#Report: CSC333

#Rajkumar Kushwaha

skip any initial comments and white space (spaces, tabs, and newlines)

import necessary modules like sys and re

function lexical\_analyzer(file):

Remove comment from file

Remove whitespaces from file

Split the file content into tokens using regular expressions

# Check if the token is a keyword

if current\_char matches a keyword from the list ['if', 'else', 'while', 'break', 'read', 'write', 'function', 'return']:

return "Token:keywords(token)"

if current\_char == '!=':

return "Token:NotEqual(!=)"

if current\_char matches the pattern of an identifier (starting with a letter or underscore, followed by letters, digits, or underscores):

return "Token:Identifiers(token)"

if current\_char matches the pattern of an integer (sequence of digits):

return "Token:integer(token)"

if current\_char matches the pattern of a floating-point number (digits, decimal point, up to three digits after the decimal):

return "Token:floating-point(token)"

if current\_char matches any of the arithmetic operators:

if current\_char == '+':

return "Token:Addition(+)"

if current\_char == '-':

return "Token:Subtraction(-)"

if current\_char == '/':

return "Token:Divide(/)"

if current\_char == '\*':

if next\_char == '\*':

return "Token:Power(\*\*)"

else:

return "Token:Multiply(\*)"

if current\_char == '%':

return "Token:Modulo(%)"

if current\_char matches any of the comparison operators:

if current\_char == '<':

return "Token:LThan(<)"

if current\_char == '>':

return "Token:GThan(>)"

if current\_char == '==':

return "Token:Equal(==)"

if current\_char == '>=':

return "Token:GThanEqual(>=)"

if current\_char == '<=':

return "Token:LThanEqual(<=)"

if current\_char == '=':

return "Token:Assignment(=)"

if current\_char matches the logical operators:

if current\_char == '&':

return "Token:and(&)"

if current\_char == '|':

return "Token:or(|)"

if current\_char matches any of the bracket types:

if current\_char == '{':

return "Token:Lpar({)"

if current\_char == '}':

return "Token:Rpar(})"

if current\_char == '(':

return "Token:LBrac(()"

if current\_char == ')':

return "Token:RBrac())"

if current\_char == ';':

return "Token:Semicolon(;)"

if current\_char == ',':

return "Token:Comma(,)"

# Handle file input and tokenization process

ask user for file\_path from terminal

open the file and read its content into a variable

# Execute lexical analysis

call lexical\_analyzer(file\_content)

# Redirect output to a file called 'output.txt'

open 'output.txt' for writing

redirect standard output to 'output.txt'

call lexical\_analyzer(file\_content)

restore standard output to terminal

# Print the location of the output file

display message with absolute path of 'output.txt'