CSC 362 Programming Assignment #6
Due Date: Friday, December 2

The bin packing problem can be stated as follows: you have a collection of items, each of a different size, to be placed into several equal sized containers (bins). How do you place the items in the bins to use the *fewest* bins. For instance, you might have bins that can handle a weight of up to 1.0 with items of weights .5, .9, .3, .4, .7, .4, .2 and .4. One solution is to just use the first bin available if it will fit, otherwise go on to the next bin. With this strategy (called the *first fit* strategy), the above weights would go as follows: bin 0 gets .5, bin 1 gets .9, bin 0 gets .3, bin 2 gets .4, bin 3 gets .7, bin 2 gets .4, big 0 gets. 2, and finally bin 4 gets .4. Thus, we solve the problem using 5 bins. Three other strategies are:

1) use the bin with the greatest available capacity (worst fit strategy)
2) use the bin with the smallest available capacity (best fit strategy)
3) try all possible combinations of items into bins (optimal strategy in terms of fewest bins but requires an enormous amount of processing time). Using this approach, our above example would fit in 4 bins (.9, .3 & .7, .4 & .4 & .2, .5 & .4).

For this assignment, you will implement the *first fit* strategy. Use an array of Bins where each Bin is a struct that stores its current available capacity (initialized to 1.0) and a pointer to a linked list of items currently placed into that Bin. The linked list will consist of struct nodes that store the name of the item (a string), the weight of the item (a double) and a pointer to the next struct node in the list. The first fit strategy is given below in pseudocode:

```
For each bin b in the array of Bins, initialize b.capacity = 1.0 and b.list to NULL
For each item, i, to be placed
    Input the item, i (from keyboard or a disk file, or it can be hard-coded into the program)
    While i is not placed in a Bin, iterate through each Bin, b, in the array of Bins
        If item i can be placed into b (that is, is b.capacity + weight(i) <= 1.0)
            Then create a node for i, filling in i.name and i.size
                and insert this node into b (in sorted order)
                and modify b.capacity
        Else go on to the next bin
When done, output number of bins used and for each bin b, its list of items, and the final
available capacity
```

NOTE: b.capacity + weight(i) may have a loss of precision, so the value may actually be slightly greater than 1.0. Use a small amount of error like .00001 to ensure that a loss of precision does not cause an issue with your program.

This program requires two structs (a node struct and a bin struct) and one array (of bins). You will have to implement these linked list functions: **ordered insert** to insert the new item into a node in its appropriate location using alphabetical ordering of item names, **traverse the list** to print the items of the list out along with the list's capacity, **destroy the list** to deallocate the nodes of the list.

Here is an example of input and the bins created for you to test.

Vase (.5), Bowling ball (.9), Book (.3), Umbrella (.4), Gold medal, (.7), Speaker 1 (.4), Walkman (.2), Speaker 2 (.4).

Using first fit, they would be placed as follows:

Bin 1 (.0 left): Vase (.5), Book (.3), Walkman (.2)
Bin 2 (.1 left): Bowling Ball (.9)
Bin 3 (.2 left): Speaker 1 (.4), Umbrella (.4)
Bin 4 (.3 left): Gold medal (.7)
Bin 5 (.6 left): Speaker 3 (.4)

The best answer is 4 bins: (Umbrella, Vase), (Bowling Ball), (Book, Gold medal), (Speaker 1, Speaker 2, Walkman).

Debug your program on the above sample input. Next, run your program on the two sets of data below. Hand in your commented program and the output of both runs. NOTE: your program must have separate functions for insert, print, destroy, but the program does not have to be placed in separate files.

NOTE: C does not like spaces in strings, so either use – or _ between words of the input as in "Small-dog" or "Small_dog".

Run 1: Small dog (.63), Moose skull (.25), Squirrel (.41), Mouse (.15), Goldfish (.06), Snake (.52), Human finger (.09), Pig head (.39), Eagle feather (.02), Shark teeth (.11), Camel hump (.24), Crocodile (.94), Elephant tusk (.63), Cat (.28), Horse manure (.04), Monkey hand (.21), Octopus eye (.05), Sheep coat (.33), Bat (.42), Chicken wing (.12)

Run 2: [These are all books] Operating Systems (.75), Computer Graphics (.62), The History of the Universe (.27), Discrete Mathematics (.78), Statistics (.41), English for Dummies (.55), American History 2000- (.20), Computer Architecture (.90), C Programming (.21), Hitchhiker's Guide (.23), English-Minbari Dictionary (.42), Zen and the Art of Programming (.33), The Joy of Cooking (.56), Cincinnati Yellow Pages (.98), Artificial Intelligence (.32), Business Programming: Why? (.12), Heart of Darkness (.21), The History of the Ohio State Buckeyes (.82)