

CODIFICA DEL TESTO

Simboli da rappresentare:

lettere a,b,c,...z,A,B,C,...Z cifre 0,1,2,3,...9 simboli di punteggiatura ;:.,|

Rappresentati mediante sequenze di bit

Una codifica dei simboli è perciò (almeno) una regola che stabilisce:

- quanti e quali simboli vogliamo rappresentare,
- quanti bit voglio usare per rappresentare ciascuno di quei simboli,
- una mappa tra valore numerico dei bit e simbolo rappresentato da quel valore valore.

Inizialmente solo lingua inglese, pochi simboli, no accenti, no simboli strani.

Solo 128 simboli, bastano sequenze di 7 bit ($2^7=128$)

CODIFICA ASCII (7bit su 8)

Si usa un byte (8bit) tenendo il bit più significativo a zero.

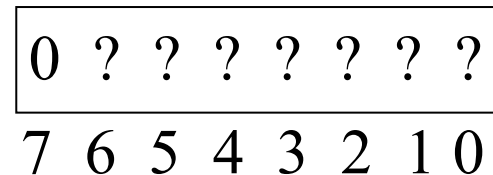
128 simboli, bastano per scrivere in linguaggio C.

caratteri stampabili (valore >32)

lettere (maiuscole e minuscole), cifre, punteggiatura,
apici vari, simboli matematici, parentesi varie,

caratteri di controllo (detti non stampabili) (valore ≤ 32)

escape, space, tab, return (invio), del (canc),



CODIFICA ASCII (7 bit su 8)

Codice ASCII	48	49	50	51	52	53	54	55	56	57
Carattere	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'

codice ASCII	65	66	67	68	69	70	71	ecc..
carattere	'A'	'B'	'C'	'D'	'E'	'F'	'g'	ecc..

Codice ASCII	97	98	99	100	101	102	103	ecc..
Carattere	'a'	'b'	'c'	'd'	'e'	'f'	'g'	ecc..

Codice ASCII	8	9	10	13
Carattere	\b	\t	\n	\r

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
int main(void) { int i; /* guardare che succede per i valori 0, 8, 9, 10 e 13 */  
    for( i=0; i<=127; i++ )  
        printf("%d \t \"%c\"\n", i, (unsigned char)i );  
    return(0);  
}
```

CODIFICHE a 8 BIT - esempio Latin-1 (iso-8859-1)

Come ASCII ma usa anche l'ottavo bit per poter rappresentare in totale 256 simboli.

I simboli con l'ottavo bit posto a zero sono ancora i caratteri ASCII.

I simboli con ottavo bit a 1 sono le aggiunte rispetto alla codifica ASCII

IL PROBLEMA DELLA FINE LINEA (EOL)

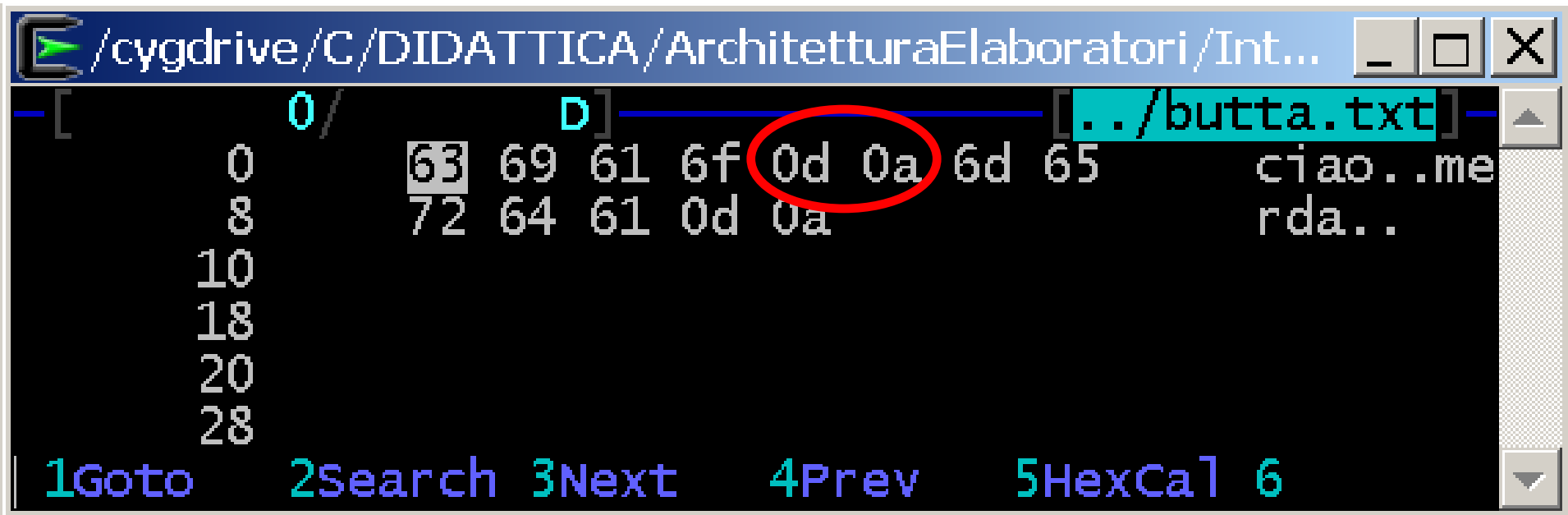
I file di testo usano un carattere speciale (o una sequenza di caratteri) per **identificare la fine di una riga**, ma ogni sistema operativo definisce una propria fine linea (end-of-line EOL).

- Linux: singolo carattere, **newline** (andata a capo), valore 10 (o 0x0A), indicato con `\n`
- Windows: due caratteri assieme, carriage return (ritorno carrello) e newline, valori 13 e 10 (0x0D 0x0A), indicati con `\r\n` in questo ordine.
- Mac: due caratteri assieme, come in Windows, ma in ordine invertito, newline e carriage return, valori 10 e 13 (o 0x0A 0x0D), indicati con `\n\r` in questo ordine.

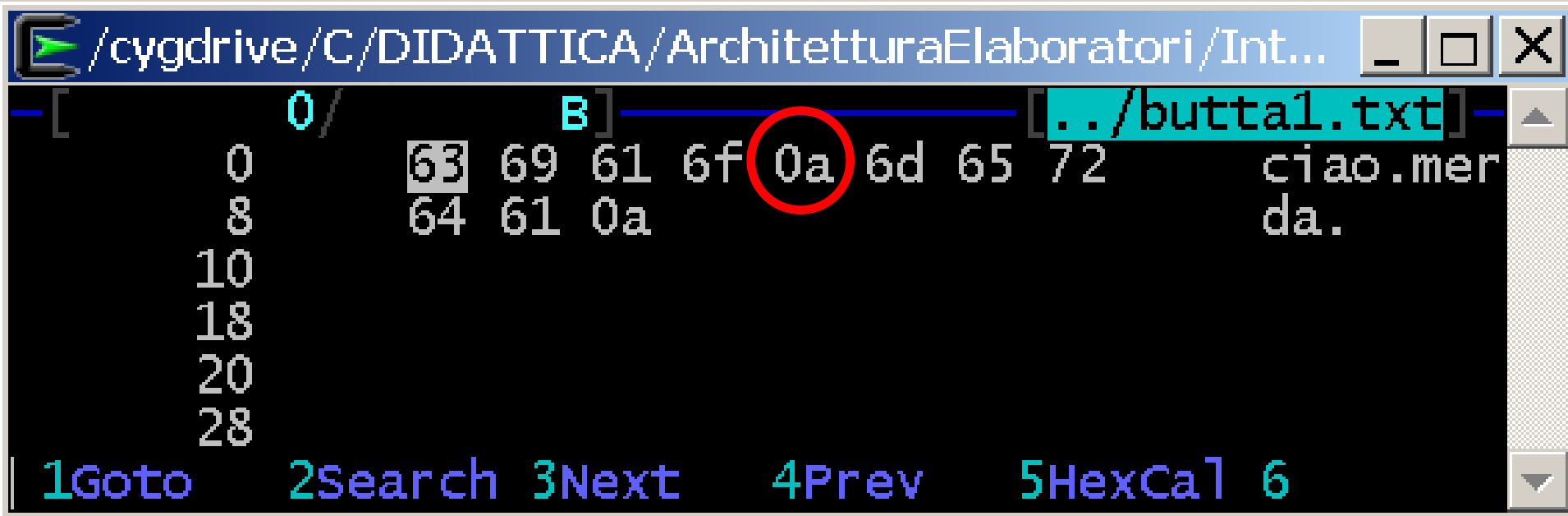
Se scrivo uno script con un editor di Windows o di Mac inserisco delle fine linea con dei `\r`. Se poi faccio eseguire quello script da una bash di un sistema Linux, la bash può considerare il carattere `\r` come un carattere estraneo, non previsto dalle specifiche, e provocare errore.

Usare un editor binario (dhex) per visualizzare anche i caratteri non stampabili !!!

2 files con EOL diversi visti con l'editor esadecimale dhex
Figura sopra generato da Windows, sotto generato da Linux



The screenshot shows the dhex editor interface for a file named `../butta.txt`. The address bar at the top indicates the file path: `/cygdrive/C/DIDATTICA/ArchitetturaElaboratori/Int...`. The main display area shows a hex dump of the file's content. The first line of the hex dump is `63 69 61 6f 0d 0a 6d 65`, which corresponds to the ASCII string `ciao..me`. The second line is `72 64 61 0d 0a`, corresponding to `rda..`. A red circle highlights the `0d 0a` sequence in the first line, indicating a Windows-style carriage return and line feed (CRLF) end-of-line character. The left margin shows addresses 0, 8, 10, 18, 20, and 28. At the bottom, there is a menu with options: `1Goto`, `2Search`, `3Next`, `4Prev`, `5HexCa`, and `6`.



The screenshot shows the dhex editor interface for a file named `../butta1.txt`. The address bar at the top indicates the file path: `/cygdrive/C/DIDATTICA/ArchitetturaElaboratori/Int...`. The main display area shows a hex dump of the file's content. The first line of the hex dump is `63 69 61 6f 0a 6d 65 72`, which corresponds to the ASCII string `ciao.mer`. The second line is `64 61 0a`, corresponding to `da.`. A red circle highlights the `0a` sequence in the first line, indicating a Linux-style line feed (LF) end-of-line character. The left margin shows addresses 0, 8, 10, 18, 20, and 28. At the bottom, there is a menu with options: `1Goto`, `2Search`, `3Next`, `4Prev`, `5HexCa`, and `6`.

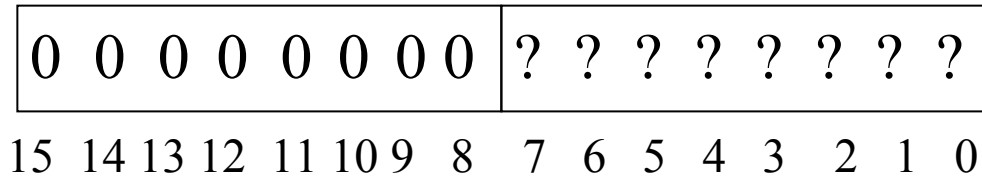
CODIFICHE WIDE-CHAR

esempio: codifica U, detta **UCS-2**, estesa alla codifica UNICODE **UTF-16**

Usa valori interi a 16 bit, inseriti in blocchi di due byte.

Rappresenta anche i codici di Latin-1 (e ASCII) mettendoli nel byte meno significativo e usando un byte più significativo posto a zero.

Char ASCII in UCS-2



Problema Endianess: stesso numero viene rappresentato in modo diverso a seconda che venga messo, in memoria, nel byte di indirizzo minore il byte più significativo (big endian) o quellomeno significativo (little endian).

L'estensione UTF-16 inserisce, ad inizio file, il **BOM byte-order-mark** una coppia di byte contenente il valore 0xFEFF che viene rappresentato in due modi diversi in big endian (prima 0xFE e poi 0xFF) ed in little endian (prima 0xFF e poi 0xFE) per distinguere l'ordine dei codici a 16 bit nel file..

Problema Efficienza: se devo rappresentare solo caratteri ASCII uso il doppio dei byte necessari.

Problema Zeri: alcuni linguaggi (C e Assembly) usano i byte a zero come indicatori di fine stringa. I programmi che usano rappresentazioni ASCII e Latin-1 si confondono e vedono la fine della stringa prima di dove è realmente.

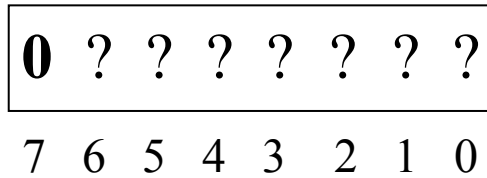
CODIFICHE MULTIBYTE

esempio: codifica F, poi detta codifica **UNICODE UTF-8**

Massima compatibilità con codifica ASCII

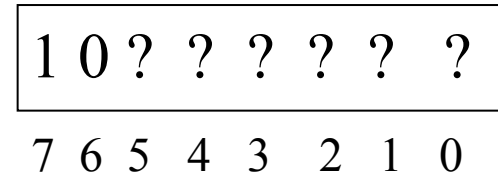
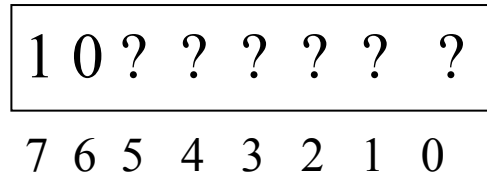
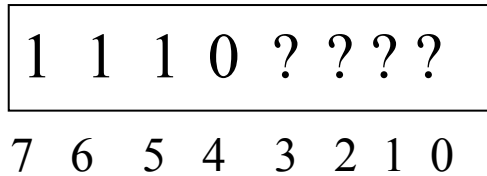
Usa più bytes da leggere uno per uno.

Se codifica codici ASCII usa un solo byte con bit più significativo posto a zero e gli altri 7 bit rappresentano il valore numerico ASCII.



Se codifica codici non ASCII il bit più significativo del primo byte è posto a 1, poi ci sono tanti bit a 1 per quanti sono i byte (oltre al primo) da usare. Seguono quel numero di bytes, ciascuno avente primo bit a 1 e secondo bit a zero e poi 6 bit da usare

primo byte secondo byte terzo byte



Problema Endianess: non esiste problema di endianess, perché interpreto ciascun byte separatamente dagli altri.

Non esiste BOM, ma **in Windows inseriscono a inizio file una tripletta di caratteri in quest'ordine 0xEF,0xBB,0xBF che confonde programmi che usano ASCII puro.**

CODIFICHE PER SCRIVERE CODICE ANSI C

Dovendo scrivere codice sorgente in linguaggio ANSI C e bash bisogna utilizzare solo caratteri appartenenti alla codifica ASCII ad 8 bit. Si possono perciò usare le codifiche ASCII e Latin-1 ma anche la codifica F (UTF-8), ma in quest'ultimo caso bisogna eliminare il BOM inserito da windows a inizio file.

Inoltre, bisogna eliminare i caratteri `\n` aggiunti a fine riga da editor in Windows e Mac.

Eliminare BOM di Windows per UTF-8

```
sed 's/^\xEF\xBB\xBF//'
```

Eliminare i vari `\r` di troppo

```
sed 's/\n\r/\n/'
```

```
sed 's/\r\n/\n/'
```

Questo è il comando per eliminare il BOM di UTF-8 ed i vari `\r` dal file `oldfile` creando il nuovo `newfile`

```
cat oldfile | sed 's/^\xEF\xBB\xBF//' $1 | sed 's/\n\r/\n/' | sed 's/\r\n/\n/' > newfile
```