Code: import re

def is\_Arithmetic\_operator(ch):

return ch in '+-\*/'

def is\_logical\_operator(ch):

return ch in '>=><==<=!='

def is\_punctuation\_operator(ch):

return ch in ',;:'

def is\_valid\_keyword(word):

keywords = {'auto', 'break', 'case', 'char', 'const', 'continue', 'default', 'do',

'double', 'else', 'enum', 'extern', 'float', 'for', 'goto', 'if', 'int',

'long', 'register', 'return', 'short', 'signed', 'sizeof', 'static',

'struct', 'switch', 'typedef', 'union', 'unsigned', 'void', 'volatile', 'while'}

return word in keywords

def remove\_comments(code):

lines = code.splitlines()

result = []# stores the non commented lines

comment = False

for line in lines:

if '/\*' in line:

comment = True

if not comment:

line = line.split('//')[0] # Remove single-line comments

result.append(line)

if '\*/' in line:

comment = False

return '\n'.join(result)

def is\_valid\_identifier(identifier):

pattern=r"[a-zA-Z]+[0-9]\*|[a-zA-Z]+"

return bool(re.match(pattern, identifier))

def remove\_spaces(code):

lines = code.splitlines()

result=[]

for line in lines:

words=line.split()

clean\_line =' '.join(filter(None,words))#selecting elements from words

result.append(clean\_line)

return '\n'.join(result)

def is\_a\_constant(s):

constant\_regex = r"\b\d+\b"

return bool(re.match(constant\_regex, s))

def tokenize(code):

code\_without\_comments = remove\_comments(code)

code\_without\_spaces = remove\_spaces(code\_without\_comments)

tokens =re.findall(r"[a-zA-Z\_]\w\*|[+\*/-]|[><=!]=?|[,;:{}:]", code\_without\_spaces)

return tokens

def generate\_lexical\_analysis(code):

tokens = tokenize(code)

token\_types = {

'Keyword': set(),

'Identifier': set(),

'Arithmetic Operator': set(),

'Constant': set(),

'Punctuation': set(),

'Parenthesis': set()

}

for token in tokens:

if is\_valid\_keyword(token):

token\_types['Keyword'].add(token)

elif is\_valid\_identifier(token):

token\_types['Identifier'].add(token)

elif is\_Arithmetic\_operator(token):

token\_types['Arithmetic Operator'].add(token)

elif is\_a\_constant(token):

token\_types['Constant'].add(token)

elif is\_punctuation\_operator(token):

token\_types['Punctuation'].add(token)

elif token in '{}':

token\_types['Parenthesis'].add(token)

return token\_types

def generate\_output(token\_types):

for token\_type, tokens in token\_types.items():

print(f"{token\_type} ({len(tokens)}): {', '.join(tokens)}")

if \_\_name\_\_ == "\_\_main\_\_":

code = input("Enter the code: ")

token\_types = generate\_lexical\_analysis(code)

generate\_output(token\_types)

Input: void main(){ int a,b,c; int a = b\*c+10}

Output:

Enter the code:

Keyword (2): void, int

Identifier (4): main, b, c, a

Arithmetic Operator (2): +, \*

Constant (1): 10

Punctuation (3): ,, ;, :

Parenthesis (2): }, {