The source code for a C program is stored in a text file, for example. In a ***binary file***, on the other hand, bytes don't necessarily represent characters; groups of bytes might represent other types of data, such as integers and floating-point numbers. An executable C program is stored in a binary file, as you'll quickly realize if you try to look at the contents of one.

Text files have two characteristics that binary files don't possess:

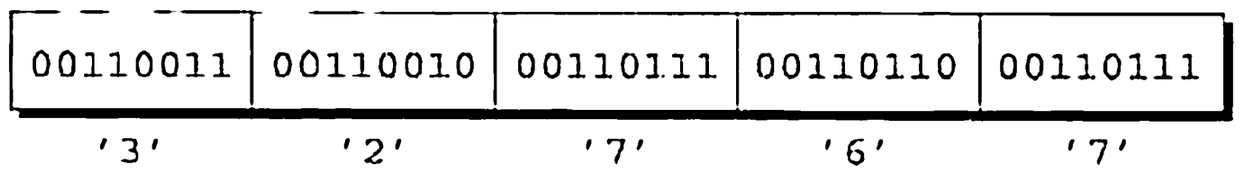
- ***Text files are divided into lines.*** Each line in a text file normally ends with one or two special characters; the choice of characters depends on the operating system. In Windows, the end-of-line marker is a carriage-return character (`'\x0d'`) followed immediately by a line-feed character (`'\x0a'`). In UNIX and newer versions of the Macintosh operating system (Mac OS), the end-of-line marker is a single line-feed character. Older versions of Mac OS use a single carriage-return character.

Q&A

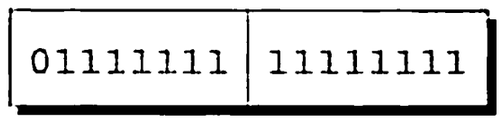
- ***Text files may contain a special "end-of-file" marker.*** Some operating systems allow a special byte to be used as a marker at the end of a text file. In Windows, the marker is `'\x1a'` (Ctrl-Z). There's no requirement that Ctrl-Z be present, but if it is, it marks the end of the file; any bytes after Ctrl-Z are to be ignored. The Ctrl-Z convention is a holdover from DOS, which in turn inherited it from CP/M, an early operating system for personal computers. Most other operating systems, including UNIX, have no special end-of-file character.

Binary files aren't divided into lines. In a binary file, there are no end-of-line or end-of-file markers; all bytes are treated equally.

Q&A When we write data to a file, we'll need to consider whether to store it in text form or in binary form. To see the difference, consider how we might store the number 32767 in a file. One option would be to write the number in text form as the characters 3, 2, 7, 6, and 7. If the character set is ASCII, we'd have the following five bytes:



The other option is to store the number in binary, which would take as few as two bytes:



little-endian order ➤*20.3* (The bytes will be reversed on systems that store data in little-endian order.) As this example shows, storing numbers in binary can often save quite a bit of space.

When we're writing a program that reads from a file or writes to a file, we need to take into account whether it's a text file or a binary file. A program that displays the contents of a file on the screen will probably assume it's a text file. A file-