CS 424: Compiler Construction

Assignment#1

Report

Arbaz Khan

2020076

Design Decisions:

Use of Regular Expressions: Regular expressions are still used for tokenizing MiniLang, ensuring an efficient and concise approach to recognizing different token patterns.

Class-Based Design: The MiniLangScanner remains implemented as a class to encapsulate the scanning functionality, offering better code organization and reuse.

Token Representation: The representation of tokens as tuples (token type, lexeme) remains unchanged, providing a standardized way to represent tokens.

Error Handling: Lexical errors continue to be handled by printing an error message with details about the line number and the problematic token.

Reading Source Code from a File: The source code is read from a text file specified as a command-line argument, enhancing flexibility for users.

Scanner Structure:

The scanner follows a simple structure:

Initialization: The scanner is initialized with the MiniLang source code and sets up data structures.

Tokenization: The scan method reads the source code line by line, ignores comments, and uses regular expressions to find and categorize tokens. Tokens are appended to the tokens list.

Token Types: Token types are determined based on the regular expression matches and the defined MiniLang specifications.

Error Handling: Lexical errors are handled by printing an error message when an unrecognized token is encountered.

How to Run the Program:

Copy the updated MiniLangScanner code into a Python file (e.g. scanner.py).

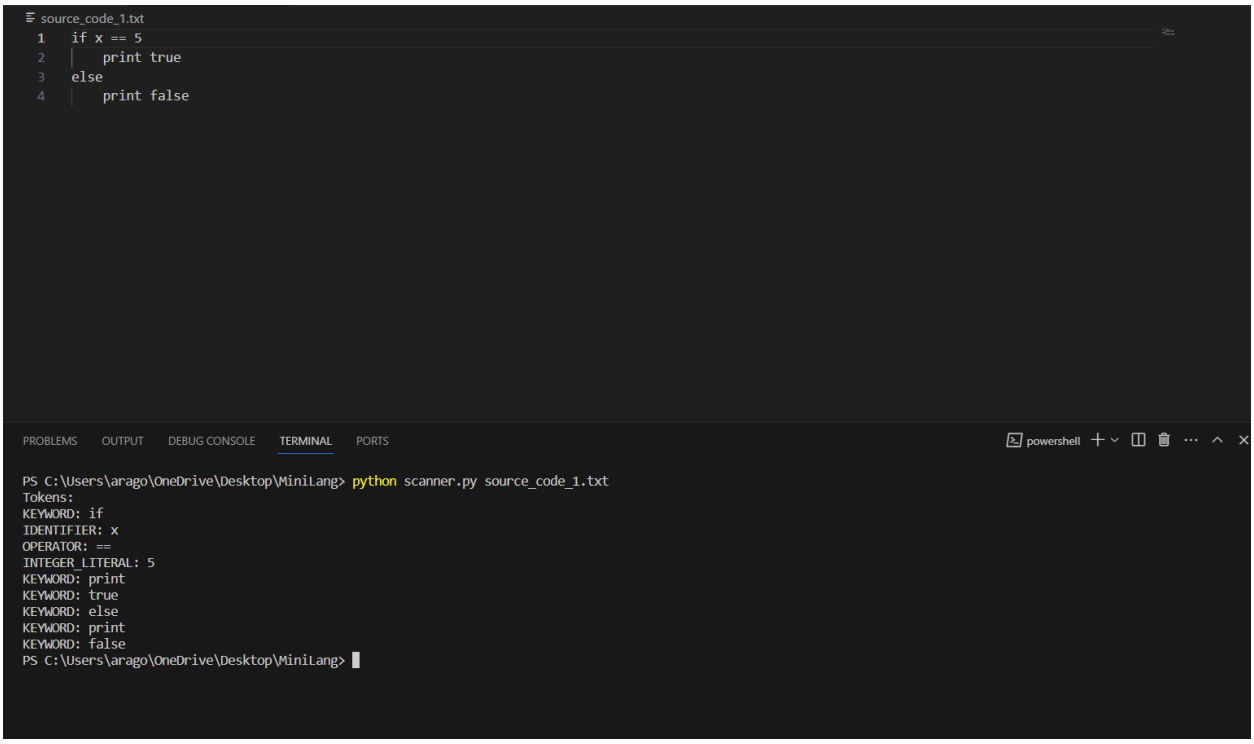
Run the program from the command line, providing the MiniLang source code file as an argument:

```
python scanner.py source_code.txt
```

The program will tokenize the source code and print the resulting tokens.

Test Cases:

Here are some test cases:



source_code_2.txt

```
if x == 5
    print true
else
    print false
    invalid_symbol @
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\arago\OneDrive\Desktop\MiniLang> python scanner.py source_code_2.txt
Tokens:
KEYWORD: if
IDENTIFIER: x
OPERATOR: ==
INTEGER_LITERAL: 5
KEYWORD: print
KEYWORD: true
KEYWORD: else
KEYWORD: print
KEYWORD: false
PS C:\Users\arago\OneDrive\Desktop\MiniLang>
```

source_code_3.txt

```
result = 3 + (y * 2)
print result
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\arago\OneDrive\Desktop\MiniLang> python scanner.py source_code_3.txt
Tokens:
IDENTIFIER: result
OPERATOR: =
INTEGER_LITERAL: 3
OPERATOR: +
OPERATOR: (
IDENTIFIER: y
OPERATOR: *
INTEGER_LITERAL: 2
OPERATOR: )
KEYWORD: print
IDENTIFIER: result
PS C:\Users\arago\OneDrive\Desktop\MiniLang>
```

source_code_4.txt

```
a = 10
b = false
// This is a comment
print a + b
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS C:\Users\arago\OneDrive\Desktop\MiniLang> python scanner.py source_code_4.txt
Tokens:
IDENTIFIER: a
OPERATOR: =
INTEGER_LITERAL: 10
IDENTIFIER: b
OPERATOR: =
KEYWORD: false
KEYWORD: print
IDENTIFIER: a
OPERATOR: +
IDENTIFIER: b
PS C:\Users\arago\OneDrive\Desktop\MiniLang>
```