CSE 384 Section 02 (Spring 2020) Midterm instructor's solution

By default, Linux (Unix) shell commands receive input from the keyboard (stdin) and produce output to user's terminal as text (stdout). Linux (Unix) provides a feature to alter the input source of a command as well as where its output and error messages are sent to.

a. Define and describe this feature with as much detail as you can?

b. Why is this feature so useful?

(25 points - full credit will depend on the completeness of your answer)

Solution:

In Linux and other Unix-like systems provide a feature known as I/O redirection.

In Linux (Unix) everything is considered a file.

Command line programs send output to a file called **standard output (stdout** – connected to the screen by default).

Command line programs usually send error and status messages to another file called **standard error (stderr** – also connected to screen by default).

Command line programs receive input from yet another called file **standard in (stdin** – connected to the keyboard by default)

1. **stdin or file descriptor 0** – is file connected to the keyboard. Command line programs read input from this file.
2. **stdout or file descriptor 1** – is file connected to the display. Command programs send output to this file
3. **stderr or file descriptor 2** – is a file connected to the display. Programs send status/error messages to this file.

I/O redirection - allows us to alter the input source of shell command as well as where its output and error messages are sent to. This is made possible by the "<" and ">" redirection operators. Specifically, redirection allows us to change where output (**stdout/stderr**) goes (e.g. the screen or another file), and where input (**stdin**) comes from (e.g. the keyboard or another file).

Examples:

`ls -l /usr/bin > ls-output.txt` - redirects stdout to file "ls-output.txt"

`ls -l /bin/usr 2> ls-error.txt` - redirects stderr to file "ls-error.txt"

`cat < lazy_dog.txt` - redirect stdin to come from text file "lazy_dog.txt"

b. The most useful and powerful thing you can do with I/O redirection is to **connect multiple commands together with what are called pipelines.** With pipelines, the standard output of one command is fed into the standard input of another. This is accomplished with the pipe (|) operator.

Example:

`ls -l /usr/bin | less` - redirect *stdout* of the ls program to *stdin* of less program

References:

https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=78

https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=85

http://linuxcommand.org/lc3_lts0070.php

2. Assume the following files exist in your home directory: "foo.txt" and "bar.txt"

a. Write a Linux command to make "foo.txt" writable and executable for the owner, readable for the group, and no permissions for everyone else**. (12.5 points)**

b. Nine characters of the file attributes (called the file mode), represent the read, write, and execute permissions for the file's owner, the file's group owner, and everybody else **(12.5 points)**

Owner	Group	World
rwx	rwx	rwx

Internally these file mode attributes are represented by 9 binary bits.

For example: the permissions attributes: "rwxrwxrwx" = 111111111_2 = (binary) and 777_8 (octal)

The permission attributes for "bar.txt" = r-xr--r-x

Please write the 9 binary permission bits (owner/group/world) for "bar.txt" ---------

Solution: (25 total points): a. 12.5, b. 12.5

a. Octal notation: **chmod 340 foo.txt** *or* Symbolic notation: **chmod u+wx,g+r,o-rwx foo.txt**
b. **101100101_2**

3. Use the Linux (Unix) *find* program and the word count program (*wc)* to write a command to find and count all of the files in your home directory (and the sub directories) with size greater than 1k (1000 bytes). The output should be the count of the files found. (25 points).

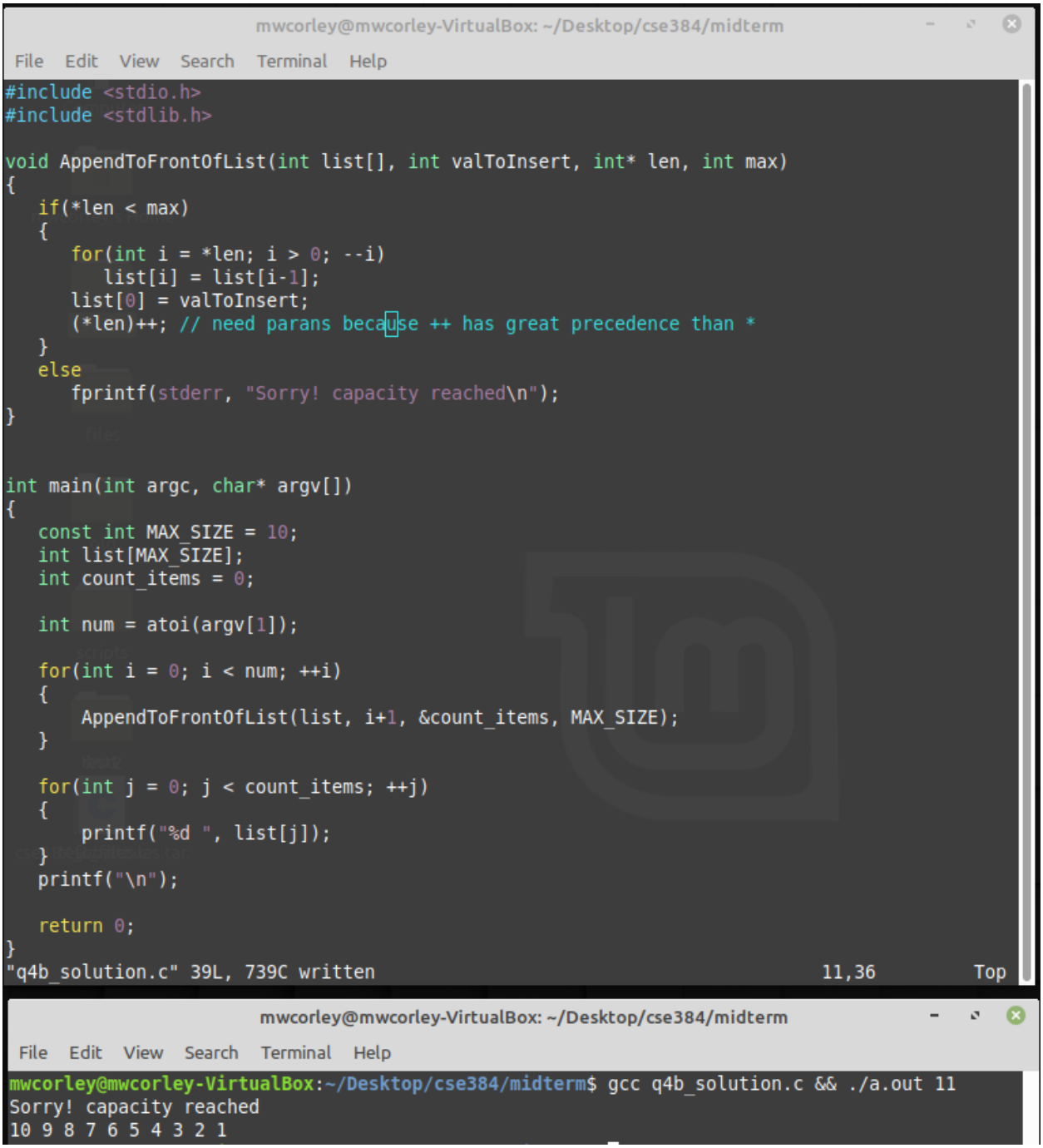
   Solution: `find ~ -type f -size +1k | wc -l`

4. Complete the function: ***void AppendToFrontOfList(int list[], int valToInsert, int* len, int max)*** shown below (25 points)

   a. accepts an integer *valToInsert*. If space exists in the array *list* (*max is the capacity*) then *valToInsert* is inserted to the front of the array (*list[0]*), and *len* is incremented by 1. If the *max* number of elements is reached (i.e. capacity) then don't insert, instead print an error message on the standard error stream and return
   b. what is the output of the program if run with the following command line:

      gcc q4b.c && ./a.out 11

   Solution:


mwcorley@mwcorley-VirtualBox: ~/Desktop/cse384/midterm
File Edit View Search Terminal Help
#include <stdio.h>
#include <stdlib.h>
void AppendToFrontOfList(int list[], int valToInsert, int* len, int max)
{
if(*len < max)
{
for(int i = *len; i > 0; --i)
list[i] = list[i-1];
list[0] = valToInsert;
(*len)++; // need parans because ++ has great precedence than *
}
else
fprintf(stderr, "Sorry! capacity reached\n");
}
int main(int argc, char* argv[])
{
const int MAX_SIZE = 10;
int list[MAX_SIZE];
int count_items = 0;
int num = atoi(argv[1]);
for(int i = 0; i < num; ++i)
{
AppendToFrontOfList(list, i+1, &count_items, MAX_SIZE);
}
for(int j = 0; j < count_items; ++j)
{
printf("%d ", list[j]);
}
printf("\n");
return 0;
}
"q4b_solution.c" 39L, 739C written
11,36
Top
mwcorley@mwcorley-VirtualBox: ~/Desktop/cse384/midterm
File Edit View Search Terminal Help
mwcorley@mwcorley-VirtualBox:~/Desktop/cse384/midterm$ gcc q4b_solution.c && ./a.out 11
Sorry! capacity reached
10 9 8 7 6 5 4 3 2 1