IRIS NICOLE CARSON 4 BS CpE February 6, 2022
ANTONIO RAFAEL CASTRO
JOSHUA MANUEL LOUISE KEMPIS

Project 1 Final Report DRAFT #4 (EARLY SUBMISSION)

Rudimentary Shell Interpreter

Overview

A C++ program was created to simulate a basic shell interface that allows UNIX-based user commands to be inputted and processed in a virtual terminal simulation. The commands are parsed into a process depending on the operator being called: pipe ('|'), input ('<'), and output ('>'). These operators utilize built-in system calls like fork(), exec(), wait(), dup2(), and pipe() which are native to Linux and MacOS systems.

Code Collaboration Overview

Each member of the group added inputs into the structure of the final code which included the parsing function, the different implementations for each operator, and the system calls that would be used to run the different commands. A minor adjust would be made for the macOS version of the cpp file which removes the `<windows.h>` header and adjust the "`_popen`" and "`_pclose`" command to "`popen`" and "`pclose`" respectively.

Implementation of Functions and Operations

Legacy Functions

```
void fileRead (string filename, string &contents)
{
  string line;
  ifstream file (filename);

  if (file.is_open())
  {
    while (getline (file, line))
    {
    cout << line << endl;
    contents = contents + line + "\n";
    }
  file.close();
  }
  else cout <<"File does not exist"<<endl;
}

void fileWrite (string filename, string line)
{
  ofstream file(filename);
  file << line;
  file.close();
}
```

A fileRead() and fileWrite() system was implemented for accessing files in the directory. These functions are used to read input files and parse them into the necessary input when the pipe or input operator is called.

This function is also used to record the output of a process and write them into an output file when needed.

```
void parsing(string cmd, vector <string> &comms,
             string &c1, string &c2, int &mode)
{
  stringstream ss(cmd);
  string word;
  while (ss>>word)
  {
    comms.push_back(word);
  }
  for (int i = 0; i < comms.size(); i++)
  {
    if (comms[i] == "|" || comms[i] == ">" || comms[i] == "<")
  {
      if (comms[i] == "|") mode = 1;
    if (comms[i] == ">") mode = 2;
    if (comms[i] == "<") mode = 3;

      for (int j=0; j < i; j++)
    {
      c1 = c1 + comms[j] + ' ';
    }
    //cout<<call<<endl;

    for (int k = i+1; k < comms.size(); k++)
    {
      c2 = c2 + comms[k] + ' ';
    }
    //cout<<file<<endl;
    }
  else mode = 0;
  }
```

The parsing() function breaks down the string command that hte user inputs into key parameters that would be used in the different operations later.

This method is done by applying the stringstream parsing system that was done in previous projects in order to isolate the input commands in groups according to their respective parameters that would be needed when executing the command itself.

This is also useful in identifying the mode of the operation whether it's a pipe ('|'), input ('<'), or output ('>').

Simulation Output Review

Basic Shell Command Execution

```
/Users/JayMB/Documents/GitHub/engg126-project1-cck $  ls
Archive.zip
ENGG 126 - Project 1 Final Report DRAFT - KEMPIS.pdf
ENGG 126 - Project 1 First Progress Report - KEMPIS.pdf
ENGG 126 - Project 1 Second Progress Report - KEMPIS.pdf
ENGG 126 - Project 1 Third Progress Report - KEMPIS.pdf
aaaaaaah.cpp
draft
draft.cpp
draft.dSYM
in.txt
log.txt
out.txt
output.txt
project1
project1.cpp
project1.dSYM
shell-interpreter
shell-interpreter-macos
shell-interpreter-macos.cpp
shell-interpreter-macos.dSYM
shell-interpreter.cpp
shell-interpreter.dSYM
shell-proj.cpp

/Users/JayMB/Documents/GitHub/engg126-project1-cck $
```

Using the "`ls`" command from the standard list of UNIX commands, the shell interpreter is able to pass the command to its execute function and is able to list down all the files that are currently in the directory.

Input (‘< ‘) *Operator*

```
/Users/JayMB/Documents/GitHub/engg126-project1-cck $  cat < in.txt
addz
x
dirf
h
etw
returnrtyr
unistddfg

hf
thet
yert
yrty
unistdbs
dirwf
vector
sdf

/Users/JayMB/Documents/GitHub/engg126-project1-cck $
```

During the parsing process, if the input command contains the ‘ < ‘ operator, it puts the shell interpreter into input mode which requests a command and a file input.

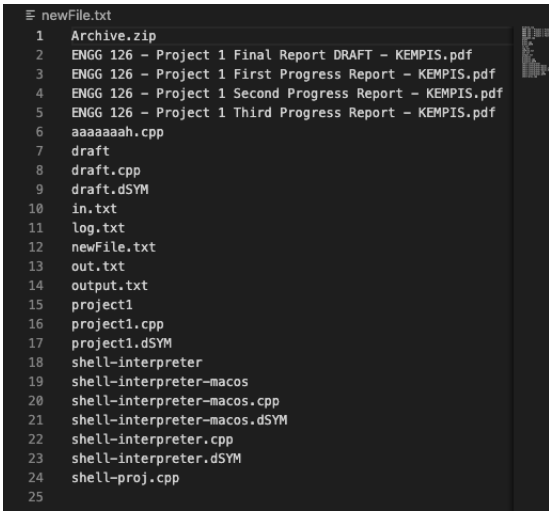
In this case, the command `cat` is used to display the contents of its input which is the `in.txt` file

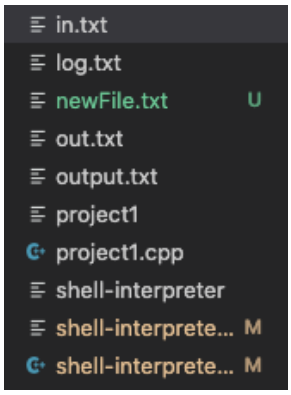
Output (‘ > ‘) *Operator*

```
/Users/JayMB/Documents/GitHub/engg126-project1-cck $  ls > newFi
txt

/Users/JayMB/Documents/GitHub/engg126-project1-cck $
```

The same `ls` command was performed but with an output operator that takes the supposed result of the command and prints it unto a file named `newFile.ttx`





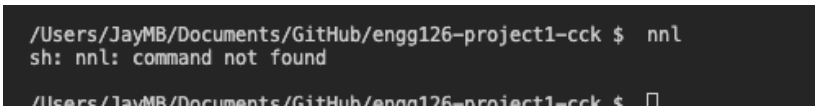
Comparing from the basic input command output from above, the > operator requests for a file destination or a filename on where the data should go after it is processed. It then takes all the output of the command and processes it in this newly created file. For existing files, these data would overwrite the previous data in the file.

Pipe (‘ | ‘) Operator

The pipe command is implemented by taking the output of the first command on the left side of the operator as the input for the command to the right of the ‘ | ‘ operator.

For this case, the cat command reads the in.txt file and takes the output as the input to the sort command which displays the sorted list of words on the terminal

Error Handling



Suppose the user enters a command that cannot be recognized by the shell interpreter, it returns an error message and returns back to requesting a new input from the user

Documentation Link

The GitHub repository for the project is made publicly available through the following:

GitHub Link	https://github.com/jaykempis/engg126-project1-cck