CIS*2500 W20 Assignment 4: Questions & Answers Part 2

"I need to assign value_t in two different ways depending on its data type.
If I come across an integer, I can simply write:

new_node->value = value;

But if I use the character version I must use strcpy().

I just wanted to know if there is a way to assign two values equal to each other without having to write two different functions for each case."

This is an important question.

~~Furthermore, it applies to not only append and push, but also to empty and destroy as you have to free the strings stored in value, but not an int. Otherwise you will have a giant memory leak!~~

[edit: prompted by an email, I thought back and remembered that I designed against the need to free the string in the Node struct by demanding an 80 char limit on the strings. Consequently, they don't have to be malloc'd and so do not have to be free'd]

There are two approaches to doing this. Only the first approach can be used without a change to various signatures, so I am only giving the other one for didactic and completeness reasons.

Approach 1:
This is the simplest approach to apply and does not require a change to assignment's design. However, it is the not as flexible as the other approach as it requires changes to the code whenever a new datatype is used for value_t and/or key_t.

The idea is to use the same #ifdef approach taken with the typedef statements for value_t and key_t.
In other words you are writing alternative code for the assignment statements for each different datatype added. For example:

```
#ifdef INT
     new_node->value = value;
#endif

#ifdef CHAR
     /* your strcpy statement */
#endif
```

~~Do the same for the free statements in empty() and/or destroy().~~

Approach 2:
Do not use this approach as it runs counter to the assignment design, although it is better design in general.

- First create a function to assign values to the key-value pair
  passed in as value_t * and key_t *
- Next modify the push(List * list, value_t value, key_t key) function so that it becomes

  push(List * list, value_t value, key_t key, void (*assign)(value_t *, key_t *))

  modify append etc. to match

- Finally, change the assignment statement inside the push/append etc. functions from
  new_node->value = value;
  to
  assign(new_node->value, new_node->key)

This would now work for any value_t / key_t without having to change the code.
You would, of course still need to recompile.

Mark Wineberg

Note: this approach can be made even more general using void * and casting. This would negate the need to recompile the library that holds the List ADT. However, that lecture will only be handed out next week and I didn't want to base our assignment on a subtle point that we had yet to cover in class -- regression only primarily affected one questions; this concept would have effected every part of the assignment making the assignment too difficult for those who might have difficulty with the concept.

“I am emailing to inquire about the function parameters for all of the remove function, and also to gain some clarity on them. I don't understand why we need to pass a value and key into the function. It would make sense to me to do that if we had to remove a node at a certain position in the next links or the sort links (searching for a certain key or value). However, since we are always able to see the first/last in the next links and the first/last in the sort links, we have everything we need to remove them. I also understand why we should have success or failure. The most obvious failing case I could think of is trying to remove anything from an empty list. In that case, the function can check for that and simply not do anything. I'm curious how you would feel about something like:

```
void remove_largest_key(Sorted_List * list){
   /* code */
}
```

Let me know! I could very well be missing something obvious and not be realizing it.”

The value_t * and key_t * parameter in the remove function is not to get information in, it is to get information out.

Without nth, which I only ask for in q2, there is no way to see what is in the list, or what is being removed.

With nth (or first/last if just looking at the removes) you could, as you say, just remove the element; you would first apply nth(list, 0, value_t *, key_t *) to get the information out, then apply remove_first, without the other two parameters.

You need this information to print out what is being removed according to the command specification.

Hope this helps.

Mark Wineberg

PS Traditionally, remove functions all pass the information out, whether or not first/last/nth exists or not.

1. What exactly are "the contents" of the list in terms of emptying vs. destroying? Should I only free the values, keys, and sorting links, or all the links (including the unsorted ones)?

First to destroy, you have to empty first. So there isn't a difference between the two in that regard (see my answer to empty vs destroy in the Q&A now posted in Courselink).

Second, you are removing nodes, not links. The node contains the links, so by destroying the node, you are destroying all links within it: both next and sort. You do use the links to find the nodes. Since both next and sort will separately find all links, you only need to traverse one chain and free the nodes as you go.

There is another issue with respect to freeing the content of a node (the value_t), but I will answer that in the Q&A part 2.

2. In terms of updating the sorting in the list, is there a specific good way to do it for this assignment? Any specific algorithms that may be useful (e.g. bubble, insertion, merge) or is there a "sneaky" way to update our lists?

Follow the class notes on sorted linked lists. The technique used is insertion sort, which is the natural sort to do for a linked list. While merge sort is the best sort for linked list, it requires very careful pointer manipulation that will be time consuming to do. There is no need to do this for the assignment as we are grading for functionality, both sort the data, not efficiency. Finally, you do not learn about merge sort until the data structures course, which requires the use of recursion or the use of a stack (not just knowledge of its existence) so I do not expect it to be applied here.

In summary: There is no sneaky way. Just use insertion sort which is outline (although not named) in the course lecture notes.

Mark Wineberg

“Sir I wanted to ask that should we implement main in 1b or not? Can u explain it to me? That how to implement main in 1b?”

Every distinct program, such as a4q1a, a4q1b, a4q2 and a4q3 has to have its own main().

The main() function is the starting point for the program. Any program that runs must have a main().

You can have .c files that compile to .o files, none of which have a main() function in it. This is the usual case as these functions can then be used in many different programs, not just a single program. However, none of the compiled code produced can run by itself. It needs to be linked with a program being compiled with a main in it.

Read over lecture 1 notes in detail to understand the idea.

Regarding the assignment, it is true that since a4q1b, (as well as a4q2) just add commands to those processed by a4q1a, separate mains would technically not be necessary. It could just be a4q1 program.

However, a4q3 modifies how the a and p commands operate. This requires a different interpretation when the input file commands are processed by your program. Consequently, it needs to be its own program.

Consequently, to be consistent, I made all parts into their own programs with their own main() functions.

Finally, to reuse the input command processing code developed for the main() in q1a, if this code actually resided in main() you would have to copy and paste it into the new main. This, however, is not the best approach. Instead you should have the command processing code in its own function in its own file. That function would then just be run in the different mains in the different programs.

Hope this helps,

Mark Wineberg