CSC4021Z: Compilers I
2021
Assignment 3

Department of Computer Science, UCT

March 26, 2021

Question 1 (20 marks)

Preparations:

- Read through the first 41 slides of the CSC4021Z_PLY_Slides.pdf (on Vula) to get an overview of the tool.
- Download PLY-3.11 from `https://www.dabeaz.com/ply/` and install it.
- Read through the example at `https://www.dabeaz.com/ply/ply.html`.

Task

1. Build a lexer, whose file is called `lexer.py`, that recognises the tokens of a simple adder as one may encounter in a programming language. Thus, it should be able to recognise, **multi-lines input** such as,
   ```
   a = 42
   b = 1
   c = a + (a + b)
   d = 3 + a
   ```
   where names are one or more characters a-z, A-Z, or _ to start with, and then any number of letters, underscores, and/or digits, and the numbers are just integers.
2. The single character "`#`" on a line is used to indicate the end of the input.
3. Print the tokens, in the format of (TYPE, token, line number, position), to check that it does it correctly.
4. For any illegal character, print `Illegal character` followed by the illegal character in single quotes.

Sample input/output

Input:

```
a=45
b=a+(78-a)*2
#
```

Output:

```
('NAME', 'a', 1, 0)
('=', '=', 1, 1)
('NUMBER', 45, 1, 2)
('NAME', 'b', 2, 5)
('=', '=', 2, 6)
('NAME', 'a', 2, 7)
('+', '+', 2, 8)
('(', '(', 2, 9)
('NUMBER', 78, 2, 10)
Illegal character '-'
('NAME', 'a', 2, 13)
(')', ')', 2, 14)
Illegal character '*'
('NUMBER', 2, 2, 16)
```

Question 2 (30 marks)

Preparations:

- Read through the rest of the CSC4021Z_PLY_Slides.pdf slides (on Vula) to get an overview of the tool.
- Read through the example at `https://www.dabeaz.com/ply/ply.html`.

Task

1. You will build a parser for the lexer you've created in the previous task, and save it in a file called `parser.py`. Besides recognising input in the correct format, such as
   ```
   a = 42
   b = 1
   c = a + (a + b)
   ```
   it should give errors on faulty input, such as
   ```
   = 42
   b 1
   c = a + (a + b))
   ```
2. It is thus your task to devise the grammar that can handle these sort of input strings, including rejecting those that are not. This includes things like checking for balanced parentheses and proper use of the operators.
3. Print whether the input is accepted by the grammar.

Sample input/output

Input Case 1:

```
a=78
c=896
d=7
e=985+d
#
```

Output Case 1:

```
Accepted
```

Input Case 2:

```
year=2020
month=01
x=8+( ]
#
```

Output Case 2:

```
Error in input
```

Question 3 (50 marks)

Lexers and parsers can be generated for lots of other things. Take, for instance, chemical structures like salt, NaCl, that has two atoms that one easily can count by hand, but caffeine with C8H10N4O2 requires a bit more effort. The longest is titin (a protein in human muscle): C169719H270466N45688O52238S911. Just how many atoms do those molecules have? It would be nice to calculate that automatically, given as input some arbitrary molecular formula.

1. The first step to automate that is to build a lexer that can process the input, being a molecular formula, and split it up into symbols and their counts, if there's more than one of that element. So then an input of `NaCl2` splits up into:
   `'SYMBOL' 'Na'`
   `'SYMBOL' 'Cl'`
   `'COUNT' 2`
   Subject domain considerations and constraints to take into account:

   - Any element from the periodic table may occur in a formula, and only those. That is, it should accept only valid symbols, i.e., any one from list of 118 symbols: `https://en.wikipedia.org/wiki/List_of_chemical_elements`.
   - The symbols (meaning in this context: element abbreviations) are case sensitive.
   - For the size of the counts: take as ballpark figure the order of magnitude of titin.

   For convenience, the input is also ended with a "`#`", as in the previous exercise.

2. Now add the grammar. Rules that will have to be in the grammar include:

   - If the length of the chemical equation is 1 that's the empty string, then there's no equation.
   - If the length of the chemical equation is 1 that's a non-empty string and it is a valid 'symbol' of the periodic table of elements (so, in the sense of the lexer you've created in the previous step), then that's the 'count' as well as the total number of atoms of the molecule.
   - The order of symbols and counts alternates in a chemical equation.
   - It always starts with a symbol and never can start with a count.

- The chemical structure ends with a symbol or with a count.
- There cannot be two successive counts. (a contiguous sequence of digits is one number).
- There cannot be two successive same symbols. (e.g., HeHe is incorrect, as it would be written as He2)
- The count may be 'empty'—well, it counts as 1 then in adding up the number of atoms, but that's not written in the equation given as input. For instance, with `NaCl2`, there's one `Na` in there, but one never writes it as `Na1Cl2`.

3. Add the adding-up to the code to complete the task of calculating the number of atoms in a molecule.
4. Consider error handling. When it receives a wrong input, such as `2NaCl` or a letter combination not in the periodic table of elements, it should return `error in formula`.
5. Name your file `molecule.py`.

Sample input:

```
NaCl2
C8H10N4O2
MoRe2PrOCeS
HoNoURs2020
NiNeTi84
#
```

Sample output:

```
3
24
7
error in formula
86
```

Submit `lexer.py`, `parser.py`, *and* `molecule.py` *in a single .zip file to the Automatic Marker.*

p.s.: it is possible to do all this without lexer and parser, but the point here is to practice with those, so if you submit code that doesn't take the lexer & parser approach with REs and CFGs, you will receive 0.