McGill University
School of Computer Science
**COMP206 Software Systems**

# Assignment #3

Due: November 15, 2014 at 23:30 on My Courses

For EACH of the following short C programming problems (A) compile them using the GNU make utility (submit make with your submission) and (B) archive your work using Git (you are permitted to replace Git with CVS or SVN if you'd like to – you cannot use a web-based tool – please submit a zipped version of the repository with your submission).

QUESTION 1 – EXPERIMENTAL DATA

A scientist has data sets stored in a file that need to be analyzed. The text file has data stored in this manner: experiment-name, data1, data2, data3, etc. Each experiment is recorded as one row of the text file, i.e. terminating with a carriage return and line feed. Each data unit is separated from the next data unit with a comma. Every row of this text file is a different experiment. A file formatted this way is called a comma-separated-vector, or CSV file. This is a common format used to transport information from one software to another (e.g. Excel uses this).

An example text file:

Bob's experiment,10,23,11,15,13
Prof's trial,12,12,13,15,14,13
Mary's experiment,11,13,12,15,13,14

Notice there are no spaces between data units, only the comma separates things. Each row is an experiment. More formally, each row is a Record and each data unit is a Field. The above example shows three experiments of various lengths conducted by different individuals. Notice that the first field, the experiment's title field, has spaces separating the words in the title. It is the comma that keeps things working. Note further that the comma is considered a special symbol with a special meaning and therefore the user is not permitted to use commas in their CSV file because it would brake the definition.

Write a simple program that uses the command-line arguments as the means to receive the name of the text file the programs must process. The program will simply calculate the average for each experiment. The average will be outputted to the screen beside each experiment's title. Call you program ExperimentalAverage. For example:

./ExperimentalAverage data.txt
Bob's Experiment 14.40
Prof's Trial 13.17
Mary's Experiment 13.00

Make sure the output is to two decimal places rounded upwards.

**What to hand in & grading**:

- 7 points in total
- Submit the .c file
- Submit the .txt file
- Submit your make and repository file
- The TA will run this with both the file you provided and their own text file.
- 1 point of your grade will be devoted to well formatted code

QUESTION 2 – THE SYSTEM COMMAND (the my shell program)

It seams to be popular for programmers to create their own shells for Unix. There are so many of them. I'm asking you to create one more, your version. Since this is a small question just do the following:

- Create a C program that when started it displays a prompt of your choosing.
- Your shell only has two commands: SET PROMPT and QUIT.
  - SET PROMPT word
    - SET PROMPT are reserved words the user must type in, as is, but in any case (not case sensitive). "word" is any single word that will replace the prompt. In other words, after executing this command your shell's prompt changes to this word.
  - QUIT
    - QUIT is a reserved words the user must type in, as is, but in any case (not case sensitive). The command QUIT exists your program returning to the Unix command-line.
- Anything else the user types in that are not the above two commands are sent to the Unix command-line for processing using C's system() function.

Here is a sample session with you shell. Your shell program's name, for this assignment, is MyShell:

./MyShell <--- no command-line arguments

```
Super Shell!! ls
my.txt stuff.csv a.out
Super Shell!! set Prompt Wow??
Wow?? ls
my.txt stuff.csv a,out
Wow?? quit
```

Notice in the above execution example after typing the a.out program MyShell the user sees the shell's prompt, in this case the string "Super Shell!!". A space separates the prompt from what the user enters (I suggest gets or fgets). This input is compared with

the two reserved command SET PROMPT and QUIT. In the first use the user entered "ls". Since that does not match the two reserved commands the input is set, as is, to the C system() function to be processed by the Unix command-line. This way all Unix commands are now part of your shell too! This continues until the user entered "quit" to exit the program.

**What to hand in & grading**:

- 6 points in total
- Submit the .c file
- Submit the make and repository files
- 1 point of your grade will be devoted to well formatted code

QUESTION 3: LINKED LIST & PRE-PROCESSOR

Write a C program that asks the user for words. Each word is a string. The word is not a sentence. It is a single word without spaces and without carriage return. It terminates with the \0 as all strings normally do. Your program does not know how many strings the user will enter. Your program only knows that the user will end the input with the following string "***END***".

Create a linked list that stores these words in the same order they were entered in by the user. After the user enters the termination string "***END***" the programs prints out the words it has stored back to the screen.

This program prompts the user. A title announcing the program is displayed: Welcome to the infinite string storage program. At each input of a word the prompt prompts: Please input a single word. If the user inputs more than one word the program only reads the first word and ignores all the rest from the input.

Use the PRE-PROCESSOR to compile your program in English or some other language. The compiled program contains only the code from one of the languages. Use the #define LANGUAGE directive to switch between languages. For example:

| | |
|---|---|
| #define ENGLISH | to create the English compiled code |
| #define SWAHILI | to create the Swahili compiled code, or whatever language. |

**What to hand in & grading**:

- 7 points in total
- Hand in the .c file
- Submit the make and repository files
- The TA will compile your program into the two languages and enter short and long lists.
- 1 point of your grade will be devoted to well formatted code

## HOW IT WILL BE GRADED

- Your program must run in order for it to be graded. If your program does not run the TA does not need to go any further and may give you zero.
- Your program must run on the Trottier computers since this is a programming course for the Unix environment. The TA will not modify your text files in any way to get them to run under Trottier's operating system.
- All grades are awarded proportionally. In other words, if a question is graded out of 4 points and you got 75% of the program running then you will get 3 out of 4, etc.
- Grading portions of your code that do not run (assuming that your program runs at least minimally) will be graded proportionally as to how correct it is.
- Make sure your code is easy to read since this may impact the quality of your grade.