Changing the meaning of type `char` to handle larger character sets isn't possible, since `char` values are—by definition—limited to single bytes. Instead, C allows compilers to provide an ***extended character set.*** This character set may be used for writing C programs (in comments and strings, for example), in the environment in which the program is run, or in both places. C provides two techniques for encoding an extended character set: multibyte characters and wide characters. It also supplies functions that convert from one kind of encoding to the other.

Q&A

Multibyte Characters

In a ***multibyte character*** encoding, each extended character is represented by a sequence of one or more bytes. The number of bytes may vary, depending on the character. C requires that any extended character set include certain essential characters (letters, digits, operators, punctuation, and white-space characters); these characters must be single bytes. Other bytes can be interpreted as the beginning of a multibyte character.

Japanese Character Sets

The Japanese employ several different writing systems. The most complex, *kanji*, consists of thousands of symbols—far too many to represent in a one-byte encoding. (*Kanji* symbols actually come from Chinese, which has a similar problem with large character sets.) There's no single way to encode *kanji*; common encodings include JIS (Japanese Industrial Standard), Shift-JIS (the most popular encoding), and EUC (Extended UNIX Code).

Some multibyte character sets rely on a ***state-dependent encoding.*** In this kind of encoding, each sequence of multibyte characters begins in an ***initial shift state.*** Certain bytes encountered later (known as a ***shift sequence***) may change the shift state, affecting the meaning of subsequent bytes. Japan's JIS encoding, for example, mixes one-byte codes with two-byte codes; "escape sequences" embedded in strings indicate when to switch between one-byte and two-byte modes. (In contrast, the Shift-JIS encoding is not state-dependent. Each character requires either one or two bytes, but the first byte of a two-byte character can always be distinguished from a one-byte character.)

In any encoding, the C standard requires that a zero byte always represent a null character, regardless of shift state. Also, a zero byte can't be the second (or later) byte of a multibyte character.

The C library provides two macros, `MB_LEN_MAX` and `MB_CUR_MAX`, that are related to multibyte characters. Both macros specify the maximum number of bytes in a multibyte character. `MB_LEN_MAX` (defined in `<limits.h>`) gives the maximum for any supported locale; `MB_CUR_MAX` (defined in `<stdlib.h>`) gives the maximum for the current locale. (Changing locales may affect the interpretation of multibyte characters.) Obviously, `MB_CUR_MAX` can't be larger than `MB_LEN_MAX`.