CIS*2500 W20 Assignment 4: Questions & Answers

There is an adage in teaching: if one student has a problem and is brave enough to ask, it is likely that another 10 have the same problem but are too afraid to speak up.

With this in mind, here is a compilation of all of the questions I have been getting by email, along with my answers.

I will be updating this file on regularly … provided I keep getting asked questions :-)

Note: The questions have been anonymized and slightly edited for clarity and conciseness.
My answers remain unchanged.

"I'm confused about what how to use the value_t and key_t definitions. I tried at the top of my main just after the headers get called, but before the int main (...) is written, where I entered typedef int value_t; and typedef float key_t; It still does not seem to work. So, I tried putting them at the bottom of my .h file and there still are errors when compiling. This is a confusing concept to me."

typedef int value_t;

must appear before

```
typedef struct NODE {
    value_t value;
    struct NODE * next;
}
```

So if typedef int value_t is in a .h, that .h must be #included before a .h that might hold the NODE typedef.

Mark Wineberg

"I'm not sure why we need the two extra parameters value_t pointer, and a key_t pointer, if we are just removing the first node. You explain that we should return the values of the remove node through the function parameters, and I don't quite understand what you mean by this."

The two parameters, value_t pointer, and a key_t pointer, are to be used as "pass-by-reference". In other words, the values for those parameters can be set inside the function and seen outside of it once the function has completed.

When you remove the node, the information contained in the node will be lost. You need to get that information to the user of the remove function. The "user" is the main or some other calling function that could then use the information from the removed node in some fashion. In the assignment, the information is simply printed, but it could be used for many other reasons.

If you do not need the information from the removed node, and just want the node removed, you can just ignore the values being returned through the parameter (use temp variables to "receive" the information, and then just don't use them).

e.g.

```
value_t my_value;
value_t my_key;
remove_first(list, &my_key, &my_value);
```

Mark Wineberg

“Hello, ive been working through A4 and came across something a bit confusing
A4 states:
"int remove_first(Sorted_list *, value_t *, key_t *)"
"returns the value and key of the removed node"
returns do you mean it stores it in the value_t* ptr?
what is the 'int' return for?”

Yes, store the results in value_t *, key_t *

The int return value is to return SUCCESS or FAILURE. With remove_first, it is not possible to fail, so I could have designed it as returning void. I will change the assignment to allow either approaches.

Mark Wineberg

"Just to clarify, should we have an array of pointers to Sorted_Lists? I was reading the descriptions for the silent commands and I am not sure what array that 'n' is index of, so would it be an array of Sorted_Lists? This is from the a and p commands specifically.

I think that making this clearer, especially at the beginning of the Q1b description could make it easier to understand. Also, if this is the case, how large should our array be or is there a max size of Sorted_Lists that we can have at one time?"

From the assignment on page 3, which begins the section on the testing program:

- Make sure you have declared an array that can hold up to 10 Sorted_List pointers
    - Do not confuse this with the array produced by map_2_array, this array holds Sorted_List pointers **not** value_t values.

It's first use is with push and append two pages later. I will add a reference back to this description in append/push section and update the assignment for clarity later this evening.

Mark Wineberg

Two different students wrote in noticing the same problem:

"In q2, count up is increasing by 2 to 2n, but in the input commands count up is increasing to n by 1 and count down is increasing to 2n by 2. Is there a swap between both the functions in the commands section? Or what? So is it a mistake or what? I am not able to understand it."

"The instructions for Q2 says to implement a function that counts up from 0 to 2n by 2, and a function that counts down from n to 0. However, the list of commands says to count UP from 0 to n, and to count DOWN from 2n to 0 by 2. Is this a typo? And if so which 2 functions should I actually make?"

Typo (more accurately, I was tired when I wrote the example).

I will correct the assignment but allow either approach as permissible (as long as they are recursive).

I will correct the assignment to be consistent to the version in the example (up by 1, down by 2) as that has more detail.

Mark Wineberg

“I was wondering if we were allowed to change your function signatures for assignment four. ie. how many arguments for each function? I want to use the action value as part of the linked list as you have the action needed to be performed along the left edge of the file in your example eg. (append key value).”

No, you would lose marks for that.

It violates the idea of the abstract data type.

You are tailoring the ADT to a specific input/output application that the ADT is being applied to.
It can no longer be available to different applications that are not command driven, or have completely different types of commands.

The ADT should have nothing to do with the application, which in our case a simple command driven script to test out the ADT without doing anything useful.

A function should do one thing, and one thing well, as should an ADT. It should not link information and process commands at the same time.

Mark Wineberg

“Hello, I was just wondering. For the ADT functions, do you want us to completely rewrite it from scratch in our own way. Or are we allowed to use what is given to us in the lecture notes, and then tweaking them to work for the assignment?”

You can use the lecture notes version and tweak if that helps you. Write it from scratch if it doesn't.

Mark Wineberg

“For A4, we’re supposed to implement the size function. What I’m confused about is the instructions say to return success for all functions except if stated. So, for that function, what is it supposed to do? Should it add 1 to the size of the ADT or do you want us to return its size? Thank you very much.”

Good point.

Return its size, as the function cannot fail.

I will update the documentation to be clearer.

Mark Wineberg

“For the first part of this assignment, are we supposed to take the input by whatever means it is from the command line or the user, then create a linked list with this data?
Also, it says we need to have a sorted list for the functions to get passed. In what way do we sort the functions by? Is it by key or value?
Also, how are we supposed to know which way the user wants the list printed, or do we print all the options (how it is put in, after it is sorted and after all the commands are read)?”

The commands come from an input file, not the command line. The command line only takes the file name.

You sort by key, just as in L4 (this is specified in the instructions).

The user has either print_all or print_sorted as commands in the input file. In other words, it is determined by the user. They can print it one way, the other way, both ways, multiple times, whatever they want.

The sorting is done when an element is entered using append or push. When it comes to printing the sorting is already done. You just follow the sort links, just as in L4.

Mark Wineberg

“In regards to the last update of the assignment, If empty also frees the list then isn't it the same as destroy list?”

No.

empty() empties the contents of the list, but the list still exists with 0 members.

destroy() gets rid of the list entirely. There are neither members, nor a list.

e.g. with empty size(list) => 0 while with destroy size(list) produces garbage (and you won't have a memory leak if you exit the program.

Of course, destory() can call empty() before freeing the list itself (what I would do).

Mark Wineberg

“i am really confused about the assignment in A4q1a_char.c. The output has 22 23 22 and 29 where did it come from? Furthermore, what about the input? Where did it go? We don’t seem to be using append either? The main is really confusing me, i wonder if you can clarify a bit?”

strlen("The sun did not shine.") == 22
strlen("It was too wet to play.") == 23
etc.

From the assignment:

- a4q1a_char.c
  - has its value_t datatype set equal to char[80]
    - i.e. it can take strings up to 79 characters in length
  - has its key_t datatype set equal to int
    - its value is set equal to the length of the string

In other words, value_t value holds the input data and key_t key holds the length of the input string that value_t value holds.

Finally, on what happened to the append, again from the assignment:

- Example using the input from the append examples

In other words, the print statement happens after all the append statements. There were just not repeated to keep the instructions shorter.

Mark Wineberg

“Do we use sort function inside push? Or when we get to print_list_sort?”

Inside push/append.

The print just follows the links that have been set up. If it follows the next links, it prints in in insertion order. If it follows the sort links, it prints it in sorted order.

If you sort every time you print, you are constantly sorting and resorting the same data into the same sorted order, over and over and over and over again. One more time every time you print.

Mark Wineberg

“In the instructions we have to create a Sortedlist ADT. I was just wondering how we use that and how it connects to the linked list.”

The sorted linked list is a way to implement the Sorted List ADT, just as a regular linked list can be used to implement the list ADT (see the lecture notes on List ADTs).

You need to create a structure to hold thinks like the head pointer (or pointers in this case) etc.

Mark Wineberg

“What I am supposed to do for the section titled "Create a Sorted List abstract data type?"

Create a structure is a manner similar to the structure created for the List ADT in the lecture notes.

value_t and key_t
This is described in the assignment itself, as well as in the updated lecture notes on Map/Filter/Reduce. value_t and key_t are typedef names, where you declare the typedefs at the head of the file and/or in a .h file. For example if the values and keys are floating point values, you would have in the .h typedef float value_t; and typedef float key_t; Then whenever you use value_t x = 2.3; it is as if you had written float x = 2.3;

typedef int value_t;

must appear before

```
typedef struct NODE {
    value_t value;
    struct NODE * next;
}
```

So if typedef int value_t is in a .h, that .h must be #included before a .h that might hold the NODE typedef.

To switch between different value_t typedef definitions, as the assignment has you use different ones for different questions, you use #ifdef as described in lecture 1 and the assignment.

Example input file and the output it produces
This will be coming out as a supplemental document sometime tomorrow.

Mark Wineberg

“Why is there Prev Tail Pointers for the non-bonus approach”

This point will be removed from the instructions.

The use of previous tail pointers was meant to help you with remove_last and remove_smallest as you need to set the tail to the node pointing to the tail. However, storing this node does not actually help you as you need to find a new prev_tail each time you have removed a node from the end, which means you have to go through the list to find the previous to last node each time; this is no different than going through the list to find the new last node, so there is no advantage is keeping this extra pointer around.

Consequently, when appending, the tail pointer helps, but with remove_last you are forced to look through the list for the previous to last node and make it a new tail, unless you used doubly linked lists. Keeping the previous tail pointer does not help, so you should not add it to your struct.

I will be updating the instructions shortly (I am just going to remove the point, not give this explanation).

Mark Wineberg

“What is the meaning of the variables types value_t and key_t?”

This is described in the assignment itself, as well as in the updated lecture notes on Map/Filter/Reduce. value_t and key_t are typedef names, where you declare the typedefs at the head of the file and/or in a .h file. For example if the values and keys are floating point values, you would have in the .h typedef float value_t; and typedef float key_t; Then whenever you use value_t x = 2.3; it is as if you had written float x = 2.3;

To switch between different value_t typedef definitions, as the assignment has you use different ones for different questions, you use #ifdef as described in lecture 1 and the assignment.

Mark Wineberg

“ Why are the function pointers type declaration missing from the function signature?”

This was deliberate. The question is trying to make sure you understand how function pointers are used. There are three parts to getting a function pointer to work, the variable declaration, the type declaration in function signatures and its use in the function body. You need to understand how all three work and demonstrate your understanding by getting the code to work. Full examples of type declarations in signatures are given in the notes on function pointers.

Mark Wineberg

“I have been struggling with sorting my linked list for some time now, and I am looking for some clarification. When you say we must insert the node into our list in the sort order as well (for example- the push function), are we to have one instance, or two instances of the linked list? For LabA4 I had three separate instances, one to move through with next, one to move through with sort1, and sort2 etc. However, from my understanding it seems that for this we only have one instance of our list, and we must be able to move through it with the next, and the sort pointers. Though I am having second thoughts and am thinking we could do it similar to LabA4, with one instance ordered using next, and another instance ordered using sort, organized by ascending keys.

Please clarify the number of “instances” of our list we can use, because I am currently running into lots of problems arranging my sort pointers properly in the same list as my next pointers.”

I guess for you I was more clear in my descriptions in A4 than L4. If you look back at L4, you will see that the struct has all three links (next, sort1 and sort2) in the single structure. Thus a single node would have all links.

You do not have to have to have a new node for separate lists for each linkage chain. Just use the same node and update the three chains running through that node, each chain using the different link pointer.

That is why in L4, at the end, I asked the value to change and have you print out the three lists. The value should have changed for all lists, as it is the same node, just positioned differently in each of the lists.

So A4 is actually exactly like L4, except a bit simpler as there is only one key. I made it simpler because we are doing so much more with it.

Finally, you cannot keep the orders in separate "instances", i.e. separate chains, one for next and one for sort. You lose the interaction between the two paths through the data. Consequently, **you will not get the same results** if you do it that way. This becomes obvious with some of the examples in Q1b. Indeed, I created those examples to demonstrate just that type of interaction effect.

So, you should try to keep the different chains using the same node. Really it is not different from using different nodes, as a node is a node to the program. It does not differentiate.

Mark Wineberg