Tipi di oggetti

- Testi
  - Testi semplici (Blocco Note)
  - Testi strutturati (Word)
    - Editabili
    - Non editabili
- Immagini
- Suoni
- Video ...

Informatica di base

Rappresentazione dei dati

Formato

- Il termine **formato** indica un modo di codificare un certo tipo di oggetto (p.es. un immagine) come una sequenza di byte
- Lo stesso tipo di oggetto può essere codificato in molti modi diversi

Rappresentazione di oggetti

- Ogni programma manipola oggetti (diversi a seconda del programma)
- Questi oggetti vengono tenuti in memoria centrale, modificati, e periodicamente (di solito, su richiesta dell'utente) salvati su un supporto magnetico (HD, floppy...).
- Poiché sia la memoria centrale che i supporti sono una sequenza di byte (=8 bit), ogni oggetto va codificato sottoforma di sequenza di byte (=ovvero sequenza di bit)

Scrittura

- Se si usa File/Salva con nome... è possibile salvare il documento in un formato diverso da quello nativo
- In particolare, se il documento è stato caricato da un file in formato non nativo, quando si salva (File/Salva) il documento verrà salvato nello stesso formato in cui era stato caricato

Un programma...

- ...mantiene i dati in memoria e li salva su disco in un certo formato (detto *formato nativo* per quel programma)
- Spesso un programma è in grado di leggere/scrivere file scritti in altri formati

Scrittura

- Alcuni programmi hanno una voce di menù speciale (File/Esporta) per salvare in altri formati
- Anche in questo caso, il salvataggio in un formato diverso comporta una conversione (in memoria, il documento viene comunque tenuto in formato nativo) non sempre senza conseguenze

Lettura

- Di solito, se da un programma (p.es. Word 2000) si tenta di leggere (menù File/Apri) un file scritto in un formato diverso, lui lo converte internamente nel suo formato nativo
- A volte, c'è una voce di menù specifica (File/Importa) che permette di importare altri formati
- La conversione non è sempre "indolore"...

Rappresentazione di testi (semplici)

- Un testo semplice è una sequenza di caratteri
- Un carattere può essere un simbolo della tastiera (p.es. lettere, cifre, simboli speciali) compresi anche i "caratteri invisibili" (spazi, a-capo...)

I formati possono essere...

- Ad-hoc
- Proprietari, ma variamente interoperabili
- Non proprietari e non standardizzati, ma variamente interoperabili
- Non proprietari, standardizzati

Idea

- Ogni byte può assumere 256 diverse configurazioni: 00000000, 00000001, 00000010, ... 11111111
- Associamo a ogni configurazione un carattere
- Una sequenza di N caratteri sarà rappresentata da altrettanti byte

Con l'avvento di Internet...

- ...è preferibile usare sempre formati non proprietari e standardizzati
- In questa lezione, ci occuperemo solo della rappresentazione di: *sequenze di caratteri* (testi semplici) e di *numeri* (interi, interi con segno, non interi)
- Non prenderemo in considerazione altri oggetti (immagini, suoni, video ecc.)!

Codice ASCII

- Nel codice ASCII, ad esempio:
  - A = 0100 0001
  - c = 0110 0011
  - (= 0011 0001
  - Spazio = 0010 0000
- Nel codice ASCII non ci sono le lettere accentate, che sono state aggiunte nell'ISO-8859-1

Codici per la rappresentazione di caratteri

- Esistono molti modi di associare a ciascun carattere una possibile configurazione di 8 bit
- Ognuno di questi modi si chiama *codice per la rappresentazione di caratteri*
- I più noti sono i codici EBCDC e ASCII

Il codice ASCII

- È uno dei formati più interoperabili in assoluto
- È standardizzato
- È leggibile da tutti i programmi che manipolano testi (strutturati e non)

Codice ASCII

- Il codice ASCII in realtà usa solo 128 delle 256 configurazioni possibili (usa solo quelle che iniziano per 0...)
- Ogni singolo Paese ha esteso questo codice aggiungendo 128 caratteri a seconda delle esigenze locali
- Nell'Europa occidentale si usa l'ISO-8859-1

Unicode

- 4 byte = 4x8 bit = 32 bit
- Con 32 bit si hanno a disposizione 2^{32}=circa 4.300.000.000 di caratteri!
- Ovviamente si spreca spazio (un testo scritto con caratteri “normali” occupa il quadruplo)
- UTF-8: un sistema di memorizzazione di Unicode con numero variabile di byte

Cosa succede, però...

- ...se salvo un file di testo qui e tento di leggerlo in Polonia?
- I caratteri ASCII (i primi 128) sono gli stessi, ma gli altri sono diversi (in Polonia usano l'ISO-8859-2)
- Ad es., può darsi che la **à** diventi ad esempio una **Ł**

Rappresentazione di numeri

- Un'esigenza frequente è quella di rappresentare numeri
- Cominciamo dai numeri interi non negativi

Internazionalizzazione (i18n)

- Internet permette di scambiare file – occorre che i formati siano altamente interoperabili
- Occorre estendere l'ASCII in modo da comprendere tutti i caratteri possibili
- Per questo è nato Unicode, che usa 4 byte per rappresentare un carattere

Notazioni posizionali

- Quando si esprime un numero x in una notazione posizionale, si rappresenta x come una sequenza di simboli, detti *cifre*
- Ogni simbolo ha un significato diverso a seconda della *posizione* in cui compare (ecco perché "posizionale")
- Ad esempio, i numeri 312 e 213 sono due numeri *diversi*

Rappresentazione di numeri

- Un numero intero può essere rappresentato (nella vita quotidiana) in molti modi:
  - Notazione posizionale (quella che usiamo abitualmente)
  - Notazione additiva/sottrattiva (numeri romani)
  - Notazione unaria (si impara alle elementari)

Notazioni posizionali in base B

- In particolare, ogni notazione posizionale è caratterizzata da una *base* B
- La rappresentazione ordinaria è la *notazione posizionale in base 10* (anche detta semplicemente "decimale")

Esempio

- Il numero 14:
  - In notazione romana: DIV
  - In notazione unaria: IIIIIIIIIIIIII
- A noi interessano le notazioni posizionali
- La nostra notazione posizionale non è l'unica possibile!

In base 10 questo sembra ovvio...

- Il numero 304 corrisponde a:

$3 \times 10^2 + 0 \times 10^1 + 4 \times 10^0 =$

$3 \times 100 + 4 \times 1 = 304$

Notazioni posizionali in base B

- In base B si hanno B cifre (ciascuna delle quali rappresenta un numero da 0 a B-1)
- Per esempio, in base 10 ci sono dieci cifre (0, 1, ..., 9)
- Un numero si scrive come una sequenza di cifre

Base 2 (notazione binaria)

- Quando la base è 2 ci sono solo 2 cifre: 0 e 1
- La base 2 è la base "naturale" del computer: ogni sequenza di bit si può interpretare come un numero in base 2

Notazioni posizionali in base B

- Ogni posizione rappresenta una potenza di B: la posizione più a destra è B^0, la penultima è B^1 ecc.

B^5	B^4	B^3	B^2	B^1	B^0

- Il valore di ogni cifra va moltiplicato per il valore della posizione in cui compare

Base 2

$101100_2 = ?$

Base 2

$110101_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 4 + 1$

$= 53$

Base 2

$101100_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

$= 32 + 8 + 4$

$= 44$

Base 2

$11111111_2 = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= 255$

Massimo valore assumibile da un byte
(il minimo è zero = 00000000)

Base 8

$100_8 = 1X8^2+0X8^1+0X8^0$

= 1 x 64

= 64

Base 8 (notazione ottale)

- Quando la base è 8 ci sono 8 cifre: 0, 1, 2, ..., 7
- La base ottale è stata molto usata, soprattutto in passato, perché è molto comoda e si traduce facilmente in base 2

Base 8

$64_8 = ?$

Base 8

$705_8 = 7X8^2+0X8^1+5X8^0$

= 7 x 64 + 5 x 1

= 453

Base 16

$AF0_{16} = 10 \times 16^2 + 15 \times 16^1 + 0 \times 16^0$

= 10 x 256 + 15 x 16

= 2800

Base 8

$64_8 = 6 \times 8^1 + 4 \times 8^0$

= 6 x 8 + 4 x 1

= 52

Base 16

$100_{16} = 1 \times 16^2 + 0 \times 16^1 + 0 \times 16^0$

= 1 x 256

= 256

Base 16 (notazione esadecimale)

- Quando la base è 16 ci sono 16 cifre: 0, 1, 2, ..., 9, A (=10), B (=11), ..., F (=15)
- La base esadecimale è molto usata in informatica

Come si trasforma un numero in una data base?

- Per esprimere il numero x in base B:
  - Dividete x/B: scrivete il **resto** della divisione come cifra in base B
  - Prendete il **risultato** e ripetete l'operazione di cui sopra finché arrivate a 0
  - Leggete le cifre *al contrario*

Base 16

$F08_{16}$ = ?

Esempio: trasformazione in base 2

364 : 2 = 182 resto 0
182 : 2 = 91 resto 0
91 : 2 = 45 resto 1
45 : 2 = 22 resto 1
22 : 2 = 11 resto 0
11 : 2 = 5 resto 1
5 : 2 = 2 resto 1
2 : 2 = 1 resto 0
1 : 2 = 0 resto 1

Quindi $364 = 101101100_2$

Base 16

$F08_{16} = 15 \times 16^2 + 0 \times 16^1 + 8 \times 16^0$

$= 15 \times 256 + 8 \times 1$

$= 3848$

Esempio: trasformazione in base 2

520 : 2 = 260 resto 0
260 : 2 = 130 resto 0
130 : 2 = 65 resto 0
65 : 2 = 32 resto 1
32 : 2 = 16 resto 0
16 : 2 = 8 resto 0
8 : 2 = 4 resto 0
4 : 2 = 2 resto 0
2 : 2 = 1 resto 0
1 : 2 = 0 resto 1

Quindi $520 = 1000001000_2$

Esempio: trasformazione in base 2

179 : 2 = 89 resto 1
89 : 2 = 44 resto 1
44 : 2 = 22 resto 0
22 : 2 = 11 resto 0
11 : 2 = 5 resto 1
5 : 2 = 2 resto 1
2 : 2 = 1 resto 0
1 : 2 = 0 resto 1

Quindi $179 = 10110011_2$

Esempio: trasformazione in base 16

364 : 16 = 22 resto 12 = C
22 : 16 = 1 resto 6 = 6
1 : 16 = 0 resto 1 = 1

Quindi $364 = 16C_{16}$

Esempio: trasformazione in base 2

Scrivete il numero 520 in base 2

Esempio: trasformazione in base 16

5200 : 16 =	325	resto	0	=	0
325 : 16 =	20	resto	5	=	5
20 : 16 =	1	resto	4	=	4
1 : 16 =	0	resto	1	=	1

Quindi $5200 = 1450_{16}$

Esempio: trasformazione in base 16

1790 : 16 =	111	resto	14	=	E
111 : 16 =	6	resto	15	=	F
6 : 16 =	0	resto	6	=	6

Quindi $1790 = 6FE_{16}$

In un computer...

- ...i numeri non negativi si rappresentano semplicemente in base 2
- Usando un byte si possono rappresentare numeri da 0 a 255
- Se si vogliono rappresentare numeri più grandi, occorre usare più di un byte!

Esempio: trasformazione in base 16

Scrivete il numero 5200 in base 16

Numeri negativi

- Per rappresentare un x>=0, si lascia a 0 il bit di segno, e si rappresenta x come al solito
- Par rappresentare un x<0, si calcola |x|-1, lo si rappresenta, e si *invertono tutti i bit* (0->1, 1->0); il bit di segno viene messo a 1

Ad esempio

- Con 2 byte (=16 bit) si possono rappresentare numeri da 0 fino a 1111111111111111_2=65535
- Con 4 byte (=32 bit) si possono rappresentare numeri da 0 fino a oltre 4 miliardi

Ad esempio...

- Il numero -55 (rappresentato in un byte):
  - |-55|-1=54
  - 54 in binario (in 7 bit) è 0110110
  - Invertendo i bit si ottiene 1001001
  - Quindi -55 si rappresenta come

    1 1001001

Numeri negativi

- Come si possono rappresentare numeri interi *con segno*?
- Si usa il primo bit per indicare il segno (0=+, 1=-) e i restanti bit per indicare il numero
- La regola esatta è un po' più complicata...

Rappresentazione di numeri decimali

- La notazione posizionale si può estendere per rappresentare numeri non interi
- Le cifre dopo il separatore (.) rappresentano *potenze negative di **B***

B^2	B^1	B^0	.	B^{-1}	B^{-2}	B^{-3}

- Ricordate che B^{-n} significa $1/B^n$

Numeri rappresentabili

- In un byte si rappresentano numeri positivi da 00000000 (0) fino a 01111111 (127), e numeri negativi da 10000000 (-128) fino a 11111111 (-1)
- Cioè, complessivamente, in un byte si rappresentano numeri da -128 a +127

Ad esempio

- In base 10, il numero 20.1743 significa:

$$2\times10^{1}+0\times10^{0}+1\times10^{-1}+7\times10^{-2}+4\times10^{-3}+3\times10^{-4}$$

$$= 2 \times 10 + 1 \times 0.1 + 7 \times 0.01 + 4 \times 0.001 + 3 \times 0.00001$$

Se si vogliono rappresentare...

- ...numeri più grandi si useranno più byte
- Ad esempio, con 2 byte si rappresentano numeri da -32768 a +32767 (verificatelo!)
- *Nota*: uno deve sapere se un(a sequenza di) byte contiene un carattere (ASCII), un numero positivo o un numero con segno per poterne interpretare il contenuto...

Rappresentazione floating-point

- Per rappresentare i numeri non interi si usa la cosiddetta rappresentazione *floating-point* (=a virgola mobile):

$$\pm d_1 . d_2 d_3 \ldots d_n \times B^{esponente}$$

- Quindi: un bit di segno, n cifre di *mantissa* (il punto si considera posto dopo la prima) e un po' di cifre per l'esponente (che avrà un segno)

In base 2

- Trasforma 10.01001_2 in base 10

$10.01001_2 = 1X2^1+0X2^0+0X2^{-1}+1X2^{-2}+0X2^{-3}+0X2^{-4}+1X2^{-5}$

$= 2 + 1 / 4 + 1 / 32$

$= 2 + 0.25 + 0.03125$

$= 2.28125$

Ad esempio

- Il numero 33.7, espresso in binario, fa 100001.10110011001100...
- Cioè $1.00001101100110011001100... \times 2^5$
- Se uso due byte per la mantissa e uno per l'esponente:

0 1000011 01100110 00000101

mantissa (con segno) esponente

In base 2

- Come si rappresenta 4.2 in base 2?
- 10.0011001100110011001100...
- Cambiando di base è possibile che un numero con rappresentazione finita diventi *periodico* (come in questo caso)!

Nella realtà...

- I computer usano (di solito):
  - *Precisione semplice:*
    - 1 bit di segno + 23 bit di mantissa + 8 bit di esponente
  - Precisione doppia:
    - 1 bit di segno + 52 bit di mantissa + 11 bit di esponente
- Rappresentazione standard IEEE 754

Notate che...

- Nell'esempio precedente, il numero veramente espresso è $1.0000110110011 \times 2^5$ cioè $1.053100586... \times 32$
- Che fa 33.69921875... e non 33.7 (il valore da rappresentare)
- La rappresentazione floating-point è *approssimata*

Nella realtà...

- I computer usano (di solito):
  - *Precisione semplice:*
    - 1 bit di segno + 23 bit di mantissa + 8 bit di esponente
  - Precisione doppia:
    - 1 bit di segno + 52 bit di mantissa + 11 bit di esponente
- Rappresentazione standard IEEE 754

AA 2002/03
© Boldi
Informatica di base
Rappresentazione dei dati