2021

ICT374 Assignment 2

PROJECT 1: A SIMPLE UNIX SHELL

MUHAMMAD AMIR HAMZA & REYMARK RODEJO

MURDOCH UNIVERSITY
PERTH, WESTERN AUSTRALIA

Discipline of Information Technology, Media and Communications
College of Arts, Business, Law and Social Sciences

ICT374 ASSIGNMENT 2
PROJECT DECLARATION

Group Members (full name and student number):

Member 1: Muhammad Amir Hamza

Member 2: Reymark Rodejo

Tutor's Name: Mr. Wayne Muller

Assignment Due Date: 10 April 2021 **Date Submitted:** 12 April 2021

Project Number (please tick):

- ✓ Project 1: A Simple Unix Shell
- ☐ Project 2: A Simple File Transfer Protocol
- ☐ Project 3: A Simple HTTP Client and Server
- ☐ Other Project (please specify): _____

Your assignment should meet the following requirements. Please confirm this (by ticking boxes) before submitting your assignment.

✓All details above are completed.
✓We have read and understood the Documentation Requirements of this assignment
✓**This assignment submission is compliant to the Documentation Requirements.**
✓The archive file (a zip file) contains the file "Assignment2.docx"
✓We have included all relevant Linux source code, executables and test files in the tar archive. The file names are chosen according to the project specification.
✓This archive file will be submitted to ICT374 Unit LMS.
✓We have kept a copy of this assignment, including this archive file, in a safe place.
✓We have completed Task Allocation and Completion Record below.
✓**We have signed the Group Declaration in the next page.**

The unit coordinator may choose to use your submission as sample solutions to be viewed by other students, but only with your permission. Please indicate whether you give permission for this to be done.

☐ Yes, we are willing to have my submission without change be made public as a sample solution.
☐ Yes, we are willing to have my submission be made public as a sample solution, as long as my submission is edited to remove all mentions of my identity.
✓ No, we are not willing to have my submission made public.

Group Declaration

As a group assignment, each member of the group is expected to make an equal contribution to the assignment and receives the same mark for the assignment.

However, we recognise that on some occasions and due to various reasons, the actual contributions to the assignment from the members could be unequal despite the best efforts of each member. In this case, we can still accept your assignment provided that all members of the group reach an agreement on their percentages of contribution to the assignment, and the agreement accurately reflects the real contribution by each member. In such a case, a member's mark is linked to his or her agreed contribution to the assignment and is calculated using the following formula:

A member's mark = minimum (group mark x the member's percentage of contribution x 2, group mark + 10, 100)

On some rare occasions, the two members of the group fail to reach an agreement on their contributions to the assignment. In such a case, in order for your assignment to be marked, each member of the group must complete a detailed *Task Breakdown List* and state his or her own claim of the percentage of contribution to the assignment. Your tutor will then award each member a mark based on his assessment of the quality of the assignment as whole as well as his assessment of that member's contribution to the assignment based on the information provided.

Please complete and sign ***one*** of the three declarations below:

*We have made **equal** contributions to this assignment. We understand that each of us will receive the same mark for this assignment.*

Signature (member 1): Muhammad Amir Hamza Date: 12 April 2021

Signature (member 2): Reymark Rodejo Date: 12 April 2021

*We have made **unequal** contributions to this assignment. The percentage of contribution by each of us is given below (note the sum of the contributions by the two members must be equal to 100%):*

Member's name: Contribution (%):

Member's name: Contribution (%):

We understand that each of us will receive a mark for this assignment that is linked to our contributions to the assignment. The mark will be calculated using the following formula:

A member's mark = minimum (group mark x the member's percentage of contribution x 2, group mark + 10, 100)

Signature (member 1): _____ Date: _____

Signature (member 2): _____ Date: _____

We are unable to reach an agreement on the percentage of our contributions to this assignment. However, in order for our tutor to be able to properly assess the work completed by each of us, each of us has completed a detailed Task Breakdown List which is included in this submission. We will accept our tutor's determination of our contributions to this assignment.

Signature (member 1): _____ Date: _____

Signature (member 2): _____ Date: _____

Table of Contents

PROJECT DECLARATION 1
Group Declaration 2
1. List of files 4
2. Project 1: A Simple Unix Shell 4
3. Self-diagnosis and evaluation 7
4. Discussion of solution 8
5. Test Evidence 9
6. Soure code listing 26
Main.c 26
Token.h 42
Token.c 43
Command.h 43
Command.c 45
makefile 48

1. List of files

Assignment2.pdf – Documentation for the project
Data – folder that contains command, token header and c files. These are used in compiling main program.
Exe – Executable solution to program
exe.c – Starter code
main.c – Main program to be compiled to Exe executable program
main.c:Zone.Identifier
main.o – main object file
makefile – makefile used to compile solution. To use type "make" in terminal.
markingguide.doc
markingguide.doc:Zone.Identifier
show – executable of show c file
show.c – show c file used in testing
showRun – executable of show
tests.tar.gz – contains test files
tmp – contains test files and test executable show to be used in shell testing.

2. Project 1: A Simple Unix Shell

Design and implement a simple UNIX shell program using the grammar specified in the later part of this section. Please allow for at least 100 commands in a command line and at least 1000 arguments in each command.

In addition to the above, the following are required:

1. ***Reconfigurable shell prompt (default %)***

   The shell must have a shell built-in command `prompt` for changing the current prompt. For example, type the following command

   ```
   % prompt john$
   ```

   should change the shell prompt to `john$`, i.e., the second token of the command.

2. ***The shell built-in command `pwd`***

   This command prints the current directory (also known as *working directory*) of the shell process.

3. ***Directory walk***

   This command is similar to that provided by the Bash built-in command `cd`. In particular, typing the command without a path should set the current directory of the shell to the home directory of the user.

4. ***Wildcard characters***

If a token contains wildcard characters `*` or `?` the token is treated as a filename. The wildcard characters in such a token indicate to the shell that the filename must be expanded. For example the command

```
% ls *.c
```

may be expanded to `ls ex1.c ex2.c ex3.c` if there are three matching files `ex1.c ex2.c ex3.c` in the current directory.

You may implement this feature using the C function `glob`.

5. ***Standard input and output redirections > and <***

For example:

```
% ls -lt >  foo
```

would redirect the standard output of process `ls -lt` to file `foo`. Similarly in the following command,

```
% cat <  foo
```

the standard input of process `cat` is redirected to file `foo`.

6. ***Shell pipeline |***

For example:

```
% ls -lt | more
```

the standard output of process `ls -lt` is connected to the standard input of process `more` via a pipe.

7. ***Background job execution***

For example:

```
% xterm &
```

The commannd line starts command `xterm` in the background (i.e., the shell will not wait for the process to terminate and you can type in the next command immediately). The following command line

```
% sleep 20 &  ps -l
```

starts command `sleep 20` and immediately execute command `ps -l` without waiting for command `sleep 20` to finish first.

8. ***Sequential job execution***

For example the command line

```
%  sleep 20 ; ps -l
```

starts command `sleep 20` first, and wait for it to finish, then execute command `ps -l`.

9. The shell environment

The shell should inherit its environment from its parent process.

10. The shell built-in command `exit`

Use the built-in command `exit` to terminate the shell program.

The behaviour of the above commands (except `prompt`) should be as close to those of the Bash shell as possible. In addition, your shell should not be terminated by CTRL-C, CTRL-\, or CTRL-Z.

Finally you must not use any existing shell program to implement your shell (for example by calling a shell through the function `system`). That would defeat the purpose of this project.

In the above, commands such as `ls`, `cat`, `grep`, `sleep`, `ps` and `xterm` are used as examples to illustrate the use of your shell program. However your shell must be able to handle *any* command or executable program. Note the commands `prompt`, `pwd`, `cd` and `exit` should be implemented as shell builtins, not as external commands.

The syntax and behaviour of the built-in commands `pwd`, `cd` and `exit` should be similar to the corresponding commands under Bash shell.

A major part of this shell is a command line parser. Please read the this note for suggestions on implementing the parser.

Definition of Command Line Syntax

The following is the formal definition of the command line syntax for the shell, defined in Extended BNF:

```
    < command line >      ::=   < job >
                              | < job > '&'
                              | < job > '&' <
command line >
                              | < job > ';'
                              | < job > ';' <
command line >
```

```
   < job >                ::=   < command >
                              | < job > '|' <
command >

   < command >            ::=   < simple command
>
                              | < simple command
> '<' < filename >
                              | < simple command
> '>' < filename >

   < simple command >     ::=   < pathname >
                              | < simple command
>  < token >
```

An informal definition plus additional explanations of the syntax is given below:

1. A ***command line*** consists of one or several ***jobs*** separated by the special character "&" and/or ";". The last ***job*** may be followed by the character "&" or ";". If a ***job*** is followed by the character "&", then it should be executed in the background.
2. A ***job*** consists of one or more ***command***s separated by pipeline characters "|";
3. A ***command*** is either a ***simple command*** or a ***simple command*** followed by an input redirection (< filename) or an output redirection (> filename);
4. A ***simple command*** consists of a single ***pathname*** followed by zero or more tokens;
5. The following five characters are the ***special characters***: &, ;, |, < , > ;
6. The ***white space characters*** are defined to be the space character and the tab character;
7. A ***token*** is either a special character or a string that does not contain space characters or special characters. In this project we do not consider quoted strings. Therefore if single quote or double quote characters appear in a string, they are treated just like any other non-special characters without its usually special meaning;
8. ***Tokens*** must be separated by one or more white spaces;
9. A ***pathname*** is either a file name, or an absolute pathname, or a relative pathname. Examples of pathnames are **grep**, **/usr/bin/grep**, **bin/grep** and **./grep**;
10. A command line must end with a newline character.

3. Self-diagnosis and evaluation

List of functional features

1. Program can be built using makefile and creates an executable
2. Basic commands such as ls, ps, etc can be run
3. Basic commands can be run repeatedly
4. Built in commands “prompt”, “pwd”, “cd” and “exit”
5. Shell can execute long commands/commands with many arguments

6. Shell has wildcard functionality
7. Shell has sequential ";" execution capabilities.
8. Shell has concurrent execution abilities
9. Shell has input and output redirection functions
10. Shell can implement simple pipeline
11. Shell can implement long shell pipeline
12. Ignore SIGINT, SIGQUIT, and SIGTSTP signals.
13. Shell claims zombie processes
14. Handling of slow system calls
15. Built in command exit
16. Shell inherits its environment from parent process

List of not fully functional features.

1. Bug in Long pipeline where output to file is non functional.
2. Wildcard functionality has bugs when used with commands with multiple arguments.

4. Discussion of solution

The solution can be simplified into the following algorithm:

```
While(Exit has not been entered)
10.1.   Print shell prompt
10.2.   Get user input
10.3.   Parse user input into commands
10.4.   For each command
            4.1 Check for Built-in commands
            4.2 Check for separators (; | &)
                    4.2.1Execute as such
            Endfor
Endwhile
```

This means that the shell will prompt the user continually until the user types exit. Whenever the user types a line other than exit, the program parses the user input using the command and the token functions and stores the parsed commands into a Command struct array. After this, the program then loops through the array of command structures and first checks if they are built-in commands. If they are not, the program then checks them for special character separators (; | &). The program then executes the commands based off the presence of the mentioned separators. If a command is written by a user without a separator, it is by default considered as a command with a sequential separator (;).

The Command structure contains values for each command that are useful in the processing of commands. These values include the arguments of the command, its separator as well as file redirection arguments and locations.

The algorithm for the processing of the commands are as follows.

```
For each Command
        1. If it is built in command
           1.1 Run builtin command
2.         Else Check for | separator
3.         Else Check for ; separator
        4. Else Check for & separator
        5. Else if command is empty
           5.1 print prompt
```

The check for the pipe separator is intentionally put above the other separators as once this is true for the command, the for loop will then search for the other commands that are involved with the pipe. This feature also allows for the processing of multiple pipes.
File redirection is processed within the processing of the command under the | ; and & separators.

Multiple pipes are handled by sending the relevant commands into another array of Command structs and then processing them. In this process, pipes are placed in between commands so their inputs and outputs are appropriately placed. File redirection is also processed here as well if it is applicable.

Zombie processes are handled by creating a signal handler that handles SIGCHLD signals from background processes. Furthermore, signal handlers are also implemented to ignore SIGINT, SIGQUIT, and SIGTSTP signals.

This solution's strength lies in its ability to effectively parse commands. It can run multiple commands as well as a combination of commands. However, it has a weakness in the use of wildcards as it does not function well when used in commands with multiple arguments. This is one clear area of improvement.

5. Test Evidence

a. Basic commands.

Explanation: Checks for simples commands. Checks if commands can be run repeatedly. Error checking.

Input:

- %ls
- %ps
- %randomCommand

Output:

```
(null)% ls
Test-Cases.txt  abcxyz.ccc  exe.c
ls_test.txt.save                q1.txt    qt.txt
test2.txt
a               asdf.txt    filename.txt lstest.txt
q11.txt   qwerty.txt     test3.txt
ab              asdfwe.txt  foo           lstestbg.txt
q12.txt   rasndf.txt     test321.txt
abc             aws.txt     foo.txt       lstestbg1.txt
q13.txt   redirect.test  test453.txt
abcx            bgls.txt    hope.txt      m
q2.txt    s_test.txt     testfile.txt
abcxy           big.txt     ilename.txt   main.c
q3.txt    show           testing
abcxyz          bigbig.txt junk
main.c:Zone.Identifier            q321.txt show.c
tests
abcxyz.a        data        junk2         main.o
q4.txt    showRun        tests.tar.gz
abcxyz.aa       e32.txt     ls.txt        makefile
q5.txt    test           users
```

```
abcxyz.b        ers         ls1.tx
markingguide.doc                  q50.txt   test.txt
abcxyz.bb       est.txt     ls2.tx
markingguide.doc:Zone.Identifier  q51.txt   test1.txt
abcxyz.c        exe        ls_test.txt    outpasd.txt
q9.txt  test123.txt
(null)% ps
  PID TTY          TIME CMD
  307 pts/0    00:00:00 ps
32578 pts/0    00:00:00 bash
32763 pts/0    00:00:00 exe
(null)% randomCommand
(null)%
```

Explanation of output: simple commands can be run and can be run repeatedly. Error checking does not show anything.

b. Built-in commands
Explanation: Checks for the command “prompt”, “pwd”, “cd” and “exit”
Input:

- %prompt myshell
- %cd data
- %pwd
- %cd
- %cd ..
- %pwd
- %exit

Output:

```
(null)% prompt myshell
myshell% cd data
myshell% pwd
/home/reymark/home/786/data
/home/reymark/home/786/data
myshell% cd
myshell% cd ..
myshell% pwd
/home
/home
myshell% exit
```

Explanation of output: Built in commands are working as expected.

c. Long commands
Explanation: Checks whether the shell can take a command with many command line arguments.

Input:

- %ls -l -t a b c

- %cd tmp
- %./show a bb ccc dddd
- %./show a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 abc123xyz

Output:

```
(null)% ls -l -t a b c
ls: cannot access 'b': No such file or directory
ls: cannot access 'c': No such file or directory
-rw-r--r-- 1 reymark reymark 0 May 19 2008 a
(null)% cd tmp
(null)% ./show a bb ccc dddd
Command line arguments:  5
Command line argument 0:   ./show
Command line argument 1:   a
Command line argument 2:   bb
Command line argument 3:   ccc
Command line argument 4:   dddd
(null)% ./show a b c d e f g h i j k l m n o p q r s
t u v w x y z 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20 abc123xyz
Command line arguments:  48
Command line argument 0:   ./show
Command line argument 1:   a
Command line argument 2:   b
Command line argument 3:   c
Command line argument 4:   d
Command line argument 5:   e
Command line argument 6:   f
Command line argument 7:   g
Command line argument 8:   h
Command line argument 9:   i
Command line argument 10:   j
Command line argument 11:   k
Command line argument 12:   l
Command line argument 13:   m
Command line argument 14:   n
Command line argument 15:   o
Command line argument 16:   p
Command line argument 17:   q
Command line argument 18:   r
Command line argument 19:   s
Command line argument 20:   t
Command line argument 21:   u
Command line argument 22:   v
Command line argument 23:   w
Command line argument 24:   x
Command line argument 25:   y
Command line argument 26:   z
Command line argument 27:   1
```

```
Command line argument 28:   2
Command line argument 29:   3
Command line argument 30:   4
Command line argument 31:   5
Command line argument 32:   6
Command line argument 33:   7
Command line argument 34:   8
Command line argument 35:   9
Command line argument 36:   10
Command line argument 37:   11
Command line argument 38:   12
Command line argument 39:   13
Command line argument 40:   14
Command line argument 41:   15
Command line argument 42:   16
Command line argument 43:   17
Command line argument 44:   18
Command line argument 45:   19
Command line argument 46:   20
Command line argument 47:   abc123xyz
```

Explanation of output: Shell can take commands with many arguments

d. Wildcards

Explanation check whether shell can run with wildcards.

Input:

- %ls *.c
- %ls *.txt
- %ls -l tmp/abc.?
- %ls -l tmp/abc*.?

Output:

```
(null)% ls *.c
abcxyz.c  exe.c  main.c  show.c
ls: cannot access '*.c': No such file or directory
(null)% ls *.txt
Test-Cases.txt asdf.txt asdfwe.txt aws.txt
bgls.txt big.txt bigbig.txt e32.txt est.txt
filename.txt
ls: cannot access '*.txt': No such file or directory
(null)% ls -l tmp/abc.?
ls: cannot access 'tmp/abc.?': No such file or
directory
(null)% ls -l tmp/abc*.?
ls: cannot access 'tmp/abc*.?': No such file or
directory
(null)%
```

Explanation of output: * wildcard works but shows error message. ? wildcard does not function. The use of * wildcard in conjunction with multiple arguments does not work.

e. Sequential execution
Explanation: checks if commands are sequentially executed.

Input:
- %sleep 10 ; echo hello
- %sleep 10 ; ls -l
- %sleep 10 ; echo hello1 ; sleep 10 ; echo hello 2

Output:

```
(null)% sleep 10 ; echo hello
hello
(null)% sleep 10 ; ls -l
total 22788
-rw-r--r-- 1 reymark reymark     1313 May 19  2008
Test-Cases.txt
-rw-r--r-- 1 reymark reymark        0 May 19  2008 a
-rw-r--r-- 1 reymark reymark        0 May 19  2008
ab
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abc
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcx
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxy
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.a
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.aa
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.b
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.bb
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.c
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.ccc
-rw------- 1 reymark reymark      426 Apr 10 22:42
asdf.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:28
asdfwe.txt
-rwxr--r-- 1 reymark reymark      455 Apr 11 11:14
aws.txt
-rwxr--r-- 1 reymark reymark      684 Apr 12 09:07
bgls.txt
```

```
-rw-r--r-- 1 reymark reymark  4511345 Nov  7  2008
big.txt
-rw-r--r-- 1 reymark reymark 13534035 Nov  7  2008
bigbig.txt
drwxr-xr-x 2 reymark reymark     4096 Apr 10 12:20
data
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:31
e32.txt
-rw ------- 1 reymark reymark        0 Apr 10 22:55
ers
-rw ------- 1 reymark reymark      292 Apr 10 22:18
est.txt
-rwxr-xr-x 1 reymark reymark    23392 Apr 12 10:53
exe
-rw-r--r -- 1 reymark reymark      584 Apr  5 16:46
exe.c
-rw ------- 1 reymark reymark      371 Apr 10 22:31
filename.txt
-rw-r--r -- 1 reymark reymark       63 May 19  2008
foo
-rwxr--r -- 1 reymark reymark     4762 Apr 12 09:44
junk
-rwxr--r -- 1 reymark reymark        0 Apr 12 09:44
junk2
-rw-r--r -- 1 reymark reymark   148992 Nov  7  2008 m
-rw-r--r -- 1 reymark reymark    14890 Apr 12 10:53
main.c
-rw-r--r -- 1 reymark reymark       98 Apr 11 10:31
main.c:Zone.Identifier
-rw-r--r-- 1 reymark reymark    15656 Apr 12 10:53
main.o
-rw-r--r-- 1 reymark reymark      396 Apr 12 10:38
makefile
-rw-r--r-- 1 reymark reymark   150016 Nov  7  2008
markingguide.doc
-rw-r--r-- 1 reymark reymark       98 Apr 10 20:18
markingguide.doc:Zone.Identifier
-rw-r--r-- 1 reymark reymark    13344 May 19  2008
show
-rw-r--r-- 1 reymark reymark      252 May 19  2008
show.c
-rw-r--r-- 1 reymark reymark    16656 Apr 10 13:25
showRun
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:09
test
-rw-r--r-- 1 reymark reymark 4813940 Apr 10 13:22
tests.tar.gz
drwxr-xr-x 2 reymark reymark     4096 Apr 12 11:09
tmp
-rw------- 1 reymark reymark      432 Apr 10 22:55
users
```

```
(null)% sleep 10 ; echo hello1 ; sleep 10 ; echo
hello 2
hello1
hello 2
(null)%
```

Explanation of output: Commands are executed in sequence with a 10 second delay before each output.

f. Concurrent execution
Explanation: Checks if commands can be run in the background or concurrently with the use of "&".
Input:

- %echo hello & echo world
- %sleep 10 & echo hello
- %ps & ls
- %echo ps-command & ps & echo ls-command & ls -l
- %sleep 10 &

Output:

```
(null)% echo hello & echo world
hello
world
(null)% sleep 10 & echo world
world
(null)% ps & ls
Test-Cases.txt  abcxyz     abcxyz.ccc bigbig.txt
exe.c         main.c                          show
users
a               abcxyz.a   asdf.txt    data
filename.txt   main.c:Zone.Identifier
show.c
ab              abcxyz.aa asdfwe.txt e32.txt
foo           main.o
showRun
abc             abcxyz.b   aws.txt     ers
junk          makefile                        test
abcx            abcxyz.bb  bgls.txt    est.txt
junk2         markingguide.doc
tests.tar.gz
abcxy           abcxyz.c   big.txt     exe         m
markingguide.doc:Zone.Identifier  tmp
(null)%   PID TTY          TIME CMD
  468 ?        00:00:00 ps

(null)% echo ps-command & ps & echo ls-command & ls
-l
ps-command
(null)% ls-command
  PID TTY          TIME CMD
```

```
  472 ?        00:00:00 ps
total 22788
-rw-r--r-- 1 reymark reymark     1313 May 19 2008
Test-Cases.txt
-rw-r--r-- 1 reymark reymark        0 May 19  2008 a
-rw-r--r-- 1 reymark reymark        0 May 19 2008
ab
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abc
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcx
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxy
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.a
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.aa
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.b
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.bb
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.c
-rw-r--r-- 1 reymark reymark        0 May 19 2008
abcxyz.ccc
-rw------- 1 reymark reymark      426 Apr 10 22:42
asdf.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:28
asdfwe.txt
-rwxr--r-- 1 reymark reymark      455 Apr 11 11:14
aws.txt
-rwxr--r-- 1 reymark reymark      684 Apr 12 09:07
bgls.txt
-rw-r--r-- 1 reymark reymark 4511345 Nov 7 2008
big.txt
-rw-r--r-- 1 reymark reymark 13534035 Nov 7 2008
bigbig.txt
drwxr-xr-x 2 reymark reymark     4096 Apr 10 12:20
data
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:31
e32.txt
-rw------- 1 reymark reymark        0 Apr 10 22:55
ers
-rw------- 1 reymark reymark      292 Apr 10 22:18
est.txt
-rwxr-xr-x 1 reymark reymark    23392 Apr 12 10:53
exe
-rw-r--r-- 1 reymark reymark      584 Apr 5 16:46
exe.c
```

```
-rw------- 1 reymark reymark      371 Apr 10 22:31
filename.txt
-rw-r--r-- 1 reymark reymark       63 May 19 2008
foo
-rwxr--r-- 1 reymark reymark     4762 Apr 12 09:44
junk
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:44
junk2
-rw-r--r-- 1 reymark reymark   148992 Nov  7  2008 m
-rw-r--r-- 1 reymark reymark    14890 Apr 12 10:53
main.c
-rw-r--r-- 1 reymark reymark       98 Apr 11 10:31
main.c:Zone.Identifier
-rw-r--r-- 1 reymark reymark    15656 Apr 12 10:53
main.o
-rw-r--r-- 1 reymark reymark      396 Apr 12 10:38
makefile
-rw-r--r-- 1 reymark reymark   150016 Nov 7 2008
markingguide.doc
-rw-r--r-- 1 reymark reymark       98 Apr 10 20:18
markingguide.doc:Zone.Identifier
-rw-r--r-- 1 reymark reymark    13344 May 19 2008
show
-rw-r--r-- 1 reymark reymark      252 May 19 2008
show.c
-rw-r--r-- 1 reymark reymark    16656 Apr 10 13:25
showRun
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:09
test
-rw-r--r-- 1 reymark reymark 4813940 Apr 10 13:22
tests.tar.gz
drwxr-xr-x 2 reymark reymark     4096 Apr 12 11:09
tmp
-rw------- 1 reymark reymark      432 Apr 10 22:55
users

(null)% sleep 10 &
(null)%
```

Explanation of output: Output from commands are generated simultaneously.

g. Standard input and output redirection “<” and “>”
Input:

- %cat < tmp/foo
- %grep line < tmp/foo
- %ls -l > tmp/junk
- %cat tmp/foo > tmp/junk2

Output:

```
(null)% cat < tmp/foo
```

```
this is file foo
file foo - line 2
line 3
line 4
the last line
(null)% grep line < tmp/foo
file foo - line 2
line 3
line 4
the last line
(null)% ls -l > tmp/junk
(null)% cat tmp/foo > tmp/junk2
(null)% cat tmp/foo > tmp/junk2
(null)% cat tmp/junk
total 22788
-rw-r--r-- 1 reymark reymark     1313 May 19  2008
Test-Cases.txt
-rw-r--r-- 1 reymark reymark        0 May 19  2008 a
-rw-r--r-- 1 reymark reymark        0 May 19  2008
ab
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abc
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcx
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxy
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.a
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.aa
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.b
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.bb
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.c
-rw-r--r-- 1 reymark reymark        0 May 19  2008
abcxyz.ccc
-rw------- 1 reymark reymark      426 Apr 10 22:42
asdf.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:28
asdfwe.txt
-rwxr--r-- 1 reymark reymark      455 Apr 11 11:14
aws.txt
-rwxr--r-- 1 reymark reymark      684 Apr 12 09:07
bgls.txt
-rw-r--r-- 1 reymark reymark  4511345 Nov  7  2008
big.txt
```

```
-rw-r--r-- 1 reymark reymark 13534035 Nov 7 2008
bigbig.txt
drwxr-xr-x 2 reymark reymark     4096 Apr 10 12:20
data
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:31
e32.txt
-rw------- 1 reymark reymark        0 Apr 10 22:55
ers
-rw------- 1 reymark reymark      292 Apr 10 22:18
est.txt
-rwxr-xr-x 1 reymark reymark    23392 Apr 12 10:53
exe
-rw-r--r-- 1 reymark reymark      584 Apr 5 16:46
exe.c
-rw------- 1 reymark reymark      371 Apr 10 22:31
filename.txt
-rw-r--r-- 1 reymark reymark       63 May 19 2008
foo
-rwxr--r-- 1 reymark reymark     4762 Apr 12 09:44
junk
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:44
junk2
-rw-r--r-- 1 reymark reymark   148992 Nov  7  2008 m
-rw-r--r-- 1 reymark reymark    14890 Apr 12 10:53
main.c
-rw-r--r-- 1 reymark reymark       98 Apr 11 10:31
main.c:Zone.Identifier
-rw-r--r-- 1 reymark reymark    15656 Apr 12 10:53
main.o
-rw-r--r-- 1 reymark reymark      396 Apr 12 10:38
makefile
-rw-r--r-- 1 reymark reymark   150016 Nov 7 2008
markingguide.doc
-rw-r--r-- 1 reymark reymark       98 Apr 10 20:18
markingguide.doc:Zone.Identifier
-rw-r--r-- 1 reymark reymark    13344 May 19 2008
show
-rw-r--r-- 1 reymark reymark      252 May 19 2008
show.c
-rw-r--r-- 1 reymark reymark    16656 Apr 10 13:25
showRun
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:09
test
-rw-r--r-- 1 reymark reymark 4813940 Apr 10 13:22
tests.tar.gz
drwxr-xr-x 2 reymark reymark     4096 Apr 12 11:27
tmp
-rw------- 1 reymark reymark      432 Apr 10 22:55
users
(null)% cat tmp/junk2
this is file foo
```

```
file foo - line 2
line 3
line 4
the last line
(null)%
```

Explanation of output: Input and output redirection functioning as expected.

h. Simple shell pipeline
Explanation: Checks simple pipeline functionality
Input:

- %cat tmp/foo | cat
- %cat tmp/foo | grep line
- %cat tmp/foo | sort
- %cat tmp/foo | sort -r

Output:

```
(null)% cat tmp/foo | cat
this is file foo
file foo - line 2
line 3
line 4
the last line
(null)% cat tmp/foo | grep line
file foo - line 2
line 3
line 4
the last line
(null)% cat tmp/foo | sort
file foo - line 2
line 3
line 4
the last line
this is file foo
(null)% cat tmp/foo | sort -r
this is file foo
the last line
line 4
line 3
file foo - line 2
(null)%
```

Explanation of output: Simple pipe works as expected.

i. Long shell pipeline:
Explanation: Checks for multiple pipes
Input:

- %cat tmp/foo | sort | sort -r | grep line
- % cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat
- % cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | cat > junk

- % cat | cat | cat | cat | cat | cat | cat | cat | cat | cat | grep line
  Output:

```
(null)% cat tmp/foo | sort | sort -r | grep line
the last line
line 4
line 3
file foo - line 2
(null)% cat | cat | cat | cat | cat | cat | cat | cat |
cat | cat | cat
sfdd
sfdd
sdf
sdf
d
d
df
df
g
g
(null)% cat | cat | cat | cat | cat | cat | cat | cat |
cat | cat | cat > junk
je
sdf
grg
dfg
sdfgrk
(null)% cat < junk
je
sdf
grg
dfg
sdfgrk
 1 reymark reymark    1313 May 19  2008 Test-Cases.txt
-rw-r--r-- 1 reymark reymark         0 May 19  2008 a
-rw-r--r-- 1 reymark reymark         0 May 19  2008 ab
-rw-r--r-- 1 reymark reymark         0 May 19  2008 abc
-rw-r--r-- 1 reymark reymark         0 May 19  2008 abcx
-rw-r--r-- 1 reymark reymark         0 May 19  2008 abcxy
-rw-r--r-- 1 reymark reymark         0 May 19  2008 abcxyz
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.a
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.aa
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.b
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.bb
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.c
-rw-r--r-- 1 reymark reymark         0 May 19  2008
abcxyz.ccc
-rw------- 1 reymark reymark       426 Apr 10 22:42
asdf.txt
```

```
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:28
asdfwe.txt
-rwxr--r-- 1 reymark reymark      455 Apr 11 11:14
aws.txt
-rwxr--r-- 1 reymark reymark      684 Apr 12 09:07
bgls.txt
-rw-r--r-- 1 reymark reymark  4511345 Nov  7  2008
big.txt
-rw-r--r-- 1 reymark reymark 13534035 Nov  7  2008
bigbig.txt
drwxr-xr-x 2 reymark reymark     4096 Apr 10 12:20 data
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:31
e32.txt
-rw------- 1 reymark reymark        0 Apr 10 22:55 ers
-rw------- 1 reymark reymark      292 Apr 10 22:18
est.txt
-rwxr-xr-x 1 reymark reymark    23392 Apr 12 09:25 exe
-rw-r--r-- 1 reymark reymark      584 Apr  5 16:46 exe.c
-rw------- 1 reymark reymark      371 Apr 10 22:31
filename.txt
-rw-r--r-- 1 reymark reymark       63 May 19  2008 foo
-rw-r--r-- 1 reymark reymark      260 Apr 10 14:38
foo.txt
-rwxr--r-- 1 reymark reymark       38 Apr 12 08:57
hope.txt
-rw------- 1 reymark reymark      371 Apr 10 22:31
ilename.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:44 junk
-rw------- 1 reymark reymark        0 Apr 10 22:23 ls.txt
-rw-r--r-- 1 reymark reymark        5 Apr 12 08:51 ls1.tx
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:52 ls2.tx
-rw------- 1 reymark reymark     1664 Apr 10 22:20
ls_test.txt
-rw------- 1 reymark reymark      318 Apr 10 22:20
ls_test.txt.save
-rw------- 1 reymark reymark      652 Apr 12 08:49
lstest.txt
-rwxr--r-- 1 reymark reymark      697 Apr 12 09:07
lstestbg.txt
-rwxr--r-- 1 reymark reymark      711 Apr 12 09:08
lstestbg1.txt
-rw-r--r-- 1 reymark reymark   148992 Nov  7  2008 m
-rw-r--r-- 1 reymark reymark    14884 Apr 12 09:27 main.c
-rw-r--r-- 1 reymark reymark       98 Apr 11 10:31
main.c:Zone.Identifier
-rw-r--r-- 1 reymark reymark    15848 Apr 12 09:25 main.o
-rw-r--r-- 1 reymark reymark      390 Apr 9 18:02
makefile
-rw-r--r-- 1 reymark reymark   150016 Nov 7 2008
markingguide.doc
-rw-r--r-- 1 reymark reymark       98 Apr 10 20:18
markingguide.doc:Zone.Identifier
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:26
outpasd.txt
-rwxr--r-- 1 reymark reymark      497 Apr 11 11:55 q1.txt
```

```
-rw------- 1 reymark reymark      512 Apr 11 12:28
q11.txt
-rwxr--r-- 1 reymark reymark      520 Apr 11 12:34
q12.txt
-rwxr--r-- 1 reymark reymark      528 Apr 11 12:36
q13.txt
-rwxr--r-- 1 reymark reymark      497 Apr 11 11:59 q2.txt
-rwxr--r-- 1 reymark reymark      497 Apr 11 12:21 q3.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 00:17
q321.txt
-rwxr--r-- 1 reymark reymark      497 Apr 11 12:21 q4.txt
-rwxr--r-- 1 reymark reymark      497 Apr 11 11:53 q5.txt
-rwxr--r-- 1 reymark reymark      536 Apr 11 12:44
q50.txt
-rwxr--r-- 1 reymark reymark      544 Apr 11 13:28
q51.txt
-rwxr--r-- 1 reymark reymark      504 Apr 11 12:27 q9.txt
-rwxr--r-- 1 reymark reymark      476 Apr 11 11:22 qt.txt
-rwxr--r-- 1 reymark reymark      480 Apr 11 11:13
qwerty.txt
-rw------- 1 reymark reymark      417 Apr 10 22:41
rasndf.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:39
redirect.test
-rw------- 1 reymark reymark      312 Apr 10 22:20
s_test.txt
-rw-r--r-- 1 reymark reymark    13344 May 19  2008 show
-rw-r--r-- 1 reymark reymark      252 May 19  2008 show.c
-rw-r--r-- 1 reymark reymark    16656 Apr 10 13:25
showRun
-rwxr--r-- 1 reymark reymark        0 Apr 12 09:09 test
-rw------- 1 reymark reymark      716 Apr 12 09:09
test.txt
-rw------- 1 reymark reymark      406 Apr 10 22:40
test1.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:18
test123.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:16
test2.txt
-rwxr--r-- 1 reymark reymark      613 Apr 12 08:23
test321.txt
-rwxr--r-- 1 reymark reymark        0 Apr 12 08:20
test453.txt
-rw------- 1 reymark reymark      396 Apr 10 22:38
testfile.txt
-rwxr--r-- 1 reymark reymark      638 Apr 12 08:36
testing
drwxr-xr-x 2 reymark reymark     4096 Apr 10 13:22 tests
-rw-r--r-- 1 reymark reymark 4813940 Apr 10 13:22
tests.tar.gz
-rw------- 1 reymark reymark      432 Apr 10 22:55 users
(null)% cat | cat | cat | cat | cat | cat | cat | cat |
cat | cat | grep line
sdf
dfs
df
```

```
sg
line
line
odfgdf line
odfgdf line
(null)%
```

Explanation of output: Output redirection to junk does not work as expected. Rest of the inputs works as expected.

j. Claim of zombies
Explanation: Checks if the shell claims zombie processes.
Input:
In the shell:

- %sleep 1 &
- %sleep 1 &
- %sleep 1 &
- %sleep 1 &
- %sleep 1 &

In another terminal

- ps -elfH | grep reymark

Output:

```
myprompt% sleep 1 &
myprompt% sleep 1 &
myprompt% sleep 1 &
myprompt% sleep 1 &
myprompt% sleep 1 &
myprompt%
```

In the other terminal

```
reymark@LAPTOP-OURFGNLB:~/home$ ps -elfH | grep
reymark
4 S reymark  32578 32577  0  80   0 -  5781 wait
09:26 pts/0    00:00:00       -bash
0 S reymark    545 32578  0  80   0 -  1131 -
11:44 pts/0    00:00:00         ./exe
4 S reymark    560   559  0  80   0 -  5781 wait
11:44 pts/1    00:00:00       -bash
0 R reymark    591   560  0  80   0 -  9450 -
11:45 pts/1    00:00:00         ps -elfH
0 S reymark    592   560  0  80   0 -  3715 pipe_w
11:45 pts/1    00:00:00         grep --color=auto
reymark
```

Explanation of output:
Shell claims zombie processes. No defunct processes in the other terminal.

k. Ignore Ctrl-C, Ctrl-\ and Ctrl-Z

Explanation: To check if shell ignores SIGINT, SIGQUIT and SIGTSTP by typing Ctrl-C, Ctrl-\ and Ctrl-Z from shell prompt.

Input:

- % Ctrl-C, Ctrl-\ and Ctrl-Z

Output:

```
(null)% ^C^\^Z
(null)%
```

Explanation of output:
Shell ignores SIGINT, SIGQUIT and SIGTSTP signals.

l. Shell inherits environment from parent process

Explanation: Checks if shell inherits environment from parent process

Input:

- %pwd

Output:

```
(null)% pwd
/home/reymark/home/786
/home/reymark/home/786
(null)%
```

Explanation of output:
Shell inherits environment from parent process

m. Combinations

Explanation: Checks if shell can run combinations.

Input:

- %ls -l > junk ; cat < junk ; ls | sort | head > junk2 & sleep 10;

Output:

```
(null)% ls -l junk ; cat < junk ; ls | sort | head >
junk2 & sleep 10 ;
-rwxr--r-- 1 reymark reymark 914 Apr 12 13:01 junk
```

```
-x 2 reymark reymark    4096 Apr 10 12:20 data
-rwxr-xr-x 1 reymark reymark   23392 Apr 12 12:31
exe
-rw-r--r-- 1 reymark reymark     584 Apr 5 16:46
exe.c
-rwxr--r-- 1 reymark reymark       4 Apr 12 13:00
junk
-rw-r--r-- 1 reymark reymark   14890 Apr 12 10:53
main.c
```

```
-rw-r--r-- 1 reymark reymark      98 Apr 11 10:31
main.c:Zone.Identifier
-rw-r--r-- 1 reymark reymark   15656 Apr 12 10:53
main.o
-rw-r--r-- 1 reymark reymark     396 Apr 12 10:38
makefile
-rw-r--r-- 1 reymark reymark 150016 Nov 7 2008
markingguide.doc
-rw-r--r-- 1 reymark reymark      98 Apr 10 20:18
markingguide.doc:Zone.Identifier
-rw-r--r-- 1 reymark reymark   13344 May 19 2008
show
-rw-r--r-- 1 reymark reymark     252 May 19 2008
show.c
-rw-r--r-- 1 reymark reymark   16656 Apr 10 13:25
showRun
-rw-r--r-- 1 reymark reymark 4813940 Apr 10 13:22
tests.tar.gz
drwxr-xr-x 2 reymark reymark    4096 Apr 12 11:28
tmp
(null)%
```

Explanation of output:

- Combinations work as expected. However, this will not work in some cases if wildcards are used.

6. Soure code listing

Main.c

```
/*************************************************************
*********************
 AUTHOR : Reymark Rodejo (33692546) & Muhammad Amir Hamza
(34123215)
 DATE   : 10-Apr-2021
 PROJECT: Simple Custom Shell
 TYPE   : Assignment 2
 UNIT   : Operating System & Programming System
 ID     : ICT
 CODE   : 374
 INSTIT : Murdoch University Dubai
 (C)opyright 2021, United Arab Emirates
*************************************************************
*********************
 MODULE: main.c
*************************************************************
*********************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <glob.h>
#include <unistd.h>
#include <errno.h>
```

```
#include <error.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "./data/token.h"
#include "./data/command.h"

#define BUFSIZE 256
#define CMD_LIST_SIZE 1000

#define logfile "/tmp/useless.log"
#define tmpfile "/tmp/shellTemp.csv"

//char str[100] = "ls -l /dev > ttylist & ps -f > plist ;
ls -l *list";

int clearTerminal();
int writeUsernameInFile();
int readUsernameFromFile();
int changeUsername(Command command);
int checkDirectory();
int changeDirectory(Command command);
int changeDirectoryToRoot(Command command);
int globForWC(Command command);
int backgdCmdExecuter(Command command);
int normalCmdExecuter(Command command);
void pipeCmdExecuter(Command pipedCommands[], int pipes);
int fileRdExecuter(Command command, int fileNo, char
*fileName);
void freeCommand(Command *command);
void zombieChild();


int daemon_init(void);
int try_deamon(void);

int main (void)
{
     //TERMINATION CONTROL
     sigset_t signal;
     sigemptyset(&signal);
     sigaddset(&signal, SIGINT);
     sigaddset(&signal, SIGTSTP);
     sigaddset(&signal, SIGQUIT);
     sigprocmask(SIG_SETMASK, &signal, NULL);

     //BG PROCESS AUTO RECURSION CONTROL
     struct sigaction cSignal;
```

```
     cSignal.sa_handler = zombieChild;
     sigaction(SIGCHLD, &cSignal, NULL);

     //VARIABLES-MAIN
     char    *cmdArr     = malloc(sizeof(char) *
CMD_LIST_SIZE);
     char    *cmdListArr = malloc(sizeof(char) *
CMD_LIST_SIZE);
     char    *tokenArr[CMD_LIST_SIZE];
     Command command[MAX_NUM_COMMANDS];
     Command  pipedCommands [MAX_NUM_COMMANDS];

     //VARIABLE-MODERATE
     char    clear[] = "clear\0";
     char    exitT[] = "exit\0";
     Command singleCmd;

     //VARIABLE-PLUS
     pid_t pid;
     int   cmdArrLen     = 0;
     int   checkExit     = 1;
     int   checkCler     = 1;
     int   no_of_tokens = 0;
     int   no_of_commnd = 0;
     int   wildCardCheck = 1;
     int   chk           = -1;
     int   again         = 1;
     char  *linept       = NULL;
     int pipes = 0;


     //SHELL-SYMBOL-HANDLER
     //will create a shell-temp file in your home dir and
writes
     //user and symbol in it
     writeUsernameInFile();

     while (checkExit != 0)
     {
          //will read a shell-temp file in your home dir
to get
          //user and symbol for printing
          readUsernameFromFile();

          //test-deamon
          //try_deamon();

          //user input
          //OLD - fgets(cmdArr, CMD_LIST_SIZE, stdin);
          linept = fgets(cmdArr, CMD_LIST_SIZE, stdin);
          while(linept == NULL && errno == EINTR)
```

```
          {
               linept = fgets(cmdArr, CMD_LIST_SIZE,
stdin);
          }

          //Searches for new-line in user cmd input
          char *cleanInput = strchr(cmdArr, '\n');
          //Overwrites new-line from input
          if(cleanInput != NULL)
          {
               *cleanInput = '\0';
          }
          //END - the program if exit is 0
          if(strcmp(exitT, cmdArr) == 0)
          {
               return 0;
          }
          else if(strcmp(cmdArr, "") == 0)
          {
               //empty (to stop prog auto recursive
execution)
          }
          else
          {
               //make token of cmd list
               no_of_tokens = tokenize(cmdArr, tokenArr);
               //seprate each command from the list
               no_of_commnd = separateCommands(tokenArr,
command);

               if(strcmp(cmdArr, "") != 0)
               {
                    //PRINTING COMMANDS
                    int j = 0;
                    //printf("\n[COMMANDS] \n");
                    for(int i=0; i<no_of_commnd; i++)
                    {
                         singleCmd = command[i];

                         //check if cmd contain '*'
                         //0 - for * & ?
                         //1 - default null

                         for(int z=singleCmd.first;
((z>=singleCmd.first) && (z<=singleCmd.last)); z++)
                        {
                            if(*tokenArr[z] == '*' ||
*tokenArr[z] == '?')
                            {
                              wildCardCheck = 0;
                            }
```

```
                        else
                        {
                          wildCardCheck = 1;
                        }
                    }


                     if(wildCardCheck == 0)
                     {

                          globForWC(command[i]);
                     }
                     else if(strcmp(cmdArr, "prompt")
== 0)
                     {

                          changeUsername(command[i]);
                     }
                     else if(strcmp(cmdArr, "pwd") ==
0)
                     {

                          checkDirectory();
                     }
                     else if(strcmp(cmdArr, "cd") ==
0)
                     {


     changeDirectory(command[i]);
                     }
                     else if(strcmp(cmdArr, "su") ==
0)
                     {


    changeDirectoryToRoot(command[i]);
                     }
                     else if(strcmp(command[i].sep,
"|") == 0) // ls | wc | wc | wc ;
                     {
                          pipedCommands[pipes] =
command[i];
                          pipes++;
                          continue;
                     }
                     if (pipes > 0)
                     {
                               pipedCommands[pipes++]
= singleCmd;
```

```
     pipeCmdExecuter(pipedCommands, pipes);


     freeCommand(pipedCommands);

                                        pipes = 0;
                          }
                          else if(strcmp(command[i].sep,
";") == 0)
                          {

     normalCmdExecuter(command[i]);
                          }
                          else if(strcmp(command[i].sep,
"&") == 0)
                          {

     backgdCmdExecuter(command[i]);
                          }
                          else
                          {

     normalCmdExecuter(command[i]);
                          }
                    }
               }
          }

          if(no_of_commnd > 0)
          {
               freeCommand(command);
          }


     }
     return 0;
}

int clearTerminal(Command command)
{
     pid_t pid, cpid;
     int   status, exit_status;

     pid = fork();

     if(pid == 0) //child-process
     {
          execvp(command.argv[0], command.argv);
          perror("error while executing 'clear'\n");
          exit(EXIT_FAILURE);
```

```
	}
	else
	{
		cpid = wait(&status);
		if(WIFEXITED(status))
		{
			exit_status = WEXITSTATUS(status);
			printf("Exit status was %d\n",
exit_status);
		}
	}
	return 0;
}

int writeUsernameInFile()
{
	FILE *fpW;
	char *currentUser = getlogin();

	fpW = fopen(tmpfile, "w");
	if(!fpW)
	{
		perror("Error while opening the file.\n");
		exit(EXIT_FAILURE);
	}
	else
	{
		fprintf(fpW, "%s%s", currentUser, "% ");
	}
    fflush(fpW);

	return 0;
}

int readUsernameFromFile()
{
	FILE *fpR;
	char symbolInFile;

	fpR = fopen(tmpfile, "r"); // read mode
	if (!fpR)
	{
		perror("Error while opening the file.\n");
		exit(EXIT_FAILURE);
	}
	else
	{
		while((symbolInFile = fgetc(fpR)) != EOF)
		{
			printf("%c", symbolInFile);
		}
```

```
	}
	fflush(fpR);

	return 0;
}

int changeUsername(Command command)
{
	if(command.last == 1)
	{
		FILE *fpR;

		fpR = fopen(tmpfile, "w");
		if (!fpR)
		{
			perror("Error while opening the file.\n");
			exit(EXIT_FAILURE);
		}
		else
		{
			fprintf(fpR, "%s%s", command.argv[1], "%
");
		}
		fflush(fpR);
	}
	else
	{
		printf("WARNING: '%s' Less or many operands
were entered\n", command.argv[0]);
	}

	return 0;
}

int checkDirectory()
{
	char pathArr[BUFSIZE];
	if (getcwd(pathArr, sizeof(pathArr)) != NULL)
	{
	  printf("%s\n", pathArr);
	}
	else
	{
	  perror("[ERROR: unknown error occured during path
finding]");
	}
	return 0;
}

int changeDirectory(Command command)
{
```

```
     //check if cd has next arguments or not
     int check = command.last - command.first;

     if(check == 0)
     {
          char* homePath = getenv("HOME");
          if(chdir(homePath) == 1)
          {
               printf("failed to access home dir
'/home/~'\n");
          }
     }
     else if(check == 1)
     {
          if(chdir(command.argv[1]) == 1)
          {
               printf("failed to access dir '%s'\n",
command.argv[1]);
          }
     }
     else
     {
          printf("failed to access dir, invalid
arguments\n");
     }

     return 0;
}

int changeDirectoryToRoot(Command command)
{
     //check if cd has next arguments or not
     if((command.last - command.first) == 0)
     {
          if(chdir("/") == 1)
          {
               perror("failed to access roor dir
'root#'\n");
          }
     }
     else
     {
          printf("failed to access dir, invalid
arguments\n");
     }

     return 0;
}

int globForWC(Command command)
{
```

```
     //VARIABLES
     int retCheck = 0;
     int psStatus = 0;

     //object of class glob
     glob_t globBuff;

     //check num of tokens between indexs
     globBuff.gl_offs = (command.last - command.first);

     if (globBuff.gl_offs != 0)
     {
          retCheck = glob(command.argv[globBuff.gl_offs],
GLOB_DOOFFS, NULL, &globBuff);

          if(retCheck != 0)
          {
               //printf("[ERROR: unknown error occured
during wildcard command execution]\n");
               return 0;
          }

          //create new process
          pid_t pid = fork();

          if(pid == 0) //child-process
          {
               for(int i = 0, k = 0; i<=
globBuff.gl_offs; i++, k++)
               {
                    globBuff.gl_pathv[i] =
command.argv[k];
                    //execvp(command.argv[0],
command.argv[1], &globBuff.gl_pathv[i]);
                    execvp(command.argv[0],
&globBuff.gl_pathv[i]);
                    printf("[ERROR: command execution
failed]\n");
                    exit(0);
               }

          }
          else if(pid > 0) //parent-process
          {
               waitpid(-1, &psStatus, 0);
          }
     }
     else
     {
          printf("[ERROR: too less arguments]\n");
     }
```

```
     //clear glob
     globfree(&globBuff);
     return 0;
}

int backgdCmdExecuter(Command command)
{
     pid_t pid;

     if ((pid=fork()) == 0)
    {
     pid = setsid();

        if(pid < 0)
        {
          printf("[0.0]\n");
             printf("Fail to create a new session\n");
               exit(1);
        }
        else
        {
          //if < or > are in use for output or input
redirection

     if((command.stdin_file!=NULL)||(command.stdout_file!
=NULL))
                {
                     int fileNo;
                     char * fileName = NULL;
                     if(command.stdin_file!=NULL)
                     {
                          fileName = command.stdin_file;
                          fileNo = 0;
                     }else if(command.stdout_file!=NULL)
                     {
                          fileName = command.stdout_file;
                          fileNo = 1;
                     }
                     command.stdin_file=NULL;
                     command.stdout_file=NULL;
                     fileRdExecuter(command, fileNo,
fileName);
                     execvp(command.argv[0],
command.argv);
                }else if(execvp(command.argv[0],
command.argv) < 0)
                {
                     //perror("Error executing");
                     exit(1);
                }
```

```
            //exit(0);
              signal(SIGINT, SIG_DFL);
            fprintf(stderr, "ERROR %s no such
program\n", command);
            exit(0);
        }
    }
    //no need to wait
}

int normalCmdExecuter(Command command)
{
     pid_t pid;

     pid = fork();
     if(pid == 0) //Child - process
     {
          //if < or > are in use for output or input
redirection

     if((command.stdin_file!=NULL)||(command.stdout_file!
=NULL))
          {
               int fileNo;
               char * fileName = NULL;
               if(command.stdin_file!=NULL)
               {
                    fileName = command.stdin_file;
                    fileNo = 0;
               }else if(command.stdout_file!=NULL)
               {
                    fileName = command.stdout_file;
                    fileNo = 1;
               }
               command.stdin_file=NULL;
               command.stdout_file=NULL;
               fileRdExecuter(command, fileNo, fileName);
               execvp(command.argv[0], command.argv);
          }
          else if (execvp(command.argv[0], command.argv)
< 0)
          {
               //perror("Error executing");
               exit(1);
          }
          //printf("Command '%s' not found, what you
mean?\n", *command.argv);
          //exit(0);
          signal(SIGINT, SIG_DFL);
        fprintf(stderr, "ERROR %s no such program\n",
command);
```

```
        exit(0);
    }
    else if(pid > 0) //Parent - process
    {
        waitpid(-1, NULL, 0);
        //exit(0);
    }
    //return 0;
}

void pipeCmdExecuter(Command pipedCommands[], int pipes)
{


    Command singleCmd;
    pid_t pid;
    int status;
    int pids[pipes];
    int backgroundCheck = 0; // if 1, then commands to
be executed in the background

    int pipeNo = pipes - 1; //Actual number of pipes.
Variable pipes contains number of commands involved in
the piping process

    int pipesArray[pipeNo][2]; // Array used for
creating pipes, second column contains pipe 0 and 1 for
read and write

    //creating pipes
    int i = 0;
    while(i < pipeNo)
    {
        int pipeCheck = pipe(pipesArray[i]);
        if(pipeCheck < 0)
        {
            perror("Error creating pipes");
            exit(1);
        }
        i ++;
    }

    //linking pipes together. creates and uses child
processes for that
    i = 0;
    int j = 0;
    while(i < pipes)
    {
        singleCmd = pipedCommands[i];
        pid = fork();
```

```
          if(strcmp(singleCmd.sep, "&") == 0)
          {
               backgroundCheck = 1;
          }

          //Child process
          if (pid == 0)
          {
               if (i == 0)
//first pipe
               {
                    dup2(pipesArray[i][1],1);
               }else if (i == pipeNo)
     //Last pipe
               {
                    dup2(pipesArray[i - 1][0], 0);
               }else
               {
                    dup2(pipesArray[i - 1][0], 0);
//Connects processes if pipe is in between
                    dup2(pipesArray[i][1], 1);
               }

               //if < or > are in use for output or input
redirection

     if((singleCmd.stdin_file!=NULL)||(singleCmd.stdout_f
ile!=NULL))
               {

                    int fileNo;
                    char * fileName = NULL;
                    if(singleCmd.stdin_file!=NULL)
                    {
                         fileName = singleCmd.stdin_file;
                         fileNo = 0;
                    }else if(singleCmd.stdout_file!=NULL)
                    {
                         fileName =
singleCmd.stdout_file;
                         fileNo = 1;
                    }
                    singleCmd.stdin_file=NULL;
                    singleCmd.stdout_file=NULL;
                    fileRdExecuter(singleCmd, fileNo,
fileName);
               }

               j = 0;
//Closes pipes so they execute correctly
               while(j < pipeNo)
```

```
                {
                        close(pipesArray[j][0]);
                        close(pipesArray[j][1]);
                        j++;
                }

                if (execvp(singleCmd.argv[0],
singleCmd.argv) < 0)
                {
                        //perror("Error executing");
                        exit(1);
                }
        }
        pids[i] = pid;
        i++;
    }
    i = 0;
    while(i<pipeNo)
    {
        close(pipesArray[i][0]);
        close(pipesArray[i][1]);
        i++;
    }
    // Parent process waiting for all children to finish
    if(backgroundCheck == 0){
        for(i = 0; i < pipes; i ++)
        {
            //waitpid(pid,&status, 0 );
            waitpid(-1, NULL, 0);
        }
    }
}

void zombieChild()
{
    int more = 1;
    pid_t pid;
    int status;

    while(more)
    {
        pid = waitpid(-1, &status, WNOHANG);
        if (pid <= 0)
        {
            more = 0;
        }
    }
}

int fileRdExecuter(Command command, int fileNo, char
*fileName)
```

```
{
     int fileDirection = 0;



     //If Input redirection
     if(fileNo == 0)
     {
          fileDirection = open(fileName, O_RDONLY);

     }else if(fileNo == 1)                          //If
Output redirection
     {
          fileDirection = open(fileName,
O_WRONLY|O_CREAT, 0766);
     }else
     {
          return 1;

     }
     if(fileDirection < 0)
     {
          perror("Error while opening the file.\n");
     }else
     {
          dup2(fileDirection, fileNo);
          close(fileDirection);
     }
     fileName == NULL;

}


int daemon_init(void)
{
     pid_t pid;
     if ((pid = fork()) < 0)
     {
          return (-1);
     }
     else if (pid != 0)
     {
          printf("server pid=%d\n", pid);
          exit(0);
     }
     /* child continues */
     setsid(); /* become session leader */
     chdir("/"); /* change current directory */
     umask(0); /* clear umask */
     return (0);
}
```

```
int try_deamon(void)
{
     FILE *log;
     pid_t pid;
     // create a log file
     log = fopen(logfile, "w+");
     if (!log)
     {
          fprintf(stderr, "cannot create log file %s\n",
logfile);
          exit(1);
     }
     // turn the process into a daemon
     daemon_init();
     // log daemon pid
     pid = getpid();
     fprintf(log, "My pid is %d\n", pid);
     fflush(log);
     // pretend to do something
     while (1)
     {
          sleep(100);
          fprintf(log, "Who says that I am useless?\n");
          fflush(log);
     }
}

void freeCommand(Command *command)
{
     command->first = 0;
     command->last = 0;
     command->sep = NULL;
     int i = 0;
     while(command->argv[i]!= NULL)
     {
          command->argv[i] = NULL;
          i++;
     }
     command->stdin_file = NULL;
     command->stdout_file = NULL;
}
```

Token.h

```
/*
 * File   : token.h
 * Author : Muhammad Amir Hamza
 * Date   : 2021.03.10
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

#define MAX_TOKEN_NUM 100
```

int tokenize (char* inputLine, char* token[MAX_TOKEN_NUM]);

Token.c

```
/*
 * File  : token.c
 * Author : Muhammad Amir Hamza
 * Date  : 2021.03.10
*/

#include "token.h"

int tokenize (char* inputLine, char* token[MAX_TOKEN_NUM])
{
	const char delimiter[2] = " ";
	int count = 0;
	char* ptr;

	if(strlen(inputLine) > 0)
	{
		ptr = strtok(inputLine, delimiter);
		token[count] = ptr;
		//printf("%s", (token[count]));
		while(ptr != NULL && count < MAX_TOKEN_NUM)
		{

			count++;
			ptr = strtok(NULL, delimiter);
			token[count] = ptr;
			//printf("%s", (token[count]));
		}
	}
	return count;
}
```

Command.h

```
// file:       command.h for Week 9
// purpose;          to separate a list of tokens into a
sequence of commands.
// assumptions:          any two successive commands in
the list of tokens are separated
//             by one of the following command
separators:
//                   "|"  - pipe to the next command
//                   "&" - shell does not wait for the
proceeding command to terminate
```

```
//                  ";" - shell waits for the proceeding
command to terminate
// author:          HX
// date:       2006.09.21
// last modified:   2006.10.05
// note:       not thoroughly tested therefore it may
contain errors

#define MAX_NUM_COMMANDS  1000

// command separators
#define pipeSep  "|"                                // pipe
separator "|"
#define conSep   "&"                                //
concurrent execution separator "&"
#define seqSep   ";"                                //
sequential execution separator ";"

struct CommandStruct {
   int first;          // index to the first token in the
array "token" of the command
   int last;           // index to the first token in the
array "token" of the command
   char *sep;         // the command separator that
follows the command,  must be one of "|", "&", and ";"
   char **argv;       // an array of tokens that forms a
command
   char *stdin_file;  // if not NULL, points to the file
name for stdin redirection
   char *stdout_file; // if not NULL, points to the file
name for stdout redirection
};

typedef struct CommandStruct Command;  // command type


// purpose:
//       separate the list of token from array "token"
into a sequence of commands, to be
//       stored in the array "command".
//
// return:
//       1) the number of commands found in the list of
tokens, if successful, or
//       2) -1, if the the array "command" is too small.
//       3) < -1, if there are following syntax errors
in the list of tokens.
//            a) -2, if any two successive commands are
separated by more than one command separator
//            b) -3, the first token is a command
separator
```

```
//             c) -4, the last command is followed by
command separator "|"
//
// assume:
//        the array "command" must have at least
MAX_NUM_COMMANDS number of elements
//
//  note:
//        1) the last command may be followed by "&", or
";", or nothing. If nothing is
//           followed by the last command, we assume it
is followed by ";".
//        2) if return value, nCommands >=0, set
command[nCommands] to NULL,
//
int separateCommands(char *token[], Command command[]);
```

Command.c

```
// file:       command.c for Week 9
// purpose;          to separate a list of tokens into a
sequence of commands.
// assumptions:          any two successive commands in
the list of tokens are separated
//             by one of the following command
separators:
//                  "|"  - pipe to the next command
//                  "&" - shell does not wait for the
proceeding command to terminate
//                  ";" - shell waits for the proceeding
command to terminate
// author:          HX
// date:       2006.09.21
// note:       not thoroughly tested therefore it may
contain errors

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "command.h"

// return 1 if the token is a command separator
// return 0 otherwise
//
int separator(char *token)
{
    int i=0;
    char *commandSeparators[] = {pipeSep, conSep, seqSep,
NULL};

    while (commandSeparators[i] != NULL) {
        if (strcmp(commandSeparators[i], token) == 0) {
            return 1;
```

```
        }
        ++i;
    }

    return 0;
}

// fill one command structure with the details
//
void fillCommandStructure(Command *cp, int first, int
last, char *sep)
{
     cp->first = first;
     cp->last = last - 1;
     cp->sep = sep;
}

// process standard in/out redirections in a command
void searchRedirection(char *token[], Command *cp)
{
     int i;
     for (i=cp->first; i<=cp->last; ++i) {
         if (strcmp(token[i], "<") == 0) {   // standard
input redirection
              cp->stdin_file = token[i+1];
              ++i;
         } else if (strcmp(token[i], ">") == 0) { //
standard output redirection
              cp->stdout_file = token[i+1];
              ++i;
         }
     }
}

// build command line argument vector for execvp function
void buildCommandArgumentArray(char *token[], Command
*cp)
{
    int n = (cp->last - cp->first + 1)    // the numner
of tokens in the command
         + 1;                             // the element
in argv must be a NULL

     // re-allocate memory for argument vector
     cp->argv = (char **) realloc(cp->argv, sizeof(char
*) * n);
     if (cp->argv == NULL) {
         perror("realloc");
         exit(1);
     }
```

```
     // build the argument vector
     int i;
     int k = 0;
     for (i=cp->first; i<= cp->last; ++i ) {
         if (strcmp(token[i], ">") == 0 ||
strcmp(token[i], "<") == 0)
             ++i;    // skip off the std in/out
redirection
         else {
             cp->argv[k] = token[i];
             ++k;
         }
     }
     cp->argv[k] = NULL;
}

int separateCommands(char *token[], Command command[])
{
     int i;
     int nTokens;

     // find out the number of tokens
     i = 0;
     while (token[i] != NULL) ++i;
     nTokens = i;

     // if empty command line
     if (nTokens == 0)
          return 0;

     // check the first token
     if (separator(token[0]))
          return -3;

     // check last token, add ";" if necessary
     if (!separator(token[nTokens-1])) {
         token[nTokens] = seqSep;
         ++nTokens;
     }

     int first=0;   // points to the first tokens of a
command
     int last;      // points to the last tokens of a
command
     char *sep;     // command separator at the end of a
command
     int c = 0;     // command index
     for (i=0; i<nTokens; ++i) {
         last = i;
         if (separator(token[i])) {
             sep = token[i];
```

```
            if (first==last)  // two consecutive
separators
                return -2;
            fillCommandStructure(&(command[c]), first,
last, sep);
            ++c;
            first = i+1;
        }
    }

    // check the last token of the last command
    if (strcmp(token[last], pipeSep) == 0) { // last
token is pipe separator
         return -4;
    }

    // calculate the number of commands
    int nCommands = c;

    // handle standard in/out redirection and build
command line argument vector
    for (i=0; i<nCommands; ++i) {
        searchRedirection(token, &(command[i]));
        buildCommandArgumentArray(token, &(command[i]));
    }

    return nCommands;
}
```

makefile

```
# makefile for c2
# the filename must be either Makefile or makefile

all: main

main: main.o ./data/token.o ./data/command.o
    gcc -Wall main.o ./data/token.o ./data/command.o -o
exe

main.o: main.c ./data/token.h ./data/command.h
    gcc -c main.c

command.o: ./data/command.c ./data/command.h
    gcc ./data/command.c -c

token.o: ./data/token.c ./data/token.h
    gcc -c ./data/token.c

clean:
    rm *.o
```