6. Files and Streams

Streams

An input stream is a source of data that provides input to our program, and an output stream is a destination for data output by the program. The streams standard input, standard output, and standard error are automatically open when a program runs. Streams can also refer to disk files, network connections, and computer devices. Streams can be redirected to change the input or output locations. This is feature of the operating system.

Files

The unix system calls to manage files are `open`, `read`, `write`, `close`. However, these are OS-dependent. For this reason, the `stdio` library is used for file access in C. `FILE` is an abstract data type declared in `stdio.h`. It is portable to non-unix operating systems and supports buffering.

Buffering

It is slow to perform a system call for little pieces of data. The `stdio` library reads a large amount of data even if has not been called for yet, and saves the excess data in a buffer until it is called for later. Similarly, when writing data, it saves a large amount of data. This is so that the system calls need to be called less often.

Library Functions for Streams

`fopen` and `fclose`

The prototype of `fopen` is `FILE *fopen(const char * restrict path, const char * restrict mode);`

`fopen` opens the file whose name is the string pointed to by `path` and associates a stream with it. `mode` points to a string beginning with one of the following letters - `"r"` for reading (fails if file does not exist), `"w"` for writing (truncates a file to zero length if it exists and creates it otherwise), and `"a"` for appending (creates a file if it does not

exist; writes always take place at the end of the file regardless of `fseek` calls). An optional `"+"` following any of the aforementioned letters opens the file for both reading and writing. `mode` can also include the letter `"b"` after either the `"+"` or the first letter. Any created files will have mode `S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH`

Upon successful completion `fopen` returns a FILE pointer. Otherwise, `NULL` is returned and `errno` is set to indicate the error.

The prototype of `fclose` is `int fclose(FILE *stream);`

`fclose` dissociates the named stream from its underlying file or set of functions. If the stream was being used for output, any buffered data is written first, using `fflush`.

`fread` and `fwrite`

The prototype of `fread` is `size_t fread(void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);`

The prototype of `fwrite` is `size_t fwrite(const void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);`

`fread` reads `nitems` objects, each `size` bytes long, from the stream pointed to by `stream`, storing them at the location given by `ptr`.

`fwrite` writes `nitems` objects, each `size` bytes long, to the stream pointed to by `stream`, obtaining them from the location given by `ptr`.

`fread` and `fwrite` advance the file position indicator for the stream by the number of bytes read or written. They return the number of objects read or written. If an error occurs, or the end-of- file is reached, the return value is a short object count (or zero).

`fread` does not distinguish between end-of-file and error. `fwrite` returns a value less than `nitems` only if a write error has occurred.

`fprintf`, `fscanf`, and `sscanf`

`fgets`

The prototype of `fgets` is `char * fgets(char * restrict str, int size, FILE * restrict stream);`

`fgets` reads at most one less than the number of characters specified by `size` from the given stream `stream` and stores them in the string `str`. Reading stops when a newline character is found, at end-of-file or error. The newline, if any, is retained. If any characters are read and there is no error, a `'\0'` character is appended to end the string.

Upon successful completion, `fgets` returns a pointer to the string. If end-of-file occurs before any characters are read, it returns `NULL` and the buffer contents remain unchanged. If an error occurs, it return `NULL` and the buffer contents are indeterminate.

```
char str[50]; char *p;
fgets(str, 50, stdin);
if((p = strchr(str, '\n'))) *p = '\0';
```

`fseek`, `ftell`, and `rewind`

The prototype of `fseek` is `int fseek(FILE *stream, long offset, int whence);`

The `fseek` function sets the file position indicator for the stream pointed to by `stream`. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by `whence`. If whence is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

`fseek` returns the value `0` if successful; otherwise the value `-1` is returned and the `errno` is set to indicate the error.

The prototype of `ftell` is `long ftell(FILE *stream);`

`ftell` obtains the current value of the file position indicator for the stream pointed to by stream.

Upon successful completion, `ftell` returns the current offset. Otherwise, `-1` is returned and `errno` is set to indicate the error.

The prototype of `rewind` is `void rewind(FILE *stream);`

`rewind` sets the file position indicator for the stream pointed to by `stream` to the beginning of the file. It is equivalent to `fseek(stream, 0L, SEEK_SET);`.

Since `rewind` does not return a value, an application wishing to detect errors should clear `errno`, then call `rewind`, and if `errno` is non-zero, assume an error has occurred.

`fflush`

The prototype of `fflush` is `int fflush(FILE *stream);`

`fflush` synchronises the state of the given stream in light of buffered I/O. For output or update streams it writes all buffered data via the stream's underlying write function. For input streams it seeks to the current file position indicator via the stream's underlying seek function. The open status of the stream is unaffected.

If the stream argument is `NULL`, `fflush` flushes all open streams.

Upon successful completion `0` is returned. Otherwise, `EOF` is returned and the global variable errno is set to indicate the error.

`getchar` and `getc`

`getchar`

- Reads one byte from the standard input.
- Returns an `int` from `0` to the maximum unsigned byte value.
- Returns `-1` to indicate end of file.
- Syntax - `getchar();`

`getc`

- Identical to `getchar` except accepts a pointer to the file to read from.
- Syntax - `getc(FILE*);`

`putchar`

- Outputs one byte to the standard output.

- Returns the byte outputted on success.
- Returns `EOF` on failure
- Syntax - `putchar(byte);`

`putc`

- Identical to `putchar` except accepts a pointer to the file to output to.
- Syntax - `putc(byte, FILE*);`

`fprintf`

- Identical to `printf` except accepts a pointer to the file to output to.
- Returns the number of characters written on success.
- Returns a negative number on failure.
- Syntax - `fprintf(FILE*, "...");`

`fscanf`

- Identical to `scanf` except accepts a pointer to the file to read.
- Returns the number of input items successfully matched and assigned on success.
- Returns `0` of failure.
- Syntax - `fscanf(FILE*, "...");`

`sscanf`

- Identical to `scanf` except reads a string instead of the standard input.
- Syntax - `sscanf(str, "...");`
