Assignment 1

Overview

In this assignment you will implement an abstract data type (ADT). Specifically, you will implement a list ADT whose interface we have specified for you. The operations to be supported by the list are:

- Adding elements to the beginning or end of the list.
- Removing elements from the beginning or end of the list.
- Getting the current size of the list.
- Checking whether a specific element is contained in the list.
- Sorting the list according to some specified ordering.
- Iterating over the elements in the list.

There should be no bound on the number of elements that may be inserted into the list, short of running out of memory.

Algorithms

The purpose of this assignment is mainly to become acquainted with our chosen pattern for implementing ADTs. The algoritms employed are a secondary concern; you are free to choose any implementation you like, as long as you correctly implement the specified interface.

Two reasonable approaches are storing the elements in a doubly-linked list, or using an array which is dynamically resized as the list grows. If time allows, feel free to implement both of these. As for sorting the list, we suggest you start out by implementing one of the so-called elementary sorting algorithms that have a complexity of $O(n^2)$. You can always return to the code later to implement a more efficient sorting algorithm.

Applications

You will implement two applications that utilize your list ADT. The first, called *sortwords*, should tokenize a set of input files into a sequence of words, and proceed to print all words in sorted order. The second, called *reversewords*, should work similarly, but should print all words in the reverse order that they appear in the input. In both cases, the set of input files to process should be listed on the command-line, like this:

```
./sortwords sometextfile someothertextfile ...
```

Code

Your starting point is the following set of files:

```
data/
```

- `hamlet.txt`: Data file for testing your implementation

`include/`

- `list.h`: Specifies the interface of the list ADT. Do not modify this file.
- `common.h`: Defines utility function for your convenience: `tokenize_file()`
- `printing.h`: Contains macros for printing and coloring terminal text.

`src/`

- `sortwords.c`: An empty stub for the sortwords program (complete this).
- `reversewords.c`: An empty stub for the reversewords program (complete this).

`Makefile`: A Makefile for compiling the code.

Besides completing the sortwords and reversewords programs, you need to add a new source file (`.c`) file in the src/ folder that implements your list ADT. Once you've created that file (let's say you called it `linkedlist.c`), you need to make a minor change to the Makefile: Change the first line of the Makefile from `LIST_SRC=` to `LIST_SRC=linkedlist.c`. If you create multiple implementations of the ADT you can change which one is used by changing the value of the `LIST_SRC`. variable in the Makefile.

To test the implementation, we have provided a text file containing Hamlet inside the data folder. You are also free to use any other text file.

We've bundled all of these files in a zip file. The zip file is located in the same folder as this document (`p1-pre.zip`).

Tools

On Unix systems (e.g. Linux, OS X), you shold be able to compile and run the code for this assignment without installing anything, simply by typing `make` from the shell. Contact your TA if you have trouble.

Deadline

This assignment is not mandatory. However, we highly recommend that you complete it. We will give you feedback on your work if you choose to hand in the assignment.