ASSIGNMENT NO# 02

NAME: DANYAL SHAH

REG. NO: 200901046

SECTION: B

SUBJECT: Compiler Construction

DATE: 30-12-2022

SUBMITTED TO: Ms. Reeda Saeed

Abstract

In this assignment,we are requested to understand and implement the working of Compiler.It consists of six phases Each of this phase helps in converting the high-level langue the machine code. The phases of a compiler are:

1.Lexical analysis

2.Syntax analysis

3.Semantic analysis

4.Intermediate code generator

5.Code optimizer

6.Code Generator

Source Code:

```
import re
#import re module from python

import ast
#import AST module from python for parsing

# Define the tokens
PLUS = r'\+'
MINUS = r'-'
TIMES = r'\*'
DIVIDE = r'/'
LPAREN = r'\('
RPAREN = r'\)'
ID = r'[a-zA-Z][a-zA-Z0-9]*'
INTEGER = r'\d+'

def tokenize(expression):
    output = []
  # Use the regular expression to find all tokens in the input string
    for token in expression:

        if token == ' ':
            pass
        elif re.match(ID,token):
            output.append(("Identifier",token))
        elif re.match(INTEGER,token):
            output.append(("Constant",token))
        elif re.match(LPAREN,token) or re.match(RPAREN,token):
            output.append(('Punctuator',token))
        elif re.match(PLUS,token) or re.match(MINUS,token) or
re.match(TIMES,token) or re.match(DIVIDE,token):
            output.append(('Operator',token))
        else:
            output.append(('Special Character',token))

    for i in output:
        print(i)

#func for parsing expression
def parse_expression(expression):
  return ast.parse(expression)

def print_ast(tree):
```

```
      print(ast.dump(tree,indent = 2))

 #main function
if __name__ == '__main__':

    print('Parse Tree')
    expression = "a + b * c"
    tree = parse_expression(expression)
    print_ast(tree)

    print('Lexical Analyzer')
    # Test the tokenize function
    expression = input("Enter the expression e.g(a+b): ")
    tokenize(expression)
```

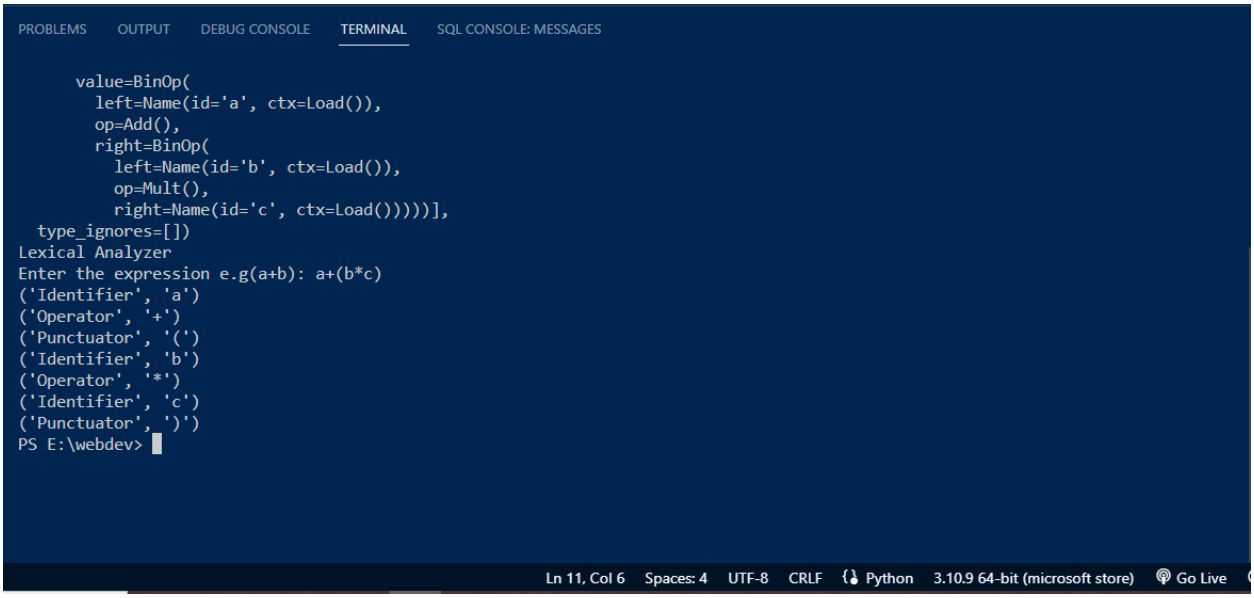
Explanation:

The tokenize function then iterates through each character in the input expression string. If the character is a space, it is ignored. If the character matches one of the regular expressions defined earlier, it is added to the output list as a tuple along with its corresponding token type. Otherwise, the character is added to the output list as a tuple with the token type "Special Character".

The code also defines a function called parse_expression which takes in a string expression and returns an abstract syntax tree (AST) representation of the input expression using the ast module. Finally, the code defines a function called print_ast which takes in an AST tree and prints it out in a pretty-printed format.

The code also includes a main block which first parses the string "a + b * c" into an AST and then prints it out. It then prompts the user to enter an expression, tokenizes it, and prints out the resulting list of tokens.

Output:



GitHub Link:

[danyalthewebdev/Implementation-of-Lexical-Analyzer: Implementation of lexical Analyzer and AST tree (github.com)]