GHARDA FOUNDATION

GHARDA INSTITUTE OF TECHNOLOGY, LAVEL

Department of Computer Engineering

Academic Year 2023-24

Evaluation Sheet

Class: **TE Computer** SEM: **VI**

Roll No.: **61**

Name: **TIWAREKAR ARYA SANJAY**

Subject: **System Programming & Compiler Construction**

Experiment No: **03**

Title of Experiment: **Implementation of Lexical Analyzer in Compiler.**

Sr. No.	Evaluation Criteria	Max Marks	Marks Obtained
1	Practical Performance	08	
2	Oral	03	
3	Timely Submission	02	
4	Neatness	02	
Total		15	

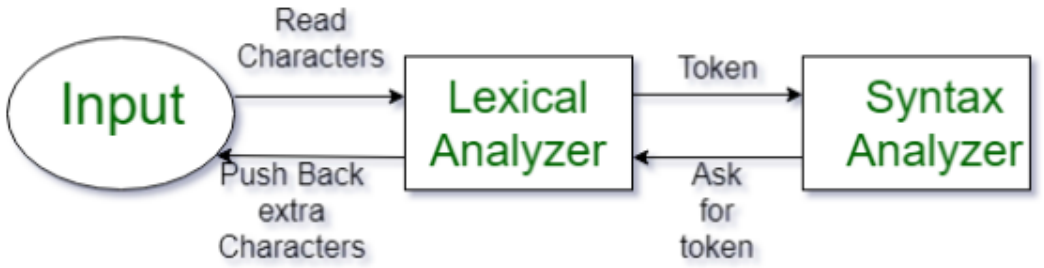
Signature of Subject Teacher

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 an 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”,“;”.

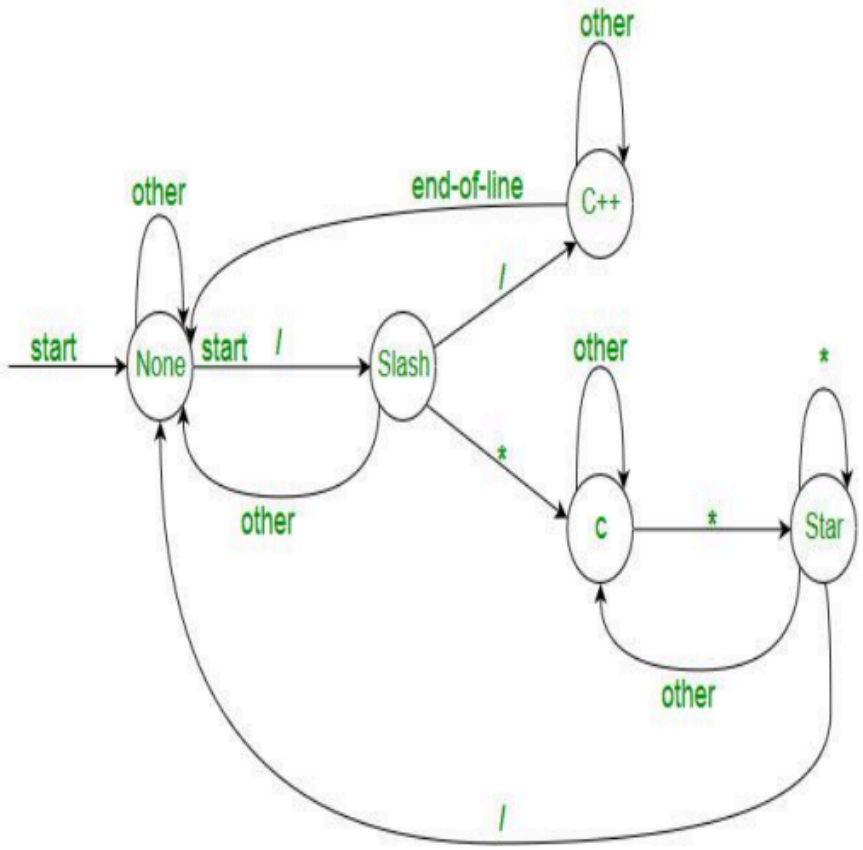
How Lexical Analyzer Function

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 its variable in the symbol table referencing all details.

For example, consider the program
int main()
{
//2 variables
int a, b;
a =10;
return 0;
}
All the valid tokens are:
'int' 'main' '(")' '{' 'int' 'a' ',' 'b' ';'
'a' '=' '10' ';' 'return' '0' ';' '}'
Above are the valid tokens.

As another example, consider the below printf statement.

There are 5 valid tokens in this printf statement.

Code:

PythonCode:

```
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
```

Input C Program 1:

```
#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);
}
```

Output 1:

```
In [3]: runfile('D:/SPCCL Arya T/Lexical_analyser.py', wdir='D:/SPCCL Arya T')
# - Symbol
include  - Keyword
stdio.h  -Identifier
# - Symbol
include  - Keyword
conio.h  -Identifier
void -Keyword
main -Identifer
(- Symbol
)- Symbol
{  -Symbol
int -Keyword
a  - Identifier
, -Symbol
b  - Identifier
; -Symbol
a  - Identifier
= -Assignment Operator
=10  - Literal
; -Symbol
b  - Identifier
= -Assignment Operator
=20  - Literal
; -Symbol
printf- Identifier
(- is symbol
"This is my first C Program" -is string constant
)- Symbol
;-Symbol
printf- Identifier
(- is symbol
"Value of a =%d and b=%d ", a,b -is string constant
)- Symbol
;-Symbol
}  -Symbol

In [4]:
```

Input C program 2:

```
#include<stdio.h>
#include<conio.h>

void main()
{
        float a, b, c;
        c = a + b;
        printf("%d", c);
        printf("Hello World");
}
```

Output 2:

```
IPdb [3]: runfile('D:/SPCCL Arya T/Lexical_analyser.py', wdir='D:/SPCCL Arya T')
# - Symbol
include  - Keyword
stdio.h  -Identifier
# - Symbol
include  - Keyword
conio.h  -Identifier
void -Keyword
main -Identifer
(- Symbol
)- Symbol
{  -Symbol
float -Keyword
a  - Identifier
, -Symbol
b  - Identifier
, -Symbol
c  - Identifier
; -Symbol
b  - Identifier
; -Symbol
printf- Identifier
(- is symbol
"%d", c -is string constant
)- Symbol
;-Symbol
printf- Identifier
(- is symbol
"Hello World" -is string constant
)- Symbol
;-Symbol
}  -Symbol

IPdb [4]:
```

Conclusion:

Thus, we understood and successfully implemented Lexical Analyser.