

Programación 1

Tema 6

Procedimientos y funciones



Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza





Índice

- **Procedimientos y funciones**
- Especificación de funciones
- Ámbito y vida
- Comunicación entre funciones



Procedimientos y funciones

- Abstracción de un conjunto de instrucciones
 - que realizan una tarea concreta
 - a las que se le da un nombre determinado
 - para ser invocadas desde algún otro punto del programa



Ejemplo

escribirTabla en la práctica 2

```
/*  
 * Escribe en la pantalla la tabla de multiplicar del «n».  
 */  
void presentarTabla(unsigned n) {  
    // Escribe la cabecera de la tabla de multiplicar del «n»  
    cout << endl;  
    cout << "LA TABLA DEL " << n << endl;  
  
    // Escribe las 11 líneas de la tabla de multiplicar del «n»  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```

1 Conjunto de instrucciones...



Ejemplo

escribirTabla en la práctica 2

2 ...que realizan una tarea concreta

```
/*  
 * Escribe en la pantalla la tabla de multiplicar del «n».  
 */  
void presentarTabla(unsigned n) {  
    // Escribe la cabecera de la tabla de multiplicar del «n»  
    cout << endl;  
    cout << "LA TABLA DEL " << n << endl;  
  
    // Escribe las 11 líneas de la tabla de multiplicar del «n»  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```



Ejemplo

escribirTabla en la práctica 2

```
/*  
 * Escribe en la pantalla la tabla de multiplicar del «n».  
 */  
void presentarTabla(unsigned n) {  
    // Escribe la cabecera de la tabla de multiplicar del «n»  
    cout << endl;  
    cout << "LA TABLA DEL " << n << endl;  
  
    // Escribe las 11 líneas de la tabla de multiplicar del «n»  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```

3 ...con un nombre determinado



Ejemplo

escribirTabla en la práctica 2

```
int main() {  
    cout << "Tabla que desea escribir ("  
        << FIN << " para acabar): ";  
    unsigned multiplicando;  
    cin >> multiplicando;  
    while (multiplicando != FIN) {  
        presentarTabla(multiplicando);  
        cout << endl;  
        cout << "Tabla que desea escribir ("  
            << FIN << " para acabar): ";  
        cin >> multiplicando;  
    }  
}
```

4 ... para ser invocadas desde algún otro punto del programa



Comparación con versión sin procedimientos

```
#include <iostream>
#include <iomanip>
using namespace std;
const unsigned FIN = 0;
int main() {
    cout << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";

    unsigned multiplicando;
    cin >> multiplicando;

    while (multiplicando != FIN) {
        cout << endl;
        cout << "LA TABLA DEL " << multiplicando << endl;

        for (unsigned i = 0; i <= 10; ++i) {
            cout << setw(3) << multiplicando << " x " << setw(2) << i << " = "
                << setw(3) << multiplicando * i << endl;
        }

        cout << endl << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";
        cin >> multiplicando;
    }
}
```



Funciones

- Pretenden ser un reflejo del concepto matemático de *función*:
 - Relación binaria entre dos conjuntos que asocia a cada elemento del primer conjunto exactamente un elemento del segundo conjunto.
 - $f: \mathbb{R} \rightarrow \mathbb{R}$
 - $x \mapsto f(x)$



Funciones

Ejemplo matemático

□ Ejemplo:

■ $f: \mathbb{R} \rightarrow \mathbb{R}$

■ $x \mapsto x^2$



Funciones

Ejemplo





Funciones

Ejemplo en C++

```
/*  
 * Devuelve el cuadrado de x  
 */  
double f(double x) {  
    return x * x;  
}
```



Funciones

Ejemplo en C++

```
/*  
 * Devuelve el cuadrado de x  
 */  
double cuadrado(double x) {  
    return x * x;  
}
```



Funciones en C++

- Sintaxis:
 - Declaración
 - Definición
 - Invocación



Más adelante en el curso, no ahora



Funciones. Sintaxis definición

```
<definición-función> ::=
    <tipo> <identificador>
        “(” [<lista-parámetros>] “)”
    <bloque-secuencial>

<lista-parámetros> ::=
    <parámetro> { “,” <parámetro> }

<parámetro> ::=
    <tipo> <identificador>
```



Funciones

Ejemplo en C++

```
/*  
 * Devuelve el cuadrado de x  
 */  
double cuadrado(double x) {  
    return x * x;  
}
```



Funciones. Sintaxis invocación

```
<invocación-función> ::=  
    <identificador>  
    “(” [ <lista-argumentos> ] “)”  
<lista-argumentos> ::=  
    <argumento> {“,” <argumento>}  
<argumento> ::= <expresión>
```



Funciones

Ejemplos de invocaciones

... cuadrado(2) ...

... cuadrado(y) ...

... cuadrado(2 + y) ...

... cuadrado(cuadrado(sqrt(y))) ...



Funciones

El resultado de la invocación a una función es un valor de un tipo de datos...

```
double y = cuadrado(2);  
cout << cuadrado(y) << endl;  
y = 3 * cuadrado(2 + y);  
if (y == cuadrado(sqrt(y))) {  
    ...  
}
```



Funciones. Ejemplo factorial

- *factorial*: $\mathbb{N} \rightarrow \mathbb{N}$
- $n \mapsto n!$



Funciones. Ejemplo factorial



Funciones. Ejemplo factorial

```
/* Definición */  
unsigned factorial(unsigned n) {  
    unsigned fact = 1;  
    for (unsigned i = 1; i <= n; i++) {  
        fact = i * fact;  
    }  
    return fact;  
}
```

```
/* Posibles invocaciones */  
factorial(8)  
factorial(n)  
factorial(m + 1)  
factorial(2 * n * m)
```

Funciones. Ejemplo factorial

Programa del tema 4 con funciones

```
/* Devuelve n! */  
unsigned factorial(unsigned n) {  
    unsigned factorial = 1;  
    for (unsigned i = 1; i <= n; i++) {  
        factorial = i * factorial;  
    }  
    return factorial;  
}
```

```
/* Programa que pide al usuario un número natural, lo lee del  
 * teclado y escribe en la pantalla su factorial. */  
int main() {  
    cout << "Escriba un número natural: ";  
    unsigned n;  
    cin >> n;  
    cout << n << "! = " << factorial(n) << endl;  
}
```



Funciones. Un tercer ejemplo: esPrimo

esPrimo: $\mathbb{N} \rightarrow \mathbb{B}$

$$n \mapsto \begin{cases} \text{true} & \text{si } n \text{ es un número primo} \\ \text{false} & \text{si } n \text{ no es un número primo} \end{cases}$$



Funciones. Un tercer ejemplo: esPrimo



Funciones. Un tercer ejemplo: esPrimo

```
/*  
 * Devuelve true si y solo si «n» es un número primo.  
 */  
bool esPrimo(unsigned n) {  
    if (n == 2) {  
        return true;  
    } else if (n < 2 || n % 2 == 0) {  
        return false;  
    } else {  
        // «divisor» indica el siguiente impar candidato a dividir a «n».  
        unsigned divisor = 3;  
        // «encontrado» indica si se ha encontrado un divisor de «n».  
        bool encontrado = false;  
  
        while (!encontrado && divisor * divisor <= n) {  
            encontrado = n % divisor == 0;  
            divisor = divisor + 2;  
        }  
        return !encontrado;  
    }  
}
```

Funciones. Un tercer ejemplo: esPrimo

```
/*  
 * Programa que pide un entero y escribe en la pantalla si es  
 * primo o no.  
 */  
int main() {  
    cout << "Escriba un número entero: ";  
    unsigned numero;  
    cin >> numero;  
  
    cout << "El número " << numero;  
    if (!esPrimo(numero)) {  
        cout << " no";  
    }  
    cout << " es primo." << endl;  
}
```



Procedimientos

Funciones que no devuelven valor

```
void presentarTabla(unsigned n) {  
    cout << endl << "LA TABLA DEL " << n << endl;  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```





Funciones. Sintaxis

- Restricciones a la sintaxis:
 - Si el tipo devuelto es distinto de **void**, el cuerpo de la función debe devolver un dato del tipo adecuado a través de la instrucción **return**.
 - El identificador de la invocación es el mismo que el de la definición.
 - La lista de parámetros (definición) y la de argumentos (invocación) tienen el mismo número de elementos.
 - El tipo del i -ésimo argumento en la lista de argumentos es el mismo (o es compatible) con el i -ésimo parámetro de la definición.

Función main()

Si el tipo devuelto es distinto de **void**, el cuerpo de la función debe devolver un dato del tipo adecuado a través de la instrucción **return**.

```
#include <cmath>
#include <iostream>
using namespace std;
/*
 * Solicita al usuario la longitud de un radio y escribe en la
 * pantalla el área del círculo correspondiente.
 */
int main() {
    cout << "Escriba el radio de un círculo: ";

    double r;
    cin >> r;

    cout << "El área de un círculo de radio " << r << " es "
         << M_PI * r * r << endl;
}
```



Función main()

```
#include <cmath>
#include <iostream>
using namespace std;
/*
 * Solicita al usuario la longitud de un radio y escribe en la
 * pantalla el área del círculo correspondiente.
 */
int main() {
    cout << "Escriba el radio de un círculo: ";

    double r;
    cin >> r;

    cout << "El área de un círculo de radio " << r << " es "
         << M_PI * r * r << endl;
    return 0;
}
```



Función main()

```
#include <cmath>
#include <iostream>
using namespace std;
/*
 * Solicita al usuario la longitud de un radio y escribe en la
 * pantalla el área del círculo correspondiente.
 */
int main() {
    cout << "Escriba el radio de un círculo: ";
    double r;
    cin >> r;

    if (r > 0) {
        cout << "El área de un círculo de radio " << r << " es "
             << M_PI * r * r << endl;
        return 0;
    } else {
        cout << "El radio tiene que ser positivo." << endl;
        return 1;
    }
}
```



Funciones

Otro ejemplo de definición

```
/*  
 * Dado un polígono regular con un número de Lados  
 * igual al valor del parámetro «numLados» de  
 * Longitud igual al valor del parámetro «Longitud»,  
 * devuelve el perímetro de dicho polígono regular.  
 * «numLados» tiene que ser mayor o igual que 3 y  
 * «Longitud» mayor que 0.0.  
 */  
double perimetro(unsigned numLados,  
                 double longitud) {  
    return numLados * longitud;  
}
```



Funciones

Ejemplo perímetro





Funciones

Ejemplos de invocaciones a perimetro

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);  
  
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



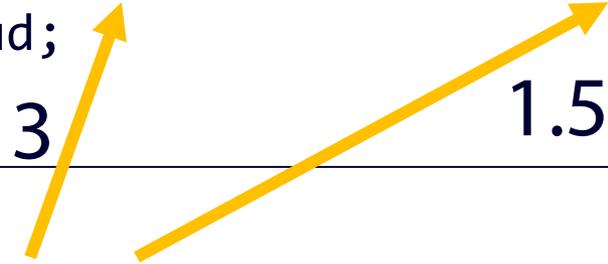
Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

3

1.5



...

```
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);
```

```
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

4.5

...

```
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);
```

```
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

4

2.2

...

```
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);
```

```
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

8.8

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);  
  
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);  
  
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```

5

3.25



Funciones

Ejemplos de invocaciones a perimetro

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

16.25

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);  
  
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Funciones

Ejemplos de invocaciones a perimetro

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

16.25

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);
```

3.25

```
unsigned lados = 5;  
double longitudLado = 3.25;  
double pentagono = perimetro(lados, longitudLado);
```

5

```
cout << perimetro(lados + 1, longitudLado - 1) << endl;
```



Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

...

```
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);
```

```
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);
```

```
cout << perimetro(numLados + 1, longitud - 1) << endl;
```

6

2.25

Funciones

Ejemplos de invocaciones a `perimetro`

```
double perimetro(unsigned numLados, double longitud) {  
    return numLados * longitud;  
}
```

13.5

```
...  
double triangulo = perimetro(3, 1.5);  
double cuadrado = perimetro(4, 2.2);  
  
unsigned numLados = 5;  
double longitud = 3.25;  
double pentagono = perimetro(numLados, longitud);  
  
cout << perimetro(numLados + 1, longitud - 1) << endl;
```



Índice

- Funciones
- **Especificación de funciones**
- Ámbito y vida
- Comunicación entre funciones



Especificación de funciones

Función perímetro

```
/*  
 * Dado un polígono regular con un número de Lados  
 * igual al valor del parámetro «numLados» de  
 * Longitud igual al valor del parámetro «Longitud»,  
 * devuelve el perímetro de dicho polígono regular.  
 * «numLados» tiene que ser mayor o igual que 3 y  
 * «Longitud» mayor que 0.0.  
 */  
double perimetro(unsigned numLados,  
                 double longitud) {  
    return numLados * longitud;  
}
```

Especificación de funciones

```
/*  
 * Pre:  P  
 * Post: Q  
 */  
void f() {  
    ...  
}
```

Si se cumple la precondición P inmediatamente antes de invocar a la función f , entonces f se ejecuta, termina y se alcanza un estado en el que se cumple la postcondición Q .



Especificación de funciones

Función perímetro

```
/*  
 * Pre:  numLados  $\geq 3$  y Longitud  $> 0.0$   
 * Post: Devuelve el perímetro de un polígono  
 *       regular con un número de lados igual al  
 *       valor del parámetro «numLados» de Longitud  
 *       igual al valor del parámetro «Longitud».  
 */  
double perimetro(unsigned numLados,  
                 double longitud) {  
    return numLados * longitud;  
}
```

Especificación de funciones

```
/*  
 * Pre: ---  
 * Post: Devuelve el valor del polinomio  
 * ax2 + bx + c  
 */  
double calcular(double a, double b, double c,  
                double x) {  
    return ((a * x + b) * x) + c;  
}
```

Especificación de funciones

```
/*  
 * Pre: n >= 0  
 * Post: Devuelve el valor de n!  
 */  
unsigned factorial(int n) {  
    ...  
}
```



Especificación de funciones

```
/*  
 * Pre: ---  
 * Post: Devuelve el valor de n!  
 */  
unsigned factorial(unsigned n) {  
    ...  
}
```

Especificación de funciones

```
/*  
 * Pre:  $1 \leq dia \leq 31, 1 \leq mes \leq 12, agno > 0$   
 * Post: Ha escrito en la pantalla una línea con  
 * La fecha definida por los valores de  
 * Los parámetros «dia», «mes» y «agno»  
 * con el siguiente formato: dia/mes/agno.  
 * Por ejemplo: 12/1/2014  
*/  
void escribirFecha(unsigned dia, unsigned mes,  
                  unsigned agno) {  
    cout << dia << "/" << mes << "/" << agno << endl;  
}
```



Ejemplo

Escriba dos enteros: 12 18

Máximo común divisor: 6

Mínimo común múltiplo: 36

Escriba dos enteros: -72 50

Máximo común divisor: 2

Mínimo común múltiplo: 1800

Escriba dos enteros: 0 0

Ambos números no pueden ser 0 a la vez.

Solución sin funciones

```
/*
 * Programa que solicita al usuario dos números enteros y escribe en la pantalla
 * su máximo común divisor y su mínimo común múltiplo.
 */
int main() {
    cout << "Escriba dos enteros: ";
    int numero1, numero2;
    cin >> numero1 >> numero2;

    if (numero1 == 0 && numero2 == 0) {
        cout << "Ambos números no pueden ser 0 simultáneamente." << endl;
        return 1;
    } else {
        if (numero1 < 0) {
            numero1 = -numero1;
        }
        if (numero2 < 0) {
            numero2 = -numero2;
        }
        unsigned a = numero1;
        unsigned b = numero2;

        while (b != 0) {
            unsigned resto = a % b;
            a = b;
            b = resto;
        }
        cout << "Máximo común divisor: " << a << endl;
        cout << "Mínimo común múltiplo: " << numero1 * numero2 / a << endl;
        return 0;
    }
}
```

Solución con funciones

```
/*
 * Pre: ---
 * Post: Devuelve |a|.
 */
unsigned absoluto(int a) {
    ...
}

/*
 * Pre: a ≠ 0 o b ≠ 0
 * Post: Devuelve el máximo común divisor de «a» y «b».
 */
unsigned mcd(unsigned a, unsigned b) {
    ...
}

/*
 * Pre: a ≠ 0 o b ≠ 0
 * Post: Devuelve el mínimo común múltiplo de «a» y «b».
 */
unsigned mcm(unsigned a, unsigned b) {
    ...
}
```



Solución con funciones

```
/*  
 * Programa que solicita al usuario dos números enteros y escribe en la  
 * pantalla su máximo común divisor y su mínimo común múltiplo.  
 */  
int main() {  
    cout << "Escriba dos enteros: ";  
    int numero1, numero2;  
    cin >> numero1 >> numero2;  
  
    if (numero1 == 0 && numero2 == 0) {  
        cout << "Ambos números no pueden ser 0 simultáneamente." << endl;  
        return 1;  
    } else {  
        numero1 = absoluto(numero1);  
        numero2 = absoluto(numero2);  
        cout << "Máximo común divisor: " << mcd(numero1, numero2) << endl;  
        cout << "Mínimo común múltiplo: " << mcm(numero1, numero2) << endl;  
        return 0;  
    }  
}
```



Índice

- Funciones
- Especificación de funciones
- **Ámbito y vida**
- Comunicación entre funciones

Ámbito y vida

- Elemento nombrado con un identificador: función, constante, variable, parámetro, ...
- **Ámbito o visibilidad** (*scope*): zona del código en la que un elemento es accesible (se puede hacer uso de él).
 - **Ámbito local** de los elementos definidos dentro de un bloque o función:
 - Desde el punto en que se definen hasta el final del bloque o función.
 - **Ámbito global** de los elementos definidos en el fichero fuera de las funciones:
 - Desde el punto en que se han definido hasta el final del fichero.
- **Duración o vida** (*lifetime*) de un elemento
 - Tiempo en el que el elemento existe durante la ejecución del programa.

Ámbito. Ejemplo

```
#include <iostream>
#include <iomanip>
using namespace std;
const int FIN = 0;

void presentarTabla(int n) {
    cout << endl;
    cout << "LA TABLA DEL " << n << endl;

    for (int i = 0; i <= 10; i++) {
        cout << setw(3) << n
            << " x " << setw(2) << i
            << " = " << setw(3) << n * i
            << endl;
    }
}

int main() {
    cout << "¿Qué tabla desea escribir?: "...
    int multiplicando;
    cin >> multiplicando;
    while (multiplicando != FIN) {
        presentarTabla(multiplicando);
        cout << endl << "¿Qué tabla desea...
        cin >> multiplicando;
    }

    return 0;
}
```

prefijo std

FIN

presentarTabla

main

n

i

multiplicando



Índice

- Funciones
- Especificación de funciones
- Ámbito y vida
- **Comunicación entre funciones**



Comunicación entre funciones

- ❑ Parámetros por valor  Predeterminado
- ❑ Parámetros por referencia
- ❑ Valor devuelto  Ya visto
- ❑ Variables globales



Otro problema distinto

```
int a, b;
```

```
...
```

```
// Si ahora: a = X y b = Y ...
```

```
intercambiar(a, b);
```

```
// ... entonces ahora: a = Y y b = X
```



Intercambiar



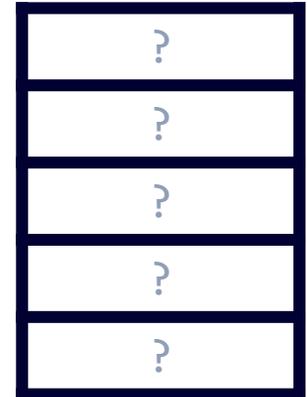
Comunicación por valor. Solución errónea

```
/*  
 * Pre:  uno =  $X_0$  y otro =  $Y_0$   
 * Post: uno =  $Y_0$  y otro =  $X_0$   
 */  
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

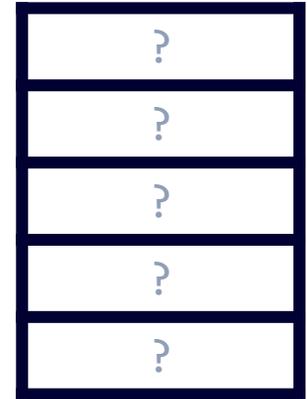
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a

20
?
?
?
?

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a

20
?
?
?
?

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

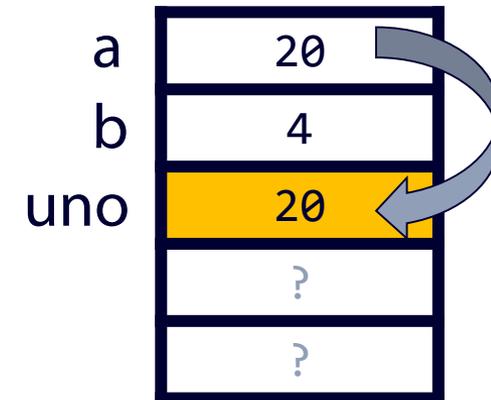
Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

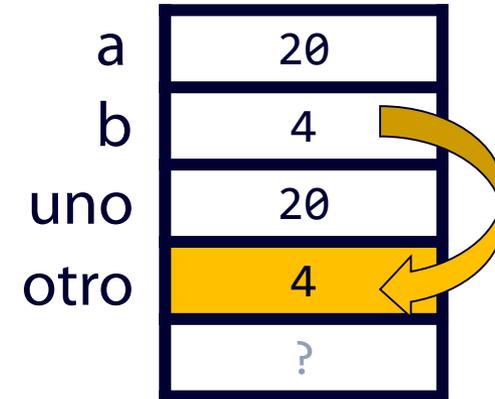
Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	20
otro	4
	?

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	20
otro	4
aux	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	20
otro	4
aux	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	4
otro	4
aux	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```



```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	4
otro	4
aux	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```



```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	4
otro	20
aux	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

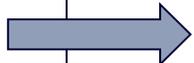
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
uno	4
otro	20
	20

Parámetros por valor

```
void intercambiar(int uno, int otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	4
	20
	20





Comunicación por valor. Solución errónea

- Ejecución en C++ Tutor

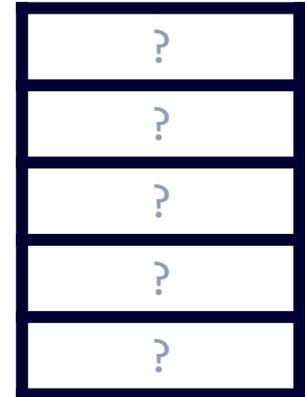


Comunicación mediante parámetros por referencia

```
/*  
 * Pre: uno =  $X_0$  y otro =  $Y_0$   
 * Post: uno =  $Y_0$  y otro =  $X_0$   
 */  
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

Parámetros por referencia

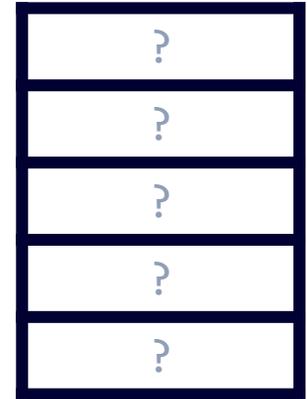
```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a

20
?
?
?
?

Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a

20
?
?
?
?

Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

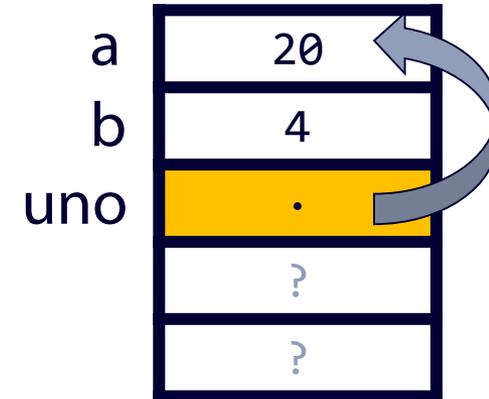
Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	20
b	4
	?
	?
	?

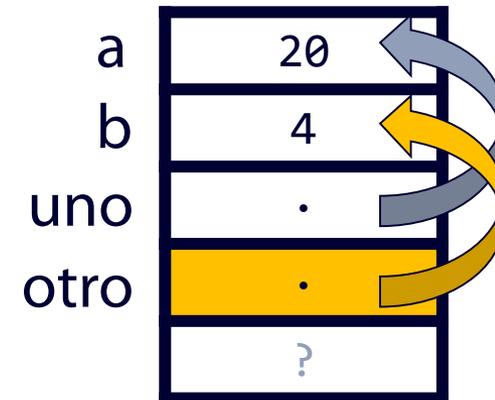
Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



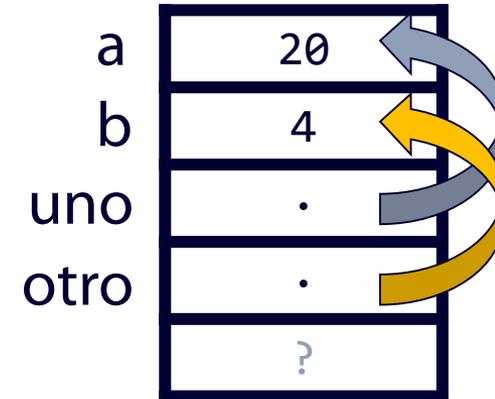
Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



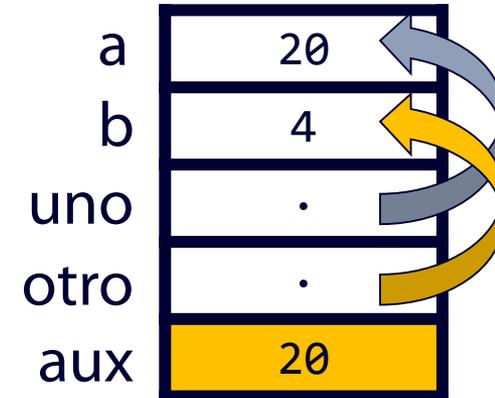
Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

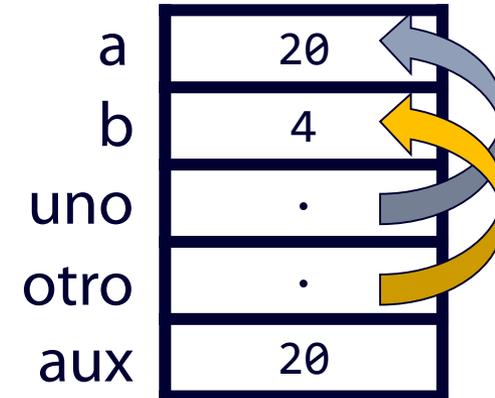
```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

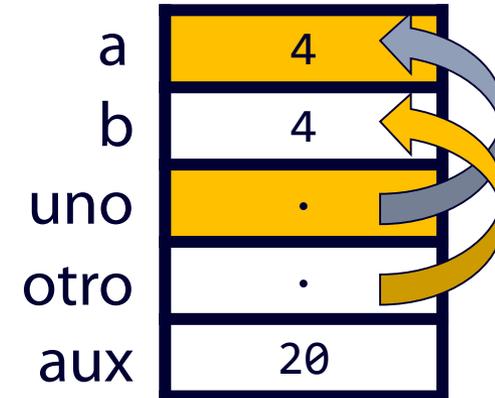
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

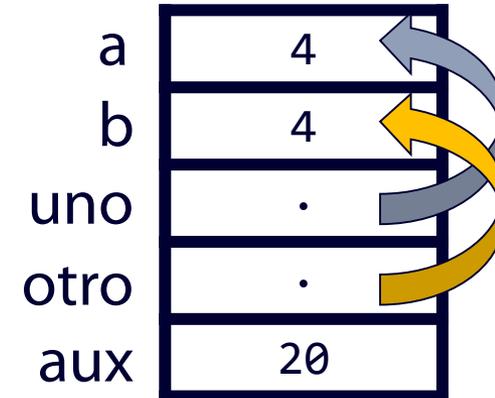


Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```



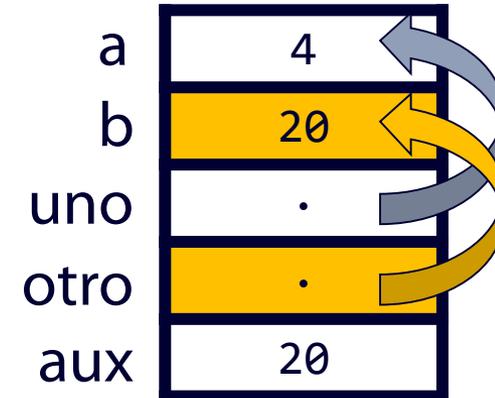
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

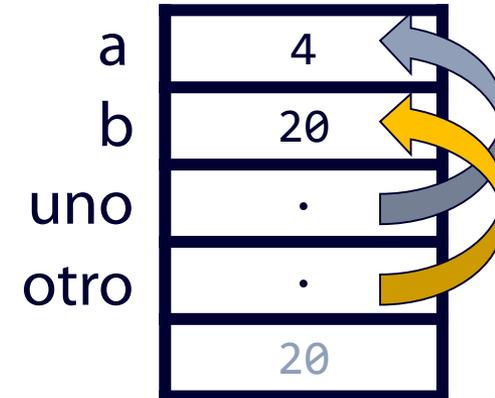
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}
```

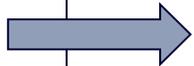
```
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```



Parámetros por referencia

```
void intercambiar(int& uno, int& otro) {  
    int aux = uno;  
    uno = otro;  
    otro = aux;  
}  
  
int main() {  
    int a = 20;  
    int b = 4;  
    intercambiar(a, b);  
    cout << a << " " << b << endl;  
    return 0;  
}
```

a	4
b	20
	.
	.
	20





Comunicación mediante parámetros por referencia

- Ejecución en C++ Tutor



Intercambiar





Comunicación mediante parámetros por referencia

```
/*  
 * Pre: ---  
 * Post: Ha asignado a los parámetros «nacimiento», «estatura» y «peso» los  
 *       valores determinados por el usuario como respuesta a tres preguntas  
 *       que le son formuladas acerca de su año de nacimiento, su  
 *       estatura y su peso.  
*/  
void preguntarDatos(int& nacimiento, double& estatura, double& peso) {  
    cout << "Escriba año de nacimiento: ";           // 1.ª pregunta  
    cin >> nacimiento;                               // lee la respuesta  
    cout << "Su estatura: ";                          // 2.ª pregunta  
    cin >> estatura;                                  // lee la respuesta  
    cout << "Su peso: ";                               // 3.ª pregunta  
    cin >> peso;                                      // lee la respuesta  
}
```



preguntarDatos





preguntarDatos





Comunicación entre funciones

	Parámetros por valor	Parámetros por referencia
Dirección de la comunicación	De la función invocante a la invocada	De la función invocante a la invocada, de la invocada a la invocante o ambas
Argumentos en la invocación	Cualquier expresión que al ser evaluada obtenga un valor compatible con el tipo del parámetro	Solo variables del mismo tipo que el parámetro



Comunicación entre funciones

- Parámetros por valor
- Parámetros por referencia
- Valor devuelto
- **Variables globales**
 - **¡PROHIBIDAS EN ESTE CURSO!**

Variables globales

Constante global → ok

```
...  
const int FIN = 0;  
int n;  
  
void presentarTabla() {  
    cout << endl << "LA TABLA DEL " << n << endl;  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}  
  
int main() {  
    cout << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
    cin >> n;  
    while (n != FIN) {  
        presentarTabla();  
        cout << endl << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
        cin >> n;  
    }  
    return 0;  
}
```

Variable global → ¡NO!



Problemas del uso de variables globales

- Diseño dependiente del nombre de las variables globales
 - Reducción de la capacidad de reutilización del código
- Efectos laterales debidos a la posibilidad de modificación de sus valores desde cualquier parte del código
- Reducción de la legibilidad de las funciones



Variables globales

Constante global → ok

```
...  
const int FIN = 0;  
int n;
```

Variable global → ¡NO!

```
void presentarTabla() {  
    cout << endl << "LA TABLA DEL " << n << endl;  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```

¿La tabla de qué número
va a escribir?

No resulta evidente; hay que leer
el código del procedimiento para
averiguarlo.

```
int main() {  
    cout << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
    cin >> n;  
    while (n != FIN) {  
        presentarTabla();  
        cout << endl << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
        cin >> n;  
    }  
    return 0;  
}
```

¿La tabla de qué número
queremos escribir?

No resulta evidente; hay que leer
el código del procedimiento
presentarTabla para averiguarlo.

Variables globales

```
...  
const int FIN = 0;  
int n;
```

```
void presentarTabla() {  
    cout << endl << "LA TABLA DEL " << n << endl;  
    unsigned i = 0;  
    while (i <= 10) {  
        cout << n << " x " << i << " = " << n * i << endl;  
        i++;  
    }  
}
```

¿Si quiero reutilizar la función `presentarTabla()` en otro programa, qué me tengo que «llevar»?

```
int main() {  
    cout << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
    cin >> n;  
    while (n != FIN) {  
        presentarTabla();  
        cout << endl << "¿Qué tabla desea escribir (" << FIN << " para acabar)?: ";  
        cin >> n;  
    }  
    return 0;  
}
```



Índice

- Funciones
- Especificación de funciones
- Ámbito y vida
- Comunicación entre funciones



¿Cómo se puede estudiar este tema?

- Repasando estas transparencias
- Trabajando con el código de estas transparencias
 - <https://github.com/prog1-eina/tema-06-funciones>
- Leyendo el material adicional dispuesto en Moodle:
 - Enlaces a tutoriales de Cplusplus.com y Tutorials Point
 - Capítulo 4 de los apuntes del profesor Martínez
- Realizando los problemas de las próximas clases de problemas
- Realizando algunos de los ejercicios básicos sobre funciones disponibles en Moodle:
 - <https://moodle.unizar.es/add/mod/page/view.php?id=3807903>