

# Aufgabenblatt zur 7. Übung

Zeitraum: 30.11. bis 04.12.2009

## 1. Aufgabe: (AGS 4.15\*)

Gegeben sei das folgende C-Programm:

```
1  #include <stdio.h>
2
3  void f (int *a, int *b);
4
5  void g (int *x, int *y) {
6      int z;
7      z = *y;          /*label 1*/
8      if (z > 0)
9          f(&z, y); /* $1 */
10     else
11         *x = z;
12     /*label 2*/
13 }
14
15 void f (int *a, int *b) {
16     *b = *a - 1;      /*label 3*/
17     while (*a > 1) {
18         g(a, b); /* $2 */ /*label 4*/
19     }
20 }
21
22 int main ()
23 {
24     int e, a;
25     scanf("%i", &e); /*label 5*/
26     f(&e, &a); /* $3 */ /*label 6*/
27     printf("%d", a);
28     return 0;
29 }
```

(a) Geben Sie den Gültigkeitsbereich jedes Objektes des Programms an.

(b) Stellen Sie eine Rechnung des Programms für die Eingabe  $e = 2$  als pulsierenden Speicher dar, wobei die aktuelle Situation bei jedem Passieren der Marken `label1` bis `label6` gezeigt werden soll. Dokumentieren Sie hierzu jeweils alle sichtbaren Variablen mit ihren Wertebelegungen. Hat eine Variable zu einem Zeitpunkt noch keinen Wert erhalten, so geben Sie anstelle des Wertes ein `?` an (also z. B.  $x = ?$ , wenn  $x$  zum Zeitpunkt der Protokollierung noch keinen Wert besitzt). Führen Sie des Weiteren auch ein Rücksprungmarkenprotokoll.

Beachten Sie: `$1$` bis `$3$` seien die bereits festgelegten Rücksprungmarken.

## 2. Aufgabe: (AGS 3.14)

Definieren Sie einen Datentyp `IntList`, der beliebig lange einfach verkettete Listen ganzer Zahlen darstellen kann. Implementieren Sie eine Funktion `void Insert(IntList *l, int n)`, die in eine bereits sortierte Liste `l` eine ganze Zahl `n` einsortiert. Schreiben Sie anschließend ein kleines Hauptprogramm zum Test der Funktion `Insert`.

## 3. Aufgabe: (AGS 3.15)

Implementieren Sie entsprechend der 2. Aufgabe eine Funktion `void Insert2(IntList *l, int n)`, die nach einer rekursiven Lösungsstrategie arbeitet. Vergleichen Sie anschließend die rekursive mit der iterativen Lösung.

## 4. Aufgabe: (AGS 3.19)

Gegeben seien die folgenden Typ- und Variablenvereinbarungen für binäre Bäume:

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

```
node Tree;
int anz;
```

(a) Schreiben Sie in *C* eine boolesche Funktion `int blatt(node t)`, die für einen beliebigen Baumknoten des gegebenen Typs ermittelt, ob es ein Blatt ist oder nicht. Liefern Sie als Funktionswert eine 1, wenn es sich um ein Blatt handelt, sonst den Wert 0.

(b) Gehen Sie davon aus, dass eine boolesche Funktion `int blatt(node t)` gemäß Aufgabe (a) existiert. Schreiben Sie in *C* eine rekursive Funktion `int anzahl(node t)`, die von einem gegebenen Baum des angegebenen Typs unter Verwendung der Funktion `blatt` die Anzahl derjenigen Knoten als Ergebnis liefert, welche keine Blätter als (unmittelbare) Nachfolger haben und selbst auch keine Blätter sind. Geben Sie abschließend ein Beispiel für einen Funktionsaufruf an.

## Zusatzaufgabe 1: (AGS 4.3\*)

Gegeben sei das folgende *C*-Programm:

```
#include <stdio.h>

void g(int x, int *y, int z) {
    /* label3 */
    if (x < 2) {
        g(x + 1, y, z); /* $2 */
        *y = *y * 2; /* label4 */
    }
    else *y = z;
}

int main() {
    int a, b, c;
    printf("\n Zahl a: ");
    scanf("%d", &a);
    printf("\n Zahl b: ");
    scanf("%d", &b); /* label1 */
}
```

```

    g(a, &c, b); /* $1 */ /* label2 */
    printf("\nDas Ergebnis lautet: %d\n\n", c);
    return 0;
}

```

(a) Geben Sie den Gültigkeitsbereich jedes Objektes des Programms an.

(b) Stellen Sie die Rechnung des Programmes für die Eingabe  $a = 0$  und  $b = 5$  als pulsierenden Speicher dar, wobei die aktuelle Situation bei jedem Passieren der Marken `label1` bis `label4` dokumentiert werden soll. Führen Sie des Weiteren auch ein Rücksprungmarkenprotokoll. Hat eine Variable noch keinen Wert durch eine Zuweisung erhalten, so geben Sie anstelle des Wertes ein `?` an (also z. B.  $x = ?$ , wenn  $x$  zum Zeitpunkt der aktuellen Marke noch unbesetzt ist).

Beachten Sie: `$1` und `$2` sind die bereits festgelegten Rücksprungmarken.

### Zusatzaufgabe 2: (AGS 3.16\*)

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 die Aussage (0 für nicht geordnet und 1 für geordnet) als Funktionswert zurückgibt.

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