Using C's heap memory for creating the concordance: `concord3.c`

In assignment 2 you used Python to implement an extended version of the concordance-generation scheme (i.e., words with identical spelling but different lettercase were considered the same keyword). For this assignment you are to write an implementation called `concord3` to provide the same functionality, but this time using C and using dynamic memory.

We will, however, re-introduce two restrictions on input in order to help you with your problem solving for your solution. That is:

- You may assume that keywords will be at most 40 characters long, *and you may use* a compile-time constant to represent this.

- You may assume the length of any input line will be at most 100 characters, *and you may use* a compile-time constant to represent this.

Note that there *is no restriction* or upper limit on the *number* of distinct keywords or exceptions words, nor is there any restriction on the *number* of lines of input. (This is similar to assignment #2.) That is, you are not permitted to use any compile-time constants for these. **Put differently, you are not permitted to *statically* declare arrays large enough to hold all the keywords, exception words, and input lines.**

Therefore in order to store keywords, exception words, and input lines, **you must to use either linked-lists** or **dynamically-sized arrays** or some combination of both.

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

- `concord3.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 consisting of words can be constructed, traversed, and destroyed.

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

- `seng265-list.[ch]`: Type definitions, prototypes, and code 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 `concord3` executable, regardless of what files (`.c` or `.h`) are modified. The Unix `make` utility will be described in lectures.