Soluzione esercizi 2° lezione

- Scritta Devanagari:
- UTF-16
  - Primo code point 0x10000 (D800 DC00): Linear B syllable B008 A
  - Ultimo code point 0x10FFFF (DBFF DFFF): non character
- UTF-24: http://stackoverflow.com/questions/10143836/why-is-there-no-utf-24
- UTF-8 interpretato come Latin-1: caffÃ¨
- helloworld.txt: UTF-16BE

प	0000 1001 0010 1010	09 2A
◌ू	0000 1001 0100 0010	09 42
र	0000 1001 0011 0000	09 30
◌्	0000 1001 0100 1101	09 4D
त	0000 1001 0010 0100	09 24
ि◌	0000 1001 0011 1111	09 3F
पूर्ति (Supply) http://translate.google.com/#hi/en/पूर्ति		

Varianti UTF-8: CESU-8

- Compatibility Encoding Scheme for UTF-16
- Codifica “complementare”: per i code point fuori dal BMP utilizza i surrogati e li codifica in UTF-8.
- Utilizza 6 byte per i caratteri fuori dal BMP: codifica il code point in UTF-16 con la coppia dei code point surrogati e quindi codifica a sua volta ciascun elemento della coppia con la codifica UTF-8 di 3 byte

CESU-8 e UTF-8: esempio

	U+004D	U+00E8	U+4E2D	U+13080
	M	è	中	
UTF-8	4D	C3 A8	E4 B8 AD	F0 93 82 80
UTF-16	00 4D	00 E8	4E 2D	D8 0C DC 80
CESU-8	4D	C3 A8	E4 B8 AD	ED A0 8C ED B2 80

Varianti UTF-8: Modified UTF-8

- U+0000 codificato come 0xC0 0x80 (schema identico a UTF-8 ma non valido perché non è la sequenza più corta possibile)
- Stringhe in questa codifica non contengono mai il byte nullo (00000000) che può essere utilizzato quindi come terminatore di stringa da certe funzioni “tradizionali” di elaborazione stringhe
- Code point fuori dal BMP come in CESU-8
- Utilizzato in Java per la serializzazione degli oggetti e altri usi “interni” di rappresentazione stringhe.

BOM

- Concepito per risolvere il problema dell'endianness nello scambio dati
- Esistono due varianti di UTF-16 (e due varianti di UTF-32): UTF-16LE e UTF-16BE.
- Il BOM (Byte Order Mark) è un code point riservato all'uso speciale di stabilire l'endian-ness di un flusso dati (o di un file)
- Viene inserito come primo code point del flusso dati o del file
- Il BOM è U+FEFF
- in UTF-16BE è codificato 0xFE 0xFF
- Il code point U+FFFE è un "non carattere": il suo uso non è consentito (e non ha senso)
- Se quindi in un file leggiamo come primo carattere il code point U+FFFE (codificato ad esempio 0xFF 0xFE) stiamo sbagliando a interpretare l'ordine dei byte: da qui conosciamo l'endianness del resto della trasmissione
- Storicamente (Unicode 2.0) il carattere ha il significato di "zero-width non-breaking space" (ZWNBSP): oggi questo uso è deprecato e tale funzione è ricoperta dal code point U+2060 (WORD JOINER).
- A causa di questa origine potrebbero succedere cose impreviste concatenando due file UTF-16 con utility UNIX come *cat*
- UTF-8 è byte oriented: non ci sarebbe bisogno di inserire il BOM. Tuttavia lo standard Unicode afferma che può essere utilizzato per rendere chiaro il fatto che si sta utilizzando la codifica UTF-8 (EF BB BF)

I caratteri di controllo (1/2)

- Quando di parla di “caratteri di controllo” si intendono i caratteri di controllo ASCII (C0 controls) e ISO 8859 (C1 controls)
- Di fatto non si tratta di caratteri: si dovrebbe parlare di *control codes*
- Si possono “digitare” alla tastiera in combinazione con il tasti CTRL (p.e. CTRL+C = U+0003)
- Erano utilizzati inizialmente per le telescriventi (C0 controls)
- Oggi hanno comunque un uso importante (break control, movimento cursore, formattazione, sequenze di escape, etc..)
- I C1 controls sono utilizzati in ISO 2022 per switchare tra diversi encoding
- L’uso originario nelle telescriventi ha reso alcuni di essi caratteri di “formattazione” (CR, LF, HT, VT, etc..) che determinano l’aspetto di un testo
- Unicode include ovviamente tutti i caratteri di controllo C0 e C1, tuttavia non standardizza la loro funzione.

I caratteri di controllo (2/2)

- Unicode introduce alcuni code points di significato speciale che potrebbero essere considerati dei veri e propri “caratteri di controllo”: Layout controls e Special operators come ad esempio LSEP, PSEP, LRM, RLM, IAFS, etc.. tuttavia questi caratteri vengono considerati “di punteggiatura” e quando si parla di caratteri di controllo si intendono i C0 e C1 controls
- Un carattere di controllo non ha una rappresentazione grafica, ma in alcuni contesti è necessario rappresentarli (p.e. istruzioni): a tale scopo esiste un blocco Unicode dedito a questa funzione (control pictures).
  Si tratta di caratteri che rappresentano i caratteri di controllo, ma sono da questi **completamente** distinti
- Esempio: il carattere di controllo ESCAPE U+001B può essere rappresentato dal carattere grafico U+241B (SYMBOL FOR ESCAPE) con un glifo specifico: ESC
- Alcuni programmi (p.e. editor di testo) potrebbero rappresentare i caratteri di controllo in qualche forma grafica (^C, CTRL-C, sigla in negativo): di fatto è come se utilizzassero una certa combinazione di glifi per rappresentare quel carattere particolare.
- http://www.cs.tut.fi/~jkorpela/chars/c0.html

Private Use Area

- PUA in BMP: 6400 code points
- SPUA-A, SPUA-B (65.534 code points ciascuno)
- Usi proprietari (Emoticons Whatsapp)
- Utilizzi privati coordinati:
  - **ConScript Unicode Registry (CSUR)**
    Progetto più seguito, contiene sistemi di scrittura di fantasia come il Klingon di Star Trek e il Tengwar di Tolkien
    http://en.wikipedia.org/wiki/ConScript_Unicode_Registry
  - **Medieval Unicode Font Initiative (MUFI)**
    Abbreviazioni, legature e varianti dei testi medievali (152 caratteri MUFI sono stati inclusi nella versione 5.2 Unicode)
  - **SIL International**
    Fonetica e sistemi di scrittura non inclusi in Unicode (obsoleto perché nel frattempo molti caratteri sono stati inclusi ufficialmente in Unicode)
- U+F8FF (ultimo code point della PUA BMP)
  - Apple logo
  - Microsoft Windows logo (in Wingdings 1)

Identità di un carattere

- L’identità di un carattere è definita dalla definizione del repertorio: non è un concetto assoluto ma dipende dal repertorio
- In ASCII il carattere “–”, chiamato hyphen, è usato sia come trattino che come simbolo “meno”: in ASCII è un carattere multipurpose
- In Unicode, tale carattere è chiamato HYPHEN-MINUS, proprio ad indicare questa sua ambivalenza
- Unicode definisce il simbolo matematico della sottrazione (meno) come carattere separato e una gran varietà di trattini e linee (dash)
- Unicode definisce caratteri come il MICRO SIGN e N-ARY PRODUCT distinti dalle lettere greche corrispondenti (small mu, capital pi): tale distinzione logica non richiede che vengano utilizzati glifi diversi per rappresentarli
- Tuttavia non esiste un carattere diverso per il pi greco π inteso come costante numerica
- Per il simbolo dell’unità di misura della resistenza elettrica (OHM SIGN: Ω) esiste un carattere specifico che però è definito avere equivalenza canonica con la lettera greca maiuscola omega: esistono due caratteri distinti ma sono considerati equivalenti.
- Viceversa il simbolo del prefisso micro e la lettera greca minuscola μ non sono considerati equivalenti (anche se c’è un mapping di compatibilità)
- Se la cosa vi sembra avere poco senso, non siete i soli, il fatto da evidenziare è che vi possono essere situazioni molto diverse per ciascun carattere, principalmente per motivi storici

Mancata visualizzazione di un carattere

- La rappresentazione di un carattere può fallire nei seguenti casi (che non si escludono a vicenda):
  - Errore nell'interpretazione dell'encoding
  - Sequenza di byte illegale (p.e. errato UTF-8 o code point surrogato singolo)
  - Non carattere
  - Unassigned code point
  - Carattere di uso privato
  - Glifo mancante
- Soluzioni / conseguenze:
  - Visualizzazione di un carattere o sequenza di caratteri errata (Mojibake)
  - Ignorare il carattere o la sequenza di byte (nessuna rappresentazione)
  - Utilizzare il carattere di controllo **substitute character** (SUB) U+001A (non ha glifo: nessuna rappresentazione)
  - Mostrare il glifo del carattere **replacement character**
  - Utilizzare il glifo di un altro font (se il carattere è valido ed è disponibile in un font installato sul sistema)
  - Mostrare un punto interrogativo (o una serie di punti interrogativi)
  - Mostrare un rettangolo vuoto (o glifo specifico del font)
  - Mostrare un rettangolo con all'interno il code point value (soluzione molto elegante adottata da Firefox per caratteri validi di cui manca il glifo in ogni font installato sul sistema)

Composizione dinamica

á	à	ä	â
é	è	ë	ê
í	ì	ï	î
ó	ò	ö	ô
ú	ù	ü	û

a e i o u + ´ ` ¨ ^

- L’idea è che si possono comporre segni diacritici (non spacing marks) con qualsiasi carattere
- Per esempio: ù = u + ` (U+0075 U+0300)
- Ha origini “antiche”: in ASCII con le telescriventi il carattere BS (08) era utilizzato a questo scopo
- usi recenti: Windows-1258 per il vietnamita

- Questo rende più logico il processo di codifica dei caratteri con segni diacritici...
- ...e rende possibile in teoria la combinazione di qualsiasi carattere con qualsiasi segno diacritico (e anche con molteplici segni diacritici)
- La maggior parte dei caratteri con segni diacritici in uso in molti script è presente in Unicode come carattere precomposto
- La tecnologia per implementare queste combinazioni è complessa, parzialmente supportata, tuttora poco diffusa e può dare luogo a risultati graficamente poco soddisfacenti
- Attenzione: l’aspetto di un glifo può dipendere dalla lingua !!
- Pagina di test di supporto nei browser http://www.alanwood.net/unicode/combining_diacritical_marks.html

Decomposizione canonica

- In virtù della composizione dinamica i caratteri con simboli diacritici precomposti in Unicode possono essere “decomposti”
- Un carattere precomposto deve essere considerato equivalente alla sequenza di caratteri formata dal carattere base + i segni diacritici che lo accompagnano
- (Quasi) ogni carattere precomposto definito in Unicode ha un mapping con una coppia di caratteri
  - Il primo carattere può essere un carattere base o, a sua volta, un carattere precomposto
  - Il secondo carattere è un diacritico
- Esempi:
  - ù (U+00F9) si decompone in u + ̀ (U+0075 U+0300)
  - ộ (U+1ED9) si decompone in ọ + ̂ (U+1ECD U+0302)
  - ọ (U+1ECD) si decompone in o + ̣ (U+006F U+0323)

Composizione: problemi

- Una stessa stringa può essere scritta in molti modi diversi
- Per esempio il carattere ộ può essere rappresentato dalle seguenti sequenze di code point:
  - U+1ED9 (ộ)
  - U+1ECD U+0302 (ọ + ̂)
  - U+00F4 U+0323 (ô + ̣)
  - U+006F U+0323 U+0302 (o + ̣ + ̂)
  - U+006F U+0302 U+0323 (o + ̂+ ̣)
- In un’applicazione un utente potrebbe immettere una qualsiasi di queste combinazioni a significare lo stesso carattere
- Se l’applicazione deve eseguire una ricerca in un testo, il testo potrebbe a sua volta contenere questo carattere secondo una qualsiasi di queste sequenze
- Con i linguaggi di markup (HTML) le cose possono anche peggiorare con l’utilizzo di named characters entities (ô) e di numbered entities (ộ)
- Matching: CSS selectors, JS identifiers e stringhe, Nomi dei font, URI path e file sul sistema operativo, sorting, regexp matching, selezione...
- ...ma non è finita qua!

Caratteri di compatibilità

- In Unicode sono presenti molti caratteri che hanno significati e usi particolari
- Tali usi e significati sarebbero meglio gestiti ad altri livelli (markup, formattazione, contesto, etc.)
- A causa dei principi di Universalità e Convertibilità di Unicode, dato che questi caratteri sono stati inclusi in altri encoding DEVONO essere inclusi in Unicode
- Alcuni esempi:
  - µ (MICRO SIGN): simbolo del prefisso micro, ha lo stesso glifo della lettera greca minuscola mu, ma definito come carattere a sé stante perché presente in LATIN-1
  - ① (CIRCLED DIGIT ONE): una serie di cifre numeriche racchiuse da un cerchio, presenti solo perché definite in altri encoding
  - ﺒ (ARABIC LETTER BEEH FINAL FORM): tutte le forme posizionali arabe, presenti solo perché definite in altri encoding
  - fi (LATIN SMALL LIGATURE FI): legature tipografiche, dovrebbero essere gestite dai font, ma sono state definite come caratteri a parte in altri encoding

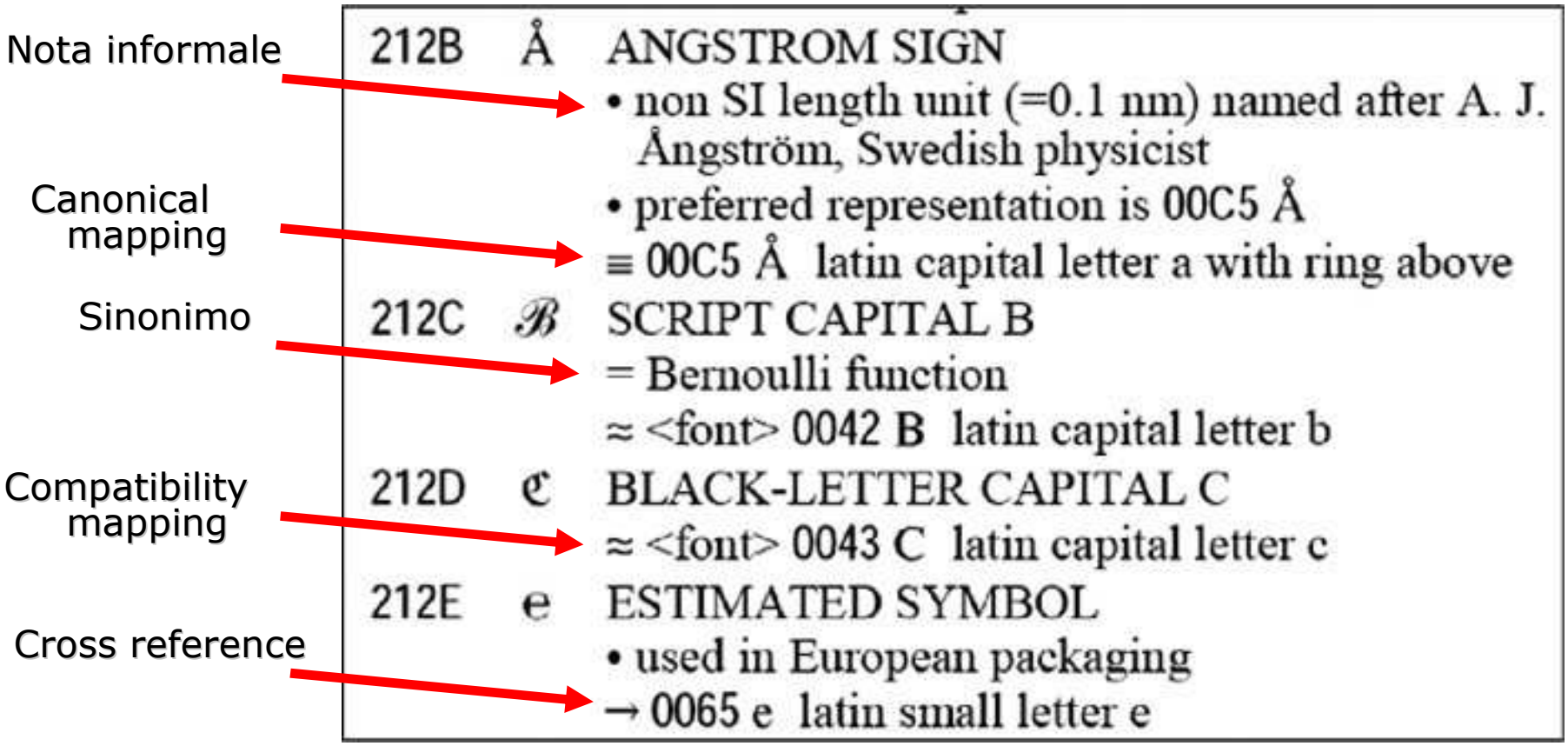
Decomposizione di compatibilità

- I caratteri di compatibilità hanno una decomposizione di compatibilità: in Unicode sono mappati ad altri caratteri, insieme ad un tag relativo al tipo di informazione necessaria a “rendere” il carattere originale
- I possibili tag sono i seguenti:
  - <circle>An encircled form
  - <compat>Otherwise unspecified compatibility character
  - <final>A final presentation form (Arabic)
  - <font>A font variant (e.g., a blackletter or italics form)
  - <fraction>A vulgar fraction form, such as ½
  - <initial>An initial presentation form (Arabic)
  - <isolated>An isolated presentation form (Arabic)
  - <medial>A medial presentation form (Arabic)
  - <narrow>A narrow (hankaku) compatibility character
  - <noBreak>A no-break version of a space, hyphen, or other punctuation
  - <small>A small variant form
  - <square>A CJK squared font variant
  - <sub>A subscript form
  - <super>A superscript form
  - <vertical>A vertical layout presentation form
  - <wide>A wide (zenkaku) compatibility character

Esempi di caratteri compatibili

- µ (MICRO SIGN) U+00B5:
  - <compat> GREEK SMALL LETTER MU (U+03BC)
- ① (CIRCLED DIGIT ONE) U+2460:
  - <circle> DIGIT ONE (U+0031)
- ﭖ (ARABIC LETTER BEEH FINAL FORM) U+FB53:
  - <final> ARABIC LETTER BEEH (U+067B)
- fi (LATIN SMALL LIGATURE FI) U+FB01:
  - <compat> LATIN SMALL LETTER F (U+0066) LATIN SMALL LETTER I (U+0069)

Cosa si trova nei code charts



ATTENZIONE!! Canonical mapping e compatibility mapping non sono relazioni simmetriche: se A ha un mapping verso B non è vero il viceversa

Mapping canonici e di compatibilità

In altre parole:

- Se A ha un canonical mapping verso B, allora A e B sono due entità che di fatto rappresentano lo STESSO carattere in Unicode. Come code point o sequenze di code point possono differire, ma hanno lo stesso valore semantico e dovrebbero anche avere lo stesso rendering (glifo)
- Se A ha un compatibility mapping a B, allora A e B denotano caratteri che sono fondamentalmente molto simili, ma che generalmente hanno significati e scopi di utilizzo diversi e dovrebbero avere glifi diversi

Normalizzazioni

- Allo scopo di ridurre la possibile variabilità di un testo Unicode sono state definite le seguenti normalizzazioni:

NFD	Normalization Form D	Canonical decomposition	
NFC	Normalization Form C	Canonical decomposition, canonical composition	W3C
NFKD	Normalization Form KD	Compatibility decomposition, canonical decomposition	
NFKC	Normalization Form KC	Compatibility decomposition, canonical decomposition, canonical composition	IDNA

Normalizzazioni: esempio

Stringa di esempio: fiancé

NFD	fiance´	Decomposizione di é
NFC	fiancé	Decomposizione e ricomposizione di é
NFKD	fiance´	Decomposizione di fi e di é
NFKC	fiancé	Decomposizione di fi e di é e ricomposizione di é

Internet Media Types

- Una sequenza di ottetti che rappresenta un testo può essere codificata in molti modi: l'informazione sulla codifica diventa a questo punto fondamentale durante lo scambio dati (Internet)
- In generale una sequenza di byte può rappresentare molte cose diverse (immagini, video, testo, etc..)
- MIME è uno standard internet (RFC 822, 2822, 5322) originariamente concepito per estendere le email in modo tale da supportare:
  - Testi in encoding non ASCII
  - Allegati non di testo
  - Corpi del messaggio con parti multiple
  - Informazioni di intestazione a proposito della codifica di carattere utilizzata
- Espansioni ad altri protocolli: HTTP, RTP, SIP

Intestazioni MIME

- MIME-Version
- Content-ID
- Content-Type (type/subtype)
- Content-Disposition (inline, attachment)
- Content-Transfer-Encoding, in SMTP:
  - 7 bit (998 ottetti nel range 1..127 per linea)
  - quoted-printable (binary-to-text encoding)
  - base64 (binary-to-text encoding)
  - 8 bit (998 ottetti per linea)
  - binary

Media Types (Content-Type)

- application
  - atom+xml
  - json
  - ...
  - pdf
  - x-www-form-urlencoded
  - ...
- audio
  - ...
  - mp4
  - mpeg
  - ogg
  - ...
- image
  - gif
  - jpeg
  - png
  - tiff
  - ...
- message
  - http
  - imdn+xml
  - partial
  - rfc822
- model
  - example
  - ...
- **multipart**
  - mixed
  - alternative
  - related
  - form-data
  - signed
  - encrypted
  - byteranges
- **text**
  - cmd
  - css
  - csv
  - html
  - javascript
  - plain
  - vcard
  - xml
  - calendar
- video
  - mpeg
  - mp4
  - ...
  - x-flv

Il type “text” può avere il parametro opzionale charset

MIME: esempio

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=0016e6d9678a34060104c90e569b

--0016e6d9678a34060104c90e569b
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

In realt=E0 questo =E8 solamente un test
--0016e6d9678a34060104c90e569b
Content-Type: image/jpeg; name="test.jpg"
Content-Disposition: attachment;
   filename="test.jpg"
Content-Transfer-Encoding: base64

/9j/4AAQSkZJRgABAgAAAQABAAD//gAEKgD/4gIcSUNDX1BST0ZJTEUAAQEAAAIMbGNtcwIQAABt
bnRyUkdCIFhZWiAH3AABABkAAwApADlhY3NwQVBQTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
32bdz2o2Nlrm9uWSyghsbuMyezaubyHeTOf42aHky591/ipzf8XCp7Hsb5OK/i5xnt/MI/4an//Z
--0016e6d9678a34060104c90e569b--
```

Quoted printable

- Codifica binary-to-text: ASCII (byte di 7bit, o meglio, con il primo bit a 0) per trasmettere (anche) byte di 8 bit (con il primo bit a 1)
- Utilizza il carattere = come carattere di escape
- Per byte il cui valore hexval > 7F: usa 3 byte di valore < 7F: "=" seguito dalle 2 cifre esadecimali (0–9A–F) che rappresentano il valore numerico del byte.
- ASCII equal sign (decimal value 61) va rappresentato con "=3D".
- Tutti i caratteri ad eccezione dei caratteri ASCII stampabili o di fine riga (CR LF) devono essere codificati in questo modo.
- Tutti i caratteri ASCII stampabili (valori decimali tra 33 e 126) possono essere lasciati invariati, tranne "=".
- ASCII HT e SPACE, decimal value 9 e 32, possono rimanere invariati, tranne nel caso in cui appaiano a fine riga.
- In tal caso, devono essere escaped come "=09" (HT) o "=20" (SPACE), oppure seguiti da a "=" (soft line break) come ultimo carattere della linea codificata; previene che il tab o lo spazio siano l'ultimo carattere della linea.
- Se il testo contiene line breaks significativi, devono essere codificati come sequenza CR LF, non con il loro valore originale, né via escape "=".
- Viceversa se i valori decimali 13 e 10 hanno significato diverso dal fine linea, devono essere codificati con =0D e =0A rispettivamente.
- Le linee di dati codificati QP devono essere inferiori a 76 caratteri.
- Per soddisfare il requisito senza alterare il testo originale, possono essere aggiunti soft line breaks.
- Un soft line break consiste di un "=" a fine linea, e non viene decodificato come line break nel testo decodificato

Quoted Printable: esempi

```
Se credi che verità=bellezza, allora la matematica è senz'altro
la più bella branca della filosofia

Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

Se credi che verit=E0=3Dbellezza, allora la matematica =E8 senz'altro=20
la pi=F9 bella branca della filosofia


Se credi che verità=bellezza, allora la matematica è senz'altro la più bella
   branca della filosofia

Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable

Se credi che verit=E0=3Dbellezza, allora la matematica =E8 senz'altro la =
pi=F9 bella branca della filosofia
```

Base64 (1/2)

- Codifica binary-to-text: ASCII (7bit) per trasmettere (anche) byte di 8 bit
- È una trasformazione di base.
- Spezza i byte di 8 bit in sequenze di 6 bit, quindi rappresenta i numeri a 6 bit in una base a 64 simboli, per ciascuno dei quali si utilizza una lettera rappresentabile in ASCII con 7 bit.

Base 64 (2/2)

M								a								n							
77								97								110							
0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
19								22				5				46							
T								W				F				u							

- I simboli utilizzati dalla base sono A-Za-z0-9+/
- Se il numero di byte da codificare non è divisibile per 3, si aggiungono in fondo byte nulli, si codifica base64, quindi, nell'ultimo gruppo di 4 cifre base64:
  - Se c'era un solo byte, si prendono solo le prime due cifre
  - Se c'erano due byte, si prendono le prime 3 cifre
  - Si fa il padding con il carattere "=" per riottenere un gruppo di 4 cifre base64
- Le linee vengono spezzate con CR LF ogni 64 simboli
- La codifica aumenta le dimensioni del dato di 4*(n/3)
- Esistono varianti di base64 (base64url: - e _ al posto di + e /)

Altre codifiche binary-to-text

- Base16, Base32
- Modified Base64 for...
  - Filenames, URLS, URL and filenames, UTF-7, XML name tokens, Regular expressions, program identifiers, etc..
- Uuencode (Unix per uucp)
- yEnc (Usenet)
- ASCII85 (aka Base85)
- Xxencode (ASCII <> EBCDIC)
- BinHex (mail Mac OS)
- **URL Encoding** (o Percent-encoding)
- Punycode

Encoded-Word

- I nomi e i valori delle intestazioni possono contenere solo caratteri ASCII (HTTP e SMTP)
- Per i valori non ASCII: MIME Encode-Word
- =?charset?encoding?encoded text?=
- encoding: Q o B
  (Quoted printable e Base64)
- si possono usare multiple encoded-words separate da CRLF SPACE
- =?iso-8859-1?Q?=A1Hola,_se=F1or!?=

Q encoding

- A differenza del Quoted Printable originale nel Q encoding il carattere “?” e lo spazio non possono essere utilizzati.
- “?” viene rappresentato con la sua sequenza di escape
- Lo spazio viene sostituito da “_”
- “_” quindi viene rappresentato con la sua sequenza di escape