Compiler Construction CS 424

Assignment #2

Abdullah Farooq - 2020023

Introduction

This report presents the design, implementation, and testing of a parser for a simple expression and statement grammar. The parser is implemented in Python as a top-down recursive descent parser. The purpose of this parser is to analyze input programs written in the defined grammar and produce a structured representation of the program's syntax.

Grammar

The final grammar used for this parser is as follows:

- stmt_list -> stmt stmt_list | ε
- stmt -> assignment_stmt | if_stmt | print_stmt
- assignment_stmt -> VARIABLE = expr ;
- if_stmt -> if (expr comparison_op expr) { stmt_list } else { stmt_list }
- print_stmt -> print (expr) ;
- expr -> term expr'
- expr' -> + term expr' | - term expr' | ε
- term -> factor term'
- term' -> * factor term' | / factor term' | ε
- factor -> NUMBER | VARIABLE | (expr)
- comparison_op -> < | > | <= | >= | ==
- NUMBER -> [0-9]+
- VARIABLE -> [a-zA-Z]+

Design Decisions

- Tokenization: The input program is tokenized using regular expressions to break it into meaningful tokens such as identifiers, numbers, operators, and special characters.
- Top-Down Parsing: The parser is implemented as a top-down, recursive descent parser. This means it starts parsing from the top-level rules and recursively breaks down the input based on grammar rules.
- Recursive Functions: To handle nested expressions and statements, recursive functions are used for parsing sub-expressions and sub-statements.
- Error Handling: The parser raises an exception (`Syntax Error`) if it encounters an unexpected token or if the input does not conform to the grammar rules.

Structure of the Parser

- `Parser` Class:
  - `__init__`: Initializes the parser with the input program and tokenizes it.
  - `tokenize`: Tokenizes the input program using regular expressions.
  - Parsing Methods:
    - `advance`: Moves to the next token in the input.
    - `error`: Raises a syntax error.
    - `expr`: Parses an expression following the grammar rules.
    - `term`, `factor`, `number`, `variable`: Helper methods for parsing different parts of an expression.
    - `assignment_stmt`, `if_stmt`, `print_stmt`: Methods for parsing assignment statements, if statements, and print statements.
    - `stmt`: Determines the type of statement and calls the corresponding parsing method.
    - `stmt_list`: Parses a list of statements.
    - `parse`: Entry point for parsing, starts with `stmt_list`.

How to Run the Program

- Instructions:
  1. Copy the `Parser` class and the `test_parser` function into a Python script or interactive environment.
  2. Define or modify test cases inside the `test_parser` function to test the parser.
  3. Run the script or execute the code.
  4. The output will display the parsed results for each test case or any syntax error encountered.

Test Cases

The following test cases demonstrate the capabilities of the parser, including edge cases:

1. `a = 5;`
   - Tests a basic assignment statement.

2. `x = 5; print(x / 5);`
   - Tests an assignment statement followed by a print statement with a division operation.

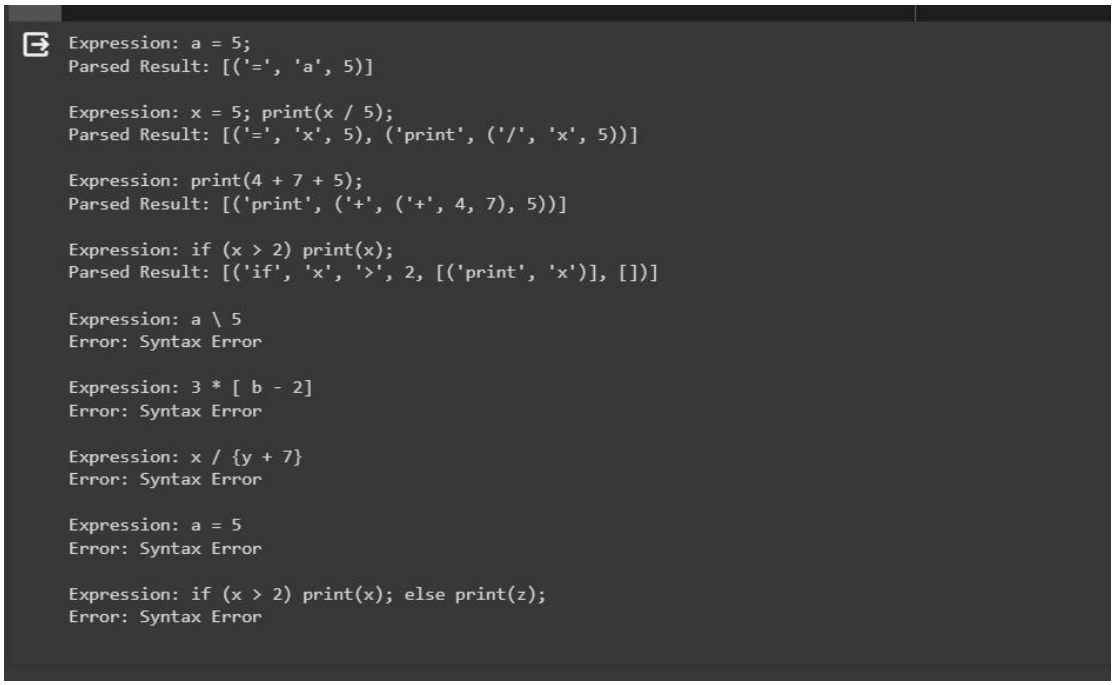
3. `print(4 + 7 + 5);`

- Tests a print statement with an arithmetic expression.

4. `if (x > 2) print(x);`
   - Tests a simple if statement with a print statement inside the if block.

5. `a \ 5`, `3 * [ b - 2]`, `x / {y + 7}`, `a = 5`
   - These are invalid syntax cases to demonstrate how the parser handles errors.

6. `if (x > 2) print(x); else print(z);`
   - Tests an if-else statement with a variable `z` in the else block. This case is mentioned as valid syntax but not parsed accurately.



```
Expression: a = 5;
Parsed Result: [('=', 'a', 5)]

Expression: x = 5; print(x / 5);
Parsed Result: [('=', 'x', 5), ('print', ('/', 'x', 5))]

Expression: print(4 + 7 + 5);
Parsed Result: [('print', ('+', ('+', 4, 7), 5))]

Expression: if (x > 2) print(x);
Parsed Result: [('if', 'x', '>', 2, [('print', 'x')], [])]

Expression: a \ 5
Error: Syntax Error

Expression: 3 * [ b - 2]
Error: Syntax Error

Expression: x / {y + 7}
Error: Syntax Error

Expression: a = 5
Error: Syntax Error

Expression: if (x > 2) print(x); else print(z);
Error: Syntax Error
```

Conclusion

In conclusion, the parser successfully parses input programs written in the defined grammar. It follows a top-down recursive descent parsing strategy to analyze the input and produce structured representations of expressions and statements. The parser is capable of handling various valid expressions and statements, as well as detecting and reporting syntax errors for invalid input.