Report

Contents

Basic Design For The Application

Description

The application can change expression to binary tree and print it as readable image on terminal. In addition, it also can report the illegal expression and reload history expressions. The application's main file is ExpressionToBinaryTree.py, and it need run with python3.x. With more details, the application also has a unit test program.

After the assessment's deadline, I will open source the code repository in my Github, the repository's url is FaterYU/ExpressionToTree: A homework of OOP (github.com) . That means my code will public after **23:00 (GMT+8) SCNU Time on Friday 27th May**

Task Comment

Task 1

Instantiate an object from Expression's Class and use input function to read expression from shell which is user type and save it in a private variable which is a string type. Use eval function to calculated the result of expression and print it to the terminal.

Task 2

Firstly, change expression to Inverse Polish expression. With task's meaning, I need to create a binary tree from expression and Inverse Polish expression is helpful for changing, so split all the character in expression variable and make them as a Inverse Polish expression.

Then, Inverse Polish expression can change to Binary Tree easily with Stack.

At last, if I want to print the tree readable on the screen, I need to decide where I want to print, terminal or GUI. Considering the compatibility of GUI or Qt execution in Linux, Windows and MacOS, printing it on terminal is a sensible decision. The example show the printing tree with tipping 90 degrees to the right, which is in task's description. I think if I print it vertical on the terminal, it will be understand more easily than printing horizontal. When I need to print it, I need to insert some space and line break into the tree's string. The amount of space is decided with the height of node, it can draw a conclusion that the amount of space in each line is geometric progression about height of node.

Task 3

At last task, printing the expression as a binary tree is solved. If I want to save the history expression, saving the origin expression is better than saving the tree. Thus, it just use a text file to save history expression in each line. When restart the program, user can choose the reload option to reload the expression and print it as a binary tree on terminal from the saving file.

Task 4

The task meaning is checking the expression and reporting why the expression is not valid. Because every expression is a string, I can checking them by using regular expression. That means if the expression match the wrong regex, I can say it is ' Not a valid expression, wrong number of operands'.

Task 5

By using the package of unittest, I need to write a class inheriting TestCase. The test unit need to include every category of the input.

Task 6

The code of expression program is easy to read because every function have annotation and every function's name describe the effect of itself.

There is a 'README' comment section explain how to start the program and test. In addition, it also include the file structure and file interpretation to help user to understand the file function.

All code includes no dead code. There are 3 packages of code which are Stack, Node, Expression. The main program import those packages to realize the function.

Description Of The Algorithmic Choices

Data Structures

Changing expression to Inverse Polish expression is use the Stack, because I need to distinguish each pair of brackets and change the sub expression to the suffix expression. For each character, if it is left bracket, push it into the symbol stack. If it is number, push it into the expression stack. If it is calculate symbol, push it into the symbol stack. If it is right bracket, pop and push the element of the symbol stack one by one till pop left bracket.

Then, changing Inverse Polish expression to binary tree also use the Stack. For each character in Inverse Polish expression, if it is a number, push it into tree stack. If it is calculate symbol, set it to the node's value, pop one element from tree stack as the right child and pop one element from tree stack as the left child, and push this node into tree stack.

At last, printing binary tree on terminal need insert some space. Get the height of the tree and conclude two geometric progression to decide the matrix's width and height. Then insert the space according to the matrix's width and height and printing it on terminal.

Width of height i is

$$2^{i^{th}} \times 3 - 1$$

Height of height i is

$$2^{i^{th}-1} \times 3$$

I use regular expression to distinguish the valid expression. For 'Not a valid expression, wrong number of operands', I use `[\+\-\*\/][0-9][\+\-\*\/]+|\([^\+\-\*\/]\)|[\+\-\*\/]\([^\(\)]+\)[\+\-\*\/]` to match the expression of wrong number of operands. For 'Not a valid expression, brackets mismatched', I use `[\(\)]` to find all brackets in the expression. Then, distinguish the error with the amount of brackets and the first bracket and the last bracket. For 'Not a valid expression, operator missing', I use `[0-9]\(|\)[0-9]` to match the expression of operator missing. For 'Not a valid expression, expression contains invalid characters', I use `[^0-9\+\-\*\/\(\)]` to match the expression of contains invalid characters. For 'Not a valid expression, expression should not be empty', it can be distinguish easily with check the length of expression.

Saving history expression in text file each line. The program can reload the history expression from text file and print it as a binary tree easily.

Classes

I set 3 package of the program, stack, node and expression.

In stack class, it has 4 public functions.

- push
- top
- pop
- size

In node class, it has 7 public functions.

- get value
- get left child
- get right child
- set left child
- set right child
- get all node as list
- get tree's height

  In expression, it has 2 public functions and 6 private functions.

- check the expression
- draw tree image
- save expression
- reload expression
- read expression
- change expression to binary tree
- change binary tree to string matrix
- print the string matrix on terminal

Test

Inherit a class from `unittest.TestCase` and design 8 test function for each situation. By using mock's patch to decorate the test functions in order to input the test expression to the program.

- test normal input
- test wrong number input
- test operator missing input
- test brackets mismatched input
- test both of input error
- test normal reload
- test other type input
- test empty input

Reference List

No reference list