Project Scope

Project Title: C Compiler on Elixir

Client: Ortigoza Márquez Norberto J.

Authors: Ruiz Aguilar Eduardo	(Project Manager)
Aguilera Ortíz Alfredo	(Integrator)
Hernández Escobar Oswaldo	(Tester)
Rodríguez García Dulce Coral	(Architect)

Due Date: MAY 29, 2020
Version: 3.0
Circulation: Internal

In this new installment we added binary operations to support basic arithmetic. It was very important because the precedence of the operator has to be analyzed correctly, as well as the associativity. Below are the operators that take two values +, -, *, /.

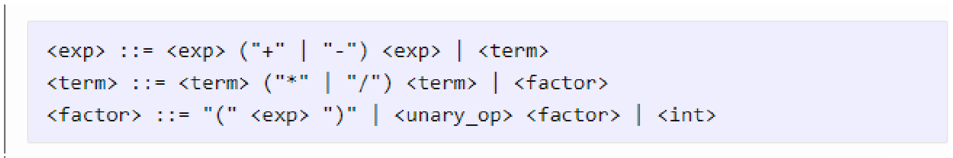
Compared to the previous installment, some additional tokens need to be admitted, complemented the list of tokens that already had.

Addition +

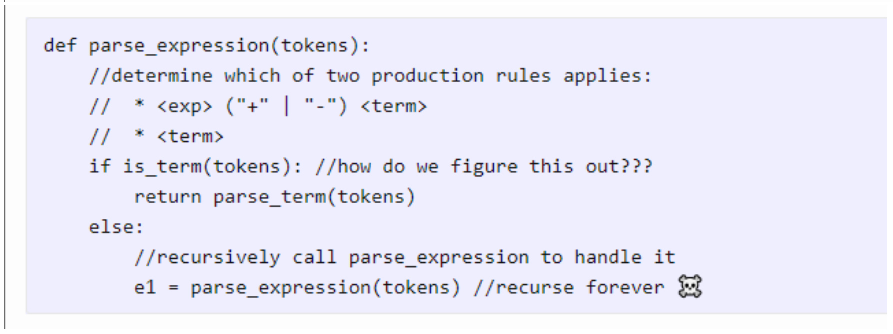
Multiplication *

Division /

The structure that the tree was going to have had to be analyzed, as an initial guide to the tree, what was proposed by Nora was taken into account.



```
<exp> ::= <exp> ("+" | "-") <exp> | <term>
<term> ::= <term> ("*" | "/") <term> | <factor>
<factor> ::= "(" <exp> ")" | <unary_op> <factor> | <int>
```



```
def parse_expression(tokens):
    //determine which of two production rules applies:
    //  * <exp> ("+" | "-") <term>
    //  * <term>
    if is_term(tokens): //how do we figure this out???
        return parse_term(tokens)
    else:
        //recursively call parse_expression to handle it
        e1 = parse_expression(tokens) //recurse forever ☠️
```

The main problem is knowing which are the operations that by priority must be carried out first, this was solved with recursion taking into account what the last term is going to be, verifying if it is actually an integer, otherwise we will do the analysis again. which is the operator that our compiler will analyze by priority.