Università degli Studi di Milano «La Statale»
Dipartimento di Informatica

La rappresentazione di testi: caratteri e stringhe

(parte B)

14

ASCII esteso

- Osservazione: un byte sono 8 bits,
  ma la codifica di un carattere ASCII ne richiede solo 7 !
- Idea nefasta, ma percorsa molte volte nei decenni pre 2000:
  - Utilizziare l'ottavo bit, ed estendere la tabella ASCII
    con 128 nuove combinazioni
  - Ottenere così 128 nuovi codici per altrettanti caratteri
    (dal numero 128 a 255)
  - Bene, ma problema: QUALI CARATTERI INTRODURRE?
- Come abbiamo visto, il codice ASCII è deficitario di moltissime cose
- Parti diverse del mondo necessitano di caratteri diversi
  - 128 non bastano certo per tutti
- Risultato: estensioni diverse su macchine diverse (di paesi diversi) ☹ ☹
  - Es: in Europa: molto popolare ISO-Latin1, con i vari ò,à,è,é…
    Es: in Russia: CP-866 che introduce i vari ц , ч , ш , ъ , ы , ….
  - Molti standard in conflitto = molti problemi

15

Esempio: ISO 8859/1 (anche detto ISO Latin-1)

- ancora appena utilizzato in poche pagine WEB, ma in declino
- Contiene nei codici 128-255 vari caratteri accentati e lettere usate dai linguaggi dell'Europa occidentale (Italiano, Francese, Spagnolo, Tedesco, Danese, ecc.):
  ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ
- almeno, è compatibile con l'ASCII, nel senso che i primi 128 codici hanno identico significato nei due standard.
  - questo vale per tutti gli altri codici ASCII «estesi»
  - alcuni, ridefiniscono anche alcuni dei caratteri «non stampabili» dell'ASCII

Architettura degli elaboratori - 16 - Aritmetica binaria

16

Unicode

- Risolve *una volta per tutte* il problema del mapping fra numeri e caratteri
- E' uno standard universale gestito da un apposito consorzio internazionale
- Ad ogni numero naturale si fa corrispondere un carattere
- In UNICODE, i primi 128 caratteri (naturali da 0 a 127) coincidono con la tabella ASCII
  - Quindi, «l'UNICODE estende l'ASCII»
- Adozione universale:
  Unicode è oggi utilizzato nella stragrande maggioranza dei testi, degli URL, dei filename nei file system, sorgenti dei linguaggi di programmazione, pagine Web, …

Architettura degli elaboratori - 18 - Aritmetica binaria

18

Unicode

- E' in perpetua evoluzione.
- Alcune tappe (partendo dall'ASCII):
  - nel 1991 (ver. 1.0.1) viene aggiunto Hiragana, Greco, Copto…
  - nel 1998 (ver 2.0) viene aggiunto il simbolo dell'euro
  - nel 2009 (ver 5.2) vengono aggiunti i geroglifici e 
  - nel 2014 (ver 8.0) vengono aggiunti il Siddhaṃ (   ) e altro
  - Nel 2019 (ver 12.1): un car. Giapponese di interesse storico

- Attualmente, dopo una trentina di revisioni, sono previsti circa 160 mila caratteri di tutte le lingue del mondo vive e morte (e simboli matematici, fisici, chimici… e tutti i *kanji* cinesi/giapponesi/coreani, ed emoji… e i pezzi degli scacchi… e i segni zodiacali… in breve, tutto quello che può venire in mente di scrivere)

Unicode: ultimo update

- Per esempio, la Versione 17.0 risale al mese scorso (Settembre 2025)
- Ha aggiunto:
  - Due o tre alfabeti usati da alcuni linguaggi molto locali
  - Il simbolo della moneta dell'Arabia Saudita (⃁)
  - E questi emoji:

Unicode e UTF

- L'unicode si occupa solo di *quale numero corrisponda a quale carattere*.
  - Concettualmente, è molto semplice!
- *Non* si occupa invece di come codificare (come una sequenza di bit) i numeri che rappresentano questi caratteri.
- Dunque*, come rappresentiamo un carattere UNICODE?*
  - cioè un numero naturale da 0 a (per ora) quasi 160,000?
  - Considerazione: per le (quasi) 160.000 combinazioni attuali servirebbero almeno 18 bits!
- Le possibili risposte a questa domanda sono detti «UTF»
  Unicode Transformation Format
- *risposta 1*: usiamo un intero senza segno a 32 bit («UTF32»).
  semplice e completo, ma molto oneroso: ben 4 byte per carattere
- *risposta 2*: usiamo un intero senza segno a 16 bit («UTF16»)
  semplice, e solo 2 byte per carattere, ma rinunciamo ad esprimere più metà dei caratteri UNICODE possibili (quanti ne esprimiamo?)

21

Unicode e UTF-8

- Risposta migliore: UTF8
  - utilizziamo solo un byte per i primi 128 caratteri (0xxxxxxx)
  - se il carattere codificato è > 127, allora il bytes comincia con un 1, e usiamo un numero variabile di byte addizionali (1,2 o 3)
    - …secondo alcune regole specificate, di seguito
  - risultato:
    maggiore è il numero (e quindi, grossomodo, più raro è il carattere), e maggiore il numero di byte necessario per codificarlo (ideale!)
  - esempio:
    per un semplice carattere ASCII, es la lettera «t»: 1 byte (olè)
    per una lettera greca, o una a accentata: 2 bytes.
    per uno delle migliaia di kanji (ideogrammi cinesi): 3 byte.
    per un geroglifico egizio: 4 bytes (e pazienza)
- UTF-8 è quindi molto conveniente.
  - inoltre, un testo in ASCII è ANCHE un testo in UTF-8 (perché?)
- UTF-8 è il più utilizzato formato attuale per i testi! (es in rete)

22

Codifica «UTF-8»

- Codifica per caratteri Unicode che richiedono un numero di bits…

	1° byte	2° byte	3° byte	4° byte
fino a 7	0XXXXXXX			
fino a 11 = 5+6	110XXXXX	10XXXXXX		
fino a 16 = 4+6+6	1110XXXX	10XXXXXX	10XXXXXX	
fino a 21 = 3+6+6+6	11110XXX	10XXXXXX	10XXXXXX	10XXXXXX

(Nota: non ti è certo richiesto imparare a memoria questo schema, ma solo di capirlo)

Unicode e fonts (una nota)

- L'unicode si occupa solo di *quale numero corrisponda a quale carattere, in astratto*.
- Non si occupa di come i caratteri vadano resi a schermo (o in stampa)
- Di questo si occupa il **font**
- Un font = la descrizione di disegno con cui raffigurare (a schermo, su una stampante, in un documento…) ciascuno dei carattere UNICODE
  - o, quasi sempre, solo un loro sottoinsieme: praticamente nessun font prescrive l'aspetto di ciascuno dei caratteri UNICODE – i caratteri non previsti vengono resi con: □
- nota: font diversi possono prevedere disegni molto diversi per uno stesso carattere UNICODE
  - per es: confronta 4 (font: «arial») e 4 (Font: «baguet»)
- in questo corso non ci occupiamo di come rappresentare internamente un font
  - si tocca però l'argomento nel corso di **Computer Graphics**

Esercizi di riepilogo (su tutta la parte di rappresentazione)

- Che differenza c'è fra:
  - il numero intero **8**,
  - il numero con virgola **8**, e
  - il carattere «**8**»?
- Individua una possibile rappresentazione **su 4 byte** di ciascuna di queste tre cose distinte, e scrivi in esadecimale i 4 byte risultanti, per ciascun caso
  - Come si chiama la seconda e la terza codifica che ha usato?

Interpretazione delle informazioni: una considerazione finale

- In un byte della memoria memorizzo la configurazione di bit 01000001.
- Come fa il calcolatore a sapere se questo byte vada interpretato come un numero intero (65) [verifica], o come un carattere («A») [verifica], o altro?
- Un altro byte vale 10000001. Come fa il calcolatore a «sapere» se rappresenta un valore senza segno (129) [verifica], uno con segno (-127) [verifica], la prima parte di un float a 32 bit, o altro?
- La risposta è semplice: non lo sa per niente.
- Sono i nostri programmi che devono ricordarsi qual sia il significato di una data cella di memoria (o variabile), per usarla di conseguenza.
- I linguaggi ad alto livello facilitano questo compito assegnando un "tipo" alle variabili, per cui (ad esempio) su una variabile di tipo «carattere di testo» non si possono fare prodotti, e su una variabile di tipo numerico non si può chiedere il maiuscolo.
- I linguaggi più a basso livello, come gli assembler, non danno questo tipo di aiuto. Sta al programmatore usare la memoria in modo consistente (se scrivo un numero in virgola mobile, sarebbe un errore leggerlo o processarlo come se fosse un intero)