

CS 2028 C 002

LAB 8 REPORT

10/31/2023

KYLE RUSSELL, russek5

Overview:

The goal of Lab 8 is to investigate the workings of a linked list. The requirements had me create an ambiguous template for a linked list, then create a class to use the linked list with. Linked lists are powerful tools, as the nodes can be anywhere in memory and do not have to be in consecutive blocks. Each list node contains a pointer to its next and previous node in the list, as well as a pointer to the data that the node holds. It is important to use pointers as they assist with the memory management process and are essential to the function of linked lists.

Task 1

The creation of task 1 was rather simple: a doubly linked list in an ambiguous template. The trick was implementing the doubly linked list, as it requires a few more checks and steps to get working. When adding an item, it checks first if the head of the linked list doesn't exist. Adding a node there is fairly simple, you just set the head to the node and set the next and previous pointers to null. Then, while location exists (is not null), the loop checks the dereferenced item against the dereferenced location's data to see if it is less. If so, create a node with the item as the data, the location as the next pointer, and the location's previous pointer as its previous. Then, set the location's previous pointer's next pointer to the new node, location's previous to new node, and the node is added. Then, we can check if there is a new head by seeing the new node's previous pointer. If it points to null, it is a new head and we can set it as such. Then we return and end the function. If we reach the end of the function, the loop will break and set the node to the end of the list.

There are a few other functions here, namely the remove item function. This simply takes the location as input and sets its surrounding items to break it out of the list. It will retain its next and previous pointers, as those are inconsequential if they are cleared properly when the data is added back into the list.

See at, see next and see previous return their respective nodes if they exist. Get item uses a node to get an item out of the list, and it will return the location node where the found item is. Is in list follows the same principle, except it returns true or false if the item is found. The rest of the functions are self explanatory.

Task 2

Task 2 was to create a data type to use with the linked list. It will be a product with a product number that will have comparison overloads, a description, price, quantity and lead time to produce if quantity is zero. These all get constructed, and the lead time is randomized for fun. Getting the part info simply concatenates the part number with the description, and returns it as a string to be used elsewhere. Over all, very simple class.

Task 3

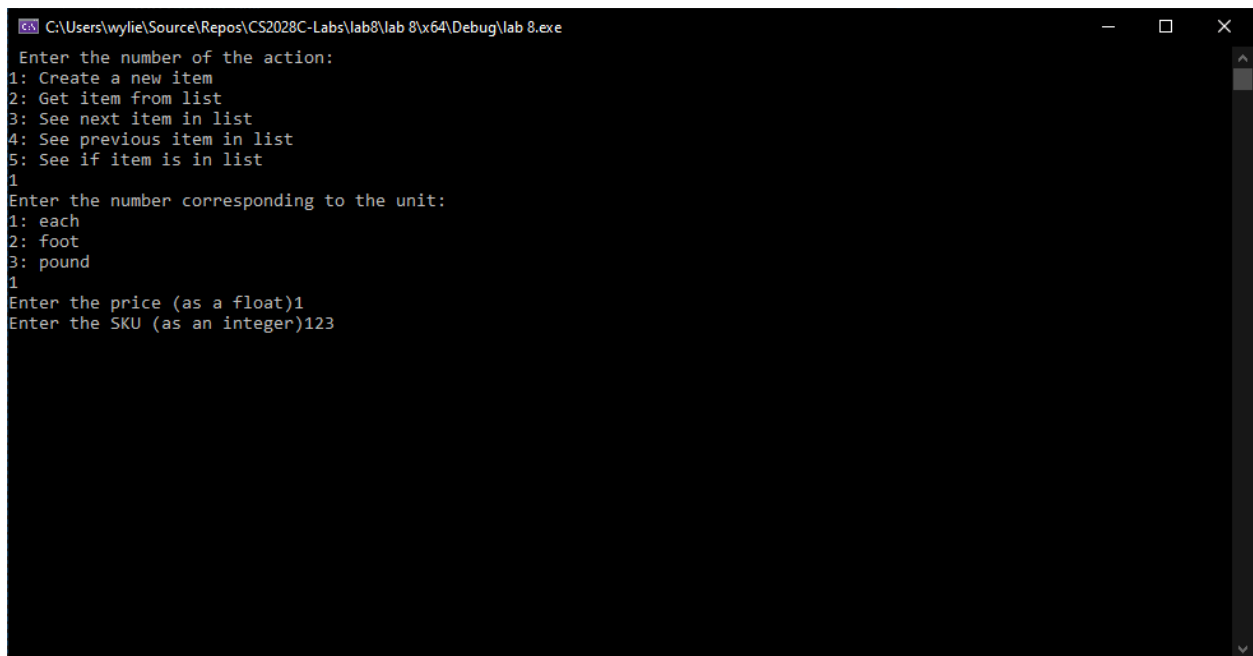
The main logic of the program consists of the following flow:

1. Select a list action
2. Execute the list action

A very simple task, as we just have to interface with the list and test whether our adding, removing and checking works as intended. Calling create, remove, get item, next item and previous item, we can conduct a fairly thorough test of how the list works.

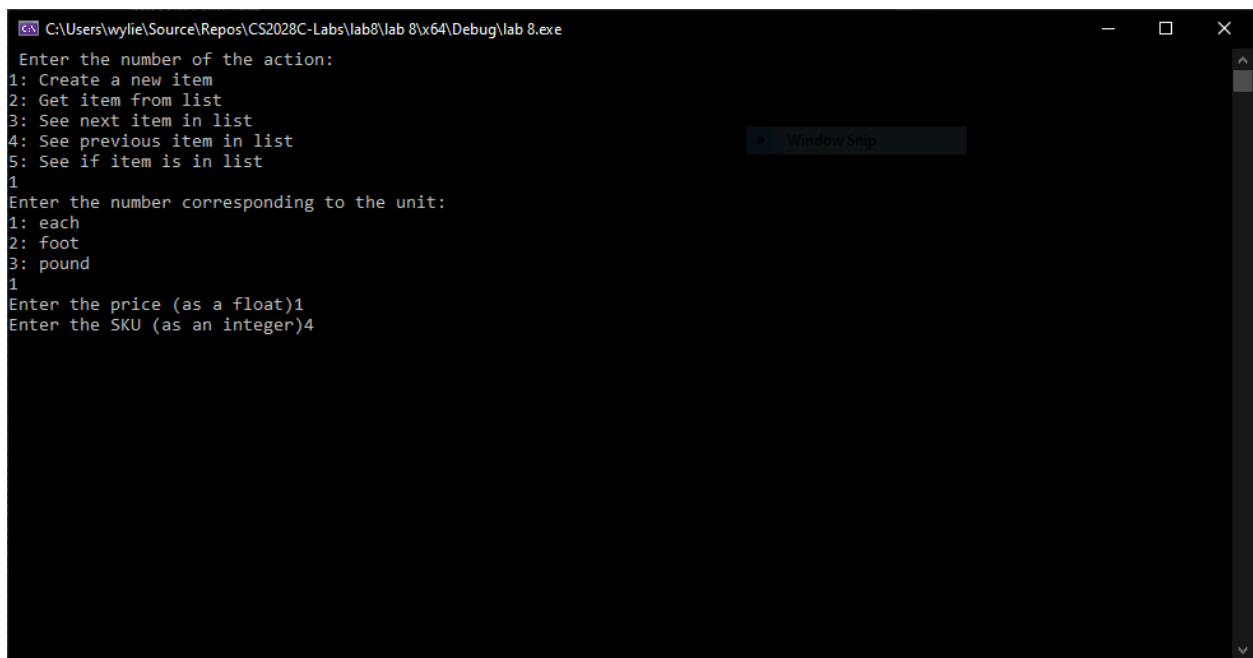
As of now, the only factor taken into account when sorting the list is the SKU, so that's all we really need to interface with the list.

The following figures and explanations are the results of my test:



```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
1
Enter the number corresponding to the unit:
1: each
2: foot
3: pound
1
Enter the price (as a float)1
Enter the SKU (as an integer)123
```

Figure 1: The first part inserted to the list



```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
1
Enter the number corresponding to the unit:
1: each
2: foot
3: pound
1
Enter the price (as a float)1
Enter the SKU (as an integer)4
```

Figure 2: The second part, should be the head

list.head_	0x00000249f5930db0 {data_=0x00000249f5926890 {sku_=4 description_=" default description" price_=" 123.00"}}
list.head_ -> next_	0x00000249f59312f0 {data_=0x00000249f592a5d0 {sku_=123 description_=" default description" price_=" 123.00"}}
list.head_ -> next_ -> data_	0x00000249f592a5d0 {sku_=123 description_=" default description" price_=" 123.00"}

Figure 3: We can see that the head is indeed 4, and the next is 123

list.head_	0x00000249f5930db0 {data_=0x00000249f5926890 {sku_=4 description_=" default description" price_=" 123.00"}}
list.head_ -> next_	0x00000249f59314d0 {data_=0x00000249f5947320 {sku_=100 description_=" default description" price_=" 123.00"}}
list.head_ -> next_ -> next_	0x00000249f59312f0 {data_=0x00000249f592a5d0 {sku_=123 description_=" default description" price_=" 123.00"}}

Figure 4: After inserting a part with SKU 100

list.head_	0x00000251bc240f60 {data_=0x00000251bc236840 {sku_=4 description_=" default description" price_=" 123.00"}}
list.head_ -> next_	0x00000251bc2411a0 {data_=0x00000251bc236c40 {sku_=100 description_=" default description" price_=" 123.00"}}
list.head_ -> next_ -> next_	0x00000251bc2412c0 {data_=0x00000251bc23a5d0 {sku_=123 description_=" default description" price_=" 123.00"}}
list.head_ -> next_ -> next_ -> next_	0x00000251bc240d80 {data_=0x00000251bc23c320 {sku_=200 description_=" default description" price_=" 123.00"}}

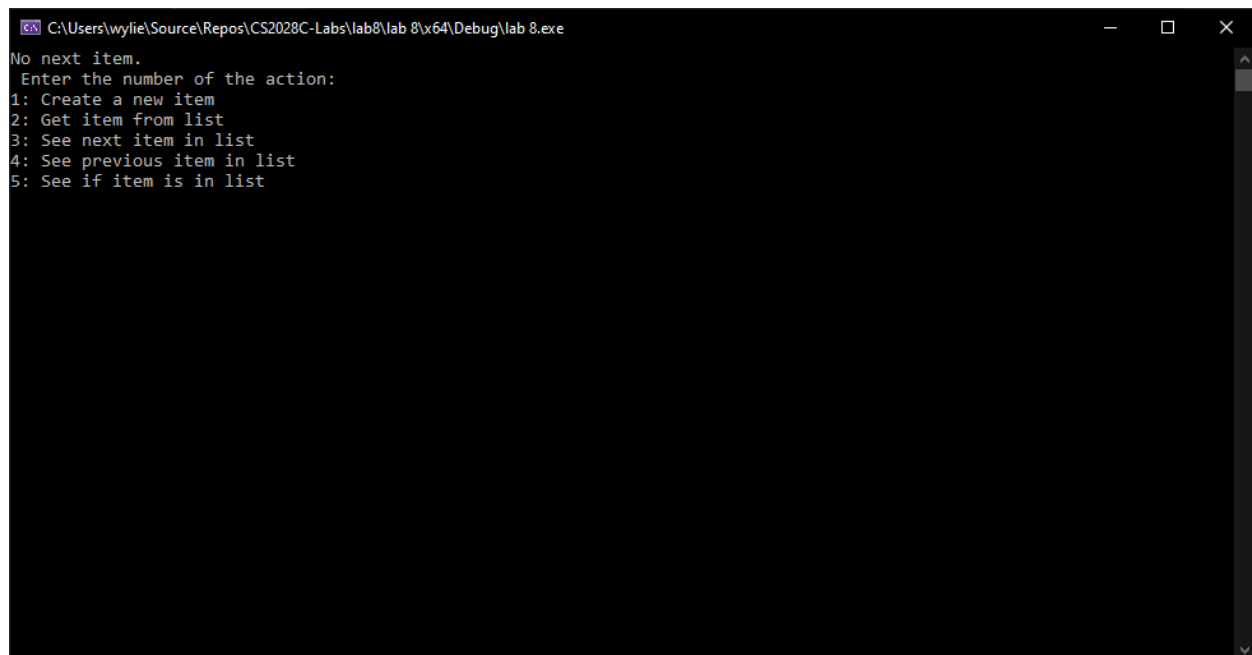
Figure 5: Finally, a part is added to the end

```

C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
123 default description Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list

```

Figure 6: Result of getting item from the list



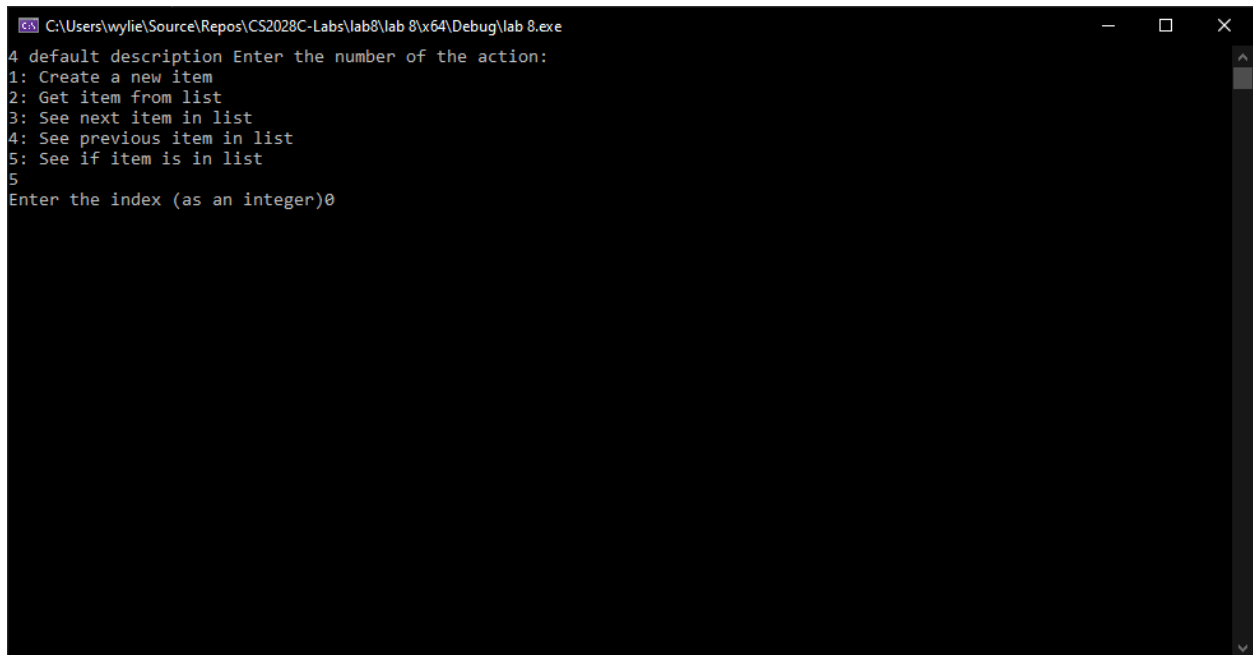
```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
No next item.
Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
```

Figure 7: The location variable points to the part with SKU 200, meaning it has no next.



```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
123 default description Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
```

Figure 8: Proved by previous being 123



```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
4 default description Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
5
Enter the index (as an integer)0
```

Figure 9: Entering index 0



```
C:\Users\wyllie\Source\Repos\CS2028C-Labs\lab8\lab 8\x64\Debug\lab 8.exe
4 default description Enter the number of the action:
1: Create a new item
2: Get item from list
3: See next item in list
4: See previous item in list
5: See if item is in list
5
4
```

Figure 10: Index 0 grabs the head, showing SKU 4

Conclusion

In conclusion, I have found that linked lists are fairly easy to use, and are also very intuitive. They are a great lesson on how pointers do and don't work in C++, and also are a good lesson in memory management.