Algorithmen und Datenstrukturen

Aufgabe 1 (AGS 3.1.10 a)

Es soll geprüft werden, ob der Inhalt des Zeichenfeldes `feld` der Länge `l` ein Palindrom ist, das heißt ob die Zeichenkette sowohl von vorne als auch von hinten gelesen gleich lautet. Ist die Zeichenkette ein Palindrom, so soll der aktuelle Parameter `korrekt` im Aufruf von `palindrom1` den Wert `1` annehmen, sonst `0`.
Von folgender Funktion wird behauptet, dass sie diese Aufgabe löst:

```
1 palindrom1(char feld[], int l, int korrekt) {
2   int i = 1;
3   l = l - 1;
4   while (i < l && korrekt) {
5     korrekt = feld[i] == feld[l];
6     i = i + 1;
7   }
8   return korrekt;
9 }
```

Prüfen Sie diese Funktion. Korrigieren Sie eventuell enthaltene Fehler.

Aufgabe 2 (AGS 3.2.46 b, AGS 3.2.41 ⋆)

Gegeben sei der folgende Datentyp für einfach-verkettete Listen:

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

(a) Geben Sie eine Funktion `void append(list *lp, int n)` an, welche ein neues Element e mit Schlüsselwert `n` erzeugt, dieses hinten an eine gegebene Liste `*lp` anhängt, und den Nachfolger von e auf `NULL` setzt. Gehen Sie davon aus, dass `lp != NULL` ist.

(b) Geben Sie Aufrufe der Funktion `append` an, sodass eine Liste mit den Elementen $4, 2, 0$ erzeugt wird.

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

(d) Implementieren Sie eine Funktion `rmEvens`, welche aus einer Liste alle Elemente mit einer geraden Zahl entfernt. Dabei soll die bestehende Liste verändert werden.

(e) Implementieren Sie eine Funktion `list odds(list lp)`, welche eine neue Liste erzeugt, die aus `lp` hervorgeht, indem alle Elemente mit einer geraden Zahl entfernt werden. Dabei soll die bestehende Liste `lp` *nicht* verändert werden.

Zusatzaufgabe 1 (AGS 3.2.49 a ⋆)

Vervollständigen Sie die C-Funktion `fill_occurrences`, so dass sie jeder Position `j` des Arrays `b` die Anzahl der Vorkommen von `a[j]` im Array `a` zuweist. Beide Arrays, `a` und `b`, haben die Länge `bound`.

```
void fill_occurrences(int a[], int b[], int bound) {
```

Zusatzaufgabe 2 (AGS 3.2.46, AGS 3.2.41 ⋆)

Geben Sie für die Funktionen in den Aufgaben 1 und 2 jeweils eine *iterative* und eine *rekursive* Lösung an.

Zusatzaufgabe 3 (AGS 4.30 ⋆)

```
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 0	x 2	y ?								

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

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