CSE 109: Systems Programming

Fall 2017

Program 1: **Due on Wednesday, September 13th at 9pm on CourseSite.**

Note: All file and folder names in this document are case-sensitive.

Collaboration Reminder:

1. You must submit your own work.
2. In particular, you may not:
   (a) Show your code to any of your classmates
   (b) Look at or copy anyone else's code
   (c) Copy material found on the internet
   (d) Work together on an assignment

Assignment:

1. Log into sunlab.cse.lehigh.edu
2. Create a folder named *Prog1* in your cse109.172 folder.

   This will contain all of your submission materials.
3. Use a text editor to create a C source code file named *prog1.c* in the *Prog1* folder.

   All relevant code for this assignment will end up in this source file.
4. Add the comment block shown at the end of this document to the top of the file named *prog1.c*.

   Make sure to transform all <item> notations into what is expected.

   Do not use your LIN, use the 6 character prefix of your Lehigh Email as your user id.
5. Create a main function and any addition functions necessary to do the following:

(a) The program will take all input from standard input, possibly transform it, and output it to standard output.

(b) The program will read in input line by line. Transformations, if any, will be done per line. Then print out the transformed line.

(c) Transformations are specified by the command line arguments provided to the executable. You will have to read those arguments and handle them.

  i. -*r*: Reverse each word on the line
  ii. -*e*: Reverse the entire line
  iii. -*t*: Change lowercase to uppercase and vice versa
  iv. -*s*: Swap alternating lines.

  Do not include *:* in your command line arguments.

(d) Multiple command line arguments may be provided, in that case, apply multiple transformations. You **must** ignore any duplicate command line arguments.

(e) If a command line argument is given that does not match one that we accept, print "Invalid command line option" to standard error and return 1 (this should result in exiting the program - it should not process any of the lines of text).

(f) If there is an odd number of lines, you won't be able to swap the last line. That's fine, just print it as with the usual transformations.

(g) Hint: You will have to read from the user until there is no more text left. Ctrl+D can be typed into the terminal to indicate there is no text left. You will have to figure out how to get your input handler to understand this. Consider looking at the man page for *scanf*.

(h) Hint: You can use the functions in *ctype.h* to help you with the transformations. Look at the man page for *isalpha* for more info.

6. Compile your program using the command:

   Make sure the command *gcc -v* specifies gcc-7.1.0

   gcc -g -Wall -o changetext prog1.c

7. Create *myfile*, a file that contains lots of text, perhaps the text from the Wartortle page on Bulbapedia.

8. Run the program as part using *myfile* as input (two different ways shown):

   ```
   cat myfile | ./changetext <options>
   ```

   ```
   ./changetext <options> < myfile
   ```

   Remember to replace <options> with whatever is appropriate for your test case.

9. Make sure to sufficiently test your code and that it makes sense and works for various combinations of the command line arguments.

10. Once ready to submit, you can package up the assignment as a .tgz file

    ```
    tar -czvf Prog1.tgz Prog1
    ```

    You must use this command in the directory that contains the *Prog1* folder, not within the directory.

11. Transfer *Prog1.tgz* to the Program 1 submission area of CourseSite.

Testing

1. Make sure your code does not leak memory by running it with Valgrind.

2. Some test cases (*.in files) will be provided in the directory:

   ```
   ~jloew/CSE109/prog1student/
   ```

3. A working executable will also be in that directory.

4. The working executable is built to the expected specifications (unless we screwed up). Any confusion in the requirements can potentially be tested by using the working executable.

5. You can generate output files from the working executable and use *diff* to compare them to the results that your code generates.

6. The test cases provided are sufficent to guarantee a 50 on the assignment. There may be other concerns to worry about so think about what to test.

7. Warnings constitute errors, they are there for a reason, fix them!

8. Valgrind can tell you what lines of code that some errors (the ones it detects) come from. You may find that useful as well.

Comment Block:

```
/*
    CSE 109: Fall 2017
    <Your Name>
    <Your user id (Email ID)>
    <Program Description>
    Program #1
*/
```