

## Angewandte Informatik – Klausur – 3. Februar 2014

Prof. Dr. Peter Gerwinski, Wintersemester 2013/14

Name:	
Matrikel-Nr.:	

Zeit: 120 Minuten

Zulässige Hilfsmittel:

- Schreibgerät
- Beliebige Unterlagen in Papierform und/oder auf Datenträgern
- Elektronische Rechner (Notebook, Taschenrechner o. ä.)  
*ohne* Zugang zu Datennetzen jeglicher Art

Nur die o. a. zulässigen Hilfsmittel dürfen sich während der Klausur im Arbeitsbereich befinden.  
WLAN-, Bluetooth- und sonstige Funkeinheiten von Notebooks o. ä. sind per Hardware auszuschalten.  
Mobiltelefone, Geräte mit mobilem Internet-Zugang u. ä. sind auszuschalten und in der Tasche zu verstauen.

Die reguläre Maximalpunktzahl beträgt 42 Punkte.  
Bei besonderen Leistungen sind Zusatzpunkte möglich.  
Mit 20 erreichten Punkten gilt die Klausur als bestanden.

Das Passwort für den Zugriff auf die Beispielpprogramme lautet: **I8eqoWcpQa23**

## Aufgabe 1: Strings

Wir betrachten die folgende Funktion (aufgabe-1.c):

```
int f (char *s0, char *p0)
{
    int found0 = 0;
    while (*s0)
    {
        char *s1 = s0++;
        char *p1 = p0;
        int found1 = 1;
        while (*s1 && *p1)
            if (*s1++ != *p1++)
                found1 = 0;
        if (found1)
            found0 = 1;
    }
    return found0;
}
```

(a) Was bewirkt diese Funktion und warum? (4 Punkte)

(b) Von welcher Ordnung (Landau-Symbol) ist die Funktion und warum?

Wir beziehen uns hierbei auf die Anzahl der **char**-Vergleiche in Abhängigkeit von der Länge der Eingabedaten **s0** und **p0**. Für die Rechnung dürfen Sie beide Längen mit  $n$  gleichsetzen, obwohl sie normalerweise nicht gleich sind. (2 Punkte)

- (c) Was passiert, wenn Sie beim Aufruf der Funktion für einen der beiden Parameter den Wert **NULL** übergeben und warum? (2 Punkte)

- (d) Beschreiben Sie, wie sich die Funktion effizienter gestalten läßt. (4 Punkte)

- (e) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Version der Funktion und warum? (2 Punkte)

## Aufgabe 2: Objektorientierte Tier-Datenbank

```
#include <stdio.h>

#define ANIMAL 0
#define WITH_WINGS 1
#define WITH_LEGS 2

typedef struct animal
{
    int type;
    char *name;
} animal;

typedef struct with_wings
{
    int wings;
} with_wings;

typedef struct with_legs
{
    int legs;
} with_legs;

int main (void)
{
    animal *a[2];

    animal duck;
    a[0] = &duck;
    a[0]—>type = WITH_WINGS;
    a[0]—>name = "duck";
    a[0]—>wings = 2;  ← ((with_wings *) a[0])—>wings = 2;

    animal cow;
    a[1] = &cow;
    a[1]—>type = WITH_LEGS;
    a[1]—>name = "cow";
    a[1]—>legs = 4;  ← ((with_legs *) a[1])—>legs = 4;

    for (int i = 0; i < 2; i++)
        if (a[i]—>type == WITH_LEGS)
            printf ("A_%s_has_%d_legs.\n", a[i]—>name,
                    ((with_legs *) a[i])—>legs);
        else if (a[i]—>type == WITH_WINGS)
            printf ("A_%s_has_%d_wings.\n", a[i]—>name,
                    ((with_wings *) a[i])—>wings);
        else
            printf ("Error_in_animal:_%s\n", a[i]—>name);

    return 0;
}
```

Das oben in Blau dargestellte Programm ([aufgabe-2a.c](#)) soll Daten von Tieren verwalten.

Beim Compilieren erscheinen die folgende Fehlermeldungen:

```
$ gcc -std=c99 -Wall -O aufgabe-2a.c -o aufgabe-2a
aufgabe-2a.c: In function 'main':
aufgabe-2a.c:31: error: 'animal' has no member named 'wings'
aufgabe-2a.c:37: error: 'animal' has no member named 'legs'
```

Der Programmierer nimmt die oben in Rot dargestellten Ersetzungen vor (Datei [aufgabe-2b.c](#)). Daraufhin gelingt das Compilieren, und die Ausgabe des Programms lautet:

```
$ gcc -std=c99 -Wall -O aufgabe-2b.c -o aufgabe-2b
$ ./aufgabe-2b
A duck has 2 legs.
Error in animal: cow
```

(a) Erklären Sie die o. a. Compiler-Fehlermeldungen. (2 Punkte)

(b) Wieso verschwinden die Fehlermeldungen nach den o. a. Ersetzungen? (3 Punkte)

(c) Erklären Sie die Ausgabe des Programms. (5 Punkte)

(c) Beschreiben Sie – in Worten und/oder als C-Quelltext – einen Weg, das Programm so zu berichtigen, daß es die Eingabedaten ("A duck has 2 wings. A cow has 4 legs.") korrekt speichert und ausgibt. (4 Punkte)

### Aufgabe 3: Dynamisches Bit-Array

Schreiben Sie die folgenden Funktionen zur Verwaltung eines dynamischen Bit-Arrays:

- **void bit\_array\_init (int n)**  
Das Array initialisieren, so daß man  $n$  Bits darin speichern kann.  
Die Array-Größe  $n$  ist keine Konstante, sondern erst im laufenden Programm bekannt.  
Die Bits sollen auf den Anfangswert 0 initialisiert werden.
- **void bit\_array\_set (int i, int value)**  
Das Bit mit dem Index  $i$  auf den Wert  $value$  setzen.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen; der Wert  $value$  darf 1 oder 0 sein.
- **void bit\_array\_flip (int i)**  
Das Bit mit dem Index  $i$  auf den entgegengesetzten Wert setzen,  
also auf 1, wenn er vorher 0 ist, bzw. auf 0, wenn er vorher 1 ist.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen.
- **int bit\_array\_get (int i)**  
Den Wert des Bit mit dem Index  $i$  zurückliefern.  
Der Index  $i$  darf von 0 bis  $n - 1$  gehen.
- **void bit\_array\_resize (int new\_n)**  
Die Größe des Arrays auf  $new\_n$  Bits ändern.  
Dabei soll der Inhalt des Arrays, soweit er in die neue Größe paßt, erhalten bleiben.  
Neu hinzukommende Bits sollen auf 0 initialisiert werden.
- **void bit\_array\_done (void)**  
Den vom Array belegten Speicherplatz wieder freigeben.

Bei Bedarf dürfen Sie den Funktionen zusätzliche Parameter mitgeben, beispielsweise um mehrere Arrays parallel verwalten zu können. (In der objektorientierten Programmierung wäre dies der implizite Parameter `this`, der auf die Objekt-Struktur zeigt.)

Die Bits sollen möglichst effizient gespeichert werden, z. B. jeweils 8 Bits in einer `uint8_t`-Variablen.

Die Funktionen sollen möglichst robust sein, d. h. das Programm darf auch bei unsinnigen Parameterwerten nicht abstürzen, sondern soll eine Fehlermeldung ausgeben.

(14 Punkte)

Abgabe auf Datenträger ist erwünscht, aber nicht zwingend. Raum für Notizen: