SP Assignment 6

Name: Yash Oswal
Div: B   Roll:38
SRN: 201901226

Input Code:

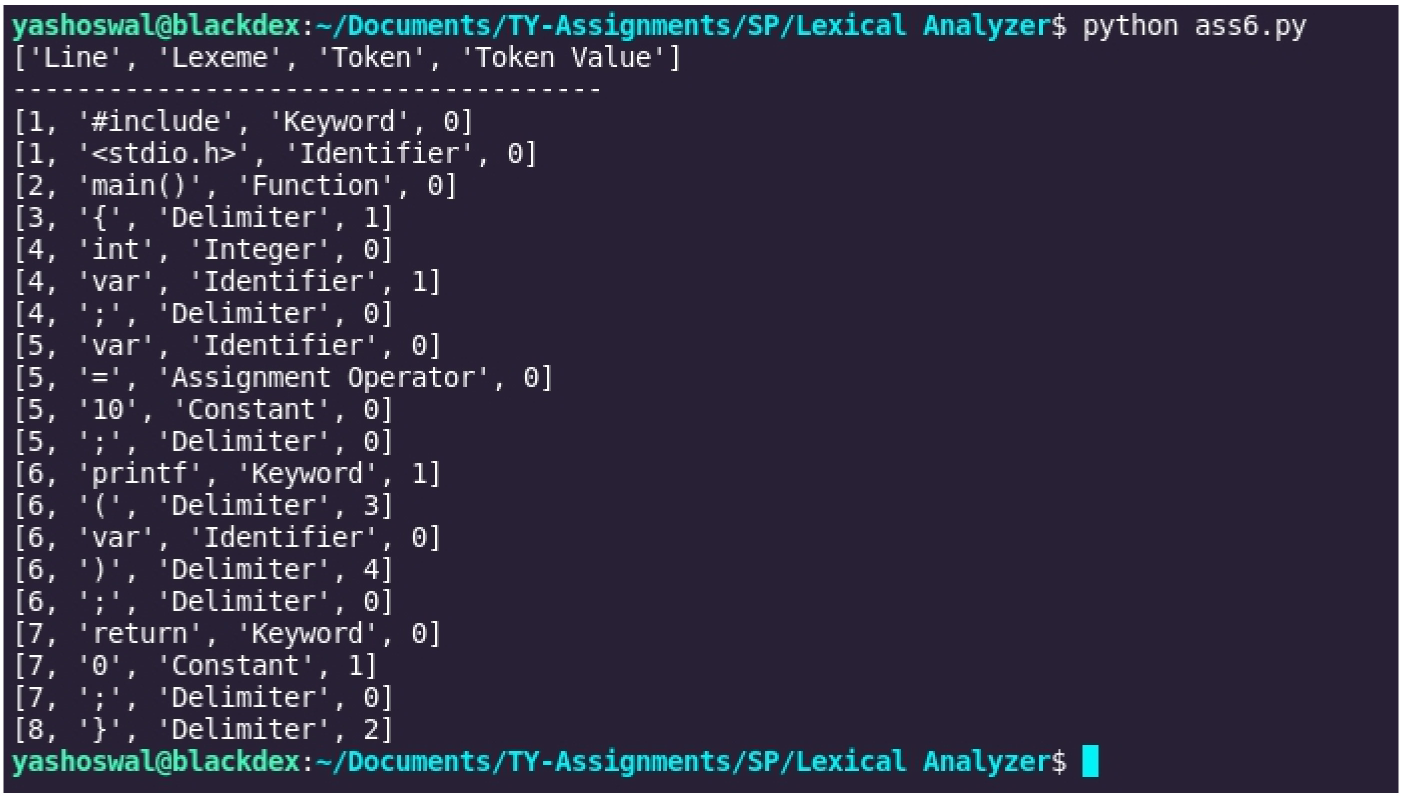
```
#include <stdio.h>
main()
{
    int var ;
    var = 10 ;
    printf ( " The value of a is %d ", var ) ;
    return 0 ;
}
```

Code:-

```
import re

f = open('input.txt','r')
output = []

operators = { '=': 'Assignment Operator','+': 'Additon Operator', '-' : 'Substraction Operator',
'/' : 'Division Operator', '*': 'Multiplication Operator', '++' : 'increment Operator', '--' :
'Decrement Operator'}
optr_keys = operators.keys()

header = {'.h': 'header file'}
header_keys = header.keys()

sp_header_files = {'<stdio.h>':'Standard Input Output Header','<string.h>':'String Manipulation
Library'}

macros = {r'#\w+' : 'macro'}
macros_keys = macros.keys()

datatype = {'int': 'Integer','float' : 'Floating Point', 'char': 'Character','long': 'long int'}
datatype_keys = datatype.keys()

keyword = {'return' : 'keyword that returns a value from a block','printf':'Print the string'}
keyword_keys = keyword.keys()

delimiter = {';':'terminator symbol semicolon (;)','{' : 'Block', '}':'Block','(':'Open
block',')':'Close Block'}
delimiter_keys = delimiter.keys()

builtin_functions = {'printf':'printf prints its argument on the console'}

non_identifiers =
['_','-','+','/','*','`','~','!','@','#','$','%','^','&','*','(',')','=','|','"',':',';','{','}
','[',']','<','>','?','/']

numerals = ['0','1','2','3','4','5','6','7','8','9','10']

# Flags
dataFlag = False
```

```
i = f.read()
cc=0
ic=0
dc=0
id=[]
count = 0
program =  i.split('\n')
output.append(['Line','Lexeme','Token','Token Value'])
for line in program:
    count = count+1
    tokens = line.split(' ')
    for token in tokens:

        if '\r' in token:
            position = token.find('\r')
            token=token[:position]

        if token in optr_keys:
            output.append([count,token,operators[token],list(optr_keys).index(token)])

        if token in macros_keys:
            output.append([count,token,macros[token],list(macros_keys).index(token)])

        if '.h' in token:
            output.append([count,token,'Identifier',list(sp_header_files.keys()).index(token)])

        if '()' in token:
            output.append([count,token,'Function',0])

        if token in id:
            output.append([count,token,'Identifier',id.index(token)])
            ic+=1
        if dataFlag == True and (token not in non_identifiers) and ('()' not in token):
            output.append([count,token,'Identifier',ic])
            id.append(token)
            ic+=1

        if token in datatype_keys:
            output.append([count,token,datatype[token],list(datatype_keys).index(token)])
            dataFlag = True

        if token in keyword_keys:
            output.append([count,token,'Keyword',list(keyword_keys).index(token)])

        if token in delimiter_keys:
            output.append([count,token,"Delimiter",list(delimiter_keys).index(token)])

        if '#' in token:
            match = re.search(r'#\w+', token)
            output.append([count,token,'Keyword',ic])
            ic+=1

        if token in numerals:
            output.append([count,token,'Constant',cc])
            cc+=1

    dataFlag = False



print(output[0],"\n-----------------------------------")
for i in range(1,len(output)):
    print(output[i])

f.close()
```

Output:



```
yashoswal@blackdex:~/Documents/TY-Assignments/SP/Lexical Analyzer$ python ass6.py
['Line', 'Lexeme', 'Token', 'Token Value']
-------------------------------------
[1, '#include', 'Keyword', 0]
[1, '<stdio.h>', 'Identifier', 0]
[2, 'main()', 'Function', 0]
[3, '{', 'Delimiter', 1]
[4, 'int', 'Integer', 0]
[4, 'var', 'Identifier', 1]
[4, ';', 'Delimiter', 0]
[5, 'var', 'Identifier', 0]
[5, '=', 'Assignment Operator', 0]
[5, '10', 'Constant', 0]
[5, ';', 'Delimiter', 0]
[6, 'printf', 'Keyword', 1]
[6, '(', 'Delimiter', 3]
[6, 'var', 'Identifier', 0]
[6, ')', 'Delimiter', 4]
[6, ';', 'Delimiter', 0]
[7, 'return', 'Keyword', 0]
[7, '0', 'Constant', 1]
[7, ';', 'Delimiter', 0]
[8, '}', 'Delimiter', 2]
yashoswal@blackdex:~/Documents/TY-Assignments/SP/Lexical Analyzer$
```