Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work nr. 3
Course: Formal languages and finite automata
Topic: Lexer & Scanner
Variant-20

Elaborated:
st. gr. FAF-223 Pereteatcu Arina

Verified:
asist. univ. Cretu Dumitru

Chişinău - 2024

Theoretical considerations

The initial stage of the compiler, also referred to as a scanner, is called **lexical analysis**. The High Level Input Program is transformed into a series of *Tokens*.

1.It is possible to apply lexical analysis using deterministic finite automata.

2.A series of tokens that are passed to the parser for syntactic analysis make up the output. In computer languages, **a lexical token** is a group of characters that can be regarded as a single unit in the grammar.

With the aid of the automation machine and the grammar of the language it is based on, the lexical analyser finds the error and outputs the row and column numbers of the error.

A lexical analyser does:

Preprocessing the input text entails cleaning it up and getting it ready for lexical analysis. This can entail eliminating whitespace, comments, and other characters from the input text that are not necessary.

Tokenization: The technique of dividing the input text into a series of tokens. Typically, to do this, a set of patterns or regular expressions that define the various types of tokens are compared to the characters in the input text.

Token classification: At this point, each token's type is ascertained by the lexer. For instance, the lexer may identify operators, punctuation symbols, identifiers, and keywords as distinct token kinds in a programming language.

Token validation: During this phase, the lexer verifies that every token complies with the language's specifications. It could verify, for instance, that an operator has the proper syntax or that a variable name is a valid identifier.

During the last phase, the lexer produces the lexical analysis process's output, which is usually a list of tokens. After that, the list of tokens can be moved on to the compilation or interpretation phase.

The **Koch snowflake** is a fractal curve, also known as the Koch island, which was first described by Helge von Koch in 1904. It is built by starting with an equilateral triangle, removing the inner third of each side, building another equilateral triangle at the location where the side was removed, and then repeating the process indefinitely.

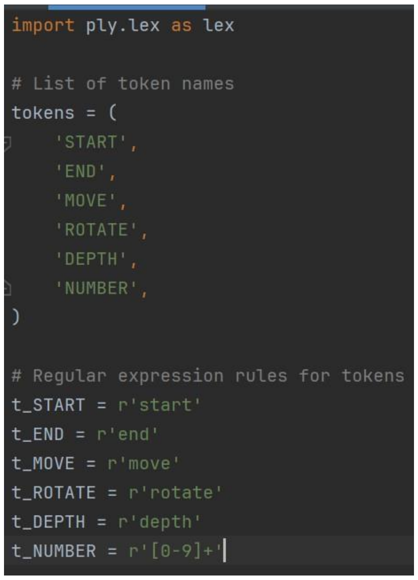
Objectives:

1. Understand what lexical analysis is
2. Get familiar with inner workings of a lexer/scanner/tokenizer
3. Implement a sample lexer and show how it works

Implementation Description:

In this laboratory work I implemented a sample lexer for building fractal Koch's Snowflake, using Python.

Code:



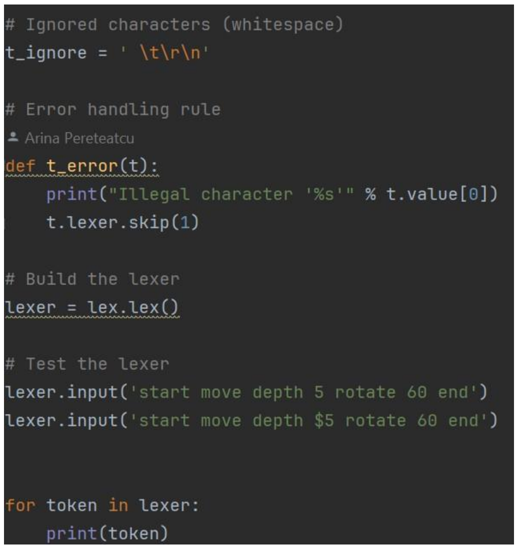
```python
import ply.lex as lex

# List of token names
tokens = (
    'START',
    'END',
    'MOVE',
    'ROTATE',
    'DEPTH',
    'NUMBER',
)

# Regular expression rules for tokens
t_START = r'start'
t_END = r'end'
t_MOVE = r'move'
t_ROTATE = r'rotate'
t_DEPTH = r'depth'
t_NUMBER = r'[0-9]+'
```

Figure1

In my code I imported the lexical analysis module lex from ply library and defined a tuple "tokens" which contains the name of the tokens that the lexer will recognize. Also, I have implemented rules for the tokens for identifying the symbols in the input.

Ply is a parsing tool that helps in constructing parsers for complex data. It is used for processing structured textual data. Ply.lex is a module that deal with lexical analysis, know as tokenization.

First, the tokens are defined and are recognized by parser. After, regular expressions to match tokens in input text. When input is written, it is broken into tokens based on those regular expressions, the library scans the input from left to right and matches. For each token matched, the lexer return a token object with attributes such as token type, value and position in the input text. The parser then processes these tokens according to the grammar rules defined.



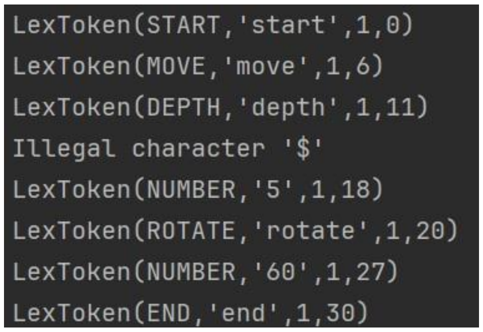
```
# Ignored characters (whitespace)
t_ignore = ' \t\r\n'

# Error handling rule
Arina Pereteatcu
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

# Build the lexer
lexer = lex.lex()

# Test the lexer
lexer.input('start move depth 5 rotate 60 end')
lexer.input('start move depth $5 rotate 60 end')


for token in lexer:
    print(token)
```

Figure2

In this part of code I defined characters which will be ignored by lexer, like whitespace, space, tab, newline. In addition, there is a function for handling errors, if the input is invalid then it prints an error message, then it is skipped to the next one.

Lastly, there are tests for the lexer.

Output:



```
LexToken(START,'start',1,0)
LexToken(MOVE,'move',1,6)
LexToken(DEPTH,'depth',1,11)
Illegal character '$'
LexToken(NUMBER,'5',1,18)
LexToken(ROTATE,'rotate',1,20)
LexToken(NUMBER,'60',1,27)
LexToken(END,'end',1,30)
```

Figure3

It is indicated the successfully recognized input at shows the line number and the position in the line where the token was found.
Where the input is invalid it is shown the message "Illegal character".

Conclusions:

To sum up, I explored the fundamental ideas of lexical analysis within the framework of compiler development during this lab activity, emphasizing the significance of this first step in the compilation process.

I investigated the function of the lexer, sometimes referred to as a scanner or tokenizer, in converting high-level input programs into a set of tokens that are used as the input for the parser's ensuing syntactic analysis. I was able to understand the essential elements and functions of lexical analysis by looking at its internal workings. To provide accurate parsing of programming languages, the lexer performs many crucial phases, which I learnt about: preparation of the input text; tokenization; token classification; and token validation. An actual demonstration of these ideas was given by putting a sample lexer for creating fractal Koch's Snowflake using Python into practice. I demonstrated how to create token rules, deal with failures, and test the lexer using different input strings through this implementation.

All things considered, this lab effort improved my comprehension of lexical analysis and its function in compiler construction. It gave me the know-how and abilities needed to create lexers for various programming languages and applications, setting the groundwork for future research and development in the areas of compiler design and language processing.

Bibliography:

1. GEEKS FOR GEEKS, *"Introduction of Lexical Analysis"* [online].[Last Accessed 16.03.2024]. Available: https://www.geeksforgeeks.org/introductionof-lexical-analysis/
2. PYTHONMEMBERS.CLUB, *"Building a lexer in python-a tutorial"* [online].[Last Accessed 16.03.2024]. Available: https://medium.com/@pythonmembers.club/building-a-lexer-in-python-atutorial-3b6de161fe84
3. EMILY FUNG, "*KOCH'S SNOWFLAKE*" [online].[Last Accessed 16.03.2024]. Available: https://personal.math.ubc.ca/~cass/courses/m308-

05b/projects/fung/page.html