prefixed by 110 and the six-bit group is prefixed by 10. For example, the code point for the character *ä* is E4 (hexadecimal) or 11100100 (binary). In UTF-8, it would be represented by the two-byte sequence 11000011 10100100. Note how the underlined portions, when joined together, spell out 00011100100.

Characters whose code points fall in the range 800–FFFF, which includes the remaining characters in the Basic Multilingual Plane, require three bytes. All other Unicode characters (most of them rarely used) are assigned four bytes.

UTF-8 has a number of useful properties:

- Each of the 128 ASCII characters is represented by one byte. A string consisting solely of ASCII characters looks exactly the same in UTF-8.
- Any byte in a UTF-8 string whose leftmost bit is 0 must be an ASCII character, because all other bytes begin with a 1 bit.
- The first byte of a multibyte character indicates how long the character will be. If the number of 1 bits at the beginning of the byte is two, the character is two bytes long. If the number of 1 bits is three or four, the character is three or four bytes long, respectively.
- Every other byte in a multibyte sequence has 10 as its leftmost bits.

The last three properties are especially important, because they guarantee that no sequence of bytes within a multibyte character can possibly represent another valid multibyte character. This makes it possible to search a multibyte string for a particular character or sequence of characters by simply doing byte comparisons.

So how does UTF-8 stack up against UCS-2? UCS-2 has the advantage that characters are stored in their most natural form. On the other hand, UTF-8 can handle all Unicode characters (not just those in the BMP), often requires less space than UCS-2, and retains compatibility with ASCII. UCS-2 isn't nearly as popular as UTF-8, although it was used in the Windows NT operating system. A newer version that uses four bytes (*UCS-4*) is gradually taking its place. Some systems extend UCS-2 into a multibyte encoding by allowing a variable number of byte pairs to represent a character (unlike UCS-2, which uses a single byte pair per character). This encoding, known as *UTF-16*, has the advantage that it's compatible with UCS-2.

Multibyte/Wide-Character Conversion Functions

```
int mblen(const char *s, size_t n);          from <stdlib.h>
int mbtowc(wchar_t * restrict pwc,
           const char * restrict s,
           size_t n);                        from <stdlib.h>
int wctomb(char *s, wchar_t wc);             from <stdlib.h>
```

Although the C89 standard introduced the concepts of multibyte characters and wide characters, it provides only five functions for working with these kinds of