Assignment 3

Due date: Friday, March 25, 11:59 pm
Submission via Git only

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines (i.e., Senjhalla or its analogous environment for students with M1 MacBooks) you installed/configured as part of Assignment 0 as this is our "reference platform". This same environment will be used by the teaching team when grading the work submitted by the SENG 265 students.

All test files and sample code for this assignment are available on the SENG server in **/seng265work/2022-spring/a3/** and you must use the scp command to get a copy of the files. For example:

```
scp NETLINKID@seng265.seng.uvic.ca:/seng265work/2022-spring/a3/* .
```

Any programming done outside of Senjhalla might result in lost marks or even 0 marks for the assignment. Make sure that your submitted files are inside of your **a3 folder of your cloned Git repository**. Hint: To verify whether you uploaded the files properly, please clone the git repository to a new directory on your computer and check that the files you intend to submit are there.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. However, sharing of code fragments is strictly forbidden. Code-similarity analysis tools are used to check submitted work for plagiarism.

Learning objectives

I. Revisit the C programming language, this time using memory allocation and dynamic data structures.
II. The starting point for A3 is a complete C program. You need to study and understand this program to complete the assignment. Thus, one of the learning objectives is to learn how to read and understand existing code and build upon it.
III. Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits. Update your git repository after every major editing session to make sure that you don't loose your work.
IV. Test your code against the provided test cases.

This assignment: process_cal3.c, Using C's heap memory

You are to write an implementation of process_cal in C such that:

a) To store event info, you need to allocate storage on the heap dynamically. Dynamic data structures must be used in the form linked-list manipulation routines (i.e., arrays of events are not permitted for this assignment).
b) The program itself consists of several C source files and a makefile for build management:
   - emalloc[.c or .h]: Code for safe calls to malloc, as is described in labs\lectures, is available here.
   - ics.h: Type definition for events.

- Linky[.c or .h]: Type definitions, prototypes, and codes for the singly-linked list implementation described in labs/lectures. To tailor the sample code to your solution, you need to modify some routines. However, you are fully responsible for any segmentation faults that occur as the result of this code's operation. The entire code has to be correct.
- makefile: This automates many of the steps required to build the process_cal3 executable, regardless of what files (.c or .h) are modified. This file can be invoked using the Bash command make.
- process_cal3.c: The main file of the program.

Relevant observations

- You must not use program-scope or file-scope variables.
- You must not use arrays of type struct event_t or event_t. Use linked lists instead. The elements of the linked list are struct types.
- You are free to use regular expressions in your solution but are not required to do so.
- Some of the limits from previous assignments that were placed on certain values are no longer needed (e.g., maximum number of events).
- Refer to the instructions provided in "README.md" for further and more technical instructions for this assignment.

Testing your solution

- Refer to the example commands in the file "TESTS.md" for appropriate command line input. Your solution must accommodate all specified command line inputs.
- Make sure to use the test files provided as part of this assignment.
- Use the test files and listed test cases to guide your implementation effort. Develop your program incrementally.
- For this assignment you can assume all test inputs are well-formed.
- Do not rely on visual inspection. Human eyes are poor at comparing white space. You can use the provided tester file (i.e., "tester.py") so you can verify the validity of your outputs in a simpler manner.

What to submit

- The seven files listed earlier in this assignment description (i.e., process_cal3.c, emalloc.c, emalloc.h, ics.h, listy.c, listy.h, makefile), submitted to the a3 folder of your Git repository.
- Following the documentation provided in the code and your own analysis of the program, you will need to complete **ONLY** the functions described below. However, you can create additional/auxiliary functions if needed.

File	**Function to Complete**
listy.c	new_node()
listy.c	add_inorder()
process_cal3.c	print_formatted_date()
process_cal3.c	print_formatted_event()
process_cal3.c	new_event()

Grading

Assignment 3 grading scheme is as follows. In general, straying from the assignment requirements will result in zero marks due to automated grading.

A grade: A submission completing the requirements of the assignment and is well-structured and clearly written. Global variables are not used. process_cal runs without any problems; that is, all tests pass and therefore no extraneous output is produced. An A-grade submission is one that passes all the tests and does not have any quality issues. Outstanding solutions get an A+ (90-100 marks), solutions that are not considered outstanding by the evaluator will get an A (85-89 marks) and a solution with minor issues will be given an A- (80-84 marks).

B grade: A submission completing the requirements of the assignment. process_cal runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written. Although all the tests pass, the solution includes significant quality issues. Depending on the number of qualitative issues, the marker may give a B+ (77-79 marks), B (73-76 marks) or a B- (70-72 marks) grade.

A submission with any one of the following cannot get a grade higher than B:
- Submission runs with warnings
- Submission has 1 or 2 large functions
- Program or file-scope variables are used

A submission with more than one of the following cannot be given a grade of higher than B-:
- Submission runs with warnings
- Submission has 1 or 2 large functions.
- Program or file-scope variables
- No documentation is provided

C grade: A submission completing most of the requirements of the assignment. process_cal runs with some problems. This is a submission that presents a proper effort but fails some tests. Depending on the number of tests passed, which tests pass and a qualitative assessment, a grade of C (60-64 marks) or C+ (65-69 marks) is given.

D grade: A serious attempt at completing requirements for the assignment (50-59 marks). process_cal runs with quite a few problems. This is a submission that passes only a few of the trivial tests.

F grade: Either no submission given, or submission represents little work or none of the tests pass (0-49 marks). No submission, 0 marks. Submissions that do not run, 0 marks. Submissions that fail all tests and show a poor to no effort (as assessed by the marker) are given 0 marks. Submissions that fail all tests, but represent a sincere effort (as assessed by the marker) may be given a few marks

Additional Criteria for Qualitative Assessment

- **Documentation and commenting:** the purpose of documentation and commenting is to write information so that anyone other than yourself (with knowledge of coding) can review your program and quickly understand how it works. In terms of marking, documentation is not a large mark, but it will be part of the quality assessment.
- **Proper naming conventions:** You must use proper names for functions and variables. Using random or single character variables is considered improper coding and significantly reduces code readability. Single character variables as loop variables is fine.

- **Debugging/Comment artifacts:** You must submit a clean file with no residual commented lines of code or unintended text.
- **Quality of solution:** marker will access the submission for logical and functional quality of the solution. Some examples that would result in a reduction of marks: solutions that read the input files several times, solutions that represent the data in inappropriate data structures, solutions which scale unreasonably with the size of the input.

Input specification

- All input is from ASCII-data test files.
- Data lines for an “event” begin with a line “BEGIN:VEVENT” and end with a line “END:VEVENT”.
- Starting time: An event’s starting date and time is contained on a line of the format “DTSTART:<icalendardate>” where the characters following the colon comprise the date/time in icalendar format.
- Ending time: An event’s ending date and time is contained on a line of the format “DTEND:<icalendardate>” where the characters following the colon comprise the date/time in icalendar format.
- Event location: An event’s location is contained on a line of the format
- “LOCATION:<string>” where the characters following the colon comprise the string describing the event location. These strings will never contain the “:” character.
- Event description: An event’s description is contained on a line of the format “SUMMARY:<string>” where the characters following the colon comprise the string describing the event’s nature. These strings will never contain the “:” character.
- Repeat specification: If an event repeats, this will be indicated by a line of the format “RRULE:FREQ=<frequency>;UNTIL=<icalendardate>”. The only frequencies you must account for are weekly frequencies. The date indicated by UNTIL is the last date on which the event will occur (i.e., is inclusive). Note that this line contains a colon (“:”) and semicolon (“;”) and equal signs (“=”).
- Events within the input stream are not necessarily in chronological order.
- Events may overlap in time.
- No event will ever cross a day boundary.
- All times are local time (i.e., no timezones will appear in a date/time string). This semester we will ignore the effect of switching to daylight savings.

Output specification

- The output required for process_cal3 will be the same used in Assignment 1. Please refer to the A1’s write up for further information.