UNICODE

Marcello Barnaba <vjt@openssl.it>

Perché UNICODE

- L’ alfabeto italiano contiene 21 caratteri
- L’ alfabeto inglese contiene 26 caratteri
- L’ alfabeto kanji (giapponese) contiene piu’ di 6000 caratteri

- Il set di caratteri ASCII, utilizzato da anni, può contenere al massimo 255 caratteri
- ASCII non è sufficiente a contenere alfabeti estesi

Rappresentazione classica delle stringhe di caratteri in memoria

```
char *str = ”pippo”;
```

p	i	p	p	o	\0
1	2	3	4	5	6

- Ogni stringa e` un array di caratteri di 1 byte
- Solo 255 caratteri disponibili
- Ad ogni carattere corrisponde un codice intero: questa tabella di corrispondenze e` chiamata ASCII
- Ogni stringa e` terminata dal carattere nullo (`char null = 0;`)

Stringhe di caratteri MultiByte (MBCS)

- Ogni carattere di una stringa e` composto da 1 o 2 byte:
- Se il primo carattere e` compreso tra 0x81 o 0x9f, e` necessario considerare il byte successivo per determinare l`intero carattere.
- Standard non riuscito, in quanto la programmazione con stringhe MBCS comporta eccessivi controlli anche solo per la lettura delle stringhe.
- Utilizzato esclusivamente nella serie 9x di Windows che utilizzano i caratteri kanji (giapponese)

UNICODE: la serie di caratteri a piu` byte

- Un nuovo metodo per rappresentare le stringhe di caratteri, omogeneo e flessibile.
- Creato nel 1988 da Apple e Xerox, e`stato ufficialmente presentato nel 1991 con la creazione di un consorzio per svilupparlo e promuoverlo, il cui sito web e` http://www.unicode.org.
- Standard affermato ed efficiente, indispensabile se si pianifica di sviluppare software che preveda una localizzazione linguistica.

Come UNICODE permette di rappresentare stringhe di caratteri

```
#include <wchar.h>

…

wchar_t *str = L"pippo";
```

\0	p	\0	i	\0	p	\0	p	\0	o	\0	\0
1	2	3	4	5	6	7	8	9	10	11	12

I caratteri in UNICODE

- Ogni stringa e` un array di caratteri, ognuno composto da 2 byte: ogni carattere e` quindi a tutti gli effetti un unsigned short
  (`typedef unsigned short wchar_t;`)
- 65535 caratteri disponibili
- Ad ogni carattere corrisponde un codice intero: la tabella di corrispondenze e` la tabella UNICODE
- Ogni stringa e` terminata dal carattere nullo
  (`wchar_t null = 0;`)

Vantaggi di UNICODE rispetto ad ASCII

- Esteso insieme di caratteri disponibili
- UNICODE e` a tutti gli effetti lo standard delle stringhe del futuro: utilizzarlo da subito significa scrivere software che non dovra` essere riscritto per UNICODE quando ascii non sara` piu` supportato
- UNICODE ingloba ASCII: I primi 127 caratteri nella tabella UNICODE corrispondono alla tabella ASCII, con tutti i vantaggi che conseguono da come sono disposti i caratteri stessi, e consentendo inoltre la compatibilita` con il software gia` scritto che fa uso delle peculiarita` di ASCII

Vantaggi di UNICODE rispetto ad MBCS

- UNICODE è più semplice e coerente di MBCS: per scorrere una stringa e` sufficiente incrementare/ decrementare un puntatore ad un wchar_t
- In UNICODE Non ci sono caratteri speciali che indicano la dimensione del successivo carattere: sono tutti di 2 byte

Organizzazione della tabella UNICODE

Codice(16 bit)	Caratteri	Codice(16 bit)	Caratteri
0000 – 007F	**ASCII**	0300 – 036F	**Simboli diacritici**
0080 – 00FF	**Caratteri Latin1 (eng)**	0400 – 04FF	**Cirillico (russo)**
0100 – 017F	**Latino Europeo**	0530 – 058F	**Armeno**
0180 – 01FF	**Latino Esteso**	0590 – 05FF	**Ebraico**
0250 – 02AF	**Fonetica Standard**	0600 – 06FF	**Arabo**
02B0 – 02FF	**Lettere Modificate**	0900 – 097F	**Devangari**

- 35.000 posizioni utilizzate: 29.000 libere e 6.000 riservate per utilizzo personale

Supporto UNICODE nella libreria standard C

- Le dichiarazioni delle stringhe e dei buffer destinati a contenerle non vanno piu` dichiarate con char ma con wchar_t:
  - `wchar_t *wptr;`
    `/* puntatore a stringa unicode */`
  - `wchar_t wbuf[64];`
    `/* buffer di 64 caratteri unicode */`

- Le funzioni per la manipolazione di stringhe hanno la corrispondenza str -> wcs

Nome funzione ASCII	Nome funzione UNICODE
`strlen()`	`wcslen()`
`strcpy()`	`wcscpy()`
`strchr()`	`wcschr()`

Attenzioni da porre quando si utilizzano stringhe UNICODE

- **Non c`è più una corrispondenza tra caratteri e byte:**

Allocazione della memoria:

- `wptr = (wchar_t *) malloc(numero_caratteri);`
  `/* ERRATO */`

- `wptr = (wchar_t *) malloc(numero_caratteri * sizeof(wchar_t)); /* CORRETTO */`

Utilizzo delle funzioni che lavorano su tipi di dati di 1 byte (memset(), …)

- `memset((void *) wptr, 0x0, numero_caratteri * sizeof(wchar_t));`
- `memset((void *) wbuf, 0x0, sizeof(wbuf));`