Datos y tipos de datos

Dato

Representación formal de hechos, conceptos o instrucciones adecuada para su comunicación, interpretación y procesamiento por seres humanos o medios automáticos.

Tipo de dato

Especificación de un dominio (rango de valores) y de un conjunto válido de operaciones a los que normalmente los traductores asocian un esquema de representación interna propio.

Clasificación de los tipos de datos

En función de quién los define:

- Tipos de datos estándar
- Tipos de datos definidos por el usuario

En función de su representación interna:

- Tipos de datos escalares o simples
- Tipos de datos estructurados

Codificación de los datos en el ordenador

En el interior del ordenador, los datos se representan en binario.

El sistema binario sólo emplea dos símbolos: 0 y 1

- Un bit nos permite representar 2 símbolos diferentes: 0 y 1
- Dos bits nos permiten codificar 4 símbolos: 00, 01, 10 y 11
- Tres bits nos permiten codificar 8 símbolos distintos: 000, 001, 010, 011, 100, 101, 110 y 111

En general,

con N bits podemos codificar $\mathbf{2^N}$ valores diferentes

N	2^N
1	**2**
2	**4**
3	**8**
4	**16**
5	**32**
6	**64**
7	**128**
8	**256**
9	**512**
10	**1024**
11	**2048**
12	**4096**
13	**8192**
14	**16384**
15	**32768**
16	**65536**

Si queremos representar X valores diferentes, necesitaremos N bits, donde N es el menor entero mayor o igual que $\mathbf{\log_2 X}$

Representación posicional

Un número se representa mediante un conjunto de cifras, cuyo valor depende de la cifra en sí y de la posición que ocupa en el número

NÚMEROS ENTEROS

Ejemplo: Si utilizamos 32 bits para representar números enteros, disponemos de 2^{32} combinaciones diferentes de 0s y 1s:

4 294 967 296 valores.

Como tenemos que representar números negativos y el cero, el ordenador será capaz de representar

del –2 147 483 648 al +2 147 483 647.

Con 32 bits no podremos representar números más grandes.

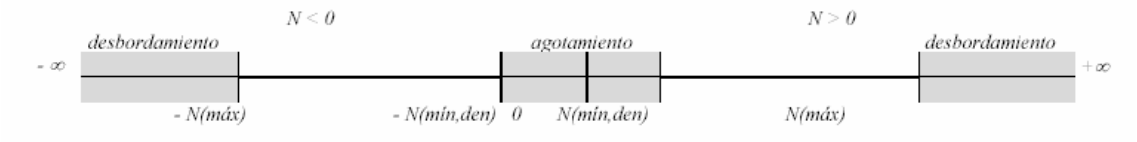
¡¡¡ 2 147 483 647 + 1 = –2 147 483 648 !!!

NÚMEROS REALES (en notación científica)

(+|–) mantisa x $2^{\text{exponente}}$

× El ordenador sólo puede representar un subconjunto de los números reales (números en coma flotante)

× Las operaciones aritméticas con números en coma flotante están sujetas a errores de redondeo.



Estándar IEEE 754

- Precisión sencilla
  (bit de signo + 8 bits exponente + 23 bits mantisa)
- Precisión doble
  (bit de signo + 11 bits exponente + 52 bits mantisa)

Representación de textos

Se escoge un conjunto de caracteres: alfabéticos, numéricos, especiales (separadores y signos de puntuación), gráficos y de control (por ejemplo, retorno de carro).

Se codifica ese conjunto de caracteres utilizando n bits.
Por tanto, se pueden representar hasta **2^n símbolos** distintos.

Ejemplos de códigos normalizados

ASCII (American Standard Code for Information Interchange)

- ANSI X3.4-1968, 7 bits (128 símbolos)
- ISO 8859-1 = Latin-1, 8 bits (256 símbolos)

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	96	`	a	b	c	d	e	f	g	h	i	j	k	L	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
80	128																
90	144																
A0	160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

UNICODE, ISO/IEC 10646, 16 bits (65536 símbolos)

Zona	*Códigos*		*Símbolos codificados*	*Nº de caracteres*
A	0000	0000 00FF	Latín-1	256
			otros alfabetos	7.936
		2000	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos	8.192
I	4000		Ideogramas	24.576
O	A000		Pendiente de asignación	16.384
R	E000 FFFF		Caracteres locales y propios de los usuarios. Compatibilidad con otros códigos	8.192

Tipos de datos primitivos en C

El lenguaje ANSI C define 6 tipos de datos básicos
que pueden ir acompañados de modificadores

Tipos de datos básicos

`char`	Caracteres
`int`	Números enteros
`float`	Números en coma flotante (32 bits)
`double`	Números en coma flotante de doble precisión (64 bits)
`void`	Tipo nulo
Punteros	Direcciones de memoria

Modificadores

Tamaño del dato

`short`	(`int` por defecto)
`long`	(`int` por defecto)

Signo

Aplicable a los tipos `char`, `short`, `int` y `long`

`signed`	(con signo)
`unsigned`	(sin signo)

Modo de almacenamiento

No se suele usar

`register`	
`auto`	(por defecto)
`static`	
`extern`	

Números enteros
`[unsigned] char, [unsigned] short, [unsigned] long`

6 tipos básicos para representar números enteros:

- 3 con signo: `char, short, long`
- 3 sin signo: `unsigned (char | short | long)`

Tipo de dato	Espacio en memoria	Valor Mínimo	Valor Máximo
`char`	8 bits	-128	127
`unsigned char`		0	255
`short`	16 bits	-32768	32767
`unsigned short`		0	65535
`long`	32 bits	-2147483648	2147483647
`unsigned long`		0	4294967295

`int`
El tipo de dato `int` es equivalente
- al tipo de dato `short` en compiladores de 16 bits
- al tipo de dato `long` en compiladores de 32 bits

`unsigned int`
El tipo de dato `unsigned int` es equivalente
- al tipo de dato `unsigned short` en compiladores de 16 bits
- al tipo de dato `unsigned long` en compiladores de 32 bits

Literales enteros

Los literales enteros pueden expresarse:

- En decimal (base 10):

  `255`

- En octal (base 8), con el prefijo `0`:

  `0377` ($3 \cdot 8^2 + 7 \cdot 8^1 + 7 = 255$)

- En hexadecimal (base 16), con el prefijo `0x`:

  `0xff` ($15 * 16^1 + 15 = 255$)

- Los literales enteros son de tipo `int` por defecto.

- Un literal entero es de tipo `long` si va acompañado del sufijo `l` o `L`:

  `123456789L` es de tipo `long`

  NOTA: Se prefiere el uso de `L` porque `l` (L minúscula) puede confundirse con `1` (uno).

- Un literal entero `unsigned` debe ir acompañado del sufijo `u` o `U`:

  `123456789uL` es de tipo `unsigned long`

Definición
Literal: Especificación de un valor concreto de un tipo de dato.

Operaciones con números enteros

Desbordamiento

Si sobrepasamos el valor máximo que se puede representar con un tipo de dato entero, nadie nos avisa de ello: en la ejecución de nuestro programa obtendremos un resultado incorrecto.

Tipo	Operación	Resultado
`char`	127 + 1	-128
`short`	32767 + 1	-32768
`long`	2147483647 + 1	-2147483648

Para obtener el resultado correcto, hemos de tener en cuenta el rango de valores de cada tipo de dato, de tal forma que los resultados intermedios de un cálculo siempre puedan representarse correctamente:

Tipo	Operación	Resultado
`short`	10000 * 10000	-7936
`long`	10000L * 10000L	100000000

División por cero

Si dividimos un número entero por cero,
se produce un error en tiempo de ejecución:

```
Divide error
```

La ejecución del programa termina de forma brusca al intentar hacer la división por cero.

Números en coma flotante

`float, double, long double`

El estándar ANSI C no establece su representación física

Tipo de dato	Espacio en memoria	Mínimo (valor absoluto)	Máximo (valor absoluto)	Dígitos significativos
`float`	32 bits	1.2×10^{-38}	3.4×10^{38}	6
`double`	64 bits	2.2×10^{-308}	1.8×10^{308}	15
`long double`	80 bits	3.4×10^{-4932}	1.2×10^{4932}	18

Literales reales

- Cadenas de dígitos con un punto decimal

```
123.45     0.0     .001
```

- En notación científica (mantisa·$10^{\text{exponente}}$)

```
123e45   123E+45  1E-6
```

Por defecto, los literales reales representan valores de tipo `double`

Para representar un valor de tipo `float`, hemos de usar el sufijo `f` o `F`:

```
123.45F   0.0f    .001f
```

El sufijo `l` o `L` se usa para los literales de tipo `long double`:

```
123.45L   0.0L    .001L
```

Operaciones con números en coma flotante

Según el estándar IEEE 754, las operaciones aritméticas en coma flotante pueden dar como resultado valores especiales:

- Cuando el resultado de una operación está fuera de rango, se obtiene `+Inf` o `–Inf` (“infinito”).
- Cuando el resultado de una operación está indeterminado, se obtiene `NaN` (“Not a Number”)

El estándar IEEE 754 establece los siguientes resultados:

Operación	Resultado
1.0 / 0.0	`+Inf`
-1.0 / 0.0	`-Inf`
0.0 / 0.0	`NaN`

No obstante, en función del compilador que utilicemos, puede que nos encontremos con un error en tiempo de ejecución:

```
Floating point error: Domain.
Abnormal program termination
```

O bien:

```
Floating point error: Divide by 0.
Abnormal program termination
```

Precisión

Las operaciones en coma flotante no son exactas debido a la forma en que se representan los números reales en el ordenador

Operación	Resultado
1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1	`0.50000000000000001`
1.0 - 0.9	`0.09999999999999998`

Operadores aritméticos

C incluye cinco operadores para realizar operaciones aritméticas:

Operador	Operación
`+`	Suma
`-`	Resta o cambio de signo
`*`	Multiplicación
`/`	División
`%`	Módulo (resto de la división)

- Si los operandos son enteros, se realizan operaciones enteras.
- En cuanto uno de los operandos es de tipo `float`, `double`, o `long double`, la operación se realiza en coma flotante.
- No existe un operador de exponenciación: para calcular x^y hay que utilizar la función `pow(x,y)` que se encuentra en `math.h`

División (/)

Operación	Tipo	Resultado
7 / 3	`int`	`2`
7 / 3.0f	`float`	`2.333333333f`
5.0 / 2	`double`	`2.5`
7.0 / 0.0	`double`	`+Inf`
0.0 / 0.0	`double`	`NaN`

- Si se dividen enteros, el resultado es entero y el resto se pierde.
- Una división entera por cero produce un error.
- Una división por cero, en coma flotante, produce `±Inf` o `NaN`.

Módulo (%): Resto de dividir números enteros
(no puede usarse con números en coma flotante)

Operación	Tipo	Resultado
7 % 3	`int`	`1`

Expresiones aritméticas
Se pueden combinar literales y operadores
para formar expresiones complejas.

Ejemplo

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9(\frac{4}{x} + \frac{9+x}{y})$$

En C se escribiría así:

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

- Las expresiones aritméticas se evalúan **de izquierda a derecha**.

- Los operadores aritméticos mantienen el orden de **precedencia** habitual (multiplicaciones y divisiones antes que sumas y restas).

- Para especificar el orden de evaluación deseado, se utilizan paréntesis.

> NOTA: Es recomendable utilizar paréntesis para eliminar interpretaciones erróneas y posibles ambigüedades

Definición
Expresión: Construcción que se evalúa para devolver un valor.

Caracteres
`char, unsigned char`

Tipo de dato	Espacio en memoria	Codificación
`char`	8 bits	ASCII

Literales de tipo carácter

Valores entre comillas simples
`'a' 'b' 'c' …  '1' '2' '3' …  '*' …`

Códigos ASCII (en hexadecimal): `\x??`
`'\x0a'` (avance de línea)
`'\x0d'` (retorno de carro)

Secuencias de escape para representar caracteres especiales:

Secuencia de escape	Descripción
\t	Tabulador (tab)
\n	Avance de línea (new line)
\r	Retorno de carro (carriage return)
\b	Retroceso (backspace)
\f	Salto de página (form feed)
\a	Sonido de alerta
\’	Comillas simples (apóstrofe)
\”	Comillas dobles
\\	Barra invertida
\?	Signo de interrogación
\0	Carácter nulo (NULL)

La biblioteca `ctype.h` define funciones básicas para trabajar con caracteres:

```
isalpha(), isdigit(), islower(), isupper()
           tolower(), toupper()
```

Cadenas de caracteres en C

En ANSI C no existen las cadenas de caracteres
como tipo predefinido:
una cadena de caracteres no es más que un vector de caracteres

Literales

Texto entra comillas dobles “ ”

```
“Esto es una cadena”
“‘Esto’ también es una cadena”
```

Las secuencias de escape son necesarias para introducir determinados caracteres dentro de una cadena:

```
“\”Esto es una cadena entre comillas\””
```

Formación de cadenas de caracteres

Para construir cadenas de caracteres en las que mostrar datos, se utilizan plantillas que se sustituirán por una representación adecuada de los valores del tipo indicado:

Plantilla	Tipo de dato
%c	`char`
%s	Cadena de caracteres
%d	`int` (en decimal)
%o	`int` (en octal)
%x	`int` (en hexadecimal)
%ld	`long`
%f	`float`
%lf	`double`
%Lf	`long double`

Datos de tipo booleano

En C no existe explícitamente un tipo de dato booleano para representar algo que pueda ser verdadero (V) o falso (F).

- Cualquier valor entero distinto de 0 se considera verdadero.

- Por convención,
  - Se usa el valor 1 para representar algo verdadero.
  - Se usa el valor 0 para representar algo falso.

Expresiones de tipo booleano

- Se construyen a partir de expresiones de tipo numérico con **operadores relacionales**.

- Se construyen a partir de otras expresiones booleanas (que en C son expresiones de tipo entero) con **operadores lógicos**.

Operadores relacionales

- Operadores de comparación válidos para números y caracteres
- Generan un resultado de tipo `int` que interpretamos como booleano

Operador	Significado
==	Igual
!=	Distinto
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

Operadores lógicos/booleanos

- Operandos booleanos.
- Tienen menos precedencia que los operadores de comparación.

Operador	Nombre	Significado
!	NOT	Negación lógica
&&	AND	'y' lógico
||	OR	'o' inclusivo
^	XOR	'o' exclusivo

Tablas de verdad

X	!X
0	1
1	0

A	B	A&&B	A||B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- NOT (!) cambia el valor booleano.
- AND (&&) devuelve verdadero si los dos son operandos son verdaderos. No evalúa el segundo operando si el primero es falso
- OR (||) devuelve falso si los dos operandos son falsos.
  No evalúa el segundo operando si el primero es verdadero
- XOR (^) devuelve verdadero si los dos operandos son diferentes.
  Con operandos booleanos es equivalente a `!=`

Ejemplos

Número x entre 0 y 10 `(0 <= x) && (x <= 10)`

Número x fuera del intervalo [0,10] `!((0 <= x) && (x <= 10))`

o bien `(0 > x) || (x > 10)`

Extra: Operadores a nivel de bits

- Se pueden utilizar a nivel de bits con números enteros.
- No se pueden usar con datos de otro tipo (p.ej. reales).

Los operadores NOT (~), AND (&), OR(|) y XOR (^)

- NOT (~) realiza el complemento a 1 de un número entero: Cambia los 0s por 1s y viceversa

- AND(&), OR(|) y XOR(^) funcionan a nivel de bits como los operadores booleanos AND (&&), OR(||) y XOR (^), respectivamente.

Operación	A nivel de bits	Resultado
~10	`~0…00001010 (1…11110101)`	`-11`
10 & 1	`0…00001010 & 0…0001 (0…00000000)`	`0`
10 & 2	`0…00001010 & 0…0010 (0…00000010)`	`2`
10 & 3	`0…00001010 & 0…0011 (0…00000010)`	`2`
10 | 1	`0…00001010 | 0…0001 (0…00001011)`	`11`
10 | 2	`0…00001010 | 0…0010 (0…00001010)`	`10`
10 | 3	`0…00001010 | 0…0011 (0…00001011)`	`11`
10 ^ 1	`0…00001010 ^ 0…0001 (0…00001011)`	`11`
10 ^ 2	`0…00001010 ^ 0…0010 (0…00001000)`	`8`
10 ^ 3	`0…00001010 ^ 0…0011 (0…00001001)`	`9`

Los operadores de desplazamiento <<, >> y >>>

- El operador de desplazamiento a la izquierda (<<) desplaza los bits del primer operando tantas posiciones a la izquierda como indica el segundo operando. Los nuevos bits se rellenan con ceros.

- El operador de desplazamiento a la derecha (>>) desplaza los bits del primer operando tantas posiciones a la derecha como indica el segundo operando. Los nuevos bits se rellenan con unos (si el primer operando es negativo) y con ceros (si es positivo).

Operación	A nivel de bits	Resultado
10 << 1	`00001010 << 1` (`0010100`)	`20` (`==10*2`)
7 << 3	`00000111 << 3` (`00111000`)	`56` (`==7*2`^{3}`)`
10 >> 1	`00001010 >> 1` (`00000101`)	`5` (`==10/2`)
27 >> 3	`00011011 >> 3` (`0000011`)	`3` (`==27/2`^{3}`)`
-50 >> 2	`11001110 >> 2` (`11110011`)	`-13` (`!=-50/2`^{2}`)`
-50 << 2	`1…11001110 << 2` (`1…1100111000`)	`-200` (`==-50*2`^{2}`)`

`x<<b` es equivalente a
multiplicar por 2^b

`x>>b` es equivalente a
realizar una división entera entre 2^b cuando `x` es positivo