【C】Day5(2)

Course	Advanced C
Study Date	@April 11, 2022

【Ch7】Input and Output

7.5 File Access

The next step is to write a program that accesses a file that is not already connected to the program.

Before a file can be read or written, a file has to be opened by the library function `fopen`. `fopen` takes an external name like `x.c` or `y.c`, returns a pointer to be used in subsequent reads or write of the file.

This 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.

The only declaration needed for a file pointer is exemplified by

```
FILE *fp;
FILE *fopen(char *name, char *mode);
```

Note: FILE is a type name, like int, not a structure tag; it is defined with a `typedef`.

The call to fopen in a program is:

```
fp = fopen(name, mode);
```

The first argument of `fopen` is a character string containing the name of the file.

The second argument is the mode, also a character string ,which indicates how one intends to use the file. Allowable modes include 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.

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, and there may be other causes of error as well, like trying to read a file when we don’t have permission.

If there is any error, `fopen` will return NULL.

The next thing needed is a way to read or write the file once it is open. There are several possibilities, of which `getc` and `putc` are the simplest.

`getc` returns the next character from a file; it needs the file pointer to tell it which file.

```
int getc(FILE *fp)
```

`putc` is an output function

```
int putc(int c, FILE *fp)
```

`putc` writes the character c to the file `fp` and returns the character written, or `EOF` if an error occurs.

Like `getchar` and `putchar`, `getc` and `putc` may be macros instead of functions.

When a C program is started, the operating system environment is responsible for opening three files and providing file pointers for them. These files are the standard input(`stdin`), the standard output(`stdout`), and the standard error(`stderr`), and are declared in `<stdio.h>`

Normally, `stdin` is connected to the keyboard and `stdout` and `stderr` are connected to the screen, but `stdin` and `stdout` may be redirected to files or pipes.

`getchar` and `putchar` can be defined in terms of `getc`, `putc`, `stdin`, and `stdout` as follows:

```
#define getchar() getc(stdin)
#define putchar(c) putc(c, stdout)
```

For formatted input or output of files, the functions `fscanf` and `fprintf` may be used. These are identical to `scanf` and `printf`, except that the first argument is a file pointer that specifies the file to be read or written; the format string is the second argument

```
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)
```