Lab: Myshell - Impl. your own shell in C

In this lab, you are asked to implement your own library to support the shell commands. We did an exercise in the last class where you implement basic `echo` command using File I/O functions in standard C library. In this lab, it is an extension to that exercise and you will implement shell commands for `cat`, redirection, `touch`.

We call our shell commands with prefix `my_` and the following is what the outcome of this lab will look like. Note that we use `@` instead of `>` to represent redirection in `myshell`.

```
./my_cat file1
./my_echo Alice
./my_echo Bob @ file1
./my_echo Alice @@ file1
./my_touch file1
```

This lab consists of the following tasks:

1. Set up the project framework using Makefile
2. Implement `my_printf` using file I/O
3. Implement `my_echo`
4. Implement `my_echo` with redirection
5. Implement `my_cat`
6. Implement `my_touch`

1.Framework Setup

This project will include two types of source files: library program and main program. The library program will realize the core functions for `myshell` and the main program translates the `myshell` commands to the calls of these C functions.

```
./main_printf.c
./main_echo.c
./main_cat.c
./main_touch.c
./Makefile
./myshell.c
./header.h
```

- Create a directory for this project.
- Write a Makefile that is compatible with the file organization as above.
    - `make echo` will compile `main_echo.c` and `myshell.c`.
    - `make echo` will generate executable `my_echo` and runs it by `./my_echo Alice @@ file1`

Makefile

```
SRCS = main_printf.c main_echo0.c main_echo.c main_cat.c myshell.c
OBJS = $(SRCS:.c=.o)
CFLAGS = -g -I.

printf: $(OBJS)
    $(CC) main_$@.o myshell.o
    ./a.out

echo0: $(OBJS)
    $(CC) main_$@.o myshell.o -o my_echo
    ./my_echo Alice

echo: $(OBJS)
    $(CC) main_$@.o myshell.o -o my_echo
    -rm file1 file2
    ./my_echo Alice @ file1
    ./my_echo Bob @@ file1
    cat file1
```

```
    ./my_echo Charlie @ file1
    cat file1
    ./my_echo David @@ file2
    cat file2

cat: $(OBJS)
    $(CC) main_$@.o myshell.o -o my_cat
    ./my_cat file1

clean:
    rm *.o *.out
```

2.Implement `my_printf`

- Write a function to find the length of an array `int my_strlen(char* str)`
- Write a library function `void my_printf(char* str)` in `myshell.c`
- Write a main function in `main_printf.c`
    - The goal is to call `my_printf("Hello\n");` in `main_printf.c`
- Put the following in file `header.h`

```c
#if !defined(HEADER_H)
#define HEADER_H
#include<stdio.h>
#include<unistd.h> //lseek, STDIN_FILENO
#include<stdlib.h>
#include <fcntl.h>
#define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int my_strlen(char* format_string);
void my_echo(int fd, char* str);
void my_printf(char* format_string);

#endif
```

3.Implement `my_echo`

- Write a library function `void my_echo(int fd, char* str)` in `myshell.c`
- Write a main function in `main_echo.c`
    - The goal is to run command `./my_echo Alice`

4.Implement `my_echo` with redirection

- Write a library function `void my_echo(int fd, char* str)` in `myshell.c`
- Write a main function in `main_echo.c`
    - The goal is to run command `./my_echo Alice @ file1`
    - and command `./my_echo Bob @@ file1`

5.Implement `my_cat`

- Write a library function `void my_cat(char* filename)` in `myshell.c`
- Write a main function in `main_cat.c`
    - The goal is to run command `./my_cat file1`

6.Implement `my_touch`

- Write a library function `void my_touch(char* filename)` in `myshell.c`
- Write a main function in `main_touch.c`
    - The goal is to run command `./my_touch file1`

Programming references

- File I/O functions

```
#include <fcntl.h>
open(char* pathname, int oflag, mode_t mode);
//open("file1", O_WRONLY | O_CREAT | O_APPEND, FILE_MODE);
int creat(const char * pathname, mode_t mode);
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbytes);
ssize_t read(int fd, void *buf, size_t nbytes);
off_t lseek(int fd, off_t offset, int whence);
```