Program #1 Review

Written by Alexander DuPree

Data Structure Performance

Program #1 focused on the use of the linear linked list data structure. As such, the data structure performed great for insertion and deletion operations. We were able to add to the list in constant time without having to resize the container.

The biggest downfall of this data structure is sequential search and access to elements within the container. This is partially mitigated by use of a tail and head pointer, we can access the front and end of the list with minimal overhead. However, accessing any other element, or searching for an item, requires that we traverse the list from begin to end.

Another requirement for this assignment was to keep the Categories sorted alphabetically. This led to two options: keep the list in a sorted state, or write a sort algorithm for the list. I chose to keep the list in a sorted state. This came at the cost of runtime performance when adding elements to the list, because we have to travel the list until we encounter an element larger than the data to be inserted.

Data Structure Recommendation

Another intuitive requirement for this assignment, was that there could be no duplicate entries. So in total we had the following requirements for our data structure:

- sorted state, or easily sorted
- dynamically sized
- searchable
- unique items only
- fast element access

The linear linked list performs great when only considering the dynamically sizeable trait, but fails in all other aspects. Searching, element access, and ensuring a unique/state are all linear time operations.

I am still learning the traits of each data structure, however a search tree or hash table would be better structures for this assignment because of the search and element access performance.

Design

A primary goal of this assignment was to abstract the implementation of the data structure. Each ADT used in this assignment had no knowledge of the linear_linked_list structure and only used its public methods. Internal types like Nodes are never referred to directly outside the linear linked class. To access elements within the list, I made use of iterators to ensure guided traversal of the list was done correctly.

Other ADT's in this assignment followed this design decision as well. Each ADT provided specific public methods to the client and did not reveal, or return anything specific to its implementation.

Inefficiences

I believe the Project ADT could be implemented more efficiently. The Project ADT is very rigid, and if we tried to change it (like how many attributes it has or the name of those attributes) we would have to change how input is handled at the highest level in main.

Feature Additions

If I had more time I would have liked to created a validator object that could scrub and handle user input as well as make the Project and Category ADT's more dynamic.

For example, one of the required attributes of the Project ADT is a due date string. However, the user can instantiate a project object with the string "I like chips" as the due date. Having a validator that can scrub the user input would ensure that data going into the ADT actually makes sense.

The Project ADT is also limited to it's 5 fields determined for this assignment. Even if the user created a category for "English Assignments" the Project ADT would still require an entry for the data_structure field. It would be a nice feature if the user could specify the fields for a given category, and instantiate an project object that only accepts those fields.