CSE 325 **Spring 2020**

Computer Project #5

Assignment Overview

For this assignment, you are to design and implement a C/C++ program that will serve as an interpreter for shell scripts, as defined below. You will design and implement additional functionality to extend a previous project.

It is worth 40 points (4% of course grade) and must be completed no later than 11:59 PM on Thursday, 2/20.

Assignment Deliverables

The deliverables for this assignment are the following files:

`proj05.makefile` – the makefile which produces **`proj05`**
`proj05.student.c` – the source code file for your solution

Be sure to use the specified file names and to submit your files for grading via the CSE Handin system before the project deadline.

Assignment Specifications

1. The purpose of the program is to interpret the contents of a text file as a series of directives (defined below). The user will supply zero or more command line arguments to control execution of the program.

a) The program will process the command line arguments from left to right.

b) If an argument begins with the character '-', it will be processed as an option which controls the behavior of the program. Valid options are "-v" and "-s" (defined below).

c) If an argument does not begin with the character '-', it will be processed as a file name. The user will supply zero or more file names, where each input file contains zero or more lines of text.

2. For each input file, the program will repeatedly read a line of text and process it. An input line is defined as a sequence of zero or more tokens (character strings), separated by one or more delimiters (blanks and tabs), ending with a newline character. There will be no more than 128 characters in a line.

If the first token is the name of a built-in directive (listed below), then the program will take the appropriate action. Otherwise, the program will assume that it is an external command.

3. The program will recognize the following built-in directives:

`exit`	terminate processing the input file
`date`	display current date and time
`env`	display environment variables
`path`	display current search path
`cwd`	display absolute pathname of current working directory
`cd`	manage current working directory
`set`	manage environment variables

Built-in directives will be completely processed by the program (the program will not create a child process to perform the processing).

a) The "exit" directive will terminate processing of the current input file. End-of-file on the current input file will be processed as an implicit "exit" directive.

b) The "date" directive will display the current date and time in a human-readable format.

c) The "env" directive will display the user's environment variables in a format similar to "setenv" in the C shell.

d) The "path" directive will display the current search path in a readable format.

e) The "cwd" directive will display the absolute pathname of the current working directory.

f) The "cd" directive will manage the current working directory and will update the user's PWD environment variable.

The directive "cd" without any other tokens will reset the current working directory to be the user's home directory.

The directive "cd DIR" will reset the current working directory to be "DIR", where that token may be a relative or absolute pathname.

The directive "cd ~USER" will reset the current working directory to be the home directory of the user with username USER. As a special case, the symbol "~" represents the home directory of the current user.

g) The "set" directive will manage the user's set of environment variables.

The directive "set VAR VALUE" will set the environment variable VAR to the character string VALUE. If the environment variable VAR does not exist, it will be added to the user's environment variables.

The directive "set VAR" will remove the environment variable VAR from the user's environment variables.

4. For external commands, the first token will be the name of a file containing an executable program. The program will create a new thread for each external command; that thread will use function "system()" to execute the external command.

5. The valid command line options are "-v" (verbose) and "-s" (silent). An option applies to file names which appear after the option on the command line.

a) When the "-v" option is in effect (which is the default), the program will display the sequence number of the current line of text from the input file (starting at 1 for each input file) and the current line of text. The sequence number will be enclosed in the characters '<' and '>'. For example:

```
<1> cwd
```

b) When the "-s" option is in effect, the program will not display the sequence number or the current line of text.

5. The program will include appropriate logic to handle exceptional cases and errors.

Assignment Notes

1. As stated above, your source code file will be named "proj05.student.c"; that source code file may contain C or C++ statements.

2. You must use "g++" to translate your source code file in the CSE Linux environment.

3. Information about system calls and library functions which might be useful for this project may be viewed using the "man" utility. For example:

```
man 3 system
man 3 setenv
man 3 unsetenv
man 3 pthread_create
man 3 pthread_join
man 3 pthread_exit
```

4. You must use the POSIX threads library and function "system()" to process external commands. That is, your program will create a unique thread to process an external command. You may NOT call function "fork()" in your program.

5. As stated above, the command-line arguments will be processed from left to right. For example, consider the following command line:

```
proj05 testA -s testB testC –v testD testE –s testF
```

The "-v" option will be in effect for files "testA", "testD" and "testE"; the "-s" option will be in effect for files "testB", "testC" and "testF".