Übungen zu Vorlesung
Praktische Informatik I
Wintersemester 2000/01

Serie 7

Listen II

Aufgabe 1: Eine andere Sicht auf Listen (100%)

In dem File `ImpList.java` auf der PI-1 Homepage finden Sie eine Implementierung der Basisoperationen für Listen im imperativen Stil. Sie unterscheidet sich von der in der Vorlesung und im Balzert zu findenden Realisierung im wesentlichen dadurch, daß die Speicherung der Listen nicht innerhalb eines Listenobjekts vorgenommen wird, sondern dem aufrufenden Hauptprogramm überlassen wird, so daß allen Funktionen die zu manipulierende Liste als ein Parameter mit übergeben werden muß. Alle Basisoperationen sind nicht–destruktiv implementiert, d. h. Listen, die als Parameterwerte dieser Funktionen verwendet werden, bleiben mit unverändertem Inhalt erhalten.
Damit besitzen die Basisfunktionen die Signaturen, so wie sie in der abstrakten (algebraischen) Sicht auf Listen üblich sind:

emptyList: $\rightarrow$ MyList [Konstruktor]
erzeugt eine leere Liste und gibt diese zurück

append: int $\times$ MyList $\rightarrow$ MyList [Konstruktor]
fügt ein Element vorne an eine gegebene Liste an und gibt die veränderte Liste zurück

head: MyList $\rightarrow$ int [Selektor]
gibt das erste Element einer Liste zurück

tail: MyList $\rightarrow$ MyList [Selektor]
gibt die Restliste, d. h. alles bis auf das erste Element einer Liste zurück

isEmpty: MyList $\rightarrow$ boolean [Recognizer]
gibt zurück, ob es sich bei der Liste um eine leere Liste handelt

Hiermit steht ein minimaler aber vollständiger Satz von Funktionen zur Verfügung, um alle komplexeren Funktionen implementieren zu können, ohne die tatsächliche Umsetzung der Listen im Speicher zu betrachten. Die Realisierungen der höheren Funktionen kommt dann mit den Basisfunktionen, `if-then-else`–Konstrukten, Rekursion, grundlegender Zahlenarithmetik, logischen Ausdrücken sowie der Verwendung von anderen höheren Funktionen mit den gleichen Voraussetzungen aus. *Insbesondere sind keine Variablen zur Speicherung von Zwischenergebnissen und keine Schleifenkonstrukte notwendig!*
Die folgenden drei Funktionen sollen hier als Beispiel für eine derartige Umsetzung dienen. Es wird eine konzeptionelle Umsetzung angegebenen; nicht–destruktive Implementierungen dieser Konzepte finden sich ebenfalls in dem Sourcecode `ImpList.java`.

length: MyList $\rightarrow$ int
gibt die Anzahl der Elemente einer Liste zurück

appendright: int × MyList → MyList
: fügt ein Element hinten an eine gegebene Liste an und gibt die veränderte Liste zurück

reverse: MyList → MyList
: dreht die Reihenfolge der Elemente einer Liste um und gibt die umgedrehte Liste zurück

Im folgenden repräsentiert x eine Variablen vom Typ `int`, l ist vom Typ `MyList`. Damit ergeben sich die Umsetzungen:

$$\begin{aligned}
length(l) &= \begin{cases} 0 & \text{falls } isEmpty(l) \\ 1 + length(tail(l)) & \text{sonst} \end{cases} \\
appendright(x, l) &= \begin{cases} append(x, l) & \text{falls } isEmpty(l) \\ append(head(l), appendright(x, tail(l))) & \text{sonst} \end{cases} \\
reverse(l) &= \begin{cases} l & \text{falls } isEmpty(l) \\ appendright(head(l), reverse(tail(l))) & \text{sonst} \end{cases}
\end{aligned}$$

Entwickeln und implementieren Sie die folgenden Funktionen in diesem Sinne, d. h. insbesondere auch ohne Schleifen und ohne Hilfsvariablen für Zwischenergebnisse. Dokumentieren Sie dabei zusätzlich Ihre Entwurfsideen.

front: MyList → MyList
: liefert die Anfangsliste zurück, d. h. die Liste, die alle bis auf das letzte Element enthält.

areEqual: MyList × MyList → boolean
: liefert genau dann `true`, wenn die beiden Listen aus Elementen mit gleichen Werten in der gleichen Reihenfolge bestehen

concat: MyList × MyList → MyList
: hängt zwei Listen aneinander und gibt das Ergebnis zurück

isPalindrom: MyList → boolean
: liefert genau dann `true`, wenn eine Liste ein Palindrom ist, d. h. wenn vorwärts- und rückwärtslesen der Werte der Listenelemente die gleiche Zahlenfolge ergibt

Beachten Sie, daß `front`, genauso wie die Basisfunktion `tail`, für die leere Liste nicht definiert ist. Für die Umsetzung in Java bedeutet dies, daß im Falle des Aufrufs mit der leeren Liste eine Exception generiert werden soll oder aber eine entsprechende Exception einer aufgerufenen Basisfunktion weitergereicht werden muß.

Abgabe: 22. – 26.1.2001 nach den jeweiligen Praktika. Die Abgabe soll sowohl elektronisch (Programm-Quellcode) als auch in gedruckter Form (mit LaTeX gesetzter kommentierter und erläuterter Quellcode) erfolgen. Dabei ist jeweils auch auf geeignete Testfälle und deren Dokumentation zu achten!