CSI 402 – Systems Programming

Programming Assignment 1

Date given: Sept. 28, 2016 **Due date:** Oct. 10, 2016
Total grade for this assignment: 100 points **Weightage:** 3%
Note: Programs that produce compiler/linker errors or result in segmentation faults will receive a grade of zero.

A. Purpose. Often, programs produce "log files", text files containing information on the operations they perform. Each line contains several fields of information, separated by commas. One of the fields is the timestamp of the information logged, i.e., the date and time at which a specific operation occurred. In the case where several processes are executed on a machine, each process writes its own log file. For example, a Web server produces a log file that contains one line each time a client requests a URL. The contents of a Web server log file would look like this:

```
DEBUG,2011-10-21 14:32:00,server starting
DEBUG,2011-10-21 14:32:01,server started
INFO,2011-10-21 14:32:10,client 132.128.1.1 requested url " spl121"
INFO,2011-10-21 14:32:15,client 123.111.10.1 requested url " spl111"
INFO,2011-10-21 14:32:18,sending content of url " spl111" to 123.111.10.1 (1230 bytes)
INFO,2011-10-21 14:32:18,sending content of url " spl111" to 132.128.1.1 (2240 bytes)
INFO,2011-10-21 14:32:19,sent content of url " spl121" to 132.128.1.1
ERROR,2011-10-21 14:32:20,error sending to 123.111.10.1
```

Your task in this assignment is to implement a program, named `mylogger`, that will merge several log files in order of timestamp. This is useful because programmers (or system administrators) can then reconstruct a single chronological order of events that occurred across several processes on a server (or in a distributed system, on several machines). The merged log file is particularly useful to understand if one event in one program caused another event in another program.

B. Description. The executable of your program must be named `mylogger` (your `makefile` must ensure this) and must support the following usage: `mylogger [logfilesollectionpath]`. The **optional** command line argument `logfilesollectionpath` is the path to a directory containing a collection of log files that your `mylogger` program should operate on. If no argument is given, your `mylogger` program should operate on the current working directory. The output of your program should be named `combinedlogs.log`. If a `combinedlogs.log` file already exists in the `logfilesollectionpath` directory, your program should overwrite the contents of that file.

In order to operate on several log files, your program must first concatenate them into a single file, and then read the result of this concatenation. You can assume that the format of all log files to be provided to your program is the same, i.e., each log starts with a line `#<name of program>` so that you know that a new log file starts when reading the concatenated file, followed by log

entries. Each entry will follows the format `<level>,<date>,<message>`. You can also assume that the timestamp format is `YYYY-MM-DD HH:MM:SS`. Note that this format is easily comparable.

For your program to efficiently sort the contents of the log files once they have been merged, you should implement a **linked list** data structure. After sorting, the contents of the list should be printed into the standard output in a single pass.

C. Example. Let the following be the contents of `server.log` and `client.log` respectively:

```
#SERVER
DEBUG,2011-10-21 14:32:00,server starting
DEBUG,2011-10-21 14:32:01,server started
WARN,2011-10-21 14:32:10,client connected
DEBUG,2011-10-21 14:32:15,received string
INFO,2011-10-21 14:32:18,sent result
ERROR,2011-10-21 14:32:19,error sending
DEBUG,2011-10-21 14:32:22,client disconnected
```

```
#CLIENT
WARN,2011-10-21 14:32:09,session started
INFO,2011-10-21 14:32:14,calling server
ERROR,2011-10-21 14:32:20,error while calling
DEBUG,2011-10-21 14:32:21,disconnecting
```

The expected output when running `mylogger` in the directory containing these two logs should be:

```
DEBUG,2011-10-21 14:32:00,server starting
DEBUG,2011-10-21 14:32:01,server started
WARN,2011-10-21 14:32:09,session started
WARN,2011-10-21 14:32:10,client connected
INFO,2011-10-21 14:32:14,calling server
DEBUG,2011-10-21 14:32:15,received string
INFO,2011-10-21 14:32:18,sent result
ERROR,2011-10-21 14:32:19,error sending
ERROR,2011-10-21 14:32:20,error while calling
DEBUG,2011-10-21 14:32:21,disconnecting
DEBUG,2011-10-21 14:32:22,client disconnected
```

Note that the lines containing `#SERVER` and `#CLIENT` are discarded.

D. Error Handling. Your program must detect the following fatal errors. In each case, your program should produce a suitable error message to `stderr` and stop.

- The number of command line arguments is more than one.

- A `logfilescollectionpath` directory doesn't exist.
- Directory `logfilescollectionpath` cannot be accessed.
- The contents of directory `logfilescollectionpath` cannot be accessed.
- Log file does not begin with `#`.
- Log entry doesn't contain at least two commas.
- Log entry doesn't "look" like `<level>,<timestamp>,<message>`.

C. Structural Requirements. Your linked list must have the following structure:

```
struct logline{
    char level[20];
    char timestamp[20];
    char message[100];
} logline_t;

typedef struct loglist {
    logline_t line;
    struct loglist *next;
} loglist_t;
```

Additionally, your submission must *at least* contain the following files:

1. A C source file with just the `main` function.
2. A C source file containing only the following function(s):
   - Function `logline* parseLine(string line)` to construct a `logline` with 3 fields based on the input string. Note that this function allocates memory.
   - Function `void deleteList(loglist* l)` to free all memory allocated for the list.
   - Function `void printLines(LineList* l)` to print all the lines contained in the list.
3. A header file containing only the prototypes for the functions in the second item.
4. A C source file containing only the following function(s):
   - Function `loglist* mergeLists(loglist* resultlist, loglist* inlist)` to merge sorted linked list `inlist` into `resultlist` and return a pointer to the head of the combined list.
   - Function `loglist* sortList(loglist* inlist)` to sort the provided linked list `inlist` and return a pointer to the head of the sorted list.
5. A header file containing only the prototypes for the functions in the fourth item.

C. Submission Instructions. Make sure you are logged in to ITSUnix, and your present working directory contains only the files you wish to submit. Use the `turnin-csi402` command to submit all of the .c and .h files, and any file named 'makefile'. Make sure there are no extra .c or .h files in the current directory. If your makefile is called 'Makefile' (with a capital M), then make sure you use a capital M in the turnin command. Instructions for using the `turnin-csi402` command have been provided in Programming Assignment 0, which you should have already submitted.