Wesleyan University, Fall 2022, COMP 211

Homework 9: Linked lists again: an editor buffer

Due by 11:59pm on November 22, 2022

---

1. Written problems

There are no written problems for this assignment.

2. Programming problems (20 points)

For this assignment, you will use doubly-linked lists to implement the buffer of a (very simple) text editor. The buffer is the data structure that represents the text being edited, along with information about the current insertion point. It efficiently supports operations to insert and delete characters at the insertion point and to move the insertion point. The code for managing the buffer is typically independent of the code for managing the *display* of the buffer contents (i.e., what the user actually sees). More accurately, an editor typically consists of three components: the buffer, a *view* of the buffer (some sort of display), and a *controller* that accepts user input. At a high level, the controller is really an infinite loop that executes the following steps, where the buffer is initially empty:

(1) Update the view, which will display the text in the buffer, along with an indication of where in the text the next event (see the next step) will have an effect.

(2) Wait for an *event* that indicates that the user wishes to change the buffer. Typical events are a key press of an ordinary character (indicating that the user wishes to insert a character into the buffer), a key press of the backspace or delete key (indicating that the user wishes to delete a character from the buffer), or a key press of an arrow key (indicating that the user wishes to change the position at which the next event will affect the buffer).

(3) Modify the buffer according to the event.

The driver program provided in the code distribution implements a text-based version of a controller and view by presenting the user with a menu of options (like insert a character, delete the character to the left of the insertion mark, etc.) and displaying the contents of the buffer as a string. Provided the buffer structure implements an appropriate interface, neither the controller nor the view need to know anything about how it represents the text and insertion point. So your job is to implement the buffer structure so that it implements a specific interface.

In a bit more detail, you will write a module in which you define the following:

- A node structure for doubly-linked lists of characters.
- A buffer structure that has a doubly-linked list of characters (using your node structure) and a reference to one of those nodes to represent the current insertion point.

- An insertion function that inserts a new character into the buffer at the current insertion point.
- A deletion function that deletes the character to the left of the current insertion point.
- A deletion function that deletes the character to the right of the current insertion point.
- Two functions to move the insertion point to the left and the right.
- A function to set the insertion point to a specific position.
- A function that returns the contents of the buffer as a structure with two fields: a string representing the contents to the left of the insertion point and a string representing the contents to the right of the insertion point. The view uses this function to display the contents buffer and the insertion mark.

You will implement the functions in `hw9.h` in the file `hw9.c`. The header file `hw9.h` specifies that there are structures for nodes and a buffer, but it is up to you to fill in the details. You may not modify the `buffer_contents` structure definition; that is part of the interface between the buffer and the view.

3. Additional notes

(1) Writing automated tests for this sort of structure is a bit challenging. An appropriate test suite will test your buffer on many sequences of arbitrary operations. It is worth thinking about how you might implement such a suite of tests.

(2) You must define what it means for a value of type `struct buffer` to be a valid bufffer. For example, part of that definition will be that the linked list is in fact linear in both directions and that every node satisfies the back-and-forth property. Every function that takes a `struct buffer` argument must verify that the argument is a valid buffer, and every function that modifies a `struct buffer` argument must verify that the argument is valid before returning.

4. Code distribution and submission

As usual, the code distribution contains a driver program (`make driver`) and a tests program (`make tests`). Submit your written work as `hw9.pdf` and your code as `hw9.c` and `hw9.h` to the Google Drive directory I have created for you named `comp211-f22-USERNAME/hw9/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! And make sure that at the top of each file you have put your name! Do not, however, change the names of the files.