31st December 2022

Compiler Construction

Assignment

2

Submitted by:

Syed Ali Raza

200901028

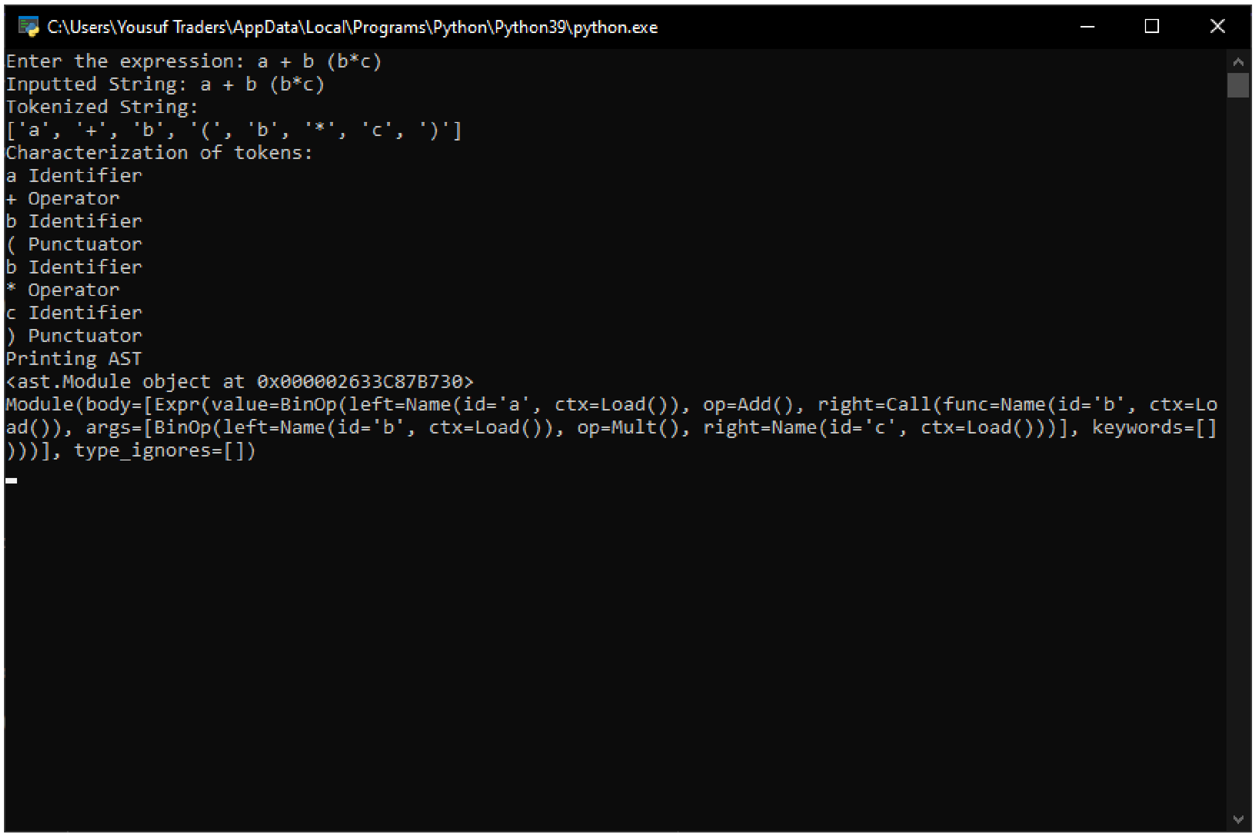
Submitted to:

Ma’am Reeda Saeed

Code

```
#Syed Ali Raza 200901028 CS-01-B

import re
import string
import ast
from pprint import pprint

#Module 1
class lexical_analyzer:
    def __init__(self, string):
        self.expression = string
        self.tokenized = []
        self.whitespace = [' ','\t','\n']
        self.constants = ['1','2','3','4','5','6','7','8','9','0']
        self.operators = ['+','-','*','/']
        self.sp_characters = ['&', '%']
        self.puncuators = [';' , ':', ',', '(',')','{','}','[',']']

    def tokenize(self):
        self.tokenized = re.findall(r'\b\d+\b|\b[a-zA-Z]+\b|[+-/*&%;:,(){}\[\]\s]',
self.expression)
        self.tokenized = [token for token in self.tokenized if token not in
self.whitespace]
        print("Tokenized String: ")
        print(self.tokenized)

    def type_of_token(self):
        print("Characterization of tokens: ")
        for char in self.tokenized:
            if char in string.ascii_letters:
                print(char + " Identifier")
            elif char in self.constants:
                print(char + " Constant")
            elif char in self.operators:
                print(char + " Operator")
            elif char in self.sp_characters:
                print(char + " Special Character")
            elif char in self.puncuators:
                print(char + " Punctuator")

if __name__ == "__main__":
    input_string = input("Enter the expression: ")
    obj1 = lexical_analyzer(input_string)
    print('Inputted String: ' + input_string)
    obj1.tokenize()
    obj1.type_of_token()

    #Module 2
    print("Printing AST")
    code = ast.parse(input_string)
    pprint(code)
    print(ast.dump(code))
    exec(compile(code, filename="", mode="exec"))
```

Output



C:\Users\Yousuf Traders\AppData\Local\Programs\Python\Python39\python.exe

```
Enter the expression: a + b (b*c)
Inputted String: a + b (b*c)
Tokenized String:
['a', '+', 'b', '(', 'b', '*', 'c', ')']
Characterization of tokens:
a Identifier
+ Operator
b Identifier
( Punctuator
b Identifier
* Operator
c Identifier
) Punctuator
Printing AST
<ast.Module object at 0x000002633C87B730>
Module(body=[Expr(value=BinOp(left=Name(id='a', ctx=Load()), op=Add(), right=Call(func=Name(id='b', ctx=Lo
ad()), args=[BinOp(left=Name(id='b', ctx=Load()), op=Mult(), right=Name(id='c', ctx=Load()))], keywords=[]
)))], type_ignores=[])
```

Explanation

Imported Modules

The code imports the following modules:

1. **re:** The re (Regular Expression) module provides functions for working with regular expressions in Python.
2. **string:** The string module provides constants and functions for working with string data in Python.
3. **ast**: The ast (Abstract Syntax Tree) module provides functions for parsing and manipulating Python code in the form of abstract syntax trees.
4. **pprint**: The pprint (Pretty Print) module provides functions for pretty-printing data structures in Python.

Class Definition

The **lexical_analyzer** class represents a lexical analyzer that can tokenize a string of code and classify the tokens. It has the following attributes:

1. **expression:** The input string to be tokenized.
2. **tokenized:** A list of the tokenized form of the input string.
3. **whitespace**: A list of characters that represent whitespace (i.e., spaces, tabs, and newlines).
4. **constants**: A list of characters that represent constants (i.e., digits).
5. **operators**: A list of characters that represent operators (i.e., +, -, *, and /).
6. **sp_characters:** A list of characters that represent special characters (i.e., & and %).
7. **puncuators**: A list of characters that represent punctuators (i.e., ;, :, ,, (,), {, }, [, and]).

Method Definitions

The **lexical_analyzer** class has two methods:

- **tokenize:** This method uses a regular expression to split the input string into tokens and stores the resulting list of tokens in the tokenized attribute.
- **type_of_token:** This method iterates over the list of tokens in the tokenized attribute and prints the type of each token.

Main Function

- In the main function, the user is prompted to enter an expression, and the input is stored in the input_string variable. An instance of the lexical_analyzer class, obj1, is created with the input_string as its argument.
- The tokenize method of obj1 is then called to tokenize the input string, and the type_of_token method is called to classify the tokens.
- Next, the ast.parse function is used to parse the input_string into an abstract syntax tree (AST). The AST is then pretty-printed using the pprint function and printed using the ast.dump function. Finally, the compile and exec functions are used to execute the code represented by the AST.