Aim

To check the different keywords and operators (tokens) in the given grammar/ Generate Lexical Analyzer (tokens from given source program)

Program Logic

1. Read the input Expression
2. Check whether input is alphabet or digits then store it as identifier
3. If the input contains operator store it as in symbol Table.
4. Check the input for keywords.
5. Check for special symbols from source program

Lab Assignment

1. What is token?

A token is a group of characters having collective meaning: typically, a word or punctuation mark, separated by a lexical analyser and passed to a parser. A lexeme is an actual character sequence forming a specific instance of a token, such as num. The pattern matches each string in the set.

2. What is lexeme?

A lexeme is a sequence of alphanumeric characters in a token. The term is used in both the study of language and in the lexical analysis of computer program compilation. In the context of computer programming, lexemes are part of the input stream from which tokens are identified.

3. What is the difference between token and lexeme?

Token: Token is a sequence of characters that can be treated as a single logical entity. Typical tokens are,

- Identifiers
- keywords
- operators
- special symbols
- constants

Lexeme: A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Example: while(y > = t) = y -3

Will be represented by the set of pairs.

Lexeme	**Token**
while	while
(	lparen
y	identifier

>=	Comparison
t	identifier
)	Rparen
y	identifier
=	Assignment
y	identifier
-	Arithmetic
3	integer
;	Finish of a statement

4. Define phase and pass?

Pass: A pass refers to the traversal of a compiler through the entire program.

Phase: A phase of a compiler is a distinguishable stage, which takes input from the previous stage, processes and yields output that can be used as input for the next stage. A pass can have more than one phase.

5. What is the difference between phase and pass?

No.	Phase	Pass
1	The process of compilation is carried out in various step is called phase.	Various phases are logically grouped together to form a pass.
2	The phases of compilation are lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization and code generation.	The process of compilation can be carried out in a single pass or in multiple passes.

6. What is the difference between compiler and interpreter?

Compiler: It is a translator who takes input i.e., High-Level Language, and produces an output of low-level language i.e., machine or assembly language.

- A compiler is more intelligent than an assembler it checks all kinds of limits, ranges, errors, etc.
- But its program run time is more and occupies a larger part of memory. It has slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.

Interpreter: An interpreter is a program that translates a programming language into a comprehensible language. –

- It translates only one statement of the program at a time.
- Interpreters often are smaller than compilers.

Compiler	Interpreter
Compiler scans the whole program in one go.	Translates program one statement at a time.
As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
Main advantage of compilers is its execution time.	Due to interpreters being slow in executing the object code, it is preferred less.
It converts the source code into object code.	It does not convert source code into object code instead it scans it line by line
It does not require source code for later execution.	It requires source code for later execution.
C, C++, C# etc.	Python, Ruby, Perl, SNOBOL, MATLAB, etc.

7. What is 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

Lab Assignment Program

Write a program to recognize:

1. Identifiers
2. Constants
3. Keywords

Code

Read.py

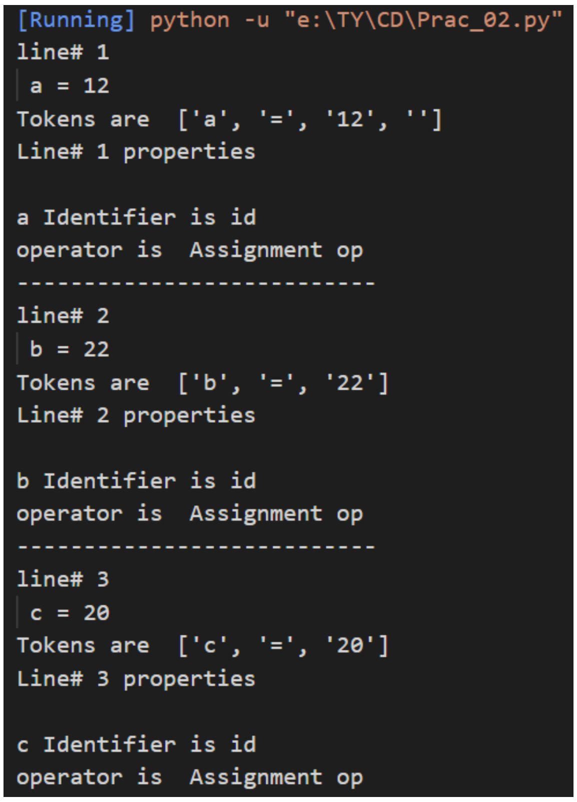
```
a = 12
b = 22
c = 20
d = a + b
m = 25
n = 10
j = m + n
```

Prac_02.py

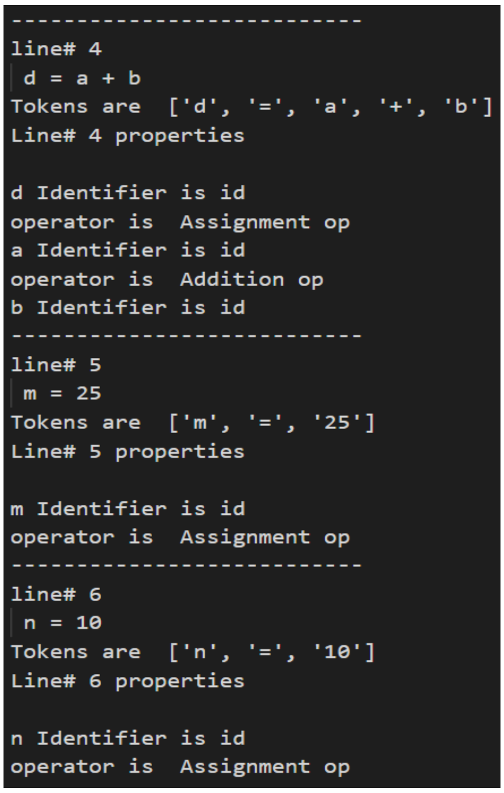
```
import re

file = open("read.py")

operators = {
        '=': 'Assignment op','+': 'Addition op','-
': 'Subtraction op','/' : 'Division op'
        ,'*': 'Multiplication op','<': 'Lessthan op','>': 'Greaterthan op'}
operators_key = operators.keys()
```

```
data_type = {'int' : 'integer type', 'float': 'Floating point' , 'char' : 'Cha
racter type', 'long' : 'long int' }
data_type_key = data_type.keys()

punctuation_symbol = { ':' : 'colon', ';' : 'semi-
colon', '.' : 'dot' , ',' : 'comma' }
punctuation_symbol_key = punctuation_symbol.keys()

identifier = {
      'a':'id','b':'id','c':'id','d':'id','e':'id'
      ,'f':'id','g' :'id','h':'id','i':'id','j':'id'
      ,'k':'id','l':'id','m':'id','n':'id','o':'id'
      ,'p':'id','q':'id','r':'id','t':'id','u':'id'
      ,'v':'id','w':'id','x':'id','y':'id','z':'id'}

identifier_key = identifier.keys()

dataFlag = False

a=file.read()

count=0
program = a.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 token in operators_key:
            print("operator is ", operators[token])
        if token in data_type_key:
            print("datatype is", data_type[token])
        if token in punctuation_symbol_key:
            print (token, "Punctuation symbol is" , punctuation_symbol[token])
        if token in identifier_key:
            print (token, "Identifier is" , identifier[token])


    dataFlag=False
    print("---------------------------")
```

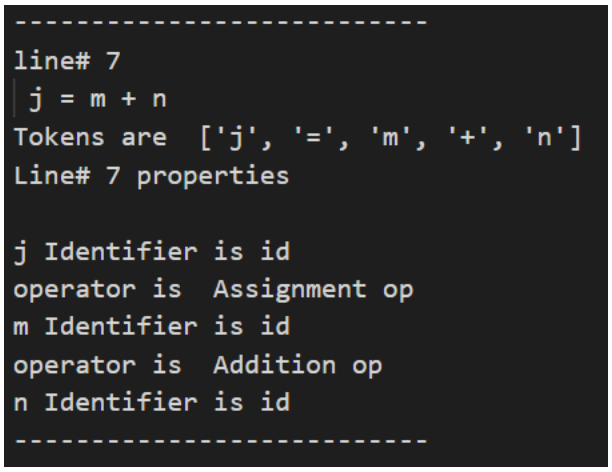
Output



```
[Running] python -u "e:\TY\CD\Prac_02.py"
line# 1
 a = 12
Tokens are  ['a', '=', '12', '']
Line# 1 properties

a Identifier is id
operator is  Assignment op
---------------------------
line# 2
 b = 22
Tokens are  ['b', '=', '22']
Line# 2 properties

b Identifier is id
operator is  Assignment op
---------------------------
line# 3
 c = 20
Tokens are  ['c', '=', '20']
Line# 3 properties

c Identifier is id
operator is  Assignment op
```



```
---------------------------
line# 4
 d = a + b
Tokens are  ['d', '=', 'a', '+', 'b']
Line# 4 properties

d Identifier is id
operator is  Assignment op
a Identifier is id
operator is  Addition op
b Identifier is id
---------------------------
line# 5
 m = 25
Tokens are  ['m', '=', '25']
Line# 5 properties

m Identifier is id
operator is  Assignment op
---------------------------
line# 6
 n = 10
Tokens are  ['n', '=', '10']
Line# 6 properties

n Identifier is id
operator is  Assignment op
```



```
---------------------------
line# 7
 j = m + n
Tokens are  ['j', '=', 'm', '+', 'n']
Line# 7 properties

j Identifier is id
operator is  Assignment op
m Identifier is id
operator is  Addition op
n Identifier is id
---------------------------
```

Conclusion

Hence, we were able to generate a Lexical Analyzer.