1. CODIFICHE DEL TESTO

Noi siamo abituati a considerare le lettere a,b,c,….z,A,B,C,….Z le cifre 0,1,2,3,…9, e i simboli di punteggiatura ;:,.| per come le vediamo rappresentate su carta o visualizzate in un editor.
In realtà in un computer ciascuno di quei simboli viene rappresentato in memoria (e anche salvato su disco) per mezzo di una particolare sequenza di bit. Quindi il computer usa una sequenza di bit diversa per rappresentare ciascun simboli. Poi ciascun programma di editazione di file visualizza a video il simbolo che corrisponde a ciascuna particolare sequenza di bit.

Analogamente a quello che abbiamo visto per gli interi, se abbiamo a disposizione N bit possiamo rappresentare 2^N diversi simboli.
All'inizio dell'uso dei computer, si utilizzava solo la lingua inglese, non era necessario rappresentare lettere accentate o altri simboli strani, quindi in pratica bastava memorizzare 128 diversi simboli, per cui bastava utilizzare 7 bit (2^7=128) per rappresentare tutte le lettere, le cifre e i simboli di punteggiatura necessari.

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 il valore numerico memorizzato in quei bit ed il simboli che viene rappresentato da quel particolare valore numerico.

2. CODIFICA ASCII

Fu definita così la codifica ASCII che usava un byte (8 bit, di cui uno non utilizzato e posto a zero) per rappresentare ciascun simbolo (detto carattere). Quindi la codifica ASCII è una mappa che associa a ciascun valore (compreso tra 0 e 127) un particolare simbolo (lettera, cifra punteggiatura). Quindi io posso riferirmi ad un particolare simbolo sia indicandolo col suo nome (ad es la lettera k) sia indicandolo con il valore numerico che nella mappa ASCII identifica univocamente quel simbolo.
Mediante questa mappa è possibile rappresentare tutti i caratteri che possiamo usare per scrivere un programma in linguaggio ANSI C.
In questa codifica ASCII sono rappresentati anche alcuni simboli di punteggiatura (;:,.|!"'?) e pure alcuni caratteri "strani" (lo spazio, il tab, la sottolineatura _), ed alcuni caratteri chiamati"non stampabili" perché sono tipicamente utilizzati per far capire agli editor che devono svolgere qualche operazione particolare (ad es. andare a capo) o per rappresentare casi inquietanti (il carattere nullo, di valore 0, che non e' visualizzabile) da un editor. I caratteri di controllo sono i primi 32 della tabella ASCII, cui seguono i caratteri stampabili, cioè visualizzabili da un comune editor. Qui di seguito un estratto dalla tabella ASCII.

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..		

Questo semplice programma in ANSI C permette di visualizzare la mappa dei caratteri ASCII:

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

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

3. LE CODIFICHE A 8 BIT

La codifica ASCII utilizza 7 bit su 8 (l'ottavo bit è posto a zero) e identifica 128 caratteri.
Il primo modo per rappresentare altri caratteri oltre a quelli definiti dalla codifica ASCII, fu appropriarsi dell'ottavo bit, aggiungendo in questo modo altri 128 possibili caratteri.
Alcune codifiche importanti sono state così ottenute, ad esempio quella denominata iso-8859-1 (anche detta Latin-1) che comprende 256 caratteri capaci di descrivere gli alfabeti di molte lingue europee occidentali. Vennero poi predisposti altri codici iso-8859-x (da iso-8859-2 a iso-8859-16) per le lingue europee con altri alfabeti (ad esempio greco, cirillico...). Le codifiche per gli alfabeti orientali (essenzialmente giapponese, cinese, coreano) vennero (in maniera simile) raggruppate sotto la famiglia di denominazioni ISO/IEC 2022.
Per quasi tutte le codifiche ISO, si fece in modo che i primi 127 caratteri (quelli con l'ottavo bit a zero) corrispondessero ai codici ASCII, in modo da conservare un qualche tipo di compatibilità con questo ultimo.

3.1 IL PROBLEMA DELLA FINE LINEA (EOL).

Nella mappa ASCII alcuni caratteri servono, quando collocati in un file di testo, ad indicare all'editor che visualizza il file per l'utente, di visualizzare qualcosa di particolare. Ad esempio, il carattere 13 (valore esadecimale 0x0D, chiamato newline ed indicato simbolicamente con \n) indica all'editor di spostarsi alla riga successiva (andare a capo). Altro esempio, premendo da tastiera il carattere back si dice all'editor di cancellare il carattere immediatamente a sinistra della posizione corrente.
Il problema è che sistemi operativi diversi utilizzano sequenze diverse di caratteri per rappresentare l'andata a capo (end-of-line, EOL).
I sistemi Unix/Linux utilizzano come carattere di andata a capo il solo carattere **10** (**0x0A**) (**\n**).
I sistemi Windows utilizzano invece una coppia di caratteri per rappresentare l'andata a capo, cioè la coppia 13 e 10 cioè (0x0D 0x0A) e in simboli (\r\n) (carriage return [ritorno carrello] il primo e newline il secondo).
I sistemi Mac utilizzano invece la coppia di caratteri invertita rispetto a Windows, cioè la coppia 10 e 13 (\n\r) (0x0A 0x0D) (newline e carriage return).

Se un file editato in Windows o Mac viene spostato su un sistema Linux appariranno ugualmente le "andate a capo riga" identificati dai \n, ma saranno anche presenti alcuni caratteri \r residui che però l'editor normalmente non visualizza poiché il corrispondente simbolo non è "stampabile".
Per vedere anche i caratteri nascosti si possono usare degli editor "binari" o "esadecimali" che visualizzano il valore numerico di ciascun byte di un file invece che il corrispondente simbolo. Uno di questi editor è **dhex**.

4. Codifiche wide-char.

Esistono però lingue che hanno (molti) più simboli dei 256 rappresentabili con un singolo byte: il cinese e il giapponese sono due fra le più importanti. Le codifiche per queste lingue presenta quindi la necessità di usare più di un byte per carattere cosa che può essere fatta in almeno due modi - ed entrambi sono stati usati in diverse codifiche.

La scelta apparentemente più naturale è quella di usare lo stesso numero di byte per la codifica di ogni simbolo. Ad esempio, per un alfabeto che abbia più di 256 ma meno di 65536 simboli, questo significa che ogni simbolo sarà codificato con due byte, da 00000000-00000000 a 11111111-11111111. Codifiche di questo genere si chiamano "wide-char" (caratteri larghi). Benché facilmente e immediatamente comprensibili, queste codifiche provocano tre problemi.

Consideriamo come esempio significativo la cosiddetta **codifica U**, anche detta, **UCS-2** che poi è stata migliorata leggermente inserendo il BOM ed è diventata **UTF-16**.

1) U è wide-char, con due byte per simbolo.
2) U utilizza i primi 256 caratteri nello stesso ordine e con lo stesso significato della codifica latin-1. Questo significa che tutte le lettere delle principali lingue europee occidentali sono contenute in un solo byte, il primo dei due. Il secondo byte vale zero.

Il primo problema (quello evidente) è l'inefficienza di U. Infatti U contiene 511 simboli che vengono codificati in sequenze che hanno almeno un byte nullo. Tuttavia, quando U viene utilizzata per codificare testi costituiti da soli caratteri occidentali, questi risultano occupare il doppio dello spazio che sarebbe necessario, perché tutti i caratteri occidentali hanno una codifica in cui il byte più significativo è nullo.

Il secondo problema (quello meno apparente) è noto come problema dell'endianness (o byte-ordering). Una coppia di bytes può essere memorizzata in due modi diversi:

a) scrivendo prima (cioè nel byte di indirizzo minore) il byte più significativo, poi quello meno significativo (big endian)
b) scrivendo prima il byte meno significativo, poi quello più significativo (little endian).

Quindi, la codifica U può essere interpretata correttamente solo dopo che chi la vuole decodificare abbia in qualche modo determinato l'endianness con cui è stata scritta.

Per risolvere il problema del riconoscimento dell'endianess, un miglioramento della codifica UCS-2 è stato introdotto dalla standardizzazione UNICODE che ha dato origine alla codifica UTF-16.
UTF 16 definisce anche un particolare valore (**Byte-Order-Mark** o **BOM**) che si può usare per capire l'endianness usata nella codifica del testo. Il BOM è rappresentato da una coppia di bytes di valore (esadecimale) FEFF che **su una macchina big-endian viene rappresentato dalla sequenza 0xFE,0xFF** e dalla sequenza **0xFF,0xFE su una macchina little endian**. Poiché in Unicode il codepoint U+FEFF (Zero-Width No-Break Space : Spazio di ampiezza zero che non consente interruzioni) non può mai essere il primo carattere di una sequenza codificata mentre il codepoint U+FFFE non è - né sarà - mai assegnato ad un carattere valido, l'apparire di uno di questi due

codepoint all' inizio di una sequenza codificata permette di dedurre la endianness dell'intera sequenza.

Un terzo problema è che nella codifica UCS-2 / UTF-16 i caratteri occidentali (quelli di Latin-1) hanno un byte con tutti i bit zero (cioè un byte di valore zero. In alcuni linguaggi (C, assembly) tale byte a zero indica la fine di una stringa di testo. La presenza di un carattere occidentale rappresentato in UTF-16 fa sembrare terminata la stringa di testo se quella stringa viene letta da un programma che usa codifica ASCII.

5. Codifiche multibyte

Un'altra famiglia di codifiche si ottiene se si decide di codificare simboli diversi con un numero variabile di byte.
Consideriamo ad esempio la codifica F (è essenzialmente quella chiamata UTF-8) così definita:

1) I primi 127 simboli sono gli stessi - e nello stesso ordine - di quelli utilizzati dalla codifica ASCII e vengono scritti con unico byte il cui bit più significativo è posto a zero. La codifica dei primi 127 simboli è quindi uguale alla codifica ASCII.
2) Quando il bit più significativo di un dato byte è uguale a 1, il byte fa parte della codifica di un simbolo che viene codificato in più byte. Se uno o più bit successivi a quello più significativo sono pari a uno e seguiti da uno zero (110xyyzz, 1110yyzz, ...) si è in presenza del primo bit della codifica, e il numero di bit iniziali pari ad uno indica quanti byte sono usati per codificare il simbolo in esame. Se invece il bit successivo a quello più significativo è pari a zero (10xxyyzz) il byte in esame è il secondo, terzo... della codifica di un dato codepoint.

La codifica F risolve alcuni problemi delle codifiche "wide", introducendo comunque altri inconvenienti. Confrontiamola con la codifica U descritta nel paragrafo precedente.

1) La parte di F che riguarda i primi 127 simboli è molto più compatta della corrispondente codifica U. Per contro F è meno compatta di U nella codifica di tutti i simboli che richiedono più di due byte (guarda caso questa è la zona riservata alla maggior parte degli alfabeti orientali), che pagano un'inefficienza di circa il 30%.
2) F è indipendente dall'endianness: ogni simbolo è concepito come una sequenza di byte (non di coppie di bytes) ordinata intrinsecamente.
3) F non contiene byte nulli, ed è compatibile con la codifica ASCII: quindi i file di testo codificati in F possono essere manipolati con strumenti "tradizionali".
4) F non è invece compatibile con la codifica latin-1 (perchè questa utilizza anche i valori con il bit più significativo ad uno).
5) Decodificare F è più difficile che decodificare U. In particolare, una codifica come F rende difficile fare cose come "trovare l'ottavo carattere di una parola". Usando una codifica come U posso infatti compiere questa operazione semplicemente estraendo l'ottava "word" della sequenza (in una codifica a byte singolo, questo si fa estraendo l'ottavo byte). Se invece la codifica in uso è F, per poter trovare il carattere richiesto devo prima leggere i byte della sequenza di ingresso e decodificarli fino ad arrivare all'ottavo codepoint.
6) F contiene alcune sequenze di byte che sono vietate (ad esempio: 110xyyzz-0qxxyyzz). Questo rende possibile stabilire con certezza che un sequenza contenente una sotto-sequenza proibita non usa la codifica F. Questa sembra una banalità ma è il caso di far notare che questa proprietà non è condivisa da molte codifiche a byte singolo o wide: in particolare, qualunque sequenza, anche

casuale, di byte può essere interpretata come corretta per una delle codifiche ISO-8859-x. Questa circostanza fa parte integrante del problema fondamentale.

Esistono molte altre possibili codifiche multibyte: in particolare esistono codifiche di tipo "shift" in cui la comparsa di una particolare sequenza di byte (upshift) cambia il significato di tutti i byte successivi fino alla ricezione di un'altra sequenza di byte definita (downshift) che ripristina la codifica precedente. Una vasta famiglia di codifiche di questo tipo è raggruppata nello standard ISO/IEC-2022, dedicato alla codifica di varie lingue orientali.

6. Unicode

Lo standard Unicode mira a codificare l'insieme di tutti i caratteri usabili. Si vuole, ad esempio, permettere l'utilizzo di testo multilingua senza dover cambiare codifica.
Lo standard Unicodeattuale contiene 1 114 112 (un milione centoquattordicimila centododici) simboli, suddivisi in 17 piani, ognuno composto di 65 536 codepoint, cioè 256 righe contenenti 256 codepoint ciascuna.

Il piano 0, costituito dai primi 65536 codepoint, è chiamato Basic Multilingual Plane (BMP) e contiene la maggior parte del repertorio di caratteri oggi in uso. Per assicurare la retro-compatibilità con ASCII, è previsto che i primi 127 codepoint coincidano con quelli definiti dalle specifiche ASCII. La più recente formulazione di UNICODE contiene gran parte di tutte le lingue in uso e del passato,i loro diacritici, simboli matematici, simboli musicali e molte altre simbologie. Inoltre più di 10 piani non sono assegnati (cioè i codepoint in essi contenuti non corrispondono ad alcun carattere) né è probabile che vengano assegnati in un futuro prossimo. Oltre a catalogare un enorme repertorio di caratteri, Unicode definisce tutta una serie di informazioni accessorie (ordinamento dei vari set di caratteri, regole per assicurare la "multi-direzionalità" del testo...) che non hanno una diretta influenza sul problema fondamentale sopra definito.

Inoltre Unicode definisce anche ciò che chiama "Unicode transformation format" (UTF) e "Universal character set" (UCS): questi non sono altre che le codifiche necessarie per la rappresentazione esterna di Unicode, cioè il necessario per rappresentare le codifiche precedenti (per esempio la U e la F).

6.1 UTF-16

UTF-16 (ex UCS-2, descritta precedentemente come codifica U): una codifica multibyte che permette la rappresentazione dell'intero repertorio Unicode e che rappresenta l'intero BMP (65536 codepoint) con una codifica di tipo "wide" costituita da due byte (questa era l'originale codifica UCS-2, che era in grado di rappresentare il solo BMP). Mentre UTF-16 e UCS-2 sono spesso confuse, UTF-16 è l'unica di uso corrente. In UTF-16 ogni carattere viene codificato in una sequenza di lunghezza variabile da 2 a quattro ottetti (byte), riservando le codifiche a quattro byte per codepoint rarissimi gestiti tramite "codepoint surrogati".
UTF 16 definisce anche il particolare valore (**Byte-Order-Mark o BOM**) che si può usare per capire l'endianness usata nella codifica del testo. Il **BOM** è rappresentato dal codepoint (esadecimale) U+FEFF che su una macchina big-endian viene rappresentato dalla sequenza 0xFE,0xFF e dalla sequenza 0xFF,0xFE su una macchina little endian. Poiché il codepoint U+FEFF (Zero-Width No-Break Space : Spazio di ampiezza zero che non consente interruzioni) non può mai essere il primo carattere di una sequenza codificata mentre il codepoint U+FFFE non è - né sarà -

mai assegnato ad un carattere valido, l'apparire di uno di questi due codepoint all' inizio di una sequenza codificata permette di dedurre la endianness dell'intera sequenza.

6.2 UTF-8

UTF-8 è una codifica multibyte che massimizza la compatibilità con ASCII (parzialmente descritta precedentemente come **codifica F**). In UTF-8 ogni carattere viene codificato in una sequenza di lunghezza variabile da 1 a quattro ottetti (byte)
In UTF-8 non esiste un BOM (poichè i caratteri sono usati singolarmente e non in coppie) anche se alcuni programmi (soprattutto operanti in ambiente windows) ne inseriscono uno (**0xEF,0xBB,0xBF**) equivalente a quello usato in UTF-16. Questo è permesso, ma sconsigliato, dallo standard, e in essenza non fa che rompere le scatole.

6.3 UTF-32 (UCS-4)

UTF-32/UCS-4: una codifica "wide" a lunghezza fissa: ogni codepoint di Unicode è rappresentato da una sequenza di 4 byte. Si applicano le considerazioni sul BOM già viste per UTF-16. Questa codifica è usata, in pratica, molto di rado.

6.4 Applicazione di UTF-8 e UTF-16

A causa dei vantaggi illustrati della codifica F sulla codifica U, UTF-8 è oggi la codifica più usata per la rappresentazione esterna di testi e testi multilingua. UTF-16 è per contro molto usata nella rappresentazione interna delle stringhe (in particolari è quella in uso in tutti i sistemi operativi Microsoft posteriori a Windows 2000).