Programming assignment 1. Creating small tools

Write programs that do one thing and do it well.
Peter H. Salus, A Quarter-Century of Unix (1994)

The main goal of this assignment is to familiarize yourselves with Markus submission system and svn, and to set up your programming environment for developing C programs.

So far you were writing small snippets of code on PCRS and learned about existing Linux utilities, but now you are ready to start writing your own software tools.

Part 1. Your SVN repository

It is important that you follow the instructions carefully, so that your work is correctly recorded. Remember, you must do all of the steps on a CDF machine from your own account. You can do it from home if you are logged into a CDF server (using NX, Putty, ssh).

1.1. Checkout your svn repository

Create a directory in your home directory called *csc209*. Change into that directory and check out your SVN repo using the repository link from MarkUs.

1.2. Copy input files

Input files for all tasks are in this tar archive: *A1_files.tar*. Copy this archive into your *A1* directory, and untar it. You should see the following input files: *testinput1.csv, testinput2.csv, testinput3.csv, starbucks.csv, tims.csv, map.html, campuscoffee.html* and 2 image files. Important note: since you do not know where these files came from (probably from the internet), make sure that you convert text files to Linux format by running `fromdos` or `dos2unix` before using them as an input to your programs.

1.3. Test svn

Create file *readme.txt* with a single line: "A1". Add this file to your *svn* repository using `svn add readme.txt`. Then commit it using `svn commit -m 'First commit'`. Once you have committed files to your repository, you can use MarkUs to make sure that what you committed is what we've got. The *Submissions* tab for the assignment will show you what is in the repository, and will let you know if you named the files correctly. You will not be able to submit files through the MarkUs web interface for the assignments.

If you need more information about svn, you can find it in this tutorial:
http://maverick.inria.fr/~Xavier.Decoret/resources/svn/

Your TA would be able to do these first steps together with you, until you understand how to submit your files by adding them to *svn* and committing your changes. Make this a habit to always test your code before the end of the work day to make sure that it compiles and runs, and commit your changes to *svn*.

Doing all these steps does not give you any marks, but if you do not submit your work into the correct repository on Markus, you will fail the entire assignment. Having all the correct code compiling and running on your local machine and even in the incorrect place on teaching server (cdf) will reduce all your hard work to the mark of zero.

Part 2. My first tool: csv2js

You need to design and implement a software tool which converts GPS location data from *csv* format into a valid JavaScript object.

The input to the program is a text stream which consists of lines, and each line contains 3 values separated by commas (csv format - comma-separated):

```
Latitude,Longitude,Label
43.665793,-79.380551,Place 1
43.665379,-79.380826,Place 2
43.664949,-79.380629,Place 3
```

Figure 1. Sample input format

The output is the same data represented as a JavaScript object (JSON format):

```
data=[
    {latitude: 43.665793, longitude: -79.380551, label: 'Place 1'},
    {latitude: 43.665379, longitude: -79.380826, label: 'Place 2'},
    {latitude: 43.664949, longitude: -79.380629, label: 'Place 3'}
];
```

Figure 2. Sample output format

Your program should read the lines from standard input, extract three values from each line using `fscanf` and write the same data in JSON format to standard output using `fprintf`.

You should write your program for Part 2 in a single text file called *csv2js.c*. Then you need to compile your program using `-Wall` flag and C99 standard into an executable called *csv2js*.

First, test your program by entering data from the default standard input - the keyboard, and printing the results to the default standard output - the screen.

If you wrote the program correctly, you should be able to set standard input to the input file *testinput1.csv* and redirect standard output to file *data.js*.

2.1. Test case 1

Testing that your program generates the correct output format.

Using *testinput1.csv* as an input, generate output file *data.js*, and put it in the same folder where *map.html* file is located. Then open *map.html* in a modern browser. You should be able to see all the map markers that correspond to locations specified in *testinput1.csv*.

2.2. Test case 2

Testing your program with a corrupt input.

The test file *testinput2.csv* contains some invalid GPS coordinates. Modify your program so it checks that the values of latitude and longitude are within valid intervals. Your program should send an error to the standard error stream, and ignore the line with invalid GPS coordinates.

Run your program using *testinput2.csv* as an input and redirect the result to *data.js*. Now test that you see the error messages on the screen, and you can open *map.html* and see all the valid locations on the map.

2.3. Test case 3

The third test file *testinput3.csv* contains a header. Without modifying your C program, create a one-line script *test3* which will produce a valid output from *testinput3.csv*. **Hint**: use `tail` utility with the corresponding options before redirecting the content of the input file to your program: see https://en.wikipedia.org/wiki/Tail_(Unix)

2.4. Test case 4

Write a short shell script which uses your *csv2js* tool and takes all 3 input files *testinput1.csv*, *testinput2.csv*, and *testinput3.csv*, and produces a single output file *data.js*. **Hint**: you can concatenate the contents of your files, remove headers where needed, and then pass the concatenated stream as an input to your tool.

Part 3. My second tool

Your friend created a map app which displays all coffee shops on campus. She found a list of all Starbucks locations in the world and the list of all Tim Hortons locations in the world (mainly in Canada). These files are provided in your *A1_files* directory. Both *starbucks.csv* and *tims.csv* contain headers. She asks if you could help her to convert the original csv format into a JavaScript object compatible with the Google map API.

Yes, of course, you have a tool for that!

However, your friend asks you that you limit the output to the locations delimited by the UofT downtown campus - roughly defined as a rectangle presented below (boundaries included):

Without changing your *csv2js* code, accomplish this task by writing an additional small tool *campus* which will process the data in two input csv files and retain only campus locations. Write your source code in file *campus.c*, and compile it as before with -Wall flag and using C99 standard into executable named *campus*.

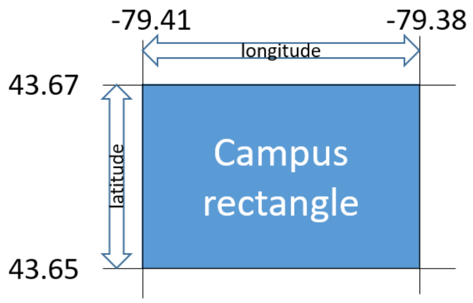

Figure 3. Definition of campus area

Now connect both tools using pipe to obtain a final output in file *data.js* Test your final results with *campuscoffee.html*.

What to submit:

Check into your svn repository 2 source code files: ***csv2js.c*** and ***campus.c***. Make sure that the files are named properly, all with lowercase letters. Submit your shell scripts for Part 2.3 and 2.4. in files ***test3*** and ***test4***. **DO NOT** submit inputs, outputs and html files.

Marking scheme:

10% - *csv2js.c* compiles without warnings

15% - test case 1 handled correctly

15% - test case 2 handled correctly

10% - script *test3* runs and produces the correct output

10% - script *test4* runs and produces the correct output

10% - *campus.c* compiles without warnings

10% - *campus* produces the correct output (in csv format)

20% - *campus* can be connected with *csv2js* using pipes and the final campus coffee map app works correctly

For a total of 5 points of the course grade.

Note: If your code does not compile on CDF, you will get a zero for this assignment.