Data Structures

On the course homepage, you'll find a source file, insertion.c, along with a sample input and expected output file. You can also download them with the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise14/insertion.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise14/input.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise14/expected.txt

This is a partial implementation of a linked-list-based insertion sort. The program is supposed to read a sequence of integer values, inserting them into a sorted linked list as it reads them. At the end, the program prints out the contents of the linked list and then frees its memory.

You'll need to add code in three places:

- First, fill in the body of the insert() function. This function should dynamically allocate a new node for the integer value passed in and insert the new node in the right location in the list to keep it in sorted order. If you like, you can use the pointer-to-pointer technique introduced in class to simplify the linked list code. Or, if you prefer, you can implement insertion the (slightly) harder way, by looking for a predecessor node and handling inserting at the head of the list as a special case. This will make your code a little longer, but it may be easier to think about.
  The insert function returns an updated head pointer for the list. If the new value has to be inserted at the front, the caller will get back an updated head pointer. If the value is inserted later in the list, the caller will get back the same head pointer they passed in.
- After you've built the list in sorted order, write code to print out the elements of the sorted list on a single output line, with a space after each value.
- At the very end, add a block of code to free the whole linked list. Be sure you don't use a node after you've freed it.

As usual, I've marked the places where you need to add code with ... in a comment.

When you're done, you should be able to run your program with input from the sample input file, and get the same set of values in sorted order:

```
$ ./insertion < input.txt
13 215 266 301 347 355 381 458 490 577 584 779 825 835 911 950
```

When you're done, submit the source, insertion.c, to the exercise_14 assignment in Moodle.