

## Hardwarenahe Programmierung / Angewandte Informatik Klausur – 13. Februar 2018

Prof. Dr. Peter Gerwinski, Wintersemester 2017/18

Name:	
Matrikel-Nr.:	
Benutzername:	12
Passwort:	8Hdy2fG2ZvIk
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: Einfügen in Strings

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

```

1  #include <stdio.h>
2  #include <string.h>
3
4  void insert_into_string (char src, char *target, int pos)
5  {
6      int len = strlen (target);
7      for (int i = pos; i < len; i++)
8          target[i+1] = target[i];
9      target[pos] = src;
10 }
11
12 int main (void)
13 {
14     char test[100] = "Hochschule_Bochem";
15     insert_into_string ('c', test, 5);
16     printf ("%s\n", test);
17     return 0;
18 }

```

Die Ausgabe des Programms lautet: Hochschhhhhhhhhh

(a) Erklären Sie die Ausgabe. (3 Punkte)

(b) Schreiben Sie die Funktion `insert_into_string()` so um, daß sie den Buchstben `src` an der Stelle `pos` in den String `target` einfügt.

Die Ausgabe des Programms müßte dann Hochschule Bochum lauten. (2 Punkte)

- (c) Was kann passieren, wenn Sie die Zeile `char test[100] = "Hochschule_Bochum";` durch `char test[] = "Hochschule_Bochum";` ersetzen und warum? (2 Punkte)



## Aufgabe 2: Speicherformate von Zahlen

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

```
1 #include <stdio.h>
2 #include <stdint.h>
3
4 typedef struct
5 {
6     uint32_t a;
7     uint64_t b;
8     uint8_t c;
9 } three_numbers;
10
11 int main (void)
12 {
13     three_numbers xyz = { 1819042120, 2410670883059281007, 0 };
14     printf ("%s\n", &xyz);
15     return 0;
16 }
```

Das Programm wird für einen 32-Bit-Rechner compiliert und ausgeführt.

(Die `gcc`-Option `-m32` sorgt dafür, daß `gcc` Code für einen 32-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -m32 aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function "main":
aufgabe-2.c:14:13: warning: format "%s" expects argument of type "char *", but
argument 2 has type "three_numbers * {aka struct <anonymous> *}" [-Wformat=]
    printf ("%s\n", &xyz);
                   ^
$ ./aufgabe-2
Hallo, Welt!
```

(a) Erklären Sie die beim Compilieren auftretende Warnung. (2 Punkte)

(b) Erklären Sie die Ausgabe des Programms. (4 Punkte)

- (c) Welche Endianness hat der verwendete Rechner? Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus? (2 Punkte)

- d) Dasselbe Programm wird nun für einen 64-Bit-Rechner kompiliert und ausgeführt.  
(Die `gcc`-Option `-m64` sorgt dafür, daß `gcc` Code für einen 64-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -m64 aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function "main":
aufgabe-2.c:14:13: warning: format "%s" expects argument of type "char *",
but argument 2 has type "three_numbers * {aka struct <anonymous> *}"
[-Wformat=]
    printf ("%s\n", &xyz);
                   ^
$ ./aufgabe-2
Hall15V
```

- (Es ist möglich, daß die konkrete Ausgabe auf Ihrem Rechner anders aussieht.)  
Erklären Sie die geänderte Ausgabe des Programms. (3 Punkte)

### Aufgabe 3: Iterationsfunktionen

Wir betrachten das folgende Programm ([aufgabe-3.c](#)):

```

1  #include <stdio.h>
2
3  void foreach (int *a, void (*fun) (int x))
4  {
5      for (int *p = a; *p >= 0; p++)
6          fun (*p);
7  }
8
9  void even_or_odd (int x)
10 {
11     if (x % 2)
12         printf ("%d_ist_ungerade.\n", x);
13     else
14         printf ("%d_ist_gerade.\n", x);
15 }
16
17 int main (void)
18 {
19     int numbers[] = { 12, 17, 32, 1, 3, 16, 19, 18, -1 };
20     foreach (numbers, even_or_odd);
21     return 0;
22 }

```

- (a) Was bedeutet **void (\*fun) (int x)**, und welchen Sinn hat seine Verwendung in der Funktion **foreach()**? (2 Punkte)

- (b) Schreiben Sie das Hauptprogramm `main()` so um, daß es unter Verwendung der Funktion `foreach()` die Summe aller positiven Zahlen in dem Array berechnet. Sie dürfen dabei weitere Funktionen sowie globale Variable einführen. (4 Punkte)

## Aufgabe 4: XBM- und PBM-Grafik

Bei einer XBM-Grafikdatei handelt es sich um ein als C-Quelltext abgespeichertes Array, das die Bildinformationen enthält:

- Jedes Bit entspricht einem Pixel.
- Nullen stehen für Weiß, Einsen für Schwarz.
- LSB first.
- Jede Zeile des Bildes wird auf ganze Bytes aufgefüllt.
- Breite und Höhe des Bildes sind als Konstantendefinitionen (**#define**) in der Datei enthalten.

Sie können eine XBM-Datei sowohl mit einem Texteditor als auch mit vielen Grafikprogrammen öffnen und bearbeiten.

Beispiel ([aufgabe-4.xbm](#)):

```
#define aufgabe_4_width 14
#define aufgabe_4_height 14
static unsigned char aufgabe_4_bits[] = {
    0x00, 0x00, 0xf0, 0x03, 0x08, 0x04, 0x04, 0x08, 0x02, 0x10, 0x32, 0x13,
    0x22, 0x12, 0x02, 0x10, 0x0a, 0x14, 0x12, 0x12, 0xe4, 0x09, 0x08, 0x04,
    0xf0, 0x03, 0x00, 0x00 };

```



Ein C-Programm, das eine XBM-Grafik nutzen will, kann die [.xbm](#)-Datei mit **#include** `"..."` direkt einbinden.

Bei einer PBM-Grafikdatei handelt es sich um ein binär 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 genau ein Trennzeichen (z. B. `"\n"`) und zuletzt 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>

```



Schreiben Sie ein Programm, das die Datei [aufgabe-4.xbm](#) mittels **#include** liest und in einer PBM-Bilddatei [test.pbm](#) abspeichert. Das Programm wandelt also eine XBM-Datei in eine PBM-Datei um.

Für Ihre Ergebniskontrolle liegt eine Datei [aufgabe-4.pbm](#) mit dem gewünschten Ergebnis bei, und die Datei [aufgabe-4.png](#) enthält dasselbe Bild.

Hinweis: Verwenden Sie `"%c"` für die Ausgabe der eigentlichen Bilddaten.

(8 Punkte)

Abgabe auf Datenträger ist erwünscht, aber nicht zwingend.

*Viel Erfolg!*