

Algorithmen und Datenstrukturen

06. Übungsblatt

Zeitraum: 21. – 25. November 2016

Übung 1 (AGS 4.25)

Gegeben sei folgendes C-Programm:

```

1  #include <stdio.h>
2
3  void g(int* y, int* u) {
4      /* label1 */
5      if (*u % 3 == 0) {
6          *u = *u / 3;
7          *y = *y + 1;
8          g(y, u); /* $1 */
9      }
10 }
11
12 void f(int* x, int v, int w) {
13     /* label2 */
14     if (v < 20) {
15         v = v * 2;
16         *x = *x + 1;
17         f(x, v, w); /* $2 */
18         /* label3 */
19     }
20     g(x, &w); /* $3 */
21     /* label4 */
22 }
23
24 int main() {
25     int a = 0;
26     /* label5 */
27     f(&a, 12, 15); /* $4 */
28     /* label6 */
29     printf("%d", a);
30     return 0;
31 }
    
```

- (a) Tragen Sie den Gültigkeitsbereich jedes Objektes in eine Tabelle ein.
 (b) Setzen Sie das folgende Speicherbelegungsprotokoll fort.

Label	Rücksprungmarken	1	2	3	4	5	6	7	8	9	10	11
<i>label5</i>	-	a										
		0										

Übung 2

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. Dabei soll die bestehende Liste verändert werden.

Geben Sie jeweils eine rekursive und eine iterative Lösung an.

Übung 3 (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 $s =$

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 3 \quad 1 \\ / \quad \backslash \\ 2 \quad 4 \end{array}$$
und $t =$

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 2 \quad 3 \end{array}$$
,
dann ist `baz(s, t) =`

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 1 \quad 0 \\ / \quad \backslash \\ 2 \quad 0 \end{array}$$
.

- (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 3.2.37)

- (a) Schreiben Sie eine Funktion `void f(int k, int n)`, welche alle nichtnegativen Vielfachen von k zwischen (einschließlich) 0 und n ausgibt ($k, n \geq 0$). 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 $t =$

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 3 \quad 5 \\ / \quad \backslash \\ 1 \quad 4 \end{array}$$
,
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. $t =$

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 3 \quad 5 \\ / \quad \backslash \\ 1 \quad 6 \\ \quad \backslash \\ \quad 4 \end{array}$$
,
so soll nach Aufruf von `dechain(t)` gelten, dass $t =$

$$\begin{array}{c} 2 \\ / \quad \backslash \\ 4 \quad 5 \\ \quad \backslash \\ \quad 6 \quad 7 \end{array}$$
.

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.