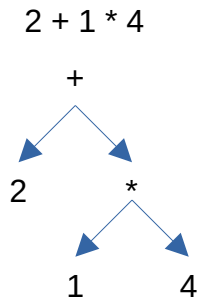
Da die Multiplikation Vorrang vor der Addition hat, ist hier + das Wurzelelement da die der mittlere sowie der rechte Ausdruck stärker durch die Multiplikation an einander gebunden werden.

Wurzelelement => +
e1 -> (intExp) 2
e2→ (MultExp) „*“



Drawing 1:

2 (IntExp ist immer ein Endknoten)
e1→ none
e2→ none

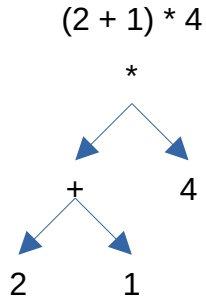
*:
e1→ (IntExp) 1
e2→ (IntExp) 4

Rekursion :

‚+‘: pretty()
‚2‘: e1→ pretty()
‚*‘: e2→ pretty()
‚1‘ : pretty()
‚4‘ : pretty()

Die Multiplikation hat zwar Vorrang vor der Addition, aber hier ist die Addition durch die Klammerung vor der Multiplikation „geschützt“. Somit wird jetzt die Multiplikation zu dem Wurzelelement.

Wurzelelement => * (MultExp)
e1 → (PlusExp) „+“
e2 → (IntExp) 4



Drawing 2:

+ (PlusExp)
e1 → (IntExp) 2
e2 → (IntExp) 1

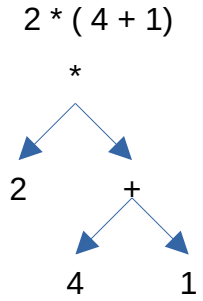
Rekursion :

„*“ : pretty()
‚+‘ : ‚(‘ + e1→pretty()
‚2‘ : pretty()
‚1‘ : pretty()
‚)‘
‚4‘: pretty()

Nun kann die Addition auch am rechten Rand stehen. Auch hier sorgt der Parser durch den entsprechenden Aufbau des Baumes dafür dass die eigentliche Ausführungsreihenfolge erhalten bleibt.

Wurzelelement => * (MultExp)
e1 → (IntExp) 4
e2 → (PlusExp) „+“

+ (PlusExp)
e1 → (IntExp) 2
e2 → (IntExp) 1



Drawing 3:

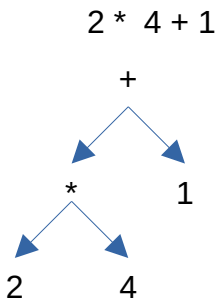
Rekursion :

‚*‘ → pretty()
‚4‘ : e1→ pretty()
‚+‘ : ‚(‘ + e2→ pretty()
‚2‘ : pretty()
‚1‘ : pretty()
‚)‘

Wird das ganze ohne Klammern notiert so wird auch hier das + zu dem Wurzelelement da die Multiplikation (mal wieder) stärker bindet.

Wurzelelement => + (PlusExp)
e1 → (MultExp) „*“
e2 → (IntExp) 1

* (MultExp)
e1 → (IntExp) 2
e2 → (IntExp) 4



Drawing 4:

Rekursion :

‚+‘: pretty()
‚*‘ : e1→ pretty()
‚2‘ : pretty()
‚4‘ : pretty()
‚1‘ : e2→ pretty()

Der Typ der beiden Blätter (e1 sowie e2) lässt sich einfach durch die Anweisung ***typeid*** feststellen (siehe Skript). Wer sich einen unnötigen Parameter „reinbastelt“, hat verloren.