TCSS 422 — Computer Operating Systems
Spring 2015 — Homework Assignment 1

Due Date: Tuesday, Apr. 14

Guidelines

Homework should be electronically submitted to the instructor by the end of the day on the due date. A submission link is provided on the course Canvas page for this assignment.

Assignment Description

This assignment is intended to help you practice using the C programming language, particularly those aspects that are different from Java. You will implement five independent functions, each spanning different aspects of C. The intellectual complexity of the functions is at an introductory programming level, i.e., TCSS 142 or 143, so that you can focus on expressing the necessary computation in C rather than determining what computation is needed. It's recommended that you use some C reference material, such as the C Tutorial posted on Canvas, to help you grasp the syntactic and semantic differences between C and Java. Starter code for this assignment is available on the course Canvas page for this assignment.

Implementation Specifications

There are seven provided files for this assignment:

- `main.c` – The implementation file containing the program entry point, `main`, and some basic driver code.
- `functions.h`, `functions.c` – A header file declaring the five functions you're to implement and an implementation file containing skeletons for the five functions.
- `books.txt` – A data file used by the third function you're to implement.
- `lexicon.h`, `lexicon.c`, `words.txt` – Additional source code and a data file used by the fifth function you're to implement.

Inside `functions.c` you'll find skeletons for the five functions you're to implement. The five functions are intended to be completed in order, but since there are no computational dependencies among the functions they may be completed in any order. You can add helper methods, new types, etc., but do not modify the interface to these five methods. You shouldn't need to modify any provided file other than `functions.c` (and possibly `functions.h`), the other provided source files are already complete.

```
void multiply(int num1, int denom1, int num2, int denom2,
              int * rNum, int * rDenom)
```

This function should multiply two fractions together, producing a product in lowest terms. The first two parameters, `num1` and `denom1`, represent the numerator and denominator of the first fraction, respectively. Similarly, `num2` and `denom2` represent the second fraction. The product's numerator should be stored at the address specified in `rNum` (short for resulting numerator) and the product's

denominator should be stored at the address in `rDenom`. Remember, the product's numerator and denominator should be in lowest terms. If the product is positive, both its numerator and denominator should be positive. If the product is negative, only its numerator should be negative.

`char * rotate(const char * str, int amount)`

This function should copy and rotate the string `str`, returning the rotated copy. The memory for the copy should be allocated dynamically and the original string should not be modified. A string rotation shifts the characters in the string left or right. If `amount` is positive, each character should be moved `amount` places to the right. If `amount` is negative, each character should be moved `amount` places to the left. Characters that "fall off" the left or right sides of the string should "wrap around" to the other side. No characters should be lost, and the length of the rotated string is the same as the original string. The magnitude of `amount` may be equal to or greater than the length of the string, the rotation continues to wrap around in that case. For example, the string `"Hello"` rotated by 2, 7, or 12 would all produce the string `"loHel"`. Lastly, this function should run in $O(n)$ time, where n is the length of the string.

`int readAndDisplayBookInformation(const char * path)`

This function should read in and display book information from a file located at `path`. The file is in a comma separated value (CSV) format with each line containing information about one book (title, author, and publication year). The information for each book in the file should be displayed in a very specific format. For example, the first line in the provided `books.txt` file is,

```
A Tale of Two Cities,Charles Dickens,1859
```

Which should be displayed as,

```
"A Tale of Two Cities" by Charles Dickens (1859)
```

All displayed lines should follow the above format. You may assume that no line in the file is longer than 80 characters. This function should return a non-zero value if there was a problem reading the file, or zero if all file I/O completed successfully.

`void initializeAndShuffleDeck(struct Card deck[52])`

This function should initialize and shuffle, i.e., randomize, an array of structures that represent a deck of 52 standard playing cards. Each card has a rank and a suit, each represented by a `char` value. Ranks use `char` values `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`, `'0'`, `'J'`, `'Q'`, `'K'`, `'A'`, and suits use `char` values `'c'`, `'d'`, `'h'`, `'s'`. Each card is a unique combination of a rank and suit. When this function is invoked the array has not been initialized, but when this function finishes the array should contain unique cards in some random order. The provided driver function for this function merely displays the contents of the array, which will likely be different every time you execute the program. For example,

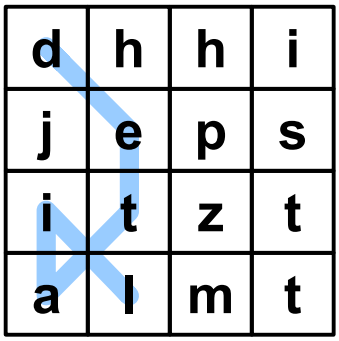
```
2c 6c Ks 4h 5s 4c Ah 3h Qh 9c 7d 3s Qc
Jd 4d 9d 0d 8d Kh Kd 2h 3d 9s 8h 6h Jh
Jc 4s 7h 5h 5d 7c 8c 0h 7s 8s Qd 0c 5c
Ac Qs 6s 9h 3c 0s Js Kc 2d 2s 6d As Ad
```

Lastly, this function should run in *O*(1) time.

`struct ListNode * findWords(const char board[4][4])`

This function should find all the words that can be formed within a 4x4 grid of letters (like in the game of Boggle). The list of found words are returned in a linked list of strings. Both the linked list nodes and the strings should be allocated dynamically.

Words can be formed starting at any of the 16 letters in the grid and following a path to adjacent letters, horizontally, vertically, diagonally. Each position in the grid can only be part of the path once. For example, in the grid shown to the right the word “detail” can be formed following the path shown in blue.



In order to identify words in the grid you'll need a collection of all possible words to compare against. This collection is already implemented and available via the functions in `lexicon.h`. The `isWord` function indicates whether a string is a legal word and the `isPrefix` function indicates whether a string is a proper prefix of some word in the lexicon. You may assume that the lexicon has been loaded and is ready for use within the `findWords` function, you shouldn't invoke the `loadLexicon` or `destroyLexicon` functions. Based on the words in the provided lexicon, the example grid above contains the words: alit, dept, detail, heil, hips, hist, ital, jilt, jilted, lait, late, lated, lied, lite, petal, pish, pished, shed, sheila, ship, shipt, sped, tail, tied, zeta.

The words in the returned linked list may be in any order, and words that can be formed multiple ways in the grid, e.g., dept, can be present multiple times in the list.

Deliverables

The following items should be submitted to the instructor. Failure to correctly submit assignment files will result in a 10% grade penalty.

1) The completed `functions.c` source file.
2) Any additional new or modified source files, e.g., `functions.h`, needed by `functions.c`.

Do not submit unmodified provided files, e.g., `lexicon.h` and `lexicon.c`. Do not include any extraneous files, such as Eclipse IDE files, object files, or subversion files.