COMPILER ASSIGNMENT

Name: Vanya Ahmed Siddiqui.

Section: BSCS-01(B)

Reg#200901095

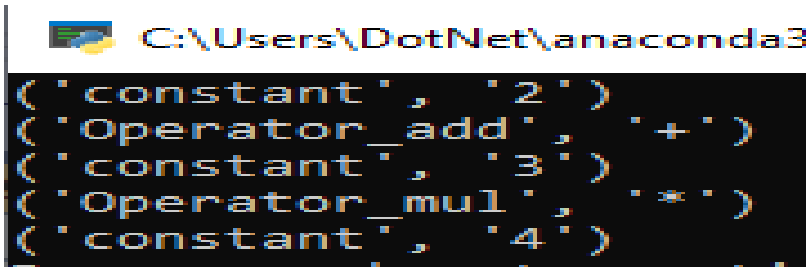
DECEMBER 31, 2022

MODULE_1

HARD CODE: (FROM AN ONLINE COMPILER)

```
import re

# Define a list of tokens that the lexical analyzer should recognize
TOKENS = [
    ('constant', r'\d+'),
    ('Operator_add', r'\+'),
    ('Operator_minus', r'-'),
    ('Operator_mul', r'\*'),
    ('Operator_div', r'/'),
    ('LPAREN', r'\('),
    ('RPAREN', r'\)'),
]

# Create a regular expression pattern that the lexical analyzer would take and
generate the tokens related to expression.
token_pattern = '|'.join('(?P<%s>%s)' % pair for pair in TOKENS)

# lexical analyzer function
def lexer(text):

    for match in re.finditer(token_pattern, text):
        # Extract the token type and value from the match
        token_type = match.lastgroup
        value = match.group()

        yield token_type, value    # Yield a tuple with the token type and value


# Give expression
for token in lexer('2 + 3 * 4'):
    print(token)
```

OUTPUT:



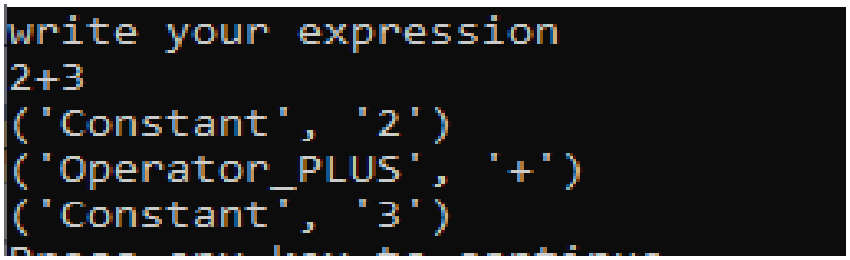
USER INPUT:

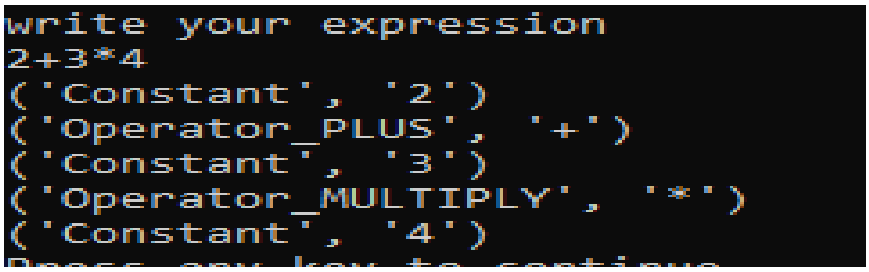
CODE:

```
import re

# Define a list of tokens that the lexical analyzer should recognize
TOKENS = [
    ('Constant', r'\d+'),
    ('Operator_PLUS', r'\+'),
    ('Operator_MINUS', r'-'),
    ('Operator_MULTIPLY', r'\*'),
    ('Operator_DIVIDE', r'/'),
    ('LPAREN', r'\('),
    ('RPAREN', r'\)'),
]

# Create a regular expression pattern that the lexical analyzer would take and
generate the tokens related to expression.
token_pattern = '|'.join('(?P<%s>%s)' % pair for pair in TOKENS)

# lexical analyzer function
def lexer(text):

    for match in re.finditer(token_pattern, text):
        # Extract the token type and value from the match
        token_type = match.lastgroup
        value = match.group()
```

```
        yield token_type, value    # Yield a tuple with the token type and value


# Give expression
X= input("write your expression\n")
for token in lexer(X):
    print(token)
```

OUTPUT:





.....

MODULE 2:

FROM USER INPUT:

```
import ast

# Parse the expression into an AST

x = input("enter your expression\n")


code = ast.parse(x)
print(code)

# Print the AST
print(ast.dump(code))
```

OUTPUT:

```
enter your expression
5+4*8
<_ast.Module object at 0x000002BA51C04E80>
Module(body=[Expr(value=BinOp(left=Constant(value=5, kind=None), op=Add(), right=BinOp(left=Constant(value=4, kind=None)
, op=Mult(), right=Constant(value=8, kind=None))))], type_ignores=[])
```

```
enter your expression
a(b+c)
<_ast.Module object at 0x00000235A020CE80>
Module(body=[Expr(value=Call(func=Name(id='a', ctx=Load()), args=[BinOp(left=Name(id='b', ctx=Load()), op=Add(), right=N
ame(id='c', ctx=Load()))], keywords=[]))], type_ignores=[])
```

FROM HARD CODE:

CODE:

```
import ast

# Parse the expression into an AST

code = ast.parse("print(1000*(8*9))")
print(code)

# Print the AST
print(ast.dump(code))
```

OUTPUT:

```
<_ast.Module object at 0x000001E5E243CEB0>
Module(body=[Expr(value=Call(func=Name(id='print', ctx=Load()), args=[BinOp(left=Constant(value=1000, kind=None), op=Mul
t(), right=BinOp(left=Constant(value=8, kind=None), op=Mult(), right=Constant(value=9, kind=None)))], keywords=[]))], ty
pe_ignores=[])
```

Reference:

https://github.com/vanyaahmed20
