IMPLEMENTATION OF LEXICAL ANALYSER

AIM: To Design a lexical analyser for given language.

LANGUAGE USED: Python 3

ALGORITHM/PROCEDURE: -

1. Write the given code in Python complier

2. We used the regular expressions built-in python tools.

3. We take a code in a given file in the system and analyse the code line by line.

4. We used some keys like operators, headers, macros, data types, identifiers, etc. to analyse the code

5. And the python code checks each line and divide the line into different cases according to the given keys.

6. Then it gives the output in a classified format to identify different cases (operators, headers, macros, data types, identifiers, etc.)

7. And the code runs in a loop till it reaches the final line and then it closes the loop.

8. We get the output of lexical analyser for a given code.

SOURCE CODE: -

```
import re


f = open('Sample1.c','r')


operators = { '=': 'Assignment Operator','+': 'Additon Operator', '-' : 'Substraction Operator', '/' : 'Division Operator', '*': 'Multiplication Operator', '++' : 'increment Operator', '--' : 'Decrement Operator'}

optr_keys = operators.keys()


comments = {r'//' : 'Single Line Comment',r'/*' : 'Multiline Comment Start', r'*/' : 'Multiline Comment End', '/**/' : 'Empty Multiline comment'}

comment_keys = comments.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'}
keyword_keys = keyword.keys()

delimiter = {';':'terminator symbol semicolon (;)'}
delimiter_keys = delimiter.keys()

blocks = {'{' : 'Blocked Statement Body Open', '}':'Blocked Statement Body Closed'}
block_keys = blocks.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()
```

```
count = 0
program =  i.split('\n')

for line in program:
    count = count+1
    print ("Line #",count,"\n",line)



    tokens = line.split(' ')
    print ("Tokens are",tokens)
    print ("Line #",count,'properties \n')
    for token in tokens:

        if '\r' in token:
            position = token.find('\r')
            token=token[:position]
        # print 1

        if token in block_keys:
            print (blocks[token])
        if token in optr_keys:
            print ("Operator is: ", operators[token])
        if token in comment_keys:
            print ("Comment Type: ", comments[token])
        if token in macros_keys:
            print ("Macro is: ", macros[token])
        if '.h' in token:
            print ("Header File is: ",token, sp_header_files[token])
        if '()' in token:
            print ("Function named", token)



 if dataFlag == True and (token not in non_identifiers) and ('()' not in token):
```

```
            print ("Identifier: ",token)
        if token in datatype_keys:
            print ("type is: ", datatype[token])
            dataFlag = True

        if token in keyword_keys:
            print (keyword[token])



        if token in numerals:
            print (token,type(int(token)))

    dataFlag = False


    print ("_____")


f.close()
```

INPUT: -

```
void main() {
int a , b , c;
a = b + 1 ;
}
```

OUTPUT: -

```
runfile('C:/Users/super/CD_EXP1_LAB.py', wdir='C:/Users/super')
Line # 1
 void main() {
Tokens are ['void', 'main()', '{']
Line # 1 properties

Function named main()
Blocked Statement Body Open
_____________________________
Line # 2
 int a , b , c;
Tokens are ['int', 'a', ',', 'b', ',', 'c;']
Line # 2 properties

type is:  Integer
Identifier:  a
Identifier:  ,
Identifier:  b
Identifier:  ,
Identifier:  c;
_____________________________
Line # 3
```

a = b + 1 ;

Tokens are ['a', '=', 'b', '+', '1', ';']

Line # 3 properties

Operator is: Assignment Operator

Operator is: Additon Operator

1 <class 'int'>

_____

Line # 4

}

Tokens are ['}']

Line # 4 properties

Blocked Statement Body Closed

_____

RESULT: Lexical Analyzer is successfully studied and implemented.