# Compilerbau

**DHBW Campus Horb** **Übungsblatt 2**

Benutzen sie für diese Übung den Lexer aus `https://github.com/JanUlrich/DHBW_Compilerbau2020`. Im folgenden ist mit dem Datentyp `Token` der gleichnamige Typ aus `Lexer.x` gemeint.

## 1 Parser

a) Schreiben sie eine Funktion `evalOneExpression`, welche genau eine Expression auswertet:

```
evalOneExpression :: [Token] -> Int
--Beispiel
evalOneExpression [IntLiteral 4, PlusOperator, IntLiteral 3] -- 7
```

*Tipp: Diese Aufgabe ist auch noch ohne Parsen oder Parserkombinatoren umsetzbar und muss nicht für verschachtelte Ausdrücke funktionieren*

b) Ändern sie die Funktion `evalOneExpression` so ab, dass sie einen Parser zurück gibt

```
evalOneExpressionParser :: [Token] -> ParserResult Token Int
--Beispiel
evalOneExpression [IntLiteral 4, PlusOperator, IntLiteral 3] -- State 7 []
```

## 2 Parserkombinatoren

a) Baue einen Parser, welchen die `evalOneExpression` Funktion benutzt, um zwei Ausrücke hintereinander zu parsen. Im ersten Schritt soll er ein Paar mit den beiden Ergebnissen der Ausdrücke zurück geben.

```
evalTwoExpressions :: [Token] -> ParserResult Token (Int, Int)
--Beispiel
evalTwoExpressions (toTokens "1+3 4+5") -- State (4, 9) []
```

Benutzen sie hierzu den Kombinator `+.+`.

b) Ändern sie den Funktion `evalTwoExpressions` so ab, dass sie nun nur noch ein Ergebnis zurück gibt, welches die Addition der beiden Teilergebnisse darstellt.

```
evalTwoExpressions :: [Token] -> ParserResult Token Int
--Beispiel
evalTwoExpressions (toTokens "1+3 4+5") -- State 13 []
```

Benutzen sie hierfür den Parserkombinator `>>>`.

c) Schreiben sie zum Abschluss eine Parser `evalExpression`, welcher einen Ausdruck von beliebiger Länge auswerten kann. Er sollte auch Subtraktionen interpretieren.

```
evalExpression :: [Token] -> ParserResult Token Int
--Beispiel
evalExpression (toTokens "1 + 3 - 4 + 5 - 1") -- State 4 []
```