Prelab 12: The Last One

Implement functions `insertBST` and `findBST` to permit a data item to be inserted into a BST or to be found in a BST, and implement a function to free memory allocated for the BST. You'll of course also need a function `freeBST`.

REMEMBER: *Any time you find yourself replicating code segments (e.g., using copy-and-paste) then that's a sure sign that the segment should be replaced with a function with a meaningful name. Long segments of code that distract from the logic of a function should also be replaced with a "helper" function. For example, instead of cluttering your insertBST implementation with multiple lines of code for allocating and initializing a node, why not make a call to a function like* `createNode`*?*

QUESTION: How can we create BST functions that can be used with generic void* objects?

ANSWER: Suppose we require the user to provide a function with the following prototype:

```
int compareBST(void *obj1, void *obj2)
```

which takes pointers to two objects and returns a negative number if obj1 is less than obj2, or zero if they are equal, or a positive number if obj1 is greater than obj2.

If the user is required to provide this function, then we can assume it is available whenever there is need to compare two of the user's objects. This should be an easy function for the user to implement because s/he knows how the objects need to be compared, e.g., using the SSN members if the pointers are to Employee structs that are to be searched according to SSNs.

Note that comparison functions can be used by any ADT to allow generic objects to be handled and compared without knowing anything about those objects. I mentioned in lecture that you can look at the C library sorting function, qsort, to see an even more general approach using pointers to functions (aka, *function pointers*).