Eric Roberts
CS 106B

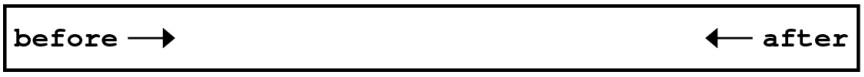
Handout #32
February 9, 2015

Section Handout #5
The `EditorBuffer` Class

Problem 1. The gap-buffer form of the stack model (Chapter 13, exercise 2, page 610)

Even though the stacks in the `stackbuf.cpp` implementation of the `EditorBuffer` class expand dynamically, the amount of character space required in the stacks is likely to be twice as large as that required in the corresponding array implementation. The problem is that each stack must be able to accommodate all the characters in the buffer. Suppose, for example, that you are working with a buffer containing *N* characters. If you're at the beginning of the buffer, those *N* characters are in the `after` stack; if you move to the end of the buffer, those *N* characters move to the `before` stack. As a result, each of the stacks must have a capacity of *N* characters.

You can reduce the storage requirement in the two-stack implementation of the buffer by storing the two stacks at opposite ends of the same internal array. The `before` stack starts at the beginning of the array, while the `after` stack starts at the end. The two stacks then grow toward each other as indicated by the arrows in the following diagram:



Reimplement the `EditorBuffer` class using this representation, which is, in fact, the design strategy used in Java's editor buffers and many other widely used editing tools. Make sure that your program continues to have the same computational efficiency as the two-stack implementation in the text and that the buffer space expands dynamically if necessary.

Problem 2: Doubly linked lists (Chapter 13, exercise 12, page 614)

Implement the `EditorBuffer` class using the strategy described in the section entitled "Doubly linked lists" at the end of Chapter 13. Be sure to test your implementation as thoroughly as you can. In particular, make sure that you can move the cursor in both directions across parts of the buffer where you have recently made insertions and deletions.

Thought question

The strategy used in Problem 2 has the advantage that all six editor operations run in constant time. Unfortunately, it does so at a high cost in memory space. Think about strategies you might use to reduce the space required while retaining the constant-time performance and be prepared to discuss these strategies in section.