A Lexical Analyzer

Report by: Rohan Venkatesha

Phase 1: Specification:

Language Specification: The self-designed language is called Cminus and it has the following features:

- It is a case-sensitive language that uses ASCII characters.
- It supports only two data types: basic type int and a standard data type float.
- It supports arithmetic, logical, relational, and assignment operators.
- It supports if-else, while, and compound statements for control flow.
- It supports single-line and multi-line comments that start with /* and end with */.
- It supports identifiers that start with a letter and can contain alphanumeric characters.
- It supports literals that are enclosed in single quotes for strings, and supports only decimal notation for floating point numbers.
- It supports keywords that are reserved for the language and cannot be used as identifiers. The keywords are: int, float, if, else, exit, while, read, write, and return.

Lexical Analyzer Specification: The lexical analyzer is a program that takes an input source code file written in Cminus and produces a list of tokens as output. A token is a pair of a token type and a token value. The token types are:

- KEYWORD: A reserved word in the language.
- IDENTIFIER: A user-defined name for a variable or a function.
- CONSTANT: A constant value of a data type.
- ARITH-OP: A symbol that performs an arithmetic operation on operands.
- LOGIC-OP: A symbol that performs a logical operation on operands.
- SEPARATOR: A symbol that separates tokens or groups them together.
- COMMENT: A text that is ignored by the compiler and used for documentation purposes.

The lexical analyzer should follow these steps:

- Read the input source code file line by line and store it in a buffer.
- Scan the buffer from left to right and identify the tokens based on the language specification.
- For each token, create a token object with the token type and the token value as attributes.
- Append the token object to a list of tokens.
- Repeat steps 2 to 4 until the end of the buffer is reached or an error is encountered.
- Return the list of tokens as output or display an error message if an error is encountered.

Phase 2: Design:

Modules and basic structures:

1. **Tokenizer**: The primary objective of this module is to tokenize the source code provided in the input file. The module utilizes regular expressions for token recognition and follows a set of predefined patterns to identify various types of tokens. It includes the following key components:
   token_patterns: This list contains tuples of token types and their corresponding regular expressions for recognition.
   tokenize_source_code(file_name): A function that takes the file name as input and returns a list of token-value pairs.
2. **Regular Expressions**: The design relies on regular expressions to identify different types of tokens in the source code. These regular expressions are stored in the token_patterns list. The token types recognized include KEYWORD, COMMENT, IDENTIFIER, CONSTANT, ARITH_OP, LOGIC_OP, and SEPARATOR.
3. **File Input Handling:** The module handles file input and reads the content line by line. It strips leading and trailing whitespace from each line and combines the lines into a single string for tokenization.
4. **Tokenization**: The tokenization process involves iterating through the source text and matching it against the defined regular expressions. When a match is found, the corresponding token type and value are recorded accordingly.
5. **Error Handling**: The module handles various error scenarios, such as file not found, unclosed comments, invalid comments, and unrecognized characters. Error messages are displayed when such situations occur.
6. **Output**: Output is stored in list named **templist** where it displays the token type and respective token values.

Phase 3: Pseudocode:

```
# Define token patterns for token recognition
token_patterns = [
  ("KEYWORD", r'\b(int|main|float|if|else|exit|while|read|write|return)\b'),
  ("COMMENT", r'\/\*[\s\S]*?\*\/|\/\*[\s\S]*$'),
  ("IDENTIFIER", r'[a-zA-Z_]\w*'),
  ("CONSTANT", r'\d+(\.\d+)?'),
  ("ARITH_OP", r'[-+*/=]'),
  ("LOGIC_OP", r'==|!=|<=|>=|&&|\|\|'),
    ("SEPARATOR", r'[(),;{}[\]]')
]
# Function to tokenize source code
function tokenize_source_code(file_name):
    try:
        # Open the file for reading
        file = open(file_name, "r")
        # Initialize an empty list to store the source code lines
        source_code = []
        # Read lines from the file
        lines = file.read_lines()
        # For each line in the file
        for line in lines:
            # Strip leading and trailing whitespace from the line
            stripped_line = line.strip()
            # Append the stripped line to the source_code list
            source_code.append(stripped_line)
        # Close the file
        file.close()
    except FileNotFoundError:
        print("Error: File not found.")
        return an empty list
    # Initialize an empty list to store the tokens
    tokens = []
    # Combine lines into a single string
    source_text = concatenate source_code into a single string
    # While there is source text to process
    while source_text is not empty:
        # Initialize match to None
        match = None
        # For each token pattern in token_patterns
        for token_type, pattern in token_patterns:
            # Create a regular expression object from the pattern
            regex = create a regular expression object using the pattern and the DOTALL flag
            # Attempt to match the regular expression with the source text
            match = match the regex with the source_text
            # If a match is found
            if match is not None:
                if token_type is not "COMMENT":
                    # Extract the matched token value
                    token_value = match.group(0)
                else:
                    # If the token type is "COMMENT"
                    expression = source_text
                    if expression starts with "/*":
                        # Find the end index of "*/" within the expression
                        end_index = find the index of "*/" within the expression
                        # If "*/" is found
                        if end_index is not -1:
                            # Extract the comment as the token value
                            token_value = extract the comment from the expression
                        else:
                            # Print a lexical error message for an unclosed comment
                            print("Lexical error: Unclosed comment. ", expression)
                            return the list of tokens
                # Append the token type and token value to the list of tokens
                append (token_type, token_value) to tokens
                # Update the source text by removing the processed token
                source_text = remove the processed token from the source_text
                # Exit the loop
                break
        # If no match is found
        if match is None:
            if the first character of source_text is whitespace:
                # Remove the first character (skip whitespace)
                source_text = remove the first character from source_text
```

```
          else:
              # Print a lexical error message for an unrecognized character
              print("Lexical error: Unable to tokenize:", the first character of source_text)
              # Remove the unrecognized character from source_text
              source_text = remove the first character from source_text
    # Return the list of tokens
    return tokens
# Call the tokenize_source_code function with the input file name
tokens = tokenize_source_code("input_file")
# Initialize a list for the simplified tokens
temporary_list = []
# Iterate through the identified tokens
for each token in tokens:
    if the type of token is "COMMENT":
        append ("COMMENT", "....") to temporary_list
    else:
        # Otherwise, keep the original token
        append the token to temporary list
# Print the temporary_list
print(temporary_list)
```

Phase 4: Testing and Output:

Output 1:

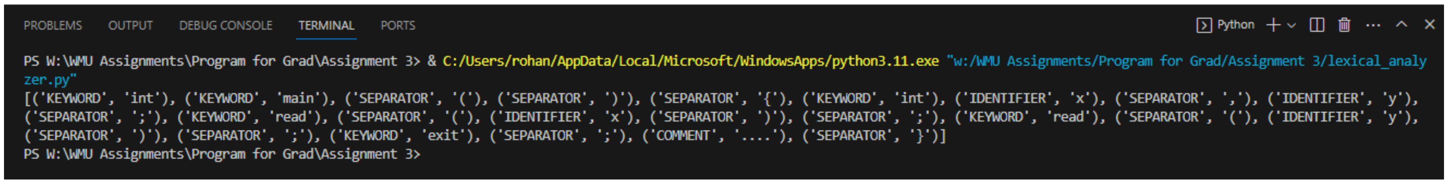
Input Source code file: source_code.cminus

```
 int main() {
     int x, y;
     read(x); read(y);
     exit;/*This is Comment */
   }
```

Output:

```
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('IDENTIFIER', 'y'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('),
```

('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]



```
PS W:\WMU Assignments\Program for Grad\Assignment 3> & C:/Users/rohan/AppData/Local/Microsoft/WindowsApps/python3.11.exe "w:/WMU Assignments/Program for Grad/Assignment 3/lexical_analy
zer.py"
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('IDENTIFIER', 'y'),
('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'),
('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]
PS W:\WMU Assignments\Program for Grad\Assignment 3>
```

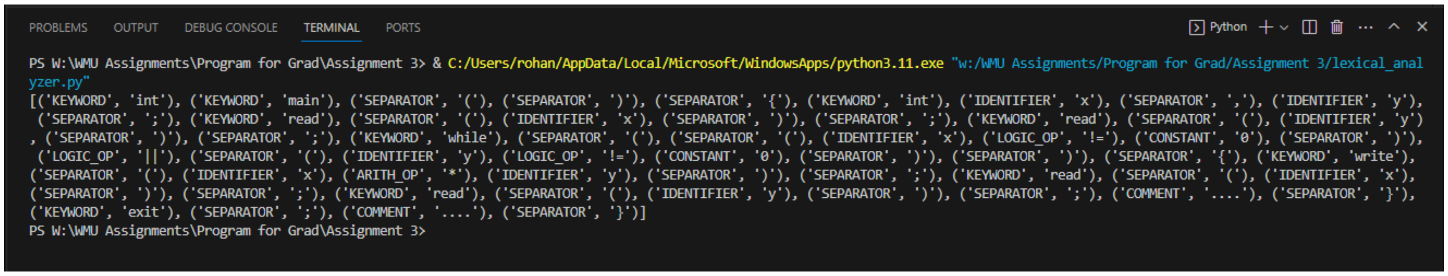
Output 2:

Input Source code file: source_code.cminus

```
 int main() {

     int x, y;

     read(x); read(y);

     while ((x != 0) || (y != 0)) {

         write(x*y);

         read(x); read(y); /* declaration"*/

     }

     exit;/*This is

     multiline Comment */

  }
```

Output:

[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('IDENTIFIER', 'y'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}'), ('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]



```
PS W:\WMU Assignments\Program for Grad\Assignment 3> & C:/Users/rohan/AppData/Local/Microsoft/WindowsApps/python3.11.exe "w:/WMU Assignments/Program for Grad/Assignment 3/lexical_anal
yzer.py"
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('IDENTIFIER', 'y'),
 ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y')
, ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'),
 ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'),
('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'),
('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}'),
('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]
PS W:\WMU Assignments\Program for Grad\Assignment 3>
```

Output 3:

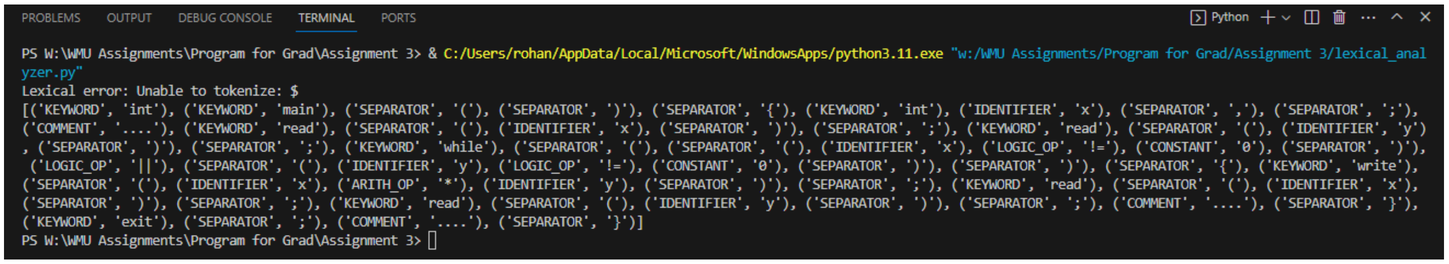
Input Source code file: source_code.cminus

```
int main() {
    int x, $ ; /* wrong declaration"*/
    read(x); read(y);
    while ((x != 0) || (y != 0)) {
       write(x*y);
       read(x); read(y); /* declaration"*/
    }
    exit;/*This is
    multiline Comment */
  }
```

Output:

Lexical error: Unable to tokenize: $

```
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('SEPARATOR', ';'), ('COMMENT', '....'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}'), ('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]
```



```
PS W:\WMU Assignments\Program for Grad\Assignment 3> & C:/Users/rohan/AppData/Local/Microsoft/WindowsApps/python3.11.exe "w:/WMU Assignments/Program for Grad/Assignment 3/lexical_anal
yzer.py"
Lexical error: Unable to tokenize: $
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('SEPARATOR', ';'),
('COMMENT', '....'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y')
, ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'),
 ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'),
('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'),
('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}'),
('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('COMMENT', '....'), ('SEPARATOR', '}')]
PS W:\WMU Assignments\Program for Grad\Assignment 3> 
```

Output 4:

Input Source code file: source_code.cminus

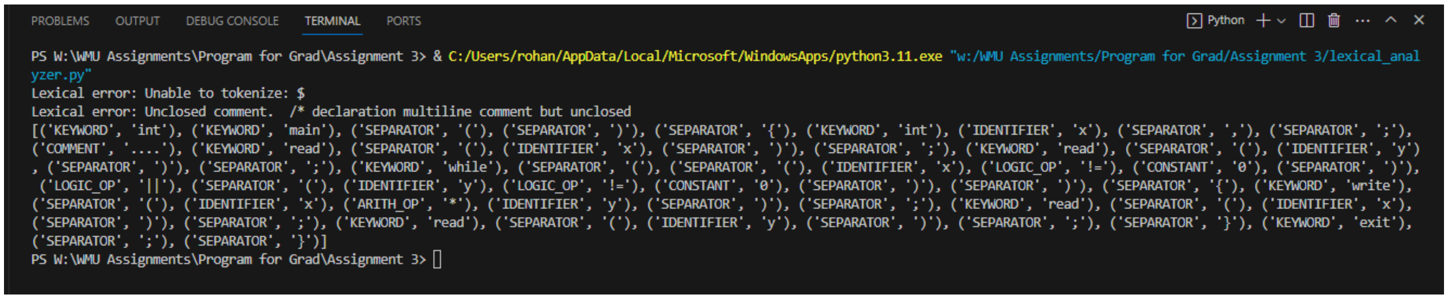
```
int main() {
    int x, $ ; /* wrong declaration"*/
    read(x); read(y);
    while ((x != 0) || (y != 0)) {
       write(x*y);
       read(x); read(y);
```

```
      }
      exit;
   } /* declaration multiline comment
         but unclosed
```

Output:

Lexical error: Unable to tokenize: $
Lexical error: Unclosed comment. /* declaration multiline comment but unclosed

```
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('SEPARATOR', ';'), ('COMMENT', '....'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('SEPARATOR', '}'), ('KEYWORD', 'exit'), ('SEPARATOR', ';'), ('SEPARATOR', '}')]
```



```
PS W:\WMU Assignments\Program for Grad\Assignment 3> & C:/Users/rohan/AppData/Local/Microsoft/WindowsApps/python3.11.exe "w:/WMU Assignments/Program for Grad/Assignment 3/lexical_anal
yzer.py"
Lexical error: Unable to tokenize: $
Lexical error: Unclosed comment.  /* declaration multiline comment but unclosed
[('KEYWORD', 'int'), ('KEYWORD', 'main'), ('SEPARATOR', '('), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'int'), ('IDENTIFIER', 'x'), ('SEPARATOR', ','), ('SEPARATOR', ';'),
('COMMENT', '....'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y')
, ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'while'), ('SEPARATOR', '('), ('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'),
 ('LOGIC_OP', '||'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('LOGIC_OP', '!='), ('CONSTANT', '0'), ('SEPARATOR', ')'), ('SEPARATOR', ')'), ('SEPARATOR', '{'), ('KEYWORD', 'write'),
('SEPARATOR', '('), ('IDENTIFIER', 'x'), ('ARITH_OP', '*'), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'x'),
('SEPARATOR', ')'), ('SEPARATOR', ';'), ('KEYWORD', 'read'), ('SEPARATOR', '('), ('IDENTIFIER', 'y'), ('SEPARATOR', ')'), ('SEPARATOR', ';'), ('SEPARATOR', '}'), ('KEYWORD', 'exit'),
('SEPARATOR', ';'), ('SEPARATOR', '}')]
PS W:\WMU Assignments\Program for Grad\Assignment 3>
```

Notes

Reference

Chat-GPT like tools for Multiline Comment handling