CS 214: Systems Programming, Fall 2013
Programming Assignment 2: Sorted List

1 Introduction

In this assignment, you will practice with more complex data structures, as well as practice using function pointers (along with using data pointers as in the last assignment).

Your task is to write a set of types and functions that implement a sorted list. The sorted list will contain opaque objects. That is, the objects will be given to you as `void*` objects. When a sorted list is first created, the caller will provide you with a pointer to a comparator function. This comparator function will understand the actual type of the objects to be stored in the sorted list, and, given two objects, will return an ordering of the two objects. Subsequently, when new objects are inserted into the list, you will use the given comparator function to insert the new objects such that the list will remain sorted in descending order; that is, objects are ordered from largest (front of the list) to smallest (end of the list).

You will also implement an *iterator* to help users walk through lists. This iterator, together with returning pointers to your sorted list objects as `void*`, will help you practice implementation hiding. That is, your implementation is similar to a Java class, where the users do not know about the implementation and so cannot access parts of the objects directly. (In C, there are obviously ways to get around your hiding; nevertheless, it is good programming practice because it requires effort to violate the hiding.)

2 Implementation

Your implementation needs to export the interface given in the attached `sorted-list.h` file. Specifically, you need to implement four functions for creating sorted lists, destroying sorted lists, and inserting and deleting an object into/from a sorted list. Your sorted-list data will be of the type `void`, so that you can pass any type into data struct. Rather, this is a way in C for you to practice a bit of implementation hiding. When writing your code for the sorted list, you will need to define a type for your sorted list objects. For example:

```
struct SortedList {
    ...
};
typedef struct SortedList* SortedListPtr;
```

You should create a pointer to `struct SortedList` object in `SLCreate()`.

```
SortedListPtr SLCreate(CompareFuncT cf) {
    SortedListPtr sl;
```

```
    ... /* do what is needed to create the sorted list object. */

    return sl;
}
```

The comparator function must obey the following semantics: return -1 if the 1st object is smaller, 0 if the two objects are equal, and 1 if the 2nd object is smaller.

You will also need to define a *helper* iterator type together with three functions for creating sorted list iterators, destroying sorted list iterators, and obtaining the objects in a sorted list one at a time. In this assignment, the iterator is just a wrapper around a sorted list that is used to help the caller walk through the list. Again, data is returned as `void*` to hide your implementation.

One complication that you must deal with is what happens if the sorted list is modified (e.g., a new object inserted or an existing object is removed) while an iterator is being used. You should explain how your implementation deals with this complication as a comment in your code.

As always, your code should be well-designed, well-organized, and well-commented. Both your design and implementation should be efficient. However, for this assignment, *y*ou may use a linear structure rather than implement a more complex data structure such as a tree, heap, or hash table to make insertion/deletion more efficient. It is sufficient that you implement your linear structure efficiently.

3 What to turn in

A sorted-list.c file containing all of your data structure code. At the top of the file, you should include as a comment a big-O analysis of the runtime of your code. You should also carefully comment all of your code. Your grade will be based on how well your code is working *as well as* how well written your code is (including analysis of runtime and comments) and how carefully you tested your code. A main.c file should including test cases and code to call the libaray.

A tarred gzipped file named `pa2.tgz` that contains a directory called pa2 with the following files in it:

- An `sorted-list.h` file containing the interface we gave you and your structure definition. *The function defintions must remain unaltered!*
- A `sorted-list.c` file containing your implementation of the sorted list.
- A `main.c` file containing a main function that exercise your sorted list implementation using the test plan outlined in `testplan.txt`.
- A `Makefile` that is used to compile your sorted list implementation into a library called `libsl.a` and an executable called `sl` that runs the code in `main.c`.
- A file called `testplan.txt` that contains a test plan for your code, including input and expected output.
- A `readme.pdf` file that contains analyses of the running time and memory usage of each of your sorted-list functions. Use big-O notation to describe the end result of each analysis.

Suppose that you have a directory called `pa2` in your account (on iLab), containing the above required files. Here's how you create the required tar file. (The `ls` commands are just to help show you where you should be in relation to `pa2`. The only necessary command is the `tar` command.)

```
$ ls
pa2
$ tar cfz pa2.tgz pa2
```

You can check your `pa2.tgz` by either untarring it or running `tar tfz pa2.tgz` (see `man tar`).

Your grade will be based on:

- Correctness (how well your code is working),
- Quality of your design (did you use reasonable algorithms),
- Quality of your code (how well written your code is, including modularity and comments),
- Efficiency (of your implementation), and
- Testing thoroughness (quality of your test cases).