Übungen zu Vorlesung
Praktische Informatik I
Wintersemester 2000/01

Serie 6

Rekursion und Listen

Aufgabe 1: Binäre Suche (rekursiv) (25%)

Implementieren Sie eine Java–Methode, welche als Parameter ein *sortiertes* Integer–Array beliebiger Länge sowie einen Integerwert bekommt und mit Hilfe der *rekursiven* Binären Suche feststellen soll, ob und falls ja, an welchem Index dieser Wert in dem Array steht. Dabei sollen für die verschiedenen Rekursionsschritte keine Kopien des Arrays angelegt werden! Die Suchmethode soll den entsprechenden Indexwert als Rückgabewert an das aufrufende Hauptprogramm zurückliefern aber den Wert *nicht* selber auf dem Bildschirm ausgeben. Das Hauptprogramm soll vom Benutzer die Eingabe einer sortierten Zahlenfolge sowie einen zu suchenden Wert verlangen und anschliessend das Ergebnis der Binären Suche auf dem Bildschirm ausgeben.

Aufgabe 2: Die Türme von Hanoi (25%)

Das Problem der Türme von Hanoi stellt eine Aufgabe dar, die sich sehr gut mit den Mitteln der Rekursion lösen läßt: Gegeben seien `n` Scheiben mit unterschiedlichem Durchmesser, die auf Platz A der Größe nach geordnet zu einem Turm aufgeschichtet sind, wobei die unterste Scheibe die größte ist. Die Scheiben sollen unter Verwendung des Hilfsplatzes C auf den Platz B transportiert werden. Dabei darf zu jeder Zeit nur *eine* Scheibe bewegt werden und diese darf *nur oben* von einem Turm abgenommen werden. Außerdem darf *niemals eine größere Scheibe auf einer kleineren liegen.*
Eine einfache rekursive Vorgehensweise zu diesem Problem besteht darin, es in kleinere Teilprobleme zu zerlegen:

1. Bewege alle bis auf die unterste Scheibe auf den Hilfsturm
2. Bewege die größte Scheibe auf den Zielturm
3. Bewege die restlichen Scheiben vom Hilfsturm auf den Zielturm

Implementieren Sie den hier angedeuteten Algorithmus in Java. Überlegen Sie sich hierzu eine geeignete Methodensignatur, um bei den rekursiven Aufrufen die notwendigen Parameter übergeben zu können. Alle tatsächlich notwendigen Bewegungen von Scheiben sollen auf dem Bildschirm in der folgenden Form ausgegeben werden: `<Scheibengröße>: <Von-Turm> -> <Nach-Turm>`, wobei die Scheibengrösse einfach mit jeder größeren Scheibe um Eins steigt und die anfangs oberste Scheibe per Definition immer die Größe 1 besitzt.

Aufgabe 3: Listenalgebra (50%)

Für die Datenstrukturen zur Realisierung einfach verketteter Listen, welche in der Vorlesung vorgestellt wurden und auch im Balzert ab S. 600 zu finden sind, sollen die grundlegenden Operationen der Listenalgebra implementiert werden. Dabei sollen hier jedoch nur Listen von Integer–Zahlen betrachtet werden. Schreiben Sie eine Java–Klasse, welche Methoden für die folgenden Basisoperationen auf Listen bereitstellt. Dabei ist ein **MyList** in den Übergabeparametern oder als Rückgabeparameter so zu verstehen, daß die interne Repräsentation der Liste geändert wird, so daß z. B. die Funktion **tail** in der Java–Implementierung weder Übergabeparameter noch einen Rückgabewert besitzt (`void tail() { ... }`).

emptyList: → MyList
: erzeugt eine leere Liste und gibt diese zurück

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

head: MyList → int
: gibt das erste Element einer Liste zurück

tail: MyList → MyList
: gibt den Rest, d. h. alles bis auf das erste Element einer Liste zurück

isEmpty: MyList → boolean
: gibt zurück, ob es sich bei der Liste um eine leere Liste handelt

Implementieren Sie zusätzlich die folgenden Operationen und setzen Sie dabei direkt auf den Java–Datenstrukturen auf:

length: MyList → int
: gibt die Anzahl der Elemente einer Liste zurück

insert: int × int × MyList → MyList
: fügt ein Element an einer gegebenen Position in die Liste ein

get: int × MyList → int
: gibt das an einer bestimmten Position liegende Element zurück

Insgesamt ist dabei zu beachten, daß nur die Listenmethoden, die Listen als Rückgabewerte produzieren, destruktiv auf der Liste arbeiten dürfen, d. h. die Liste kann direkt verändert werden, es muß keine Kopie angelegt werden. Bei allen Methoden ist außerdem darauf zu achten, daß Fehler beim Aufruf in geeigneter Weise erkannt und auf dem Bildschirm ausgegeben werden (z. B. bei **get** bei dem Zugriff auf ein nicht vorhandenes Element). Zur Definition und Dokumentation von Testfällen empfiehlt es sich, zusätzlich eine **print**–Methode zu implementieren, die alle Elemente einer Liste auf dem Bildschirm ausgibt!

Abgabe: 15. – 19.1.2001 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!