Lab 10

COMP9021, Session 2, 2015

1 ☞ Building a general tree

Consider a file named `tree.txt` containing numbers organised as a tree, a number at a depth of N in the tree being preceded with N tabs in the file. The file can also contain any number of lines with nothing but blank lines. Using the module `general_tree.py`, write a program that reads the contents of the file. If the file does not contain a proper representation of a tree then the program outputs an error message; otherwise, it builds the tree (an instance of GeneralTree()) and prints it out using the same representation as in the file (except for the possible blank lines of course). Here is a possible interaction:

```
$ python
...
$ cat tree.txt

2
        3
                1
        4




                5
                        7
                                8
                9

                        10
                        11
                        12



        6
1
$ python exercise_1.py
tree.txt does not contain the correct representation of a tree.
```

```
$ cat tree.txt

2
        3
                1
        4




                5
                                7
                                8
                9

        6
$ python exercise_1.py
tree.txt does not contain the correct representation of a tree.
$ cat tree.txt

2
        3
                1
        4




                5
                        7
                                8
                9

                        10
                        11
                        12



        6
$ python exercise_1.py
2
        3
                1
        4
                5
                        7
                                8
                9
                        10
                        11
                        12
        6
$
```

2 Back to fully parenthesised expressions

Modify the second exercise from Lab 9, that deals with arithmetic expressions written in infix, fully parenthesised, and built from natural numbers using the binary +, -, * and / operators, still using a stack but to build an expression tree rather than to evaluate the expression (that is, representing an expression of the form `(first_argument operator second_argument)` as a tree whose value is `operator`, and whose left and right nodes are the subtrees that represent `first_argument` and `second_argument`, respectively. The function `evaluate()` is then reimplemented so as to recursively evaluate the expression from the tree. Interacting with this program is exactly as with the program from Lab 9.