AUTOMATA PROJECT REPORT

Presented to : DR/ Mahmoud Ali

SIMPLE CALCULATOR

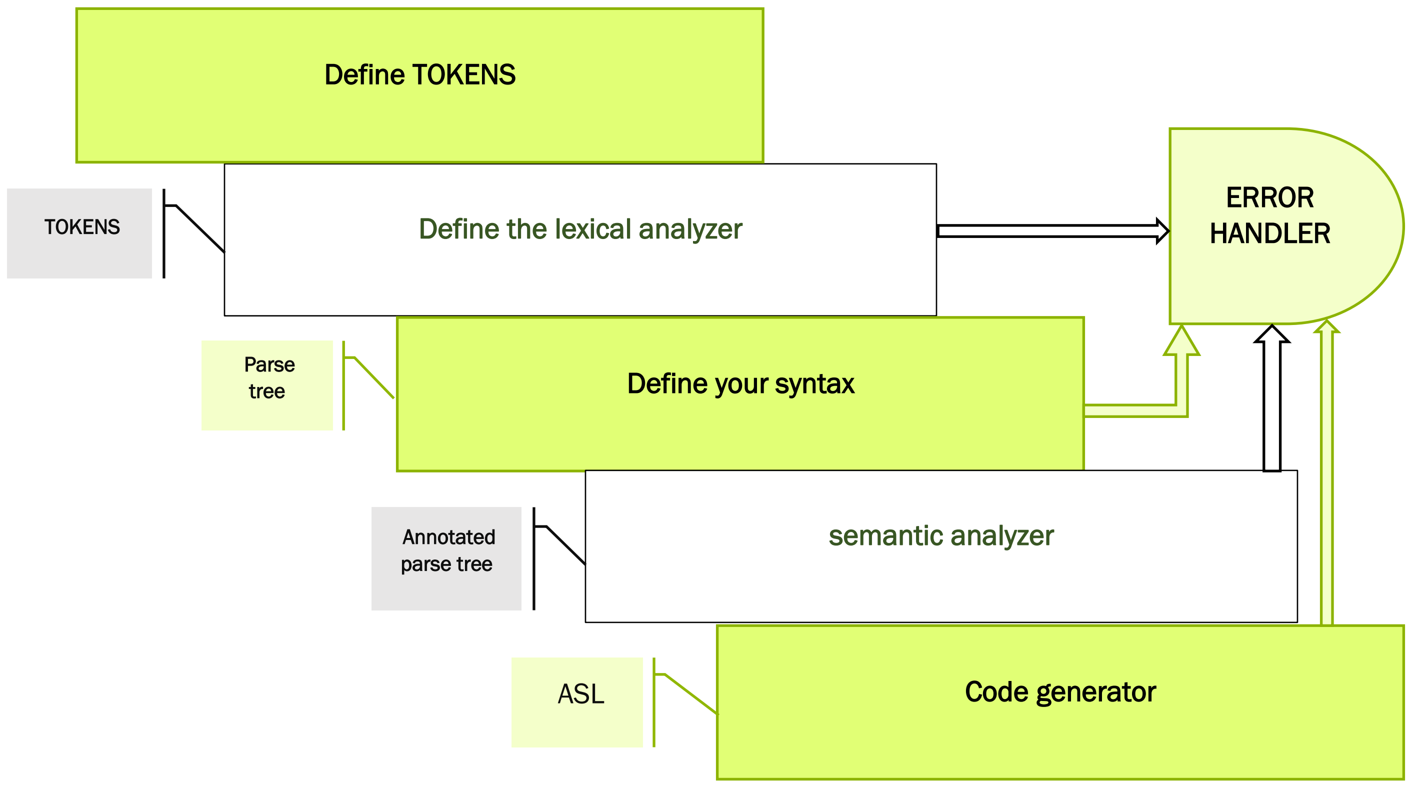
Contents

Presented from : Abdelrahman Gamal Mahdy

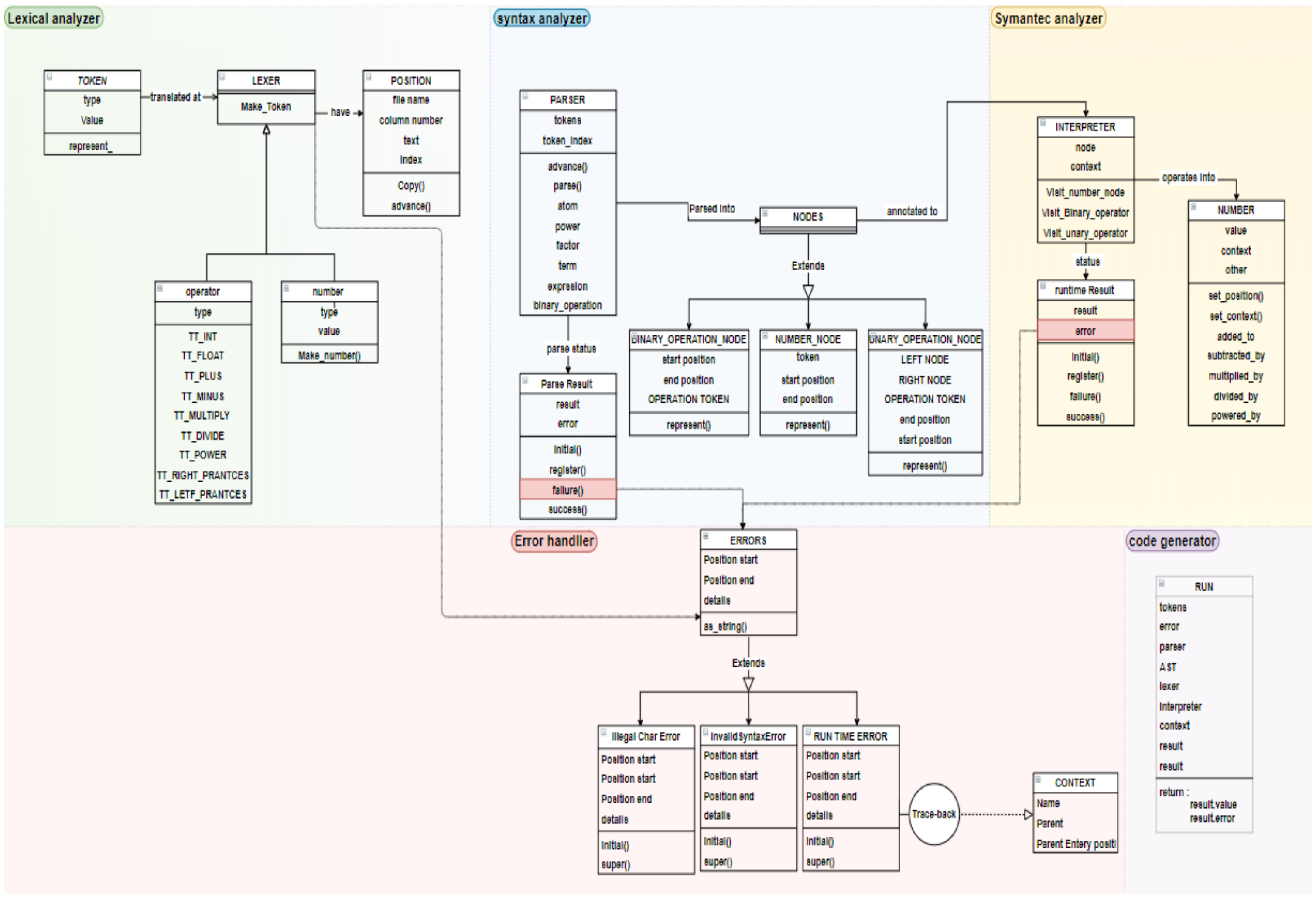
Project explanation

The project is a simple a compiler with so simple language which is the numbers and the basic operation .

The design main blocks



The detailed class diagrams



**Please check the html copy for better quality .

Presented from : Abdelrahman Gamal Mahdy

Stages explaination:

Tokens:

For simple calculator, the language tokens “characters”

Is {integers , floats , plus , minus , multiply , divide , power , left parentheses , right parentheses }.

CONSTANTS

Will be the digits or the numbers *{0,1,2,3,4,5,6,7,8,9}* .

Lexical analyzer

Will turns the meaningless string into a flat list of two things :"numbers , "operator". Which :

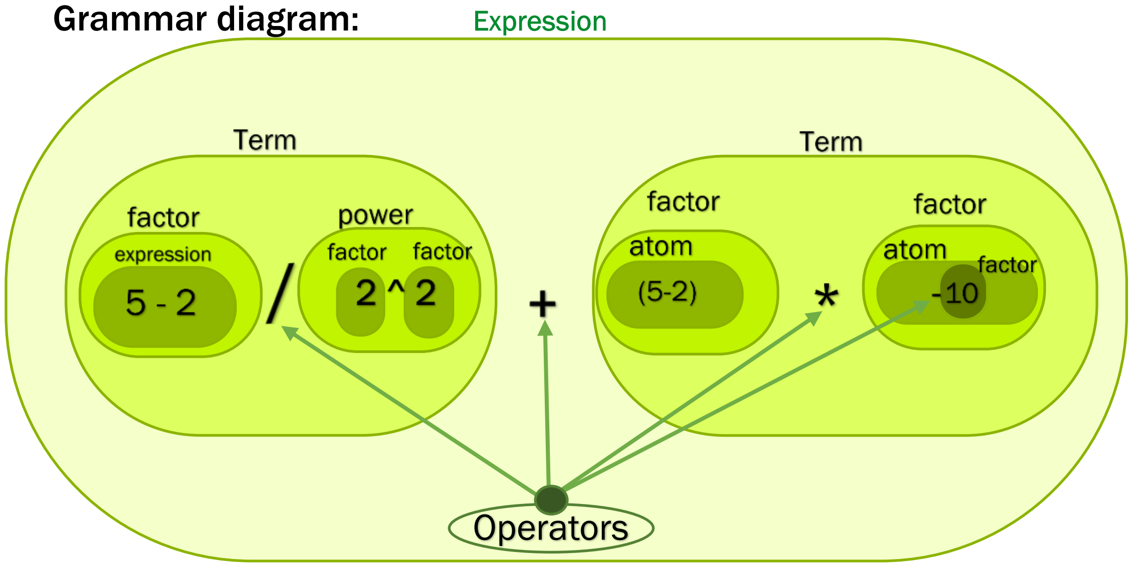
- number has value and type
- operator has only type

so, I had to initiate the value with None as default .

syntax analyzer

parser will take the input from a **lexical analyzer** in the form of token streams. The parser analyzes the source code (token stream) against the grammar to detect any errors in the code. The output of this phase is a parse tree ↓.

Grammar diagram:



Semantic analyzer

The Interpreter uses syntax tree (parse tree) and symbol table (constants) to check whether the given program is semantically consistent with language definition. It gathers type information and stores it in either syntax tree or symbol table.

Position

Positioning is very important for tracing back is there's any error occurring, so every token and every part of code have :

- Position index with : line number , column number , file name .
- Context : which traces the parent for each function to be executed .

Error handlers :

There are three types of errors (for each phase | stage) :

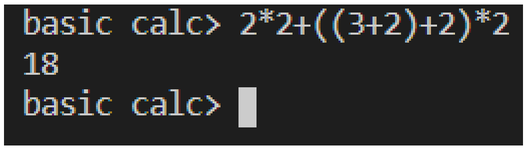
- Lexical error : which shows **Illegal Char Error**
- Syntax error : which shows **Invalid Syntax Error**
- Runtime error : which shows the runtime error **like divide by zero**

When tracing back the error its shown by an arrow ^ or group of arrows from starting position till the end and tracing back the function parents

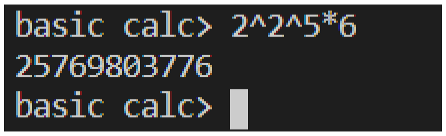
Code generator

Run shell or code generator which generates and executes the code from the ANNOTATED PARSE TREE and the Abstract Syntax Tree (AST) by creating object of the input and pass the stages checking errors every time.

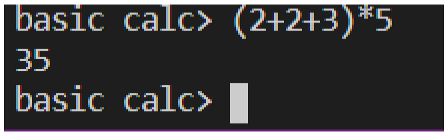
EXAMPLES



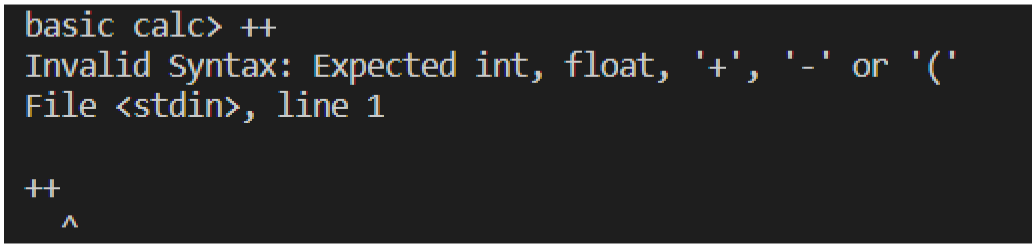
```
basic calc> 2*2+((3+2)+2)*2
18
basic calc> 
```



```
basic calc> 2^2^5*6
25769803776
basic calc> 
```

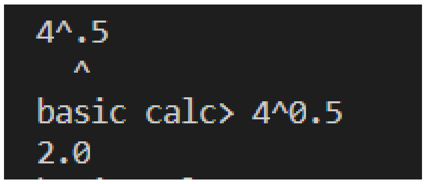


```
basic calc> (2+2+3)*5
35
basic calc> 
```

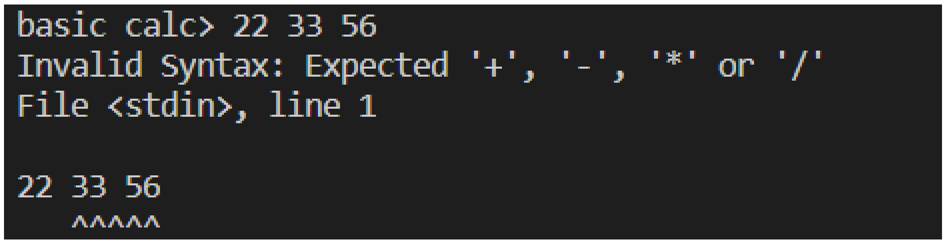


```
basic calc> ++
Invalid Syntax: Expected int, float, '+', '-' or '('
File <stdin>, line 1

++
 ^
```

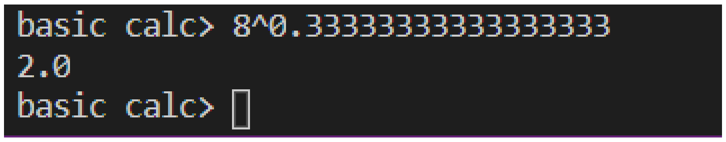


```
4^.5
  ^
basic calc> 4^0.5
2.0
```

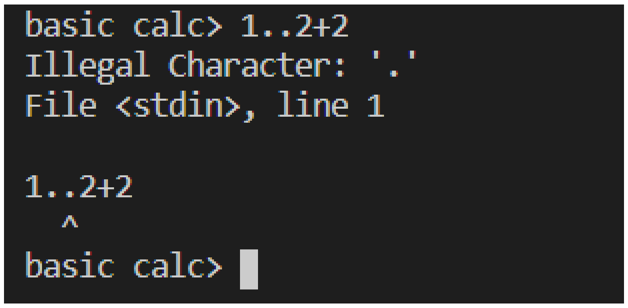


```
basic calc> 22 33 56
Invalid Syntax: Expected '+', '-', '*' or '/'
File <stdin>, line 1

22 33 56
   ^^^^^
```

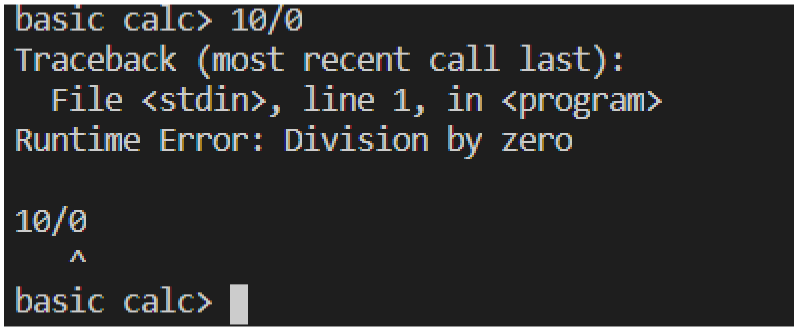


```
basic calc> 8^0.3333333333333333
2.0
basic calc> 
```



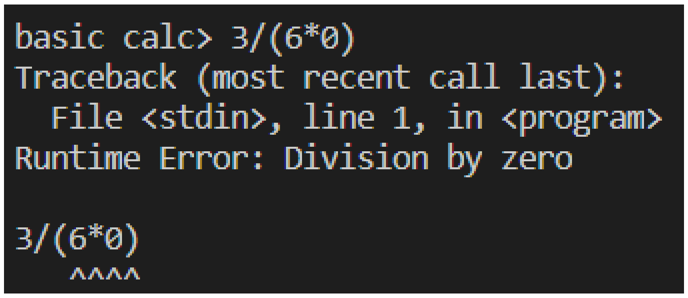
```
basic calc> 1..2+2
Illegal Character: '.'
File <stdin>, line 1

1..2+2
  ^
basic calc>
```



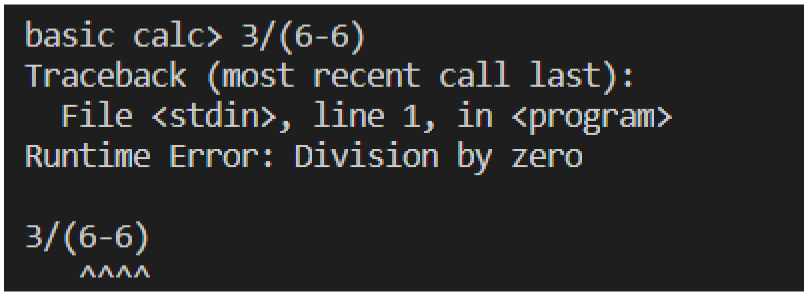
```
basic calc> 10/0
Traceback (most recent call last):
  File <stdin>, line 1, in <program>
Runtime Error: Division by zero

10/0
   ^
basic calc>
```



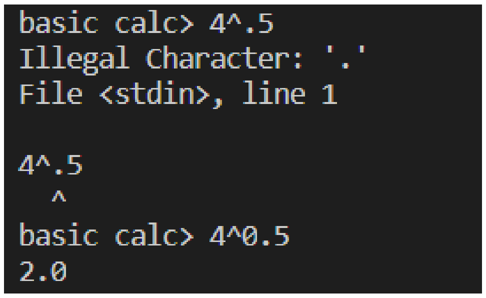
```
basic calc> 3/(6*0)
Traceback (most recent call last):
  File <stdin>, line 1, in <program>
Runtime Error: Division by zero

3/(6*0)
   ^^^^
```



```
basic calc> 3/(6-6)
Traceback (most recent call last):
  File <stdin>, line 1, in <program>
Runtime Error: Division by zero

3/(6-6)
   ^^^^
```



```
basic calc> 4^.5
Illegal Character: '.'
File <stdin>, line 1

4^.5
  ^
basic calc> 4^0.5
2.0
```