Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 14: Representación de la información

Gracias al Prof. David González Márquez
por compartir materiales para esta clase.

Datos

La computadora almacena y opera con información binaria (o digital).

El medio físico permite almacenar bits: ceros (0) y unos (1) representados (por ejemplo) mediante diferencias de tensión.

Entonces, la computadora trabaja siempre con secuencias de bits que se interpretan de distintas formas: como enteros, como caracteres, etc.

Veremos cómo representar los tipos usando secuencias de bits de cierta longitud.

Ejemplo con 4 bits:

Binario (base 2)	Decimal (base 10)
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Conversión de binario a decimal

Queremos convertir el número binario **11001.00101** a notación decimal.

$$\begin{array}{ccccccccccc} 1 & 1 & 0 & 0 & 1 & . & 0 & 0 & 1 & 0 & 1 \\ \times & \times & \times & \times & \times & & \times & \times & \times & \times & \times \\ 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \\ \hline 2^4 & 2^3 & \cancel{2^2} & \cancel{2^1} & 2^0 & & \cancel{2^{-1}} & \cancel{2^{-2}} & 2^{-3} & \cancel{2^{-4}} & 2^{-5} \end{array}$$

$$2^4 + 2^3 + 2^0 + 2^{-3} + 2^{-5} = 16 + 8 + 1 + \frac{1}{8} + \frac{1}{32} = 25.15625$$

Luego, la representación decimal de `11001.00101` es 25.15625

Conversor online, útil para revisar nuestras cuentas:

`https://www.rapidtables.com/convert/number/decimal-to-binary.html`

Conversión de decimal a binario

Queremos representar el número decimal **25.79** en notación binaria.

Parte entera: 25

```
25 // 2 = 12 | 25 % 2 = 1
12 // 2 =  6 | 12 % 2 = 0
 6 // 2 =  3 |  6 % 2 = 0
 3 // 2 =  1 |  3 % 2 = 1
 1 // 2 =  0 |  1 % 2 = 1
```

Luego, 25 → `11001`

Parte fraccionaria: 0.79

```
0.79 * 2 = 1.58
0.58 * 2 = 1.16
0.16 * 2 = 0.32
0.32 * 2 = 0.64
0.64 * 2 = 1.28
0.28 * 2 = 0.56
0.56 * 2 = 1.12
...
```

Luego, 0.79 → `1100101`...

Luego, la representación binaria de 25.79 es `11001.1100101`...

En el **??** de la Guía 6 se describen
los algoritmos para hacer estas conversiones.

Números enteros

Los enteros (`int`) para una computadora son similares a los enteros matemáticos (el conjunto $\mathbb{Z}$):

$$\ldots, -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4, \ldots$$

Pero en las computadoras están acotados por encima y por debajo:

$$\mathrm{MIN}, \ldots, -4,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4, \ldots, \mathrm{MAX}$$

donde MIN y MAX dependen de la cantidad de bits usados para representar un entero (ej.: 8 bits, 16 bits, etc.) y de la notación elegida.

Veamos algunas formas posibles de representar enteros mediante secuencias de bits:

- Notación Sin Signo
- Notación Signo y Magnitud
- Notación Complemento a 2
- Notación Exceso-e

Números enteros

Notación Sin Signo (n bits)

- Usa n bits para el valor.
- Rango representable 0 a $2^n - 1$.
- No hay números negativos.

Conversión X (Decimal) $\rightarrow$ Y (Sin Signo):
Y = `base2(X)`

Conversión Y (Sin Signo) $\rightarrow$ X (Decimal):
X = `base10(Y)`

Ejemplo con 4 bits:

Binario	Sin Signo
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Números enteros

Notación Signo y Magnitud (n bits)

- Usa 1 bit para signo y $n-1$ bits para el valor.
- Rango representable $-(2^{n-1}-1)$ a $2^{n-1}-1$.
- Bit de signo: 0 = positivo, 1 = negativo.
- Los bits restantes determinan la magnitud.

Conversión X (Decimal) $\rightarrow$ Y (Signo-Magnitud):

Si X ≥ 0: Y = `concatenar(0, base2(X))`
Si X ≤ 0: Y = `concatenar(1, base2(abs(X)))`

Conversión Y (Signo-Magnitud) $\rightarrow$ X (Decimal):

Si Y_{n-1} == 0: X = base10($Y_{n-2} \ldots Y_0$)
Si Y_{n-1} == 1: X = −base10($Y_{n-2} \ldots Y_0$)

Observación: Por convención, Y_{n-1} refiere al bit más significativo (el primero de la izquierda).

Ejemplo con 4 bits:

Binario	Con Signo
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Números enteros

Notación Complemento a 2 (n bits)

- Usa n bits para el valor y signo.
- Rango representable $-(2^{n-1})$ a $2^{n-1}-1$.

Conversión X (Decimal) $\rightarrow$ Y (Complemento2):

Si X ≥ 0: Y = `base2(X)`
Si X < 0: Y = `BitwiseNot(base2(abs(X)))`$+1$

Conversión Y (Complemento2) $\rightarrow$ X (Decimal):

Si Y_{n-1} == 0: X = `base10(Y)`
Si Y_{n-1} == 1: X = $-$(`base10(BitwiseNot(Y)`$+1$))

Definición:
BitwiseNot = Invertir bit a bit
Ej: BitwiseNot(1011) = 0100

Ejemplo con 4 bits:

Binario	Compl.a 2
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Números enteros

Notación Exceso-e (n bits)

- Usa n bits, no considera bit para el signo.
- Rango representable $-e$ a $2^n - e - 1$.
- Permite representar un rango arbitrario.

Conversión X (Decimal) $\rightarrow$ Y (Exceso-e):

Y = base2(X+e)

Conversión Y (Exceso-e) $\rightarrow$ X (Decimal):

X = base10(Y)−e

Ejemplo con 4 bits:

Binario	Exceso-5
0000	-5
0001	-4
0010	-3
0011	-2
0100	-1
0101	0
0110	1
0111	2
1000	3
1001	4
1010	5
1011	6
1100	7
1101	8
1110	9
1111	10

Números enteros

Algunos ejemplos de uso:

- Sin signo: `unsigned int` en C++, 32 bits, 0 a 4294967295.
- Signo y Magnitud: primeras computadoras (ej: IBM 7090, de 1958).
- Complemento a 2: tipo `int` en C++, de 32 bits, -2147483648 a 2147483647.
- Notación Exceso-e: Representación del exponente en IEEE-754 (ver slide 19).

En los lenguajes que tienen un máximo entero representable fijo (ej: C++, Java), cualquier operación que arroje un resultado >`MAX` o <-`MAX` dará un error de *overflow*.

Python3, Ruby y otros lenguajes modernos usan secuencias de bits de longitud variable para representar enteros. No limitan a priori el tamaño de los enteros representables, aunque en la práctica siempre habrá un límite dado por la memoria física disponible.

Números reales

Los números en una computadora son muy diferentes de los reales matemáticos (el conjunto $\mathbb{R}$).

Las siguientes igualdades son matemáticamente ciertas:

$$\sqrt{2.0} = 1.4142135623730950488016887242096980785696718753769\ldots$$
$$\sqrt{2.0}^2 = 2.0$$

En una computadora, para representar números se usa una cantidad acotada de bits. Por lo tanto, no hay forma de representar todo el desarrollo de la parte decimal de $\sqrt{2.0}$.

$$\sqrt{2.0} = 1.4142135623730951$$

¡Hay infinitos reales representados por 1.4142135623730951!

1.4142135623730951 $\rightsquigarrow$ 1.4142135623730951
1.4142135623730951234 $\rightsquigarrow$ 1.4142135623730951
1.4142135623730951412976 $\rightsquigarrow$ 1.4142135623730951
...

Números reales

Los números en una computadora son muy diferentes de los reales matemáticos (el conjunto $\mathbb{R}$).

Las siguientes igualdades son matemáticamente ciertas:

$\sqrt{2.0} = 1.4142135623730950488016887242096980785696718753769\ldots$

$\sqrt{2.0}^2 = 2.0$

En una computadora, $\sqrt{2.0} = 1.4142135623730951$

Luego $\sqrt{2.0}^2 = 1.4142135623730951^2 = 2.0000000000000004$

Luego $\sqrt{2.0}^2 \neq 2.0$

Están acotados no solo por encima y por debajo (como los enteros), sino también en su precisión. Esto lleva a errores numéricos en los cálculos. Cómo lidiar con estos errores se ve en *Métodos Computacionales*.

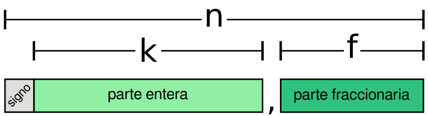
Veamos dos formas de representar números reales con secuencias de bits:

- Notación de Punto Fijo
- Notación de Punto Flotante

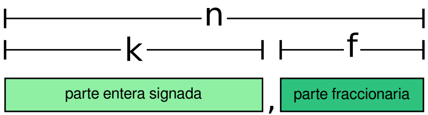
Punto Fijo

Utilizamos n bits en total para representar el número.

- Un bit de signo, k para la parte entera y f para la parte fraccionaria.



- Alternativamente, el signo puede ser codificado en la parte entera. Por ejemplo, codificado como complemento a 2.



Punto Fijo - Ejemplo

Suponer una codificación de la forma:

signo (1 bit)	parte entera (9 bits)	parte fraccionaria (6 bits)

- **Ejemplo 1:** Secuencia de bits 0011001010001001
  La interpretamos como: (0)011001010.001001
  $\begin{cases} (0) \rightarrow \text{Positivo} \\ 011001010 \rightarrow 202 \\ 001001 \rightarrow 0.140625 \end{cases}$
  Luego, 0011001010001001 $\rightarrow 202.140625$

- **Ejemplo 2:** Secuencia de bits 1000000110000111
  La interpretamos como: (1)000000110.000111 $\rightarrow$ -6.109375
  $\begin{cases} (1) \rightarrow \text{Negativo} \\ 000000110 \rightarrow 6 \\ 000111 \rightarrow 0.109375 \end{cases}$
  Luego, 1000000110000111 $\rightarrow -6.109375$

La notación de punto fijo se usó en lenguajes importantes como COBOL y Ada. Luego cayó en desuso, siendo reemplazada por la notación de punto flotante.

Números de punto flotante (`float`)

Un real r se representa en punto flotante por una terna (s, e, f) tal que:

$$r \approx s \times f \times 2^e \qquad \text{donde} \quad s \in \{-1, 1\} \quad \text{y} \quad 1 \leq f < 2$$

s **(signo)**	e (exponente)	f (fracción)

El signo s es un bit: 0=positivo; 1=negativo.

El exponente e es un entero representado con alguna de las notaciones vistas. Puede ser negativo.

En la fracción f (también llamada *mantisa*, *coeficiente*, o *significando*), el 1 de la parte entera está implícito. Los bits `xxxxxxxxx` se interpretan como `1.xxxxxxxxx`.

Punto Flotante · ¿Cómo se interpreta una secuencia de bits?

Suponer codificación: | signo (1 bit) | exponente (6 bits) | fracción (9 bits) |

El exponente está representado en complemento a 2.

- **Ejemplo 1:** Secuencia de bits 1000101101010000
  La interpretamos como: $(1)1.101010000 \times 2^{000101}$
  $$\begin{cases} (1) \to \text{Negativo} \\ 1.101010000 \to 1.65625 \\ 000101 \to 5 \end{cases}$$
  Luego, 1000101101010000 $\to -(1.65625 \times 2^5) \to -53$

- **Ejemplo 2:** Secuencia de bits 0111110111010111
  La interpretamos como: $(0)1.111010111 \times 2^{111110}$
  $$\begin{cases} (0) \to \text{Positivo} \\ 1.111010111 \to 1.919921875 \\ 111110 \to -2 \end{cases}$$
  Luego, 0111110111010111 $\to 1.919921875 \times 2^{-2} \to 0.48$

Punto Flotante · ¿Cómo se construye una secuencia de bits?

Suponer codificación: | signo (1 bit) | exponente (6 bits) | fracción (9 bits) |

El exponente está representado en complemento a 2.

- **Ejemplo 1:** Número -53

$$\begin{cases} \text{Negativo} \rightarrow (1) \\ 53 \rightarrow 110101 = 1.10101 \times 100000 = 1.101010000 \times 2^5 \\ 5 \rightarrow 000101 \end{cases}$$

Luego, escribimos -53 como 1000101101010000

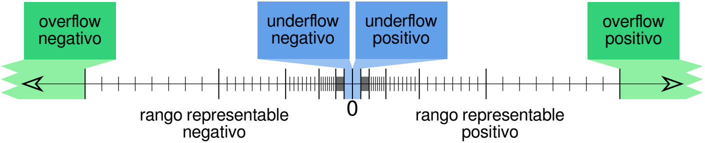
- **Ejemplo 2:** Número 0.48

$$\begin{cases} \text{Positivo} \rightarrow (0) \\ 0.48 \rightarrow 0.01111010111\ldots = 1.111010111 \div 100 = 1.111010111 \times 2^{-2} \\ -2 \rightarrow 111110 \end{cases}$$

Luego, escribimos 0.48 como 0111110111010111

Punto Flotante - Rango de representación

La representación en punto flotante no es uniforme sobre la recta númerica.

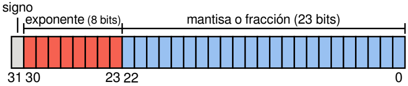


- **Error de overflow**: Magnitudes que grandes que el máximo valor absoluto representable.
- **Error de underflow**: Magnitudes más chicas que el mínimo valor absoluto representable (distinto de cero).

Punto Flotante · IEEE-754

Una codificación en punto flotante muy usada es el estándar IEEE-754, que viene en dos versiones: de 32 bits y de 64 bits.

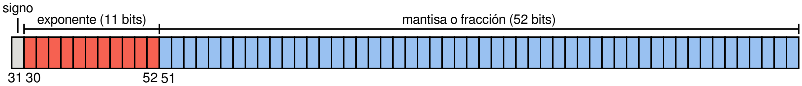
32 Bits:



$$(\texttt{signo})\ 1.\text{fracción} \times 2^{\text{exponente}-\textit{exceso}}$$

para 32 bits, *exceso*$=127$
para 64 bits, *exceso*$=1023$

64 Bits:



C++ y Java tienen los tipos `float` y `double`, siguiendo el estándar IEEE-754 de 32 y 64 bits, respectivamente.

El tipo `float` de **Python** sigue el estándar IEEE-754 de 64 bits.

Caracteres

Existen muchas formas de ~~representar~~ codificar caracteres.

Las codificaciones se basan en **tablas**, que indican qué bits corresponden a cada carácter.

Dependiendo de la cantidad de bits/bytes usados para codificar cada carácter, pueden ser de tamaño fijo o variable.

Algunos ejemplos:

- `ASCII`: Fija, 1 byte. Aunque solo se usan 7 bits para codificar caracteres.
- `UTF-8`: Variable, 1 a 4 bytes. Codificación Unicode de longitud variable.
- `UTF-16`: Variable, 2 o 4 bytes. Codificación Unicode optimizado para caracteres multilingües.
- `UTF-32`: Fija, 4 bytes. Codificación Unicode simple.
- `Latin-1 (ISO-8859-1)`: Fija, 1 byte. Caracteres latinos, tildes, diéresis, cedilla, eñe, etc.
- `GB 18030`: Variable, 1 a 4 bytes. Estándar utilizado en China.

Representación de caracteres · ASCII

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	16	10	DLE	32	20		48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
1	01	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	02	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	03	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	04	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	05	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	06	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	07	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	08	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	09	HT	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	0A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	0B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	0C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	0D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	0E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	0F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Representación de caracteres · UTF-8

Los caracteres se codifican según el rango al que pertenezcan.
Los primeros 127 corresponden a la codificación ASCII.

#bytes	desde	hasta	byte 1	byte 2	byte 3	byte 4
1	0	127	0xxxxxxx			
2	128	2047	110xxxxx	10xxxxxx		
3	2048	65535	1110xxxx	10xxxxxx	10xxxxxx	
4	65536	1114111	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

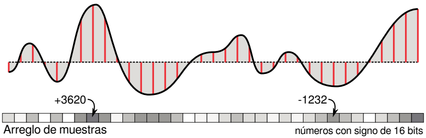
El primer byte indica cuántos bytes a continuación se deben leer (mismo prefijo).

Ejemplos:

U+03A3 → ce a3 → GREEK CAPITAL LETTER SIGMA → Σ
U+2197 → e2 86 97 → NORTH EAST ARROW → ↗
U+10890 → f0 90 a2 90 → NABATAEAN LETTER FINAL LAMEDH → 𐢐

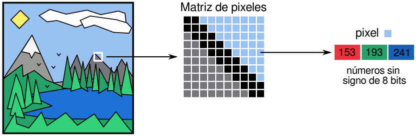
Representación de datos

Sonido



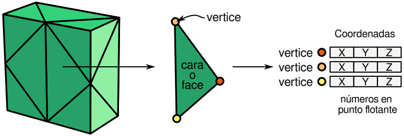
Ejemplo: `Formato WAV`
Almacena muestras de señales de audio sin comprimir, permite (p.ej.) frecuencias de muestro de 16 bits a 44kHz.

Imagen



Ejemplo: `Formato BMP`
Guarda mapas de bits sin compresión. Permite guardar imágenes en escala de grises y en colores de 24 o 32 bits.

Diseño 3D



Ejemplo: `Formato STL`
Permite representar superficies 3D por medio de la descripción de triángulos en coordenadas cartesianas.

Repaso de la clase de hoy

- Representación de números enteros
  - Sin Signo; Signo y Magnitud; Complemento a 2; Exceso-e.
  - Error de overflow.
- Representación de números reales
  - Punto Fijo; Punto Flotante.
  - Errores de overflow y underflow.
- Representación de caracteres
  - ASCII; UTF-8.

Bibliografía complementaria:

- Tanenbaum, "Organización de Computadoras. Un Enfoque Estructurado", 4ta Edición, 2000. Apéndices: "Números Binarios" y "Números de Punto Flotante" (disponibles en la sección *Extras* de la página de la materia).

Con lo visto, ya pueden resolver toda la Guía de Ejercicios 6.