7. Übung (26.–30. November 2018)

Algorithmen und Datenstrukturen

Aufgabe 1 (AGS 3.2.34)

(a) Gegeben sei folgende Typdefinition für einfach verkettete Listen:

```
1 typedef struct element *list;
2 typedef struct element { int value; list next; } element;
```

Schreiben Sie eine **iterative** Funktion `f(list l)`, die genau dann `1` zurück gibt, wenn in der Liste alle jeweils benachbarten Elemente maximal um `1` voneinander abweichen. Ansonsten soll die Funktion `0` zurück geben.

(b) Gegeben sei die folgende Typdefinition für binäre Bäume:

```
1 typedef struct node *tree;
2 typedef struct node { int key; tree left, right; } node;
```

Ein *Blattknoten* ist ein Knoten, dessen linker und rechter Teilbaum jeweils leer ist. Schreiben Sie eine **rekursive** Funktion `defol(tree *p)`, welche alle Blattknoten des übergebenen Baumes entfernt.

Aufgabe 2 (AGS 6.1.10)

Wenden Sie den Quicksort-Algorithmus auf die Folge $4, 7, 6, 2, 9$ an. Die Zahlen sollen aufsteigend sortiert werden. Dokumentieren Sie

- Kennzeichnung das Pivotelements,
- Stellung der Indizes i und j unmittelbar vor dem Tausch und vor dem rekursiven Aufruf, und
- Teilfolgen nach den rekursiven Aufrufen.

Aufgabe 3 (AGS 6.2.12)

Wenden Sie den Heapsort-Algorithmus auf die Folge $2, 0, 9, 3, 5, 8, 4, 1, 6, 7$ an. In Phase 2 brauchen Sie nur zwei Sortierschritte auszuführen. Dokumentieren Sie:

- In Phase 1:
  - das Einordnen in einen binären Baum, und
  - das schrittweise Herstellen der Heap-Eigenschaft; hier insbesondere die Veränderungen durch die Funktion *sinkenlassen*.
- In Phase 2: die notwendige Anzahl an Sortierschritten bestehend aus
  1. einem Austausch, und
  2. der Anwendung der Funktion *sinkenlassen*.

Bei der Funktion *sinkenlassen* brauchen Sie keine Einzelschritte dokumentieren.

Zusatzaufgabe 1 (AGS 3.2.21)

Gegeben seien die folgende Typdeklaration für einen Binärbaum:

```
1 typedef struct node *tree;
2 struct node { int key, summe; tree left, right; };
```

sowie die Typdeklaration für eine Liste:

```
1 typedef struct element *list;
2 struct element { int value; list next; };
```

(a) Schreiben Sie in C eine Funktion `summen`, die an jedem Knoten eines beliebigen Binärbaumes des obigen Typs in das Knotenmerkmal `summe` die Summe aller geradzahligen Schlüsselwerte des Teilbaumes, der diesen Knoten als Wurzelknoten hat, einträgt.
Hinweis: Falls Sie dafür Hilfsfunktionen benötigen, müssen Sie diese vollständig angeben!

(b) Schreiben Sie in C eine Funktion `void tree_to_list(list *l, tree t)`, die eine **fallend** geordnete Liste der Schlüsselwerte, die in dem binären **Suchbaum** (des angegebenen Typs) auftreten, erzeugt.
Geben Sie den Aufruf Ihrer Funktion an!
Hinweis: Sie können eine Funktion `void append(list *l, int n)` vorraussetzen, die ein Element mit dem Wert `n` hinten an die liste `l` hängt.

Zusatzaufgabe 2 (AGS 6.2.13 ⋆)

Wenden Sie den Heapsort-Algorithmus auf die Folge $7, 0, 8, 1, 6, 5, 2, 3, 4, 9$ an. In Phase 2 brauchen Sie nur zwei Sortierschritte auszuführen.

Zusatzaufgabe 3 (AGS 6.1.1 ⋆)

Wenden Sie den Quicksort-Algorithmus auf die Folge $9, 5, 4, 2, 3, 8, 1$ an.