Project Three Pseudocode

Andy Nguyen

CS – 2336.003

Smith

Main Class

Ask Input File Name

Parameters: Nothing

Return: string containing the file name

- Call a scanner object
- Ask for a file to be inputted by the user and store input into string

Read File Line

Parameters: scanner object

Return: binary tree of polynomials

- Parse the “integral”/determine whether it is indefinite or definite and store the boundaries if the integral is definite into variables to be stored later
    - If indexOf “|” not equal to -1 and the index is not equal to zero
        - Create an integers which will store digits and negative operator for boundaries
        - Use Integer.parseInt left of the “|”, the part of the string from beginning of string to last index before “|”
        - Use a for loop to go right of “|” character until it encounters a white space and store the index of the space
        - Use Integer.parseInt right of the “|”, the part of string from right of the “|” to the index before the white space
        - Remove the boundaries from the string
    - While the indexOf “x^” is not equal to -1
        - Traverse to the left of the indexOf “x^” until it encounters an operator or is at the beginning of the equation, extracting only digits and subtraction operator and putting it into a new string.
        - Reverse the order of the new coefficient string

        - Traverse to the right of the indexOf “x^”, if there is a “-“ to the right of the indexOf “x^”, extract digits and “-“ until another operator is encountered. Else traverse until another operator is encountered, put the extracted values into a new coefficient string

        - Remove the term along with its leading operator if it has one from the file line string

        - Take the newly created coefficient and exponent string and turn them into integers

        - Store the integers into a term object and create generic node with term and **Insert Node into Tree**

    - While the indexOf “x” is not -1
        - Traverse to the left of the indexOf “x” until it encounters an operator or is at the beginning of the equation, extracting only digits and subtraction operator and putting it into a new string.

        - Reverse the order of the new coefficient string

        - Remove the term along with its leading operator if it has one from the file line string
        - Take the new exponent string and turn them into integer
        - Store the integers into a term object and create generic node with term and **Insert Node into Tree**

    - For loop which loops through remain file line string
        - If character at index is a “-“
            - Add to new string
        - Keep adding only digits to the new string until another operator is encountered
        - If another operator is encountered, take the term string and turn into integer
        - Store the integers into a term object and create generic node with term and **Insert Node into Tree**
        - Remove the term along with its leading operator if it has one from the file line string

Display Binary Trees

Parameters: array of binary search trees, parallel array which holds the boundaries of each integral if it has one

Return: Nothing

- For loop which loops through each binary tree
    - **Traverse In Order**
    - If the index of the BST array has boundaries
        - Traverse the tree within main and extract the double values from each term
        - Print out the result of definite integral
    - Else
        - Print out a constant

Main

Parameters: Nothing

Return: Nothing

- **Ask Input File Name**
- Create an array list of binary search trees for each equation in the input file
- If the input file from user exists go ahead
    - While scanner object has a next line
        - **Read File Line** and add returned binary search tree to array list
- Else
    - Tell the user that the file could not be opened

Binary Tree Class

Insert Node into Tree

Parameters: Node received node aka the root at the start of the recursion

Return: The root node

- If the root node is equal to null
    - Set root equal to the received node
    - Return the root
- Else if the value of the received node is less than value of the current node // Utilizing compareTo of the Comparable Interface
    - Set the left root of current node equal to **Insert Node into Tree** // Keep traversing the binary search tree recursively
- Else if the value of the received node is greater than value of the current node
    - Set the right root of current node equal to **Insert Node into Tree** // Keep traversing the binary search tree recursively
- Else if the value of the received node is equal to the value of the current node
    - Return the current node // Combining polynomials of with same exponent performed in term class
- Return the root

Search into Tree

Parameters: Generic datatype, in this case being a term object

Return: Boolean value of whether or not the datatype was found in the binary search tree

- If the root is equal to null return false

- Else **Search Recursively into Tree**

Search with Recursion into Tree

Parameters: A node used for traversal (which is usually the root, and the generic datatype desired term sought for

Return: Return the node it found

- If the current node's value is equal to the desired value of generic datatype or it is null
    - Return the current node
- If the current node's data value is less than what is sought for
    - Recursively return the function with the right child of current node as the parameter
- If the current node's data value is greater than what is sought for
    - Recursively return the function with left child of current node as the parameter

Delete into Tree

Parameters: Generic datatype, in this case being a term object

Return: Nothing

- Set the root equal to **Delete with Recursion into Tree**

Delete with Recursion into Tree

Parameters: A node used for traversal (which is usually the root, and the generic datatype desired term sought for

Return: Return the node it found

- If the root is equal to null, return the null root
- Else if the value of the received node is less than value of the current node // Utilizing compareTo of the Comparable Interface
    - Set the left root of current node equal to **Delete with Recursion into Tree**// Keep traversing the binary search tree recursively
- Else if the value of the received node is greater than value of the current node
    - Set the right root of current node equal to **Delete with Recursion into Tree** // Keep traversing the binary search tree recursively
- Else
-
    - If the current node or the "root"'s left child is null
        - Return the right child
    - Else if the current node or the "root"'s right child is null
        - Return the left child
    - Using the unique characteristics of a binary tree we can swap data of certain nodes to delete nodes with two children. Since the left most leaf always have the minimum value for BST

    - Set the current node’s term equal to the **Minimum Value** with the parameter being the current node’s right child
    - Set the current node’s right child equal to the **Delete with Recursion into Tree**
- Return the root

Minimum Value

Parameters: A node to start traversing from

Return: Return an integer which holds the minimum value

- Create an integer which is set equal to the data contained within the node
- While the current node’s left child is not equal to null
    - Set the minimum value integer equal to the data of left child of current node
    - Set the current node equal to the current node’s left child
- Return the minimum value, in this case would be a term with the smallest exponent

Traverse In Order

Parameter: A node used to help traverse the tree, in this case being the root of binary tree

Return:

- If the current node is equal to null
    - Return to break out of the recursion cause the tree is empty
- Call **Traverse In Order** with the parameter being the current node’s left node
- Print out formatted term of the node
- Call **Traverse In Order** with the parameter being the current node’s right node

Node Class

Compare To (Generic)

Parameters: Node that needs to be compared

Return: While returning an integer, call the Compare To function of comparable interfaces and compare the generic data within the node to the generic data from the received node

Term Class

- A Boolean variable which says whether or not the instance of the binary tree is definite or indefinite integral
- And integer variables holding the boundaries of a definite integral if the antiderivative requires it
- Integers holding the coefficient and the exponent of the term
- If there are boundaries, make sure to pass in the boundaries for each term upon creation

Compare To

Parameters: Term object which contains a defined variables within it

Return: An integer representative of whether the term is greater or less than the received term

- Node that needs to be compared
- Call the Compare To function of comparable interfaces and compare the generic data within the node to the generic data from the received node
- If the exponent from the received node is equal to the current node
    - Add the coefficient from the received node to the integer coefficient integer within the current term

Find Antiderivative

Parameters: Nothing

Return: Nothing

- Set a new integer exponent equal to the current exponent + 1
- Set a new integer coefficient equal to the current coefficient divided by the value of the new integer exponent.
- If the new current coefficient does result in a modulus of zero, then use double division and store the value into a double
- If there are values declared in the boundary variables
    - Evaluate the boundaries for the term

To String

Parameters: Nothing

Return: A string containing the term

- Returns a formatted string with the contents of the term like the coefficient, exponent, and the result of evaluated boundaries if the integral was definite

Test Cases

1) Check the programs output when there are multiple instances with the same exponent like “5x^2” + ”6x^2” (testing for combining terms)
2) Check the output of the program if an equation which does not have any polynomials containing x
3) Check the binary search tree’s ability to order the terms in order from highest exponent to to lowest and making sure it is following BST ordering
4) Make sure the output of the program can handle initials polynomials that start with a negative
5) Check the output of the terms when they result in fractional output. Make sure that it outputs the fraction and if the integral is definite, then make sure it holds the double value of the result so the output is correct
6) Check the output when the resulting integration has fractional output and ensure that the fractions outputted are simplified

7) Check the output of the program when an anti derivative contains like 30 terms
8) Check the binary search tree class for handling other data types such as integers to ensure that the class is true generic
9) Check that the insertion of nodes is correct by proper traversal display of of the anti derivative from the greatest exponent to least
10) Check the deletion of a node in a binary search tree and ensure that BST ordering is maintained especially if the deletion requires a node with two children.