The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 214– Programming Principles and Practice

Assignment 6

Date Due: November 27, 2020

Total Marks: 21

1 Submission Instructions

- Assignments must be submitted using Canvas.
- C programs must conform to the C11 standard.
- BASH commands must be able to run on Tuxworld. Test your solutions on tuxworld!
- If asked, submit `.txt` files, not `.doc`, `.rtf`, etc. There can be weird side effects depending on the machine that is being used to open them.
- Always include the following identification in your solutions: your name, NSID, student ID, instructor's name, course name, and course section number.
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.
- **VERY IMPORTANT**: Canvas is very fragile when it comes to submitting multiple files. We insist that you package all of the files for all questions for the entire assignment into a **ZIP** archive file. This can be done with a feature built into the Windows explorer (Windows), or with the **zip terminal command** (LINUX and Mac). We **cannot accept** any other archive formats. This means no tar, no gzip, no 7zip. Non-zip archives will not be graded. We will not grade assignments if these submission instructions are not followed.
- Instructions on "how to create zip archives" can be found here `https://canvas.usask.ca/courses/9771/pages/how-to-zip-slash-compress-your-files`

2 Background

This background section serves as a "translation" for man pages that can be found in tuxworld. Even though this is provided to you it is still recommended that you spend some time looking at the man pages to see the translations for yourself. The use cases of the commands presented here will help you solve questions 3 and 4.

2.1 `tr(1)`

`tr` is short for translate. This command will translate, squeeze, or delete characters from stdin, and write them to stdout.

2.1.1 Usage

`tr <ch1> <ch2>` reads from *stdin* and writes this input to *stdout* but with each character `<ch1>` replaced with the charater `<ch2>`.[1]

`tr -s <ch1>` operates in the same fashion but will replace (or squeeze) any consecutive occurrences of the given character with a single occurrence of that character.

2.2 `cut(1)`

`cut` reads from *stdin* by default (a file name can also be given as an argument) and can (among other things) be used to select certain columns from an input stream where the text data is organized into columns (such as the output of `ps`) where each line has the same number of columns. The selected columns will then be written to to *stdout*.

2.2.1 Usage

Selecting a delimeter: The `cut -d` option is used to specifies the delimeter used in the file to separate columns. By default, cut will use `\t` as a delimiter, if you wish to set it as something else, the `-d` flag will allow you to do so. For example, if you wish to use spaces instead of tabs as a delimiter, you can use `cut -d' '`.

Selecting fields: The `cut -f` option select a specific field, or set of fields to extract and output. In the documentation for `cut`, they define a *field* as what in the textbook we have called a *column*. For example, with the `ps` command we talked about *columns* of output that show the owner, PID, etc. `cut` calls these fields. So if you want to extract certain *columns* (what `cut` calls a *field*), you can use the -f option.

Fields are numbered starting from 1, and you can specify multiple fields with a comma-separated list. For example, if you want to select the second and fourth columns (fields), you could use `cut -d2,4`.

The `-f` and `-d` options can be used at the same time.

Sidebar: Fields versus Columns in cut(1)

The man page for `cut` also references *columns* as well as *field*, but where the man page references columns, it is referring simply a character position, e.g. the *n*-th character on a line.

[1] `<ch1>` and `<ch2>` can also be strings of the same length, in which case each occurrence of the i-th character of `<ch1>` in the input is replaced with the i-th character of `<ch2>`.

3 Your Tasks

Question 1 (13 points):

It's pretty sus when a Crewmate just randomly appears behind you in a room when you know that you were alone. It's even more sus when a body is found but everyone claims not to be the Impostor. Pink is fed up with this, Brown has said that they've been in MedBay when a body was found in Security one too many times. They created a program that will list all of the possible vents on a map and all possible destinations from every vent, but they have no idea how to actually compile it. One option would be to completely re-factor the code into one giant file, but Pink has spent too much time on this and has put out a call for help. They're looking at you to help them, and it would be pretty sus of you to say no.

The Problem

You are provided with three .c files, two .h files, and three.txt files. **None of these files should be altered**:

list.c: This file contains the implemented functions `create_list`, `create_node`, `destroy_list`.

list-adders.c: This file contains the implemented functions `add_end`.

vents.c: This file contains the bulk of the program. It includes the `main` function which is in charge of calling the proper functions to create and populate an array that stores linked lists. Where the head of each linked list represents the starting vent, and each node following it represents possible vents you can travel to. Other implemented functions include: `find_index`, `read_file` , `print_vent_locations`, and `print_possible_starting_vents`.

list.h This is a header file that contains all function prototypes specific to linked lists, along with the linked list and node data structures.

vents.h This is the header file that contains all function prototypes and macros specific to vents.c.

skeld.txt This file is structured similarly to `travel-log.txt` from the midterm. At the top of the file there are two numbers, *N* and *K*, where *N* represents the number of unique vents, and *K* represents the number of `<src> <dst>` pairs. The difference between this file and the `travel-log.txt` file, is that these pairs are bidirectional. This file contains all possible pairs for the **Skeld** map in Among Us.

mira.txt This file is structured similarly to `travel-log.txt` from the midterm. At the top of the file there are two numbers, *N* and *K*, where *N* represents the number of unique vents, and *K* represents the number of `<src> <dst>` pairs. The difference between this file and the `travel-log.txt` file, is that these pairs are bidirectional. This file contains all possible pairs for the **Mira HQ** map in Among Us.

polus.txt This file is structured similarly to `travel-log.txt` from the midterm. At the top of the file there are two numbers, *N* and *K*, where *N* represents the number of unique vents, and *K* represents the number of `<src> <dst>` pairs. The difference between this file and the `travel-log.txt` file, is that these pairs are bidirectional. This file contains all possible pairs for the **Polus** map in Among Us.

Complete the following tasks:

1. Write a `makefile` that will compile an executable program named `venting`. To do this you will need to:
   (a) Create a target rule for `list.o`
   (b) Create a target rule for `list-adders.o`

  (c) Create a target rule for `venting.o`
  (d) Create an **all** rule for the target rule that creates the `venting` executable

2. In your `makefile` you should also create a **clean** rule that removes all intermediate build targets, similar to the example in the textbook.

Note: Be sure to make use of appropriate Makefile variables/macros so that the standard compiler options that we have been using in CMPT 214 since the beginning are used to build the program.

Sample Output

Here is what the output might look like. This demonstrates the form of the executable output only.

```
Which map are you using?  [Skeld, Polus, Mira]
Skeld (USER INPUT)
Loading vent connections for SKELD map...
Here is a list of all possible vents:
UpperEngine
UpperReactor
LowerReactor
LowerEngine
Security
Electrical
MedBay
Admin
Cafeteria
HallwayByShieldsAndNav
Shields
UpperNavigation
LowerNavigation
Weapons
Which vent are you looking to go to?
Electrical (USER INPUT)
You can get to Electrical from:
MedBay, Security,
```

Question 2 (4 points):

Git is a popular version control tool that you will likely use in future classes, in industry, and possibly for any side projects you might wish to pursue in the future. In first year you have used Git locally, however this differs greatly from remote projects, which is what the purpose of this question will expose you to.

Complete the following tasks:

1. Create a project on the department's Git Lab (`https://git.cs.usask.ca`). If you've done the git walkthrough from Topic 16 you should already know how to do this.
2. Clone said project onto your local machine.
3. Research yourself to find a `git` comment that will list the shortname of all remote servers you have configured, as well as the URL associated with the shortname.
4. Take a screenshot of the terminal output resulting from you cloning the remote repository, and the output for your git command to include in your submission.

Sample Output

Here is what the output might look like.

```
poa681@skorpio:~$ git clone https://git.cs.usask.ca/poa681/dummy.git
Cloning into 'dummy'...
Username for 'https://git.cs.usask.ca': poa681
Password for 'https://poa681@git.cs.usask.ca':
remote: Enumerating objects: 6601, done.
remote: Counting objects: 100% (6601/6601), done.
remote: Compressing objects: 100% (3200/3200), done.
remote: Total 6601 (delta 3394), reused 6598 (delta 3394), pack-reused 0
Receiving objects: 100% (6601/6601), 17.14 MiB | 37.58 MiB/s, done.
Resolving deltas: 100% (3394/3394), done.
poa681@skorpio:~$ cd dummy
poa681@skorpio:~/dummy$ git <your-command-from-task-3>
origin   https://git.cs.usask.ca/poa681/dummy.git (fetch)
origin   https://git.cs.usask.ca/poa681/dummy.git (push)
```

Question 3 (2 points):

On `tuxworld` all student home directories are subdirectories of `/student`.

Write a single BASH command to find all subdirectories of `/student` that have the read permission set for all users who are neither the file's owner nor member's of the file's group. Then output the permissions (first column), owner (third column), and group (fourth column) to a text file called `a6q3.txt` along with your command.

Note: This question must be done without the use of `sed(1)` or `awk(1)` (which we have not taught!). Furthermore, when trying to get the specific columns the output of your previous pipe will have inconsistent space delimiters. Therefore, multiple spaces will need to be replaced with one space before actually being able to select the columns.

When you think you have the command, copy and paste the it to a new file called `a6-bash-commands.txt`. IMPORTANT: Clearly identify that this command is for Question 3 as this file will also contain the command used in Question 4! If your marker can not easily make out what command is for which question you risk not getting a grade for these two questions. You will be submitting both `a6-bash-commands.txt` and `a6q3.txt` for this question submission.

Question 4 (2 points):

The input file `a6q4-intput.txt` is a mess. It has a mix of single words on some lines, and multiple words on others. Write a single BASH command that will read in the file, and create two new files. `a6q4-pt1.txt` with the entire contents of `a6q4-intput.txt` all on one line, and `a6q4-pt2.txt` that has each word from `a6q4-intput.txt` being on its own line.

When you think you have the command, copy and paste the it to a your file called `a6-bash-commands.txt`. IMPORTANT: Clearly identify that this command is for Question 4 as this file will also contain the command used in Question 3! If your marker can not easily make out what command is for which question you risk not getting a grade for these two questions. You will be submitting all `a6-bash-commands.txt`, `a6q4-pt1.txt`, and `a6q4-pt2.txt` for this question submission.

4 Files Provided

list.c C Source file for question 1 containing linked list implementation functions.

list-adders.c C Source file for question 1 containing linked list implementation functions.

vents.c C Source file for question 1 containing program implementation.

list.h Header file for question 1 for linked list.

vents.h Header file for question 1 for vents.

mira.txt Text file containing vent information for Mira HQ map for question 1.

polus.txt Text file containing vent information for Polus map for question 1.

skeld.txt Text file containing vent information for Skeld map for question 1.

a6q4-input.txt: An input file that will be used in question 4.

5 What to Hand In

Hand in a `.zip` file archive which contains the following files:

list.c C Source file for question 1 containing linked list implementation functions.

list-adders.c C Source file for question 1 containing linked list implementation functions.

vents.c C Source file for question 1 containing program implementation.

list.h Header file for question 1 for linked list.

vents.h Header file for question 1 for vents.

mira.txt Text file containing vent information for Mira HQ map for question 1.

polus.txt Text file containing vent information for Polus map for question 1.

skeld.txt Text file containing vent information for Skeld map for question 1.

makefile Your completed `makefile` for question 1.

a6q2.jpg A screenshot of you cloning your remote repository and output of the command used.

a6q3.txt .txt file created by your single BASH command in question 3.

a6q4-pt1.txt .txt file created by your single BASH command in question 4.

a6q4-pt2.txt .txt file created by your single BASH command in question 4.

a6-bash-commands.txt Both BASH commands used in questions 3 and 4, clearly identifiable.

6 Grading Rubric

The grading rubric can be found on Canvas.