Practical Exercise: Development of a Recursive Descent Interpreter

In this guided practical exercise, we will approach the design of an Interpreter with basic resources to review the main concepts of a Recursive Descent Parser. To avoid dealing with a large and complicated grammar, we will restrict the domain to the typical arithmetic expression calculator. This way, we can obtain results with a reduced number of production rules.

We will begin with a very elementary approach, and complicate it in successive steps:

1. A parser for very simple operations.
2. A calculator for very simple operations (Parser + Semantic Routines).
3. Inclusion of expressions with parentheses.
4. Inclusion of operator precedence, and unary signs.

1. A Parser for very simple operations

The simplest approach corresponds to an input sequence of integer operands and arithmetic operators (+, -, * /) that ends by a line break. No precedence or associativity will be considered. Neither will we try to obtain the numerical result of the expression. We just want to check the syntax correction of the input. It should be able to recognize the following sequences:

```
1+2*3
       OK
3*2+1
       OK
3
       OK
```

2. A calculator for very simple operations (Parser + Semantic Actions)

We now need to elaborate Parser Analyzer that is able to check the syntactic correction of the input, and in addition, to evaluate the sequence and return the numeric value of the expression. It should be able to recognize and evaluate the following sequences:

```
1+2*3
       Value 7 OK
3*2+1
       Value 9 OK
3
       Value 3 OK
```


Juan Manuel Alonso Weber

3. Including expressions with parentheses

The next change is proposed to see how to modify the code when the grammar is expanded. We are going to limit ourselves to operate with expressions including parentheses. It should be able to recognize and evaluate the following sequences:

```
1+2*3
      Value 7
3*2+1
      Value 9
3
      Value 3
1+(2*3)
      Value 7
3*(2+1)
      Value 9
(3)*(2)+(1)
      Value 9
(3)
      Value 3
```

4. Inclusion of operator precedence, and unary signs

The last section proposes the extension of the calculator to

- Accept unary signs.
- Apply the adequate precedence to the arithmetic operators.