|  |  |  |
| --- | --- | --- |
| American University of SharjahCollege of Engineering Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Lab Instructor:** Praveena Kolli  **Office:** EB2126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2020 |

**CMP 321 - Programming languages Laboratory**

**Lab 9**

**Objectives**

* Understand and develop Parse tree.
* Test parse tree with different notations.

In this lab, algebraic expressions consist of unary and binary operators applied to integer variables or constants. The possible binary operators are: **+** (addition), **-** (subtraction), **\*** (multiplication), **/** (division), and **^** (exponent). The possible unary operators are **-** (negation) and **#** (square root). Variable names consist of a single letter, either uppercase or lowercase, while constants are integer literals.  Algebraic expressions are stored as strings.

Three notations are available: infix, prefix, and postfix. (Note that each is typical of several programming languages e.g., C/C++/Python for infix, Lisp/Scheme for prefix, Forth/PostScript for postfix notations.)

For example, the average of two variables is written using infix notation as "(a+b)/2". Note that, for simplicity's sake, parentheses are to be used systematically to denote calculation order; spaces are optional. The same average in prefix notation is "(/ (+ a b) 2)". Parentheses and spaces are both mandatory.  In postfix notation the expression becomes ”a b + 2 /". There are no parentheses; spaces are mandatory.

Regardless of notation, the above three expressions compute the same result hence have the same parse tree. A parse tree can be concisely represented as a Python list of symbols, using prefix style again. The list representation of the parse tree for the above average is thus   ['/', ['+', 'a', 'b'], '2'].  A parse tree can also be "pretty printed" as a tree.

Another example would be the roots of the quadratic equation  *ax*² + *bx* + *c* = 0 , which are *x*1,2 = (-*b* ± √*b*² - 4*ac*) / 2*a.*

*Infix notation:* "((-b)+(# ((b\*b)-((4\*a)\*c))))/(2\*a)"

*Prefix notation:* "(/ (+ (- b) (# (- (\* b b) (\* 4 (\* a c))))) (\* 2 a))"

*Postfix notation:* "b - b b \* a c \* 4 \* - # + 2 a \* /"

*List representation*: ['/', ['+', ['-', 'b'], ['#', ['-', ['\*', 'b', 'b'], ['\*', '4', ['\*', 'a', 'c']]]]], ['\*', '2', 'a']]

Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Infix** | **Postfix** | **Prefix** | **Notes** |
| A \* B + C / D | A B \* C D / + | + \* A B / C D | multiply A and B, divide C by D, add the results |
| A \* (B + C) / D | A B C + \* D / | / \* A + B C D | add B and C, multiply by A, divide by D |
| A \* (B + C / D) | A B C D / + \* | \* A + B / C D | divide C by D, add B, multiply by A |

## Converting between these notations

|  |  |  |
| --- | --- | --- |
| **Infix** | **Postfix** | **Prefix** |
| ( (A \* B) + (C / D) ) | ( (A B \*) (C D /) +) | (+ (\* A B) (/ C D) ) |
| ((A \* (B + C) ) / D) | ( (A (B C +) \*) D /) | (/ (\* A (+ B C) ) D) |
| (A \* (B + (C / D) ) ) | (A (B (C D /) +) \*) | (\* A (+ B (/ C D) ) ) |

First, you should code a **Parse Tree** class, which implements a simple binary tree. Code all necessary class functions, and only those.

1. Write three functions that create a parse tree out of a given expression string:

fromInfix(), fromPrefix(), and fromPostfix().

1. Write three functions that create from the parse tree of the three expression strings:

infix(), prefix(), and postfix().

1. Write a function PrettyPrint that prints the tree as a tree structure.

Make sure your functions raise an exception if a string contains invalid expression syntax.

So for example,

print(ParseTree().fromInfix("(a+b)/2")) should pretty print the parse tree for this expression.

list(ParseTree().fromPostfix("x y ^") should return the list ['^', 'x', 'y']

ParseTree().fromInfix("(a+b)/2").postfix()should return the string  "a b + 2 /".

Note also that, for instance, the expression below should produce the same postfix expression xs that is given as input:

ParseTree().fromInfix(*ParseTree().*fromPostfix( xs).infix()).postfix()

For simplicity sake consider the below conditions for infix, prefix and postfix input strings,

* Infix: parentheses are to be used systematically to denote calculation order; spaces are optional.
* Prefix: Parentheses and spaces are both mandatory.
* Postfix: There are no parentheses; spaces are mandatory.

You may continue the given ParseTree class:

class ParseTree:

    class Node:  
        def \_\_init\_\_(self, data, left = None, right = None):

            self.data, self.left, self.right = data, left, right  
        def \_\_str\_\_(self):

            return self.data

    def \_\_init\_\_(self, root = None):

        self.root = root

    def \_\_str\_\_(self):

         pass # to be implemented

    def fromPostfix(self, expression=""):

         pass # ...