RYAN FILGAS
CS 163
KARLA FANT
1-18-2020

Program 1 Efficiency Write-up

1. How well did the data structure selected perform for the assigned application?

An LLL of LLLs performed well in this case because the program needed the ability to have flexibility to increase, or decrease the size of a list without having to copy the data over to a new list repetitively. Had this list had a million data members, it would be more efficient to segment the data even more, or switch to a more efficient data structure. Having to traverse two lists a node at a time isn't extremely efficient, but it's flexible. A branching structure of LLLs that has pointers to groupings of nodes by each letter could be a start, followed by a sub_branch of the second letter etc. could potentially be a faster way to do this with linear linked lists if in fact there were a very large amount of resolutions. It would be harder to implement, but it would have better search performance for massive lists, and definitely not for short ones. Of course, having indexes to use, and doing a binary search could be incredibly useful. LLLs are great for small lists that need flexibility, but have diminishing returns the more items that are added.

2. Would a different data structure work better? If so, which one and why...

As mentioned above, if the list is larger, then a different structure would be more efficient. I don't know much about hash tables, but from what I understand they are fast at accessing data with many members.

3. What was efficient about your design and use of the data structure?

Design wise, my classes are divided into .cpp files, and each class manages its own private data, and operations on that data. This makes it easy to maintain. Functions will return different integers depending on why they may or may not have failed.

Other than things I would do differently with more time, my program did have some positive efficiency improvements. When doing an insertion sort for the tasks, my program inserts before the smaller priority node instead of after where more conditions would be needed to address a list with two nines when it needed to add an 8 for example. A condition would need to be added to keep traversing until it stopped seeing 9s, but my solution avoids that extra work. The program is also space efficient. All arrays are dynamically allocated, and all functions pass either by reference or by pointer as well for efficiency. This makes retrieval, and storage faster / more efficient.

RYAN FILGAS
CS 163
KARLA FANT
1-18-2020

4. What was not efficient?

My program did have some efficiency errors, specifically with how the classes work together, and also just implementation. Since I used classes for every data type, resolution list, resolutions, tasks, the data members were private, so I had to create a function for the user to fill a task, and then copy that task into the selected node by passing it into the list member function as a task object rather than directly taking the information in as arguments. This may have been the intention of the assignment however; I was slightly confused by the exact implementation it's supposed to have. I'm thinking it may have been more straight-forward had I used structs for resolutions, and tasks, but I'm still learning about implementation details for classes and structs. The biggest thing, is the program had to copy strings quite a bit for comparing in addition to converting them to lowercase for the comparison so the classes could interact with each other, and I'm not sure it's the best use of processing power

5. What would you do differently if you had more time to solve the problem?

If I had more time for this assignment, and the opportunity to start over, I would do a few things.

a. I would segment the work into smaller functions. Specifically I need a function to Traverse resolutions, and check for a name match. I could have made a function to do ordered insert on a LLL of tasks. I could have made more use of recursion and wrapper functions to reduce lines of code traversing.
b. I could have made a second .h file for the test program (I tried, but couldn't implement successfully in a short amount of time).
c. I could have made more functions for a more complete data type. Clear list, delete task, delete
all tasks etc.
d. I could have packaged my test programs arguments into a struct, or kept them as local variables to reduce the argument lists.