JC1503 Object-Oriented Programming 2021-2022

Assessment 2: Expression Binary Tree

Important

This assessment is worth 30% of the overall marks for course JC1503 Object-Oriented Programming.

This assessment has two components, Code and Report; the Code element counts for 40 marks, and the Report element for 60 marks.

You must **submit your code and report before 23:00 (GMT+8) SCNU Time on Friday 27th May**. Make sure your solution is ready to run before then. You must submit the **pdf** file of your report to the folder named "Assessment 2 Report Submission". You must compress to **zip** format your code files and upload the zip file to the folder named "Assessment 2 Code Submission".

The markers will look at your code and will visually check its output. If the code does not run, or the output is significantly different from that expected, you will get a much lower mark.

Remember that as this is an individual assessment, the work submitted must be your own.

If you have not already done so, you should familiarise yourself with the University guidance on plagiarism, available at https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

If you reuse other people's code or libraries, you must explicitly mention this in your code and acknowledge their work. This includes:

I. Where did you acquire them (e.g., URLs, books, GitHub)?

II. Who are the authors? They cannot be your classmates, of course!!

III. How did you modify them? In particular, if you make minor changes to a piece of other people's code, you must clearly mark in your code which part is your own code.

Introduction:

In Q1 of JC1503 Practical on Trees (Practical 10), you were given the following expression:

"(((5+2) *(2−1))/((2+9)+((7−2)−1)) *8) "

and were asked to convert it to a binary tree.

In this assessment, you are asked to design a python program that can automatically convert a mathematical expression into a binary tree after the user enters it as an input. Specifically, the expression must be in the same format as the above, and a valid expression can be recursively defined as follows:

It must be in the form of (X?Y) where X and Y are either numbers or valid expressions, and ? stands for operators (*, /, +, -)

For instance,

(4*5) is a valid expression;

((2*3)*5) and ((2*4)*(5*6)) are also valid expressions;

(4*5*6) is not a valid expression because it has three operands within one pair of brackets; ((4*(5+6) is not a valid expression as the brackets are mismatched (final bracket missing).

To simplify the problem, we assume all numbers are single-digit ones (i.e., ranging from 0 to 9) and all operators are: *, /, +, -

Hint: You may want to use a combination of data structures, for instance, stacks and trees.

Tests

Writing test for your code is a good practice. Therefore, some unit tests should be included in your code to show their use, and to confirm that your data structures hold expected values.

Hint: Test the data structures, not output to the screen, as unit testing console output is challenging

Marking Criteria for Code:

1. If your program runs, then that is 4 points, and if it runs but only does some parsing, then that is 6 points. If it both runs, and evaluates the input as a valid expression, then that is 8 points. [8pts]

   For example, given the input string: (((2*(3+2))+5)/2)

   The calculated result will be: 7.5

   Note that this example is for illustrative purposes, and during the marking process a different string will be used to test your solution.

2. Your program can visualise the generated binary tree with output in the terminal. If it has some buggy code for the traversal, then this is 4 points, and if it works fine, then this is 6 points. [6pts]

   The expected output tree for the input using an inorder traversal as follows (you can print it on the screen, and you don't need to print the lines connecting the tree nodes if it is not convenient):

   You can "read" the tree if you imagine it tipped 90 degrees to the right. This makes " / " the root of the tree, and unlike the lectures, we read The tree from right to left.

3. Your program can save the binary tree into a file. After you quit your program, you can restart your program and reload the tree from the same file into memory and visualise it again. If you can save the tree to a file, then this is 4 points. If you can save, and read the file

back in, then this is 6 points. [6pts]

Hint: You may consider using the `pickle` package for serialisation. You may also use other packages, or you could write your own functions for serialisation.

4. Your program can additionally report why the expression is not valid. If you have some reporting on in-valid expressions, then that is 4 points. If you have a more extensive list for reporting, such as below, then that is 6 points. [6pts]

(4*3*2)	Not a valid expression, wrong number of operands.
(4*(2))	Not a valid expression, wrong number of operands.
(4*(3+2)*(2+1))	Not a valid expression, wrong number of operands.
(2*4)*(3+2)	Not a valid expression, brackets mismatched.
((2+3)*(4*5)	Not a valid expression, brackets mismatched.
(2+5)*(4/(2+2)))	Not a valid expression, bracket mismatched.
(((2+3)*(4*5))+(1(2+3)))	Not a valid expression, operator missing.

5. You include some unit tests for your code. If there are working tests, then this is 4 points. If the tests match the criteria below, then that is 6 points. [6pts]

Test names are explanatory, with explicit tests, and it makes sense that they are there.

6. Sensible coding rules are followed. There should be two points for each item below. [8pt]

You should follow suitable coding conventions:

- The code is easy to read; self-describing method names, which show intent, with short methods, and clear control flow.
- The comments say why you are doing something and include references if appropriate.
- Your code is maintainable and includes no dead code, which is never reached, or used, and is modularized in an understandable manner.
- There is a ‘readme’ comment section at the top of the file explaining how to run the app (and any tests), plus any other relevant information.

Report:

The pdf report to be uploaded to Assessment 2 should contain:

a. A contents page

b. A basic design for the application (600 words) including:

i. A written description of the application.

ii. Comment about the implementation of all the six tasks.

c. A description of the algorithmic choices you made for the application (600 words) including:

i. Justification of selecting and implementing particular and/or specific Data Structures for your program/application.

ii. Description and justification of the Classes/Sub Classes you used for your program/application.

iii. Provide snippets of the unit tests or other tests you used in your program/application. Describe and comment on the outputs of those tests.

d. A Reference list showing items you have used in your learning that are correctly cited in the body

of the report

Marking scheme for the report:

1) A basic report which explains the operation of the application as well as the selection of Data Structures and programming choices, but does not go into detail regarding the structure and/or the choices and/or unit testing made. The report may not contain the correct sections. [Pass]

2) The report explains the operation and structure of the application. The selection of Data Structures, programming choices and unit testing used in the code are assessed and discussed briefly. The report is correctly formatted. [Good]

3) The report details the operation and structure of the application. The selection of Data Structures, programming choices and unit testing used in the code are explained and justified in their selection. The report is correctly formatted. [Very Good]

4) The report fully details the operation and structure of the application. The selection of Data Structures, programming choices and unit testing used in the application are fully explained and assessed for their use in this application. The report is correctly formatted. Greater marks can be achieved in this section by the comprehensiveness of the report. [Excellent]

Final Notes:

Remember that as this is an individual assessment, the work submitted must be your own. If you have not already done so, you should familiarise yourself with the University guidance on plagiarism, available at https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

1. Your code will be further checked by a software plagiarism checker. If plagiarism exists, your mark may be changed to a lower one or even zero. You will be informed by the Course Coordinator if this happens.
2. If you re-use other people's code or libraries, you must explicitly mention this in your code and acknowledge their work. This includes:
   a. Where did you acquire them (e.g., URLs, books, GitHub)?
   b. Who are the authors? They cannot be your classmates, of course!!
   c. How did you modify them? In particular, if you make minor changes to a piece of other people's code, you must clearly mark in your code which part is your own code and which is theirs.

See the following guide for Python documentation:

http://docs.python-guide.org/en/latest/writing/documentation/