Binary Vs Text File in C

From the programming angle there are three main areas where text and binary mode files are different. These are:

Handling of newlines

Representation of end of file

Storage of numbers

Let us explore these three differences:

In text mode, a newline character is converted into the carriage return-linefeed combination before being written to the disk. Likewise, the carriage return-linefeed combination on the disk is converted back into a newline when the file is read by a C program. However, if a file is opened in binary mode, as opposed to text mode, these conversions will not take place.

The second difference between text and binary modes is in the way the end-of-file is detected. In text mode, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file. If this character is detected at any point in the file, the read function would return the EOF signal to the program. As against this, there is no such special character present in the binary mode files to mark the end of file. The binary mode files keep track of the end of file from the number of character present in the directory entry of the file.

The last difference is storage of numbers. In text file the text and characters are stored one character per byte, as we expect. But numbers are stored as strings of characters. Thus, 65112, even though it occupies 4 bytes in memory, when transferred to the disk using fprintf(), would occupy 5 bytes, one byte per character. Here if large amount of data is to be stored in a disk file, using text mode may turn out to be insufficient. The solution is to open the file in binary mode and use those functions (fread() and fwrite())) which store the numbers in binary format. It means each number would occupy same number of bytes on disks as it occupies in memory.