Name : Sanket Chandrashekhar Harvande
Roll No : 19

Experiment No.- 04

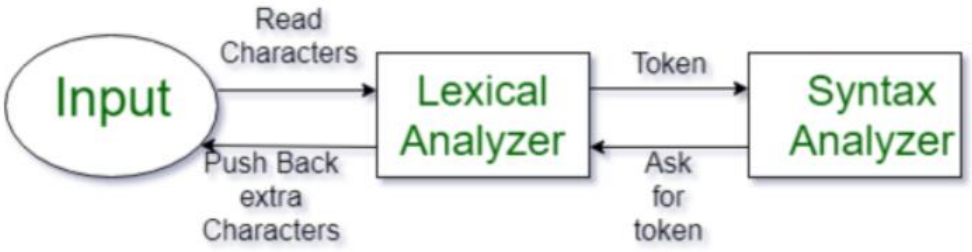
AIM : Implementation of Lexical Analyser.

Theory:

Lexical Analyser:

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of **Tokens**.

- Lexical Analysis can be implemented with the **Deterministic finite Automata**.
- The output is a sequence of tokens that is sent to the parser for syntax analysis



What is a token?
A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming languages.

Example of tokens:

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)
- Alphabetic tokens (keywords)

Keywords; Examples-for, while, if etc.

Identifier; Examples-Variable name, function name, etc.

Operators; Examples '+', '++', '-' etc.

Separators; Examples ',' ';' etc.

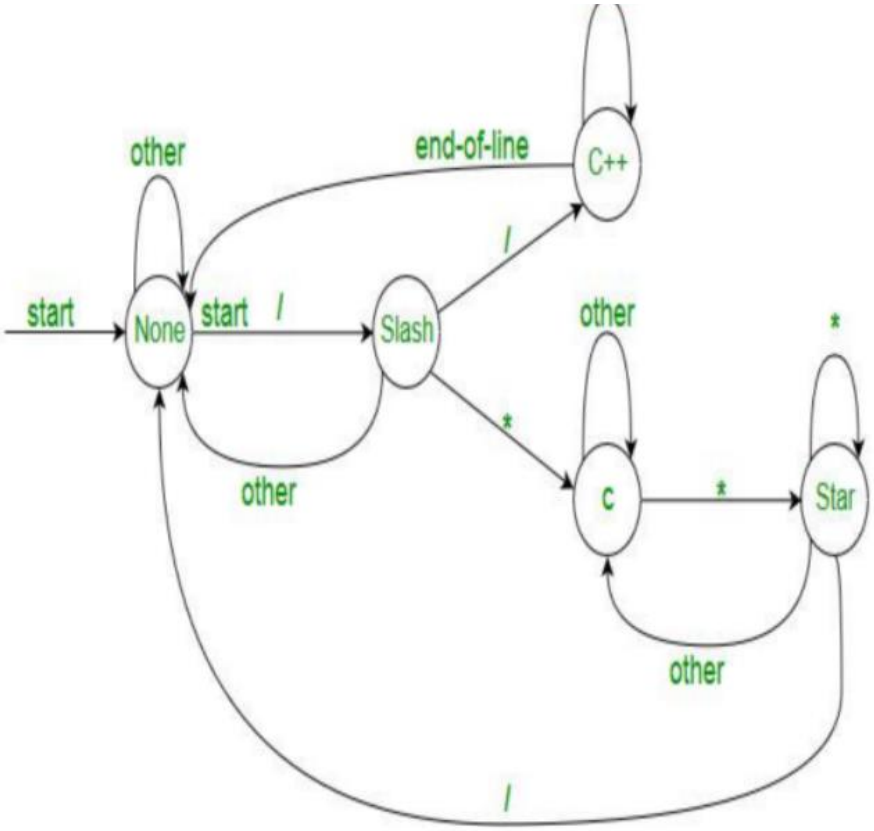
Example of Non-Tokens:

- Comments, preprocessor directive, macros, blanks, tabs, newline, etc.

Lexeme: The sequence of characters matched by a pattern to form the corresponding token or a sequence of input characters that comprises a single token is called a lexeme. eg- "float", "abs_zero_Kelvin", "=", "-", "273", ";" .

How Lexical Analyzer functions

1. Tokenization i.e. Dividing the program into valid tokens.
2. Remove white space characters.
3. Remove comments.
4. It also provides help in generating error messages by providing row numbers and column numbers.



The lexical analyzer identifies the error with the help of the automation machine and the grammar of the given language on which it is based like C, C++, and gives row number and column number of the error.

Suppose we pass a statement through lexical analyzer –

a = b + c ; It will generate token sequence like this:
id=id+id; Where each id refers to it's variable in the symbol table referencing all details

CODE:-

```
#This is lexcial analyzer for sample c program
#where we can have
#inclusion of header file
#declaration of symbolic constant
#declaration of int, float variable (give space between two varaiable name)
#you can write any printf statement
#assignment of a value to variable should not be at the time of varaiable
declration

#Sample c code
#
##include<stdio.h>
##include<conio.h>

#
#
#void main()
#   {
#   int a, b;
#   a=10;
#   b=20
#   printf("This is my first C Program");
#   printf("Value of a =%d and b=%d ", a,b);
#   }

import re

cprogram = open("cprogram.c", "r")
keyword=['void','int','float']
symbol=['(',')',',',';','{','}','<','>']
sys_ident=['main','printf','scanf']


#print(cprogram)
for line in cprogram:        #for every line in source code
    line=line.strip()       #remove begining and trailing white spaces

    if re.search('^#',line):      #if line start with # (#include / #define)
        print("# - Symbol")      #print # token
        #print(line)

        if re.search('<',line):       #if statement is #include
           ans=re.findall('[#](.+)<',line)  #find string inbetween # and < &
print it
           print(ans[0]," - Keyword")
           ans=re.findall('.+<(.+)>',line)   #find header file & print
```

```
           print(ans[0]," -Identifier")

        else:                           #if it is #define
            splitline=line.split();       #split line based on white space

            if(len(splitline)==3):        #if it has 3 parts
                ans=re.findall('[#](.+)',splitline[0]) #extract the define
word
                print(ans[0]," - Keyword")      #print them
                print(splitline[1]," -Identifier")
                print(splitline[2]," -literal")

            else:
                print("unidentified......")


    elif re.search('^printf',line):  #if printf statement
        print("printf- Identifier")  #print identifier
        start=line.find('(')
#        if start>0:
        print('(- is symbol')

        end=line.find(')')
        print(line[start+1:end],"-is string constant")  #seperate the string
constant inside
        print(')- Symbol')                              #the " "
        if line.endswith(';'):       #search for ; and also display
            print(";-Symbol")


    else:
        temp=line.split()
        #print(temp)
        if len(temp)==1:   #if line is of assignment
            if temp[0] in symbol:
                print(temp[0]," -Symbol")
                flag=0
            start=temp[0].find('=')
            if(start>0):
                print(temp[0][:start]," - Identifier")  #identify variable
name
                print("= -Assignment Operator")
                print(temp[0][start:len(temp[0])-1]," - Literal") #identify
literal
                if temp[0].endswith(';'):
                    print("; -Symbol")
        if len(temp)>1:
            for token in temp:   #for rest
```

```
                #print(token,'*****')
                if token in keyword:
                    print(token ,"-Keyword")
                    flag=1

                if flag!=1:
                    start=token.find('(')
                    end = token.find(')')
                    if end-start==1:
                        print(token[:start],"-Identifer")
                        print('(- Symbol')
                        print(')- Symbol')

                    start=token.find(';')
                    if start>0:
                        print(token[:start]," - Identifier")
                        print("; -Symbol")

                    start=token.find(',')
                    if start>0:
                        print(token[:start]," - Identifier")
                        print(", -Symbol")


                flag=0


##    else:
##        temp=line.split()
##        if (len(temp)!=0) :
##            for item in temp:
##                if re.search('[a-z]+',item):
##                    print(item, "-identified")
##                else:
##                    print(item, " -----------not identified")
##









##                    if '(' in item:
##                        ans=re.findall('[a-z]+',item)
##                        if ans[0] in ident:
##                            print(ans[0]," is a identifier")
##                        else:
##                            print(ans[0]," ******eroor******")
```

```
##                             ans=re.findall('.*"(.*)"',item)
##                         if(len(ans)!=0):
##                             print(ans[0]," is a string")
##                     else:
##                         if item in keyword:
##                             print(item, "is a keyword")
##                         elif item in symbol:
##                             print(item, "is i symbol")
##                         else:
##                             print(item," ********")
```

```
    ##         if re.search("^printf",line):
##             print("prinf is an identifier")
##             if re.search('(',line):
##                 print("( is a symbol")
##             ans=re.findall('.*"(.*)"',line)
##             if(len(ans)!=0):
##                 print(ans[0]," is a string")
```

Output: