CU CS 4995-03: Internet Systems Programming

Internationalization

Overview

- commonly known as I18N
- same program for different languages, nations, cultures
- character sets
- messages
- number presentation
- collation

Locale

Locale = collection of definitions specific to country, language, …

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

LC_ALL	all categories
LC_CTYPE	character typing
LC_NUMERIC	decimal, thousands (. vs. ,)
LC_TIME	date and time format
LC_COLLATE	lexicographic sort order
LC_MONETARY	currency symbols
LC_MESSAGES	messaging functions

Examples of Locales

"C"	default, ASCII
"POSIX"	U.S. ASCII
"en_US"	English, U.S.
"de"	German

```
setlocale(LC_ALL, "en_US");
```

or, e.g., `$ locale yesexpr`

Other Locale Information

- time zone (TZ); generally automatic
- keyboard mapping (usually transparent)
- `printf("%2$s %1$s\n", "first", "second");`

Messages

- error messages: `gettext();` `xgettext` creates from C file
- message database, generated from `.po` file in (e.g.,) `en_US/LC_MESSAGES` directory:

```
msgid  "msg 1"
msgstr "The first message"

domain "help"
msgid "help 1"
msgstr "The first help message"
```

- convert to binary `.mo` file(s) with `msgfmt` utility
- set location with `NLSPATH` environment variable

Message

```
/* bind domain to directory */
b = bindtextdomain("messages", "/home/hgs/src/test/");
printf("bindtextdomain %s\n", b);

textdomain("messages");
setlocale(LC_MESSAGES, "en_US");
printf("English text: %s\n", gettext("msg 1"));

setlocale(LC_MESSAGES, "de");
printf("German text: %s\n", gettext("msg 1"));
```

Character Sets

- ASCII, 0–127
- ISO 8859-x, $x \in 1 \ldots 15$: 8-bit character sets for Latin, Cyrillic and Arabic alphabets (e.g., ä, ß, $\ldots$)
- 8859-1 is default character set for HTML; 0-127 is always ASCII
- Windows: similar to 8859-1, with additional symbols
- lots of variants for Korean, Japanese, Russian
- ISO 10646 ($\approx$ Unicode): 16 bits/character

ISO 8859-x

8859-1	Latin 1	W. European (French, German, Dutch)
8859-2	Latin 2	E. European (Czech, Hungarian, Polish)
8859-3	Latin 3	S. European (Esperanto, Turkish)
8859-4	Latin 4	N. European (Estonian, Greenlandic)
8859-5	Cyrillic	Bulgarian, Russian
8859-6	Arabic	but non Persian
8859-7	Greek	modern monotonic
8859-8	Hebrew	
8859-9	Latin 5	like Latin1, but Turkish
8859-10	Latin 6	similar to Latin4
8859-11	Thai	
8859-13	Latin 7	Baltic Rim
8859-14	Latin 8	Celtic languages
8859-15	Latin 9	updates Latin1, Euro

Wide Characters

- `#include <wchar.h>`
- hold Unicode ($\approx$ ISO 10646) characters: UCS-4 (4 bytes), UCS-2 (2 bytes)
- 0-127 = ASCII
- C constants: `L"x"`, `L'x'`
- UTF-8 = variable length representation

ISO 10646

- includes all major scripts in the world $\longrightarrow$ multilingual text
- fixed length, instead of escape sequences
- currently, 18,000 unassigned
- characters instead of glyphs (shapes)
- combining marks follow base character
- Han unification: 120,000 $\longrightarrow$ 20,902

ISO 10646

- UCS-4: four octets
- UCS-2: two octets = basic multilingual plane

Scripts supported

Arabic	Armenian	Bengali	Bopomofo
Cyrillic	Devanagari	Georgian	Greek
Gujarati	Gurmkhi	Han	Hangul
Hebrew	Hiragana	Kannada	Katakana
Latin	Lao	Malayalam	Oriya
Phonetic	Tamil	Telugu	Thai
Tibetan			

+ numbers, diacritics, punctuation; general, mathematical & technical symbols; dingbats, arrows, blocks, box drawing forms, and geometric shapes

Planned Characters

- Klingon, Ethiopic, Cherokee, Braille, Etruscan, Tengwar, basic Egyptian hieroglyphics, …
- language tagging (multi-lingual)
- collation algorithms

UTF-8

RFC 2044:

- UTF-8: variable-length representation of ISO 10646
- all new Internet protocols use UTF-8 for text
- 0000 0000 to 0000 007F (US-ASCII repertoire) $\longrightarrow$ 00–7F
- U.S. ASCII doesn't appear elsewhere
- first octet indicates number of octets in sequence
- character boundaries easily found from anywhere
- values FE and FF never appear

UTF-8

UCS-4 range (hex.)	UTF-8 octet sequence (binary)
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-001F FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF	1111110x 10xxxxxx ... 10xxxxxx

Conversion of UCS-4 to UTF-8

1. determine the number of octets required
2. prepare the high-order bits
3. bits marked x from lower-order bits into low order

Functions for Wide Characters

All standard string functions, e.g.:

```
int wprintf(const wchar_t *format, ...);
int iswalpha(wint_t);
int wscmp(char *, char *);
int wscoll(char *, char *);
```

Character conversions

8859-1 to UTF-8:

```
iconv_t cd = iconv_open("UTF-8", "8859-1");
strcpy(from, "Just ä string");
inleft = strlen(from);
outleft = sizeof(to);
i = iconv(cd, &from, &inleft, &to, &outleft);
printf("conv=%d (left=%d,%d): %s -> %s\n",
  i, inleft, outleft, from, to);
iconv_close(cd);
```