****

Name: Anish Vaidya
Roll: 62
Div: A
SRN: 201900160

****

COMPILER DESIGN ASSIGN-1: Design a Lexical analyzer for the subset of Java Language. Read input from the file. Also create symbol table.Detect any one lexical error.Output in 4 columns Line No, Lexeme, Token and Token Value.

Input:

```
class HelloWorld {
   // This is a Comment
   public static void main(String[] args) {
      int a,34mean,b;
      /*This is comment*/
      a = 25;
      System.out.println("Hello, World");
      b = 'qwer ty';
      /*An unterminated comment
      a = "This is an unterminated string;
   }
}
```

Output:

Line No, Lexeme, Token and Token Value:

```
Anish Vaidya Roll-62
Compiler Design Assignment-1
Design a Lexical analyzer for the subset of Java Language.

Token Table
```

Line	Lexeme	Token	Token Value
1	class	Keyword	['KW', 7]
1	HelloWorld	Identifier	['ID', 1]
1	{	Delimiter	['DL', 1]
3	public	Keyword	['KW', 25]
3	static	Keyword	['KW', 28]
3	void	Keyword	['KW', 49]
3	main	Identifier	['ID', 2]
3	(	Delimiter	['DL', 3]
3	String[]	Identifier	['ID', 3]
3	args	Identifier	['ID', 4]
3	)	Delimiter	['DL', 4]
3	{	Delimiter	['DL', 1]
4	int	Keyword	['KW', 40]
4	a	Identifier	['ID', 5]
4	,	Delimiter	['DL', 5]
4	34mean	Identifier	['ID', 6]
4	,	Delimiter	['DL', 5]
6	=	Operator	['OP', 6]
6	25	Number	['C', '25']
6	;	Delimiter	['DL', 6]
7	System	Identifier	['ID', 8]
7	.	Operator	['OP', 17]
7	out	Identifier	['ID', 9]
7	.	Operator	['OP', 17]
7	println	Identifier	['ID', 10]
7	(	Delimiter	['DL', 3]
7	"	Delimiter	['DL', 8]
7	Hello, World	String Constant	['C', 'Hello, World']
7	"	Delimiter	['DL', 8]
7	)	Delimiter	['DL', 4]
7	;	Delimiter	['DL', 6]
8	b	Identifier	['ID', 7]
8	=	Operator	['OP', 6]
8	'	Delimiter	['DL', 7]
8	qwer ty	String Constant	['C', 'qwer ty']
8	'	Delimiter	['DL', 7]
8	;	Delimiter	['DL', 6]
10	a	Identifier	['ID', 5]
10	=	Operator	['OP', 6]
10	"	Delimiter	['DL', 8]
10	This is an unterminated string;	String Constant	['C', 'This is an unterminated string;']
11	}	Delimiter	['DL', 2]
12	}	Delimiter	['DL', 2]

Symbol table:

```
Symbol Table
+-------+------------+
| Index |    Name    |
+-------+------------+
|   1   | HelloWorld |
|   2   |    main    |
|   3   |  String[]  |
|   4   |    args    |
|   5   |     a      |
|   6   |   34mean   |
|   7   |     b      |
|   8   |   System   |
|   9   |    out     |
|   10  |  println   |
+-------+------------+
```

Errors:

```
Errors
+------+----------------------------------+-------------------------+
| Line |              Lexeme              |          Error          |
+------+----------------------------------+-------------------------+
|  4   |              34mean              | Invalid Identifier name |
|  9   |    /*An unterminated comment     |   Unterminated Comment  |
|  10  | "This is an unterminated string; |   Unterminated String   |
+------+----------------------------------+-------------------------+
```

Source Code:

```
from prettytable import PrettyTable

delimiter = ["{","}","(",")",",",";"]
string_delimiter = {"":7,"":8}
operator = ["-","+","*","/","%","=","++","--","<",">","<<",">>","==",">=","<=","&","."]
keyword =
["abstract","assert","boolean","byte","case","catch","class","default","do","enum","
```

```
extends","final","finally","implements","import","instanceof","interface","long","native","new","null","package","private","protected","public","return","short","static","strictfp","super","switch","synchronized","this","throw","throws","transient","try","volatile","while","int","if","char","break","continue","double","else","float","for","void"]

with open("javaprgm.txt","r") as program:
    prog = program.read().split("\n")

space_free_prog = []
found = False
final_table = PrettyTable(["Line","Lexeme","Token","Token Value"])
st = []
symbol_table = PrettyTable(["Index","Name"])
error_table = PrettyTable(["Line","Lexeme","Error"])
str=""
index = ""

def remove_space(lista):
    listb = []
    for ele in lista:
        if ele != " ":
            listb.append(ele)
        else:
            listb.append("`")
    return listb

def find_index(tstr):
    if tstr in st:
        return st.index(tstr)+1
    else:
        st.append(tstr)
        return st.index(tstr)+1
```

```
for p in prog:
    p=list(p)
    p=remove_space(p)
    space_free_prog.append(p)

for line_index,line in enumerate(space_free_prog):
    str=""
    for word_index,word in enumerate(line):
        if found:
            str=""
            found = not found

        if len(str) > 0 and (str[0] in ("'",'"') or str[0:2] == "/*"):
            if word == "`":
                str += " "
            else:
                str += word
        elif word != "`":
            str += word

        if len(str) > 0:
            if str[:2] == "/*":
                if str[-2:] == "*/":
                    found = not found
                    continue
                else:
                    continue
            if str[:2] == "//":
                found = not found
                break

            if len(str) >= 2 and str[0] in ("'",'"') and str[0] == str[-1]:
```

```
final_table.add_row([line_index+1,str[0],"Delimiter",["DL",string_delimiter[str[0
]]]])
            # index = find_index(str[1:-1])
            final_table.add_row([line_index+1,str[1:-1],"String
Constant",["C",str[1:-1]]])

final_table.add_row([line_index+1,str[-1],"Delimiter",["DL",string_delimiter[str[-
1]]]])
            found = not found
            continue

        if str.isnumeric() and not line[word_index + 1].isnumeric() :
            if(line[word_index+1] in operator or line[word_index+1] in
delimiter):
                final_table.add_row([line_index+1,str,"Number",["C",str]])
                found = not found
                continue

        elif str in keyword:

final_table.add_row([line_index+1,str,"Keyword",["KW",keyword.index(str)+1]]
)
            found = not found
            continue

        elif str in delimiter:

final_table.add_row([line_index+1,str,"Delimiter",["DL",delimiter.index(str[0])+
1]])
            found = not found
            continue
```

```
        elif str in operator:
            try:
                if line[word_index+1] in operator:
                    continue
                else:

final_table.add_row([line_index+1,str,"Operator",["OP",operator.index(str)+1]]
)
                    found = not found
                    continue
            except:
                pass

        try:
            if line[word_index+1] in operator or line[word_index+1] in delimiter
or line[word_index+1] == "`":
                if str[len(str)-1] in ("'",'"'):
                    error_table.add_row([line_index+1,str,"Unterminated String"])
                    # index = find_index(str[:-1])
                    final_table.add_row([line_index+1,str[:-1],"String
Constant",["C",str[:-1]]])

final_table.add_row([line_index+1,str[-1],"Delimiter",["DL",string_delimiter[str[-
1]]]])
                    found = not found
                    continue
                if str[0] not in ("'",'"'):
                    if( str[0].isnumeric()):
                        error_table.add_row([line_index+1,str,"Invalid Identifier
name"])
                    elif(len(str) > 20):
                        error_table.add_row([line_index+1,str,"Identifier character
limit exceeded"])
```

```
                index = find_index(str)
                final_table.add_row([line_index+1,str,"Identifier",["ID",index]])
                found = not found
                continue
        except:
            pass
    if not found:
        if str[0] in ("'",""):
            error_table.add_row([line_index+1,str,"Unterminated String"])

final_table.add_row([line_index+1,str[0],"Delimiter",["DL",string_delimiter[str[0
]]]])
            # index = find_index(str[1:])
            final_table.add_row([line_index+1,str[1:],"String Constant",["C",str[1:]]])
            found = not found
        if str[:2] == "/*":
            error_table.add_row([line_index+1,str,"Unterminated Comment"])



print("\n\n")
print("\033[32mAnish Vaidya Roll-62\033[0m")
print("\033[32mCompiler Design Assignment-1\033[0m")
print("\033[33mDesign a Lexical analyzer for the subset of Java
Language.\033[0m")
print("\nToken Table")
print(final_table)
print("\n")

for entry_index,entry in enumerate(st):
    symbol_table.add_row([entry_index+1,entry])
print("Symbol Table")
print(symbol_table)
```

```
print("\n")

print("Errors")
print(error_table)
```