8. Übung (9.–13. Dezember 2019)

Algorithmen und Datenstrukturen

Aufgabe 1 (AGS 3.2.46 b–c, Klausuraufgabe vom WS 2018/19)

(a) Gegeben sei der folgende Datentyp zur Darstellung von Listen mit Werten vom Typ **int**:

```
struct element { int key; struct element *next; };
```

Geben Sie eine Funktion `void append(struct element **lp, int n)` an, welche ein neues Element e mit Schlüsselwert `n` erzeugt, dieses hinten an eine gegebene Liste `*lp` anhängt, und den Nachfolger von e auf `NULL` setzt. Gehen Sie davon aus, dass `lp != NULL` ist.

(b) Gegeben sei der folgende Datentyp zur Darstellung von Binärbäumen mit Schlüsselwerten vom Typ **int**:

```
struct node { int key; struct node *left, *right; };
```

Schreiben Sie eine Funktion `treeToList`, welche aus einem beliebigen Binärbaum t des oben genannten Typs eine Liste l der geraden Schlüsselwerte (`key`) von t generiert. Dabei soll l die Schlüsselwerte in folgender Reihenfolge enthalten:

- zunächst die geraden Schlüsselwerte im linken Teilbaum von t,
- dann gegebenenfalls den Schlüsselwert am Wurzelknoten, und
- zuletzt die geraden Schlüsselwerte im rechten Teilbaum von t.

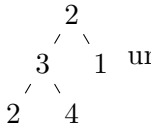
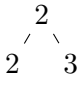
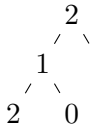
Geben Sie alle dazu erforderlichen Variablendeklarationen und einen **Aufruf** Ihrer Funktion an. *Hinweis*: Nutzen Sie die Funktion `append` aus Aufgabe 1 (a).

Aufgabe 2 (AGS 3.2.36 a)

Gegeben sei die folgende Typdefinition für binäre Bäume:

```
struct node { int key; struct node *left, *right; };
```

Schreiben Sie eine Funktion `struct node *baz(struct node *s, struct node *t)`, so dass der Baum `baz(s, t)` aus `s` hervorgeht, indem jede Knotenbeschriftung n in `s` durch die *Anzahl* der Knoten mit Beschriftung n in `t` ersetzt wird. Die Bäume `s` und `t` sollen dabei nicht verändert werden!

Beispiel: Wenn `s` =  und `t` =  , dann ist `baz(s,t)` =  .

Aufgabe 3 (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 den Rechenablauf, indem Sie

- das Pivot-Element jeder Teilfolge kennzeichnen und
- die Teilfolgen und Stellung der Indizes i, j jeweils
  - unmittelbar vor und nach jedem Tausch von Elementen, sowie
  - unmittelbar vor und nach jedem rekursiven Aufruf angeben.

Aufgabe 4 (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 3.2.34)
(a) Gegeben sei folgende Typdefinition für einfach verkettete Listen:

```
struct element { int key; struct element *next; };
```

Schreiben Sie eine **iterative** Funktion `f(struct element *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:

```
struct node { int key; struct node *left, *right; };
```

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

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