Binary File Input Output

Binary and Text Mode

C has the ability to read and write files containing arbitrary binary data: `fopen()` can either open a file in text mode (the default), or in binary mode (when the mode parameter contains a “b”, as in "rb" or "wb")

A binary file can contain arbitrary binary data

In this circumstance there’d have to be a way of knowing how big the string is, perhaps a leading byte count, or perhaps the binary file structure would mandate a 32 byte string at a particular point in the file

If you need to be certain that the file content matches the data you write to it byte-for-byte, especially if you need to write zero bytes, use binary mode. Otherwise use text mode

Reading and Writing Binary Data

For reading and writing binary data, we use the following two functions:

```
size_t fread( void *ptr, size_t size, size_t nmemb, FILE *stream );

size_t fwrite( const void *ptr, size_t size, size_t nmemb, FILE
*stream );
```

- `ptr` is the address of the data to read or write
- `size` is the size of each element in bytes
- `nmemb` is a count of elements to read/write
- `stream` is the file being read/written to

Each function returns the number of elements read/written

A value less than `nmemb` indicates an error

You’ll find four examples of binary file manipulation in the code/fileio directory: (write/read) double array binary.c which writes out a large array of doubles (and

then reads it back in again), and (write/read) struct binary.c which writes out a structure which includes two large array members (and then reads it back in again)

Binary files are usually non-portable - especially reading raw binary values as in the above. But they may be faster to read/write on a single platform

Random Access IO

We have the ability to seek to random locations within (most) files using `fseek()`:
`int fseek( FILE *stream, long offset, int origin );`

- `offset` – The new position, specified as an offset in bytes from the supplied origin
- `origin` – The location from which the new position is calculated
  - `SEEK_SET` – The start of the file
  - `SEEK_CUR` – The current position
  - `SEEK_END` – The end of the file

`fseek()` returns -1 on failure. Note that you can't seek on `stdin`, `stdout` or `stderr`

There is a related function `long ftell( FILE *stream );` that reports the current file position (offset from the start of the file)

These two functions may be combined to find the size of an open file: `fseek` to the end, then `ftell`, then seek back to the beginning (via `fseek(stream,0L,SEEK_SET)` or `rewind(stream)`)