T-603-THYD Compilers
Project: A *lexer* for ON

Objective

The objective of this project is to get you starting with writing a compiler/interpreter. The focus of this part of the project is on lexical analysis.

You should work on the project in a group of two. You need instructor permission to do the project individually.

Description

We will develop an interpreter/compiler for a sizeable part of the Python programming language in the course. We will start with a much-simplified language, but as the term progresses and our knowledge expands, we will gradually add new program constructs to the language. The first version of our programming language we encounter is ***ON***.

The programming language ***ON*** supports basic statements like assignments and *if*- and *while*-statements. It also allows us to form complex expressions using Boolean, arithmetic, and comparison operators. For the most part, the constructs are legitimate Python constructs, *except* that we:

- are using curly brackets and parenthesis to enclose blocks and test conditions, respectively;
- have mandated the use of semi-colons to end simple statements (the statements can spawn more than one line);
- have introduced a new keyword ***let*** in assignment statements.

Those exceptions make tokenization and parsing easier. However, we will remove those exceptions in the following language version and use proper Python syntax.

The appendix gives the detailed syntax of the ***ON*** programming language.

Your task is to write a lexical analyser for the programming language ***ON***. Study the grammar in the appendix carefully to identify all the tokens your lexer should recognize.

A simple lexer for arithmetic expressions is provided --- use it as your starting point. To simplify the testing of your submitted code, you are not allowed to change the interface to the Lexer class (i.e., calling the *constructor* and then the *next* method repeatedly should be sufficient to tokenize the input).

An ***ON*** source file may include Python-style comments. Note that Python does not have block comments, only line comments (#).

In Canvas, **hand-in** a single Python file called *onlexer.py*.

APPENDIX: The syntax of the *ON* language *(expessed in EBNF-like style)*

program: *stmt** **EOI**

stmt: *simple_stmt* '**;**' | *compound_stmt*

simple_stmt: *assign_stmt* | *pass_stmt* | *break_stmt* | *continue_stmt*

compound_stmt: *if_stmt* | *while_stmt*

assign_stmt: '**let**' identifier '**=**' *expression*

pass_stmt: '**pass**'

break_stmt: '**break**'

continue_stmt: '**continue**'

if_stmt: '**if**' '**(**' *expression* '**)**' *block* ('**elif**' '**(**' *expression* '**)**' *block*)* ['**else**' *block*]

while_stmt: '**while**' '**(**' *expression* '**)**' *block*

block: *stmt* | '**{**' *stmt** '**}**'

expression: or_expression

or_expression: *and_expression* ('**or**' *and_expression*)*

and_expression: *not_expression* ('**and**' *not_expression*)*

not_expression: '**not**' *not_expression* | *comparison*

comparison: arithmetic_expr [*comparison_op arithmetic_expr*]

comparison_op: '**<**' | '**>**' | '**==**' | '**>=**' | '**<=**' | '**!=**'

arithmetic_expr: *term* (('**+**'|'**-**') *term*)*

term: *factor* (('*****' | '**/**' | '**%**' | '**//**') *factor*)*

factor: ('**+**'|'**-**') *factor* | *atom*

atom: *variable* | ***NUMBER*** | ***STRING*** | '**None**' | '**True**' | '**False**' | '**(**' *expression* '**)**'

variable: ***IDENTIFIER***

- ***NUMBER*** is token identifying integer-literals (one or more *digits*), e.g., *42*.
- ***IDENTIFIER*** is a token consisting of one or more *alphanumeric* and *understore* characters; an identifier is not allowed to start with a *digit*. (e.g., *name_10_beers*).
- ***STRING*** is a string-literal, starting and ending with a pair of either single (') or double (") quotes, e.g. *'A string'*, *"Another string"*, *'A string with " in it.'* , *"A string with 2 ' and ' in it."*). Strings are not allowed to span over lines and do not (yet) support *escaping* characters.