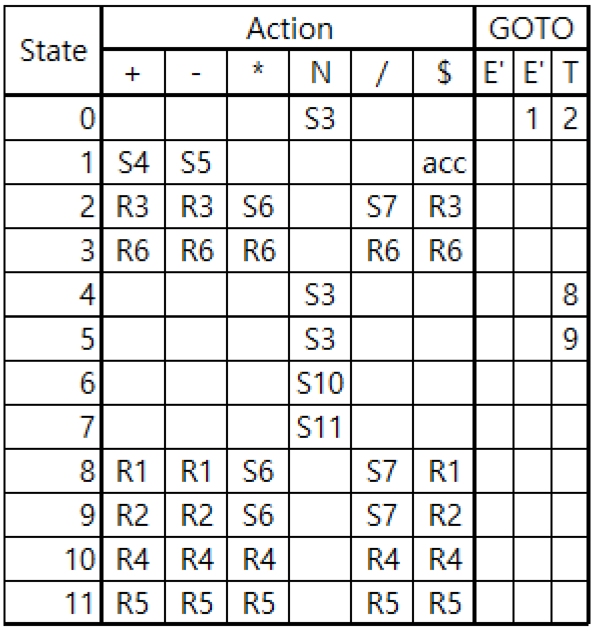
# Implementing of Parsing Algorithm

## 0. Parsing Table



| State | Action | | | | | | GOTO | | |
|---|---|---|---|---|---|---|---|---|---|
| | + | - | * | N | / | $ | E' | E' | T |
| 0 | | | | S3 | | | | 1 | 2 |
| 1 | S4 | S5 | | | | acc | | | |
| 2 | R3 | R3 | S6 | | S7 | R3 | | | |
| 3 | R6 | R6 | R6 | | R6 | R6 | | | |
| 4 | | | | S3 | | | | | 8 |
| 5 | | | | S3 | | | | | 9 |
| 6 | | | | S10 | | | | | |
| 7 | | | | S11 | | | | | |
| 8 | R1 | R1 | S6 | | S7 | R1 | | | |
| 9 | R2 | R2 | S6 | | S7 | R2 | | | |
| 10 | R4 | R4 | R4 | | R4 | R4 | | | |
| 11 | R5 | R5 | R5 | | R5 | R5 | | | |

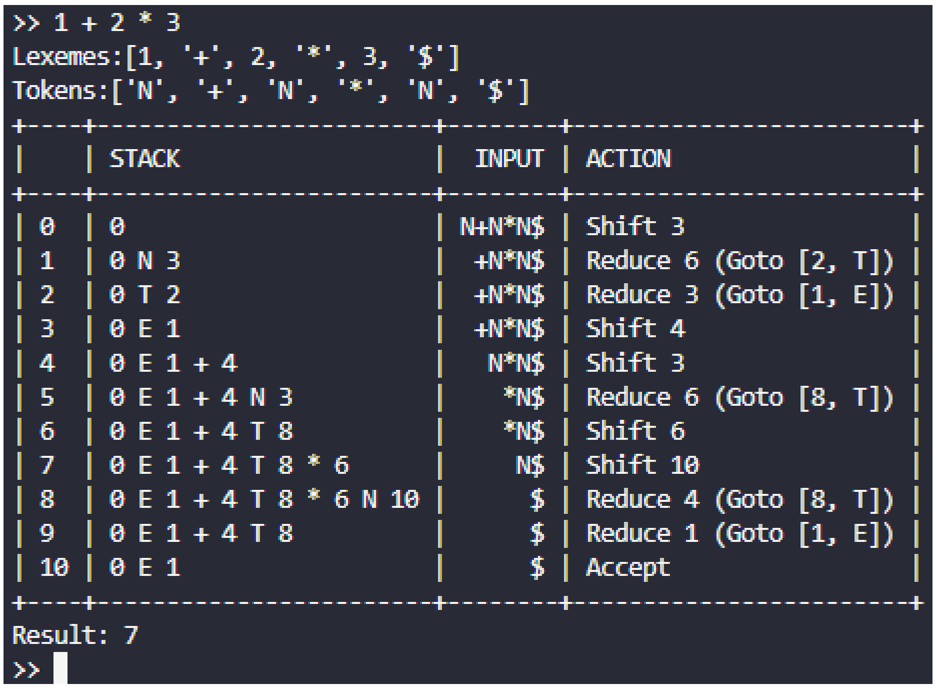
## 1. Lexical Analyzer

- Implemented in `lexer.py'
  - number : N
  - operator : +, -, *, /
  - end of input : $
  - other : unknown
- Result of Lexical Analyzer is in 2 and 3.

## 2. Shift-Reduce Algorithm

- Implemented in `LRParser.py`, and set variable `SA` in `SyntaxAnalyzer.py` to `LRParser`.

### Result



```
>> 1 + 2 * 3
Lexemes:[1, '+', 2, '*', 3, '$']
Tokens:['N', '+', 'N', '*', 'N', '$']
+----+------------------------+--------+------------------------+
|    | STACK                  |  INPUT | ACTION                 |
+----+------------------------+--------+------------------------+
| 0  | 0                      | N+N*N$ | Shift 3                |
| 1  | 0 N 3                  |  +N*N$ | Reduce 6 (Goto [2, T]) |
| 2  | 0 T 2                  |  +N*N$ | Reduce 3 (Goto [1, E]) |
| 3  | 0 E 1                  |  +N*N$ | Shift 4                |
| 4  | 0 E 1 + 4              |   N*N$ | Shift 3                |
| 5  | 0 E 1 + 4 N 3          |    *N$ | Reduce 6 (Goto [8, T]) |
| 6  | 0 E 1 + 4 T 8          |    *N$ | Shift 6                |
| 7  | 0 E 1 + 4 T 8 * 6      |     N$ | Shift 10               |
| 8  | 0 E 1 + 4 T 8 * 6 N 10 |      $ | Reduce 4 (Goto [8, T]) |
| 9  | 0 E 1 + 4 T 8          |      $ | Reduce 1 (Goto [1, E]) |
| 10 | 0 E 1                  |      $ | Accept                 |
+----+------------------------+--------+------------------------+
Result: 7
>>
```

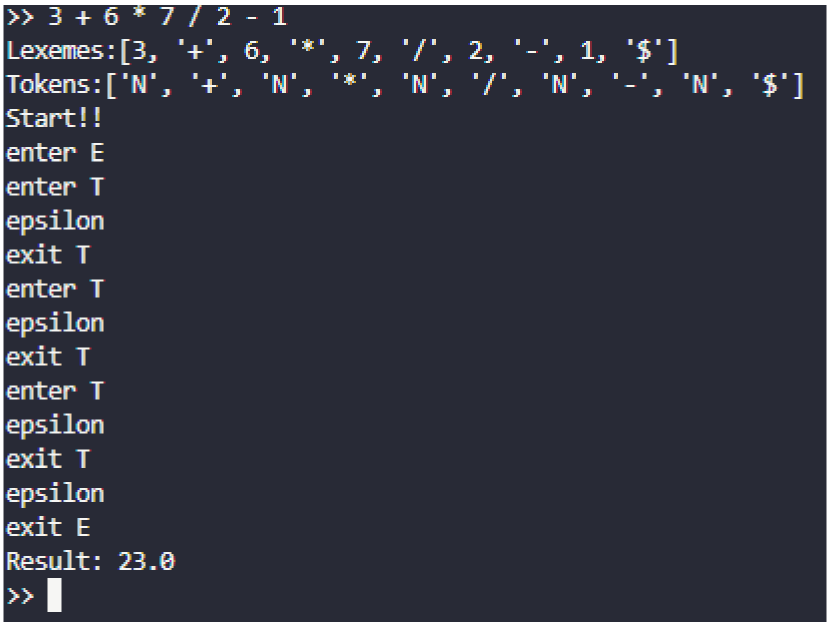
## 3. Recursive Descent Parsing

- Implemented in `LLParser.py`
- Using EBNF Grammer

```
E ::= T { + T | - T }
T ::= N { * N | / N }
N ::= number
```

- Set variable `SA` in `SyntaxAnalyzer.py` to `LLParser` to use this algorithm.

### Result



```
>> 3 + 6 * 7 / 2 - 1
Lexemes:[3, '+', 6, '*', 7, '/', 2, '-', 1, '$']
Tokens:['N', '+', 'N', '*', 'N', '/', 'N', '-', 'N', '$']
Start!!
enter E
enter T
epsilon
exit T
enter T
epsilon
exit T
enter T
epsilon
exit T
epsilon
exit E
Result: 23.0
>>
```

## 4. Test

- Test program is in `test.py`
  - Test for Lexical Analyzer : run, empty input, single input, random input, invalid input
  - Test for LL Parser : run, empty input, single input, zero division, random input, invalid input
  - Test for LR Parser : same as LL Parser