3.1: $LL(k)$ und $LR(k)$

Es sei die folgende Grammatik zum Parsen von Ausdrücken in der sog. *Polnischen Notation* gegeben, die sowohl die Kriterien für eine $LL(1)$- als auch für eine $LR(0)$-Grammatik erfüllt (Produktionen in Großbuchstaben seien dabei die Regeln für den Lexer):

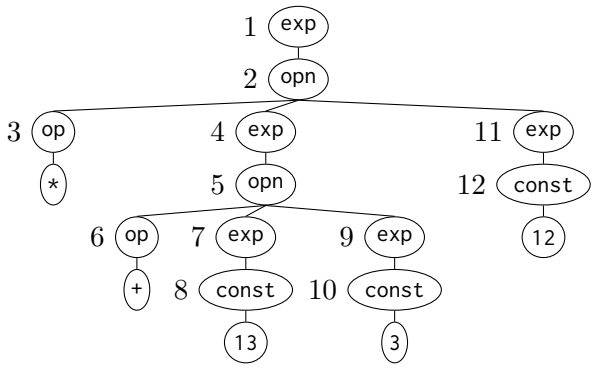
```
PLUS: '+';
MINUS: '-';
MUL: '*';
DIV: '/';
NUMBER: [0-9]+;
WHITESPACE: '␣'+; // skip

expression: constant | operation ; // start
constant: NUMBER ;
operation: operator expression expression ;
operator: PLUS | MINUS | MUL | DIV ;
```

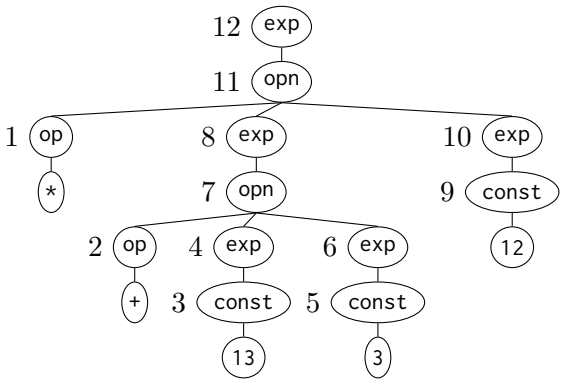
Mit Hilfe dieser Grammatik soll die folgende Eingabe geparst werden:

```
* + 13 3 12
```

Beim Parsen mit einem LL-Parser (links) bzw. einem LR-Parser (rechts) werden die folgenden Strukturbäume erzeugt (die Nummern neben den Nicht-Terminalen geben die Reihenfolge der Anwendung der Regeln an):



(für die Entscheidung bzgl. der Alternativen von `expression` und `operator` brauchen wir ein Token Vorausschau $\rightsquigarrow LL(1)$)



(wir können alleine anhand der bereits auf den Stack geschobenen Token entscheiden, was zu tun ist $\rightsquigarrow LR(0)$)

Erkennbar: Beim LL-Parsen wird der Strukturbaum in *Pre-Order-Reihenfolge* aufgebaut, beim LR-Parsen hingegen in *Post-Order-Reihenfolge*.