**Abgabetermin: Mi, 03. 05. 2023**

☒ **Gr. 1,** Dr. S. Wagner ☐ **Gr. 2,** Dr. D. Auer ☐ **Gr. 3,** Dr. G. Kronberger

**Name** Elias Wurm **Aufwand in h** 1h 30min

**Punkte** _____ **Kurzzeichen Tutor / Übungsleiter*in** _____ / _____

## 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 =                                  Fact =
  Term                                    Fact                                    number ↑n   sem Write(n); endsem
  { '+' Term     sem Write('+'); endsem   { '*' Fact     sem Write('*'); endsem   | ident ↑id  sem Write(id); endsem
  | '-' Term     sem Write('-'); endsem   | '/' Fact     sem Write('/'); endsem   | '(' Expr ')' .
  } .                                     } .
```

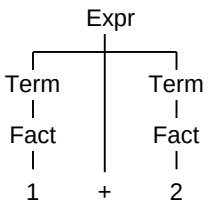
a) Entwickeln Sie eine ATG zur Transformation einfacher arithmetischer Ausdrücke von der Infix- in die Präfix-Notation, also z. B. von (*a* + *b*) * *c* nach * + *a b c*.

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

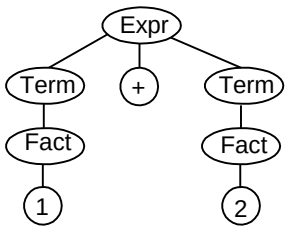
## 2. Syntaxbäume in kanonischer Form (4 + 6 + 4 Punkte)

Die Struktur von arithmetischen Ausdrücken kann auf Basis obiger Grammatik durch ihren Syntaxbaum dargestellt werden. Folgende Abbildungen zeigen zwei unterschiedliche Darstellungen des Syntaxbaums für den Beispielausdruck 1 + 2:

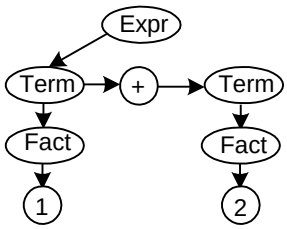
"Konventionelle" Darstellung




Baumdarstellung




Syntaxbäume sind somit Bäume, deren Knoten beliebig viele Nachfolgeknoten haben können. Will man Syntaxbäume in Form von dynamischen Datenstrukturen abbilden, tritt ein Problem auf: Wie viele Zeiger braucht ein Knoten? Eine einfache Implementierung für solche *allgemeinen Bäume* besteht darin, diese auf einen Spezialfall von *Binärbäumen* zurückzuführen, indem jeder Knoten einen Zeiger auf sein erstes „Kind“ (in der Komponente `firstChild`) und einen Zeiger auf die einfach-verkettete Liste seiner „Geschwister“ (in der Komponente `sibling`) hat. Jeder Knoten kommt dann mit zwei Zeigern aus, unabhängig davon, wie viele Geschwister er hat. Man nennt diese Darstellung *kanonische Form*. Der Syntaxbaum für das obige Beispiel sieht in kanonischer Form wie folgt aus:




(a) Entwickeln Sie aus der oben angegebenen Grammatik eine attributierte Grammatik (ATG), die für arithmetische Ausdrücke den Syntaxbaum in kanonischer Form aufbaut.

(b) Implementieren Sie diese attributierte Grammatik. Verwenden Sie dazu folgende Deklarationen:

```
TYPE
  NodePtr = ^Node;
  Node = RECORD
    firstChild, sibling: NodePtr;
    val: STRING; (* nonterminal, operator or operand as string *)
  END; (* Node *)
  TreePtr = NodePtr;
```

(c) Für Testzwecke wollen Sie nun eine graphische Ausgabe der Syntaxbäume erzeugen, indem Sie *Graphviz* einsetzen (https://graphviz.org). *Graphviz* ist ein Werkzeug, das aus einer textuellen Beschreibung der Knoten und Kanten eines Graphen (oder eines Baumes) eine grafische Darstellung erzeugt. Beispielsweise erhält man mit der folgenden Beschreibung die Baumdarstellung für den Ausdruck 1 + 2.

```
digraph G {
  n1 [label="Expr"];
  n2 [label="Term"];
  n3 [label="Fact"];
  n4 [label="1"];
  n5 [label="+"];
  n6 [label="Term"];
  n7 [label="Fact"];
  n8 [label="2"];
  n1 -> n2;
  n2 -> n3;
  n3 -> n4;
  n1 -> n5;
  n1 -> n6;
  n6 -> n7;
  n7 -> n8;
}
```

Erweitern Sie den Datentyp `Node` um eine eindeutige ID für Knoten und implementieren Sie Prozeduren, um aus einem Syntaxbaum in kanonischer Form eine passende Graphviz-Beschreibung zu erzeugen. Testen Sie Ihre Implementierung mit verschiedenen arithmetischen Ausdrücken und geben Sie mit Ihrer Lösung die generierten Beschreibungen und die von Graphviz erzeugten Bilder ab.

Das Werkzeug *Graphviz* können Sie lokal installieren (https://graphviz.org/) oder online verwenden, z.B. hier: https://dreampuf.github.io/.

### Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
2. Dokumentieren und kommentieren Sie Ihre Algorithmen.
3. Bei Programmen: Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Programm funktioniert, und dass es auch in Fehlersituation entsprechend reagiert.