Asian Institute of Technology

AT70.07

Programming Language and Compilers

Professor: Phan Minh Dung

Calculator Assignment : 2024

Phone Myint Zaw - st124359

Github link of the project

https://github.com/phonemz/compiler-calculator

Grammar of the language

Lexer

The lexer defines the tokens our parser will recognize: NUMBER, PLUS, TIMES, LPAREN (left parenthesis), and RPAREN (right parenthesis). It ignores spaces and tabs (' \t') because they don't affect the meaning of the expressions.

Parser

It defines the precedence of operators: PLUS has higher precedence than TIMES, meaning addition will be evaluated before multiplication.

There are rules for different types of expressions:

expr PLUS expr represents addition.

expr TIMES expr represents multiplication.

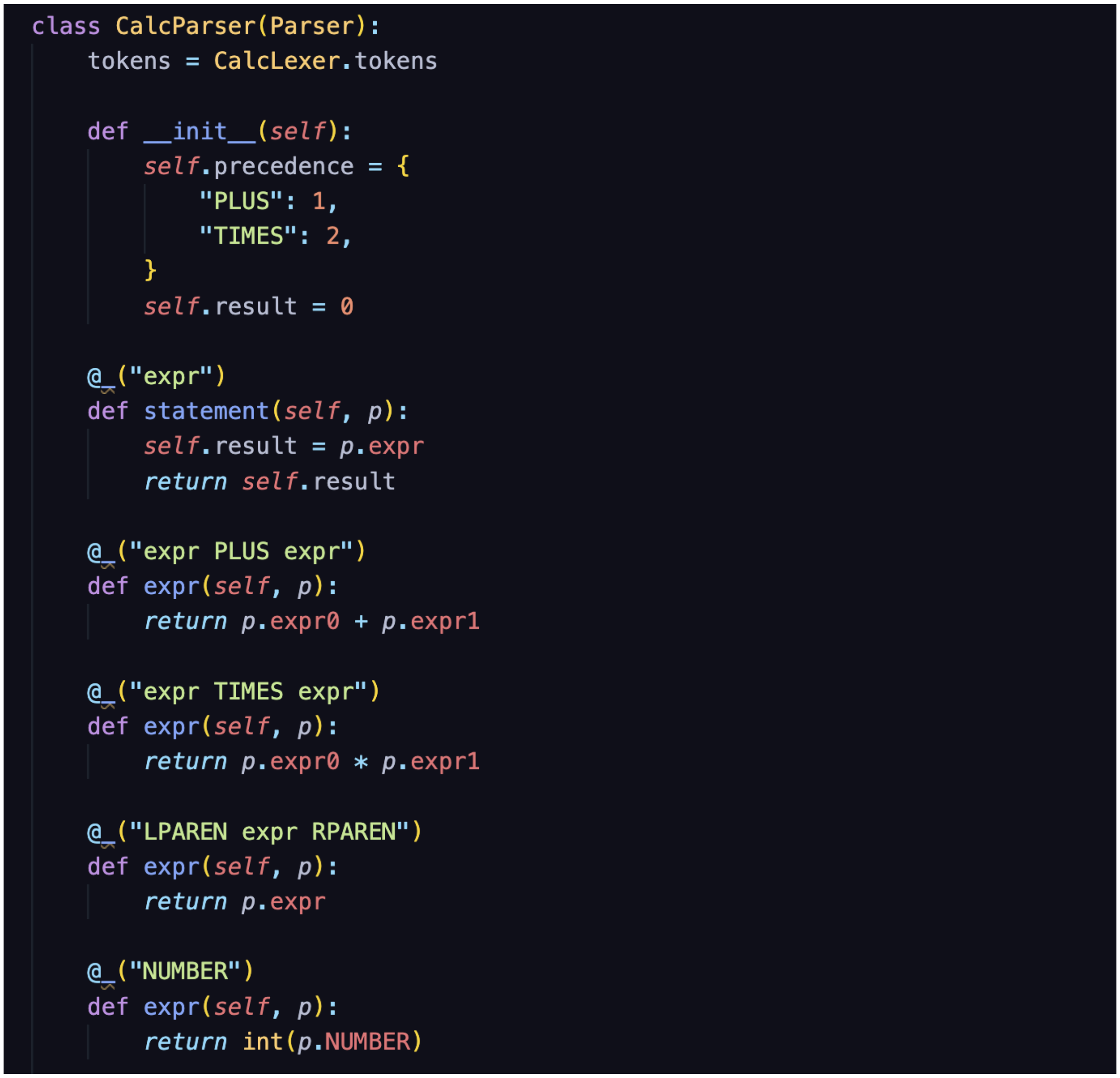
LPAREN expr RPAREN handles expressions within parentheses.

NUMBER represents individual numbers.

The error method handles syntax errors, providing meaningful error messages.

Example Expression

For example, given the expression 2 + 1 * 3. The addition operation 2+1 will be evaluated first due to its higher precedence. The result of 2+1 (which is 3) will then be multiplied to 3, resulting in the final calculation of 3*3, resulting 9.



```python
class CalcParser(Parser):
    tokens = CalcLexer.tokens

    def __init__(self):
        self.precedence = {
            "PLUS": 1,
            "TIMES": 2,
        }
        self.result = 0

    @_("expr")
    def statement(self, p):
        self.result = p.expr
        return self.result

    @_("expr PLUS expr")
    def expr(self, p):
        return p.expr0 + p.expr1

    @_("expr TIMES expr")
    def expr(self, p):
        return p.expr0 * p.expr1

    @_("LPAREN expr RPAREN")
    def expr(self, p):
        return p.expr

    @_("NUMBER")
    def expr(self, p):
        return int(p.NUMBER)
```

Parser Type

I've utilized the SLR(1) parser for my project, and it has proven to be a reliable choice. Its simplicity in implementation makes it accessible, while its ability to handle a variety of context-free grammars ensures versatility. Despite its simplicity, the SLR(1) parser demonstrates efficiency in parsing tasks, providing an effective solution for analyzing input strings according to defined grammatical rules.

Method of translation

The method of translation involves recursively evaluating the parsed expression to calculate the result.

Prefix notation is generated by traversing the expression tree in a pre-order manner, placing operators before operands.

Conversely, postfix notation is generated by traversing the expression tree in a post-order manner, placing operators after operands. This approach ensures that expressions are correctly represented in both prefix and postfix notations.

The integration of the parser and translation

The integration of the parser and translation in the project occurs seamlessly within the parsing process. After parsing an expression, the result is passed to the translation function, which generates equivalent prefix and postfix expressions based on the parsed structure. This integrated approach ensures that parsed expressions are efficiently converted into alternative notations while preserving the grammar-defined structure and semantics.

How to run the application

Install the required dependencies and run `python3 main.py` command. A pop up GUI box will appear and you can use the calculator app now.

Conclusion

In conclusion, the development of the simple calculator application involved defining a grammar for arithmetic expressions, implementing a SLR(1) parser for parsing and evaluation, generating prefix and postfix notations, and providing a user-friendly interface for input and output. The project successfully demonstrated the application of parsing techniques in building practical software solutions.