CE/CS 1337 – PROJECT 4 – Wario World Economics

Pseudocode Due: 10/31 by 11:59 PM (No late submission)

Final Code Due: 11/19 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in zyLabs.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
    - -Wall
    - -Wextra
    - -Wuninitialized
    - -pedantic-errors
    - -Wconversion
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objectives:

- Create multiple classes that interact with each other
- Implement a linked list with classes.
- Implement overloaded operators in a class

Problem: Wario is up to his old tricks again, trying to make as much money as possible through a new Mushroom Kingdom pyramid scheme. As any good financial analytic knows, the best way to maximize profits is through calculus (Specifically derivatives). Having no advanced math skills, Wario has hired you to create a program that will help him create derivatives necessary for his financial analysis.

Pseudocode: Your pseudocode should describe the following items

- Main.cpp
    - Detail the step-by-step logic of the main function
    - List other functions you plan to create
        - Determine the parameters
        - Determine the return type
        - Detail the step-by-step logic that the function will perform
- You do not need to write any pseudocode for any of the class functions
- Your pseudocode should describe how you are using the objects that will be created from the classes.

Details

- This project will be written in pieces
- This project must use two classes

- Linked List class
    - Node class
- Review homework #7 will focus on building the basic structure of the classes and testing them
- Review homework #8 will focus on adding the overloaded operators and testing them
- Once review homework #8 is complete, you will use the remaining time to create the main function that utilizes the classes you built and solves the problem
- Use proper naming conventions for the class accessors and mutators
- **Linked List class**
    - `LList.h` and `LList.cpp`
    - Variables
        - Head – node pointer
    - Functions
        - Default constructor
        - Overloaded Constructor
            - Make copy of list passed in
        - Destructor
            - Delete the list
        - Accessors and mutators
        - Overloaded operators – created in review homework #8
            - Overloaded [ ]
                - Return pointer to a constant node at the given index
            - Overloaded << operator
                - Display the linked list
                - Use [ ] notation to treat the linked list like an array
                - See output format below
            - Overloaded >> operator
                - Add node to head of linked list
        - Sort
            - Sort the linked list in descending order by exponent
- **Node class**
    - `Node.h` and `Node.cpp`
    - Variables (variable name in parentheses)
        - Outer coefficient (outer)
        - Inner coefficient (inner)
        - (optional) numerator and denominator variables if doing extra credit portion (num)/(denom)
        - Exponent (exp)
        - Trig identifier (trig)
        - Node pointer (next)
    - Functions
        - Default constructor
        - Overloaded constructor
        - Accessors and mutators
        - Overloaded << operator – created in review homework #8

        - Display a single node
        - Format: <outer>x^<exponent>
        - This function should not print the sign\
            - That should be handled in the LList << function
- Some of the functions required in the classes may not be used to solve the problem, but will be tested for the project grade
- All nodes will be dynamically created
    - There should only be enough nodes to hold data for the current expression
    - You will have to consider a way to reuse the linked list for the next expression
- All input will be read from a file
- Each term in the expression will be stored into a node and added into the linked list
- Each line in the file will be a mathematical function that can be derived
- The number of lines in the file is unknown
- Each calculated derivative will be written to a file

User Interface: There will be no user interface for this program

Input: All input will be read from a file. Prompt the user for the name of the input file. Each line in the file will be a mathematical function with the following parameters:

- Consist of polynomial terms - the highest degree will be 10
- May also contain trig functions
- Exponents will be represented by the ^ character.
- Exponents may be positive or negative
- Do not assume that the expression will be in order from highest to lowest exponent.
- All coefficients will be integers.
- The absence of a coefficient should be interpreted as a coefficient of 1
- Trigonometric functions may have coefficients
- Trigonometric functions will not be exponentiated
- The variable will always be ‘`x`’.
- There will be spaces around the operators between terms
- If a trig function is used, there will be a space between the trig function and the coefficient of x
- Each line will end with a newline with the exception of the last line
    - The last line may or may not have a newline
- **Example Input:**
    - `3x^2 + 2x + 1`
    - `x^-2 + 3x + 4`
    - `4x – x^3`
    - `3sin x + cos x`
    - `1 – cos 4x`
    - `3x^4 – 6x^2 + tan 10x`

Output: All output will be written to a file.

- The file will be named `derive.txt`.
- Each derivative will be written on a separate line.

- Use the '^' character to represent exponents.
- The terms of the derivative must be ordered from highest to lowest exponent
  - Trig functions should be listed at the end in the order they were encountered in the original expression
- The format of the output for each term will be the same as the input format
- Do not use double operators
  - Invalid format: `2x^2 + -3x`

EXTRA CREDIT: Implement the ability to derive functions with fractional coefficients (potential 25 extra points for this project)

- Fractional coefficients will be enclosed within parentheses (for both input and output)
- Each coefficient in the derivative should be simplified as much as possible
- Fractional coefficients with a denominator of 1 after derivation will be written as a whole number without parentheses
- **Example Input**
  - `x – (1/4)sin 4x`
  - `(3/5)x^5 – 2x^3 - 10cos 10x`