Algorithmen und Datenstrukturen

6. Übungsblatt

Zeitraum: 23. November – 27. November 2015

Übung 1 (AGS 4.19)
Gegeben sei folgendes C-Programm.

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

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

(b) Setzen Sie das folgende Speicherbelegungsprotokoll fort. Dokumentieren Sie die aktuelle Situation beim Passieren der Marken (*label1* bis *label7*). Geben Sie jeweils den Rücksprungmarkenkeller und die *sichtbaren* Variablen mit ihrer Wertebelegung an. Die Inhalte von Speicherzellen nicht-sichtbarer Variablen brauchen Sie nur bei Änderungen einzutragen. Beachten Sie: *$1* bis *$3* sind die bereits festgelegten Rücksprungmarken.

Haltepunkt	RM	Umgebung									
		1	2	3	4	5	6	7	8	9	10
label6	–	a	n	c							
		6	6	0							

Übung 2
Im Auftrag einer bekannten Universität implementieren Sie in C ein System zur Studentenverwaltung (Projektname: yExam). Folgender Code ist bereits gegeben:

```
#define MAXLEN 32

struct exam_result {
  char exam_name[MAXLEN];
  int exam_grade;
};

struct student {
  char name[MAXLEN];
  struct exam_result *exams[MAXLEN];
};
```

```
int main()
{
  struct student* db[2]; int i;

  db[0] = new_student("Max Mustermann");
  add_exam(db[0], "GTI", 4);
  add_exam(db[0], "SyA", 3);

  db[1] = new_student("Sepp Wurz");
  add_exam(db[1], "GTI", 1);
  add_exam(db[1], "SyA", 1);

  for(i = 0; i < 2; ++i) print_student(db[i]);
  return 0;
}
```

(a) Implementieren Sie folgende Funktionalitäten:
- Eine Funktion new_student passenden Typs, welche dynamisch Speicher für eine neue student-Struktur allokiert und diese initialisiert.
- Eine Funktion add_exam passenden Typs, welche einem bestehenden Studenten ein Prüfungsergebnis hinzufügt.
- Eine Funktion print_student, welche den Namen und die Ergebnisse eines Studenten ausgibt.

(b) *(Zusatzaufgabe)* Der Exzellenzbeauftragte der Universität ist besorgt, dass die Studentendatenbak db nur begrenzt viele Studenten aufnehmen kann. Außerdem wollen manche Studenten mehr als 32 Prüfungsleistungen einbringen. Lösen Sie diese Probleme, indem Sie die vorliegenden Datentypen und Funktionen abändern!

(c) *(Zusatzaufgabe)* Implementieren Sie weitere Funktionen, z.B.: zum Bestimmen von Studenten mit guten/schlechten Ergebnissen, ...

Übung 3 (AGS 3.1.9)
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:

```
palindrom1(char feld[], int l, int korrekt) {
  int i;
  i = 1;
  l = l - 1;
  while (i < l && korrekt) {
    korrekt = feld[i] == feld[l];
    i = i + 1;
  }
  return korrekt;
}
```

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.

Zusatzaufgabe 1 (AGS 4.16)
Gegeben sei folgendes C-Programm:

```
 1  #include <stdio.h>
 2
 3  void g(int *a, int *b) {
 4      int t = *a;
 5      /* label1 */
 6    if (*a < *b) {
 7      *a = *b;
 8      *b = t;
 9        /* label2 */
10    }
11  }
12
13  void f(int *c, int d, int *e) {
14      /* label3 */
15    g(c, &d) /* $1 */;
16      /* label4 */
17    if (*c <= d)
18      return;
19    *c = *c - d;
20    f(c, d, &d) /* $2 */;
21      *e = d + 1;
22      /* label5 */
23  }
24
25  int main() {
26    int n, m;
27    n = 6; m = 12;
28      /* label6 */
29    f(&n, m, &m) /* $3 */;
30      /* label7 */
31    return 0;
32  }
```

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

Setzen Sie das folgende Speicherbelegungsprotokoll fort.

Haltepunkt	RM	Umgebung										
		1	2	3	4	5	6	7	8	9	10	11
label6	–	n	m									
		6	12									
label3	3			c	d	e						
				#1	12	#2						