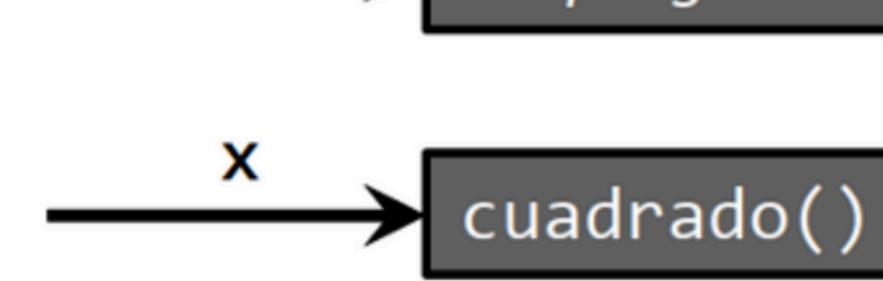
Subprograma que dado un número devuelve su cuadrado:



Datos de entrada/salida:
Aceptados y modificados



Subprograma que dada una variable numérica la eleva al cuadrado:



Parámetros en C++

Declaración de parámetros

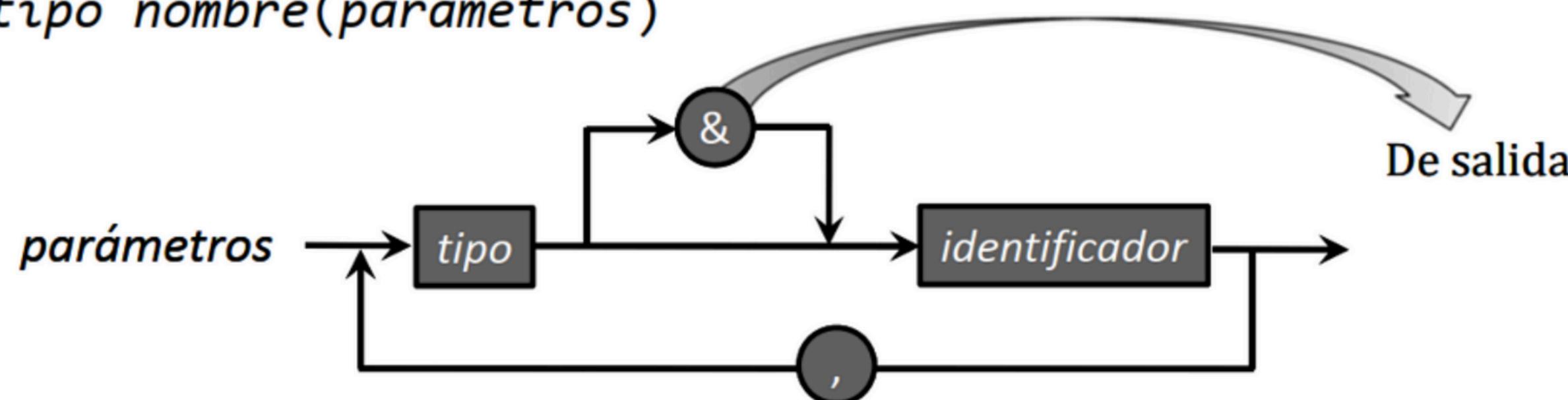
Sólo dos clases de parámetros en C++:

- Sólo de entrada (*por valor*)
- De salida (sólo salida o E/S) (*por referencia / por variable*)

Lista de parámetros formales

Entre los paréntesis de la cabecera del subprograma

tipo nombre(parámetros)



Parámetros por valor

Reciben copias de los argumentos usados en la llamada

```
int cuadrado(int num)
```

```
double potencia(double base, int exp)
```

```
void muestra(string nombre, int edad, string nif)
```

```
void proc(char c, int x, double a, bool b)
```

Reciben sus valores en la llamada del subprograma

Argumentos: Expresiones en general

Variables, constantes, literales, llamadas a función, operaciones

Se destruyen al terminar la ejecución del subprograma

¡Atención! Los arrays se pasan por valor como constantes:

```
void insertar(const int array[], int array_size)
```

Parámetros por referencia

&

Misma identidad que la variable pasada como argumento

```
void incrementa(int &x)
```

```
void intercambia(double &x, double &y)
```

```
void proc(char &c, int &x, double &a, bool &b)
```

Reciben las variables en la llamada del subprograma: *;Variables!*

Los argumentos pueden quedar modificados

;No usaremos parámetros por valor en las funciones!

Sólo en procedimientos



Puede haber tanto por valor como por referencia

;Atención! Los arrays se pasan por referencia sin utilizar &

```
void insertar(int array[], int array_size)
```

Argumentos

Llamada a subprogramas con parámetros

nombre(argumentos)

- Tantos argumentos como parámetros y en el mismo orden
- Concordancia de tipos argumento-parámetro
- Por valor: Expresiones válidas (se pasa el resultado)
- Por referencia: *¡Sólo variables!*

Se copian los valores de las expresiones pasadas por valor
en los correspondientes parámetros

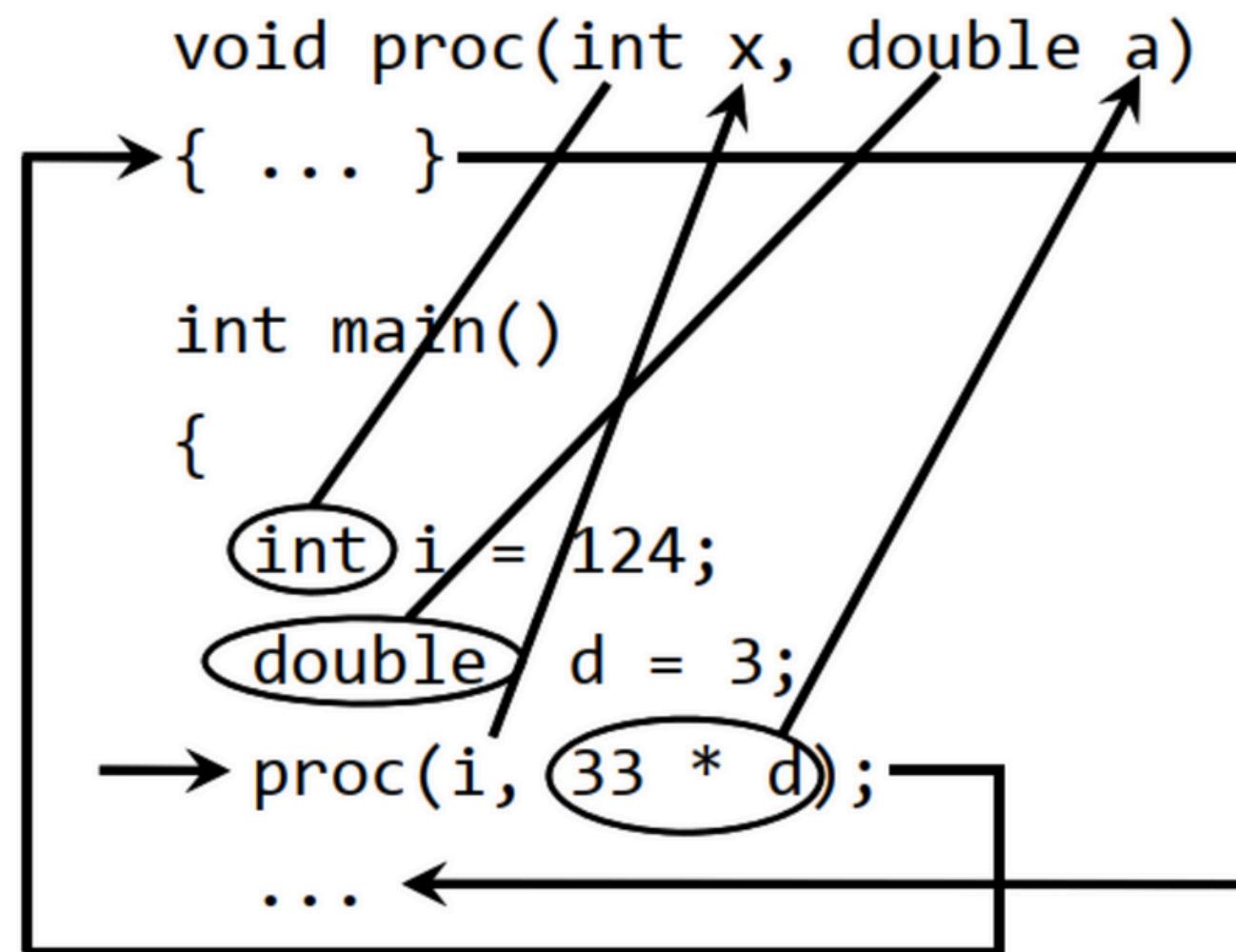
Se hacen corresponder los argumentos pasados por referencia
(variables) con sus correspondientes parámetros

Argumentos pasados por valor

Expresiones válidas con concordancia de tipo:

```
void proc(int x, double a) → proc(23 * 4 / 7, 13.5);  
→ double d = 3;  
proc(12, d);  
  
→ double d = 3;  
int i = 124;  
proc(i, 33 * d);  
  
→ double d = 3;  
int i = 124;  
proc(cuad(20) * 34 + i, i * d);
```

Argumentos pasados por valor

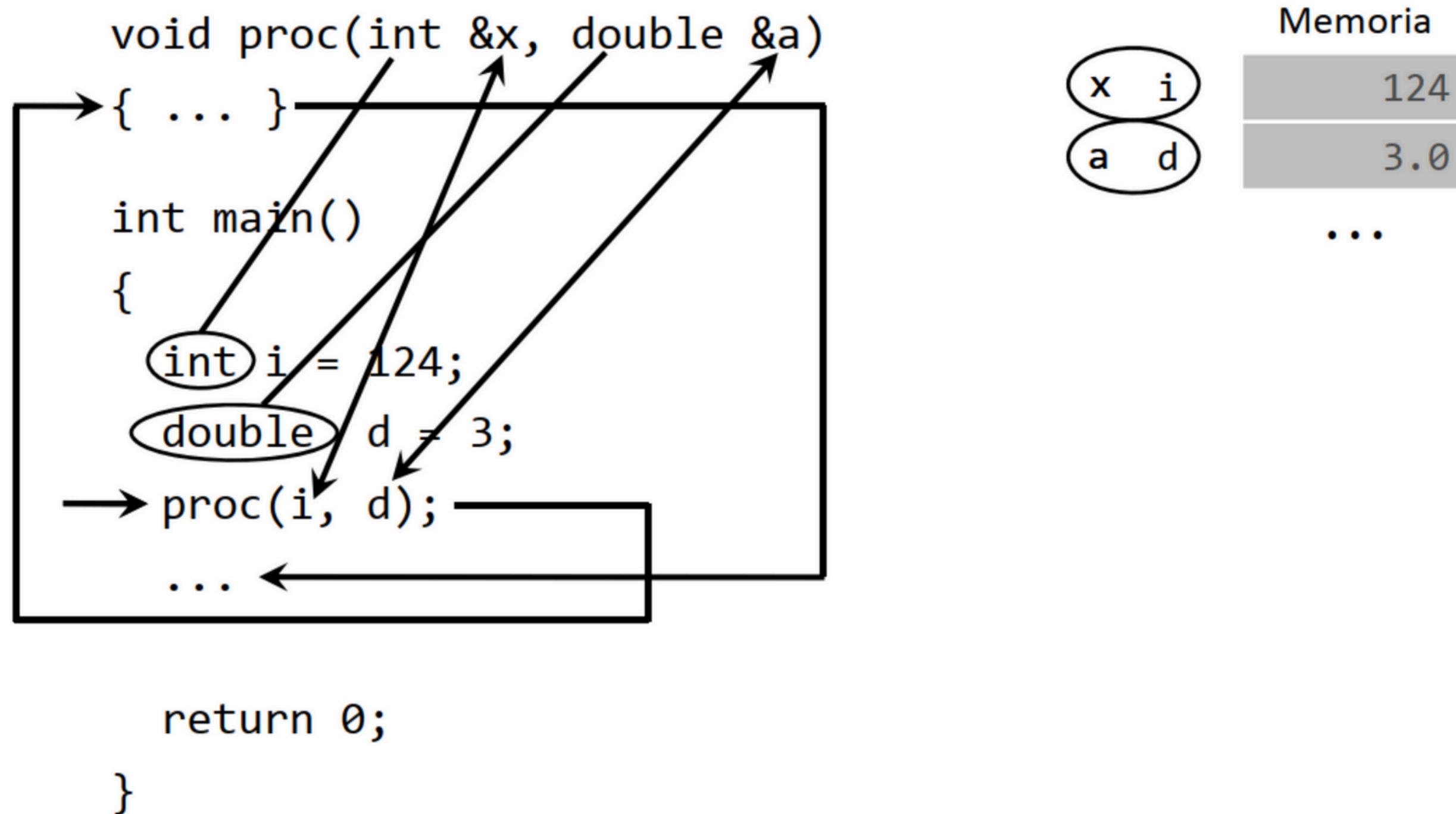


```
return 0;
```

```
}
```

Memoria	
i	124
d	3.0
	...
	...
x	124
a	99.0
	...

Argumentos pasados por referencia



¿Qué llamadas son correctas?

Dadas las siguientes declaraciones:

```
int i;
```

```
double d;
```

```
void proc(int x, double &a);
```

¿Qué pasos de argumentos son correctos? ¿Por qué no?

- | | | |
|----------------------|---|---------------------------------------|
| proc(3, i, d); | ✗ | Nº de argumentos ≠ Nº de parámetros |
| proc(i, d); | ✓ | |
| proc(3 * i + 12, d); | ✓ | |
| proc(i, 23); | ✗ | Parámetro por referencia → ¡variable! |
| proc(d, i); | ✗ | ¡Argumento double para parámetro int! |
| proc(3.5, d); | ✗ | ¡Argumento double para parámetro int! |
| proc(i); | ✗ | Nº de argumentos ≠ Nº de parámetros |