The Endian Issue

Kaminski / cs3310

[Read this after reading about binary files, dump/HexEdit – and skimming over Wikipedia's coverage of "Endianess"]

The endian issue relates to how numeric data (int's, double's, . . .) is stored in memory (not directly relevant in CS3310) and thus how such data appear when written to a binary file. This is important when viewing a dump of a binary file (e.g., using a HexEditor or linux/unix' `od`) in CS3310 projects. This may also be relevant when binary files are transferred between systems (ftp, networks, uploaded/downloaded from web servers) since the 2 systems may use different endian conventions.

Endian-ness refers to byte-order for integer and floating point number storage. For example, an int type variable uses 4 bytes.

- BIG-Endian means that the MOST significant byte is first
  Humans and sparc/unix systems use this convention.
- LITTLE-Endian means that the LEAST significant byte is first (i.e., "reverse" order)
  [NOTE: This does NOT mean that the bits within the bytes are reversed,
  just the 4 bytes themselves appear in reverse order].
  Windows and linux systems use this convention.

A dump (using od) of a32-bit (4-byte) integer value of decimal 18 stored as an int then written to a binary file looks like:

```
022  \0  \0  \0
```
on a PC (Windows, Linux) [Little-Endian]

vs.

```
\0  \0  \0 022
```
on a sparc/unix system [Big-Endian]

where

`\0` is the ASCII char for the byte 0000 0000 (in bits)

`022` is displayed for the byte 0001 0010 (in bits) because that's an unprintable-ASCII char, so the octal value 022 is shown instead.

NOTE: The bits within each byte are not "reversed", just the order of the 4 bytes –
e.g., for the hex integer 01234567:

a PC (Windows/Linux) hex dump (HexEdit) would show: `67 45 23 01`

a sparc (unix) hex dump would show: `01 23 45 67`

[NOTE: This issue does NOT apply to char strings of digits (i.e., their ASCII codes)].

This issue for binary files is due to how different systems store integers (& floating point numbers) in memory. And the binary file just contains a copy of exactly what is in memory when a the variable (or the struct containing the variable) is written to a binary file. [Of course, when writing to an ASCII text file, the numeric data is converted to an set of digits (char's), so a normal big-endian (human) convention is used].