In this assignment, you will implement a command line interpreter or, as it is more commonly known, a **shell**. The shell should operate in this basic way:

1. You type in a command (in response to its prompt).
2. Shell should parse command.
3. It create a child process that executes the command you entered. New process should be created for each command except for some inbuilt commands.
4. Finally it prompts for more user input when it has finished executing previous process.

Shell Functionality:

Basic Shell:

Your basic shell is basically an interactive loop: it repeatedly prints a prompt "mysh> " (note the space after the percent sign). This is repeated until the user types "exit". The name of your final executable should be mysh :

```
prompt> ./mysh
mysh>
```

For example, if the user types "ls -la /tmp" , your shell should run the program ls with all the given arguments and print the output on the screen.

Note that the shell itself does not "implement" ls or really many other commands at all. All it does is find those executables and create a new process to run them.

The maximum length of a line of input to the shell is 512 bytes (excluding the carriage return).

Built in Commands:

In this project, you should implement cd , pwd , ls, wait and exit. The formats for these functions are:

```
[optionalSpace]exit[optionalSpace]
[optionalSpace]pwd[optionalSpace]
[optionalSpace]cd[optionalSpace]
[optionalSpace]cd[oneOrMoreSpace]dir[optionalSpace]
[optionalSpace]wait[optionalSpace]
[optionalSpace]ls[optionalSpace]
```

Redirection:

Your shell should support redirection that is instead of displaying output on screen it should send it to output file like:

```
ls -la /tmp > output
```

Nothing should be printed on the screen. Instead, the output of the ls program should be rerouted to the output file. “>” is used for redirection.

If the "output" file already exists before you run your program, you should simple overwrite the file. If the output file is not specified (e.g. "ls > "), print an error message.

Here are some redirections that should not work:

```
ls > out1 out2
ls > out1 out2 out3
ls > out1 > out2
```

Batch Mode:

Your program should implement batch mode. In batch mode you specify all your commands in a file instead of writing them on command line. The path of batchfile is specified at runtime like:

./mysh [batchFile] Any errors should be printed on prompt simultaneously.

Python Scripts

When you type in the name of a python file, your shell should recognize this and instead run the python interpreter (e.g., /usr/bin/python) with the file name as an argument. For example, typing:

```
prompt> hello.py
```

would actually run the python interpreter, with hello.py as the argument.

Program Errors:

You should print this one and only error message whenever you encounter an error of any type:

```
char error_message[30] = "An error has occurred\n";
write(STDERR_FILENO, error_message, strlen(error_message));
```

Print error in case of following situations:

- An incorrect number of command line arguments to your shell program.
- A command does not exist or cannot be executed.
- A very long command line (over 512 characters, excluding the carriage return)

Your shell should also be able to handle the following scenarios below, which are not errors .

- An empty command line.
- Multiple white spaces on a command line. For example: mysh> ls , mysh> ls > a , mysh> ls>a