Algorithmen und Datenstrukturen

7. Übungsblatt

Zeitraum: 30. November – 04. Dezember 2015

Übung 1

Gegeben sei der Typ

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

(a) Implementieren Sie eine Funktion `sum`, welche die Werte einer solchen Liste aufsummiert!

(b) Implementieren Sie eine Funktion `rmEvens`, welche aus einer Liste alle Elemente mit einer geraden Zahl entfernt.

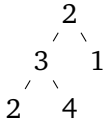
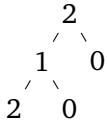
Geben Sie jeweils eine rekursive und eine iterative Lösung an.

Übung 2 (AGS 3.2.36)

Gegeben sei die folgende Typdefinition für binäre Bäume:

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

(a) Schreiben Sie eine Funktion `tree baz(tree s, tree 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!

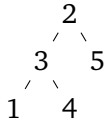
Beispiel: Wenn `s =`  und `t =`  , dann ist `baz(s,t) =`  .

(b) Implementieren Sie eine Funktion `int leafprod(tree t)`, welche für einen Eingabebaum `t` das Produkt der Knotenbeschriftungen der *Blätter* in `t` berechnet!

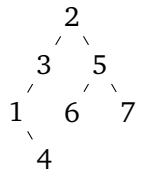
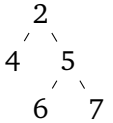
Übung 3 (AGS 3.2.37)

(a) Schreiben Sie eine Funktion `void f(int k, int n)`, welche alle nichtnegativen Vielfachen von `k` zwischen (einschließlich) `0` und `n` ausgibt (`k`, `n` ≥ 0). Diese sollen dabei *in absteigender Reihenfolge* ausgegeben werden. So soll z.B. `f(3, 7)` die Zahlenfolge `6 3 0` ausgeben.

(b) Schreiben Sie eine Funktion `int evenSum(tree t)`, welche in einem Baum `t` die Summe der Beschriftungen jener Knoten berechnet, deren Entfernung von der Wurzel geradzahlig ist.

Beispiel: Wenn `t =`  , dann ist `evenSum(t) = 2 + 1 + 4 = 7`.

(c) Implementieren Sie eine Funktion **`void`** `dechain(tree *t)`, welche aus einem Binärbaum `t` genau die Knoten löscht, welche exakt einen Nachfolger haben.

Ist z.B. `t =`  , so soll nach Aufruf von `dechain(t)` gelten, dass `t =`  .

Zusatzaufgabe 1 (AGS 3.2.35)
Gegeben sei die folgende Typdefinition für binäre Bäume:

```
typedef struct node *tree;
typedef struct node {
    tree left, right;
} node;
```

(a) Der *Rang* eines Baumes t vom Typ `tree` ist die Länge des Pfades von der Wurzel von t bis zum am weitesten rechts stehenden Blatt von t. Schreiben Sie eine Funktion `int rank(tree t)`, welche den Rang des übergebenen Baums `t` berechnet! Für `t == NULL` soll `rank(t) == 0` sein.

(b) Ein Baum t heißt *Linksbaum*, falls für jeden Knoten v von t der Rang (vgl. **(a)**) des linken Teilbaums von v größer oder gleich dem Rang des rechten Teilbaums von v ist. Schreiben Sie eine Funktion `int isLeftist(tree t)`, welche für einen Baum `t` als Eingabe den Wert 1 ausgibt, falls `t` ein Linksbaum ist, ansonsten den Wert 0.

(c) Die Summe aller natürlichen Zahlen kleiner oder gleich 16, die Vielfache von 3 oder von 5 sind, ist $3+5+6+9+10+12+15 = 60$. Implementieren Sie eine Funktion `int s(int n)`, welche für den Parameter `n` die Summe aller natürlichen Zahlen kleiner oder gleich `n` ausgibt, welche Vielfache von 3 oder von 5 sind! Tipp: `n` ist genau dann durch `m` teilbar, wenn `n % m == 0` gilt.

Zusatzaufgabe 2 (AGS 3.2.3)
Gegeben sei die folgende Deklaration für Elemente einer verketteten Liste:

```
typedef struct list_ele *list;
typedef struct list_ele {
  int key;
  list next;
} l_ele_type;
```

(a) Schreiben Sie in *C* eine Funktion `void delete_n(list *l, int n)`, die aus einer beliebigen Liste mit Elementen des oben aufgeführten Typs *alle* Elemente mit dem Schlüsselwert `n` entfernt.

(b) Geben Sie in *C* eine Funktion `int ordered(list l)` an, welche ermittelt, ob eine solche Liste aufsteigend geordnet ist oder nicht, und je nachdem `1` oder `0` als Funktionswert zurückgibt.

Beachten Sie: Schlüsselwerte dürfen mehrfach vorkommen; in diesem Fall müssen gleiche Schlüsselwerte nebeneinander stehen.