Semantics (VII)

File Access

- `fopen`:
  - Before it can be read or written, a file has to be opened
  - `fopen` takes an external name like `x.c` does some housekeeping and negotiation with the operating system, and returns a pointer to be used in subsequent reads or writes of the file.
  - **`FILE *fopen(char *name, char *mode)`**
  - The pointer called the file pointer
    - Points to a structure that contains information about the file, such as the location of a buffer, the current character position in the buffer, whether the file is being read or written, and whether errors or end of file have occurred.
    - Users don't need to know the details, since the definitions obtained from `<stdio.h>` include a structure declaration called `FILE`.
    - The only declaration needed for a file pointer is exemplified by **`FILE *fp;`**
    - Notice that **`FILE` is a type name**, like `int`, not a structure tag; it is defined with a `typedef`.
  - The call to `fopen` is **`fp = fopen(name, mode)`**

- `mode`
  - a character string indicates how one intends to use the file
  - read (“`r`”), write (“`w`”), and append (“`a`”)
  - some systems distinguish between text and binary files; for the latter, a “`b`” must be appended to the mode string. E.g. “`rb`” means reading binary file.
  - If a file that does not exist is opened for writing or appending, it is created if possible.
  - Opening an existing file for writing causes the old contents to be discarded, while opening for appending preserves them.
  - Trying to read a file that does not exist is an error. Trying to read a file when you don't have permission is also an error. If there is any error, open will return NULL.

  - `int fclose(FILE *fp)`
    - Free the file pointer for another file
    - Since most operating systems have some limit on the number of files that a program may have open simultaneously, it is a good idea to free the file pointers when they are no longer needed.