Operating Systems
HW: extend the shell

In this homework, you will finish and further extend the shell you have been working on in the labs. With the simple shell, `simple_shell.c`, as a starting point, you should add the following extensions, making sure that all these features work together, except in the cases noted below.

Directly from the lab exercise:

- Extend the shell to handle commands with multiple arguments, with a max of 100 tokens on a line.
- Extend the shell so that if the very last token on a command line is "`&`", the command runs in the background, rather than the foreground.

Additions to items from the lab exercise:

- Extend the shell so that it can make a backup of its entire execution. If the shell is called with an argument, interpret it as a filename. *Append the shell prompts, the user-entered commands themselves, as well as their output, to this file.* When the shell is finished, this file will contain a backup of the full user interaction that took place. *Be sure to have the command output go to the screen, as well as to the backup file, just as though there was no backup copy being made at all.*
    - **Hints**: Consider doing this by using temporary file(s), or by using a pipe (see manpage for the system call `pipe`) to the UNIX command `tee` (see manpage)

- Extend the shell to allow general I/O redirection *(including <, > and >>) of `stdout`/`stdin` on each line.* If redirection takes place, you may assume that those directives will be at the end of the line (except for &, which will come last if it is there). Assume there will be no more than one input redirection and one output redirection together on the same line.

New functionality (do this last after everything else works):

- Extend the shell to allow using the command-line pipe "`|`" between commands on the same line, where pipe means feeding the `stdout` of the command before to the `stdin` of the command after. You can assume that no I/O redirection or backgrounding occurs in the case where a line contains one or more pipes; however, commands in the pipeline may have arguments. Assume there are no more than 10 commands piped together. See the hints above to help you solve this.

Since you started this work in the lab, you may have collaborated with your lab classmates on the solution you submitted for the lab (even though you all submitted an independently-typed solution). All collaboration must stop from the point that this homework assignment is posted.

There should be some reasonable error handling that prevents your shell from acting crazy. Permissions of files handled by the shell should be checked and appropriate error messages should be printed. You can assume that the command lines are all correctly formatted, with a space between each token, and obeying the rules set above. However, make no assumptions about what the tokens are and whether or not the commands are called legally.

Some sample legal command line inputs that we may test with:

```
more a.c b.c c.c e.c f.c
ls >> out.txt
cat < file.c > new_file.c
diff a.c b.c | wc –l | tee out.tmp
meow –cat -dog
```