CMPS 101
Algorithms and Abstract Data Types
Programming Assignment 2

In this assignment you will create a program with the same functionality as pa1, but now in C. You will again create a List ADT module and use it to alphabetize the lines in a file. Another goal of this project is to make sure everyone is up to speed with C, especially pointers and structures, and to build an ADT that will be re-used in future assignments. Be sure to read the two handouts "ADTs and Modules in Java and C" and "Additional Remarks on ADTs in C" before proceeding, paying special attention to the second handout. (Note some changes to these handouts have been made since last week with respect to naming conventions in C. Print them out again if necessary.)

As before the executable file for this project will be called Lex, and will be operated by doing

```
            % Lex <input file> <output file>
```

at the command prompt `%`. The program `FileIO.c` on the class webpage shows one way that file input and output can be accomplished in C. Program operation and file formats for this project will be identical to those described in pa1. As before your List ADT will be a double ended queue with cursor (possibly undefined) underscoring a distinguished element in the list. The underlying data structure for the list will be a doubly linked collection of Node objects. You may use the examples Queue.c and Stack.c as starting points for your design. Your List module will export a `List` type along with the following operations.

```
// Constructors-Destructors ---------------------------------------------------
List newList(void);
void freeList(List* pL);

// Access functions -----------------------------------------------------------
int length(List L);
int getIndex(List L);
int front(List L);
int back(List L);
int getElement(List L);
int equals(List A, List B);

// Manipulation procedures ----------------------------------------------------
void clear(List L);
void moveTo(List L, int i);
void movePrev(List L);
void moveNext(List L);
void prepend(List L, int data);
void append(List L, int data);
void insertBefore(List L, int data);
void insertAfter(List L, int data);
void deleteFront(List L);
void deleteBack(List L);
void delete(List L);

// Other operations -----------------------------------------------------------
void printList(FILE* out, List L);
List copyList(List L);
```

Function `newList` returns a `List` which points to a new empty list object. Function `freeList` frees all heap memory associated with its `List*` argument, and sets `*pL` to `NULL`. Function `printList()`

prints the `L` to the file pointed to by `out`, formatted as a space-separated string. This function plays roughly the same role as the `toString()` function in Java. The operation of the other functions, and their preconditions, are described in the pa1 specifications. Note that the `int` type in C will stand in for `boolean` in java, with 1 being true and 0 false. All of the above functions are required for full credit, but you may add the following function whose operation is described in pa1.

```
List concatList(List A, List B);
```

Your program will be structured in three files: a client program `Lex.c`, a List implementation file `List.c`, and a List header file `List.h`. Also turn in three additional files: `README`, `Makefile`, and `ListClient.c` (posted on the webpage) which is to be unaltered. Thus you will turn in 6 files in all. Note that these file names are *not* optional. Points will be deducted if you turn in wrongly named files, or extra files such as data files or object files. Each file you turn in must begin with your name, user id, and assignment name.

Your `Makefile` must create an executable called `Lex` and must include a `clean` utility that removes all object files, including `Lex`. A simple Makefile for this assignment is posted on the webpage under the examples section. You may alter it as you see fit. The webpage contains links to some good Makefile tutorials. You can also go to my webpage for cmps 12B, follow the lab assignments link

```
http://ic.ucsc.edu/~ptantalo/cmps12b/Summer13/lab.html
```

then read lab assignment 1 which contains a section on Makefiles. Note that the compile operations mentioned in the above Makefile call the gcc compiler. It is a requirement of this and all other assignments in C that your program compile without warnings or errors under gcc, and run properly in the IC Unix computing environment provided by Information and Technology Services (ITS). In particular you should not use the cc compiler. Your C programs must also run without memory leaks. Test them using `valgrind` on unix.ucsc.edu by doing `valgrind program_name argument_list`. Submit your project to the assignment name `pa2`.