Programming Assignment 2:

MyShell

Description: A shell is a computer program that presents a command line interface which allows you to control your computer using commands entered with a keyboard instead of controlling graphical user interfaces (GUIs) with a mouse/keyboard combination. Now there are many different types of shells, but the objective was to write a simple shell that took basic commands, can handle sequential commands, file redirect, and use pipelines. I was able to implement a simple shell that can take basic commands and handle sequential commands. This implementation cannot use file redirection and pipelines.

Prerequisites (how to use it): The program was designed to run on a computer utilizing the UNIX/LINUX operating system. Therefore, you need either a computer with UNIX/LINUX installed or a virtual box running UNIX/LINUX on your Windows system. Next you need to open your terminal on your system and run the command “sudo g++ -o <whatever name you choose> main.cpp” this will compile the c++ script. Once completed you can now enter “./<the name you entered> <ink>” and watch the program do its job.

How it works: The Program accepts whatever command it was given and tries to call the corresponding method indicated by the command. The commands are ls, pwd, wc, sort, grep, clear, and exit. Ls will list all the files and folders in the current directory. Pwd prints the path of the current directory. Wc counts the words, lines, and characters of the file given. Grep takes in a word to search for and prints the lines in the file that match the word. Clear will clear the previous lines entered in the shell. Lastly, exit will exit the program. Once all of these commands have been checked to make sure they have a linked method call the output is printed and if they don’t fit an error will be shown. The source code with a more detailed look into the workings of the program is shown below. The comments explain what each section of code does for a better understanding. The text files used for testing are included in the folder titled Text Documents.

#**include** <iostream>

#**include** <string.h>

#**include** <algorithm>

#**include** <vector>

#**include** <sstream>

#**include** <dirent.h>

#**include** <unistd.h>

#**include** <fstream>

#**include** <sys/types.h>

**using** **namespace** std;

**void** **list**() {

**char** tmp[4096]; // creates a buffer to hold the path of the given directory

getcwd(tmp, 4096);

DIR \*pwd = opendir(tmp);// captures the current working directory and stores it in pwd

**struct** dirent \*dirRead;

vector<string> files; // creates a string vector to hold all the files/folders

**if** (pwd != nullptr) {

**while** ((dirRead = readdir(pwd)) != nullptr) {

files.push\_back(dirRead->d\_name); // goes through the entire directory and adds the files/folders to the vector

}

closedir (pwd);

} **else** {

perror ("opendir"); // prints an error fi the directory doesnt exist

cout << "Error: Exit failure" << endl;

}

cout << "Your current working directory is: " << tmp << " and contains: " << endl; // prints out the files/folders in the directory by traversing the vector

**for** (**int** i = 0; i < files.size(); i++) {

cout << files[i] << " ";

}

cout << endl;

}

**void** **printDir**() {

**char** tmp[4096]; // creates a buffer to hold the path of the given directory

getcwd(tmp, 4096); // gets the current working directory and stores it

cout << "Your current working directory is: " << tmp << endl; // prints the pwd

}

**void** **wordCount**(string n) {

**int** lineCount = 0; // creates counter variables to hold the number of lines words and chars

**int** wordCount = 0;

**int** charCount = 0;

string fileLines[4096]; // string that holds the lines of the file in order

ifstream myFile;

myFile.open(n); // opens the file with the given name

string line;

**int** i = 0;

**if** (myFile.is\_open()) { // traverses the file and adds every line to the array

**while** (myFile.good()) {

getline(myFile, line);

fileLines[i] = line;

i++;

}

} **else** {

cout << "Couldn't open file" << endl; // prints an error if the file couldnt be opened

}

**for** (**int** k = 0; k < i; k++) { // traverses every line in the file

lineCount++; // increases the amount of lines by 1 per line read

charCount += fileLines[k].size();// increases the amount of characters by the number of characters in each line per line read

stringstream **stream**(fileLines[k]);

string oneWord;

**while** (stream >> oneWord) { // counts every word in the file and increases the word count by one per word

wordCount++;

}

}

cout << lineCount << " lines, " << wordCount << " words, " << charCount << " characters in file " << n << endl;

}

**void** **sort**(string n) {

vector<string> lines; // vector of strings that holds the lines of the file

ifstream myFile;

myFile.open(n); // opens the file with the given name

string line;

**if** (myFile.is\_open()) {

**while** (myFile.good()) {

getline(myFile, line);

lines.push\_back(line); // traverses the file and adds every line to the vector

}

} **else** {

cout << "Couldn't open file" << endl;// prints an error if the file couldnt be opened

}

sort(lines.begin(), lines.end()); // sorts the lines in the file

**for** (**int** k = 0; k < lines.size(); k++) {//prints out each line in the sorted order

cout << lines[k] << endl;

}

}

**void** **grep**(string search, string n) {

string fileLines[4096]; // string that holds the lines of the file in order

ifstream myFile;

myFile.open(n); // opens the file with the given name

string line;

**int** i = 0;

**if** (myFile.is\_open()) { // traverses the file and adds every line to the array

**while** (myFile.good()) {

getline(myFile, line);

fileLines[i] = line;

i++;

}

} **else** {

cout << "Couldn't open file" << endl;// prints an error if the file couldnt be opened

}

**for** (**int** k = 0; k < i; k++) { // goes through every line in the file

**size\_t** found = fileLines[k].find(search);

**if** (found != std::string::npos) {

cout << fileLines[k] << endl;// prints out the line with the matching word

}

}

}

**void** **clear**() {

cout << string(100,'\n'); // prints out 100 new lines to clear the shell

}

**void** **whatCmd**(string cmd) {

string c;

vector<string> commands;// creates a vector that holds the command and its argument

stringstream **i**(cmd);

**while** (getline(i, c, ' ')) // adds each seperate command to the vector

commands.push\_back(c);

**if** (commands[0] == "ls") {// checks the command for one of the options below and runs the corresponding command with or without the argument

list();

} **else** **if** (commands[0] == "pwd") {

printDir();

} **else** **if** (commands[0] == "wc") {

**if** (commands.size() == 2) {

wordCount(commands[1]);

} **else**

cout << "Error: wc needs a file" << endl;

} **else** **if** (commands[0] == "sort") {

**if** (commands.size() == 2) {

sort(commands[1]);

} **else**

cout << "Error: sort needs a file" << endl;

} **else** **if** (commands[0] == "grep") {

**if** (commands.size() == 3) {

grep(commands[1], commands[2]);

} **else**

cout << "Error: grep needs a file" << endl;

} **else** **if** (commands[0] == "clear") {

clear();

} **else** **if** (commands[0] == "exit") {

exit(0);

} **else** {

cout << "Error: incorrect command" << endl;// if the command is incorrect print the error

}

}

**void** **sequentialCmd**(string cmd) {

string c;

vector<string> seqCommands;// creates a vector that holds the commands seperated by ;

stringstream **i**(cmd);

**while** (getline(i, c, ';'))

seqCommands.push\_back(c); // adds each seperate command to the vector

**for** (**int** i = 0; i < seqCommands.size(); i++) { // runs each command sequentially

**if** (seqCommands[i].at(0) == ' ')

whatCmd(seqCommands[i].substr(1));

**else**

whatCmd(seqCommands[i]);

}

}

**void** **pipedCmd**(string cmd) {

cout << "not implemented" << endl;

}

**int** **main**() {

string command;

**while** (command != "exit") {

cout << "Enter your command: " << endl;

getline(cin, command);

**if** (command.find(";") != std::string::npos) {

sequentialCmd(command);

} **else** **if** (command.find("|") != std::string::npos) {

pipedCmd(command);

} **else** {

whatCmd(command);

}

}

**return** 0;

}