Algorithmen und Datenstrukturen

7. Übungsblatt

Zeitraum: 01. – 05. Dezember 2014

Übung 1 (AGS 4.21)
Gegeben sei folgendes C-Programm:

```
 1  #include <stdio.h>
 2
 3  void g(int d, int* e);
 4
 5  void f(int* a, int* b) {
 6    int c;
 7    c = *a;
 8    /* label1 */
 9    while (c < *b) {
10      c = c + 1;
11      /* label2 */
12      g(c, b); /* $1 */
13      *a = c;
14      /* label3 */
15    }
16  }
17
18  void g(int d, int* e) {
19    /* label4 */
20    if (d < *e) {
21      *e = *e - 1;
22      /* label5 */
23      f(&d, e); /* $2 */
24    }
25    /* label6 */
26  }
27
28  int main() {
29    int m, n;
30    m = 2;
31    n = 4;
32    /* label7 */
33    f(&m, &n); /* $3 */
34    /* label8 */
35    return 0;
36  }
```

(a) Bestimmen Sie den Gültigkeitsbereich jedes Objektes. Nutzen Sie dazu die Zeilennummern.
(b) Erstellen Sie ein Speicherbelegungsprotokoll für das angegebene Programm. Dokumentieren Sie die aktuelle Situation beim Passieren der Haltepunkte (*label1* bis *label8*). Geben Sie jeweils den Rücksprungmarkenkeller und die *sichtbaren* Variablen mit ihrer Wertebelegung an. Die Inhalte von Speicherzellen nicht-sichtbarer Variablen sollen Sie nur bei Änderungen eintragen. Beachten Sie: *$1* bis *$3* seien die bereits festgelegten Rücksprungmarken.

Übung 2
Gegeben seien die folgenden Typdefinitionen für verkettete Listen.

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

(a) Implementieren Sie eine Funktion **int** `sum(list l)`, welche die Summe der Werte der in der Liste `l` vorkommenden Elemente berechnet. Geben Sie sowohl eine iterative als auch eine rekursive Lösung an!
(b) Implementieren Sie eine Funktion `dub`, welche in jedem Element einer übergebenen Liste `l` den entsprechenden Wert verdoppelt. Geben Sie eine Lösung an, welche die Liste `l` verändert, sowie eine, welche die beschriebene Operation auf einer Kopie von `l` durchführt!

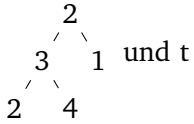
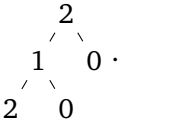
Übung 3
Wir setzen wieder die Typdefinitionen aus Aufgabe 2 voraus.

(a) Eine Liste heißt *geordnet,* falls die Werte ihrer Elemente aufsteigend sortiert sind, wenn man sie in der Reihenfolge vom Anfang zum Ende der Liste betrachtet.
Implementieren Sie eine Funktion `insert`, welche einen Parameter **int** `i` als Schlüsselwert in eine geordnete Liste `list l` einfügt. Dabei soll die Eigenschaft, dass die Liste geordnet ist, erhalten bleiben. Überlegen Sie sich dabei genau den Typ von `insert`!
(b) Schreiben Sie eine Funktion `removeEvens`, welche aus einer Liste `list l` all die Elemente mit geradzahligen Werten löscht. Auch hier ist der Typ von `removeEvens` wichtig!

Implementieren Sie beide Teilaufgaben jeweils iterativ und rekursiv.

Übung 4 (AGS 3.2.35 (a), (b))
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!

Zusatzaufgabe 1 (AGS 3.2.4 ⋆)
Gegeben sei die folgende Typdefinition für Elemente einer verketteten Liste:

```
typedef struct l_ele *LPtr;
typedef struct l_ele {
  int key;
  LPtr next;
} l_element;
```

(a) Schreiben Sie in C eine rekursive Funktion `anfuegen(...)`, die ein neues Listenelement mit dem Wert `k` hinten an eine Liste des oben aufgeführten Typs anfügt.
(b) Schreiben Sie in C eine Funktion `entfernen(...)`, die aus einer beliebigen Liste des oben aufgeführten Typs ein Element mit dem Wert `k` entfernt. Der Wert `k` kommt in der Liste höchstens einmal vor.

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

```
1 typedef struct list_ele *list;
2 typedef struct list_ele {
3   int key;
4   list next;
5 } 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 Wert `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 die Aussage (`0` für nicht geordnet und `1` für geordnet) als Funktionswert zurückgibt.
Beachten Sie: Werte dürfen mehrfach vorkommen; in diesem Fall müssen gleiche Werte nebeneinander stehen.