COMP 2160 Programming Practices
Assignment 3

Due Date: Thursday, November 15th at 11:59 pm

Objectives

- Separate compilation, makefiles
- Abstract Data Types
- Testing

Question 1

Take the sample solution from Assignment 2 and factor the linked list code into a **Table** (implemented internally using a Linked List). To do this, create `Table.c` and `Table.h` files, with the solution from Assignment 2 providing the implementation of `insert()`, `firstItem()`, `nextItem()`, `search()`, `clearTable()` and `validateTable()` (called `clearList()` and `validateList()` in the sample solution). You are also required to add a `delete()` function (which deletes the first occurrence of an item).

To support the use of multiple Tables within an application, your Table module must also include the ability to manage multiple tables. Create a struct to contain the 'instance' variables that describe a Table. Add routines `createTable()` and `destroyTable()` (which uses `clearTable()`...) and modify all other routines so that they take a table pointer as their first parameter (they need to know which table to work with -- note that this is how the 'this' pointer is implemented in OO languages such as Java and C++). Note that anything you add (including, but not limited to, the new routines) **must** adhere to the principles of Design by Contract, as per the requirements in Assignment 2.

Finally, implement a `main.c` program that runs a suite of tests (unit tests) to ensure that the Table correctly handles insertions, removals, searches and traversals. Your tests should focus on the operations on a table, not the manipulation of multiple tables. Be sure to factor your tests into logical units that are implemented as routines. You must include textual output indicating each test performed (e.g. "Searching for elements not in the table:") and whether or not they passed (e.g. "george found" or "fred not found"). Don't forget to run tests on an empty table. Upon completion, include a summary of the number of tests run, the total number that passed and the total number that failed (note that with assertions off you should always run to completion and get this summary).

Requirements

- You must include a Makefile for compiling your complete program.
- Make sure you test with assertions turned off. Boundary conditions should not cause the program to crash.
- You cannot change the internal behaviour of the Table (as defined by the sample solution from Assignment 2).
- You cannot use your own solution to Assignment 2. Learning to work with other people's code is very important.

Question 2

You have been provided with a Set ADT developed by a third party. This consists of a header file, set.h, defining the interface and the object file, compiled for use in our Mac lab (rodents). You must implement a complete unit test suite to validate all of the functionality defined in the header file.

Requirements/Notes

1. Hand in your main.c file (containing your test suite implementation) and a Makefile to compile your code and link it with set.o.
2. You will be provided with, at least, two different object file implementations of Set. Include a brief analysis (in your README file) for each object file, indicating whether or not they passed your tests. If an object file didn't pass your tests include a description of each failure condition that must be addressed.
3. If, at any time, the code crashes then the problem is with your code. An object file *may* provide you with erroneous results but will never cause your program to fail outright. That is, how you're testing the object file is what is causing the crash and you need to adjust your tests accordingly.
4. The test framework from question 1 should form a good foundation upon which to build the test framework for this question.
5. Bonus marks will be awarded for a *good* test framework making use of files to define the test suite. If attempted, include a note in your README file (along with how to run your tests) and include any test files required to perform your tests.
6. Note that .o files are not platform independent. The .o files we provided only work on the Mac lab. So you should only work on the Mac lab for this question.

Hand-in Instructions

Create a directory called `comp2160-a3-<yourlastname>-<yourstudentid>`. Within the directory, create two sub-directories called `Q1` and `Q2`. Place your `.c` files, `.h` files, `makefile, and README (if available)` for Question 1 inside the Q1 directory. Place your `.c` files, `makefile, and README` for Question 2 inside the Q2 directory.

Then run the command:

```
handin 2160 a3 comp2160-a3-<yourlastname>-<yourstudentid>
```

- You may *optionally* include a README file in your directory that explains anything unusual about compiling or running your programs.
- You may resubmit your assignment as many times as you want, but **only the latest** submission will be kept.
- We only accept homework submissions via handin. Please **do not** e-mail your homework to the instructor or TAs – it will be ignored.
- You **must** submit an Honesty Declaration (digitally in UMLearn). Assignments submitted without a corresponding Honesty Declaration **will not be graded**.