Abgabetermin: Mi in der KW 18

☐ **Gr. 1, Dr. G. Kronberger** **Name** _____ **Aufwand in h** _____
☐ **Gr. 2, Dr. H. Gruber**
☐ **Gr. 3, Dr. D. Auer** **Punkte** _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Transformation arithmetischer Ausdrücke (4 + 6 Punkte)

Wie Sie wissen, können einfache arithmetische Ausdrücke in der Infix-Notation, z. B. (*a* + *b*) * *c*, durch folgende Grammatik beschrieben werden:

```
Expr  =  Term { '+' Term | '-' Term } .
Term  =  Fact { '*' Fact | '/' Fact } .
Fact  =  number | ident | '(' Expr ')' .
```

Die folgende attributierte Grammatik (ATG) beschreibt die Transformation einfacher arithmetischer Ausdrücke von der Infix- in die Postfix-Notation, z. B. von (*a* + *b*) * *c* nach *a b* + *c* *.

```
Expr =
  Term
  { '+' Term     sem Write('+'); endsem
  | '-' Term     sem Write('-'); endsem
  } .
```

```
Term =
  Fact
  { '*' Fact     sem Write('*'); endsem
  | '/' Fact     sem Write('/'); endsem
  } .
```

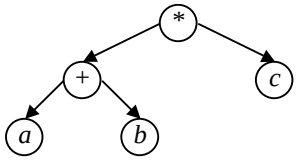
```
Fact =
  number ↑n      sem Write(n); endsem
  | ident ↑id    sem Write(id); endsem
  | '(' Expr ')' .
```

a) Entwickeln Sie eine ATG zur Transformation einfacher arithmetischer Ausdrücke von der Infix- in die Präfix-Notation, also z. B. von (*a* + *b*) * *c* nach * + *a b c*.

b) Implementieren Sie die ATG aus a) und testen Sie Ihre Implementierung ausführlich.

2. Arithmetische Ausdrücke und Binärbäume (4 + 6 + 1 + 3 Punkte)

Arithmetische Ausdrücke können im Hauptspeicher auch in Form von Binärbäumen dargestellt werden. Z. B. entspricht dem Infix-Ausdruck (*a* + *b*) * *c* der rechts dargestellte Binärbaum.



a) Entwickeln Sie eine ATG, die arithmetische Infix-Ausdrücke in Binärbäume (gemäß der Deklarationen unten) umwandelt.

```
TYPE
  NodePtr = Node;
  Node = RECORD
    left, right: NodePtr;
    txt: STRING, (*operator or operand, both in textual representation*)
  END; (*Node*)
  TreePtr = NodePtr;
```

b) Implementieren Sie die ATG aus a) und testen Sie Ihre Implementierung ausführlich.

c) Geben Sie die Ergebnisbäume durch entsprechende Baumdurchläufe *in-order*, *pre-order* und *post-order* aus: Was stellen Sie dabei fest?

d) Implementieren Sie eine rekursive Funktion

```
FUNCTION ValueOf(t: TreePtr): INTEGER;
```

die den Baum "auswertet", also den Wert des Ausdrucks berechnet, der durch den Baum repräsentiert wird. (Hinweis: In einem *post-order*-Baumdurchlauf zuerst den Wert des linken Unterbaums, dann den Wert des rechten Unterbaums berechnen und zum Schluss in Abhängigkeit vom Operator in der Wurzel den Gesamtwert berechnen).