

Aritmética de punteros



¿Qué es?

Antes de ver que es la aritmética de punteros, repasemos algunos conceptos que vamos a necesitar usar, como por ejemplo, como se ven las variables en memoria.



Como veníamos viendo la memoria

```
int nota_parcial = 7;    &nota_parcial = 0;  
int nota_final = 8;     &nota_final = 1;
```

0	1	2	3
7	8	?	?



Como es en realidad

```
int nota_parcial = 7;    &nota_parcial = 0;  
int nota_final = 8;     &nota_final = 4;
```



0 1 2 3 4 5 6 7 8

00000000	00000000	00000000	00000111	00000000	00000000	00000000	00001000	?
----------	----------	----------	----------	----------	----------	----------	----------	---

Y los vectores?

```
int vector[2];  
vector[0] = 7;  
vector[1] = 8;
```

```
vector = 0;  
vector[0] = 0;  
vector[1] = 4;
```



0

1

2

3

4

5

6

7

00000000	00000000	00000000	00000111	00000000	00000000	00000000	00001000
----------	----------	----------	----------	----------	----------	----------	----------

¿Por qué cada entero ocupa 4 cuadritos?

Los números en memoria están en binario (es decir, se escriben solo con 1 y 0), para poder representar muchos números necesito juntar varios bytes.

Si solo usara un byte, solo podría representar 256 números (mitad negativos y mitad positivos). Este número sale de hacer 2^8 .



Algunos tamaños y valores posibles

Tipo de dato	Tamaño	Rango de valores (min, max)
char	1 byte	(0, 255)
short	2 bytes	(-32,768, 32,767)
unsigned short	2 bytes	(0, 65,535)
int	4 bytes	(-2,147,483,648, 2,147,483,647)
unsigned int	4 bytes	(0, 4,294,967,295)
long	8 bytes	(-2 ⁶³ , (2 ⁶³) - 1)
unsigned long	8 bytes	(0, (2 ⁶⁴) - 1)

Operador sizeof()

Por suerte, C no espera que sepamos cuánto ocupa cada tipo de dato, y nos proporciona una herramienta para que podamos consultarlo cuando necesitamos. Supongamos que tenemos

```
int numero;
```

```
size_t          tamaño          =          sizeof(numero);
```

La variable tamaño va a almacenar el valor **4** (sizeof() devuelve los **bytes** que ocupa cada tipo de dato en memoria)



Ahora sí, aritmética y punteros

La aritmética es una de las ramas más antiguas (y simples) de las matemáticas, estudia las operaciones básicas entre los números (como suma y resta).

Por otro lado, los punteros son un tipo de dato que almacena una dirección de memoria.



Aritmética de punteros

Teniendo un puntero, puedo aplicarle las operaciones:

- Suma (++)
- Resta (--)
- Referenciación (&)
- Desreferenciación (*)



Referenciación (&)

Permite acceder a la dirección de memoria de una variable.

Ejemplo:

```
int nota = 10;
```

```
int* ptr_nota = &nota;
```

```
printf("El puntero 'ptr_nota' apunta a %p.\n", ptr_nota);
```

```
// Imprimirá algo como: "El puntero 'nota' apunta a 0x7fff3ae6128c."
```

Desreferenciación (*)

Permite ver el VALOR que está en la dirección asignada.

Ejemplo:

```
int nota = 10;
```

```
int* ptr_nota = &nota;
```

```
printf("El valor a lo que apunta 'ptr_nota' es %i.\n", *ptr_nota);
```

```
// Imprimirá: "El valor a lo que apunta 'ptr_nota' es 10."
```

Suma (++)

Incrementa en una unidad el valor de la dirección de memoria, siendo la unidad la cantidad de bytes del tipo de dato del puntero.

Ejemplo:

```
char* inicial = 5;
```

```
inicial++;
```

```
printf("El puntero 'nota' apunta a %lu.\n", (unsigned long) nota);
```

```
// Imprimirá: "El puntero 'nota' apunta a %lu 6."
```

Suma (++)

Otro ejemplo:

```
int* nota = 5;
```

```
nota++;
```

```
nota++;
```

```
printf("El puntero 'nota' apunta a %lu.\n", (unsigned long) nota);
```

// Imprimirá: "El puntero 'nota' apunta a %lu **13**."

Resta (--)

Decrementa en una unidad el valor de la dirección de memoria, siendo la unidad la cantidad de bytes del tipo de dato del puntero.

Ejemplo:

```
char* inicial = 5;
```

```
nota--;
```

```
printf("El puntero 'nota' apunta a %lu.\n", (unsigned long) nota);
```

```
// Imprimirá: "El puntero 'nota' apunta a %lu 4."
```

Resta (--)

Otro ejemplo:

```
int* nota = 15;
```

```
nota--;
```

```
nota--;
```

```
printf("El puntero 'nota' apunta a %lu.\n", (unsigned long) nota);
```

```
// Imprimirá: "El puntero 'nota' apunta a %lu 7."
```


Y esto, ¿Para qué me sirve?

Si tuviéramos muchas variables conjuntas en memoria, y tuviéramos el puntero a la primera de esas variables, podríamos ir accediendo a cada una de las variables haciendo únicamente aritmética de punteros.

Y esto es lo mismo que recorrer un **vector**. Cada vez que accedemos a un elemento de un vector estamos haciendo aritmética de punteros.



Operador []

Supongamos que tenemos

```
int vector[2]
```

Y queremos acceder a `vector[0]` sin usar `[]`, como podemos hacer?

Hacemos ***vector**

Operador []

Y si queremos acceder a `vector[1]`?

Podemos hacer

`* (vector + 1)`

El operador `[]` no hace otra cosa más que aritmética de punteros



Ejemplos

