Algorithmen und Datenstrukturen

Aufgabe 1 (AGS 3.1.9 ⋆)

Gegeben sei die Ackermann-Funktion ack: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Implementieren Sie diese in C.

$$\begin{aligned} \operatorname{ack}(0, y) &= y + 1 && (y \geq 0) \\ \operatorname{ack}(x, 0) &= \operatorname{ack}(x - 1, 1) && (x > 0) \\ \operatorname{ack}(x, y) &= \operatorname{ack}(x - 1, \operatorname{ack}(x, y - 1)) && (x, y > 0) \end{aligned}$$

Aufgabe 2 (AGS 3.1.16)

(a) Das folgende C-Programm ist gegeben.

```c
#include <stdio.h>

void swoop(int a, int b) {
  /* label 1 */
  a = b;
  b = a;
  /* label 2 */
}

int main() {
  int x = 3, y = 6;
  /* label 3 */
  swoop(x, y); /*$1*/
  /* label 4 */
  printf("x = %d, y = %d", x, y);
  return 0;
}
```

Geben Sie ein Speicherbelgungsprotokoll für dieses Programm an.

(b) Schreiben Sie eine C-Funktion `swap` mit zwei Parametern, welche die Werte der aktuellen Parameter vertauscht.

Aufgabe 3 (AGS 4.22)

Gegeben sei folgendes C-Programm.

```c
#include <stdio.h>

void g(int x, int *y);

void f(int *x, int y){
  /* label1 */
  while (*x < y){
    *x = *x * 3;
    /* label2 */
    g(*x, &y);    /* $1 */
  }
}

void g(int x, int *y){
  /* label3 */
  if (*y < x){
    *y = *y * 2;
    /* label4 */
    if (x > *y)
      f(&x, *y);  /* $2 */
  }
  /* label5 */
}

int main(){
  int a, b;
  a = 3;
  b = 6;
  /* label6 */
  f(&a, b);       /* $3 */
  /* label7 */
  printf("%d", a);
  return 0;
}
```

(a) Tragen Sie den Gültigkeitsbereich jedes Objektes in eine Tabelle ein. Nutzen Sie dazu die Zeilennummern.

(b) Setzen Sie das folgende Speicherbelegungsprotokoll fort.

Haltepunkt	RM	1	2	3	4	5	6	7	8
`label6`	–	`a` 3	`b` 6						

Zusatzaufgabe 1 (AGS 3.1.6)

Schreiben Sie für die Berechnung der Fakultätsfunktion eine C-Funktion, welche eine natürliche Zahl als Parameter fordert und den zugeordneten Funktionswert zurückgibt. Geben Sie eine *iterative* und eine *rekursive* Lösung an!

Zusatzaufgabe 2 (AGS 4.18)

Gegeben sei folgendes C-Programm.

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

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

(b) Setzen Sie das folgende Speicherbelegungsprotokoll fort.

Haltepunkt	RM	1	2	3	4	5	6	7	8
`label6`	–	`x` ?							

Zusatzaufgabe 3 (AGS 3.1.10)

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 Parameter `korrekt` den Wert `true` repräsentieren, sonst `false`.

(a) Von folgender Funktion wird behauptet, dass sie diese Aufgabe löst:

```
1  palindrom1(char feld[], int l, int korrekt) {
2    int i;
3    i = 1;
4    l = l − 1;
5    while (i < l && korrekt) {
6      korrekt = feld[i] == feld[l];
7      i = i + 1;
8    }
9    return korrekt;
10 }
```

Prüfen Sie diese Funktion. Korrigieren Sie eventuell enthaltene Fehler.

(b) Schreiben Sie eine Funktion `palindrom2`, die rekursiv arbeitet. Überlegen Sie sich hierbei als erstes einen geeigneten Funktionskopf.