CS 2337 – PROJECT 3 – Recreating the Speed Force

Pseudocode Due: 3/27 by 11:59 PM

Core Implementation Due: 4/2 by 11:59 PM

Final Submission Due: 4/9 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.
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date.
    - zyLabs will provide you with an opportunity to see how well your project works against the test cases. Although you cannot see the actual test cases, a description will be provided for each test case that should help you understand where your program fails.
- **Type your name and netID in the comments at the top of all files submitted. (- 5 points)**

Objective: Use object-oriented programming to implement and utilize a binary tree

Problem: Harrison Wells is working to create an artificial speed force for Barry Allen (The Flash) so that he can recharge. The calculations for this task involve polynomial integration. A program is needed to develop and evaluate anti-derivatives. Since it is your first day at S.T.A.R. Labs, this task has been handed to you to complete.

Pseudocode: Your pseudocode should describe the following items

- Identify any functions you intend to create with the main function
    - For each function, identify the following
        - Determine the parameters
        - Determine the return type
        - Detail the step-by-step logic that the function will perform
- Describe the logic that the main function will perform
- If you plan to create additional functions in any of the classes that are not overridden from Object or an interface, please describe those in your pseudocode.

Core Implementation:

- Read the file
- Parse file with uniform spacing and each term has a coefficient
- Build nodes for each term
- Add nodes into tree
- Simplify fractions that represent whole numbers
- Print anti-derivative

- Evaluate definite integral
- Not required
    - Recursive traversals
    - Combining terms with same exponent
    - Parsing terms without a coefficient
    - Handling negative first term
    - Delete function
        - Will be tested in final submission with unit test
    - Parsing function with non-uniform spacing
    - Definite integrals where `a > b`
    - Reducing fractions

Details:

- Start the program by prompting the user for the input filename
    - This would normally be hardcoded in an application, but zyLabs requires a filename for multiple test files
- This program will read basic integrals from a file and evaluate them.
- If the integral is definite, the program will create the anti-derivative and evaluate for the given interval.
- If the integral is indefinite, then just the anti-derivative should be created.

Classes

- Use good programming practice for classes – proper variable access, mutators and accessors, proper constructors, etc.
- Remember that classes exist to be used by other people. Just because you don't use it in the program doesn't mean it shouldn't be coded and available for others to use in theirs.
- As with previous projects, you are open to design the classes as you see fit with the minimum requirements listed below
- All classes must be of your own design and implementation.
    - **Do not use Java's built-in Binary Tree class (-20 points if used)**
- **Requirements**
    - Binary Tree class
        - The file will be named **`BinTree.java`**
        - Root pointer
        - Will contain functions for basic functionality of a binary tree
            - Insert – void return type, Generic parameter
            - Search – Node return type, Generic parameter
            - Delete – Node return type, Generic parameter
        - May contain other functions if the functions are specific to a binary search tree
        - Any function traversing the tree (partially or in full) must be recursive (-10 points per function if not)
            - This includes functions that add, delete, or search the tree
        - The order of the nodes matters

        - A binary tree Is better than a linked list
    - Node class
        - The file will be named **`Node.java`**
        - Generic class (-10 points if not)
        - Left and right pointers
        - Generic reference variable to hold data
    - Payload class
        - The files will be named **`Payload.java`**
        - Coefficient
        - Exponent

Input: All expressions will be read from a file. There is no limit to the number of expressions that can be in the file. Each expression will be on a separate line (see examples below). There is no input validation.

Expressions

- Consist of simple polynomial terms - the highest degree will be 10
    - If multiple terms include the same exponent, combine those terms
    - If combined term has coefficient of zero – delete node from tree
    - There will be no terms with an exponent of -1
- Exponents will be represented by the ^ character.
- Do not assume that the expression will be in order from highest to lowest exponent.
- All coefficients will be integers.
- The | (pipe) character will be used to represent the integral symbol
- If an integral is definite, there will be a number before and after the | character
    - $\int_a^b x\,dx$ = `a|b x dx`
    - The endpoints of the interval for the definite integral will be integer values
    - Do not assume `a < b`
- The variable will always be 'x' and the integral will always end with `dx`
- There will always be a space before the first term of the integral and before `dx`
    - Do not assume there will be spaces anywhere else in the expression
- If a term has no coefficient, it is assumed to be 1
- **Example Input:**
    - `| 3x^2 + 2x + 1 dx`
    - `1|4 x^-2 + 3x + 4 dx`
    - `-2|2 x^3 - 4x dx`

Output:

- All output will be written to the console
- Each anti-derivative will be displayed to a separate line
    - Definite integrals will also include the interval and value (see examples below)
        - Values will be to 3 decimal places
- Use the ^ character to represent exponents
- Order the terms from greatest to least exponent

- Fractions will be simplified
    - All fractions will be enclosed in parentheses
- A space will proceed and follow each operator between terms (+ or -)
    - If the first term has a negative coefficient, do not add spaces around the negation
    - If the first term is a negative fraction, align the negation with the numerator inside the parentheses
- Do not display a term with a zero coefficient unless it is the only term in the expression
- Do not display a coefficient or exponent of 1
- Indefinite anti-derivative format
    - `<expression><space><plus><space><capital C><newline>`
- Definite anti-derivative format
    - `<expression><comma><space><upper bound><pipe><lower bound><space><equal><space><value><newline>`
- **Example Output (based on input above):**
    - `x^3 + x^2 + x + C`
    - `(3/2)x^2 + 4x - x^-1, 1|4 = 35.250`
    - `(1/4)x^4 – 2x^2, -2|2 = 0.000`

EXTRA CREDIT: Add to your program to allow evaluation of indefinite integrals containing trigonometric functions (potential 15 extra points)

- **The trig terms must be stored in the tree with all other nodes.**
- Simple expressions inside the trig functions
    - The term inside the trig function will not have an exponent
- The trig anti-derivatives will be listed at the end of the expression in the order encountered
- There will be a space between the trig function and the inner coefficient of the trig function
    - This is true for both input and output
- **Example Input:**
    - `| sin x + cos x dx`
    - `| 1 – cos 4x dx`
    - `| 3x^4 – 6x^2 + 2sin 10x dx`
- **Example Output:**
    - `-cos x + sin x + C`
    - `x – (1/4)sin 4x + C`
    - `(3/5)x^5 – 2x^3 – (1/5)cos 10x + C`