Assignment#2
CS-424
Compiler Construction

Murtaza Rasheed

2020388

We have chosen to tackle the MiniLang parsing job using a recursive descent parser. The decision was made due to the language's ease of use and the ease with which its grammatical rules could be converted into recursive functions. Particularly well suited for LL(k) grammars, where k denotes the required number of lookahead symbols—are recursive descent parsers.

These are the rules of grammar that apply to MiniLang:

- Program: consists of a series of assertions.
- Statement: Can be an assignment, an if statement, or a print statement.
- Assignment: Follows the pattern of IDENTIFIER '=' Expression.
- IfStatement: Begins with 'if' followed by an Expression and a Program block, with an optional 'else' clause followed by another Program block.
- PrintStatement: Involves the 'print' keyword followed by an Expression.
- Expression: Consists of Terms combined with addition or subtraction operators.
- Term: Consists of Factors combined with multiplication or division operators.
- Factor: Can be a nested expression within parentheses, an identifier, an integer literal, or a boolean literal.

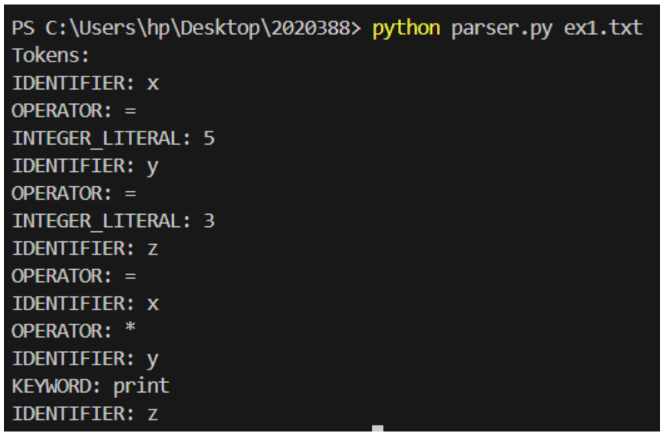
Tree of Abstract Syntax (AST)

Our parser builds an abstract syntax tree (AST) to depict the MiniLang source code's hierarchical structure. Every node in the tree represents a linguistic construct, making it easier to do analysis or write code later on.

Error management features built within the parser allow it to report syntactic errors together with helpful messages. When an error occurs, the parser gives precise information about the sort of error, where it occurred (line number), and why it happened. This helps users find and fix syntactic errors in their MiniLang programs.

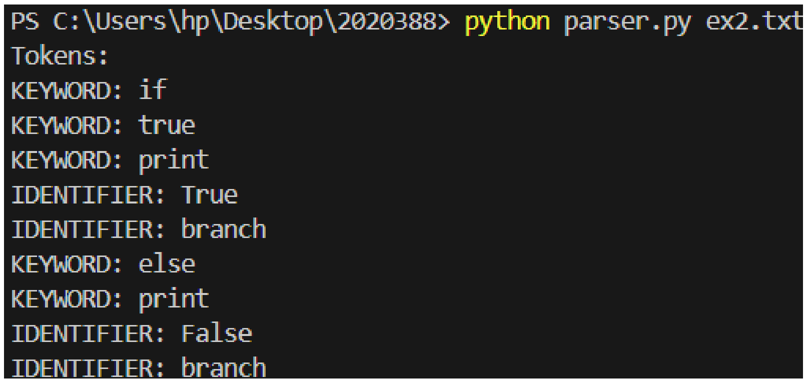
MiniLangScanner and MiniLangParser are the two main classes that make up our parser's architecture. Tokenization of the source code is handled by the scanner, and the tokens are used by the parser to build the abstract syntax tree (AST). Every grammar rule is implemented as a method in the MiniLangParser class, which encourages modularity and organization throughout the codebase.

Example1:



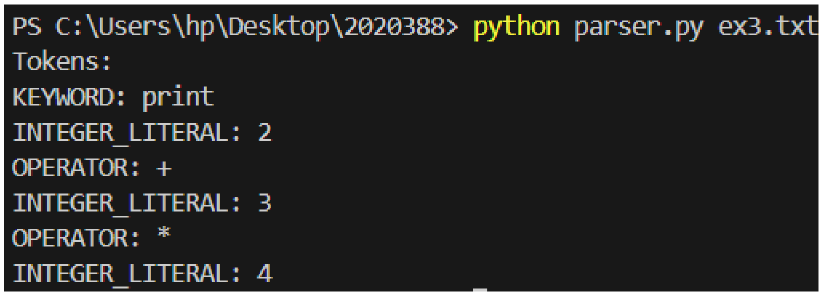
```
PS C:\Users\hp\Desktop\2020388> python parser.py ex1.txt
Tokens:
IDENTIFIER: x
OPERATOR: =
INTEGER_LITERAL: 5
IDENTIFIER: y
OPERATOR: =
INTEGER_LITERAL: 3
IDENTIFIER: z
OPERATOR: =
IDENTIFIER: x
OPERATOR: *
IDENTIFIER: y
KEYWORD: print
IDENTIFIER: z
```

Example2:



```
PS C:\Users\hp\Desktop\2020388> python parser.py ex2.txt
Tokens:
KEYWORD: if
KEYWORD: true
KEYWORD: print
IDENTIFIER: True
IDENTIFIER: branch
KEYWORD: else
KEYWORD: print
IDENTIFIER: False
IDENTIFIER: branch
```

Example3:



```
PS C:\Users\hp\Desktop\2020388> python parser.py ex3.txt
Tokens:
KEYWORD: print
INTEGER_LITERAL: 2
OPERATOR: +
INTEGER_LITERAL: 3
OPERATOR: *
INTEGER_LITERAL: 4
```