Prelab 8: Week of March 22nd

For this prelab you are to implement the following two additions to your List ADT from last week:

```
/* This function inserts the object in its ascending
   order position according to the user-defined global
   comparison function cmp(void*,void*). */
int insertInOrder(void*, List)  // Returns an error code

/* This function deletes all objects that are
   evaluated as equal to the object of the first
   parameter according to the user-defined global
   comparison function cmp(void*,void*). */
int removeEq(void*, List)     // Returns number of items removed
```

The user-defined global comparison function `cmp(void*,void*)` is assumed to return a negative number if the first object is less than the second object; zero if they are equal; and a strictly positive number if the first object is greater than the second object. It is the responsibility of the user to implement this function because only the user knows what it means for one object to be less than, equal to, or greater than another object. For example, if the user is working with Employee structs then the `cmp` function might be implemented to compare the SSN member, e.g., so that objects are inserted in sorted order by employee social security numbers.

The important thing to note is that the List ADT does not and should not need to know anything about the objects that are being stored on the list. However, if we want to support functions that maintain some ordering of the objects then the *user* must be required to provide us with a function that allows us to determine whether one object is less than another object.

Even though you're only being asked to implement two functions, the more substantial part of this prelab will come from your testing of those functions using different kinds of objects, e.g., Employee, using different implementations of the `cmp()` function. How can that be done? Look at the following and verify that it does what we need if we want to compare Employee records according to ascending sorted order by social security number.

```
int cmp(void* obj1, void* obj2) {
      Employee *a, *b;
      a = obj1;  b = obj2;
      return (a->SSN - b->SSN);
}
```

Everything will click for you once you study understand the above example. The key take-away from this prelab is that whenever a collection ADT requires information that is specific to the objects that are being stored, that information will have to be provided via a user-defined function. (Note that removeEq will also need to use the user-defined comparison function.)

To keep things simple, this prelab assumes a global comparison function with a specific name (`cmp`). It turns out that there's a much more general way to do things using *function pointers*. If you look up the C library sorting function `qsort` you'll see how it uses a pointer to a user-defined comparison function to allow it to sort objects of any data type. (You'll also see why this prelab uses something a bit simpler for now.)

- JKU