Aufgabe 1 Legislative-Algebra **25 Punkte**

In dieser Aufgabe sollen Sie eine Algebra für eine (an das deutsche System angelehnte und stark vereinfachte) Legislative entwickeln. Dabei soll ein Kosmos aus einer Menge von mindestens 598 Parlamentsabgeordneten, einer amtierenden Regierung und einer Menge von Gesetzen verwaltet werden.

Jede(r) der Abgeordneten hat einen Namen und gehört einer Partei an. Dabei darf jeder Name nur höchstens einmal im Parlament vorkommen.

Weiterhin beinhaltet die Legislative eine Regierung, die sich aus einem Kanzler/einer Kanzlerin, einer Menge von Ministerinnen/Ministern (mindestens acht, höchstens 16) und einer nichtleeren Menge von Parteien, die die Regierung bilden, zusammensetzt. Dabei muss der Kanzler/die Kanzlerin im Parlament vertreten sein und einer der Regierungsparteien angehören. Ebenso handelt es sich bei jedem Minister/jeder Ministerin ebenfalls um einen/eine Abgeordnete des Parlaments und Mitglied einer der Regierungsparteien. Die Regierungsparteien müssen im Parlament zusammen über eine Mehrheit von mindestens einem Abgeordneten verfügen.

Jedes der vorhandenen Gesetze ist einem bestimmten Gesetzbuch (identifiziert durch seinen Namen) zugeordnet, besitzt darin eine eindeutige positive Nummer sowie einen Gesetzestext. Es ist möglich, Gesetze hinzuzufügen, zu verändern oder zu entfernen. Bei diesen Verfahren gelten stets folgende Abstimmungsregeln: Jeder/Jede Abgeordnete hat genau eine Stimme und kann mit Ja oder Nein stimmen oder sich enthalten. Ebenso besteht die Möglichkeit, dass der/die Abgeordnete bei der Abstimmung nicht anwesend ist. Eine Abstimmung ist erfolgreich, falls mindestens 50% der Abgeordneten abgestimmt haben und es mehr Ja- als Nein-Stimmen gibt. Jedem der gesetzrelevanten Operatoren muss (u.a.) die Zahl der Ja- und Nein-Stimmen und der Enthaltungen übergeben werden. Erzielt das jeweilige Vorhaben nicht die erforderliche Mehrheit, so bleibt die Gesetzeslage unverändert.

Ihre Algebra soll folgende Operationen bereitstellen:

erzLeg: Erzeugt eine Legislative aus einem Parlament, einer Regierung und einer Menge von Gesetzen.

nGes (neues Gesetz): Die Abgeordneten des Parlaments stimmen über ein neues Gesetz ab, das in ein bestimmtes existierendes Gesetzbuch eingetragen werden soll. Im Erfolgsfall wird es dort aufgenommen und erhält die niedrigste verfügbare Nummer.

äGes (Gesetzesänderung): Die Parlamentsabgeordneten stimmen über die Ersetzung eines bestehenden Gesetzes durch ein neues ab. Dafür müssen das Gesetzbuch und die Nummer des bisherigen Gesetzes sowie der neue Gesetzestext übergeben werden.

aGes (Gesetzesaufhebung): Die Abgeordneten des Parlaments stimmen über die Aufhebung eines bestehenden Gesetzes ab, das durch sein Gesetzbuch und seine Nummer bestimmt wird.

pü (Parteiübertritt): Ein bestimmter Abgeordneter/Eine bestimmte Abgeordnete des Parlaments wechselt zu einer anderen Partei. Sollte die bestehende Regierung daraufhin über keine Mehrheit mehr verfügen, wird die Legislative daraufhin auf einen undefinierten Wert gesetzt. Falls der/die angegebene Abgeordnete gar nicht im Parlament vertreten ist, bleibt alles beim Alten.

mv (Misstrauensvotum): Stimmen mindestens zwei Drittel der Abgeordneten für eine Abwahl des Kanzlers/der Kanzlerin, wird die Regierung auf einen undefinierten Wert gesetzt. Dagegen bleibt die Legislative unverändert, falls die Abstimmung keinen Erfolg hat.

8 Punkte (a) Geben Sie die Sorten der Algebra inklusive geeigneter Trägermengen an.

5 Punkte (b) Geben Sie die Signaturen aller genannten Operationen an.

12 Punkte (c) Geben Sie für jede Operation eine geeignete Funktion an.

Hinweise:

Setzen Sie *bool* ($\{true, false\}$), *int* (die Menge der ganzen Zahlen), *real* (die Menge der reellen Zahlen) und *string* (die Menge der Zeichenketten) als bekannt voraus. Bezeichnen Sie einen undefinierten Wert mit dem Symbol $\perp$. Sie können selbst entscheiden, ob Sie den undefinierten Wert mit in eine Sorte aufnehmen oder nicht.

Der Quantor $\exists$ („es existiert mindestens ein“) sollte Ihnen bereits bekannt sein. Sie dürfen außerdem die Quantoren $\exists!$ („es existiert genau ein“) sowie $\nexists$ („es existiert kein“) verwenden.

Beachten Sie die korrekte Verwendung von runden Klammerpaaren (Tupel), spitzen Klammerpaaren (Folgen/Sequenzen), geschweiften Klammerpaaren (Mengen). Beachten Sie bei der Verwendung von Tupeln, dass etwa $f(a, b)$ eine andere Funktion darstellt als $f((a, b))$! Während das erste Beispiel eine binäre Funktion (d.h. eine Funktion mit zwei Parametern, nämlich a und b) darstellt, beschreibt das zweite Beispiel eine unäre Funktion mit nur einem Parameter, nämlich dem Tupel (a, b).

Denken Sie daran, dass die Trägermengen ausschließlich die gewünschten Objekte beinhalten und durch die Operationen keine unerwünschten Objekte erzeugt werden dürfen!

Es ist mitunter hilfreich, neben den (gemäß Aufgabenstellung) offensichtlich geforderten weitere Sorten und/oder Operationen und Funktionen zu verwenden, beispielsweise bei der Definition von Trägermengen und in den Funktionsdefinitionen.

Sie können zur Vereinfachung der Schreibweise den folgenden generischen Infix-Operator „.“ verwenden, um auf die i-te Komponente eines n-Tupels t zuzugreifen:

$$. : (T_1 \times T_2 \times ... \times T_n) \times \{1, 2, ..., n\} \to T_i$$
$$(e_1, ..., e_n).i = e_i, 1 \leq i \leq n$$

Wenn in einer Formel die Tupelkomponenten benannt werden, können Sie anstelle des Komponentenindizes i den (Variablen-)Namen der Komponente verwenden. Somit verwendet man nach der Variablenbindung $t_1 = (name, alter)$ etwa $t_1.name$ anstelle von $t_1.1$. Dies erhöht die Lesbarkeit der Formeln. Sie kennen diese Notation natürlich bereits aus dem Zugriff auf Attribute und Methoden in Java-Objekten.

Aufgabe 2 Linienzug-Algebra **20 Punkte**

In dieser Aufgabe sollen Sie Linienzüge in der euklidischen Ebene durch eine Algebra formalisieren. Ein Linienzug besteht aus einer Folge von Segmenten, wobei stets der Endpunkt eines Segments dem Startpunkt seines Nachfolgers entspricht. Ein einzelnes Segment wird wiederum durch seine beiden Endpunkte dargestellt, die jeweils zwei reelle Koordinaten besitzen.

Ihre Algebra soll folgende Operationen bereitstellen:

makePoint: Erzeugt einen Punkt aus zwei reellen Koordinaten.

makeSegment: Erzeugt ein Segment aus zwei Punkten.

createLine: Erzeugt einen leeren Linienzug.

append: Hängt ein Segment ans Ende eines Linienzugs an. Wenn der Startpunkt des neuen Segments nicht dem Endpunkt des letzten Segments des Linienzugs entspricht, so muss ein weiteres Segment eingefügt werden, das diese beiden Punkte verbindet – es sei denn, der Endpunkt des neuen Segments ist gleich dem Endpunkt des letzten Segments des Linienzugs, dann werden Start- und Endpunkt des neuen Segments vertauscht. Schließlich wird der erweiterte Linienzug zurückgegeben.

prepend: Analog zu *append*, außer dass das neue Segment am Anfang des übergebenen Linienzugs eingefügt wird.

update: Hier wird ein Punkt innerhalb eines Linienzugs verschoben. Dafür müssen neben dem Linienzug die Nummer des Segments sowie ein boolscher Wert, der be-

schreibt, ob der Start- oder der Endpunkt des Segments verschoben werden soll, und der neue Punkt übergeben werden. Beachten Sie mögliche Auswirkungen auf benachbarte Segmente. Falls die angegebene Segmentnummer kleiner als 1 oder größer als die Anzahl der Segmente im Linienzug ist, so findet keine Änderung statt.

getLength: Berechnet die Länge eines Segments bzw. eines Linienzugs.

getStartEndDistance: Gibt den Abstand zwischen dem Startpunkt des ersten und dem Endpunkt des letzten Segments eines Linienzugs zurück. Im Fall eines leeren Linienzugs ist das Ergebnis 0.

getAngle: Bestimmt den Winkel zwischen zwei Segmenten, falls der Endpunkt des ersten dem Startpunkt des zweiten entspricht. Ist dies nicht der Fall oder hat mindestens eins der Segmente die Länge 0, so ist der Winkel undefiniert.

5 Punkte (a) Geben Sie die Sorten der Algebra inklusive geeigneter Trägermengen an.

5 Punkte (b) Geben Sie die Signaturen aller genannten Operationen an.

10 Punkte (c) Geben Sie für jede Operation eine geeignete Funktion an.

Hinweise:

Setzen Sie *int*, die Menge der ganzen und *real*, die Menge der reellen Zahlen, als bekannt voraus. Bezeichnen Sie einen undefinierten Wert mit dem Symbol $\perp$.

Beachten Sie die korrekte Verwendung von runden Klammerpaaren (Tupel), spitzen Klammerpaaren (Folgen/Sequenzen), geschweiften Klammerpaaren (Mengen). Beachten Sie bei der Verwendung von Tupeln, dass etwa $f(a, b)$ eine andere Funktion darstellt als $f((a, b))$! Während das erste Beispiel eine binäre Funktion (d.h. eine Funktion mit zwei Parametern, nämlich a und b) darstellt, beschreibt das zweite Beispiel eine unäre Funktion mit nur einem Parameter, nämlich dem Tupel (a, b).

Denken Sie daran, dass die Trägermengen ausschließlich die gewünschten Objekte beinhalten und durch die Operationen keine unerwünschten Objekte erzeugt werden dürfen!

Es ist mitunter hilfreich, neben den (gemäß Aufgabenstellung) offensichtlich geforderten weitere Sorten und/oder Operationen und Funktionen zu verwenden, beispielsweise bei der Definition von Trägermengen und in den Funktionsdefinitionen.

Sie können zur Vereinfachung der Schreibweise den folgenden generischen Infix-Operator „.“ verwenden, um auf die i-te Komponente eines n-Tupels t zuzugreifen:

$$. : (T_1 \times T_2 \times ... \times T_n) \times \{1, 2, ..., n\} \to T_i$$
$$(e_1, ..., e_n).i = e_i, 1 \leq i \leq n$$

Wenn in einer Formel die Tupelkomponenten benannt werden, können Sie anstelle des Komponentenindizes i den (Variablen-)Namen der Komponente verwenden. Somit verwendet man nach der Variablenbindung $t_1 = (name, alter)$ etwa $t_1.name$ anstelle von $t_1.1$. Dies erhöht die Lesbarkeit der Formeln. Sie kennen diese Notation natürlich bereits aus dem Zugriff auf Attribute und Methoden in Java-Objekten.

Aufgabe 3 Algorithmen auf allgemeinen Bäumen **25 Punkte**

Im Kurstext werden Implementierungsmöglichkeiten für allgemeine Bäume vorgestellt. Ein allgemeiner Baum sei hier nun mittels binärer Bäume dargestellt. Geben Sie rekursive Algorithmen für folgende Aufgaben an:

(a) Ausgabe aller Schlüssel des Baumes. *5 Punkte*

(b) Bestimmung des maximalen Grades aller Knoten des Baumes. *10 Punkte*

(c) Konvertierung des Baumes in eine Implementierung mit Arrays; gehen Sie dabei davon aus, dass ein Befehl $newnode_{arr}(d)$ zur Verfügung steht, der einen neuen Knoten (für die Array-Implementierung) mit einem Array der Größe d erzeugt. Dabei soll d für alle Knoten der maximale Grad aller Knoten sein. *10 Punkte*

Aufgabe 4 Postfix-Ausdrücke **30 Punkte**

Neben der *Infix*-Notation für mathematische Ausdrücke (der Operator steht zwischen seinen Operanden), können diese auch in der *Postfix*-Notation (bei der der Operator hinter seinen Operanden steht) angegeben werden. Während die erste Schreibweise stark verbreitet ist, ist die *Postfix*-Notation für die Verarbeitung in einem Programm einfacher. So kann bei dieser Schreibweise auf Klammerung und Operatorprioritäten verzichtet werden. Einige Beispiele sind:

algebraische Notation	*Postfix*-Notation
$3 + 5 * 2$	$3\ 5\ 2 * +$
$7 * (3 + 5) * 29$	$7\ 3\ 5 + *\ 29 *$

Die Auswertung solcher Ausdrücke erfolgt mit Hilfe eines Stacks. Der Ausdruck wird schrittweise durchlaufen. Ist das aktuelle Symbol ein Operand, so wird dieser auf den

Stack gelegt. Handelt es sich hingegen um einen Operator, werden soviele Operanden vom Stack genommen, wie der Operator Argumente benötigt, der Teilausdruck ausgewertet und das Ergebnis wieder auf den Stack gelegt.

In dieser Aufgabe soll eine Java-Klasse *Postfix*, die Ausdrücke in *Postfix*-Notation auswertet, implementiert werden. Dazu soll diese Klasse eine Methode

```
double eval(String expr);
```

anbieten. Die Methode soll den Wert des gegebenen Ausdrucks zurückliefern. Falls es sich nicht um einen gültigen Ausdruck handelt, soll das Ergebnis **null** sein. Die einzelnen Symbole des Arguments seien durch Leerzeichen voneinander getrennt. Die Klasse *Postfix* soll nicht auf eine feste Menge von Operatoren beschränkt sein. Vielmehr ist ein Konstruktor der Form:

```
Postfix(Operator[] operators);
```

bereitzustellen, wobei das Argument-Array alle erlaubten Operatoren enthält. Überlegen Sie, welche Methoden die Klassee *Operator* bereitstellen muss, damit die *eval*-Methode implementiert werden kann.

Statt Operanden gesondert zu behandeln, sollen sie wie 0-wertige Operatoren benutzt werden. So wird beispielsweise beim Symbol 8.35 kein Symbol vom Stack genommen und der Wert 8.35 auf den Stack gelegt.

15 Punkte (a) Implementieren Sie die Klasse *Postfix* sowie die Klasse (das Interface) *Operator*.

15 Punkte (b) Schreiben Sie *Operator*-Klassen für konstante Zahlen sowie für die Operationen $*$, /, + und −. Geben Sie eine *main*-Methode an, um *Postfix*-Ausdrücke, die als Argument gegeben sind, auswerten zu lassen.

Hinweise: Mit Hilfe der Klasse *StringTokenizer* kann der Argument-String in seine einzelnen Symbole zerlegt werden. Für den Stack können Sie die Java-eigene Klasse *Stack* verwenden, die im Paket *java.util* zu finden ist. Verwenden Sie jedoch höchstens die im Kurs vorgestellten Stack-Operationen sowie die *size*() Methode. Sie können folgendes Codefragment verwenden, um einen String in eine Zahl umzuwandeln:

```
try {
   double value = Double.parseDouble(string);
   // verarbeite value weiter
} catch (Exception e) {
   // Fehlerbehandlung
}
```