Nick Hoffman and Kelvin Abrokwa-Johnson

Professor James Deverick

Computer Science 241

October 9 2015

**Project 2 Write Up**

This project uses a two class and one sub-class system to create a program that could create and modify sentinel linked lists. The main application of the program is to add two linked lists of functions together and create a new linked list containing the sum of the two. However, the program has more functions in addition to that. The road to creating the code was rocky, but in the end, through many test codes, we smoothed out the bugs so the program functions correctly.

**Issues in Coding**

*\_\_main\_\_ Function*

Initially, when we first ran the test code, the output of p3 came out as [ 3x^1990, 10x^1012, 3x^14, 5x^14, 11x^1, 1x^0], which stemmed from the remove node function and the index variable p3\_idx. When we changed the p3\_idx to p3\_idx+1 in this context after fixing the remove node function, the function returned the proper output for p3\_idx of [ 3x^1990, 10x^1012, 3x^14, 11x^1, 6x^0]

*Remove Node Function*

The remove node function initially failed to unlink the initial node from the list, which would result in an output that we see in the unsuccessful outcome of p3. By ensuring that the initial node is properly removed by removing the previous/next links to the respective nodes, the node is now successfully removed.

**Test Cases**

We made ensured that our test cases were rigorous and comprehensive. First and foremost we tested the constructor by ensuring that the list initiated with a length of 0. Without this assurance, iterating over this list in several other places in the code would fall apart. Secondly, we ensured that the class does not break down with invalid input by inserting and removing elements at invalid indices. Next, we ensured that the list always retained an appropriate length; that is, its length incremented by 1 whenever you successfully called `insert\_element\_at` or `append\_element` and decreased by 1 when you call `remove\_element\_at`. We also tested the `get\_element\_at` method by adding an element at a known index and then getting the element back and making sure it is the same value. We approached the tests for `append\_element` and `insert\_element\_at` similarly by adding elements at known indices and retrieving them to make sure they are what we expect.

In order to test the polynomials, we created an array with the expected values and then we added the the two polynomial linked lists, iterated over the new list’s entries, and compared them to the values in the expected array’s entries.

**Why is Node Reference Order Important?**

When referencing nodes, if the order is not what is perceived, the incorrect node could be removed, or a new node could be added in the wrong place. In order to maintain the correct order, we need to ensure that the pointers are duplicated before they are deleted. If the pointers are not, then the list will be abbreviated at the point where the node references were not correctly redirected. This means that certain nodes that you need to access will be garbage collected and won’t be available for future use. Hence, the best approach is to always ensure the new node references are created before the old are deleted, to ensure that the list does not become unintentionally shortened.

**Possible Applications**

Linked lists have a wide array of applications. Firstly, we can use them to store data much like a regular array, however they offer several advantages over arrays. They are able to store arbitrarily large sets of data without having to determine the size at compile time. For a language list python where the standard array is both a queue and a stack, this doesn’t necessarily offer advantages, but in lower level languages like C, linked lists offer a solution in situations where a programmer doesn’t necessarily how much data will go into the list. Appending and prepending to a linked list is also a much cheaper operation than appending to an array (assuming you store the length of the list internally).

On an abstract level, they can be used as the basis for other data types. With the methods we implemented specifically, this linked list can be easily used as a stack, and stacks have a wide array of applications (in memory management, for example). It can also be easily used as a queue or deque. Similar to the `Poly\_Val` class, we can combine the linked list with a hashing function to create a hash table.

**Conclusion**

After many tweaks, the program now has two successfully functioning classes and can complete the task of adding two linked lists together. With the code, we can apply a variety of applications on a single linked list, multiple lists, or between multiple lists.

We affirm that our submission conforms to the W&M Honor Code guidelines, and that all of our work on this project was completed collaboratively as a team.