I/O Redirection and Filters in Shell

[Description] [Do Nots] [Hints] [Grading Guideline]

Description

This assignment helps you learn about I/O redirection and inter-process communication. You are asked to write a simple shell program called `mysh`.

Here is a sample code to get you started. It reads a line of input, breaks it up into individual tokens, and then waits for next line of input. Typing "exit" terminates the shell.

This shell must work as follows. You start the shell by running `mysh` program. This will give a prompt of your shell as follows:

```
mysh>
```

From here onwards, you should be able to execute and control **any program/command** just as you would in a normal shell. For instance

```
mysh> Command arg1 arg2 ... argN
[ Output of Command shown here ]
mysh>
```

Additionally, your shell should be able to do the following:

1. Redirect the input of a command from a file. For example:

```
mysh> Command < input_file
```

3. Redirect the output of a command to a file. For example:

```
mysh> Command > output_file
```

5. Implement filters, i.e., redirect the stdout of one command to stdin of another using pipes. For example:

```
mysh> ls -l | wc -l

mysh> cat somefile | grep somestring | less
```

   Ideally, your shell should be able to handle any number of filter components. **Hint:** Use of recursion is recommended but not mandatory.

7. Combine filters and file redirection. For example:

```
mysh> Command1 | Command2  > output_file
```

```
9.          mysh> Command1 < input_file  | Command 2
10.                 mysh> Command1 < input_file  | Command 2 | Command 3 >
   output_file
11.                 [Or any such valid combinations]
```

Do Nots:

- DO NOT use any special wrapper libraries or classes to borrow the basic functionality required in this assignment. If in doubt, ask the instructor first BEFORE doing so.
- DO NOT use the **system(...)** syscall to execute the programs in your shell directly.
- DO NOT write five or six different programs, one for each feature. Write **one single program** that includes all the above features.

Hints:

- Build and test one functionality at a time.
- Make backup copies of partially working versions of your code. This way, if you irreparably screw up your code, then you can at least roll back to your last backup copy.
- First design your data structures and code-structure before you begin coding each feature. Anticipate specific problems you will face.
- Check out man page for the following:
    - fork()
    - execv(), execl(), execlp(), execvp() (which one should you use?)
    - waitpid()
    - dup2() (for stdin/stdout redirection)
    - pipe()
    - open()
    - close()

Grading Guidelines

This is how we will grade your assignment. So please prioritize your work accordingly.

- 10 - Executing a single command without arguments
- 10 - Executing a single command with arguments
- 10 - Input redirection from file: Executing a single command that takes standard input (stdin) from a file

- 10 - Output redirection to file: Executing a single command that sends standard output (stdout) to a file
- 20 - Filters: Execute a filter chain with two commands
- 20 - Filters: Execute a filter chain with more than two commands
- 10 - Combination: Executing a filter chain while redirecting the input of first command from a file and/or output of last command to a file
- 10 - README, Makefile, Compilation without errors, Code Modularity, and Error Handling: Don't write long big functions that do everything. Write small functions that call other small functions. Check and handle the errors returned by ALL systems calls used in your program. Also check for other common error conditions in your program. But don't go overboard with error checking. We will not try to intentionally break your code with bad input that may be irrelevant to the assignment's goals.
- Total = 100