ECE250 Project 0 Design Document- Linked Lists

Devon Miller-Junk 20763552 September 13, 2019

Project 0: Linked Lists

For the design of project 0, my general plan was to create classes in a way such that they can be applied in a variety of applications and used in a variety of ways. My goal was to create code that was readable, editable code so that programmers, myself included, could use the classes in the future to accomplish different goals: saving time down the road. I created a generic linked list class, that is a singly linked list class that stores doubles but can be easily modified to take full advantage of templates or other variable types. After having created the linked list class, I created a polynomial class that used the linked list class in its implementation.

**Note:** To compile and run my program please use the following:

g++ -std=c++11 -o polynomialtest polynomialtest.cpp

./polynomialtest <inputFile >outputFile

1. **Node Struct**

The node struct I created contains a double value and a pointer to the next node. These nodes will be linked together to form the linked list. With this implementation, the list is singly linked. I made the decision to make the list a singly linked list because doubly linking the list meant that each node would take up more space, and each operation dealing with changing the order of the list would take longer. With the operations implemented in the linked list class, it did not make sense to make it a doubly linked list.

1. **Linked List Class**

The linked list class creates a singly linked list of nodes that each contain a double and a pointer to the next node. The linked list itself contains a pointer to the start node, and an integer representing the size of the linked list. Storing the integer size of the linked list makes sense, because integer values take up very little space, but save a significant amount of time because the operation to get the size of a singly linked list takes Θ(n) time, whereas it takes Θ(1) time when storing the size as a value.

In this class, I have 2 constructors and one destructor. The two constructors are a default constructor, which defaults the size to 0 and the pointer to the null pointer, and a constructor which takes a size and array of double values. I chose these two constructors because I wanted a replacement to the default constructor, and I wanted a constructor that was appropriate for the input I was receiving from the file. The destructor I created removes nodes from the end of the list. This process takes Θ () time because it uses the pop function I created, which removes nodes from the back of the list. This decision was made to reduce complexity of the code since having multiple redundant functionalities will result in more errors and potential problem points.

The last functions I have for the linked list are a push, pop, getStart, and getSize. The push function takes a double and appends to the linked list a node whose value is equal to that of the parameter passed in. The pop function removes the last node from the list. The getStart function returns a pointer to the starting node of the array, which allows me to access the start of the array without making the start variable public, and thus editable. I wanted to keep the start variable private so that the pointer was not editable globally. The getSize function was implemented for the same reason, to keep the size value private while allowing access to view the size of the linked list.

1. **Polynomial Class**

The polynomial class I implemented represents an N degree polynomial with positive integer degree terms. The polynomial class represents the coefficients of the polynomial as doubles and stores them in a singly linked list implemented using the linked list class mentioned above. The coefficients are stored in ascending order (e.g. the constant value first), with the size of the linked list being one greater than the degree of the polynomial.

The polynomial class has 2 constructors and 1 destructor. There is a constructor with no parameters that defaults the linked list to an empty linked list. The second constructor takes an integer size and an array of doubles that represent the coefficients of the polynomial. No copy constructor was used because I overloaded the equality operator. The destructor simply calls the destructor for the linked list.

In the polynomial class I overload the single equals, double equals, addition and multiply operators. The single equals operator is used for ease of declaring polynomials as equal and receiving the input commands. These operators all take a polynomial by reference as a parameter. The doubly equals operator as well as the multiply and add operators were chosen to be overloaded due to the specifications in the project requirements. The equals and double equals operator each take Θ(n) time. The addition operator takes Θ(n+k) time where n is the size of the first polynomial, and k is the size of the second. The multiplication operator takes Θ(nk) time.

The last few functions used in the polynomial class are the evaluate, print, set and push functions. The print function is only used for developmental and testing purposes but was left in in case it was of use. The set function changes the value of a single term in the polynomial. It takes an integer index and a double value sets the value at the index to the double value. The evaluate function evaluates the polynomial at a given double value passed in and returns the result as a double. One interesting idea was to use Horner’s method for evaluating the polynomial, as it reduces the chance of overflow, but it was decided otherwise because the method would take Θ () time in a singly linked list. The push function appends a term onto the end of the polynomial, taking a double as a parameter.

1. **Other**

I also implemented a utility file to contain any common functions that I wanted across many different files. This utility file I can add to as the term goes along with other useful functions and constants that might be of use for certain projects and data structures. It currently only holds a double comparison function that allows for small errors that occur due to limited memory for double variables.

1. **References**

Alguindigue, T. (2019, September). *ECE 250 C Review Tutorial*. *ECE 250 C++ Review Tutorial*. Waterloo.

Ward, P. (2018, Fall). *ECE 150 Lectures*. *ECE 150 Lectures*. Waterloo.