Vivian Do
CSC 4330
Nov 27 2023

Homework 8

Write a lexer and a parser using PLY (very much like that for the WAE minilanguage) for the simplified C language with the following grammar, with nonterminals surrounded by angular braces < >, the -> is a production, and everything else is literal (terminal) symbols:

```
<stmt> -> <id> = <expr> ;
<stmt> -> { <stmtlist> }
<stmt> -> if ( <expr> ) <stmt>
<stmt> -> while ( <expr> ) <stmt>

<stmtlist> -> <stmt>
<stmtlist> -> <stmtlist> <stmt>

<expr> -> <id>
<expr> -> <num>
<expr> -> <expr> <optr> <expr>
```

where <optr> can be any operator from the set {>=, <=, ==, >, <, +, -, *, /}, <id> can be any legal identifier in most programming languages, such as x, y, Xy, y_1, z2, etc., and <num> can be any real number, such as 8, -8, +9, 9., -9., +10.5, 3.14159, etc.

Note that this grammar is very similar to that presented on slide 6 "Example" of the slides on context-free languages on iCollege. In fact, this grammar is more general, in that it can do everything the grammar on slide 6 can do, but also has a while loop, a richer set of operators (than just > and +), can represent identifiers beyond just x and y, and can represent any real number (not just 0, 1 and 9).

You need only write a lexer and a parser (like with the WAE example), which outputs the "parse tree". For example, since this language is more general than that presented on slide 6, it can handle the input:

* Note: the output I am getting does not look exactly 1-to-1 with the given output in the problem, but I did my best to make it look as close as possible!

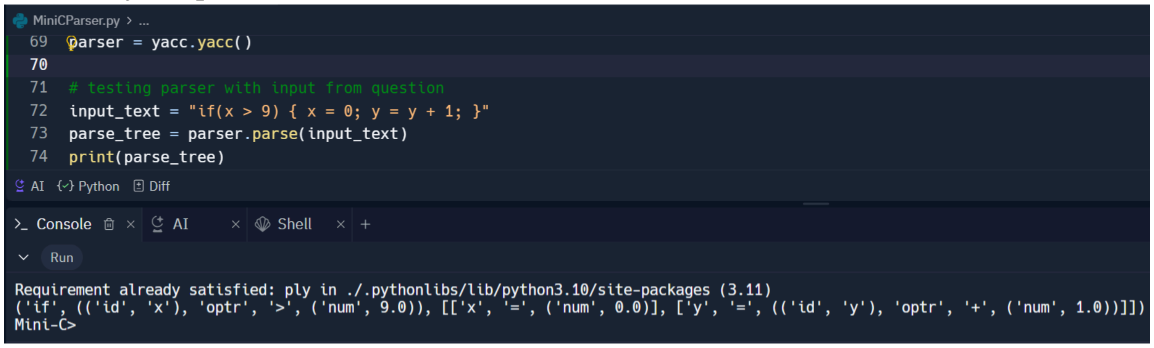
Input & Output Parse Tree 1:

if(x > 9) { x = 0; y = y + 1; }

outputting parse tree:

['if', [['id', 'x'], ['optr', '>'], ['num', 9.0]], [['id', 'x'], '=', ['num', 0.0], ['id', 'y'], '=', [['id', 'y'], ['optr', '+'], ['num', 1.0]]]]

My Output:



MiniCParser.py > ...

```python
parser = yacc.yacc()

# testing parser with input from question
input_text = "if(x > 9) { x = 0; y = y + 1; }"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
('if', (('id', 'x'), 'optr', '>', ('num', 9.0)), [['x', '=', ('num', 0.0)], ['y', '=', (('id', 'y'), 'optr', '+', ('num', 1.0))]])
Mini-C>
```

Input & Output Parse Tree 2:

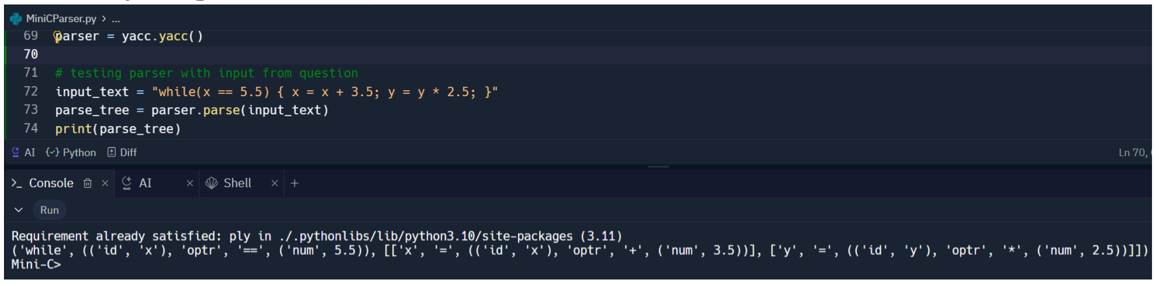
but also something more general, like:

while(x == 5.5) { x = x + 3.5; y = y * 2.5; }

outputting parse tree:

['while', [['id', 'x'], ['optr', '=='], ['num', 5.5]], [['id', 'x'], '=', [['id', 'x'], ['optr', '+'], ['num', 3.5]], ['id', 'y'], '=', [['id', 'y'], ['optr', '*'], ['num', 2.5]]]]]

My Output:



```python
parser = yacc.yacc()

# testing parser with input from question
input_text = "while(x == 5.5) { x = x + 3.5; y = y * 2.5; }"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
('while', (('id', 'x'), 'optr', '==', ('num', 5.5)), [['x', '=', (('id', 'x'), 'optr', '+', ('num', 3.5))], ['y', '=', (('id', 'y'), 'optr', '*', ('num', 2.5))]])
Mini-C>
```

Input & Output Parse Tree 3:

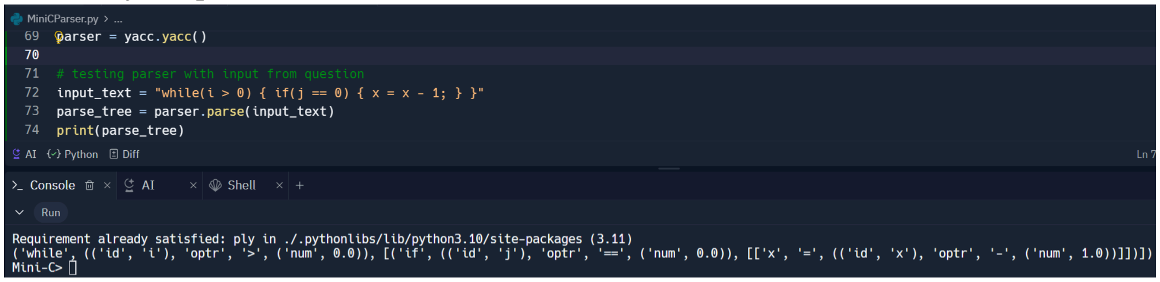
Of course, since an "if" or "while" passes control to some "statement" underneath the "if" (or "while") condition, this "statement" could itself be an "if" or "while", and so (arbitrary) nesting is possible:

while(i > 0) { if(j == 0) { x = x - 1; } }

outputting parse tree:

['while', [['id', 'i'], ['optr', '>'], ['num', 0.0]], ['if', [['id', 'j'], ['optr', '=='], ['num', 0.0]], [['id', 'x'], '=', [['id', 'x'], ['optr', '-'], ['num', 1.0]]]]]

My Output:



```python
parser = yacc.yacc()

# testing parser with input from question
input_text = "while(i > 0) { if(j == 0) { x = x - 1; } }"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
('while', (('id', 'i'), 'optr', '>', ('num', 0.0)), [('if', (('id', 'j'), 'optr', '==', ('num', 0.0)), [['x', '=', (('id', 'x'), 'optr', '-', ('num', 1.0))]])])
Mini-C>
```

Input & Output Parse Tree 4:

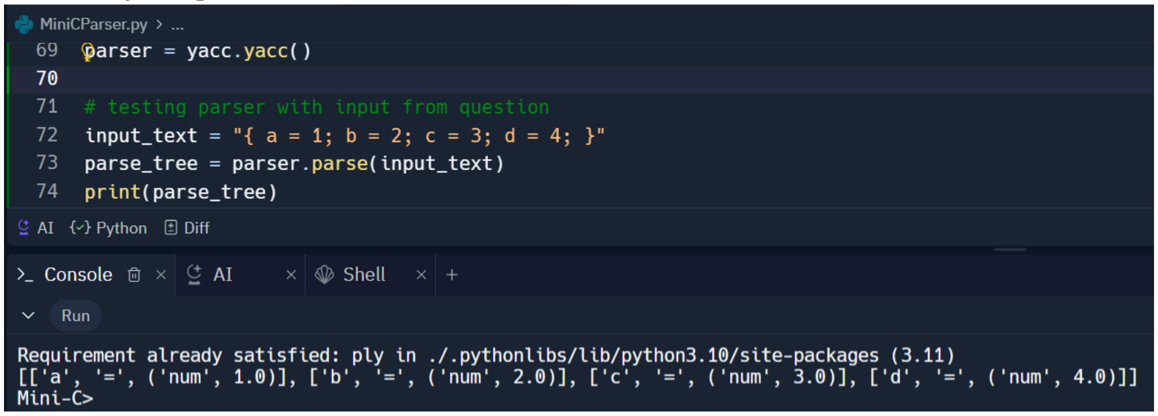
Since "statement lists" can be arbitrary in length, it is possible to have:

{ a = 1; b = 2; c = 3; d = 4; }

outputting parse tree:

[['id', 'a'], '=', ['num', 1.0], ['id', 'b'], '=', ['num', 2.0], ['id', 'c'], '=', ['num', 3.0], ['id', 'd'], '=', ['num', 4.0]]

My Output:



```
parser = yacc.yacc()

# testing parser with input from question
input_text = "{ a = 1; b = 2; c = 3; d = 4; }"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
[['a', '=', ('num', 1.0)], ['b', '=', ('num', 2.0)], ['c', '=', ('num', 3.0)], ['d', '=', ('num', 4.0)]]
Mini-C>
```

Input & Output Parse Tree 5:

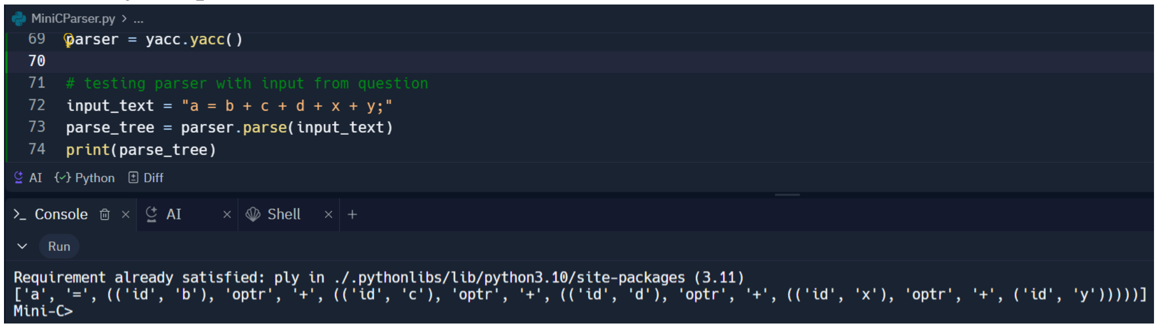
and expressions (interleaved by operators) can also be arbitrary in length, and so it is possible to have:

a = b + c + d + x + y;

outputting parse tree:

[['id', 'a'], '=', [['id', 'b'], ['optr', '+'], [['id', 'c'], ['optr', '+'], [['id', 'd'], ['optr', '+'], [['id', 'x'], ['optr', '+'], ['id', 'y']]]]]]

My Output:



```
parser = yacc.yacc()

# testing parser with input from question
input_text = "a = b + c + d + x + y;"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
['a', '=', (('id', 'b'), 'optr', '+', (('id', 'c'), 'optr', '+', (('id', 'd'), 'optr', '+', (('id', 'x'), 'optr', '+', ('id', 'y')))))]
Mini-C>
```

Input & Output Parse Tree 6:

hence a combination of the two is possible:

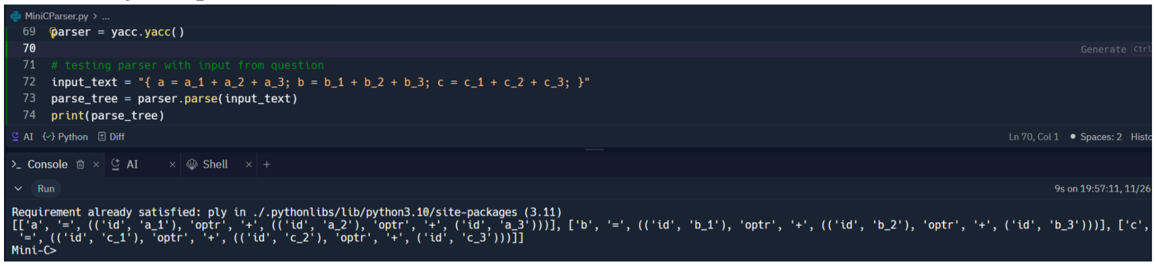
{ a = a_1 + a_2 + a_3; b = b_1 + b_2 + b_3; c = c_1 + c_2 + c_3; }

outputting parse tree:

[['id', 'a'], '=', [['id', 'a_1'], ['optr', '+'], [['id', 'a_2'], ['optr', '+'], ['id', 'a_3']]], ['id', 'b'], '=', [['id', 'b_1'], ['optr', '+'], [['id', 'b_2'], ['optr', '+'], ['id', 'b_3']]], ['id', 'c'], '=', [['id', 'c_1'], ['optr', '+'], [['id', 'c_2'], ['optr', '+'], ['id', 'c_3']]]]

or all of this can be inside a nested "if"/"while" statement, etc...

My Output:



```python
parser = yacc.yacc()

# testing parser with input from question
input_text = "{ a = a_1 + a_2 + a_3; b = b_1 + b_2 + b_3; c = c_1 + c_2 + c_3; }"
parse_tree = parser.parse(input_text)
print(parse_tree)
```

```
Requirement already satisfied: ply in ./.pythonlibs/lib/python3.10/site-packages (3.11)
[['a', '=', (('id', 'a_1'), 'optr', '+', (('id', 'a_2'), 'optr', '+', ('id', 'a_3')))], ['b', '=', (('id', 'b_1'), 'optr', '+', (('id', 'b_2'), 'optr', '+', ('id', 'b_3')))], ['c',
 '=', (('id', 'c_1'), 'optr', '+', (('id', 'c_2'), 'optr', '+', ('id', 'c_3')))]]
Mini-C>
```