CSC209H Worksheet: Compiling and Running Programs

To make sure you understand the terminology we have been using, answer the following questions and then discuss your answers with two or three people sitting nearby.

1. Suppose you have a program named `prog.c`. What is the instruction you would type on the command line to compile this program and create an executable named `prog`?

enables debugging

gcc -Wall -std=gnu99 -g -o prog prog.c

standard

2. For each of the arguments you gave to the `gcc` command, write down what it means.

printall warnings

give the executable file the name prog

3. Now that you have an executable named `prog` in your current working directory, give the command to run that executable with the command-line arguments `-k 3 myfile`.

./prog -k 3 myfile

4. Assume that the executable is in your *parent* directory; give the command to run this executable without any command-line arguments.

../prog

5. Assume you have changed back into the same directory as the executable. Give the command to run the executable where the resulting output is redirected to a file named `test1.out`.

./prog > test1.out

6. When you run the program, it interacts with the user expecting the user to type input. Imagine that up until now you've been providing input from the keyboard. Give the command to run the program and redirect the input so that the executable reads from the file `somefile.txt`.

./prog < somefile.txt

7. Put it all together. Show the command to run the executable `prog` with the command-line arguments `-k 3 myfile`, reading input from standard input redirected from `somefile.txt` and redirecting the output to `test1.out`.

input redirection

./prog -k 3 myfile < somefile.t~~e~~xt > test1.out |

arguments

output

8. Run `prog` with a command-line argument of `hello`, and pipe the output to the Unix utility program `wc`. This allows you to count the number of lines, words, and characters this program outputs.

pipe symbol

./prog hello | wc

9. Write a shell command to remove all the files in the current working directory that end in `.o`

rm *.o

10. Suppose you have a directory with a bunch of C source code files. You would like to print out all the unique `#include` files there are included in these source files. Use the commands `grep`, `cut`, `sort`, `uniq` and pipes to display the unique list of include lines.

For example, when I run the full pipeline of commands on the files in `/u/csc209h/winter/pub/bin`, the output is:

```
#include <dirent.h>
#include "helper.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

- `grep` should look only in files that end in `.c`
- Look at the output of your grep command. Which character could you use as a field delimiter to isolate the include part of the line from the filename that grep also outputs?
- If you haven't used `cut` before, you will want to look at the man page. Run `man cut` to read how this command works.
- Build up each component of the pipeline one command at a time and see if the output is what you would expect.

grep include /u ..../bin/*.c | cut -d ":" -f 2

| sort | uniq (all on one line)