NORTH SOUTH UNIVERSITY

Department of Electrical & Computer Engineering

Assignment On

Course Code: *CSE425*

Course Title: *Concepts of Programming Language*

Submitted by_

Name : Sajan Kumer Sarker
ID# : 2111131642
Email : sajan.sarker@northsouth.edu
Section : 08
Assignment Topic : Lexical Analyzer
Submission Date : 19th, April 2024

Submitted to_

Instructor : Dr. Mohammad Abdur Rouf (MDAR)
Email : mohammad.rouf@northsouth.edu

Source Code:

```
# Filename: lexical_analyzer.py
# Problem: Write a program using either C/C++, Java or Python language for a lexical
analyzer that can tokenize a C statement and can detect C keyword.
# Solution: The program reads a C program from a file or from user, removes comments,
newlines, and spaces from the program, and then extracts tokens from the processed
program using Python Programming Language. Source Code is Given Below:

########################################
###  Submitted by: Sajan Kumer Sarker  ##
###  ID: 2111131642                    ##
########################################

# import the required libraries
import re   # Regular expression library
import nltk # type: ignore  # Natural Language Toolkit library
import sys # System-specific parameters and functions
import os # Miscellaneous operating system interfaces
nltk.download('punkt') # Download the 'punkt' package from the NLTK library for
tokenization if not available.

### patterns for regular expressions for different tokens
keywords =
r"auto|break|case|#include|char|const|continue|default|do|double|else|enum|extern|flo
at|for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|switch|type
def|union|unsigned|void|volatile|while|string|class|struc|include"
assignment_operators = r"(\+=)|(-=)|(\*=)|(/=)|(%=)|(&=)|(=)|(\^=)|(>>=)|(<<=)"
relational_operators = r"(==)|(!=)|(<)|(>)|(<=)|(>=)"
logical_operators = r"(&&)|(\|\|)|(!)"
bitwise_operators = r"(&)|(\|)|(\^)|(~)|(<<)|(>>)"
addition_operators = r"(\+)"
subtraction_operators = r"(-)"
multiplication_operators = r"(\*)"
division_operators = r"(/)"
modulus_operators = r"(%)"
constant = r"^(\d+)$"
special_characters = r"[@&~#!$\^\|:?,\.']|\""
identifiers = r"^[a-zA-Z_]+[a-zA-Z0-9_]*"
headers = r"([a-zA-Z]+\.[h])"
semicolon = r";"
parentheses = r"\(|\)"
braces = r"\{|\}"
square_brackets = r"\[|\]"
angle_brackets = r"\<|\>"

def remove_comments(program):
    """
    Removes both single-line and multiple-line comments from the given program.

    Args:
        program (str): The program code from which comments need to be removed.

    Returns:
        str: The program code without any comments.
    """
    single_line_comments = r"//.*"
    multiple_line_comments = r"/\*[^*]*\*+(?:[^/*][^*]*\*+)*/"
    sub_multiple_line_comments = re.sub(multiple_line_comments, "", program)
    program_without_comments = re.sub(single_line_comments, "",
sub_multiple_line_comments)
    return program_without_comments

def remove_newline(program):
    """
    Removes the '\\n' characters from each line of the given program and returns the
modified program.
```

```
    Args:
        program (str): The input program with '\\n' characters.

    Returns:
        list: The modified program with '\\n' characters removed from each line.
    """
    prog = program.split('\n')
    scanned_program = []
    for line in prog:
        if '\\n' in line:
            line = line.replace('\\n', '')
            scanned_program.append(line.strip())
        else:
            scanned_program.append(line.strip())
    return scanned_program

def remove_spaces(program):
    """
    Removes leading and trailing spaces from each line in the program.

    Args:
        program (list): The program as a list of strings, where each string
represents a line of code.

    Returns:
        list: The program with leading and trailing extra spaces removed from each
line.
    """
    scanned_program = []
    for line in program:
        if line.strip() != '':
            scanned_program.append(line.strip())
    return scanned_program

def get_tokens(program):
    """
    Tokenizes the given program and prints the tokens along with their corresponding
lexemes.

    Args:
        program (list): The program code as a list of lines.

    Returns:
        None
    """
    source_code = []
    for line in program:
        source_code.append(line)
    count = 0
    for line in source_code:
        if line.startswith("#include"):
            tokens = nltk.word_tokenize(line)
        else:
            tokens = nltk.wordpunct_tokenize(line)
        for token in tokens:
            count += 1
            #print(token)
            if(re.findall(keywords, token)):
                print("Token is KEYWORD,{:>10} Lexeme is-> {}.".format("",token))
            elif(re.findall(headers, token)):
                print("Token is HEADER,{:>11} Lexeme is-> {}.".format("",token))
            elif (re.findall(identifiers, token)):
                if 'include' or '.h' not in token:
                    print("Token is IDENTIFIER,{:>7} Lexeme is->
{}.".format("",token))
            elif(re.findall(assignment_operators, token)):
```

```
                print("Token is ASSIGN OPERATOR,{:>2} Lexeme is->
{}.".format("",token))
            elif(re.findall(relational_operators, token)):
                print("Token is RELATION OPERATOR,{:>0} Lexeme is->
{}.".format("",token))
            elif(re.findall(logical_operators, token)):
                print("Token is LOGICAL OPERATOR, Lexeme is-> {}.".format(token))
            elif(re.findall(bitwise_operators, token)):
                print("Token is BITWISE OPERATOR,{:>1} Lexeme is->
{}.".format("",token))
            elif(re.findall(addition_operators, token)):
                print("Token is ADDITION OPERATOR, Lexeme is-> {}.".format(token))
            elif(re.findall(subtraction_operators, token)):
                print("Token is SUBTRACT OPERATOR, Lexeme is-> {}.".format(token))
            elif(re.findall(multiplication_operators, token)):
                print("Token is MULTIPLICATION OPERATOR, Lexeme is->
{}.".format(token))
            elif(re.findall(division_operators, token)):
                print("Token is DIVISION OPERATOR, Lexeme is-> {}.".format(token))
            elif(re.findall(modulus_operators, token)):
                print("Token is MODULUS OPERATOR,{:>1} Lexeme is->
{}.".format("",token))
            elif(re.findall(constant, token)):
                print("Token is CONSTANT,{:>9} Lexeme is-> {}.".format("",token))
            elif(re.findall(special_characters, token)):
                if '.h' not in token:
                    print("Token is SPECIAL CHARACTER, Lexeme is->
{}.".format(token))
            elif(re.findall(semicolon, token)):
                print("Token is SEMICOLON,{:>8} Lexeme is-> {}.".format("",token))
            elif(re.findall(parentheses, token)):
                print("Token is PARENTHESE,{:>7} Lexeme is-> {}.".format("",token))
            elif(re.findall(braces, token)):
                print("Token is BRACE,{:>12} Lexeme is-> {}.".format("",token))
            elif(re.findall(square_brackets, token)):
                print("Token is SQUARE BRACKET,{:>10} Lexeme is->
{}.".format("",token))
            elif(re.findall(angle_brackets, token)):
                print("Token is ANGLE BRACKET,{:>4} Lexeme is->
{}.".format("",token))
            else:
                print("Token is UNKNOWN,{:>10} Lexeme is-> {}.".format("",token))
    print("Total number of tokens are: ", count)

def main():
    """
    Entry point of the program.

    Reads a file, removes comments, newlines, and spaces from the file content,
    and then extracts tokens from the processed program.
    """
    try:
        #print(os.getcwd())  # Current working directory
        f = open('D:/Programing/University-Assignment/cse425/Lexical
Analyzer/code.c')      # Take the input from a .c file
        main_program = f.read()
        #main_program = input("Enter the Expression: ") # Take the input from the
user
        without_comments = remove_comments(main_program)
        without_newline = remove_newline(without_comments)
        program = remove_spaces(without_newline)
        get_tokens(program)
    except FileNotFoundError:
        sys.exit("File not found")  # Exit the program if the file is not found

if __name__ == "__main__":  # Run the main function if the script is executed
    main()
```

Input: (As a .c File)

```
/*** File Name: code.c ***/
int a = b + c;     // Expression 1
int x = ab + bc - 30 + previous / 0y;     // Expression 2
```

Output: (Through Terminal)

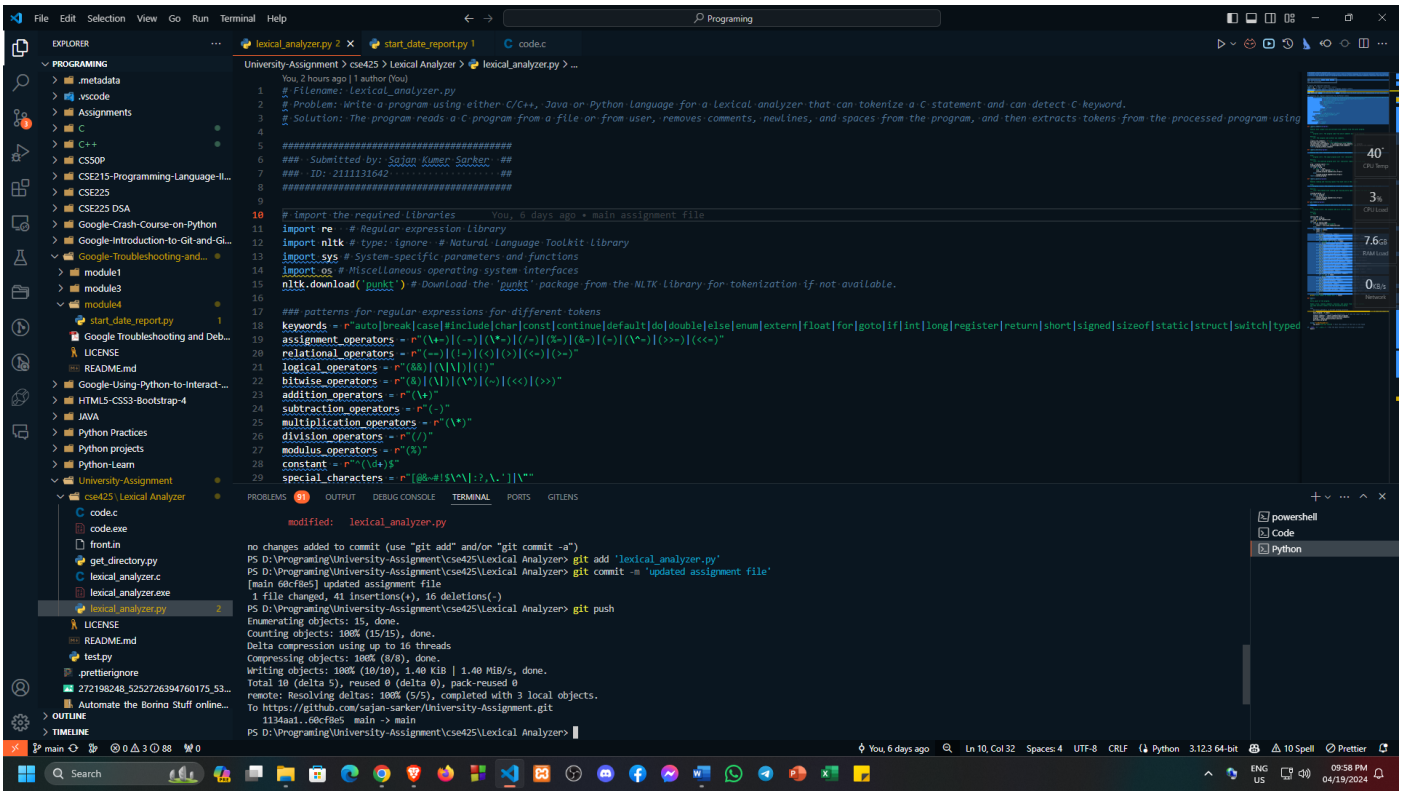
```
PS D:\Programing\University-Assignment\cse425\Lexical Analyzer> &
C:/Users/sajan/AppData/Local/Programs/Python/Python312/python.exe
"d:/Programing/University-Assignment/cse425/Lexical Analyzer/lexical_analyzer.py"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sajan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Token is KEYWORD,            Lexeme is-> int.
Token is IDENTIFIER,         Lexeme is-> a.
Token is ASSIGN OPERATOR,    Lexeme is-> =.
Token is IDENTIFIER,         Lexeme is-> b.
Token is ADDITION OPERATOR, Lexeme is-> +.
Token is IDENTIFIER,         Lexeme is-> c.
Token is SEMICOLON,          Lexeme is-> ;.
Token is KEYWORD,            Lexeme is-> int.
Token is IDENTIFIER,         Lexeme is-> x.
Token is ASSIGN OPERATOR,    Lexeme is-> =.
Token is IDENTIFIER,         Lexeme is-> ab.
Token is ADDITION OPERATOR, Lexeme is-> +.
Token is IDENTIFIER,         Lexeme is-> bc.
Token is SUBTRACT OPERATOR, Lexeme is-> -.
Token is CONSTANT,           Lexeme is-> 30.
Token is ADDITION OPERATOR, Lexeme is-> +.
Token is IDENTIFIER,         Lexeme is-> previous.
Token is DIVISION OPERATOR, Lexeme is-> /.
Token is UNKNOWN,            Lexeme is-> 0y.
Token is SEMICOLON,          Lexeme is-> ;.
Total number of tokens are:  20
```

Input & Output: (User Input & Output Through Terminal)

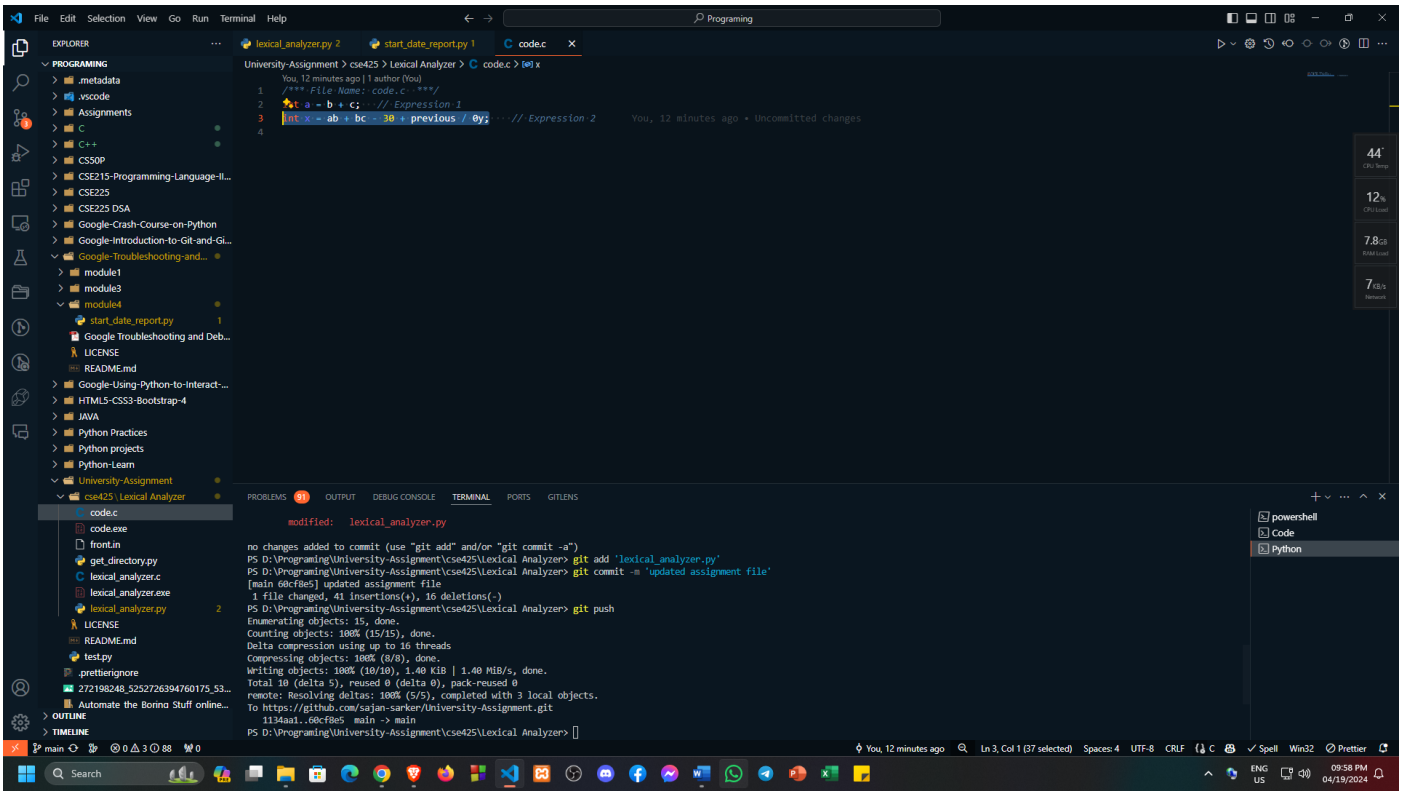
```
PS D:\Programing\University-Assignment\cse425\Lexical Analyzer> &
C:/Users/sajan/AppData/Local/Programs/Python/Python312/python.exe
"d:/Programing/University-Assignment/cse425/Lexical Analyzer/lexical_analyzer.py"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sajan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Enter the Expression: result = oldsum - value / 100;
Token is IDENTIFIER,         Lexeme is-> result.
Token is ASSIGN OPERATOR,    Lexeme is-> =.
Token is IDENTIFIER,         Lexeme is-> oldsum.
Token is SUBTRACT OPERATOR, Lexeme is-> -.
Token is IDENTIFIER,         Lexeme is-> value.
Token is DIVISION OPERATOR, Lexeme is-> /.
Token is CONSTANT,           Lexeme is-> 100.
Token is SEMICOLON,          Lexeme is-> ;.
Total number of tokens are:  8
```

Screenshots:

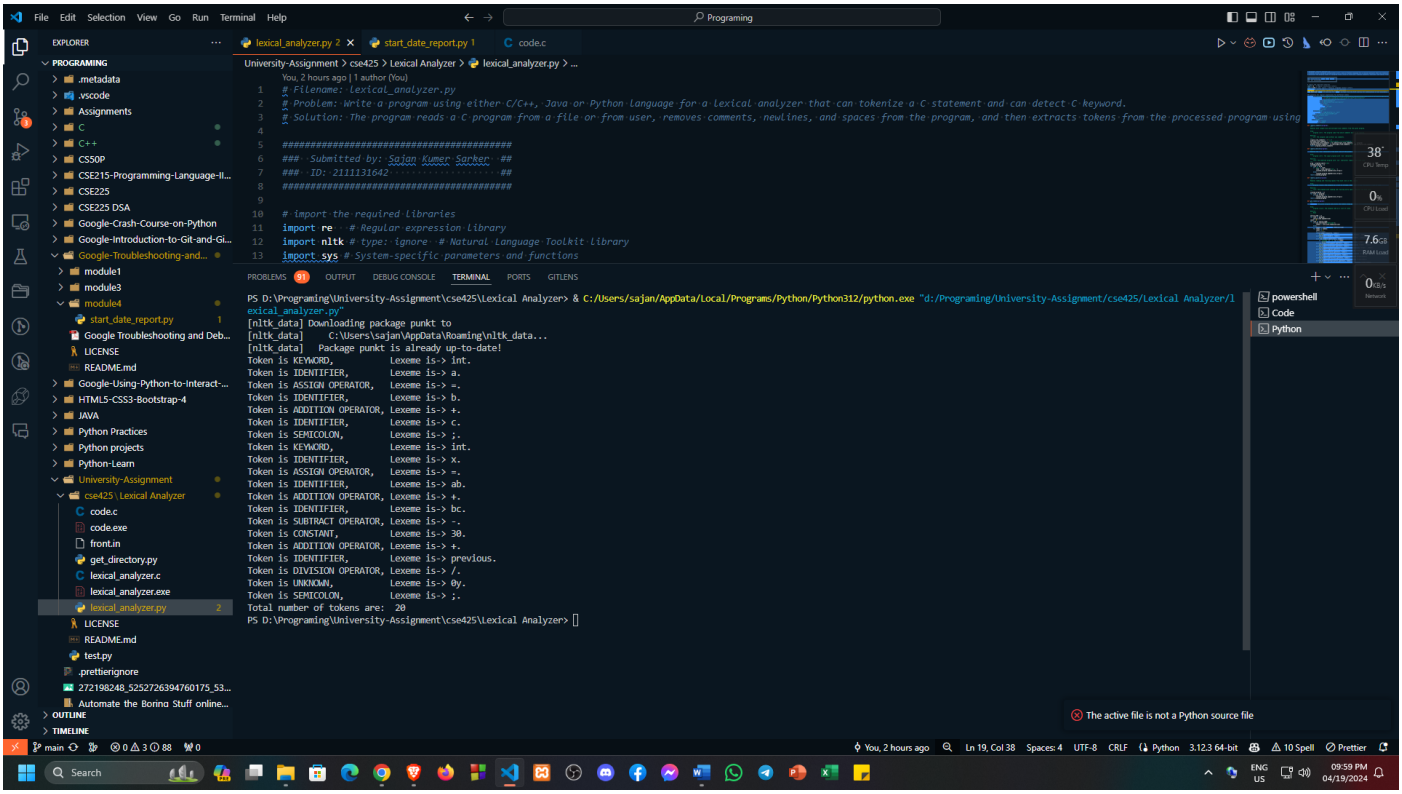
- Before running the code



- .c file for input (2 C programing expression)



- Output for the .c file



- Input and output through terminal (User Input: C programming Expression)