# Algorithmen und Datenstrukturen

### Aufgabe 1 (AGS 4.30, Klausuraufgabe vom WiSe 2018/19)

```
1  #include <stdio.h>
2
3  int z = 0;
4  void g(int*, int);
5
6  void f(int* a, int b) {
7    /* label1 */
8    if (b == 0)
9      *a = 1;
10   else {
11     f(&z, b - 1);   /* $1 */
12     /* label2 */
13     g(&z, z);       /* $2 */
14     /* label3 */
15     *a = b - z;
16   }
17 }
18
19 void g(int* c, int d) {
20   /* label4 */
21   if (d == 0)
22     *c = 0;
23   else {
24     g(&z, d - 1);   /* $3 */
25     /* label5 */
26     f(&z, z);       /* $4 */
27     /* label6 */
28     *c = d - z;
29   }
30 }
31
32 int main() {
33   int x, y;
34   x = 2;
35   /* label7 */
36   f(&y, x);         /* $5 */
37   /* label8 */
38   printf("%d\n", y);
39   return 0;
40 }
```

(a) Geben Sie den Gültigkeitsbereich jedes Objektes des Programms an. Nutzen Sie dazu die Zeilennummern.

(b) Führen Sie jedes der drei folgenden Speicherbelegungsprotokolle um jeweils vier Schritte weiter.

| Haltepunkt | RM (wächst nach links) | Umgebung 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `label7` | – | z<br>0 | x<br>2 | y<br>? | | | | | | | | |

| Haltepunkt | RM (wächst nach links) | Umgebung 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `label4` | 2 : 1 : 5 | z<br>1 | 2 | ? | 3 | 2 | 1 | 1 | c<br>1 | d<br>1 | | |

| Haltepunkt | RM (wächst nach links) | Umgebung 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `label5` | 2 : 5 | z<br>0 | 2 | ? | 3 | 2 | c<br>1 | d<br>1 | | | | |

### Aufgabe 2 (AGS 3.2.31 a)

Gegeben sind zwei Felder `a` und `b` vom Typ `int` mit den Längen `m` bzw. `n`. Das *Skalarprodukt* der beiden Felder ist definiert als `a[0]*b[0] + a[1]*b[1] + ... + a[k−1]*b[k−1]` wobei $k = \min\{m, n\}$. Schreiben Sie eine Funktion `int skalarProdukt(int a[], int m, int b[], int n)` die das Skalarprodukt zweier Felder berechnet.

### Aufgabe 3 (AGS 3.2.41 ⋆)

Gegeben sei der folgende Datentyp für einfach-verkettete Listen:

```
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.

### Zusatzaufgabe 1 (AGS 3.2.41 ⋆)

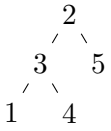
Geben Sie für die Funktionen in den Aufgaben 3 (a) and 3 (b) jeweils eine *iterative* und eine *rekursive* Lösung an.

### Zusatzaufgabe 2 (AGS 3.2.37 b, c)

Gegeben sei der folgende Datentyp für Binärbäume:

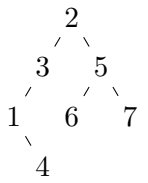
```
typedef struct element *tree;
struct element { int value;  tree left, right; };
```

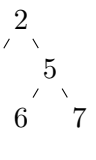
(a) Schreiben Sie eine Funktion `int evenSum(tree t)`, welche in einem Baum `t` die Summe der Beschriftungen jener Knoten berechnet, deren Entfernung von der Wurzel geradzahlig ist.

Beispiel: Wenn `t =` 

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

(b) 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 =` 

.