## 1. Instructions

You will be work with your group on this project and it will be submitted in two phases. Your whole group will only need to submit one completed project for each phase. This can be done from any group members Blackboard account. Your solution must be uploaded to Blackboard by the deadline, read section 4 to learn what to submit and by when.

## 2. Project Objectives

Phase 1:

The objective of this part is to introduce you to your group and acclimate you to the basics of team based software engineering. You will need to use Subversion or Git with multiple users so that you can become familiar with software versions in development. You will need to compile a program (provided for you) and develop a basic Make file for it. My hope is that you will do this work from the Linux terminal so that you become familiar with how to move around in the new environment.

Phase 2:

The objective of this part is to have you become familiar with developing a procedural program with a team of engineers using the C programming language. There are many elements of C that lend themselves to well-engineered C programs. These include but are not limited to: Functions, Global Constants, Pointers, and Constant parameters and functions. Along with using the C language, my hope is that you become familiar with the importance of good design and testing your code.

## 3. Project Parts

This project will be composed of two parts which you will turn in over the course of three weeks.

Phase 1: First answer the following questions

(1) Get to know your team members. Send them your email, or meet them in class or even outside of class and discuss this project and who will take on what role in the team. Everyone gets points for this section, as long as the group submits the assignment.

(2) Create a Github.com or Google code hosted account if you have not already done that and invite all the team members to join that account (everyone will need an account to join).

(3) Find the C source code for a hello world program and copy the text to a .c source file. A link to one is http://groups.engin.umd.umich.edu/CIS/course.des/cis400/c/hworld.html.

(4) Write a Make file that compiles this program file into a program called “hello”. The Linux lab manual has some information on this.

(5) Setup the code repository account with the source file and the make file, then have every team member check out the project and change or add a source code comment in either file. Each team member must then commit their changes back to the repository. Take a screenshot of the source changes page of the repository website showing how each team member has committed to the source. Provide an explanation below the screenshot identifying which team member relates to which id in the repository.

(6) Use make to compile your program and paste the terminal output into your solution document. If you redirect stdout to a file this process may be easier.

(7) Use GDB to set a breakpoint at the main function, then step through your program. It should only have one line in the code so this shouldn’t take long. Paste the terminal output to your solution document. It probably won’t be possible to redirect stdout for this process, since you will need to see what GDB is doing on the screen.

(8) Run your application and paste the terminal output into your solution document.

Now you will work on the design component, where you will need to create a pseudocode design for each function of the program outlined in the problem statement. You will also need to create a test design document outlining the tests you will run for each functional test. This should be put into a spreadsheet where each row is the test you will run and the columns are as follows: for each parameter you will need one column for each partition class, the final column is your expected output.

Put your solution document (answers to the questions), your source code file (hello world source), your make file, the compiled program, all of your pdf design documents (which includes your planned tests) into a tarball archive file.

Phase 2:

The second component will include the coded implementation of your design. You will need to have code for each test case, as well as, code for the program as well. You must create comments for every function in the program (it’s not a bad idea to comment every line of code as well, don’t forget team work). The code must compile for anything more than 50% credit. Finally, your source code must be split into two files, one where all the print functions exist, and the second where you have all other functions. There must be a single header file for functions prototypes and global constants. You must have a make file for your implementation, which compiles your code into an executable called “lister”.

You will be graded on the correctness of your implementation, but you can gain some of your points back by providing a bug report. This is a report of each bug that is contained in your code and what is the root cause of the bug. This report will then be graded on correctness as well. Even if your code doesn't work correctly it must compile for anything more than 50% credit. You must have a make file for your implementation, which compiles your code into an executable called "lister".

**4. Problem Statement**

You will be reading source code from a file and printing it to the screen. You will write a source code lister utility program that requires the name of the source file be in the command line when it is run. In the main routine you fetch this name as argv[1] and pass it to function called init_lister. There, you copy the name into variable called "source_name", open the source file parameter "source_file", and set the date and time string.

In the main routine's while loop, you should call boolean function "get_source_line" repeatedly to read and print source lines. That function returns TRUE if it successfully reads and prints a line. If it reached the end of the source file instead it returns FALSE.

In function "get_source_line", we call fgets to fill the variable "source_buffer" with the contents of the next source line. Variable "line_number" keeps track of the line number. You call sprintf to print the source line, along with its line number, into the variable "print_buffer". You then ship "print_buffer" off to function "print_line" to print it.

In function "print_line", variable "line_count" keeps track of how many lines have been printed on the current page. If the number of lines exceeds "MAX_LINES_PER_PAGE", you call function "print_page_header" to skip to the top of the next page and print a page header. In any case, you make sure that the current line will fit on one printed line. If the line is too long, you truncate it before printing it, and then restore it to its original length.

Note the difference between variables "line_number" and "line_count". You initialize "line_number" to zero and increment it by one for each source line. Its final value is the total number of source lines. "line_count" counts the number of lines on the current page. You reset it to one for each new page. Initializing it to "MAX_LINES_PER_PAGE" is a clever trick to force the very first source line to trigger a new page and page header.

**5 What to Submit for Grading and Assignment Deadline**

The first part will be due Thursday February 13th at **11:59:59PM.** You should include your design document(s) and a screen shot of your Repository hosted site showing how all the team members contributed to the design document. A portion of your grade will be determined by everyone contributing something to the repository. The documents must be in pdf format and submitted as one tar file. Please see project one for more information on this.

The second part will be due Sunday February 23rd at **11:59:59PM.** You should include your commented source code and test code files and a screen shot of your Google Code site in a separate pdf. Please put all files together as a tar file.

You will need to include a URL of your repository for each phase of the project, and a list of each team members name and their corresponding repository ID. I will be going to your site and checking the actual contribution of each team member. If I see everyone is contributing then all team members will receive equal points. If I see some team members have contributed significantly less than the others I will further investigate and deduct a percentage of that team members points based on their contribution. If you are worried that one team member didn't contribute much to the repository, but they were instrumental in the project (for example someone acted as a team leader), then you can comment on this when you submit the assignment.

Please put your group members names at the top of every document you submit including code (this must be a comment). The file name of your tarball should be cse220_lab2_phaseX.tar, where X stand for the project phase (1, or 2). Please see lab one for more information on the penalty for not submitting a pdf file for documents that must be in pdf format.