# Lab 3

**Daniel Gisolfi**

## Crafting a Compiler

### 4.7

#### Grammer

$$\begin{gathered}
Start => E\$ \\
E => T + E \\
=> T \\
T => T * F \\
=> F \\
F => (E) \\
=> num
\end{gathered}$$

Leftmost Derivation of...

$$\begin{gathered}
num + num * num + num\$ \\
start => E\$ \\
E\$ => T + E\$ \\
T + E\$ => T + T * F\$ \\
T + T * F\$ => T + T * (E)\$ \\
T + T * (E)\$ => T + T * T + E\$ \\
T + T * T + E\$ => T + T * T + T\$ \\
T + T * T + T\$ => num + T * T + T\$ \\
num + T * T + T\$ => num + num * T + T\$ \\
num + num * T + T\$ => num + num * num + T\$ \\
num + num * num + T\$ => num + num * num + num\$
\end{gathered}$$

### 5.2

$$\begin{gathered}
Start \rightarrow Value\$ \\
Value \rightarrow num \\
\rightarrow LParen Expr RParen \\
Expr \rightarrow plus Value Value \\
\rightarrow prod Values \\
Values \rightarrow Value Values \\
\rightarrow \lambda
\end{gathered}$$

```python
 def consume(token):
        cur_token = tokens.pop(0)

def match(cur_token, expected_token):
    retval = False
    if cur_token is expected_token:
        consume(cur_token)
        retval = True
     return retval

def parse():
    parseValue()
    match(token, 'T_EOP')

def parseValue():
    if match(current_token, 'T_NUM'):
        return True
    elif match(current_token, 'T_L_PAREN'):
        parseExpr()
        if match(current_token, 'T_R_PAREN'):
            return True
        else:
             error()
    else:
        error()

def parseExpr():
     if match(current_token, 'T_ADDITION_OP'):
        parseValue()
        parseValue()
     elif match(current_token, 'T_PROD_OP'):
        parseValues()
      else:
        error()

def parseValues()
    if match(current_token, 'T_NUM') or  match(current_token, 'T_L_PAREN'):
        parseValue()
        parseValues()
    else:
        error()
```

## Dragon

### 4.2.1 ...My version had no a or b or c

1. Left Most Derivation

```
S = SS*
SS* => SS+S*
SS+S* => aS+S*
aS+S* => aa+S*
aa+S* => aa+a*
```

2. Right Most Derivation

```
S => SS*
SS* => Sa*
Sa* => SS+a*
SS+a* => Sa+a*
Sa+a* => aa+a*
```

3. CST

```
-[S]
--[S]
---[S]
----[a]
---[S]
----[a]
---[+]
--[S]
---[a]
--[*]
```