Humza Salman

CS 2337.003

Professor Jason Smith

28 September 2019

**Project 3 Pseudocode**

* **Main**
  + **Main**
    - Retrieve input file name
    - Create binary tree of type Payload
    - Open input file
    - Check to see if opened
      * While not end of input file
        + String line
        + Bool definite = false
        + getLine(input, line);
        + int evaluateFrom, evaluateTo
        + evaluateFrom = line.charAt(0)
        + for (int i = 0; I < line.length; i++)

int exponent;

int coefficient;

if (line.charAt(i) == ‘|’ && evaluateFrom != ‘|’)

if true, evaluateTo = line.charAt(i+1);

set definite to true

else if(line.charAt(i) == ‘x’)

if the character before x is not blank

coefficient = line.charAt(i-1);

else

coefficient = 1

if ‘^’ after x

exponent = line.charAt(i+2);

Else if blank space after x

exponent = 1

else if(line.charAt(i) == ‘d’)

break

Payload\* p = new Payload(coefficient, exponent)

Node<Payload>\* node = new Node<Payload>(p);

Tree.insert(node)

* + - * + string expression = “”
        + double calculation = 0;
        + traverse through tree()

string expression = tree.evaluateIntegral();

if(definite)

tree.calculate(from, to)

* + - * + output to console according to format

highest order expression first (most right node of tree), and then the next lowest, and then the next lowest, etc.

Print the payload as noted above

If(definite)

Output setPrecision(3), fixed();

Output “,” + from + “|” + to = “ + calculation << endl;

Else

Output “ + C” << endl

* + - close input file
* **BinTree – template <typename T>**
  + **Members**
    - Header, Tail, size – all private
  + **Functions – (parameters) – return type**
  + **Insert –(parent, node) – void**
    - if (root == null)
      * root = node
    - else if (node data == parent data)
      * node data + parent data
    - else if (node data < parent data)
      * if (parent left == null)
        + parent left = node
      * else
        + insert(parent left, node)
    - else if (node data > parent data)
      * if (parent right == null)
        + parent right = node
      * else
        + insert(parent right, node)
  + **Search – (node, key) – bool**
    - If the node is not null
      * If node is the key
        + Return node
      * If node is less than key
        + Search (node, key)
      * If node is greater than key
        + Search (node, key)
  + **Delete – (parent, node) – T**
    - if (node == null)
      * return nullptr
    - if(root == null)
      * return root
    - if(node has both left and right child)
      * successor = node’s right
      * successor’s parent = node
      * while(successor left != null)
        + successor’s parent = node
        + successor = successorLeft
      * node data = successor data
      * delete(successorParent,successor
    - else if ( node is tree root)
      * if(node left is not null)
        + root = node left
      * else
        + root = node right
    - else if (node left is not null)
      * if(parent left is node)
        + parent left = node left
      * else
        + parent right = node left
    - else
      * if (parent left is node)
        + parent left = node right
      * else
        + parent right = node right
    - return node;
* **Node – template <typename T>**
  + **Members**
    - T data -- generic value to hold object --
    - Right and Left pointers – all private
* **Payload – (**Coefficient (numerator/denominator), exponent))
  + **Members**
    - Coefficient—Struct(numerator, denominator)--private
    - Exponent – private
    - Variable = x -- private
    - Definite, Indefinite, -- private
  + **Functions – parameters – return type**
    - Operator+ -- (Payload& p) -- void
      * Add this operator and the passed in coefficients
        + Add like a fraction
    - Operator== -- (Payload& p) – bool
      * Return this.exponent == p.exponent
    - Operator < -- (Payload& p) – bool
      * Return this.exponent < p.exponent
    - Operator > -- (Payload& p) – bool
      * Return this.exponent > p.exponent
    - evaluateIntegral() – std::string
      * coefficient = edivide coefficient by exponent +1
      * exponent = increment exponent
      * return coefficient + x + ^ + exponent
    - calculateIntegral – (int from, int to) – double
      * return [coefficient \* power function(to, exponent) ] – [coefficient \* power function(from, exponent) ]