Pointer to Pointer Practice

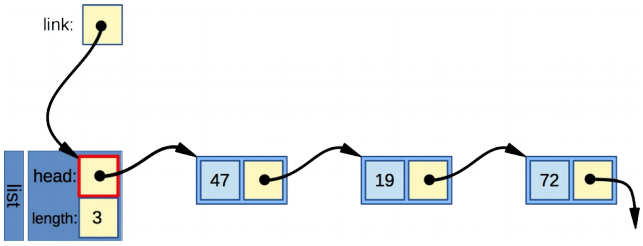
You'll find a source file, p2p.c, and text input/output files on the course homepage. You should also be able to download them with the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise20/p2p.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise20/input.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise20/expected.txt

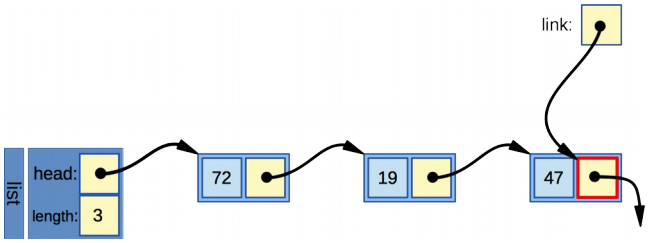
This program reads a list of numbers and stores them in a linked list. In fact, it does this three times using three different functions to build the list in a different order each time. You'll fill in the missing code in these functions. This will give you a good chance to practice using pointer-to-pointer in a safe, friendly environment.

Here's what the functions are supposed to do:

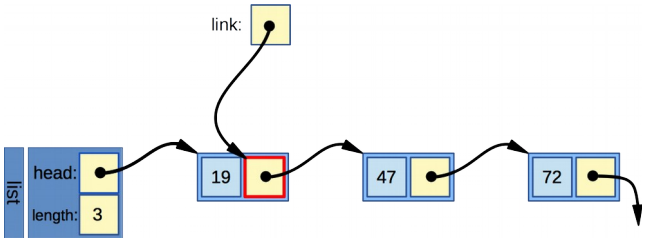
- The `insertAtHead()` function inserts the new value at the head of the list. We don't really need pointer-to-pointer to simplify this code, but it'll be a good warm-up, and we can re-use some of the code we figure out for this function in the later functions. As illustrated below, this function creates a pointer-to-pointer, `link` and sets it to point to the head pointer. Your job is to create a new node holding the given value, and add insert at the start of the list by using `link` to modify the head pointer.



- The `insertAtTail()` function inserts the new value at the tail of the list. Here, the `link` pointer is a little more useful. We're going to start it out pointing to the head pointer. Then, we're going to use `link` to walk down the list until it points to the null pointer marking the end of the list (even if that's the head pointer itself, like it will be when the list is empty). Then, just like with the previous function, we can use `link` modify the value of the pointer it points to and add a new node to the list.



- The `insertSorted()` function inserts the new value wherever it needs to go to keep the list in sorted order. This sounds like it might be tricky, but, really, it's almost the same as the `insertAtTail()` function. Again, we're going to start `link` out pointing to the head pointer. Then, we're going to use it to walk down the list until it points to either the null pointer at the end of the list or a pointer to a node containing a larger value than the one we're inserting. This will leave it pointing to the link we need to change to insert the new value. We can use `link` modify this pointer and insert the new node at the right spot.



As usual, I've marked the places where you need to add code with ... in a comment. You'll need to add a line or two of code for each of these, instead of a whole block of code.

When you're done, you should be able to run your program with input from the sample input file, with the following output. Here, the first list has all the number from the input file in reverse order. The second has them in the same order as the file, and the last has them in sorted order.

```
$ ./p2p input.txt
12 elements : 29 51 75 15 88 12 36 8 31 47 19 72
12 elements : 72 19 47 31 8 36 12 88 15 75 51 29
12 elements : 8 12 15 19 29 31 36 47 51 72 75 88
```

When you're done, submit the source, p2p.c, to the exercise_20 assignment in Moodle.