

## Hardwarenahe Programmierung / Angewandte Informatik Klausur – 6. Februar 2017

Prof. Dr. Peter Gerwinski, Wintersemester 2016/17

Name:	
Matrikel-Nr.:	
Benutzername:	29
Passwort:	ksVY3vujsxPY
Prüfsumme der hochgeladenen Datei:	

Zeit: 150 Minuten

Zulässige Hilfsmittel:

- Schreibgerät
- Beliebige Unterlagen in Papierform und/oder auf Datenträgern
- Elektronische Rechner (Notebook, Taschenrechner o. ä.)
- Zugang zum Klausur-WLAN

Nur die o. a. zulässigen Hilfsmittel dürfen sich während der Klausur im Arbeitsbereich befinden.

Der einzige zulässige Zugang zu Datennetzen jeglicher Art (LAN, WLAN, Bluetooth, ...) ist der Zugang zum Klausur-WLAN, das während der Klausur unter der ESSID [klausur](#) mit Passwort [klausurklausur](#) zugänglich ist. Sonstige Funkeinheiten (z. B. Bluetooth) von Notebooks o. ä. sind auszuschalten; ggf. dafür vorhandene physische Schalter sind zu benutzen. 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.

Die Beispielpprogramme werden Ihnen über das **Klausur-WLAN** unter der URL <http://klausur> zum Herunterladen angeboten. Unter derselben URL finden Sie auch ein Web-Interface zum Hochladen *einer einzigen Datei* (normalerweise eine Archiv-Datei) mit Ihren Klausurergebnissen. Bei mehrfachem Hochladen wird die vorherige Version überschrieben. Zulässige Archiv-Dateiformate sind [tar.gz](#), [tar.bz2](#), [tar.xz](#), [zip](#) und [7z](#).  
**Wichtig: Bitte tragen Sie nach dem Hochladen die Prüfsumme oben auf diesem Blatt in das dafür vorgesehene Feld ein**, damit wir die Datei eindeutig Ihnen zuordnen können.

Wenn Sie nicht über einen Zugang zum Klausur-WLAN verfügen, stellen wir Ihnen alternativ die Beispielpprogramme auf Datenträger (USB-Stick) zur Verfügung. Die Abgabe von digital gelösten Aufgaben hat dann auf demselben Datenträger zu erfolgen.

## Aufgabe 1: Strings umsortieren

Wir betrachten das folgende Programm ([aufgabe-1.c](#)):

```

1  #include <stdio.h>
2  #include <string.h>
3
4  void fun (char *s)
5  {
6      int len = strlen (s);
7      for (int i = 0; i < len; i++)
8          {
9              for (int j = i - 1; j >= 0; j--)
10                 {
11                     if (s[i] < s[j])
12                         {
13                             char sx = s[i];
14                             s[i] = s[j];
15                             s[j] = sx;
16                         }
17                 }
18             }
19  }
20
21  int main (void)
22  {
23      char s[] = "BAECD";
24      fun (s);
25      printf ("%s\n", s);
26      return 0;
27  }

```

Auf einem Rechner, der den ASCII-Zeichensatz verwendet, lautet die Ausgabe: `ABCDE`

- (a) Beweisen Sie (z. B. anhand eines Gegenbeispiels), daß die Funktion allgemein **fun** nicht dazu geeignet ist, die Buchstaben eines Strings gemäß ihrer ASCII-Reihenfolge zu sortieren. (2 Punkte)

- (b) Was kann passieren, wenn die Zeile `char s[] = "BAECD";` durch `char s[] = { 'B', 'A', 'E', 'C', 'D' };` ersetzt wird, und warum? (2 Punkte)

- (c) Von welcher Ordnung (Landau-Symbol) ist die Funktion  $\text{fun}$  und warum?

Wir beziehen uns hierbei auf die Anzahl der Vergleiche `s[i] < s[j]` in Abhängigkeit von der Länge des Eingabe-Strings "BAECD". (1 Punkt)

- (d) Beschreiben Sie – in Worten und/oder als C-Quelltext –, wie sich die Funktion `fun` so abwandeln läßt, daß sie die Buchstaben des Strings `s` gemäß ihrer ASCII-Reihenfolge sortiert. Von welcher Ordnung (Landau-Symbol) ist Ihre Version der Funktion und warum? (3 Punkte)

## Aufgabe 2: Speicherformate von Zahlen

Wir betrachten das folgende Programm ([aufgabe-2.c](#)):

```
1 #include <stdio.h>
2 #include <stdint.h>
3
4 int main (void)
5 {
6     uint16_t numbers[] = { 25928, 27756, 11375, 30496, 29295, 25708, 2593, 0 };
7     printf ("%s", numbers);
8     return 0;
9 }
```

```
$ gcc -Wall aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function 'main:
aufgabe-2.c:7:3: warning: format '%s' expects argument of type 'char ',
                but argument 2 has type 'uint16_t ' [-Wformat]
$ ./aufgabe-2
Hello, world!
```



Dieses Programm stellt eine allgemeine Sortier-Funktion `sort` zur Verfügung, die prinzipiell beliebige Arrays sortieren kann – also z. B. sowohl Strings als auch Zahlen.

- (a) Erklären Sie die Struktur des Arrays `numbers`. Was ist in dem Array gespeichert, und wo befinden sich die Zahlen? Was ist der Unterschied zwischen `&zero` und dem Wert `NULL`? (3 Punkte)

- (b) Was bedeutet `int (*compare) (void *x1, void *x2)`, und für welchen Zweck wird es verwendet? Stellen Sie einen Bezug zur objektorientierten Programmierung her. (3 Punkte)

- (c) Was kann passieren, wenn man in den Arrays auf das letzte Element **NULL** verzichtet und warum? (1 Punkt)

- (d) Ergänzen Sie das Fragment zu einem funktionsfähigen Programm, das beide Arrays `strings` und `numbers` sortiert ausgibt. (5 Punkte)

Abgabe über das Klausur-WLAN oder auf Datenträger ist erwünscht, aber nicht zwingend. Für Notizen verwenden Sie nötigenfalls die Rückseite des letzten Klausurbogens und/oder zusätzliche Blätter.

## Aufgabe 4: PBM-Grafik

Bei einer PBM-Grafikdatei handelt es sich um ein abgespeichertes C-Array von Bytes (`uint8_t`), das die Bildinformationen enthält:

- Die Datei beginnt mit der Kennung `P4`, danach folgen Breite und Höhe in Pixel als ASCII-Zahlen, danach ein Trennzeichen und die eigentlichen Bilddaten.
- Jedes Bit entspricht einem Pixel.
- Nullen stehen für Weiß, Einsen für Schwarz.
- MSB first.
- Jede Zeile des Bildes wird auf ganze Bytes aufgefüllt.

Viele Grafikprogramme können PBM-Dateien öffnen und bearbeiten. Der Anfang der Datei (Kennung, Breite und Höhe) ist auch in einem Texteditor lesbar.

Beispiel (`aufgabe-4.pbm`):

```
P4
14 14
<Bilddaten>
```



In dem untenstehenden Programmfragment (`aufgabe-4.c`) wird eine Grafik aus Textzeilen zusammengesetzt, so daß man mit einem Texteditor „malen“ kann:

```
#include <stdio.h>

[...]

int main (void)
{
    pbm_open (14, 14, "test.pbm");
    pbm_line ("                ");
    pbm_line ("      XXXXXX      ");
    pbm_line ("   X      X   ");
    pbm_line ("  X      X  ");
    pbm_line (" X      X ");
    pbm_line ("X  XX  XX  X");
    pbm_line (" X  X  X  X");
    pbm_line (" X      X ");
    pbm_line (" X  X      X  ");
    pbm_line (" X  X      X  ");
    pbm_line (" X  XXXX  X  ");
    pbm_line ("   X      X   ");
    pbm_line ("      XXXXXX      ");
    pbm_line ("                ");
    pbm_close ();
    return 0;
}
```

Ergänzen Sie das Programmfragment so, daß es eine Datei `test.pbm` erzeugt, die die Grafik enthält.

Das Programm soll typische Benutzerfehler abfangen (z. B. weniger Zeilen als in `pbm_open` angegeben), keine fehlerhaften Ausgaben produzieren oder abstürzen, sondern stattdessen sinnvolle Fehlermeldungen ausgeben.

Zum Vergleich liegt eine Datei `aufgabe-4.pbm` mit dem gewünschten Ergebnis bei, und die Datei `aufgabe-4.png` enthält dasselbe Bild.

(10 Punkte)

Abgabe über das Klausur-WLAN oder auf Datenträger ist erwünscht, aber nicht zwingend. Für Notizen verwenden Sie nötigenfalls die Rückseite des letzten Klausurbogens und/oder zusätzliche Blätter.

*Viel Erfolg!*

Raum für Notizen