Aufgabe 1 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. d soll hierbei für alle Knoten der maximale Grad aller Knoten sein. *10 Punkte*

Aufgabe 2 Datentyp für Mengen **33 Punkte**

Ein Datentyp für Mengen soll folgende Operationen unterstützen:

contains	testet, ob ein Element x in der Struktur vorhanden ist.
insert	fügt ein Element x in die Datenstruktur ein. Jedes Element kann *höchstens einmal* vorhanden sein.
remove	entfernt *irgendein* beliebiges vorhandenes Element aus der Struktur und liefert es zurück. Man spezifiziert also beim Aufruf *nicht*, welches Element entfernt werden soll.

Die Elemente der Menge können durch Zahlen von 1 bis n identifiziert werden. Außerdem ist n klein genug, um Speicher der Größe $O(n)$ für die Struktur allokieren zu können.

(a) Entwerfen Sie eine Implementierung für diesen Datentyp, so daß die Laufzeitforderung in Aufgabenteil (b) erfüllt werden kann. *15 Punkte*

(b) Geben Sie Algorithmen für *contains, insert* und *remove* an, die je eine Laufzeit von $O(1)$ haben. *18 Punkte*

Aufgabe 3 Auswahl von Datenstrukturen **22 Punkte**

In dieser Aufgabe wird zunächst ein Anwendungsfall beschrieben. Ihre Aufgabe besteht darin, die geeigneten Datentypen für die Implementierung der Anwendung auszuwählen.

Eine wichtige Teilaufgabe im Übersetzerbau ist die Syntaxanalyse der Eingabesymbolfolge. Eine Variante dieser Syntaxanalyse ist die sogenannte Top-Down-Analyse. Hierbei wird ausgehend von einem gegebenen Satz von Regeln, der bestimmte Bedingungen erfüllen muss, die hier aber nicht weiter interessieren sollen, überprüft, ob eine gegebene Eingabe diese Regeln erfüllt.

Ein möglicher Regelsatz, der Teile einer Programmiersprache beschreibt, ist:

(1)	*stmt*	→	*assignment*
(2)		|	*cond*
(3)		|	*loop*
(4)	*assignment*	→	**id** := *value*
(5)	*cond*	→	**if** *boolexpr* **then** *stmt restcond*
(6)	*restcond*	→	**fi**
(7)		|	**else** *stmt* **fi**
(8)	*loop*	→	**while** *boolexpr* **do** *stmt* **do**
(9)	*boolexpr*	→	*value* **cop** *value*
(10)	*value*	→	**id**
(11)		|	**const**

Die kursiv gedruckten Symbole werden in der Analyse Nichtterminale genannt und die fettgedruckten Symbole Terminale. Für jedes Nichtterminal gibt es ein oder mehrere durch „| “von einander abgetrennte Ersetzungsregeln. So kann z.B. *stmt* durch *assignment, cond* oder *loop* ersetzt werden. Um zu entscheiden, welche Regel angewendet werden muss, wird das aktuelle Symbol der Eingabefolge mitbetrachtet. Es gibt hierzu eine Übersicht, die für alle möglichen Kombinationen aus Nichtterminalen und Terminalsymbolen festlegt, wie weiter zu verfahren ist. Für die Kombination von *stmt* mit **if** folgt dann z.B. eine Ersetzung von *stmt* durch *cond*. Für die aktuelle Kombination von *cond* und **if** wird *cond* rekursiv durch **if** *boolexpr* **then** *stmt restcond* ersetzt. Stände statt **if** ein anderes Symbol in der Eingabefolge, würde die Verarbeitung nach Blick in die Übersicht mit einer Fehlermeldung abgebrochen, da im Regelsatz *cond* nur durch eine mit **if** beginnende Folge ersetzt werden kann. Steht nun auch in der aktuellen Regel ein Terminalsymbol, wird das Terminalsymbol des aktuellen Zustands mit dem aktuellen Terminalsymbol der Eingabefolge verglichen. Sind diese identisch, werden beide als korrekt bearbeitet gelöscht und die Bearbeitung wird mit den nächsten Symbolen im aktuellen Zustand und in der Eingabefolge fortgesetzt. Stimmen die Terminalsymbole nicht überein, wird die Bearbeitung mit einer Fehlermeldung abgebrochen.

Betrachten wir als Beispiel die Analyse der folgenden **if**-Anweisung:

if $3 < 5$ **then** $a = 3$ **else** $a = 5$ **fi**

Zunächst werden für den Syntaxcheck Konstanten und Bezeichner durch entsprechende Terminalsymbole ersetzt, da der eigentliche Wert der Variablen für die Syntaxanalyse keine Rolle spielt. Als Eingabefolge erhalten wir deshalb:

if const cop const then id = const else id = const fi $

Hierbei zeigt das **$**-Symbol das Ende der Eingabesymbolfolge an.

Zu Beginn der Analyse betrachtet die Anwendung das Startsymbol *stmt* und das erste Symbol der Eingabefolge **if**. *stmt* ist ein Nichtterminal, das auf verschiedene Arten ersetzt werden kann. In der Übersicht befindet sich für die Kombination (*stmt*, **if**) in Regel (2) die Ersetzung von *stmt* durch *cond*. Wir ersetzen also *stmt* durch *cond* und geben die Nummer der angewendeten Regel aus. Anschließend prüfen wir die Kombination (*cond*, **if**) in der Übersicht. Nach Regel (5) wird *cond* durch **if** *boolexpr* **then** *stmt restcond* ersetzt. Wir handeln entsprechend. Als nächstes vergleichen wir das **if** aus der Regel mit dem Symbol **if** vom Eingabeband. Diese sind identisch, wir entfernen also das **if** vom Eingabeband und aus dem aktuellen Zustand und fahren mit der nächsten Symbolkombination (*boolexpr*, **id**) in der Analyse fort. Gemäß Regel (9) erfolgt die Ersetzung von *boolexpr* durch *value* **cop** *value*. Als nächstes wird das erste *value* durch **id** ersetzt und in einem weiteren Schritt **id** mit **id** verglichen und wieder in der Eingabe und in der Regel je ein Symbol weitergerückt. So wird die Verarbeitung fortgesetzt, bis entweder ein Fehler auftritt, der zum Abbruch der Bearbeitung führt oder die Prüfung beim Erreichen des **$**-Symbols in der Eingabefolge und leerer Zustandsfolge erfolgreich beendet wird.

Hinweis: Im Kurs wird das Array nicht explizit als Datentyp im engeren Sinne eingeführt, sondern nur als programmiersprachliches Werkzeug fester Größe vorgestellt. Bei der Lösung dieser Aufgabe dürfen Sie jedoch Arrays als Datentyp verwenden.

(a) Welche Datenstrukturen des Kurstextes bieten sich für die interne Darstellung der Eingabefolge, der Übersicht, der Regeln und des aktuellen Zustands in der Regelabarbeitung an? Begründen Sie jeweils Ihre Entscheidung. *11 Punkte*

(b) Mit welchen von den Datenstrukturen zur Verfügung gestellten Operationen können die Anforderungen der Anwendung realisiert werden? *11 Punkte*

Aufgabe 4 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 beschreibt, 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.

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

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:

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

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 Punktnotation natürlich bereits aus dem Zugriff auf Attribute und Methoden in Java-Objekten.