MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 111

Fall 2022

Take-Home Exam 4

REGULATIONS

Due date: 23:59, 23 January 2023, Monday *(Not subject to postpone)*

Submission: Electronically. You should save your program source code as a text file named `the4.py`, and submit this file via the ODTUCLASS page of the course.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

In this THE, we will construct trees representing calculations as defined by a set of functions of a fixed variable '`x`'. Function definitions will conform to the following rules:

```
<function-def>  ==> <function> '(' <variable> ')' '=' <term> <operator> <term>
<term>          ==> <variable> | <constant> | <function> '(' <variable> ')'
<function>      ==> 'a' | 'b' | .. | 'z'           # excluding x
<variable>      ==> 'x'
<constant>      ==> '1' | '2' | .. | '100'
<operator>      ==> '+' | '-' | '*' | '^'
```

where a vertical bar ('`|`') separates alternatives; and `'+'`, `'-'`, `'*'`, `'^'` respectively denote addition, subtraction, multiplication and exponentiation. For exponentiation, you can assume that either the base or the exponent `<term>` is a `<constant>`. A "running" example set of functions conforming to these rules is the following:

```
f(x) = g(x) * x
g(x) = x + 20
h(x) = x * 100
```

PROBLEM & SPECIFICATIONS

- In this THE, we expect you to write a function named `construct_forest` with the following parameter and return values:
  - Parameter: `[function-def1, function-def2, ...]`
    - Description: A list of strings where each string defines a function.
    - Example: `["g(x) = x + 20", "h(x) = x * 100", "f(x) = g(x) * x"]`
  - Return: `[Tree1, Tree2, ...]`

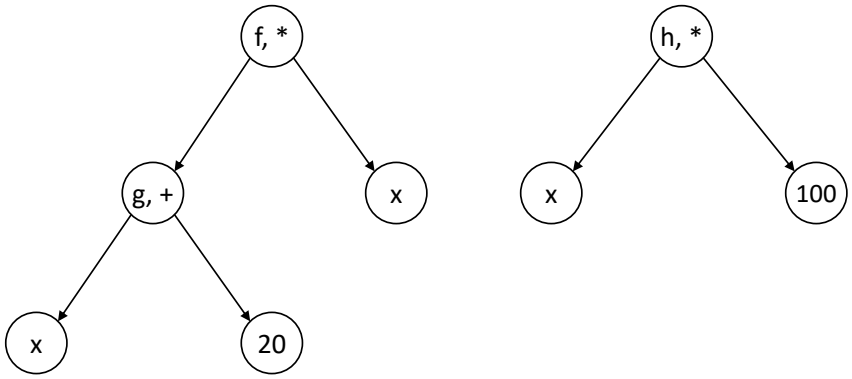

Figure 1: The forest (set of trees) representing the set of function definitions for our running example.

* Description: A list of trees for each linked set of function definitions such that (i) each tree starts with a function node and through the branches the linked function definitions are represented, and (ii) each independent set of function definitions is represented as a separate tree.
* Syntax for a tree: A function node is represented as:

  `[function, operator, term1, term2]`,

  where `function` (str), `operator` (str), `term1` (list) and `term2` (list) represent concepts as defined in Introduction. `term1` and `term2` are lists representing terms, which may be defining another function, a constant or a variable (`x`). A constant or a variable term is represented as `[constant]` or `['x']` respectively, as lists.
* Example: For `["g(x) = x + 20", "h(x) = x * 100", "f(x) = g(x) * x"]`, the expected return value is as follows:

  `[['f', '*', ['g', '+', ['x'], ['20']], ['x']], ['h', '*', ['x'], ['100']]]`

- To simplify the problem, you can assume that no two trees share functions.
- A function definition (str) can contain arbitrary number of spaces between function-names, operators, variables, constants and parentheses. E.g., the following are all valid and equal: `"f (x)=    x*   20"`, `"f(x)   =x*20"`, `"f(  x  )= x *20"`.
- The order of trees in the forest is not important. However, the order of branches should follow the order of the terms in the definitions.
- There may not be recursion or cyclic links between functions.

SAMPLE RUN

```
>>> Defs = ["g(x) = x + 20", "h(x) = x * 100", "f(x) = g(x) * x"]
>>> Forest = construct_forest(Defs)
>>> print(Forest)
[['f', '*', ['g', '+', ['x'], ['20']], ['x']], ['h', '*', ['x'], ['100']]]
```

RESTRICTIONS and GRADING

- **You are not allowed to import any libraries**.
- Your solution will not be tested with erroneous parameters.
- Comply with the specifications outlined. Do not expect any input from the user or print anything on screen. Conform to the specifications and the expected return type. Your program will be graded through an automated process and any violation of the specifications will lead to errors (and reduction of points) in automated evaluation.
- Your program will be tested with multiple data (a distinct run for each data). Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall THE4 grade in the range of [0,30].
- A program based on randomness will be graded zero.
- The glass-box test grade is not open to discussion nor explanation.