Um analisador Sintático p/ Expressões c/ Precedência

Gramática:

$E \to n$

$E \to (\ E\ )$

$E \to E - E$

$E \to E * E$

Construtor:

```
Exp Num (num)
    N/A

Exp Bin ('-', e1, e2)

Exp Bin ('*', e1, e2)
```

Gramática Sem Ambiguidade:

$E \to E - T$

$E \to T$

$T \to T * F$

$T \to F$

$F \to$ '(' E ')'

$F \to$ 'n'

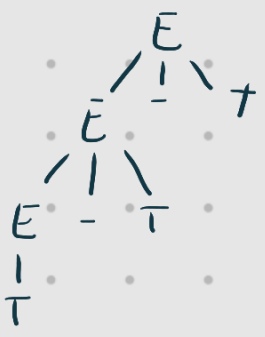
```
func ParseF():
    if prox == '(':
        come ('(')
        e = ParseE()
        come (')')
        return e
    else if prox == 'n':
        num = come ('n')
        return ExpNum (num)
```

else:
    ERRO! Esperava encontrar uma expressão mas encontrei um $prox.

---

func consc(toktype)
    se prox == toktype:
        avança a posição de leitura
        retorna o valor associado ao token
    else:
        ERRO! Esperava $toktype
              Encontrei $prox.

$E \Rightarrow E - T \Rightarrow E - T - T \Rightarrow T - T - T$



OBS: Existe uma diferença de utilização da gramática de recursão e a gramática que queremos

---

func eval(e):
    se exp é ExpNum:
        return e.num
    se exp é ExpBin e e.op == '-'
        return eval(e.e1) - eval(e.e2)
    se exp é ExpParen:
        return eval(e.exp)

Exemplo 10 - 5 * 3

⇓

Ambiguidade →
ExpBin(*, ExpBin(-, ExpNum(10), ExpNum(5)), ExpNum(3))
ou
ExpBin(-, ExpNum(10), ExpBin(*, ExpNum(5), ExpNum(3)))

```
func parseE()
    e = parseT()
    while prox == '-':
        come('-')
        e2 = parseT()
        e = ExpBin('-', e, e2)
    return e
```

n - n - n - n

↳ Não conseguimos diferenciar olhando de cima para baixo, então vamos olhar de baixo para cima

```
func parseT():
    e = parseF()
    while prox == '*':
        come('*')
        e2 = parseF()
        e = ExpBin('*', e, e2)
    return e
```

1 * 2 - 3 * 4