

QACADV_v1.0



Appendix - Internationalisation

Advanced C

transforming performance
through learning

Internationalisation

- **K&R C** was linked implicitly to American English and the Roman character-set, specifically the ASCII characters
- Now an implementation may support multiple locales, each with a different character set, date format, etc.
- There is always a default "C" locale which has generally accepted (USA/Latin) defaults
- Implementation may define others, but are not required by the standard
- **Support for "wide" (multi-byte) characters for non-Latin character-sets**
 - L prefix to character constants and strings
 - Various functions in library

Wide Characters and Strings

- The wide character type `wchar_t` is defined in `<stdlib.h>` and `<stddef.h>`
- Wide character values can be specified by preceding the character value with `L`
- Wide character strings can be specified by preceding the string with an `L`

```
wchar_t wideChar = L'a';
```

```
wchar_t * ws = L"hello world";  
wchar_t wt [ ] = L"howdy";  
wchar_t wu [ ] = { L'h', L'o', L'w', L'd', L'y', 0 };  
wchar_t wv [ ] = { 'h', 'o', L'w', 'd', L'y', '\0' };
```

The standard specifies that the language provides support for the following:

A single-byte set representing the minimal C character-set.

An additional single-byte set that is implementation specific.

A further multi-byte set that is implementation specific.

A multi-byte character is one which requires a 2 or more byte sequence. This is a natural requirement for non-Latin alphabets such as Kanji. The byte sequence representation relies on *state dependent encoding*, i.e. an interpretation which depends on bytes early on in the sequence. With this method, characters from both the conventional C character-set or from alternative single-byte sets can be easily represented. Each character is also associated with an integer representation, referred to as *wide character encoding* and the integer type used is typedef'd in `stddef.h` and named `wchar_t`.

The prefix `L` is used to specify a wide char constant, a string of wide chars and an array of wide chars.

Wide Character & Multi-Byte Functions

- **#include <stdlib.h>** (MB_LEN_MAX is in limits.h)

mblen()	length of a multi-byte string
mbtowc()	convert multi-byte character to wide character
mbstowcs()	convert multi-byte string to wide character string
wctomb()	convert wide character to multi-byte character
wcstombs()	convert wide character string to multi-byte string
MB_LEN_MAX	longest possible multi-byte sequence (all locales)
MB_CUR_MAX	longest possible multi-byte sequence (current locale)

Multi-byte sequences are used for general processing such as input and output. The wide char is merely a 16-bit integer used for convenience within the program. It is important to be able to convert between the two representations. A conversion between the multi-byte and wide character representation is provided by the standard functions `mblen`, `mbstowcs`, `mbtowc`, `wcstombs` and `wctomb` (found in `stdlib.h`) and by two macros `MB_LEN_MAX` and `MB_CUR_MAX` (which are found in `limits.h` and `stdlib.h` respectively).

Locale Functions

- **The function `setlocale()` sets up locale information**
 - The only ANSI standard locale is "C" (default)
- **Function `localeconv()` retrieves current numeric format information**
 - Returns a pointer to struct `lconv`
 - Numeric output functions may call `localeconv()` to determine locale specifics and behave accordingly
- **`#include <locale.h>`**

A "locale" summarises the conventions used by a particular spoken language, country, set of nations, and so on. Conventions include, for example, the way in which the time and date are formatted and monetary values displayed. The conventions are stored as members in a `struct` of type `lconv`. The locale is set to a particular convention at start up and can be changed during program execution.

The locale mechanism is supported through the header `locale.h` that defines `NULL`, several macros which all begin with "LC_", and `struct lconv`, which is to hold the conventions and the prototypes for the two functions `setlocale()` and `localeconv()`.

The name of the locale is held in a string which is used as a handle into the `setlocale` function. The default locale is named "C" - the only one defined in the standard.

The `localeconv` function returns a pointer to the appropriate `lconv` structure. The conventional settings for the current locale stored in this `struct` can then be accessed directly through this pointer. Many library functions make use of the `localeconv` for formatting information.

setlocale()

- **Locale categories: constants used by setlocale() function to specify which portion of a program's locale information will be changed**

```
char *setlocale ( int category, char * localename );
```
- **Locale name: string indicating the name of the locale**
 - ANSI standard specifies "C" locale as default
 - Other implementations may provide other standard locales
- **setlocale() returns a locale name or NULL if invalid**

The category refers to one of the LC_ macros defined in `locale.h`. These include some or all of the following:

LC_COLLATE	Affects collating used in <code>strcoll</code> and <code>strxfrm</code> .
LC_CTYPE	Affects character handling and multi-byte functions.
LC_MONETARY	Affects monetary information returned by <code>localeconv</code> .
LC_NUMERIC	Affects decimal point information returned by <code>localeconv</code> and decimal point processing used in numeric conversion and I/O functions.
LC_TIME	Affects time conversion in the <code>strftime</code> function.
LC_ALL	Affects all categories.

Inside struct lconv

- **Only decimal_point is used by standard functions; all other information is advisory only**

```
char *decimal_point ;      /* Non-monetary decimal-point character */
char *thousands_sep ;     /* Non-monetary digit-group separator */
char *grouping ;          /* Non-monetary digit-group size */
char *int_curr_symbol ;    /* International currency symbol */
char *currency_symbol ;    /* Current locale currency symbol */
char *mon_decimal_point ;  /* Monetary decimal-point character */
char *mon_thousands_sep ; /* Monetary digit-group separator */
char *mon_grouping ;       /* Monetary digit-group size */
char *positive_sign ;      /* Non-negative monetary sign */
char *negative_sign ;      /* Negative monetary sign */
char int_frac_digits ;     /* International monetary fractional digits */
char frac_digits ;         /* Monetary fractional digits */
char p_cs_precedes ;       /* Non-negative currency-symbol placement */
char p_sep_by_space ;      /* Non-negative currency-symbol separator */
char n_cs_precedes ;       /* Negative currency-symbol placement */
char n_sep_by_space ;      /* Negative currency-symbol separator */
char p_sign_posn ;         /* Sign placement in non-negative monetary value */
char n_sign_posn ;         /* Sign placement in negative monetary value */
```

The above list of members are not in any recognised order and can be added to or replaced by others - there is no standard set.

The `char*` members all point to character strings, e.g. "DM". The `char` members contain integer values or `CHAR_MAX` when not appropriate for a particular locale convention. Some of the `char` members represent intended integer values, e.g. `frac_digits` has the value 2, some are codes and others are booleans.

Locale Functions: Example

- Find name of current locale
- Get numeric format information

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

int main (void)
{
    char          *curr_locale ;
    struct lconv  *curr_numeric ;

    curr_locale   = setlocale(LC_ALL,  NULL) ;
    curr_numeric  = localeconv ( ) ;

    printf ("Current locale is %s\n", curr_locale) ;
    printf ("Decimal is %s\n",  curr_numeric->decimal_point) ;

    return 0;
}
```

This example shows how to determine the name of the current locale via the `setlocale()` function, and then determine the symbol representing a decimal point in the current locale using the `localeconv()` function. The name of the default locale is "C" and the character "." is used in the United Kingdom for a decimal point.

Other locale settings can be obtained in a similar way by inspecting the various members of the `struct lconv` variable.

Note that the header file `locale.h` must be included for the locale prototypes, structure template and `#define` values.