

(Versión de fecha 23 de octubre de 2024)

1 RESUMEN TIPOS DE DATOS

TIPOS DATOS

- int → 4 o 2 bytes

<stdint.h> → int8_t
int16_t
int32_t
int64_t
↓
%u

printf → %d

scanf → %d

- double — 8 bytes

printf %f %lf

scanf %lf

long float

precis ≈ 15 decim

```
#include <stdint.h>
```

```
int n;
```

```
int32_t n;
```

- char 'a' 1 byte

printf %c

scanf %c

```
int n;  
scanf("%d", &n);
```

24 \n

→ n = 24

\n

```
int m;  
scanf("%d", &m);
```

\n 32 \n

→ m = 32

\n

```
char p;
```

```
scanf("%c", &p);
```

\n

→ TRUENO

" %c "

↑
ESPACIO

Operadores relacionales numero \otimes numero $\rightarrow \{0,1\}$

`<stdbool.h>` \rightarrow bool $\rightarrow \begin{cases} \text{true} \\ \text{false} \end{cases}$

`if (x < y == false) { ... }`

`while (n < 10 == true) { ... }`

Si bloque de código solo una línea, puede omitir las llaves.

```
if (x == 3)
    printf("Hola");
else
    printf("Adios");
```

```
if (x == 3) printf("—");
else printf("—");
```

OPERADORES LÓGICOS

logical \otimes logical \rightarrow logical

AND `&&` solo es cierto si los dos operandos son ciertos

OR `||` si alguno de los operandos es cierto, es cierto.

NOT `!`

`(x < 5) && (y > 8)`

AND

<code>&&</code>	1	0
1	1	0
0	0	0

OR

1	1	0
1	0	1
0	1	1

NOT

!	
1	0
0	1

Nota: `\n` residuales en el búfer del teclado

Para comprender mejor el tema de los `\n` residuales en el búfer del teclado, echa un vistazo al siguiente vídeo:

<https://www.youtube.com/watch?v=qwv9JfnvPAI&t=12m46s>

2 PRIMER MÚLTIPLO DE 3 EN UN ARRAY (PRÁCTICA 2, FASE 2.1)

Ejemplo 1 Primer múltiplo de 3 del array

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

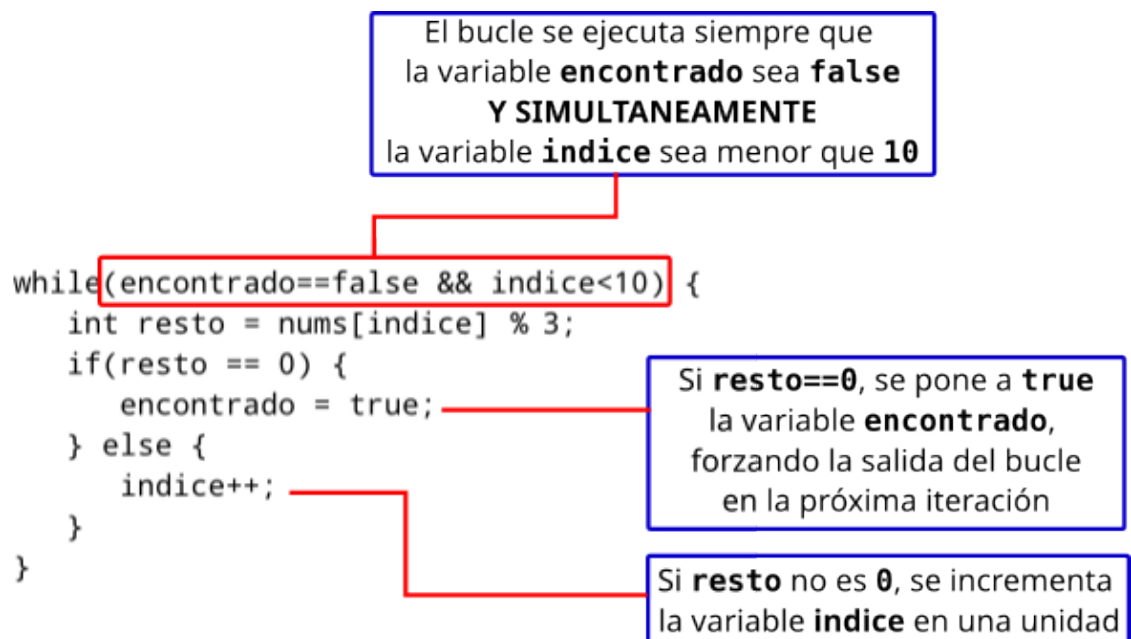
int main() {

    //int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9}; // Array con múltiplos de 3
    int nums[] = {1, 2, 2, 4, 5, 5, 7, 8, 8}; // Array sin múltiplos de 3

    bool encontrado = false;
    int indice = 0;

    while(encontrado==false && indice<10) {
        int resto = nums[indice] % 3;
        if(resto == 0) {
            encontrado = true;
        } else {
            indice++;
        }
    }

    if(encontrado == true) {
        printf("Primer múltiplo: nums[%d] = %d \n", indice, nums[indice]);
    } else {
        printf("No se encontraron múltiplos de 3\n");
    }
}
```



3 ESCRIBIR UN ARRAY EN ORDEN INVERSO (PRÁCTICA 2, FASE 2.2)

Se va a resolver con un bucle *for* y con un bucle *while*. En ambos casos, el procedimiento es empezar por escribir el último elemento e ir disminuyendo el índice hasta que valga 0.

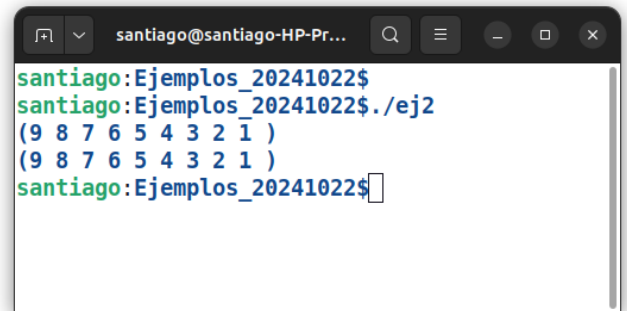
Ejemplo 2 Imprimir array invertido

```
#include <stdio.h>

int main() {
    // Imprime array en orden inverso

    int nums[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    // Solución con bucle for
    printf("(");
    for(int i=9; i>0; i--) {
        printf("%d ", nums[i]);
    }
    printf("\n");

    // Solución con bucle while
    printf("(");
    int indice = 9;
    while(indice>0) {
        printf("%d ", nums[indice]);
        indice--;
    }
    printf("\n");
}
```



4 ALGORITMOS DEL MÁXIMO Y DEL MÍNIMO (PRÁCTICA 2, FASE 2.3)

El algoritmo del máximo es un algoritmo clásico que conviene memorizar. Consiste en determinar el valor máximo de una colección de valores. Para ello, se crea una variable que guardará el valor máximo a la que se asigna inicialmente el valor del primer elemento de la colección. A continuación, se recorren todos los elementos de la colección y, en cada uno de ellos, se comprueba si es mayor que el valor del máximo guardado. Si es mayor, el valor guardado se sustituye por el nuevo máximo.

Cuando se hayan recorrido todos los elementos de la colección, la variable máximo guardará el valor del máximo de la colección. El siguiente ejemplo calcula el máximo de entre los valores de un array de 10 elementos.

Ejemplo 3 Algoritmos del máximo y del mínimo

```
#include <stdio.h>

int main() {
    int nums[] = {0, -2, 3, 5, 8, -3, 2, 2, 9, -1};

    int maximo = nums[0];
    for(int i=1; i<10; i++) {
        if(nums[i]>maximo) {
            maximo = nums[i];
        }
    }
    printf("máximo = %d \n", maximo);
}
```

El algoritmo del mínimo se resuelve de manera similar, sustituyendo la comparación *ser mayor* por una comparación *ser menor*. También puede suceder que inicialmente no conozcamos el array ni el número de elementos. El siguiente ejemplo pide al usuario una serie de números enteros. Cuando el usuario teclee 0, se dejan de leer

elementos y se muestra en pantalla el mínimo de los números tecleados por el usuario. El último 0 tecleado por el usuario no interviene en la determinación del mínimo.

Ejemplo 4 Algoritmo del mínimo

```
#include <stdio.h>

int main() {
    int num, minimo;
    printf("Teclee un número (0 para finalizar): ");
    scanf("%d", &num);
    minimo = num;

    while(num != 0) {
        printf("Teclee un número (0 para finalizar): ");
        scanf("%d", &num);
        if(num < minimo) {
            minimo = num;
        }
    }

    if(minimo != 0) {
        printf("Mínimo = %d \n", minimo);
    } else {
        printf("No se tecleó ningún número\n");
    }
}
```

Observa que hay una repetición de código, al tener que hacer fuera del bucle la lectura del primer número y así poderlo usar como mínimo inicial. Una alternativa sería utilizar como valor mínimo inicial el valor del mínimo entero que proporciona la librería *limits.h*, llamado *INT_MIN*, como se hace en la siguiente versión alternativa del programa anterior:

Ejemplo 5 Utilización de la constante *INT_MIN*

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

int main() {
    int num=1;
    int minimo = INT_MIN;

    while(num != 0) {
        printf("Teclee unhttps://es.wikipedia.org/wiki/Stdlib.h número (0 para finalizar): ");
        scanf("%d", &num);
        if(num < minimo) {
            minimo = num;
        }
    }

    if(minimo != 0 && minimo != INT_MIN) {
        printf("Mínimo = %d \n", minimo);
    } else {
        printf("No se tecleó ningún número\n");
    }
}
```

En la práctica 2 se pide el máximo de las distancias entre cada dos elementos sucesivos del array. El problema es similar a los anteriores, con la precaución de recorrer de manera correcta el array. Hay que tener en cuenta que en un array de 10 elementos solo hay 9 distancias. También hay que calcular de manera correcta la distancia entre dos números enteros, que es el valor absoluto de la diferencia entre ambos, según la siguiente fórmula:

$$dist = abs(x_i - x_{i-1});$$

En la fórmula anterior se calcula la distancia del punto i al punto anterior, el $i-1$. La función $abs()$ se encuentra en la librería `stdlib.h`. La solución podría ser similar a la siguiente:

Ejemplo 6 Máxima distancia entre dos elementos consecutivos

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

int main() {

    int nums[] = {-1, 2, 3, 7, -2, 4, 6, 8, 12, 3};

    int max_dist = 0;
    int dist;
    for(int i=1; i<10; i++) {
        dist = abs(nums[i] - nums[i-1]);
        if(dist>max_dist) {
            max_dist = dist;
        }
    }
    printf("Máxima distancia: = %d \n", max_dist);
}
```

Observa los valores inicial y final utilizados en el bucle. Sería equivalente calcular la distancia de cada elemento con el siguiente, pero habría que cambiar la fórmula del cálculo de la distancia y los valores inicial y final del bucle:

Ejemplo 7 Máxima distancia entre dos elementos consecutivos

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

int main() {

    int nums[] = {-1, 2, 3, 7, -2, 4, 6, 8, 12, 3};

    int max_dist = 0;
    int dist;
    for(int i=0; i<9; i++) {
        dist = abs(nums[i+1] - nums[i]);
        if(dist>max_dist) {
            max_dist = dist;
        }
    }
    printf("Máxima distancia: = %d \n", max_dist);
}
```

Observa que se comienza asignando el valor 0 a la distancia máxima. En este problema, al no existir distancias negativas, se puede usar este truco.

5 ALGORITMO DE LA SUMA (PRÁCTICA 2, FASE 2.4)

Para sumar los valores de los elementos de un array, se fija una variable para guardar la suma, cuyo valor inicial es cero y luego se recorre el array sumando cada elemento a dicha variable, como se hace en el siguiente ejemplo:

Ejemplo 8 Algoritmo de la suma

```
#include <stdio.h>

int main() {

    int nums[] = {-1, 2, 3, 7, -2, 4, 6, 8, 12, 3};

    int suma = 0;
    for(int i=0; i<10; i++) {
        suma = suma + nums[i];
    }

    printf("Suma: = %d \n", suma);
}
```

Para calcular el producto de los elementos de un array, el algoritmo es parecido pero el valor inicial de la variable que acumula los productos es 1 y los elementos se van multiplicando:

Ejemplo 9 Algoritmo del producto

```
#include <stdio.h>

int main() {

    int nums[] = {-1, 2, 3, 7, -2, 4, 6, 8, 12, 3};

    int producto = 1;
    for(int i=0; i<10; i++) {
        producto = producto * nums[i];
    }

    printf("Suma: = %d \n", producto);
}
```

En la Practica 2 lo que se pide es la media de los elementos del array. Se trata, por tanto, de calcular la suma de los elementos y dividirla entre el número de elementos, en este caso 10. Pero hay que hacer una consideración: los elementos del array y su suma son números enteros, pero la media es un número con decimales, un *double*. Si se hace directamente la división entre dos números enteros, el resultado será la división entera y se truncará la parte decimal.

Observa la siguiente instrucción:

```
double x = 10/4; // Asigna a x el valor 2.00
```

Como resultado de la instrucción anterior, la variable *x* valdrá 2.00. Estamos ante un operador de asignación: el compilador evalúa la expresión que aparece a la derecha del signo igual y el resultado se lo asigna a la variable que aparece a la izquierda del operador. En este caso, la expresión es la división entera 10/4, cuyo resultado es el número entero 2. El compilador convierte ese 2 a *double* y se lo asigna a *x*, resultando que *x* valdrá 2.00.

Para hacer la división como *double*, hay que convertir el numerador o el denominador en *double*. Esto se consigue haciendo que uno de ellos tenga el punto decimal, como se hace en la siguiente instrucción:

```
double x = 10.0/4; // Asigna a x el valor 2.50
```

Como resultado de esta instrucción, a *x* se le asignará el valor 2.5, que es el resultado que se buscaba. En la Práctica 2, se puede resolver el problema dividiendo la suma de los elementos del array por 10.0:

```
double media = suma/10.0;
```

Casting entre tipos

Y, ¿cómo podríamos resolver el asunto si tanto el numerador como el denominador son variables enteras y no podemos poner el punto decimal en ninguna de ellas? Imagina que el número de elementos del array estuviera en una variable entera llamada *num_elementos*:

```
double media = suma / num_elementos;
```

En una expresión como la anterior no podemos poner el punto decimal. Se utiliza un mecanismo llamado **casting**, que permite convertir entre distintos tipos de datos. Consiste en poner entre paréntesis y delante de la variable en cuestión el tipo de datos al que la queremos convertir:

```
double media = (double)suma / num_elementos;
```

El siguiente código muestra un ejemplo completo utilizando el casting:

Ejemplo 10 Casting

```
#include <stdio.h>

int main() {

    int suma = 10;
    int num_elementos = 4;

    double media = (double)suma / num_elementos;

    printf("%.2f \n", media); // Imprime 2.50

}
```