Name: Thompson Pham

LinkedList.h

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
Class LinkedList
{
        private:
        Head (Node pointer) = nullptr;
        Size (int) = 0;

        public:
        //Default constructor
        LinkedList();
        // Overload Constructor
        LinkedList(Head (Node pointer), Size (int));
        // Destructor
        ~ LinkedList();
        // Accessors
        const *getHead () {return Head; }
        // Mutators
        Head *returnPtr(head ptr, int size)
        Void addEnd( head ptr, int size)
        Void sortList(head ptr)
};
#endif
```

LinkedList.cpp

```
# include <LinkedList.h>

LinkedList::LinkedList()
{
        Create new node
        Insert value into new node
        Make next ptr of new node nullptr
        if (head ptr  = nullptr){
                Head = new node
        }
        else{
```

```
        Make temp ptr, which points to head
        While loop to move to end of linked list
        Add new node to end of linked list
        }

}

LinkedList::returnPtr (head ptr, int size)
{
        Create ptr = head
        Ptr = list[size]
        if(Ptr == nullptr){
                Return nullptr;
        }
        return ptr;
}

LinkedList::addEnd(head ptr, int size)
{
        Node ptr = returnPtr(head ptr, int size)
        Create new node
        Insert value into new node
        Make next ptr of new node nullptr
        Head ptr += node ptr
        if(head ptr == nullptr){
        Head = newNode
        }else{
        Head -> next = newNode;
        }
}

LinkedList::sortList(head ptr)
{
bool flag = true;
      Node *ptr;
      Node *ptr2 = NULL;
      do{
      flag = false;
      ptr = head ptr;
         while(ptr -> next != ptr2){
```

```
            if(ptr -> numeral > ptr -> next -> numeral){
               Node *prevX = NULL, *currX = head ptr;
               while(currX -> numeral != ptr -> numeral){
                  prevX = currX;
                  currX = currX -> next;
                  }
               Node *prevY = NULL, *currY = head ptr;
               while(currY -> numeral != ptr -> next -> numeral){
                  prevY = currY;
                  currY = currY -> next;
               }
                  if (prevX != NULL){
                     prevX -> next = currY;
                  }else{
                     ptr = currY;
                  }
                  if (prevY != NULL){
                     prevY->next = currX;
                  }else{
                     ptr = currX;
                  }
                     Node* temp_swap = currY -> next;
                     currY -> next = currX -> next;
                     currX -> next = temp_swap;
                     ptr = currY;
                     flag = true;
            }
            ptr = ptr -> next;
         }
         ptr2 = ptr;
      }while(flag);

}
```

Main.cpp

```
#include <iostream>
#include <string>
```

```
using namespace std;

#include “LinkedList.h”
#include “Node.h”

Int main() {
        Function which reads from file, line by line where an expression will be inserted into a
        node(Void function - No parameters){
        While loop(read each line until end of file)
                Going to create a class for nodes (Void Function -  String line from file)
                Take line from file given from function above and insert each term into a node

                After a term is inserted into a node, it will call the linked list class

                Which will look at position in linked list is valid and return
                Then the node with the term will be added to the end of the list
                Then the linked list will be sorted by sort function down by exponent

                Function will look at linked list and calculate the polynomial (float function -
                LinkedList ptr)

                Function which will print the results (Void function - Linked list pointer, float
                calculated value)

                Delete function that will delete linked list for current line (Parameter - LinkedList
                ptr)

                // Move onto next line and repeat the process

        }
        Return 0;
}
```

Test Cases:

1) Polynomial with only a constant and a variable ex: F(14) = 5x
2) Polynomial with only a constant, variable, and exponent ex: F(14) = 5x^2
3) Polynomial with multiple terms and different exponents, check to see if function is sorted correctly
4) Function which only contains a constant value Ex F(5) = 18
5) Function were the polynomial is zero f(2) = 0

6) Function where the variable will be a float value Ex f(2.81) = 5x^2
7) Function where the polynomial contains only subtractions Ex f(2.81) = 5x^2 - 4x - 3x^4
8) Function where the polynomial contains only additions Ex f(2.81) = 5x^2 + 4x + 3x^4
9) Function where the polynomial contains both addition and subtraction
   Ex f(2.81) = 5x^2 + 4x - 3x^4
10) Function where the polynomial will result in a negative value