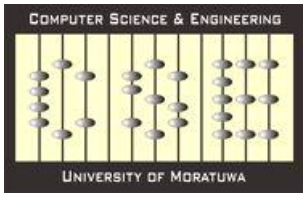


CS1033 Programming Fundamentals

Laboratory Exercise 9

BSc Engineering - Semester 1 (2020 Intake)

August – December 2021

Department of Computer Science and Engineering

Faculty of Engineering

University of Moratuwa

Sri Lanka

Lab 9: Application of Data Structures

Remarks

- **Exercise L9.E1** in this lab session will be **evaluated**.
- All the source codes related to Exercise L9.E1 should be submitted to the respective lab activity on Moodle.

Prerequisites

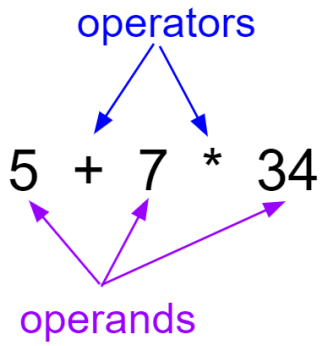
- This lab is based on lectures 8, 9, and 10. Students are expected to be familiar with the lecture content.

L9.1 Practice Programs

No special practice programs are provided.

L9.2 Exercises

Exercise L9.E1 **-** You are required to implement a simple symbolic equation solver. The equation should be stored in a **binary tree**. An equation consists of operands and operators.



Each operand or operator should be stored as a **tuple** of the form (TYPE, VALUE). Examples are (OPERAND, 5), (OPERAND, 7), (OPERAND, 34), (OPERATOR, '+') or (OPERATOR, '*').

Following operators should be supported: addition (+), subtraction (-), multiplication (*), and exponentiation (^). Grouping of terms in the equation using brackets should be supported. However, nested brackets need not be supported. For example, the expression "1 + (2 * 3) + 3 ^ 2" should be supported but the expression "1 + ((2 * 3) + 3) ^ 2" need not be supported. In addition, the expressions will have a maximum of only one operator inside the brackets. For example, you do not have to consider expressions such as "1 + (2 * 3 + 7) + 3 ^ 2" which contains two operators '*' and '+' inside the brackets.

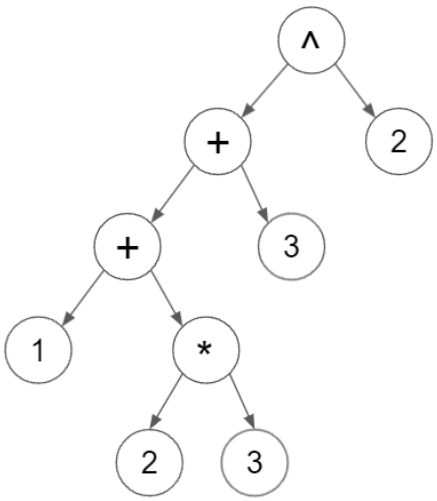
Evaluation of terms should be done **left-to-right** subjected to precedence given by brackets. (i.e. all the operators have equal precedence except the brackets). For example, the expression "1 + (2 * 3) + 3 ^ 2" will result in 100.

Explanation:

Total = 1
Total = 1 + (2*3) = 7 *(2*3 is evaluated first since they are within brackets)*
Total = 7 + 3 = 10
Total = 10 ^ 2 = 100

Note that these are not our usual arithmetic operations where we follow the BODMAS order.

Hint: Sample binary tree for the expression "1 + (2 * 3) + 3 ^ 2"



A basic template for your program is provided below.

```
class Node:

    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    def get_output(self):
        '''
        Print the output depending on the evaluated value.
        If the 0 <= value <= 999 the value is printed.
        If the value < 0, UNDERFLOW is printed.
        If the value > 999, OVERFLOW is printed.

        :return: None
        '''
        value = self.evaluate()
        if value > 999:
            print('OVERFLOW')
        elif value < 0:
            print('UNDERFLOW')
        else:
            print(value)

    ########################################################################
    ######### Your task is to implement the following methods. ##########
    ########################################################################
```

```
    def insert(self, data, bracketed):
        '''
        Insert operators and operands into the binary tree.

        :param data: Operator or operand as a tuple. E.g.: ('OPERAND', 34),
            ('OPERATOR', '+')
        :param bracketed: denote whether an operator is inside brackets or
            not. If the operator is inside brackets, we set bracketed as
            True.
        :return: self
        '''
        return self

    def evaluate(self):
        '''
        Process the expression stored in the binary tree and compute the final
            result.
        To do that, the function should be able to traverse the binary tree.

        Note that the evaluate function does not check for overflow or
            underflow.

        :return: the evaluated value
        '''
        pass
```

This template is available as `lab9_template.py` in the Lab 9 folder on Moodle.

Your task is to complete the body of the functions, `insert` and `evaluate`.

- The parameter `data` will be used to store the tuple (e.g. (OPERAND, 34), (OPERATOR, '+')).
- The insert function is used to insert a node into the binary tree. The parameter `bracketed` in the `insert` function is used to denote whether an operator is inside brackets or not. If the operator is inside brackets, we set `bracketed` as True (see the examples provided at the end).
- The `evaluate` function should be able to process the expression stored in the binary tree and compute the final result. To do that, the function should be able to traverse the binary tree. Observe how the `evaluate` function is used inside the `get_ouput` function. For example, if the expression stored in the binary tree is "1 + (2 * 3) + 3 ^ 2", the `evaluate` function should return 100.
- The `get_output` function returns the final result (i.e., the output of the `evaluate` function) if it is in the range [0, 999]. If the final result is less than 0, it returns UNDERFLOW and if the final result is greater than 999, it returns OVERFLOW.
- You can create and use additional functions as required. However, you are supposed to form the tree using the `insert` function and evaluate the result using the `evaluate` function.

The following examples will give you additional information about how the functions are expected to behave. The test cases for this lab exercise are also formed similarly.

Expression: 1 + (2 * 3) + 3 ^ 2

```
# Use the insert method to add nodes
# Begin with forming the root node for the tree.
root = Node(('OPERAND', 1))
# Form the rest of the tree by inserting data to the root node.
root = root.insert(('OPERATOR', '+'), False)
root = root.insert(('OPERAND', 2), False)
root = root.insert(('OPERATOR', '*'), True)
root = root.insert(('OPERAND', 3), False)
root = root.insert(('OPERATOR', '+'), False)
root = root.insert(('OPERAND', 3), False)
root = root.insert(('OPERATOR', '^'), False)
root = root.insert(('OPERAND', 2), False)
# Get the output.
root.get_output()
# Should print 100
```

Expression: 1 + 2 * 3 + 3 ^ 2

```
root = Node(('OPERAND', 1))
root = root.insert(('OPERATOR', '+'), False)
root = root.insert(('OPERAND', 2), False)
root = root.insert(('OPERATOR', '*'), False)
root = root.insert(('OPERAND', 3), False)
root = root.insert(('OPERATOR', '+'), False)
root = root.insert(('OPERAND', 3), False)
root = root.insert(('OPERATOR', '^'), False)
root = root.insert(('OPERAND', 2), False)

root.get_output()
# Should print 144
```

Expression: -15 - (99 * 10)

```
root = Node(('OPERAND',-15))
root = root.insert(('OPERATOR','-'),False)
root = root.insert(('OPERAND',99),False)
root = root.insert(('OPERATOR','*'),True)
root = root.insert(('OPERAND',10),False)

root.get_output()
# Should print UNDERFLOW
```