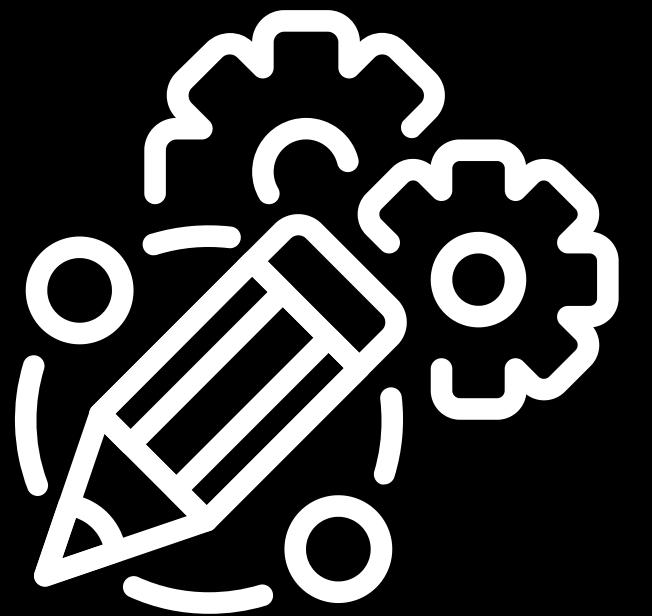


Funciones



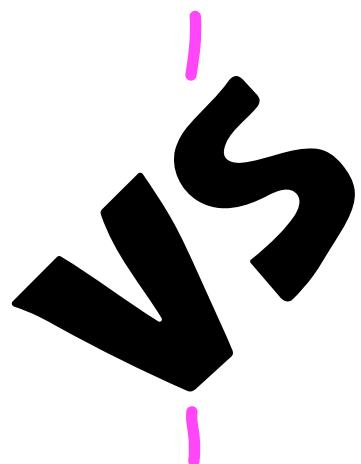
Pasaje por valor

Desde el algoritmo principal mandamos el valor de una variable como parámetro a un subprograma.

El subprograma recibe una copia de ella, y cualquier modificación que haga no se verá reflejada en el algoritmo principal ni en otra función o procedimiento, pero sí dentro del mismo subprograma.

Es la forma por defecto de enviar parámetros.

Pasaje por referencia



En vez de pasar el valor alojado en una variable, se pasa la referencia a esa variable.

La referencia de una variable es la dirección de memoria en donde está alojada.

En este caso, los cambios que realice el subprograma se verán reflejados fuera de este.

Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
    // Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            cout << i << " entre " << j << " da un cociente de "
                << cociente << " y un resto de " << resto << endl;
        }
    }

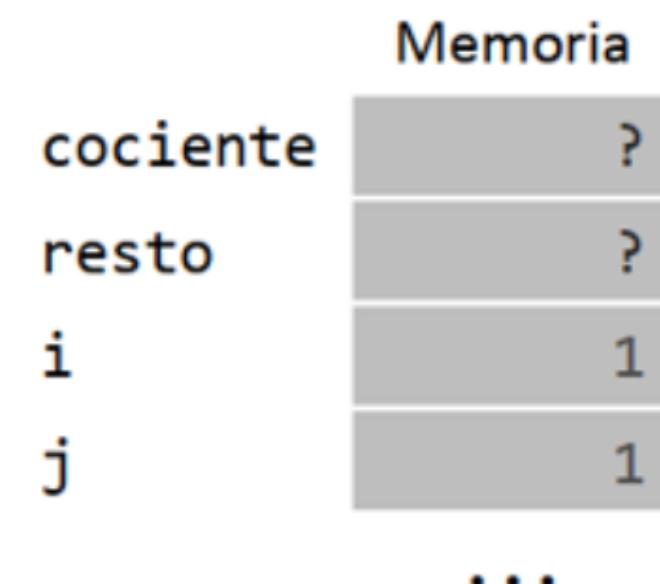
    return 0;
}
```

Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
    // Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            → divide(i, j, cociente, resto);
            ...
        }
    }

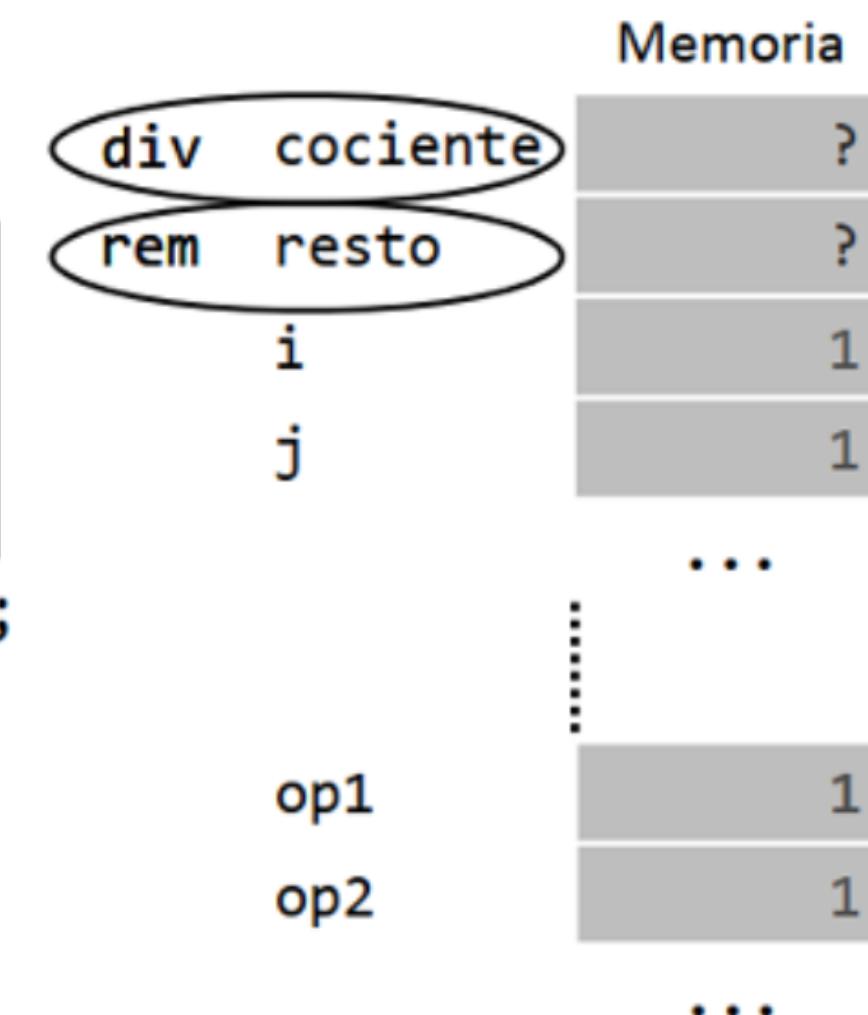
    return 0;
}
```



Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
    → // Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

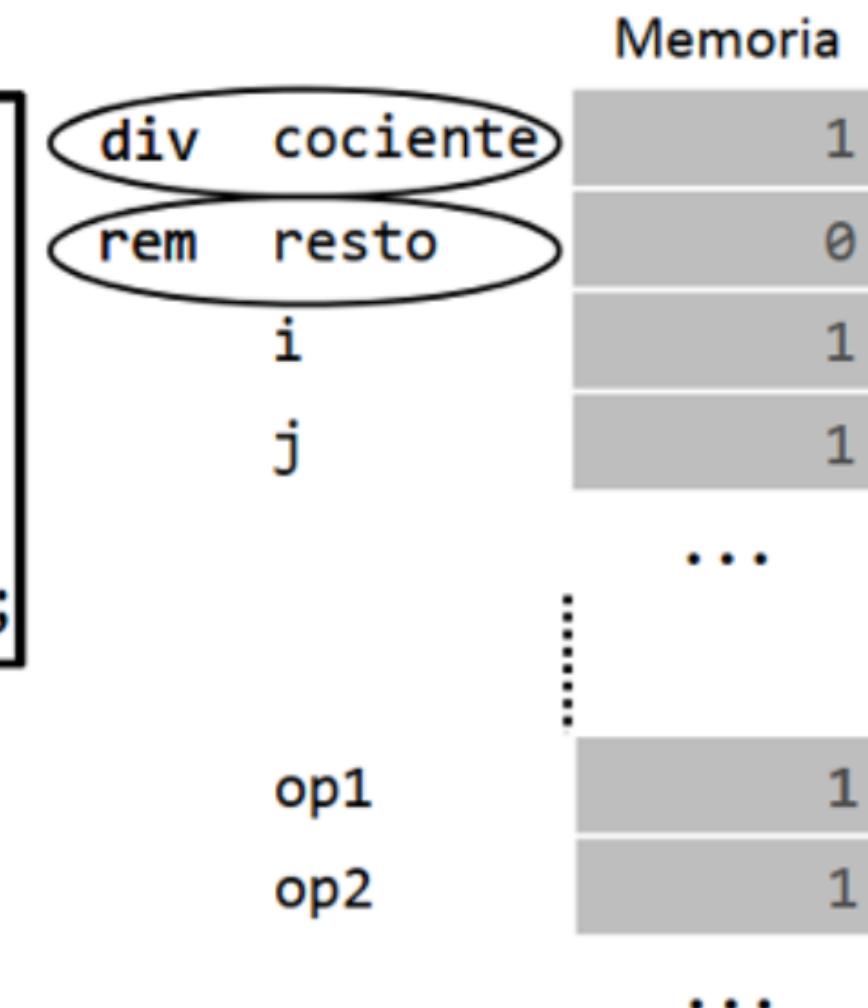
int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            ...
        }
    }
    return 0;
}
```



Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
    // Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            ...
        }
    }
    return 0;
}
```



Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
    // Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            ...
        }
    }

    return 0;
}
```

Memoria	
cociente	1
resto	0
i	1
j	1
	...

Notificación de errores

En los subprogramas se pueden detectar errores

Errores que impiden realizar los cálculos:

```
void cambio(double precio, double pago, int &euros, int &cent50,
           int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {
    → { if (pago < precio) { // Cantidad insuficiente
        cout << "Error: El pago es inferior al precio" << endl;
    }
    ...
}
```

¿Debe el subprograma notificar al usuario o al programa?

→ Mejor notificarlo al punto de llamada y allí decidir qué hacer

```
void cambio(double precio, double pago, int &euros, int &cent50,
           int &cent20, int &cent10, int &cent5, int &cent2, int &cent1,
           → bool &error) {
    if (pago < precio) { // Cantidad insuficiente
        → error = true;
    }
    else {
        → error = false;
```

Notificación de errores

cambio.cpp

Al volver de la llamada se decide qué hacer si ha habido error...

- ✓ ¿Informar al usuario?
- ✓ ¿Volver a pedir los datos?
- ✓ Etcétera

```
int main() {
    double precio, pago;
    int euros, cent50, cent20, cent10, cent5, cent2, cent1;
    → bool error;
    cout << "Precio: ";
    cin >> precio;
    cout << "Pago: ";
    cin >> pago;
    cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,
           cent1, error);
    → if (error) {
        cout << "Error: El pago es inferior al precio" << endl;
    }
    else {
        ...
    }
}
```

Resultado de la función

Resultado de la función

Una función ha de devolver un resultado

La función ha de terminar su ejecución devolviendo el resultado

La instrucción `return`:

- Devuelve el dato que se indica a continuación como resultado
- Termina la ejecución de la función

El dato devuelto sustituye a la llamada de la función en la expresión

```
int cuad(int x) { ←  
    return x * x; ←  
    x = x * x; ←  
}  
                                int main() {  
                                cout << 2 * cuad(16);  
                                return 0;  
}
```

Esta instrucción
no se ejecutará nunca

256

```
graph TD; main["int main() { cout << 2 * cuad(16); return 0;}"] --> cuad["int cuad(int x) { return x * x; }"]; cuad --> assign["x = x * x;"]; assign --> note["Esta instrucción no se ejecutará nunca"];
```

Un único punto de salida

```
int compara(int val1, int val2) {  
    // -1 si val1 < val2, 0 si iguales, +1 si val1 > val2  
    if (val1 == val2) {  
        return 0;      —————→  
    }  
    else if (val1 < val2) {  
        return -1;    —————→  ¡3 puntos de salida!  
    }  
    else {  
        return 1;  
    }  
}
```



Un único punto de salida

```
int compara(int val1, int val2) {  
    // -1 si val1 < val2, 0 si iguales, +1 si val1 > val2  
    int resultado;  
  
    if (val1 == val2) {  
        resultado = 0;  
    }  
    else if (val1 < val2) {  
        resultado = -1;  
    }  
    else {  
        resultado = 1;  
    }  
  
    return resultado; → Punto de salida único  
}
```



¿Cuándo termina el subprograma?

Procedimientos (tipo void):

- Al encontrar la llave de cierre que termina el subprograma o
- Al encontrar una instrucción return (sin resultado)

Funciones (tipo distinto de void):

- SÓLO al encontrar una instrucción return (con resultado)

Nuestros subprogramas siempre terminarán al final:

- ✓ No usaremos return en los procedimientos
- ✓ Funciones: sólo un return y estará al final



Para facilitar la depuración y el mantenimiento,
codifica los subprogramas con un único punto de salida

Prototipos

¿Qué subprogramas hay en el programa?

¿Dónde los ponemos? ¿Antes de `main()`? ¿Después de `main()`?
→ Los pondremos después de `main()`

¿Son correctas las llamadas a subprogramas?

En `main()` o en otros subprogramas

- ¿Existe el subprograma?
- ¿Concuerdan los argumentos con los parámetros?

Deben estar los prototipos de los subprogramas antes de `main()`

Prototipo: cabecera del subprograma terminada en ;

```
void dibujarCirculo();
void mostrarM();
void proc(double &a);
int cuad(int x);
...
```



main() es el único subprograma que no hay que prototipar

Ejemplos

intercambia.cpp

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

void intercambia(double &valor1, double &valor2); // Prototipo

int main() {
    double num1, num2;
    cout << "Valor 1: ";
    cin >> num1;
    cout << "Valor 2: ";
    cin >> num2;
    intercambia(num1, num2);
    cout << "Ahora el valor 1 es " << num1
        << " y el valor 2 es " << num2 << endl;
    return 0;
}

void intercambia(double &valor1, double &valor2) {
    double tmp; // Variable local (temporal)
    tmp = valor1;
    valor1 = valor2;
    valor2 = tmp;
}
```



Asegúrate de que los prototipos coincidan con las implementaciones