

Computadors

Representació de Dades

Grau en Ciència i Enginyeria de Dades

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

Creative Commons License

Aquest document utilitza Creative Commons Attribution 3.0 Unported License



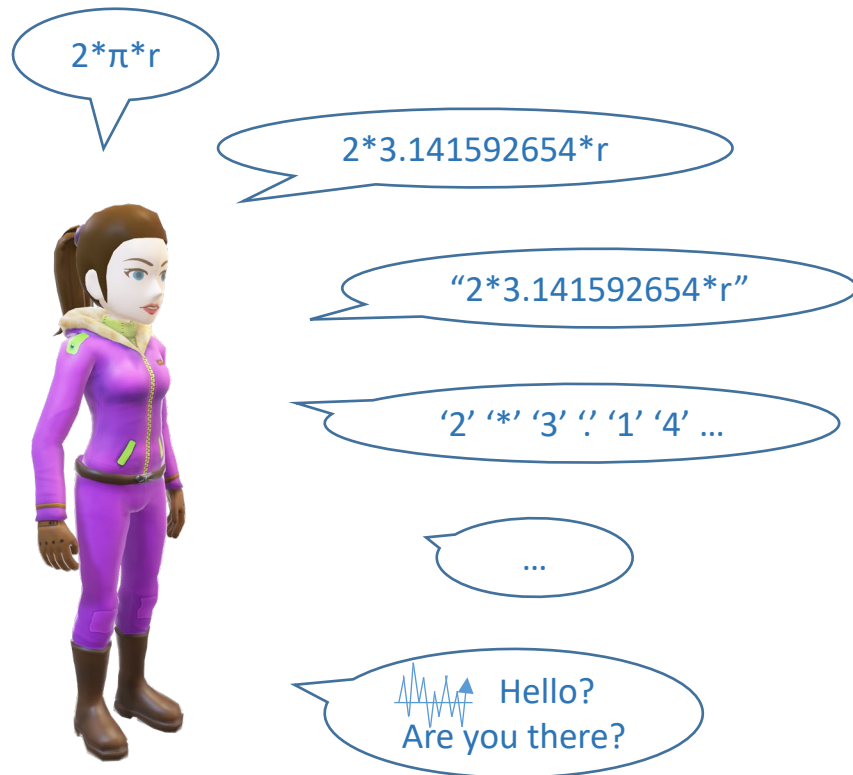
Els detalls d'aquesta licencia es poden trobar a <https://creativecommons.org/licenses/by-nc-nd/4.0>

Index

- Conceptes Bàsics
 - Representació de dades en computadors
- Representació del Tipo de Dades
 - Impacte de l'error de precisió
 - Impacte de incompatibilitats
 - Dependències amb Hardware i Software

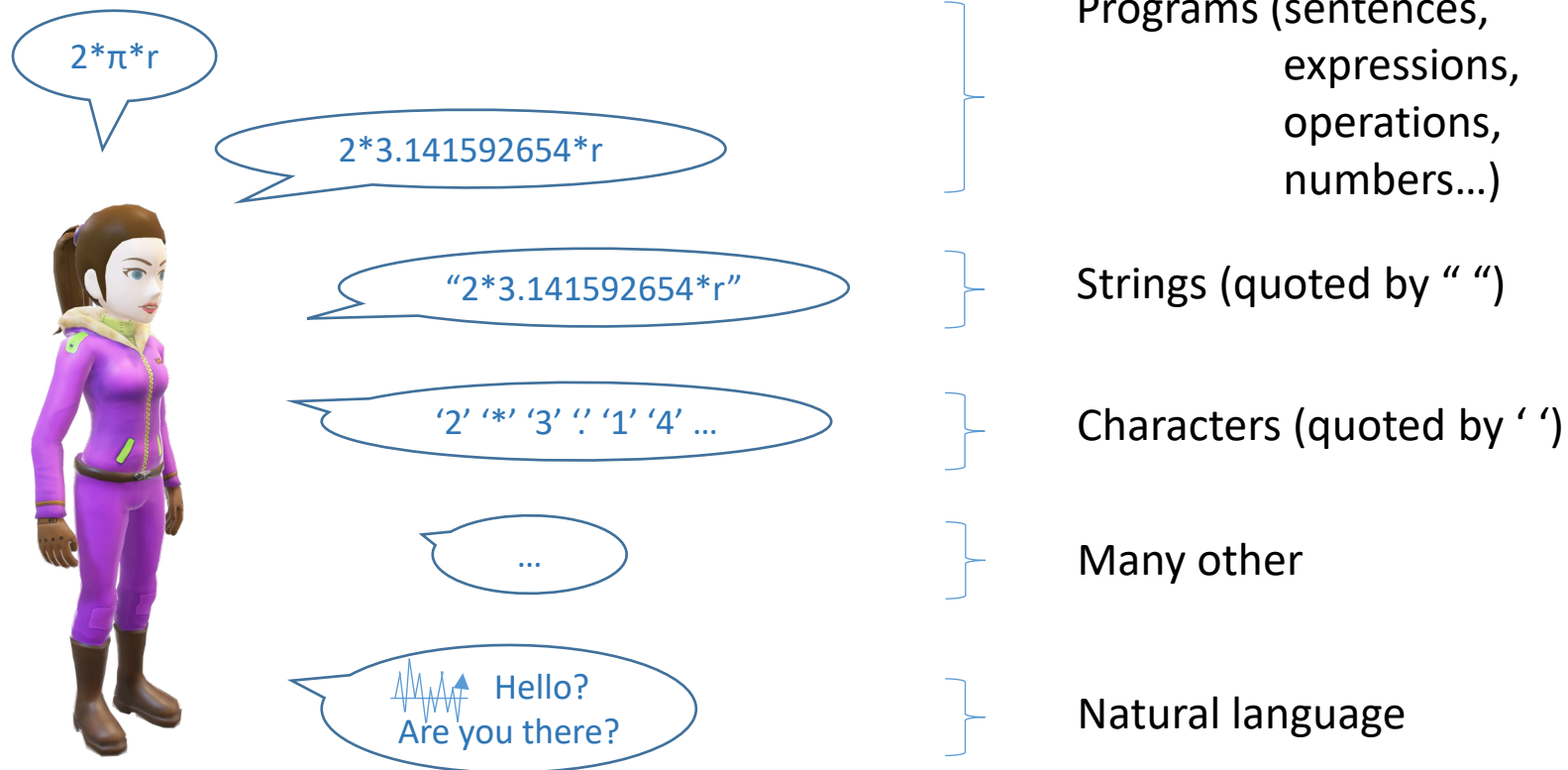
Conceptes Bàsics

- Necessitat de representar dades en l'ordinador



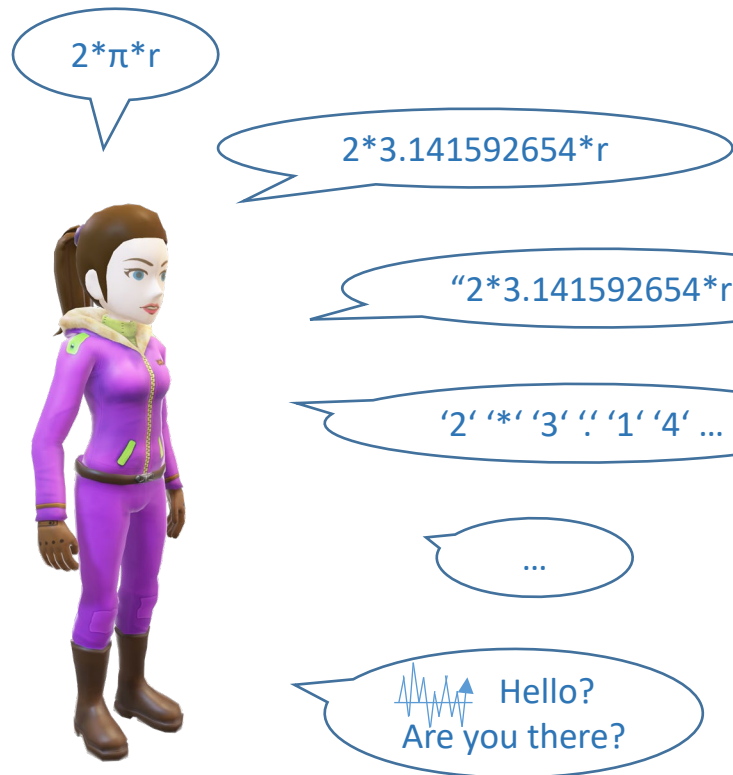
Conceptes Bàsics

- Necessitat de representar dades en l'ordinador



Conceptes Bàsics

- L'ordinador es basa en transistors
 - Qualsevol dada hauria de poder-se representar amb 0s i 1s



Instructions: 0010 1010 1100 1010 1101
0010 1010 110010010000111 1010 1101

Strings: "0110 1001 0111 1000 0101 1100 ..."

Characters:
0110 1001 0111 1000 0101 1100 ...

Natural language, sound, images...

 $\rightarrow$ 0¹00¹0¹00¹0¹0¹100¹1...



Conceptes Bàsics

- Dígit Binari: número que pot tenir dos valors: 0 o 1 (obert o tancat)
 - Moltes interpretacions: true/false, positive/negative, on/off, 0/1, etc
- Bit: contracció de “Binary Information Digit” (*John W. Tukey, 1947*)
 - És la màxima quantitat d’informació que pot ser expressada per un dígit binari
 - Un dígit binari pot expressar entre zero i un bit d’informació

Conceptes Bàsics – codificació d'un caràcter

- Byte: unitat d'informació digital, normalment 8 bits
 - És la unitat més petita de memòria adreçable en moltes, però no totes, arquitectures de processador
 - Es va introduir durant els '60s i popularitzat per Intel (8008) en els '70s
 - S'utilitza per codificar caràcters de text (ASCII)

...

0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---

 (interpretat com un caràcter ASCII) 'M'

0	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

 (interpretat com un caràcter ASCII) 'N'

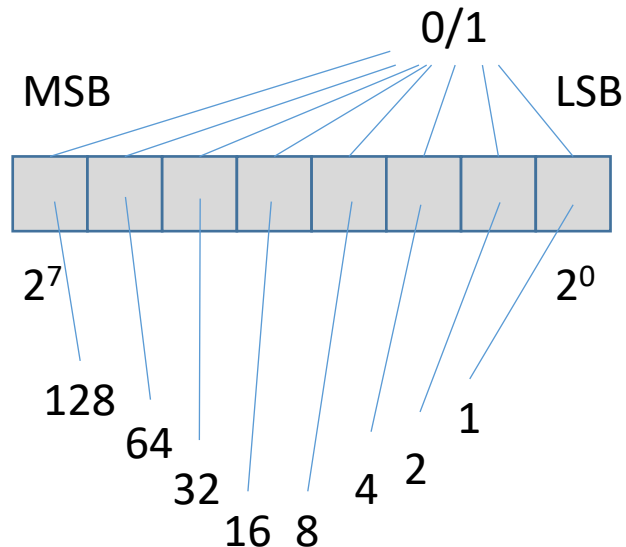
0	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

 (interpretat com un caràcter ASCII) 'O'

...

Conceptes Bàsics – 8-bit (byte) nombre binari

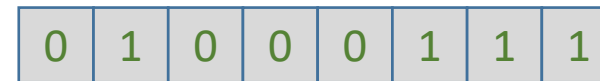
- LSB: Least Significant Bit -
- MSB: Most Significant Bit -



dona el valor 0 o $1 * 2^0 = 0$ o **1**

dona el valor 0 o $1 * 2^7 = 0$ o **128**

Example



$$64 + 4 + 2 + 1 = \mathbf{71}$$

Interpretat com un “número” **71**

Interpretat com un “caràcter” ‘**G**’ (ASCII)

- **NOTA: de moment, només podem representar nombres Naturals [0...255]**

Conceptes Bàsics – 8-bit (byte) nombre binari

- Per passar de decimal a binari
 - Procés de divisions per 2
- Per passar de binari a decimal
 - Procés de multiplicacions per 2
 - Tenir en compte el pes de cada dígit

Conceptes Bàsics – 8-bit (byte) nombre binari

- Exemples:

- Nombre Decimal: 97
- Representació binaria?



(també interpretat com ASCII) 'a' (lowercase)

- Nombre Decimal: 53
- Representació binaria?



(també interpretat com ASCII) '5' (digit)

Representacions de Nombres amb Signe

- Un nombre binari pot representar valors negatius utilitzant diferents mètodes
 - MSB = 1 representa números negatius
 - Optimitzar les restes
 - E.g. sumar números negatius
- Els mètodes més coneguts i usats són:
 - Signe i magnitud
 - Normalment per representar la mantisa de números de coma flotant (veure transparències posteriors)
 - Complement a 1
 - Primers ordinadors i alguns usos actuals, però particulars
 - Complement a 2
 - La majoria d'ordinadors
 - Representació Biased
 - Exponent dels números de coma flotant (veure transparències posteriors)

Complement a 2

- Rang:
 - De $-2^{(N-1)}$ a $+(2^{(N-1)}-1)$
- Avantatge respecte complement a 1
 - Ignorar el bit d'overflow en operacions (e.g. resta)
- Números negatius:
 - Número Decimal -> invertir bits -> sumar +1
- Exercici:
 - Número: -23
 - Quina és la seva representació en Complement a 2?

Example

0	1	0	0	0	1	1	1
---	---	---	---	---	---	---	---

-2^7 2^6

64 + 4 + 2 + 1 = **71**

1	1	0	0	0	1	1	1
---	---	---	---	---	---	---	---

-2^7 2^6

-128 + 64 + 4 + 2 + 1 = **-57**

1	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

-2^7 2^6

-128 + 32 + 16 + 8 + 1 = **-71**

0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

-2^7 2^6

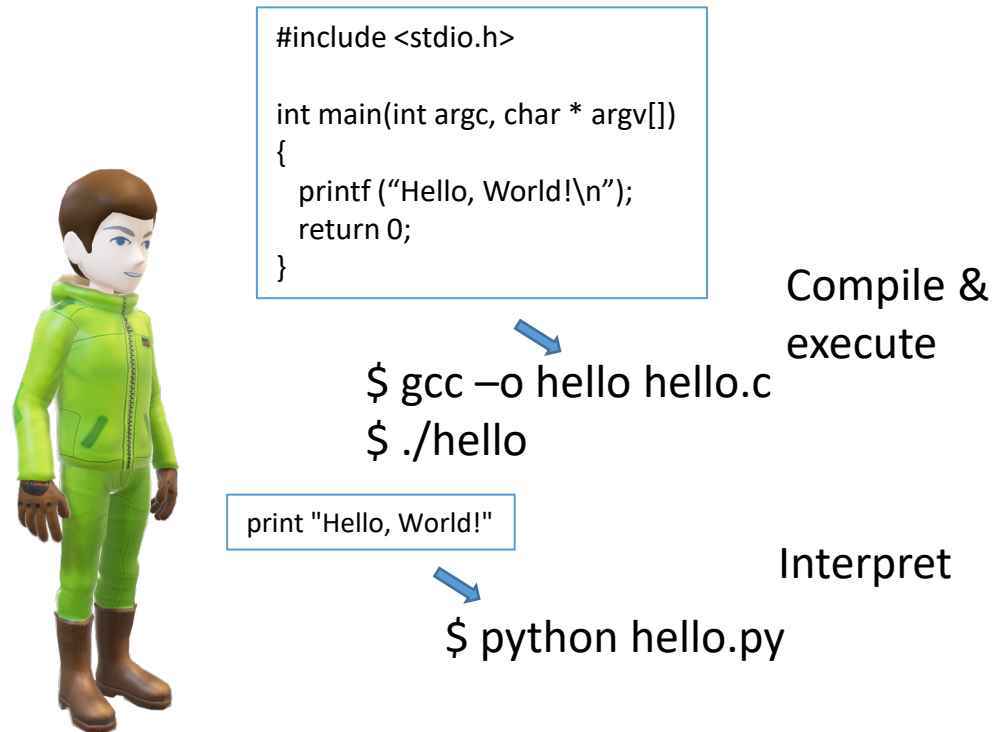
32 + 16 + 8 + 1 = **57**

Conceptes Bàsics

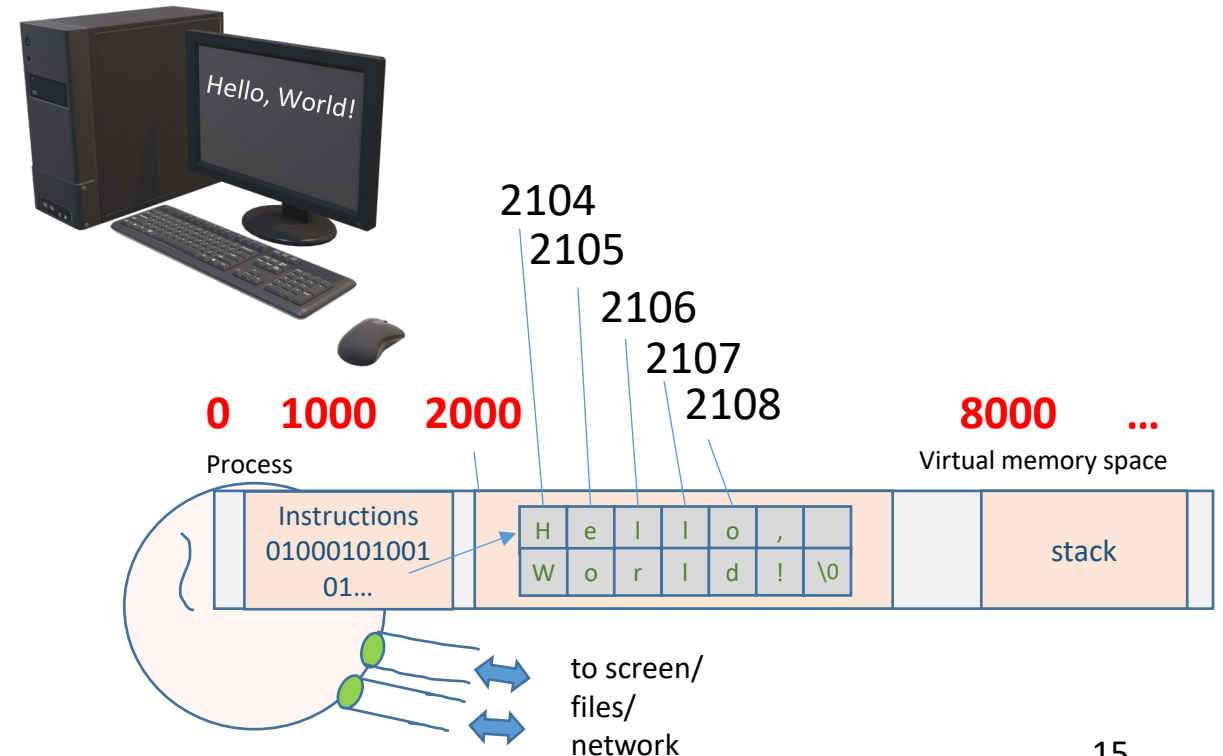
- El tipus de Dada és una classificació que permet al compilador/intèrpret (veurem més endavant) saber com es tractarà la dada
- Impacte directe en el tamany i format
- Software i hardware suport
 - Compilador/Intèrpret i arquitectura
 - 32-bit vs 64-bit

Conceptes Bàsics

Compilador/Intèrpret



Memòria



Aquests conceptes es completaran en sessions posteriors

Tipus de Dades – Clasificació Bàsica

- **Tipus de dades Escalar:** un únic valor
 - Arithmetic (numbers), symbols and characters, boolean, enumeration, pointers
- **Tipus de dades Especials:** Void -- equivalent a no-data (de mida 0), o tipus de dada no especificada
- **Tipus de dades Compostes/Agregades:** combinació d'un o més tipus scalars
 - Arrays, structures, unions, objectes

Conceptes Bàsics

- Endianness

- Ordre dels valors dels bytes en memòria

E.g. $(1234)_{10} = (04\ D2)_H$

- Big-Endian

- El Byte amb **més** valor està en la primera posició (menor @ memòria)
- Xarxa i mainframes

@	Value
0x0000	04
0x0001	D2

- Little-Endian

- El Byte amb **menor** valor està en la primera posició (menor @ memòria)
- Família de processadors Intel x86 i la majoria de microprocessadors

@	Value
0x0000	D2
0x0001	04

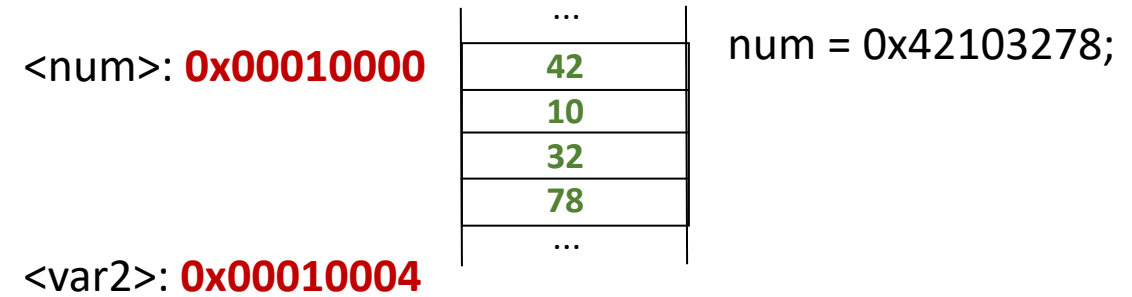
- Algunes architectures suporten les dues

- E.g. Arm i IBM POWER totalment; recents x86 i x86-64 suport limitat (movbe instruction)
 - Movbe instruction: Move Data After Swapping Bytes

Conceptes Bàsics

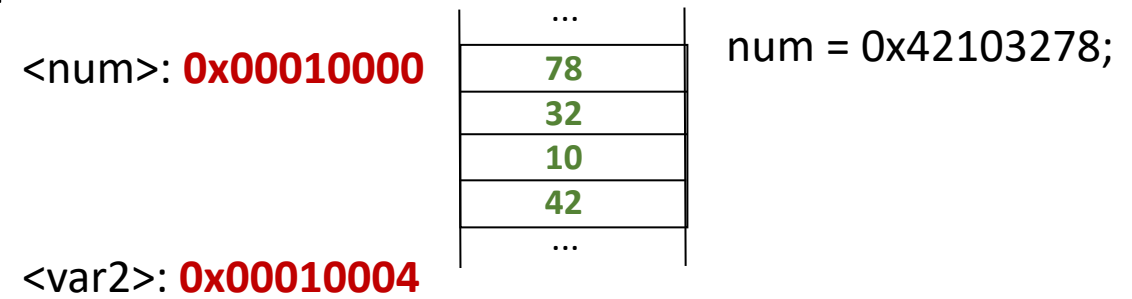
- Big Endian: la @ de memòria apunta al “Big end” del número

(com escriure bytes d'esquerra a dreta)



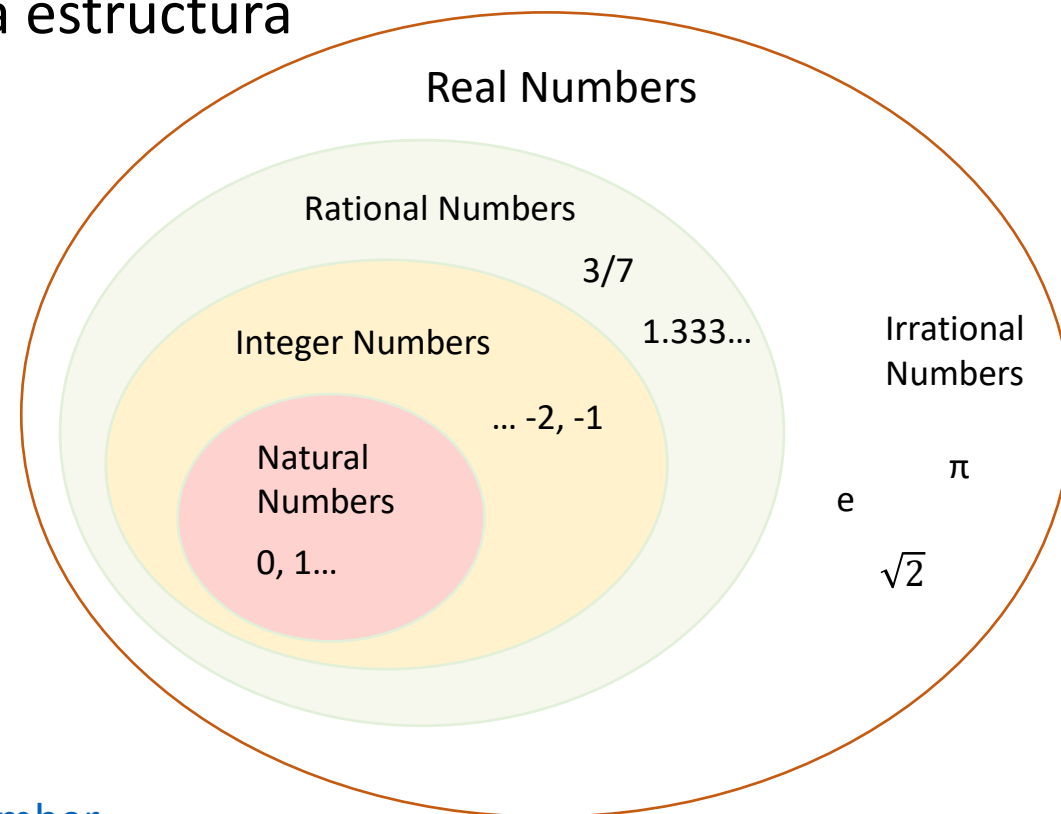
- Little Endian: la @ de memòria apunta al “Little end” del número

(com escriure bytes de dreta a esquerra)



Tipus de Dades Escalars: Arithmetic (números)

- Nombres Enters i Reals
 - Números Complexos es guarden com una estructura
- Signe i sense signe
- Sistemes numèrics
 - Binari, Octal, Decimal, Hexadecimal



Enters

- Char: 1 Byte
 - Signed: de -128 a 127
 - Unsigned: de 0 a 255
- Short: 2 Bytes
 - Signed: de -32,768 a 32,767
 - Unsigned: de 0 a 65,535
- Integer: 4 Bytes
 - Signed: de -2,147,483,648 a 2,147,483,647
 - Unsigned: de 0 a 4,294,967,295
- Long Long: 8 Bytes (si és possible)
 - Signed: de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
 - Unsigned: de 0 a 18,446,744,073,709,551,615
- Es poden expressar utilitzant diferents notacions:
 - Normalment decimal, octal, hexadecimal



1 byte



2 bytes



4 bytes



8 bytes

Notacions numèriques

- Segons l'ús/significant, unes poden ser més útils que altres

Binary	Octal	Decimal	Hexadecimal
111	07	7	0x7
1000	010	8	0x8
1001	011	9	0x9
1010	012	10	0xA (also 0xa, or 0x 000 a)
1011	013	11	0xB (0xb)
1100	014	12	0xC (0xc)
1101	015	13	0xD (0xd)
1110	016	14	0xE (0xe)
1111	017	15	0xF (0xf)
10000	020	16	0x10
10001	021	17	0x11
11100110	0346	118	(11100110) 0xE6

Integer vs Long Integer

- Long integer depèn tant de l'arquitectura com del Sistema Operatiu

OS	Arch	Size (Bytes)
Windows	IA-32	4
Windows	x86_64	4
Linux	IA-32	4
Linux	x86_64	8
Mac OS X	IA-32	4
Mac OS X	x86_64	8

- Cuidado amb el codi
 - **El comportament pot canviar**
 - E.g.: IA-32 vs x86_64 Linux, x86_64 Win vs x86_64 Linux

Operacions amb enters

- Operacions Binaries: suma, resta, multiplicació, divisió...
shift, logical and, logical or, logical exclusive-or, logical not
 - Operacions bit a bit, com el sistema decimal
 - Implementat amb transistors

Addition

0	0	1	1	0	0
---	---	---	---	---	---

8 + 4 = 12

+

0	0	0	1	0	1
---	---	---	---	---	---

4 + 1 = 5

0	1	0	0	0	1
---	---	---	---	---	---

16 + 1 = 17

Shift left

0	0	0	1	0	1
---	---	---	---	---	---

↓ (like *2)

0	0	1	0	1	0
---	---	---	---	---	---

Shift right

0	1	1	0	1	0
---	---	---	---	---	---

↓ (like /2)

0	0	1	1	0	1
---	---	---	---	---	---

Substraction

0	0	1	1	0	0
---	---	---	---	---	---

8 + 4 = 12

-

0	1	0	1	0	1
---	---	---	---	---	---

16 + 4 + 1 = 21

1	1	0	1	1	1
---	---	---	---	---	---

32+16 + 4+2+1 = 55
-32+16 + 4+2+1 = -9

Operacions amb enters

- Els Flags poden indicar condicions concretes:
 - Zero: el resultat de l'operació és 0 (estalvia temps si hi ha una comprovació posterior)
 - Negative: el resultat de l'operació és <0 (també pot estalviar temps)
 - Carry: només per nombres **sense signe**, indica que el resultat no es pot representar
 - Succeeix quan l'operació genera un 1 que no cap en la quantitat de bits usats per representar el número

Sumar: (11111) + (00001)

- Overflow: només per nombres **amb signe**, indica que el resultat no es pot representar
 - Succeeix quan el bit de carry de la posició 2^{n-1} i el bit de carry (el flag) són diferents

Sumar: (01111) + (00001)

Into sign	Into carry	Overflow
0	0	0
0	1	1
1	0	1
1	1	0

Operacions Lògiques

- Només operacions a nivell de bit (no és possible tenir carry)
- Normalment amb números Naturals (unsigned) (no és possible overflow)

Logical and

0 0 1 1 0 0

8 + 4 = 12

And

0 1 0 1 0 1

16 + 4 + 1 = 21

0 0 0 1 0 0

4 = 4

0 0 1 1 0 0 or 0 1 0 1 0 1 = 0 1 1 1 0 1

0 0 1 1 0 0 xor 0 1 0 1 0 1 = 0 1 1 0 0 1

not 0 1 0 1 0 1 = 1 0 1 0 1 0

Truth table
(taula de veritat)

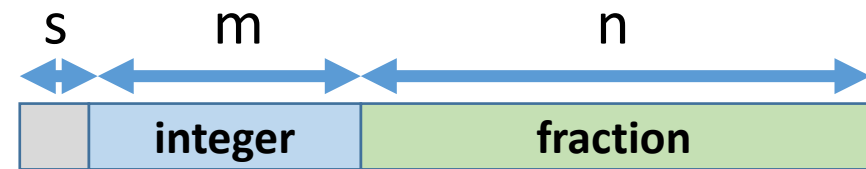
Src 0	Src 1	AND	OR	XOR	NOT (Src 0)
0	0	0	0	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	1	0	

Nombres Reals

- Números amb un component fraccional
- Dos representacions
 - Fixed-point vs Floating-point
 - El punt és fixe o pot “flotar” a qualsevol lloc del número
 - És el símbol per separar la part entera de la fraccional d'un número real
- Implementació: compromís entre cost i precisió
 - Falta de recursos hardware
 - E.g.: Decodificadors Multimedia
 - Optimitzar el rendiment encara que es degradi la precisió
 - E.g. Playstations, Doom

Números “Fixed-point”

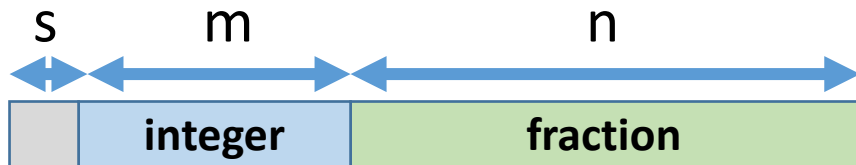
- Bits = 1 + m + n
 - 1 bit per signe (si en té)
 - m bits pel component enter
 - n bits pel component fraccional



- Notació: $Q_{m.n}$
 - Número enter sense component fraccional ($Q_{m.0}$)
 - Número fraccional sense component entera (Q_n)

Números “Fixed-point”

- **Valor** = $-2^m b'_s + 2^{m-1} b'_{m-1} + \dots + 2^1 b'_1 + 2^0 b'_0 + 2^{-1} b_{n-1} + 2^{-2} b_{n-2} + \dots + 2^{-n} b_0$
- Suport del llenguatge de programació
 - C i C++ no tenen suport directe, però es poden implementar
 - Embedded-C ho suporta (implementat en GCC)
 - Python té support directe mitjançant un mòdul
 - I.e. decimal module



Exemple: 1110

$$Q3.0: -2^3 + 2^2 + 2^1 = -2 \quad (1110.)$$

$$Q1.2: -2^1 + 2^0 + 2^{-1} = -2 + 1 + 0.5 = -0.5 \quad (11.10)$$

$$Q3: -2^0 + 2^{-1} + 2^{-2} = -1 + 0.5 + 0.25 = -0.25 \quad (1.110)$$

Números “Fixed-point”

- Bits = 1 + m + n
 - 1 bit per signe (si en té)
 - m bits pel component enter
 - n bits pel component fraccional

- Exemple, 1 byte

- Sign, 1 bit
- Integer, 5 bits -> 0 .. 31, amb signe -32 .. +31
- Fraction, 2 bits -> 0, 0.25, 0.50, 0.75
- Rang: -31.75 .. +31.75

0	1	0	0	0	1	1	1
---	---	---	---	---	---	---	---

$$\begin{array}{ccccccc} -2^5 & 2^4 & & & & 2^0 & 2^{-1} & 2^{-2} \\ & 16 & & & & 1 & +0.5 & +0.25 \end{array} \quad + \quad = \quad \mathbf{17.75 \quad (=71/4)}$$

1	1	0	0	0	1	1	1
---	---	---	---	---	---	---	---

$$\begin{array}{ccccccc} -2^5 & 2^4 & & & & 2^0 & 2^{-1} & 2^{-2} \\ -32 & +16 & & & & 1 & +0.5 & +0.25 \end{array} \quad + \quad = \quad \mathbf{-14.25 \quad (= -57/4)}$$

Conversió de la part fraccional

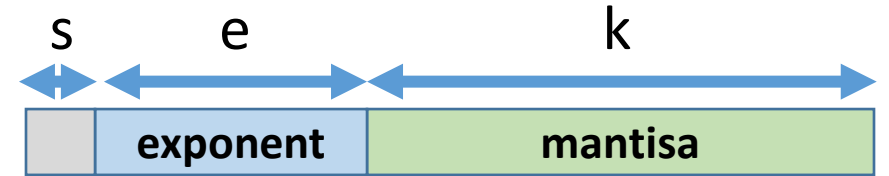
- Per passar de decimal a binari
 - Procés de divisions per 2
- Per passar de binari a decimal
 - Procés de multiplicacions per 2
 - Tenir en compte el pes de cada dígit
- Provem a transformar de decimal a binari
 - 0,3125
 - 0,3

Números “Fixed-point”: problemes de precisió

- Pèrdua de precisió i overflow
- Els resultats poden necessitar més bits que els operands
 - Arrodonir o truncar
 - Especificar una mida diferent pel resultat
- Números límit per prevenir overflow
- Excepció: overflow flag
 - Si té el suport del hardware

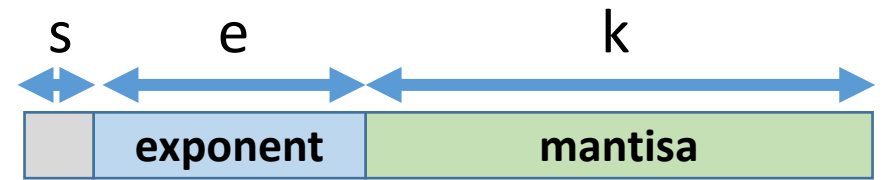
Números “Floating-point”

- Bits = 1 + e + k
 - 1 bit per signe (si en té)
 - e bits pel exponent $\{1, \dots, (2^e - 1) - 1\}$
 - k bits per la mantissa (fracció)
 - Hi ha un bit implícit (a l'esquerra) igual a 1, a no ser que l'exponent sigui igual a zero
- La majoria de processadors segueixen l'estàndar IEEE de coma flotant
 - Primera versió en el 1985
 - Formats estandaritzats
 - Valors Especials



Números “Floating-point”

- **Valor** = $(-1)^{\text{sign}} * (1 + \sum_1^k b_{(k-i)} 2^{-i}) * 2^{(e-(E_{\text{max}}))}$



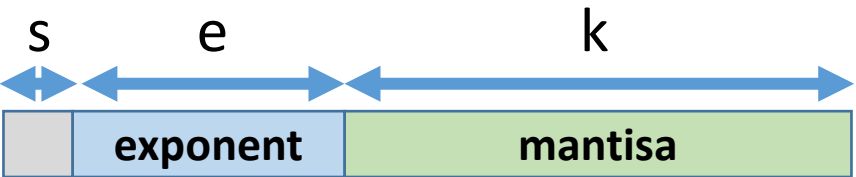
- Float: 4 Bytes
 - Sign bit, 8-bit exponent, 23-bit mantisa
- Double: 8 Bytes
 - Sign bit, 11-bit exponent, 52-bit mantisa
- Alguns llenguatges suporten 10 bytes
 - Sign bit, 15-bit exponent, (1+63)-bit mantisa

Biased Exponent (bias = 127):

0	Denormals numbers 0.xxx
1	Smallest normal exponent
127	E+0
254	Largest normal exponent
255	Infinite / NaNs

Números “Floating-point”

- Codificació IEEE754**



Sign Biased exp.

Magnitude

Single precision	8 bits	23 bits	(1+31)
Double precision	11 bits	52 bits	(1+63)
Double extended (Intel) precision	15 bits	1+63 bits	(1+1+78)

(represented
in this table)

Class		Sign	Exp.	Mantisa
Positive	$+\infty$	0	11 ... 11	1 . 00 ... 00
	+Normals	0	11 ... 10	1 . 11 ... 11
	+3.40 E+38	...	...	...
	+1.000 E+0	0	01 ... 11	1 . 00 ... 00
	+1.17 E-38	0	00 ... 01	1 . 00 ... 00
	+Denormals	0	00 ... 00	0 . 11 ... 11
	+1.17 E-38	...	...	...
Negative	+1.40 E-45	0	00 ... 00	0 . 00 ... 01
	+Zero	0	00 ... 00	0 . 00 ... 00
	-Zero	1	00 ... 00	0 . 00 ... 00
	-Denormals	1	00 ... 00	0 . 00 ... 01
	-1.40 E-45	...	...	...
	-1.17 E-38	1	00 ... 00	0 . 11 ... 11
	-Normals	1	00 ... 01	1 . 00 ... 00
NaN	-1.000 E+0	1	01 ... 11	1 . 00 ... 00
	...	...	...	...
	-3.40 E+38	1	11 ... 10	1 . 11 ... 11
	$-\infty$	1	11 ... 11	1 . 00 ... 00
	SNaN	X	11 ... 11	1 . 0X ... XX
	QNaN	X	11 ... 11	1 . 1X ... XX
	-QNaN	1	11 ... 11	1 . 10 ... 00

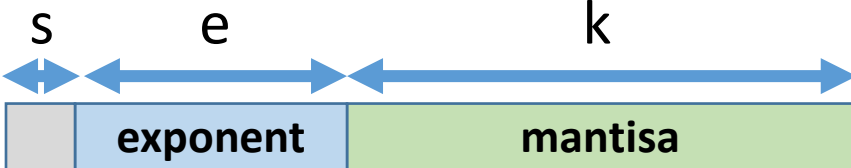
“floating point” de Doble precisió

- Taula equivalent per 64-bit

Class		Sign	Exp.	Mantissa
Positive	$+\infty$	0	11 ... 11	1 . 00 ... 00
	+Normals	0	11 ... 10	1 . 11 ... 11
	+1.79 E+308	...	...	...
	+1.000 E+0	0	01 ... 11	1 . 00 ... 00
	+2.22 E-308	0	00 ... 01	1 . 00 ... 00
	+Denormals	0	00 ... 00	0 . 11 ... 11
	+2.22 E-308	...	...	...
	+4.94 E-324	0	00 ... 00	0 . 00 ... 01
	+Zero	0	00 ... 00	0 . 00 ... 00
Negative	-Zero	1	00 ... 00	0 . 00 ... 00
	-Denormals	1	00 ... 00	0 . 00 ... 01
	-4.94 E-324	...	...	...
	-2.22 E-308	1	00 ... 00	0 . 11 ... 11
	-Normals	1	00 ... 01	1 . 00 ... 00
	-1.000 E+0	1	01 ... 11	1 . 00 ... 00
	...	...	...	...
	-1.79 E+308	1	11 ... 10	1 . 11 ... 11
NaN	$-\infty$	1	11 ... 11	1 . 00 ... 00
	SNaN	X	11 ... 11	1 . 0X ... XX
	QNaN	X	11 ... 11	1 . 1X ... XX
	-QNaN	1	11 ... 11	1 . 10 ... 00

Números “Floating-point”

• **Valor** = $(-1)^{\text{sign}} * (1 + \sum_{i=1}^k b_{(k-i)} 2^{-i}) * 2^{(e-(E_{\max}))}$



Exemple: $(23,46875)_{10} = (23)_{10} + (0,46875)_{10} =$
 $= (10111)_2 + (0.01111)_2 = (10111.01111)_2 =$
 $= (1.011101111)_2 * 2^4$

Representació 32-bit floating point

Signe (1-bit) = 0

Exponent (8-bit) = 4 = $(131 - E_{\max}) = (131 - 127)$
 $= (10000011)_2$

Mantisa (23-bit) = (0111011110...0)

0 10000011 011101111000000000000000
(41BB C000)_H

Números “Floating-point”: suport

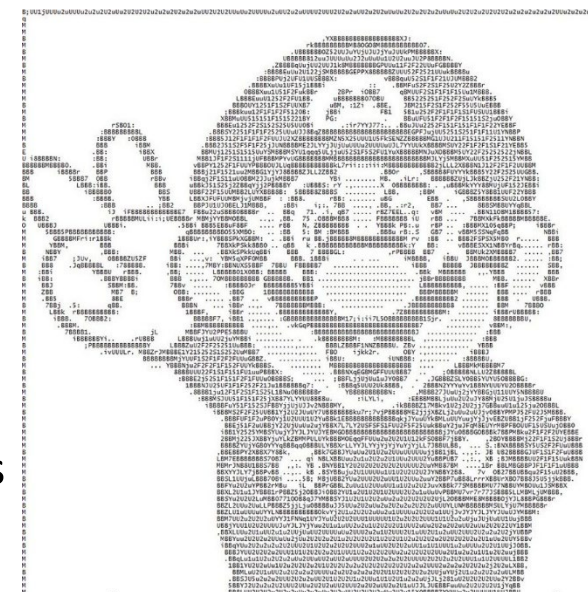
- Float
 - C i C++: single-precision (32-bit)
 - Python: built-in double-precision (64 bit)
- La majoria d'arquitectures 32-bit tenen suport per 64-bit FPU (floating-point unit)
 - IA-32 i x86-64 tenen 80-bit floating-point (double-extended precision format)
 - Des de 1989: x87 FPU (80-bit)
- Quad-precision (128-bit)
 - Suport software
 - Poques architectures ofereixen el suport hardware necessari
 - E.g. Processador IBM POWER9 (MareNostrum 4)

Números “Floating-point”: problemes de precisió

- Nombres que no poden ser representats de manera exacte com fraccions binaries
 - E.g. $1/10$
 - 0.000110011001100110011001100110011001100110011001100110011001...
- Conversions a enter poden perdre precisió durant el procés de truncat i arrodoniment
 - E.g. $56 / 7 = 8$; $0,56 / 0,07 = \text{pot ser } 7$
 - E.g. explosió del cohet Ariane 5 (1996)
 - <http://www-users.math.umn.edu/~arnold/disasters/ariane.html>
- Propietat conmutativa SI, però associativa i distributiva NO necessàriament
 - “ $(a + b) + c$ ” podria NO ser igual a “ $a + (b + c)$ ”
 - “ $(a + b) * c$ ” podria NO ser igual a “ $a * c + b * c$ ”

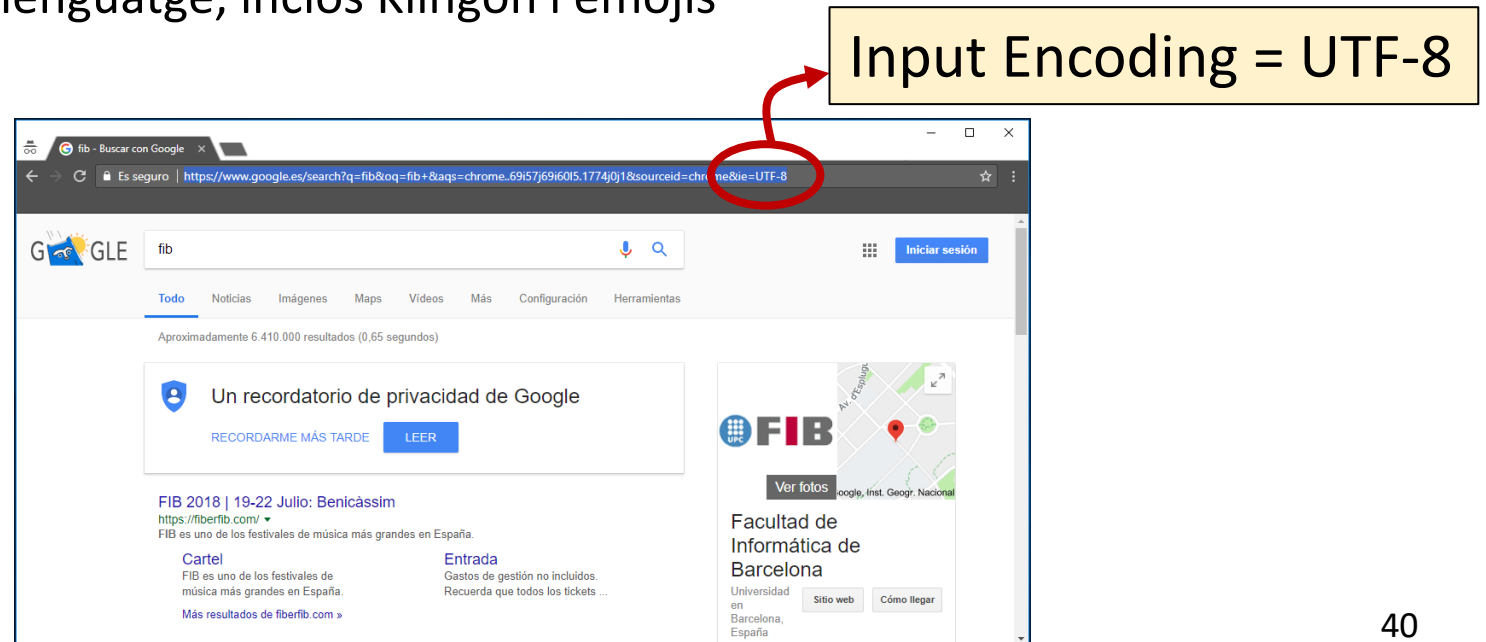
Tipus de Dades Escalar: Símbols i Caràcters

- Char
 - Codifica valors alfanumèrics i símbols
- Diferents codificacions de caràcters
 - Codi ASCII (American Standard Code for Information Interchange)
 - Un Byte utilitzant els últims 7 bits. De 0 a 127
 - És l'estàndard del començament de HTML
 - Codi ISO-8859-1 (Latin Alphabet)
 - 1 Byte (256 caràcters): extensió de ASCII
 - És l'estàndard des de HTML 2.0 a HTML 4.01
 - Problemes amb alguns símbols
 - Codi Windows-1252 (CP-1252): també conegut com ANSI
 - És un super conjunt de ISO-8859-1 (més caràcters representables)
 - Utilitzat per defecte en components "legacy" de Microsoft Windows
 - **UNICODE**



Chars: codificació

- Unicode Standard (creat a finals dels 80s per Xerox i Apple)
 - **UTF-8** (des de 1 Byte fins a 4 bytes, si fos necessari)
 - És la codificació dominant
 - És l'estàndard de HTML5 i pàgines web
 - Per suportar qualsevol llenguatge, inclòs Klingon i emojis
 - Diferents amplades
 - UTF-16 (2-4 Bytes)
 - UTF-32 (4 Bytes)



Chars: encoding issues

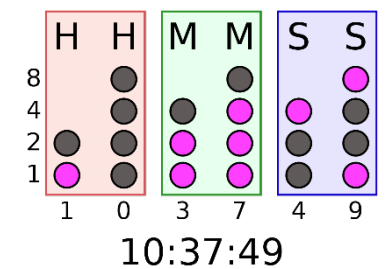
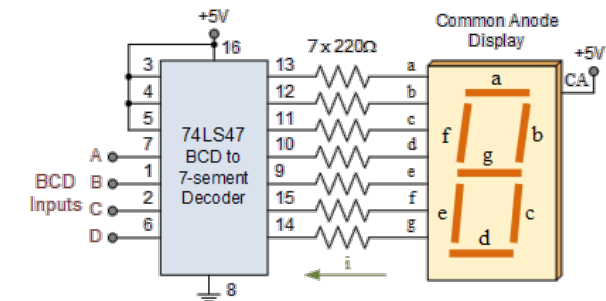
- Un Byte vs varis bytes
 - Caràcters de mida fixa (fixed-size) vs mida variable (variable-size)
 - Necessitat de representar conjunts de caràcters que no poden ser representats en un únic byte (e.g. Japanese, Chinese)
 - MBCSs: MultiByte Character Sets (old and legacy technology)
 - Unicode Standard
- Problemes de compatibilitat
 - Els programes han de ser compatibles amb Unicode per manipular correctament les dades

Chars: Binary-Coded Decimal (BCD)

- Algunes aplicacions necessiten una precisió exacte
 - Dígits (0..9) són codificats amb 4 bits (1 nibble)
 - Les combinacions per sobre de 1001 (9) no es usen; poden ser utilitzats per codificar +/- o el punt decimal (.)
 - Versió sense empaquetar: 1 byte → 1 dígit BCD
 - 1 dels 2 nibbles no s'utilitzen
 - Versió empaquetada: 1 byte → 2 dígits BCD
- Moltes altres possible condicions
- Suport en IBM System/360 i (limitat) en Intel CPUs
- Suport natiu en Ada, Cobol i PL/I
- Suport en C/C++ i Python mitjançant llibreries

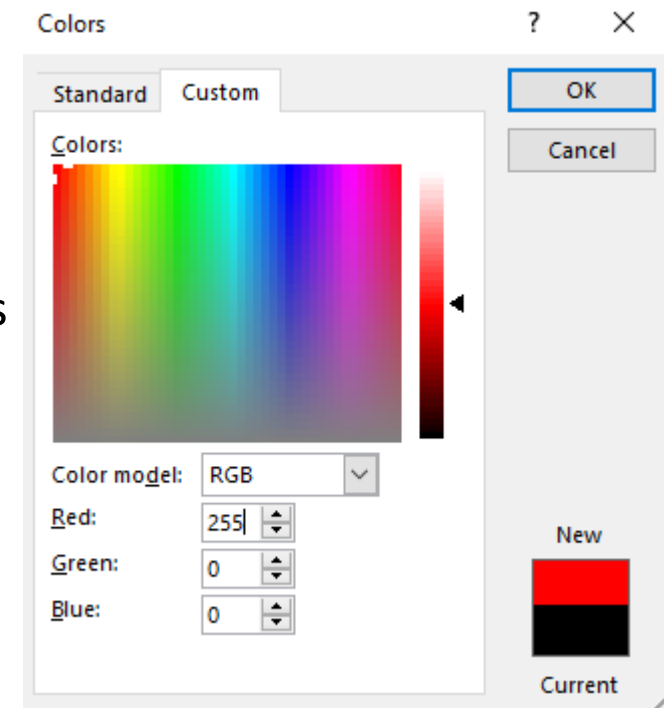
https://en.wikipedia.org/wiki/Binary-coded_decimal

Decimal digit	BCD encoding
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1



Mostrar Informació

- Modes de mostrar informació a pantalla
 - Text
 - Caràcters i atributs es representen automàticament
 - Diferent nombre de files i columnes (e.g. 80x24)
 - Gràfic
 - Resolució en pixels (FullHD 1920x1080)
 - Pixel (Picture Element): codificació de color, normalment... 24 bits
 - Red (R): 8 bits
 - Green (G): 8 bits
 - Blue (B): 8 bits

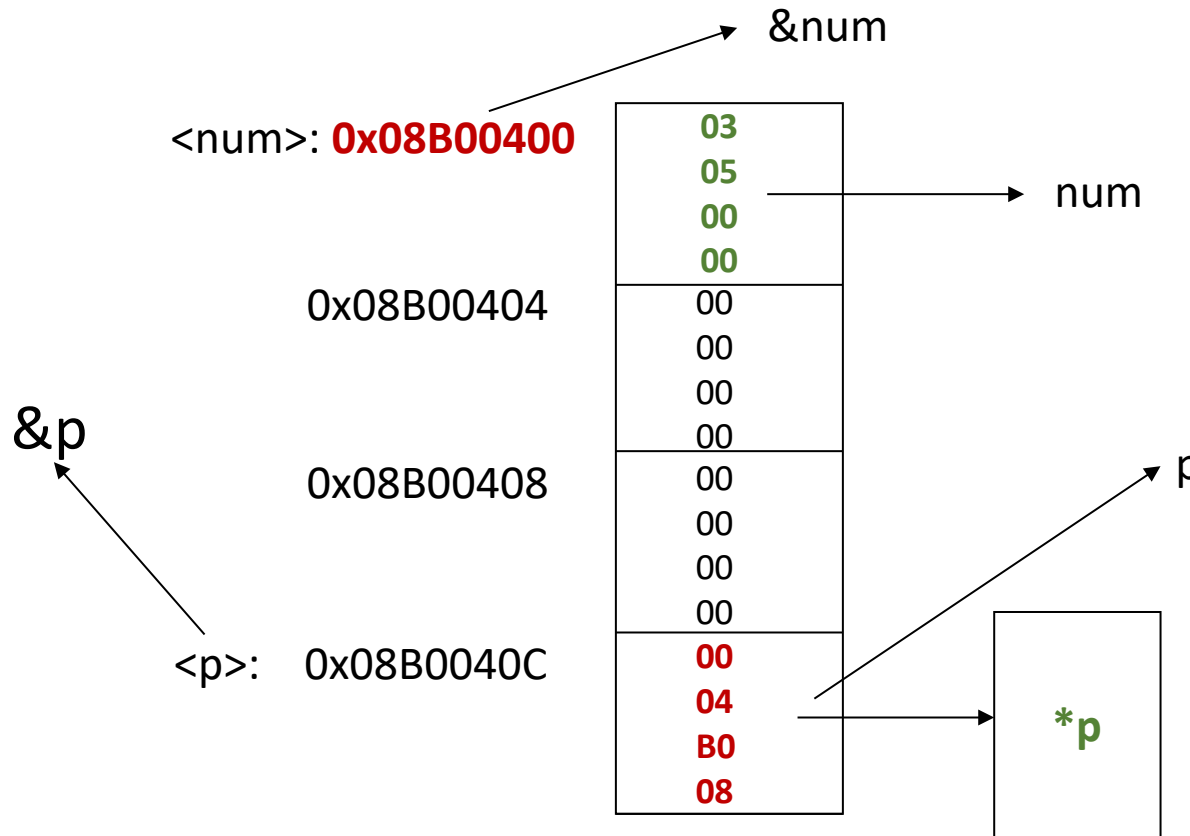


Tipus de Dades Escalar: enumeration i boolean

- Enumeration:
 - Llista ordenada de noms simbòlics assignats a valors únics
 - E.g.: colours {red, green, blue}
 - Tamany d'un enter, per defecte
- Boolean:
 - True o false
 - Built-in data type en C++ i Python
 - Però NO en C (valor enter: 0 vs 1)
 - Encara que només necessita 1bit, agafa 1 Byte
 - Ha de ser adreçable

Tipus de Dades Escalar: Pointer

- Valor que fa referència a un altre valor guardat en algun altre lloc (**address == pointer**)



```
int num = 0x0503; //1283
```

```
int *p;
```

```
p = &num;
```

```
num = 0x00000503
```

```
&num = 0x08B00400
```

```
p = 0x08B00400
```

```
&p = 0x08B0040C
```

```
*p = 0x00000503
```

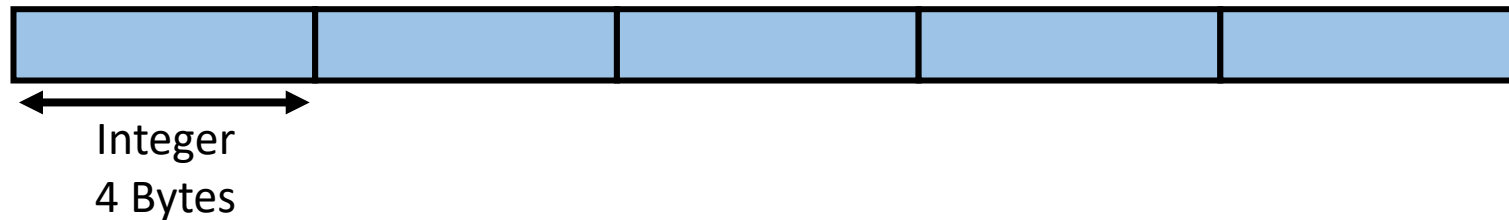
Tipus de Dades Especial: void i void*

- El tipus de dades “void” és una paraula especial que fa referencia a un lloc que guarda un tipus de dades que no representa dades!!!!
- Diferents significats
 - Void function (en C/C++): la funció no retorna res
 - Void parameter (en C/C++): la funció no necessita paràmetres
- **Void *** es diferent...
 - És un punter que apunta a un tipus de dades no especificat, és a dir, qualsevol!!!!
 - És molt útil, però cuidado en el seu ús...

Tipus de Dades Agregades: arrays

- Una col·lecció d'elements que poden ser seleccionats/accedits utilitzant clau/s d'identificació

- E.g.: Una col·lecció de 5 enters en C/C++: `int array[5];`



- Complexitats de la implementació
 - Elements de (mateix vs diferent) tipus/mida de dades
 - Claus d'indexació: valors enters vs arbitraris
 - Mida de l'array estàtica vs dinàmica
 - Dimensions: una vs múltiples
- Arrays s'utilitzen per implementar Tensors en IA i ML

Arrays: compromisos

- Disposició en memòria de arrays multidimensionals
 - Per-fila vs per-columna vs profunditat (per 3D) vs...

A	B	C
D	E	F
G	H	I

A	B	C	D	E	F	G	H	I
---	---	---	---	---	---	---	---	---

Per Fila

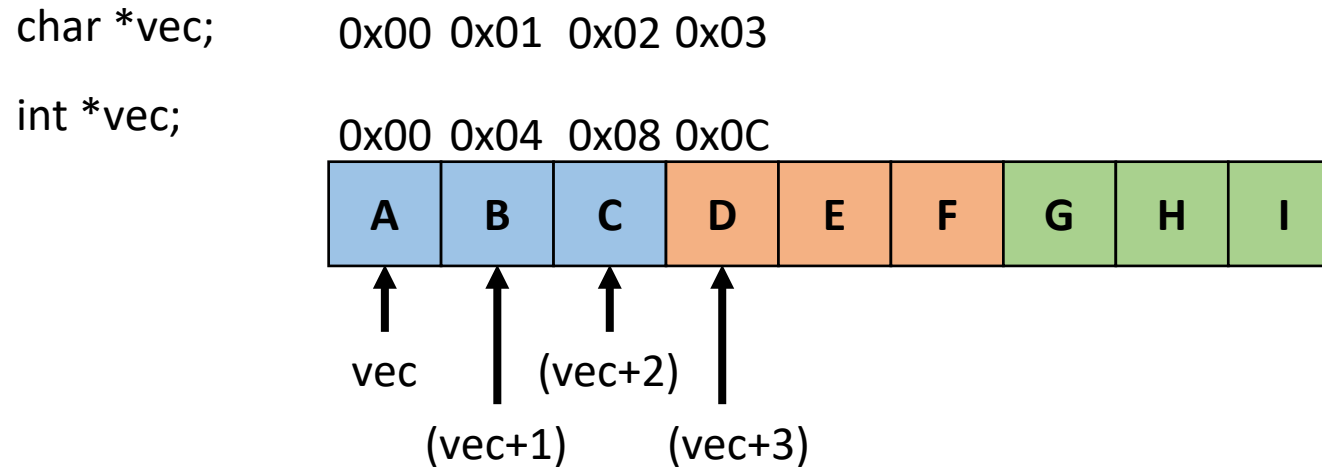
A	D	G	B	E	H	C	F	I
---	---	---	---	---	---	---	---	---

Per Columna

- Segons el llenguatge de programació
 - Per fila: C, C++, Python
 - Per columna: Fortran, MatLab, R
- Suport Hardware per potenciar el rendiment
 - Registres especials

Aritmètica de Punters

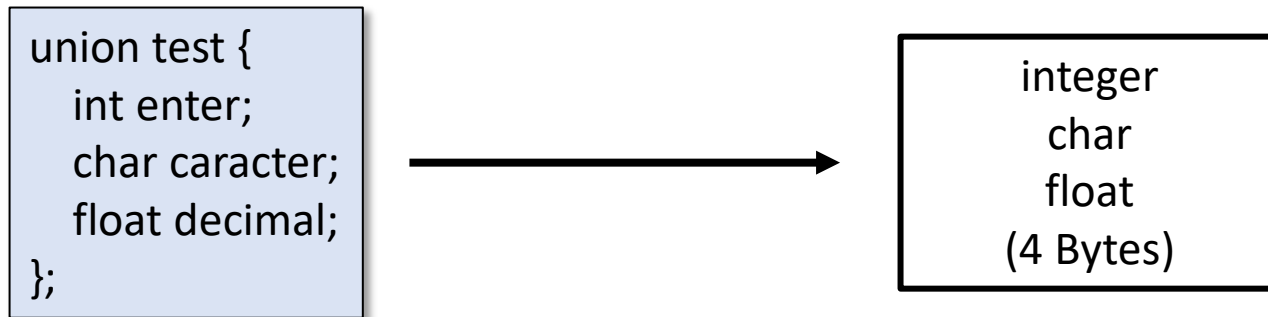
- Els punters poden apuntar @s de memòria posteriors en funció del tipus de dades que apunten
 - E.g. diferència entre **char*** (1@ de memòria) i **int*** (4@ de memòria)



- Els punters apunten a una @ de memòria, independentment de més detalls
 - Veurem més sobre @s de memòria en temes posteriors...

Tipus de Dades Agregades: unions

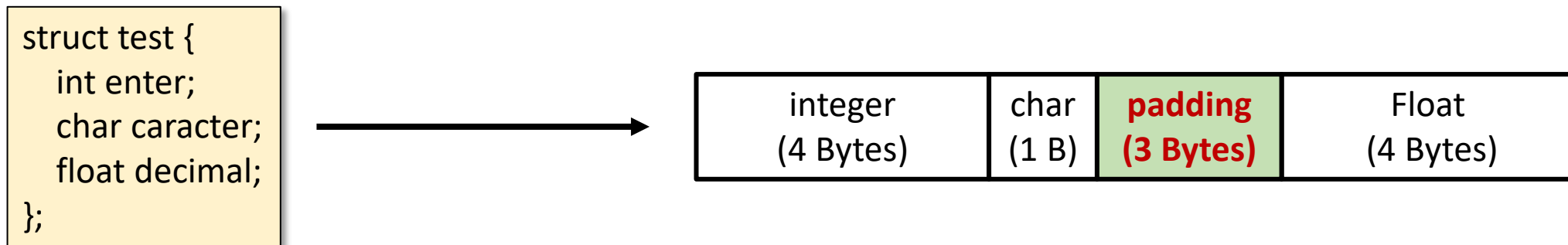
- Diversos valors de diferents tipus que poden ser accedits en la mateixa @ de memòria
 - Només té un valor a la vegada
 - Els valors es sobreescrueixen



- És una manera eficient de utilitzar la mateixa @ de memòria per diferents propòsits

Tipus de Dades Agregades: structs

- És un registre que agrupa diverses variables i les col·loca en un bloc de memòria, accedit per un únic punter o nom de variable



- Padding bytes serveix per alinear camps en memòria i optimitzar rendiment (veurem més detalls en temes posteriors)
 - Relacionat amb l'arquitectura del processador (32 bits vs 64 bits)

Tipus de Dades Agregades: relacionat amb OOP

- Object Oriented Programming (OOP): paradigma de programació
 - Un objecte és una instància d'una classe: és una combinació de variables, funcions i structs
 - E.g.: C++ and Python
- Classes **són una evolució dels structs**
 - Continguts (fields) amb accés restringit
- String és diferent que un array de chars
 - Tipus de dades basat en classe (C++)
 - Amb funcions i camps integrats
 - **Seqüència de caràcters finalitzat amb '\0'** (C/C++)
 - Un array de chars és bàsicament un array de chars!!! (no està el '\0' en el final)

Bibliografia

- Computer Systems – A Programmer's perspective (3rd Edition)
 - Randal E. Bryant, David R. O'Hallaron, Person Education Limited, 2016
 - https://discovery.upc.edu/permalink/34CSUC_UPC/11q3oqt/alma991004062589706711
 - Chapter 2: Representing and manipulating information
- Data Type Summary
 - MSDN (Microsoft)
 - <https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/data-type-summary>
 - Summary list of data types and their respective size in Windows based software
- Computer Organization and Design (5th Edition)
 - D. Patterson, J. Hennessy, and P. Alexander
 - http://cataleg.upc.edu/record=b1431482~S1*cat
 - Several chapters introduce different types of data
- C++ Language Tutorial
 - <http://www.cplusplus.com/doc/tutorial/>
 - Summary of basic and compound data types for C++