LEXICAL ANALYSIS

Name: Ananda Mohon Ghosh

ID: 110201

Source Code:

TestingLibrary.py

```
#Name:Ananda Mohon Ghosh
#ID:110201
#Language: Python
#Variables: array_keywords, array_data_type,
array_operator,array_bracket,array_logical_operator,array_delemiter,array_var_separator,array_co
mments,arry_floatpoint
#User Define Function: is_operator(self, temp_char),is_delimeter(self, temp_char),
is_comments(self, temp_char,next_char),is_ogical_operator(self, temp_char),is_bracket(self,
temp_char)
#is_var_separator(self, temp_char),is_whitespace(self, temp_char),is_data_type(self,
temp_string),is_keywords(self, temp_string),is_floatingpoint(self, temp_string)


__author__ = 'LazYCodeR'
class Testing:
    array_keywords =
['case','catch','class','continue','default','do','else','final','finally','for','if','native','
new','public','return',
'static','switch','this','throw','throws','try','void','while','false','true','null','printf','s
canf']
    array_data_type=['boolean','byte','char','float','int','short','double','long','string']
    array_operator = ['/','+','-','*','%','<','>','<','>','=','&','|',' ','!']
    array_bracket =['(',')','{','}','[',']']
    array_logical_operator = ["++","--","<=",">=","==","&&","||","*=","/=","+=","-=","!="]
    array_delemiter=[';']
    array_var_separator=[',']
    array_comments =['//','/*','*/']
    arry_floatpoint=['.','_']

    def is_operator(self, temp_char):
        if  temp_char in self.array_operator:
            return 1
        return 0
    def is_delimeter(self, temp_char):
        if(temp_char   in self.array_delemiter):
            return 1
        return 0
    def is_comments(self, temp_char,next_char):
        temp_char = temp_char+next_char
        if (temp_char  in self.array_comments):
            return 1
        return 0
    def is_ogical_operator(self, temp_char):
        if (temp_char in self.array_logical_operator):
            return 1
        return 0
    def is_bracket(self, temp_char):
        if (temp_char in self.array_bracket):
            return 1
        return 0

    def is_var_separator(self, temp_char):
        if (temp_char  in self.array_var_separator):
            return 1
        return 0
    def is_whitespace(self, temp_char):
        if (temp_char in self.array_whitespace):
            return 1
        return 0
    def is_data_type(self, temp_string):
        if (temp_string  in self.array_data_type):
            return 1
        return 0
    def is_keywords(self, temp_string):
        if (temp_string  in self.array_keywords):
            return 1
        return 0
    def is_floatingpoint(self, temp_string):
        if (temp_string  in self.arry_floatpoint):
            return 1
        return 0
```

MainClass.py

```
#Name:Ananda Mohon Ghosh
#ID:110201
#Language: Python
#Variables: result_array_symbolic_table, result_array_token_table,
value_dictionary,data_type,temp_char,temp_string,line
#User Define Function: token(cls, string)  -- To insert into Symbolic table & Token Table
#User Define Function:  main_method()     --- Main Method. Execution will be start from Here
#

__author__ = 'LazYCodeR'
class Mainclass:
    result_array_symbolic_table=[]
    result_array_token_table=[]
    value_dictionary={"(":"opening parenthesis",")":"closing parenthesis","[":"Left
brace","]":"Right  brace","{":"Left Curly brace","}":"Right Curly brace" ,"+":"addition","-
":"subtraction","/":"division", "*":"Multiplication",
"%":"Modulo","=":"Assignment","==":"Compare To","++":"increment by 1","--":"Decrement by
1",">":"Grater Than","<":"less Than",">=":"Grater Than Equal","<=":"less Than
Equal","&":"bitwise AND","|":"Bit Wise OR","&&":"logical AND","||":"Logical OR"}
    data_type=''
    @classmethod
    def token(cls, string):
        from  TestingLibrary import Testing
        testinglib2=Testing();
        if(testinglib2.is_data_type(string)):
            #print("datatype")
            Mainclass.result_array_token_table.append(string+"            "+string+"
--    ")
            Mainclass.data_type=string
        elif(testinglib2.is_keywords(string)):
            #print("key")
            Mainclass.result_array_token_table.append(string+"            "+string+"
--  ")
            Mainclass.data_type=''
        elif(testinglib2.is_operator(string)):
            #print("ope")
            Mainclass.result_array_token_table.append(string+"             "+"Operator
"+Mainclass.value_dictionary[string])
            Mainclass.data_type=''
        elif(testinglib2.is_ogical_operator(string)):
            #print("lop")
            Mainclass.result_array_token_table.append(string+"            "+"Operator
"+Mainclass.value_dictionary[string])
            Mainclass.data_type=''
        elif(testinglib2.is_bracket(string)):
            #print("bra")
            Mainclass.result_array_token_table.append(string+"            "+"Special Sysmbol
"+Mainclass.value_dictionary[string])
            Mainclass.data_type=''
        elif(testinglib2.is_delimeter(string)):
            #print("spec")
            Mainclass.result_array_token_table.append(string+"             "+"Special Sysmbol
"+"   Semiclone")
            Mainclass.data_type=''
        elif(testinglib2.is_var_separator(string)):
            #print("spec")
            Mainclass.result_array_token_table.append(string+"           "+"Special Sysmbol"+"
Comma")
            # Mainclass.data_type=''
        else:
            if(string!=''):
                #print(string+"  coming-------------")
                if(string.isdigit() or '.' in string):
                    Mainclass.result_array_token_table.append(string+"             "+"Numberl"+"
Constant")
                    Mainclass.data_type=''
                    #print(string+"  cod-------------Digit")

                elif('_' in string):
                    #print(string+"  cod-------------__")
                    Mainclass.result_array_token_table.append(string+"                "+"id"+"
Pointer To Symbolic Table")
                    if(Mainclass.data_type!=''):
```

```
                        Mainclass.result_array_symbolic_table.append(string+"           id
"+Mainclass.data_type)
                else:
                    if(string=='main'):
                        Mainclass.result_array_token_table.append(string+"
"+"id"+"                    --")
                        if(Mainclass.data_type!=''):
                            Mainclass.result_array_symbolic_table.append(string+"
id                  --")
                            #print(string+" cod2-------------")
                    else:
                        Mainclass.result_array_token_table.append(string+"
"+"id"+"              Pointer To Symbolic Table")
                        if(Mainclass.data_type!=''):
                            Mainclass.result_array_symbolic_table.append(string+"           id
"+Mainclass.data_type)
                            #print(string+" cod2-------------")
```

```
    @staticmethod
    def main_method():
        file = open("input.txt")
        from  TestingLibrary import Testing
        testinglib=Testing();
        temp_string='';
        while 1:
            line = file.readline()
            if not line:
                break;
            else:
                x =0;

                #print(line)
                while(x<line.__len__()):
                    temp_char =line[x]

                    if(temp_char.isdigit()):
                        temp_string +=temp_char

                    elif(temp_char.isalpha()):
                        temp_string +=temp_char
                    elif(testinglib.is_floatingpoint(temp_char)):
                        temp_string +=temp_char
                    elif(testinglib.is_delimeter(temp_char)):
                            #print(temp_string);
                            Mainclass.token(temp_string)
                            temp_string=''
                            #print temp_char;
                            Mainclass.token(temp_char)
                            temp_char=''

                    else:
                        if(temp_char.isspace()):
                            #print(temp_string)
                            Mainclass.token(temp_string)
                            temp_string = ''
                            temp_char =''

                        elif(testinglib.is_var_separator(temp_char)):
                            #print(temp_string);
                            Mainclass.token(temp_string)
                            temp_string = ''
                            Mainclass.token(temp_char)
                            temp_char =''

                        elif(testinglib.is_bracket(temp_char)):
                            #print(temp_string);
                            Mainclass.token(temp_string)
                            temp_string = ''
                            #print(temp_char);
```

```
                            Mainclass.token(temp_char)
                            temp_char =''
                        elif(testinglib.is_operator(temp_char)):
                            #print(temp_string);
                            Mainclass.token(temp_string)
                            temp_string = ''
                            if( testinglib.is_operator(line[x+1])):
                                if(testinglib.is_ogical_operator((temp_char+line[x+1]))):
                                    #print((temp_char+line[x+1]))
                                    Mainclass.token((temp_char+line[x+1]))

                                elif((temp_char+line[x+1])=="//"):
                                    break
                                elif((temp_char+line[x+1])=="/*"):
                                    while 1:
                                        if("*/" in line):
                                            line=line[line.find("*/"):line.__len__()]
                                            #print(line)
                                            x=0;
                                            break
                                            #exit()
                                        else:
                                            line =file.readline()
                                    x+=1

                            else:
                                #print(temp_char)
                                Mainclass.token(temp_string)
                                temp_char =''
                        else:
                            #print("comment el")
                            #print(temp_char);
                            x+=1
                            continue
                    x+=1;
Mainclass.main_method()
out= open('output.txt','w+')
out.write("Tokens:\n")
out.write("-----------------------------------------\n")
out.write("Lexeme        Token Name        Attribute Value\n")
out.write("-----------------------------------------\n")
for qq in xrange(Mainclass.result_array_token_table.__len__()):
    out.write(Mainclass.result_array_token_table[qq]+'\n')
out.write("-----------------------------------------\n\n\n\n\n\n\n")

out.write("Symbol Table:\n")
out.write("-----------------------------------------\n")
out.write("Symbol      Token      Data Type    Pointer To Symbol Table Entry\n")
out.write("-----------------------------------------\n")
for qq in xrange(Mainclass.result_array_symbolic_table.__len__()):
    out.write(Mainclass.result_array_symbolic_table[qq]+"   "+str(qq)+'\n')



#Input File
# int main()
# {
#   int m1x,n,o;
#   float number=200.5;
#   string x,y,nm_25;
#  int count =1;
#  long a=1000.9,v2x=10;
#
#   for(m=1;m<=number;m++)
#   {
#       for(n=1;n<=number-m;n++)
#       {
#           printf("  ");
#       }
#       for(o=1;o<=(2*m-1);o++)
#       {
#        count= count+ /*wellcome I am a comment. He He .. Try to scrip me. uhhh.*/ 50;
#            if(count%2==0)
#            printf(" 0");
#            else
```

```
#             printf(" 1");
#             count++;
#             //Hi I am a Comment. You Can Skrip Me. He He He ......
#       }
#       printf("\n");
#   }
#   boolean pq_x=false, new_var;
#   char pq_x, new_var;
#
#
#   return 0;
# }


#Output File
# Tokens:
# ------------------------------------------------------------------------
# Lexeme        Token Name        Attribute Value
# ------------------------------------------------------------------------
# int            int                  --
# main                id               --
# (            Special Sysmbol      opening parenthesis
# )            Special Sysmbol      closing parenthesis
# {            Special Sysmbol      Left Curly brace
# int            int                  --
# m1x                 id             Pointer To Symbolic Table
# ,            Special Sysmbol              Comma
# n                id             Pointer To Symbolic Table
# ,            Special Sysmbol              Comma
# o                id             Pointer To Symbolic Table
# ;              Special Sysmbol          Semiclone
# float             float               --
# number                id             Pointer To Symbolic Table
# 200.5             Numberl               Constant
# ;              Special Sysmbol          Semiclone
# string             string               --
# x                id             Pointer To Symbolic Table
# ,            Special Sysmbol              Comma
# y                id             Pointer To Symbolic Table
# ,            Special Sysmbol              Comma
# nm_25               id              Pointer To Symbolic Table
# ;              Special Sysmbol          Semiclone
# int            int                  --
# count               id             Pointer To Symbolic Table
# 1              Numberl              Constant
# ;              Special Sysmbol          Semiclone
# long             long                --
# a                id             Pointer To Symbolic Table
# 1000.9            Numberl                Constant
# ,            Special Sysmbol              Comma
# v2x                id            Pointer To Symbolic Table
# 10             Numberl              Constant
# ;              Special Sysmbol          Semiclone
# for            for                  --
# (            Special Sysmbol      opening parenthesis
# m                id             Pointer To Symbolic Table
# 1              Numberl              Constant
# ;              Special Sysmbol          Semiclone
# m                id             Pointer To Symbolic Table
# <=             Operator           less Than Equal
# number                id             Pointer To Symbolic Table
# ;              Special Sysmbol          Semiclone
# m                id             Pointer To Symbolic Table
# ++             Operator           increment by 1
# )            Special Sysmbol      closing parenthesis
# {            Special Sysmbol      Left Curly brace
# for              for                  --
# (            Special Sysmbol      opening parenthesis
# n                id             Pointer To Symbolic Table
# 1              Numberl              Constant
# ;              Special Sysmbol          Semiclone
# n                id             Pointer To Symbolic Table
# <=             Operator           less Than Equal
# number                id             Pointer To Symbolic Table
# m                id             Pointer To Symbolic Table
```

```
# ;            Special Sysmbol          Semiclone
# n                id             Pointer To Symbolic Table
# ++           Operator          increment by 1
# )            Special Sysmbol      closing parenthesis
# {            Special Sysmbol      Left Curly brace
# printf           printf                --
# (            Special Sysmbol      opening parenthesis
# )            Special Sysmbol      closing parenthesis
# ;            Special Sysmbol          Semiclone
# }            Special Sysmbol      Right Curly brace
# for            for                   --
# (            Special Sysmbol      opening parenthesis
# o                id             Pointer To Symbolic Table
# 1            Numberl                Constant
# ;            Special Sysmbol          Semiclone
# o                id             Pointer To Symbolic Table
# <=           Operator          less Than Equal
# (            Special Sysmbol      opening parenthesis
# 2            Numberl                Constant
# m                id             Pointer To Symbolic Table
# 1            Numberl                Constant
# )            Special Sysmbol      closing parenthesis
# ;            Special Sysmbol          Semiclone
# o                id             Pointer To Symbolic Table
# ++           Operator          increment by 1
# )            Special Sysmbol      closing parenthesis
# {            Special Sysmbol      Left Curly brace
# count                id             Pointer To Symbolic Table
# count                id             Pointer To Symbolic Table
# 50             Numberl               Constant
# ;            Special Sysmbol          Semiclone
# if           if                  --
# (            Special Sysmbol      opening parenthesis
# count                id             Pointer To Symbolic Table
# 2            Numberl                Constant
# ==           Operator          Compare To
# 0            Numberl                Constant
# )            Special Sysmbol      closing parenthesis
# printf           printf                --
# (            Special Sysmbol      opening parenthesis
# 0            Numberl                Constant
# )            Special Sysmbol      closing parenthesis
# ;            Special Sysmbol          Semiclone
# else           else                   --
# printf           printf                --
# (            Special Sysmbol      opening parenthesis
# 1            Numberl                Constant
# )            Special Sysmbol      closing parenthesis
# ;            Special Sysmbol          Semiclone
# count                id             Pointer To Symbolic Table
# ++           Operator          increment by 1
# ;            Special Sysmbol          Semiclone
# }            Special Sysmbol      Right Curly brace
# printf           printf                --
# (            Special Sysmbol      opening parenthesis
# n                id             Pointer To Symbolic Table
# )            Special Sysmbol      closing parenthesis
# ;            Special Sysmbol          Semiclone
# }            Special Sysmbol      Right Curly brace
# boolean          boolean                 --
# pq_x               id               Pointer To Symbolic Table
# false            false                 --
# ,          Special Sysmbol             Comma
# new_var              id              Pointer To Symbolic Table
# ;            Special Sysmbol          Semiclone
# char           char                  --
# pq_x               id               Pointer To Symbolic Table
# ,          Special Sysmbol             Comma
# new_var              id              Pointer To Symbolic Table
# ;            Special Sysmbol          Semiclone
# return           return                --
# 0            Numberl                Constant
# ;            Special Sysmbol          Semiclone
# }            Special Sysmbol      Right Curly brace
# ----------------------------------------------------------------------
```

```
#
#
#
#
#
# Symbol Table:
# ------------------------------------------------------------------------
# Symbol      Token      Data Type    Pointer To Symbol Table Entry
# ------------------------------------------------------------------------
# main              id                --   0
# m1x           id                int   1
# n           id                int   2
# o           id                int   3
# number           id                float   4
# x           id                string   5
# y           id                string   6
# nm_25           id                string   7
# count           id                int   8
# a           id                long   9
# pq_x           id                boolean   10
# pq_x           id                char   11
# new_var           id                char   12
#------------------------------------------------------------------------
```