Overview
You are to write and test a C/C++ program that implements a simple shell whose executable is called `xssh` which is a modified form of `xssh0`. This document specifies a basic `xssh`. The follow-on to this project will add a few more features and lift some simplifications assumed here. It has an optional `-x` flag:

```
xssh [-x]
```

`xssh` normally reads commands from stdin, one command per line. The `-x` flag indicates that the command line after variable substitution should be displayed before the command is evaluated. It has the following builtin (internal) commands (NOTE: `fork-exec` is never used in implementing a builtin command):

- `chdir <Pathname>`: Change the current directory to `<Pathname>` which can be an absolute or relative Unix pathname. If `<Pathname>` is not given, change the current directory to the path given by the environment variable `HOME`. The current directory should be maintained in an environment variable called `PWD`.

- `environ`: Display all of the environment variables; i.e., the name-value pairs, one per line just like `printenv` does. There are only three environment variables: `HOME`, `PWD` and `PATH`. `PATH` has the same format as in any other shells; i.e., `:` separates directory pathnames.

- `echo <Word> <Word> ...`: Display the arguments followed by a newline. Multiple spaces/tabs may be reduced to a single space.

- `quit <status>`: Quit the shell with integer status code `<status>`.

- `"bg ..."`: The remainder of the line should be run in the background with the word following `bg` treated as a non-builtin command.

- `"wait <pid>"`: The shell should wait for the process with PID `<pid>`; if there is no PID, then it should wait for all of its backgrounded processes to complete.

- `"pause <pid>"`: The process with PID `<pid>` should be put to sleep.

- `"resume <pid>"`: The process with PID `<pid>` should be allowed to continue to run.

- `"status"`: Display the status of immediate child processes. The display should show for each process the PID and the run state ("RUNNING" or "PAUSED").

- `"set"`: Set a variable to a value. A variable name is a single letter. For example, `set X 32` sets the variable X to the string 32. The value of X is denoted by `$X`; i.e., `$` denotes "the value of". (NOTE: This will be generalized to handle more general names in the extended form of this project.)
Here are other features of \texttt{xssh}: 

a) Multiple spaces/tabs are reduced to a single space during the substitution and line scanning phase.

b) The command line prompt should be the pathname of the current directory followed by the three character sequence '>>' (i.e., > >, >, space).

c) Define the root shell to be the first instance of \texttt{xssh0} and all other instances of \texttt{xssh0} to be subshells. The root shell inherits HOME, PWD and PATH from its parent process.

d) $XY$Z is the string resulting from the concatenation of the value of the variable X, the character 'Y' and the value of the variable Z.

e) The # character signifies the beginning of a comment. All other characters following and including the # should be ignored during interpretation.

f) A non-built-in command is assumed to be a Unix executable that can be found in a directory listed in the PATH environment variable.

g) \texttt{xssh} understands the shell variables $$, $$? and $$!, and these variables have the same meaning as in sh and bash.

Note that there is simple variable substitution, but there is no filename substitution nor command substitution.

\textbf{Implementation Notes} 

In this implementation, you can assume that the system will be small and therefore simple data structures are appropriate (i.e., there is no need at this time for sophisticated data structures). It is up to you to determine what you will need, but remember that simplicity will be a virtue in this assignment. Furthermore, for this part of Project A, you can assume that an error should terminate your shell with an error message.

\textbf{Additional Guidelines} 

- Code readability is of the utmost importance. The Web page will contain a summary of coding guidelines that you should follow in spirit. I am not rigid about these guidelines, but unreadable code will be penalized.

- All actual system calls should be wrapped so that any fatal errors will cause an error message to be displayed followed by an exit. I will not examine any solutions that do not follow this rule. By convention, the wrapped system call name will be the same as the actual system call name except the first character should be capitalized (e.g., \texttt{fork} is the wrapped version of \texttt{fork}).
What to Submit by Oct. 11

Submit three items in class:

1) A hardcopy source listing of your latest version of the bash interpreter. As a minimum, this code should contain the control structure for handling all built-in commands with stub code for each evaluation function. The stub code should display a message that says that the function was entered.

2) The output of your interpreter for the test case labeled ”Oct 11 Test” on the course Web page.

3) A single-page design guide containing the most important conventions, ideas and implementation notes related to your project. Do not use any font size less than 10 points.

4) Complete the status form on the Web page indicating which features are working, which have buggy or incomplete code, and which have not even been attempted. For buggy or incomplete code, give an indication of its incompleteness.

Since this submission will not be returned to you, make a copy of whatever you will need to work on the project. There will be only three grades assigned: 0, 5 or 10 points. The grade assigned will be subjective and based on perceived effort.

What to Submit by Oct. 20

The CS422S Web page contains a link to the documentation template. You should complete the template and submit it in both hardcopy AND electronic form. Submit the completed documentation template AND a listing of the source code. The electronic submission (described below) should include the completed documentation template, the source code, the Makefile, test scripts, and test output. The electronic copy is due by midnight Oct. 20. The hardcopy can be submitted in class on Oct. 20 or to my office by noon Oct. 21. This submission is worth 100 points.

Words of Caution

Here are some observations from my years of experience with projects like this:

- The documentation template is non-trivial. Do not expect to complete it in less than an hour. Furthermore, working code but no documentation is almost useless. So, don't forget to fill out the documentation template.

- Trivial bugs can consume tens of hours of time. Yes, I said tens of hours, not just hours. You need to start small, test the control structure, and incrementally add features on top of what seems like rock solid code.

- Keep different versions of your ”rock solid code versions” so that you can rollback to and recover from a stable version. This also helps if you mistakenly delete your latest source code!!!

- Don’t ignore error messages and think they will disappear on their own. They don’t. They just come back and bite you when you least want to be bitten.

- Try to understand the origins of your bugs rather than always doing trial and error changes. (Some trial and error may be appropriate in small test cases.)
• If the approach you are taking seems like it will be a nightmare to implement, then don’t implement it. Find a better way or better understand the system calls you are trying to use.

• The more lines of code you have, the more chances for bugs to appear.

• Have a plan. Don’t try to do everything at once.

Electronic Submission
The end result should be that you mail to kenw@arl.wustl.edu a single shar (shell archive) file containing your files. Do NOT submit object code or executables. The following commands will create a shar file named A.shar containing the files xssh.c, in1.txt, out1.txt, and other files and then send mail to me:

```
shar README Makefile xssh.c ... in1.txt out1.txt ... > A.shar
mail -s A.shar kenw@arl.wustl.edu < A.shar  # mail is usually in /bin
```

The README file is the completed documentation template.

Late Policy
There is no late submission date for either of the February deadlines. You must submit in class on that date.

The final submission can be one week late for a 20% penalty. Note that you should submit something even if the final version still has bugs.