

Intro a C

Tipos

With ❤ by @vichoeq & @KnowYourselfes

Bases Numéricas #1

Base Decimal

Los números que usamos normalmente son en base 10

The diagram illustrates the expansion of the decimal number 293706 into its place value components. On the left, the number 293706 is shown with each digit in a different color. Lines connect each digit to its corresponding place value multiplier: 2 to 100000, 9 to 10000, 3 to 1000, 7 to 100, 0 to 10, and 6 to 1. These are listed as 'x 1', 'x 10', 'x 100', 'x 1000', 'x 10000', and 'x 100000'. Arrows point from these multipliers to the right, where the expanded form is shown as a sum: 200000 + 90000 + 3000 + 700 + 00 + 6. The final sum, 293706, is shown below a horizontal line, with each digit color-coded to match the original number.

$$\begin{array}{r} 293706 \\ \begin{array}{l} \text{---} \times 1 \\ \text{---} \times 10 \\ \text{---} \times 100 \\ \text{---} \times 1000 \\ \text{---} \times 10000 \\ \text{---} \times 100000 \end{array} \\ \begin{array}{r} 6 \\ 00 \\ 700 \\ 3000 \\ 90000 \\ 200000 \end{array} + \\ \hline 293706 \end{array}$$

Base Decimal

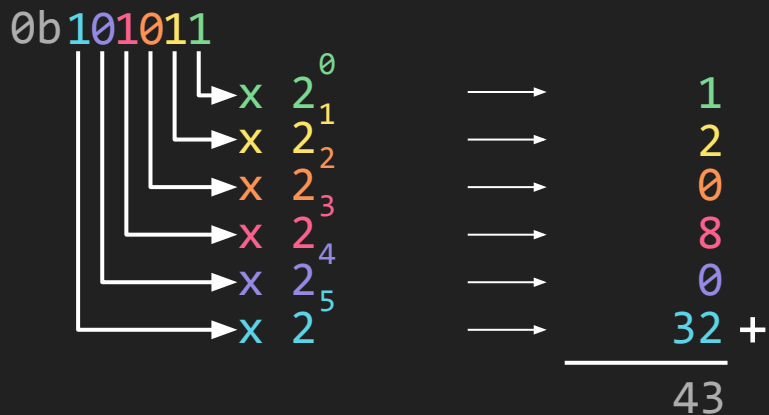
Los números que usamos normalmente son en base 10

The diagram illustrates the expansion of the decimal number 293706 into its place value components. On the left, the number 293706 is shown with each digit in a different color. Lines connect each digit to its corresponding power of 10: 2 to 10^5 , 9 to 10^4 , 3 to 10^3 , 7 to 10^2 , 0 to 10^1 , and 6 to 10^0 . Each connection is labeled with a multiplication sign 'x'. On the right, the expanded form is shown as a sum of these products, with each term aligned to the right. The terms are: 200000 (cyan), 90000 (purple), 3000 (pink), 700 (orange), 00 (yellow), and 6 (green). A horizontal line is drawn under the sum, and the final result, 293706, is shown below it in the same color-coded digits.

$$\begin{array}{r} 200000 + 90000 + 3000 + 700 + 00 + 6 \\ \hline 293706 \end{array}$$

Base Binaria

Un dígito en base 2 se conoce como bit



En un PC la información está guardada en grupos de 8 bits = 1 byte

Variables

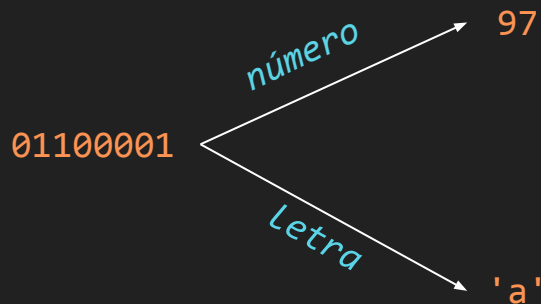
Variables en un programa

Cada **variable** tiene:

- Una **dirección** de memoria
- Un **tipo**
- Un **valor**

El **tipo** indica dos cosas:

- Tamaño en memoria
- Interpretación



tipos

printf



```
printf("%i %i %f\n", 5, 7, 2.8);  
  
printf("Hello %s\n", "World!");
```

```
$ gcc main.c -o main  
$ ./main  
5, 7, 2.800000  
Hello World!
```



```
print(f"{5} {7} {2.8}")  
  
print(f"Hola {'World!'}")
```

```
$ python3 main.py  
5, 7, 2.8  
Hello World!
```

printf - Salto de línea



```
printf("%i %i %f", 5, 7, 2.8);  
  
printf("Hello %s", "World!");
```

```
$ gcc main.c -o main  
$ ./main  
5, 7, 2.800000Hello World!
```



A diferencia de `print`, `printf` no agrega un salto de línea automáticamente, por lo cual hay que agregarlo explícitamente.

int - Número entero



```
int a = 1287;
printf("a = %i\n", a);
printf("size: %zu\n", sizeof(int));
```

```
$ gcc main.c -o main
$ ./main
a = 1287
size: 4
```

Tamaño: 32 bits (4 bytes)

Admite números positivos y negativos

Para imprimir: `%i`, `%d`

float - Número decimal



```
float b = 2.72181;
printf("b = %f\n", b);
printf("size: %zu\n", sizeof(float));
```

```
$ gcc main.c -o main
$ ./main
b = 2.721810
size: 4
```

Tamaño: 32 bits (4 bytes)

Admite números positivos y negativos

Para imprimir: %f

double - Número decimal de doble precisión



```
double c = 3.141592;  
printf("c = %lf\n", c);  
printf("size: %zu\n", sizeof(double));
```

```
$ gcc main.c -o main  
$ ./main  
c = 3.141592  
size: 8
```

Tamaño: 64 bits (8 bytes)

Tiene más precisión que un *float*

Para imprimir: *%lf*

char - Character / Letra



```
char d = 'h';  
printf("d = %c\n", d);  
printf("size: %zu\n", sizeof(char));
```

```
$ gcc main.c -o main  
$ ./main  
d = h  
size: 1
```

Tamaño: 8 bits (1 byte)

Rodeado de comillas simples

Para imprimir: %c

char - ☢ **WARNING! ACHTUNG! PELIGRO!** ☢



```
char d = 'h';  
printf("d = %c\n", d);
```

```
$ gcc main.c -o main  
$ ./main  
d = h
```



```
char d = "h";  
printf("d = %c\n", d);
```

```
$ gcc main.c -o main  
Warning: initialization from pointer  
$ ./main  
d = ✦
```

Modificar *tipos*

casting - Interpretar un *tipo* como otro



```
double e = 64.7901;
char f = e; // Casting implícito
char g = (char) e; // Casting explícito
printf("f = %c, g = %c\n", f, g);
printf("f = %i, g = %i\n", f, g);
```

```
$ gcc main.c -o main
$ ./main
f = @, g = @
f = 64, g = 64
```

El **casting implícito** es sólo al asignar o al pasarlo a una **función** con **tipo** compatible

El **casting explícito** fuerza el cast.

En este caso **char** se interpreta como un número ASCII.

casting - División decimal



```
int h = 5;  
int i = 2;  
double j = h / i;  
printf("j = %lf\n", j);
```

```
$ gcc main.c -o main  
$ ./main  
2.000000
```



```
int h = 5;  
int i = 2;  
double j = (double) h / i;  
printf("j = %lf\n", j);
```

```
$ gcc main.c -o main  
$ ./main  
2.500000
```

typedef - Alias de *tipos*



```
typedef int cool_int;
```

```
cool_int k = 42;
```

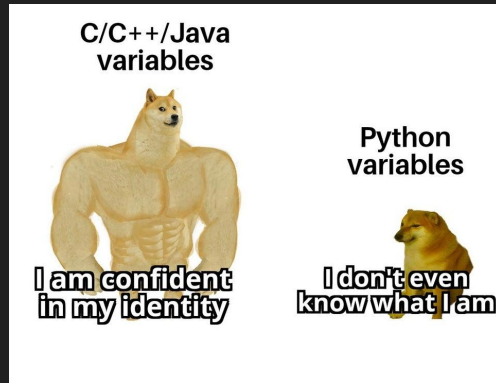
```
cool_int l = 1313;
```

```
typedef TIPO_EXISTENTE_DE_NOMBRE_LARGO ALIAS;
```

```
ALIAS k = valor;
```

Con **typedef** podemos crear un alias para el *tipo*. Va a ser muy útil para referirnos a *tipos* con nombres muy largos.

¡Muchas Gracias!



With ❤ by @vichoeq & @KnowYourselfes