Name: Mihir Thakkar
Class: TY A

Roll no: 59

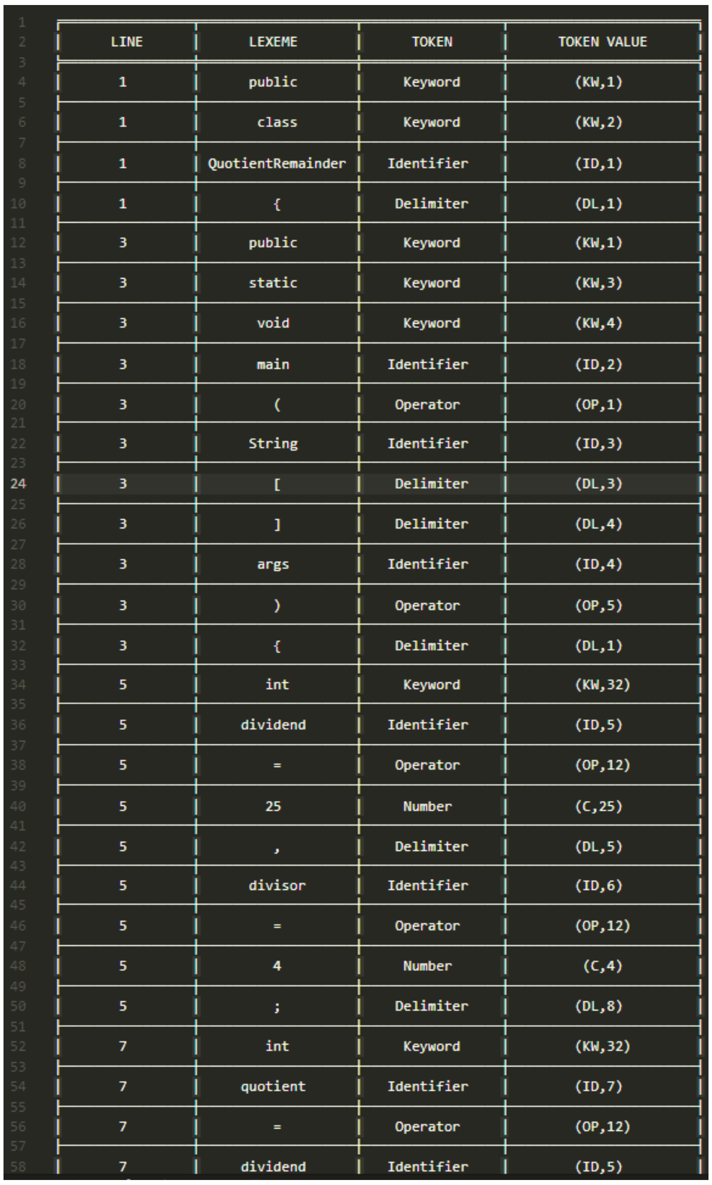
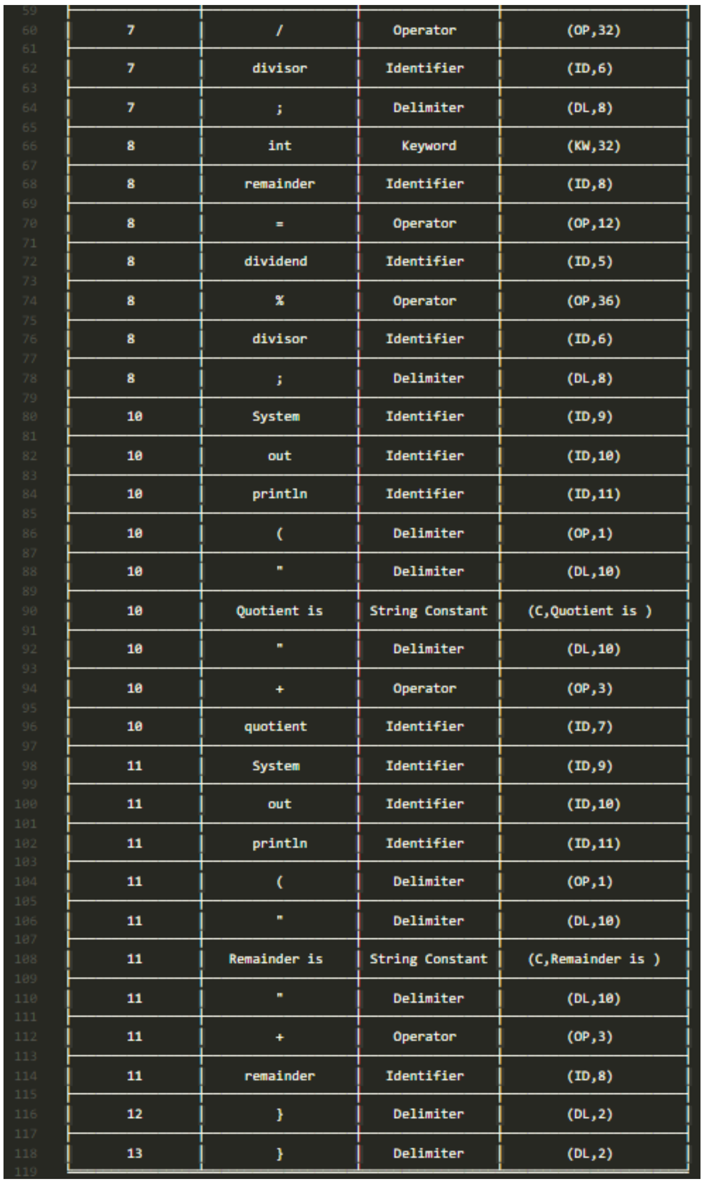
SRN: 201901267

Compiler Design Assignment-1
Problem Statement:
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. Upload single file containing input, output and source code.

Input code(java):

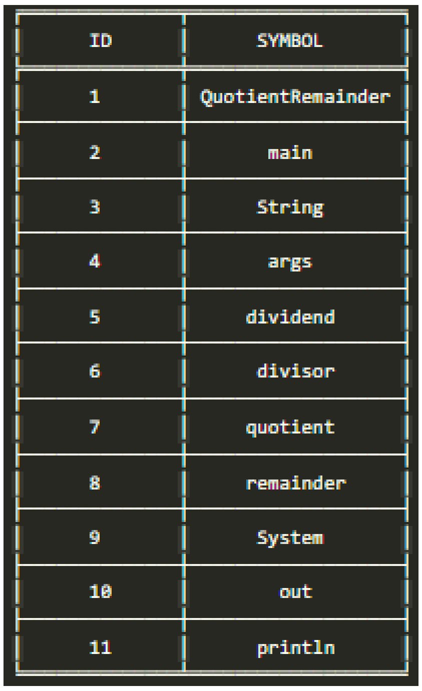
```
public class QuotientRemainder {


  public static void main(String [ ] args) {


    int dividend = 25, divisor = 4;


    int quotient = dividend / divisor;
    int remainder = dividend % divisor;


    System.out.println("Quotient is " + quotient);
    System.out.println("Remainder is " + remainder);
  }
}
```

Output:

LINE	LEXEME	TOKEN	TOKEN VALUE
1	public	Keyword	(KW,1)
1	class	Keyword	(KW,2)
1	QuotientRemainder	Identifier	(ID,1)
1	{	Delimiter	(DL,1)
3	public	Keyword	(KW,1)
3	static	Keyword	(KW,3)
3	void	Keyword	(KW,4)
3	main	Identifier	(ID,2)
3	(	Operator	(OP,1)
3	String	Identifier	(ID,3)
3	[	Delimiter	(DL,3)
3	]	Delimiter	(DL,4)
3	args	Identifier	(ID,4)
3	)	Operator	(OP,5)
3	{	Delimiter	(DL,1)
5	int	Keyword	(KW,32)
5	dividend	Identifier	(ID,5)
5	=	Operator	(OP,12)
5	25	Number	(C,25)
5	,	Delimiter	(DL,5)
5	divisor	Identifier	(ID,6)
5	=	Operator	(OP,12)
5	4	Number	(C,4)
5	;	Delimiter	(DL,8)
7	int	Keyword	(KW,32)
7	quotient	Identifier	(ID,7)
7	=	Operator	(OP,12)
7	dividend	Identifier	(ID,5)
7	/	Operator	(OP,32)
7	divisor	Identifier	(ID,6)
7	;	Delimiter	(DL,8)
8	int	Keyword	(KW,32)
8	remainder	Identifier	(ID,8)
8	=	Operator	(OP,12)
8	dividend	Identifier	(ID,5)
8	%	Operator	(OP,36)
8	divisor	Identifier	(ID,6)
8	;	Delimiter	(DL,8)
10	System	Identifier	(ID,9)
10	out	Identifier	(ID,10)
10	println	Identifier	(ID,11)
10	(	Delimiter	(OP,1)
10	"	Delimiter	(DL,10)
10	Quotient is	String Constant	(C,Quotient is)
10	"	Delimiter	(DL,10)
10	+	Operator	(OP,3)
10	quotient	Identifier	(ID,7)
11	System	Identifier	(ID,9)
11	out	Identifier	(ID,10)
11	println	Identifier	(ID,11)
11	(	Delimiter	(OP,1)
11	"	Delimiter	(DL,10)
11	Remainder is	String Constant	(C,Remainder is)
11	"	Delimiter	(DL,10)
11	+	Operator	(OP,3)
11	remainder	Identifier	(ID,8)
12	}	Delimiter	(DL,2)
13	}	Delimiter	(DL,2)

SYMBOL TABLE:



ID	SYMBOL
1	QuotientRemainder
2	main
3	String
4	args
5	dividend
6	divisor
7	quotient
8	remainder
9	System
10	out
11	println

Source Code(py):

```
import re

import sys

import pandas as pd

from collections import Counter

from tabulate import tabulate

from colorama import Fore, Back, Style  #To change error colour to red

import nltk


input_program=open(r"/home/rishabh/Sem6/CD/cd_ass1.txt")

#print(input_program)

list_1 = input_program.readlines()

for n in range(0,len(list_1)):

   list_1[n]=list_1[n].strip('\n')

   # print(list_1)

# print(list_1)


Keywords =
["public","class","static","void","abstract","assert","boolean","break","byte","case","catch","char","c
lass","continue","const","default","do","double","else","enum","exports","extends","final","finally",
"float","for","goto","if","implements","import","instanceof","int","interface","long","module","nativ
e","new","package","private"]

# Keywords=RE_Keywords.split("|")

Operators = "+|++|-|=|*|/|%|--|<=|>=|."

RE_Operators = Operators.split("|")

# print(RE_Operators)

RE_Constants=[]

for num in range(0,100001):

   RE_Constants.append(str(num))

RE_Delimiter = ["{","}","[","]",",","(",")",";","'"]

# quote = '\"'

# add_q = "" + quote
# RE_Delimiter.append(add_q)
# print(RE_Delimiter)
RE_Identifiers = "^[a-zA-Z_]+[a-zA-Z0-9_]*"
RE_Illegal = ["@","$"]

def check_code(file):
   line =[]
   code = []
   type =[]
   tok_id=[]
   s_id = 1
   sym_line = []
   sym_name = []
   sym_id = []
   token_value = {}

   # print(file)
   for i in range(0,len(file)):
       n = i+1
       # print(n)
       input_program_tokens = nltk.wordpunct_tokenize(file[i])
       # print(input_program_tokens)
       # print(input_program_tokens)
       for token in input_program_tokens:
           if(token in Keywords):
               # print(token , "-------> Keyword")
               if not(token in token_value.keys()):
                   token_value[token] = ["KW",Keywords.index(token)+1]
               code.append(token)
               type.append("Keyword")
               line.append(n)
               s1 = "(" + token_value[token][0] + "," + str(token_value[token][1]) + ")"
               tok_id.append(s1)
           elif(token in RE_Operators):
               for i in token:
                   if i in RE_Operators:
                       # print(i, "-------> Operator")
                       if not (i in token_value.keys()):
                           token_value[i] = ["OP", RE_Operators.index(i) + 1]
                       code.append(i)
                       type.append("Operator")
                       line.append(n)
                       s1 = "("+token_value[i][0]+","+str(token_value[i][1])+")"
                       tok_id.append(s1)
           elif(token in RE_Constants):
               # print(token, "-------> Number")
               code.append(token)
               type.append("Number")
               line.append(n)
               s1="(C,"+token+")"
               tok_id.append(s1)
           elif(token in RE_Delimiter or '"' in token):
               # print(token)
               for i in token:
                   # print(i)
                   if True:
                       if i in RE_Delimiter and not(i == '"'):
                           # print(i, "-------> Delimiter")
                           if not (i in token_value.keys()):
                               token_value[i] = ["DL", RE_Delimiter.index(i) + 1]
                           code.append(i)
                           type.append("Delimiter")
           line.append(n)
           s1 = "(" + token_value[i][0] + "," + str(token_value[i][1]) + ")"
           tok_id.append(s1)
else:
    if not (i in token_value.keys()):
        token_value[i] = ["DL", len(RE_Delimiter)+1]
    code.append(i)
    type.append("Delimiter")
    line.append(n)
    s1 = "(" + token_value[i][0] + "," + str(token_value[i][1]) + ")"
    tok_id.append(s1)
    s = ""
    test = 0
    test2 = 0
    q_err = 1
    const_ind=[]
    for k in range(input_program_tokens.index(token)+1,len(input_program_tokens)):
        if test == 1:
            break
        else:
            # print(input_program_tokens[k])
            if test2 == 1:
                const_ind.append(k-1)
                test2 = 0
            temp = list(input_program_tokens[k])
            # print(temp)
            for l in temp:
                # print(l)
                if l != '"':
                    s = s + l
                else:
```

```
                code.append(s)
                type.append("String Constant")
                line.append(n)
                s1 = "(C," + s + ")"
                tok_id.append(s1)
                code.append(l)
                type.append("Delimiter")
                line.append(n)
                s1 = "(" + token_value[l][0] + "," + str(token_value[l][1]) + ")"
                tok_id.append(s1)
                temp.pop(temp.index(l))
                # print(temp)
                str1=""
                for z in temp:
                    str1 = str1+z
                input_program_tokens[k]=str1
                # print(input_program_tokens[k])
                test = 1
                q_err=0
                break
            test2 = 1
            # print(s)
        s = s + " "
        # print(input_program_tokens[k])
        # input_program_tokens.pop(k)

            # input_program_tokens[k].pop(l)
# print(const_ind)
if q_err==1:
    print("Error : Unterminated String")
    sys.exit(0)
```

```
            for l in range(len(const_ind)-1,-1,-1):
                # print(l)
                input_program_tokens.pop(const_ind[l])

        elif i in RE_Operators:
            # print(i, "-------> Operator")
            if not (i in token_value.keys()):
                token_value[i] = ["OP", RE_Operators.index(i) + 1]
            code.append(i)
            type.append("Operator")
            line.append(n)
            s1 = "(" + token_value[i][0] + "," + str(token_value[i][1]) + ")"
            tok_id.append(s1)

elif(re.findall(RE_Identifiers,token)):
    # print(token, "-------> Identifiers")
    if not(token in sym_name):
        sym_name.append(token)
        sym_id.append(s_id)
        s_id = s_id + 1
    if not (token in token_value.keys()):
        token_value[token] = ["ID", sym_id[-1]]
    code.append(token)
    type.append("Identifier")
    line.append(n)
    s1 = "(" + token_value[token][0] + "," + str(token_value[token][1]) + ")"
    tok_id.append(s1)
    # print(code)
    # print(type)
    display(line,code,type,tok_id)
    sym_print(sym_id,sym_name)

def display(line,code,type,value):
    head = ["    LINE    ","    LEXEME    ","    TOKEN    ","    TOKEN VALUE    "]
    tab = []
    for i in range(0,len(code)):
        tab.append([line[i],code[i],type[i],value[i]])

    # print(tab)
    print(tabulate(tab, headers=head,
colalign=("center","center","center","center"),tablefmt="fancy_grid"))

def sym_print(id,code):
    head = ["    ID    ","    LEXEME    "]
    tab = []
    for i in range(0,len(code)):
        tab.append([id[i],code[i]])

    # print(tab)
    print(tabulate(tab, headers=head, colalign=("center","center"),tablefmt="fancy_grid"))

check_code(list_1)
```