Homework #4: **my_shell** (100 pts)
Submit a compressed (.tgz) file with **source code** and **Makefile**

- Your job is to write **C++ code** that implements a *simplified* version of a UNIX **shell** (called **my_shell**).
  - **my_shell** will present a command line prompt to the user (on *stdout*)
  - Much like any UNIX shell program, **my_shell** will execute commands entered by the user (on *stdin*) and then return to the command prompt.
  - **my_shell** will *exit* when the user enters **exit**
  - **my_shell** must support *file redirection.*
    - Redirecting a *file* to a process's **stdin** (using **<**),
    - Redirecting process **stdout** to a *file* (using **>**)
    - Appending process **stdout** to a *file* (using **>>**)
    - Redirecting process **stderr** to a *file* (using **2>**)
  - **my_shell** must support *pipes*
    - If there is a pipe (**|**) between two processes, **my_shell** must redirect **stdout** of the *left process* as **stdin** to the *right process*
    - Your shell must support *any number* of **pipes**

- **my_shell** can be called in one of two ways:
  - **UNIX> ./my_shell**
    - If there **is no** command line argument, **my_shell** will use the default command line prompt "**my_shell>** ".
  - **UNIX> ./my_shell PROMPT**
    - If there **is** a command line argument, **my_shell** will use the argument as the command line prompt: "**PROMPT> "**

- You **CANNOT** use the **system()** function
- You **MUST** use C-style I/O and strings (char *)
- You **MAY** use C++ data structures from the standard template library

- HINTS:
  - This is a **difficult** and **non-trivial** assignment. Please start early.
  - Work incrementally: e.g.,
    - Parse user input for simple commands (ignoring redirection and pipes)
    - Fork, exec, and wait for simple commands
    - Work on file redirection
    - Extend your code to include pipes
  - Try writing small test programs to experiment with redirection and pipes
    - **DO NOT** try to write everything at once!!!
  - You will need data structures to hold command sequences
  - **strtok** is a very useful function to parse a *line* of user input
    - Parse lines of user input for pipes first…
    - Then parse individual commands for redirection..

- Examples: For clarity, input is RED, output is BLUE, **my_shell** command prompt is GREEN

```
UNIX> ./my_shell
my_shell> ls
Makefile  my_shell  my_shell.cpp
my_shell> exit
UNIX>

UNIX> ./my_shell CMD
CMD> echo hello there
hello there
CMD> echo hello there > output.txt
CMD> cat < output.txt
hello there
CMD> echo #1 >> output.txt
CMD> echo #2 >> output.txt
CMD> echo #3 >> output.txt
CMD> echo #4 >> output.txt
CMD> cat output.txt
hello there
#1
#2
#3
#4
CMD> ls -l
total 72
-rw-r--r-- 1 student student    92 Jan  7 20:22 Makefile
-rwxrwxr-x 1 student student 48442 Jan 26 13:00 my_shell
-rw-r--r-- 1 student student  8859 Jan 26 13:00 my_shell.cpp
-rw-rw-r-- 1 student student    12 Jan 26 13:15 output2.txt
-rw-rw-r-- 1 student student    24 Jan 26 13:16 output.txt
CMD> ls -l | sort -nr
total 72
-rwxrwxr-x 1 student student 48442 Jan 26 13:00 my_shell
-rw-rw-r-- 1 student student    24 Jan 26 13:16 output.txt
-rw-rw-r-- 1 student student    12 Jan 26 13:15 output2.txt
-rw-r--r-- 1 student student    92 Jan  7 20:22 Makefile
-rw-r--r-- 1 student student  8859 Jan 26 13:00 my_shell.cpp
CMD> ls -l | sort -nr | head -n 3
total 72
-rwxrwxr-x 1 student student 48442 Jan 26 13:00 my_shell
-rw-rw-r-- 1 student student    24 Jan 26 13:16 output.txt
CMD> ls -l | sort -nr | head -n 3 | wc -l
3
CMD> ls -l | sort -nr | head -n 3 | wc -l > output3.txt
CMD> cat output3.txt
3
CMD> exit
UNIX>
```