[System Programming] Assignment #5

Spring 2025

Seulki Lee
Yunseok Lee
Kyu Hwan Lee
Jiwoon Chang
Yeojin Lee

CONTACT
Ulsan National Institute of Science and Technology
Address 50 UNIST-gil, Ulju-gun, Ulsan, 44919, Korea
Tel. +82 52 217 0114 **Web.** www.unist.ac.kr

Computer Science and Engineering
106 3rd Engineering Bldg
Tel. +82 52 217 6333 **Web.** https://cse.unist.ac.kr/

Assignment #5 (100 points)

- Due June 13, 2025: 11:59pm

- Platform
  - We will work on Ubuntu 22.10 (latest version)
  - https://releases.ubuntu.com/kinetic/ (Desktop image)

- If you use MAC, please use Docker Desktop on Mac
  - https://docs.docker.com/desktop/install/mac-install/

- If you cannot make this environment, please contact our TA
  - Yunseok Lee: walk1009@unist.ac.kr
  - Kyu Hwan Lee: hanbitchan@unist.ac.kr
  - Jiwoon Chang: jwc9876@unist.ac.kr
  - Yeojin Lee: yeojin@unist.ac.kr

Goal

1. Design your own core input and output functions of the C standard I/O library.

2. Implement the C I/O functions by using UNIX I/O functions.

3. Consider <u>permission</u>, while using a file.

C I/O Functions we need to implement

1. **myfopen:** opens a file (with specific mode)
2. **myfclose:** closes a file
3. **myfseek:** moves the file position to a specific location in a file
4. **myfread:** reads from a file
5. **myfwrite:** writes to a file
6. **myfflush:** synchronizes a stream with an actual file
7. **myfgets:** retrieving a string from a file
8. **myfputs:** writing a string to a file

myfopen()

```
myFILE *myfopen(const char *pathname, const char *mode);
```

The **myfopen**() function opens the file whose name is the string pointed to by **pathname** and associates a stream with it.
- Only a single input mode is allowed

Return value

Upon successful completion, return a *myFILE* pointer. Otherwise, NULL is returned.

Ex) myfopen("/bin/sh", "r+");

myfopen()

*mode

‘r’ & ‘r+’ mode returns NULL, when there is no such file

r	Open text file for reading. The stream is positioned at the beginning of the file.
r+	Open for reading and writing. The stream is positioned at the beginning of the file.
w	Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
w+	Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
a	Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
a+	Open for reading and appending (writing at end of file). The file is created if it does not exist. Output is always appended to the end of the file. POSIX is silent on what the initial read position is when using this mode, but, the stream must be positioned at the end of the file in this work.

myfclose()

int myfclose(myFILE **stream*);

The ***myfclose()*** function shall cause the stream pointed to by *stream* to be flushed and the associated file to be closed. Any unwritten buffered data for the stream shall be written to the file; any unread buffered data shall be discarded.

`

myfclose()

Return value

Upon successful completion, ***myfclose()*** shall return 0; otherwise, it shall return EOF.

myfseek()

int myfseek(myFILE **stream*, int *offset*, int *whence*);

The ***myfseek()*** function shall set the file position indicator for the stream pointed to by *stream*. The new position, measured in bytes from the beginning of the file, shall be obtained by adding *offset* to the position specified by *whence*. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

myfseek()

Return value

The ***myfseek()*** and function shall return 0 if they succeed. Otherwise, they shall return -1.

myfread()/myfwrite()

```
int myfread(void *ptr, int size, int nmemb, myFILE *stream);
int myfwrite(const void *ptr, int size, int nmemb, myFILE *stream);
```

The function **myfread**() reads *nmemb* items of data, each *size* bytes long, from the stream pointed to by *stream*, storing them at the location given by *ptr*.
The function **myfwrite**() writes *nmemb* items of data, each *size* bytes long, to the stream pointed to by *stream*, obtaining them from the location given by *ptr*.

myfread()/myfwrite()

Return value

On success, **myfread()** and **myfwrite()** return the number of items read or written. This number equals the number of bytes transferred only when *size* is 1. If an error occurs, or the end of the file is reached, the return value is zero. The file position indicator for the stream is advanced by the number of bytes successfully read or written.

myfflush()

int myfflush(FILE *stream);

For writable streams, **myfflush()** forces a *write* of buffered data in *wrbuffer* to the file if the last operation was *write*.
For readable streams, **myfflush()** discards any buffered data in *rdbuffer* that has been fetched from the underlying file, but has not been consumed by the application. The open status of the stream is unaffected.

myfflush()

Return value

Upon successful completion **0** is returned. Otherwise, **EOF** is returned.

myfputs()

int myfputs(const char *str, myFILE* stream);

myfputs() writes the null terminated string str to stream, without its terminating null byte ('\0’).
myfputs() continues to write data until a terminating null byte is reached.

myfputs()

Return value

Upon successful completion **0** is returned. Otherwise, **EOF** is returned.

myfgets()

Char* myfgets(char* str, int num, myFILE *stream);

myfgets() reads up to **num-1** characters from the given stream and stores them into the buffer pointed to by str. It stops when a **newline character** is found, or the **end-of-file** is reached, whichever comes first. A null **character '\0' is written** immediately after the last character read into the buffer. If a **newline** is read, it is stored into the buffer.

myfgets()

Return value

On success, the function returns str. If no characters were read and the stream has reached the end-of-file, NULL is returned.

File Format for grading

1. **File Type: Text File (not directory)**
2. **File Size: 0 to 2^31-1 bytes**
3. **Path Name: 0 to 512 bytes (Can be NULL)**
4. **Permission: can be anything (using rwx)**

What You Need To Do

1. **Use the skeleton code offered to you.**
2. **Implement after "Implement after here" in the skeleton code.**
3. **<u>Do Not Use</u> the default "stdio.h". This will make your point as 0 at grading.**

Examples

You can verify whether the custom I/O functions are working properly using ‘main.c’.

Here are some examples for this.

Example 1

```
#include "mystdio.h"

int main() {
        myFILE *file = myfopen("test_flush.txt", "w");
        myfputs("This is a test.", file);
        myfflush(file);
        myfclose(file);
        return 0;
}
```

Example 2

```
#include "mystdio.h"

int main() {
		myFILE *file = myfopen("test_seek.txt", "r+");
		char buffer[BUFSIZE];

		myfread(buffer, sizeof(char), BUFSIZE, file);
		myfseek(file, 0, SEEK_SET); // Rewind to the start of the file
		myfread(buffer, sizeof(char), BUFSIZE, file);
		printf("%s", buffer);
		myfclose(file);
		return 0;
}
```

Example 3

```
#include "mystdio.h"

int main() {
		myFILE *file = myfopen("test_read_write.txt", "w+");
		const char *text = "Sample text to write to file.";

		myfwrite(text, sizeof(char), strlen(text), file);
		myfseek(file, 0, SEEK_SET); // Go back to the start of the file

		char buffer[BUFSIZE];
		myfread(buffer, sizeof(char), strlen(text), file);
		printf("Read from file: %s\n", buffer);
		myfclose(file);
		return 0;
}
```

Q&A

1. Consider only valid inputs except Permission and File Format Error related inputs
2. Only use offered headers (Do not include other headers in the skeleton code.)
3. Use C (not C++ grammar)
4. Rename or Redefining function (i.e., function name, parameter type, return type): Not allowed
5. Add or Sub members inside myFILE structure for yourself: allowed
6. wrbuffer: You don't need to implement rdbuffer(If you want, you may implement this), Instead, use wrbuffer when implementing file write functions such as fwrite, fputs.

Reference

1. **Check this URL, if you want to better understand how standard C I/O functions work.**

https://www.youtube.com/watch?v=HQNsriyMhtY

2. Check this URL to understand basics of Linux file permissions.

https://linuxize.com/post/understanding-linux-file-permissions/

Error Handling

1. **Permission Error : Print “Permission Denied”, when the file doesn’t have specific permission**

2. **File Format Error : If the file does not meet given file format, print “File Format Error”**

!!! What you need to submit !!!

1. **The header file: “mystdio.h”**
2. **Code Description file: “report.pdf”**

But, compress them in a <u>zipfile</u>!
Your *zipfile* name must be your student ID#_name_assn5.
e.g., 20219876_name_assn5.zip
- including ”mystdio.h” and “report.pdf”

- You should submit your code with a code description that explains your code (e.g., comments in the file). In the description, your code must be well commented to explain your algorithm.

If you did not follow the rule, your point will get a penalty point (-3).