Software Engineering 265
Software Development Methods
Spring 2021

Assignment 3

Due: Wednesday, March 24, 11:55 pm by submission via `git`
(no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines you installed as part of Assignment #0 (which I have taken to calling `Senjhalla`). This is our "reference platform". This same environment will also be used by the course instructor and the rest of the teaching team when evaluating submitted work from students.

All test files for this assignment re available on the UVic Unix server in `/home/zastre/seng265/assign3` (which consist of tests from the first assignment plus some additional, new tests). The starter files for the assignment can be found in this directory; more details of these files are provided on the next page.

Any programming done outside of `Senjhalla` might not work during evaluation.

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 without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools are used to examine submitted work.)

Objectives of this assignment

- Revisit the C programming language, this time using dynamic memory.
- Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits.
- Test your code against the 15 provided test cases (ten of which come from A#1/A#2).
- Use `valgrind` to determine how effective your solution is in its management of dynamic memory.

`icsout3.c:` Using C's heap memory

You are to write an implementation again in C such that:

- **only** dynamic memory is used to store event info, and

- **only** linked-list routines are used (i.e. arrays of events *are not* permitted).

In addition to these requirements, the program itself now consists of several files, some of which are C source code, one of which is for build management:

- `icsout3.c`: The majority of your solution will most likely appear in this file. Some demo code (protected with an `#ifdef DEBUG` conditional-compilation directive) shows how a simple list with two nodes of events can be allocated, assigned values, printed, and deallocated.

- `emalloc.[ch]`: Code for safe calls to malloc, as is described in lectures, is available here.

- `ics.h`: Type definition for events.

- `linky.[ch]`: Type definitions, prototypes, and codes for the singly-linked list implementation described in lectures. You are permitted to modify these routines or add to these routines in order to suit your solution. Regardless of whether or not you do so, however, you are fully responsible for any segmentation faults that occur as the result of this code's operation.

- `makefile`: This automates many of the steps required to build the `icsout3` executable, regardless of what files (`.c` or `.h`) are modified. The Unix `make` utility will be described in lectures.

You must ensure all of these files are in the `assign3/` directory of your repo, and must also ensure that all of these files are added, committed, and pushed. Do not add any extra files.

A call to `icsout3` will use identical arguments to that from the previous assignment. For example:

```
./icsout3 --start=2021/2/14 --end=2021/2/14 --file=one.ics
```

A few more observations:

- All allocated heap memory is automatically returned to the operating system upon the termination of a Unix process or program (such as `icsout3`). This is true regardless of whether the programmer uses `free()` to deallocate memory in the program or not. However, it is always a good practice to

deallocate memory via `free()` – that is, one never knows when their code may be re-used in the future, and having to rewrite existing code to properly deal with memory deallocation can be difficult. A program where all memory is properly deallocated by the programmer will produce a report from `valgrind` stating that all heap blocks were free and that the heap memory in use at exit is "0 bytes in 0 blocks". `valgrind` will be discussed during labs.

- You must **not use program-scope or file-scope variables**. You must **not use arrays of type `struct event_t` or `event_t`.**[1]

- You must **make good use of functional decomposition**. Phrased another way, your submitted work **must not** contain one or two giant functions where all of your program logic is concentrated.

- 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, maximum line length, etc. *Also note that some events have an ending time that is on the day following the starting time.*

What you must submit

- The seven files listed earlier in this assignment description (`icsout3.c`, `emalloc.c`, `emalloc.h`, `ics.h`, `listy.c`, `listy.h`, , `makefile`).

---

[1] This also means you may not have an array or arrays of `node_t` where each array element is simply a linked list that is one node in length.

Evaluation

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment which is well-structured and very clearly written. All tests pass and therefore no extraneous output is produced. `valgrind` produces a report stating that no heap blocks or heap memory is in use at the termination of `icsout3`.

- "B" grade: A submission completing the requirements of the assignment. `icsout3` can be used without any problems; that is, all tests pass and therefore no extraneous output is produced. `valgrind` states that some heap memory is still in use.

- "C" grade: A submission completing most of the requirements of the assignment. `icsout3` runs with some problems.

- "D" grade: A serious attempt at completing requirements for the assignment. `icsout3` runs with quite a few problems; some non-trivial tests pass.

- "F" grade: Either no submission given, or submission represents very little work, or no tests pass.