SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Computer Systems Engineering

Week 1: Lab 1 (25 marks)

Objective: Create a Shell Interface using Java or C

This lab exercise is to write a Java or C program to build an user shell interface. The interface accepts user commands and then create new process to execute the inserted command.

A shell interface provides the user with a prompt, after which the user enters the next command. The example below illustrates the prompt *jsh>* for java (*csh>* for C) and the user's next command: cat Prog.java. This command displays the file *Prog.java* on the terminal using the UNIX cat command.

jsh> cat Prog.java
csh> cat Prog.c

Perhaps the easiest technique for implementing a shell interface is to have the program first read what the user enters on the command line (here, *cat Prog.java*) and then create a separate external process that executes the command.

The Java and C program that provides the basic operations of a command-line shell is given in starting code. The *main()* method presents the prompt *jsh>* (for java shell) or *csh>* (for C shell) and waits to read input from the user.

This lab is organized into three questions: (1) creating the external process and executing the command in that process, (2) modifying the shell to allow changing directories, and (3) adding a history feature.

Q1: Creating an External Process (5 marks)

The first part of this project is to modify the main() method of the given program so that an external process is created and executes the command specified by the user. Initially, the command must be parsed into separate parameters and passed to ProcessBuilder object. For example, if the user enters the command:

```
jsh> cat Prog.java
```

the parameters are (1) cat and (2) Prog.java, and these parameters can be passed through *.command()* in ProcessBuilder class. For example:

```
ProcessBuilder pb = new ProcessBuilder();
pb.command(yourcommand);
Process p = pb.start();
```

If the user enters an invalid command, the *start()* method in the *ProcessBuilder* class throws an *java.io.IOException*. If this occurs, your program should output an appropriate error message and resume waiting for further commands from the user.

Notes:

- You can assume that the commands will not be more than 8000 characters.
- You can assume that the commands will not have escaped whitespaces or whitepace within quotes. (e.g. mkdir "some dir", mkdir some\ dir)

Suggested Function JAVA:

Function	Class which function belongs to
.command()	*ProcessBuilder*
.start()	*ProcessBuilder*
.getMessage()	*Exception*
.getInput	*Stream() Process*
.readline()	*BufferedReader*

Test Command JAVA:

```
jsh> ls -l
jsh> mkdir sampleFolder
jsh> ls
jsh> wronginput
```

Suggested Function C:

Function	Use for What	Needed headers

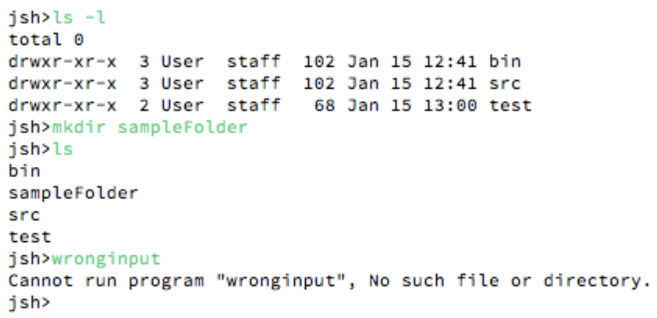
main()	*the core of every program and it is required in each c program*	#include <stdio.h>
		#include<stdlib.h>

fget	*To take input from user*	
system	*Execute command*	

Test Command C:

csh> ls -l
csh> mkdir sampleFolder
csh> ls
csh> wronginput

Test output example:



```
jsh>ls -l
total 0
drwxr-xr-x  3 User  staff  102 Jan 15 12:41 bin
drwxr-xr-x  3 User  staff  102 Jan 15 12:41 src
drwxr-xr-x  2 User  staff   68 Jan 15 13:00 test
jsh>mkdir sampleFolder
jsh>ls
bin
sampleFolder
src
test
jsh>wronginput
Cannot run program "wronginput", No such file or directory.
jsh>
```

Q2: Changing Directories (10 marks)

The next task is to modify the program in Q1 so that it changes directories. In UNIX systems, we encounter the concept of the *current working directory (pwd),* which is simply the directory you are currently in. The *cd* command allows a user changing current directory. Your shell interface must support this command. For example, if the current directory is */usr/tom* and the user enters *cd music*, the current directory becomes */usr/tom/music*. Subsequent commands relate to this current directory. For example, entering *ls* will output all the files in */usr/tom/music*.

When the start() method of a subsequent process is invoked, the new process will use this as the current working directory. For example, if one process with a current working directory of */usr/tom* invokes the command *cd music*, subsequent processes must set their working directories to */usr/tom/music* before beginning execution; then if the new directory invokes the command *cd ..* , the working directory will back to its parent directory */usr/tom*. It is important to note that your program first make sure the new path being specified is a valid directory. If not, your program should output an appropriate error message. If the user enters the command *cd*, change the current working directory to the user's home directory.

Notes:

- The "..", ".", and "~" shorthands for the parent, current and home directories respectively should be supported.

For JAVA:

The ProcessBuilder class provides the following method for setting the working directory:

```
ProcessBuilder pb = new ProcessBuilder();
pb.command(yourcommand);
pb.directory(newDirectory);
Process p = pb.start();
```

The home directory for the current user can be obtained by invoking *.getProperty()* method in the System class as follows:

```
System.getProperty("user.home");
```

Suggested Function:

Function	Class which function belongs to
.directory()	ProcessBuilder
.getProperty() System	*System*
.isDirectory() File	*File*
.getAbsolutePath() File	*File*
.getParent() File	*File*
.separator File	*File*

Test Command:

```
jsh> pwd
jsh> cd ..
jsh> pwd
jsh> cd myfolder
jsh> pwd
jsh> cd unknowfolder
```

For C:

The C language provides the following function to change the current working directory: *int chdir(const char *path);*

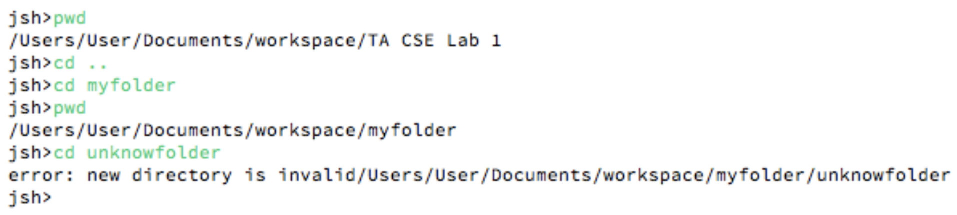
Suggested Function:
Function

Function	Use for	Needed headers
strtock	*Parsing the command*	#include<string.h>
chdir	*Change directory*	#include<unistd.h>

Test Command:

csh> pwd
csh> cd ..
csh> pwd
csh> cd myfolder
csh> pwd
c*sh> cd unknowfolder*

Test output example:



```
jsh>pwd
/Users/User/Documents/workspace/TA CSE Lab 1
jsh>cd ..
jsh>cd myfolder
jsh>pwd
/Users/User/Documents/workspace/myfolder
jsh>cd unknowfolder
error: new directory is invalid/Users/User/Documents/workspace/myfolder/unknowfolder
jsh>
```

Q3: Adding a History Feature (10 marks)

Many UNIX shells provide a *history* feature that allows users to see the history of commands they have entered and to rerun a command from that history. The history includes all commands that have been entered by the user since the shell was invoked. For example, if the user entered the history command and saw as output:

1 pwd
2 ls -l
3 cat Prog.java

The history would list pwd as the first most recent command entered, *ls -l* as the second most recent command, and so on. Modify your shell program so that commands are entered into a history.
Your program must allow users to rerun commands from their history by supporting the following three techniques:

1. When the user enters the command history, you will print out the contents of the history of commands that have been entered into the shell, along with the command numbers. The history list should be able to hold at least 10 most recent commands.
2. When the user enters *!!*, run the previous command in the history. If there is no previous command, output an appropriate error message.
3. When the user enters <integer value *i*>, run the *i*th most recent command in the history. For example, entering *3* would run the third most recent command in the command history.

Notes:

- You will only need to store valid commands (excluding "history", "!!" and "<integer value i>") in the history list.
- The commands can have consecutive tabs and spaces as whitespace, along with leading and trailing whitespace.

Make sure you perform proper error checking to ensure that the integer value is a valid number in the command history.

Suggested Function JAVA:

Function	Class which function belongs to
.insertElementAt()	*Vector<>*
.addElementAt()	*Vector<>*
. substring	*String*

Test Command:

```
jsh> ls -l
jsh> pwd
jsh> history
jsh> !!
jsh> 1
jsh> 10
```

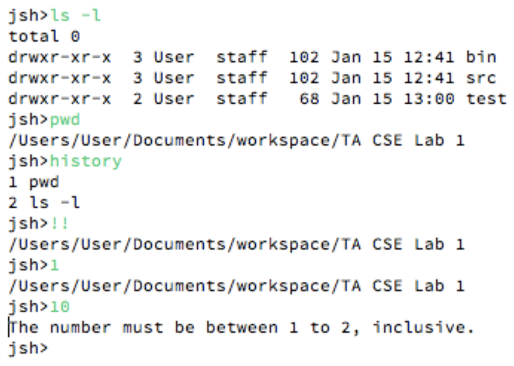
Suggested Function C:

Function	User for	Needed headers
malloc()	*Dynamic memory allocation*	
strcmp	*Compare two strings*	
strcpy	*Copy between strings*	
atoi	*Convert string to integer*	#include<ctype.h>

Test Command:

```
csh> ls -l
csh> pwd
csh> history
csh> !!
csh> 1
csh> 10
```

Test output example:



```
jsh>ls -l
total 0
drwxr-xr-x  3 User  staff  102 Jan 15 12:41 bin
drwxr-xr-x  3 User  staff  102 Jan 15 12:41 src
drwxr-xr-x  2 User  staff   68 Jan 15 13:00 test
jsh>pwd
/Users/User/Documents/workspace/TA CSE Lab 1
jsh>history
1 pwd
2 ls -l
jsh>!!
/Users/User/Documents/workspace/TA CSE Lab 1
jsh>1
/Users/User/Documents/workspace/TA CSE Lab 1
jsh>10
The number must be between 1 to 2, inclusive.
jsh>
```

Notes for Java:

- **Rename the "SimpleShell_startingCode.java" as "SimpleShell.java"**
- **How to compile your java code?**

javac SimpleShell.java

- **How to execute?**

java SimpleShell

Example:

Notes for C:

- **To build and compile your code using gcc, you need to use the command as the following:**

gcc yourCode.c –o yourCode

- **Then for the execution:**

./yourCode

Example:

Lab1 submission:

Submit your Java/ C source code (modified from the starting code) which contains all the functions in Q1&Q2&Q3 and a doc/pdf file (illustrate your interface result for in Q1&Q2&Q3) to eDimension before next Lab