Übungsblatt 1

Abgabe: 06. Mai 2020 um 8:30 Uhr

Abgabe von Übungen: Übungen werden über den Moodle-Lernraum abgegeben. Zu jedem Übungsblatt haben wir ein Abgabesystem angelegt, wo ihr pro Gruppe genau eine Abgabe tätigt. Wir akzeptieren als Dateiformate nur `.pdf`, `.sh` und `.c`. Ihr könnt gerne die Hausaufgaben auch handschriftlich lösen, diese Einscannen und als ein einziges `.pdf` abgeben. Das heißt, Teillösungen bitte zu einem Dokument zusammenfassen (exklusive Code). Achtet bitte darauf, dass die Namen aller Teammitglieder inklusive derer Matrikelnummern auf dem Dokument stehen! Bezüglich der Abgabe von Code-Aufgaben: Euer Code muss mit `gcc -Wall` ohne Warnings kompilieren. Tut er dies nicht, müssen wir die Aufgabe leider mit 0 Punkten bewerten.

Aufgabe 1.1: Linux-, Shell und C-Playground (0 Punkte!)

Wir fangen mit ein paar 'Übungen' zum warm werden an, um uns ein bisschen mit Linux vertraut zu machen.

Achtung! Zu dieser Aufgabe gibt es nichts abzugeben und es werden somit auch keine Punkte vergeben.

a) **Get and install Linux**

Zum Bearbeiten der Aufgaben benötigt ihr ein Linux – also einfach das aktuelle Windows-System plattmachen und eine Linux-Distribution runterladen und installieren. ;)
Um sich einen Überblick über verschiedene Distributionen zu verschaffen, kann man hier nachschauen: `http://distrowatch.com/dwres.php?resource=major` Für Anfänger geeignet ist z.B. Linux Mint mit Mate Desktop (`https://linuxmint.com/download.php`).

Wer lieber bei seinem aktuellen System bleiben und/oder nicht gleich einen Dualboot anlegen will, kann eine vollständige Linux-Distribution auch einfach in einer virtuellen Maschine z.B. mittels VirtualBox installieren: `http://www.virtualbox.org/`. Es gibt bereits fertige VirtualBox-Images auf `http://virtualboxes.org/images/`, so entfällt die Installation in der VM.

Installiert auch die sogenannten GuestAdditions mit – sie erlauben z.B. die Einrichtung gemeinsamer Ordner zwischen der virtuellen Maschine und dem Gastsystem in VirtualBox.

b) **RTFM - Read The 'Friendly' Manual!!!**

Wer Probleme mit den Shell- und teilweise auch den C-Aufgaben hat, kann natürlich Google oder Ähnliches zur Hilfe heranziehen – es geht aber auch professioneller. Öffnet dazu eine Shell und gebt die folgenden Befehle ein (ohne $):

```
$ man man
$ man grep
...
```

Hinweis: Read them. Really!

Aufgabe 1.2: Grundlagen der Betriebssysteme (1+1,5+1 = 3,5 Punkte)

a) Was ist der Kernel- und User-Mode? Welchen Vorteil bringt diese Unterscheidung?

b) Welche der folgenden Operationen/Treiber sollten nur im Kernel-Mode ausgeführt werden, welche können im User-Mode ausgeführt werden? Begründen Sie jeweils Ihre Wahl.

- Aufruf einer Trap
- Grafikkartentreiber (für High-End Gaming PC)

- Druckertreiber

c) Wann ist es sinnvoll, Polling einzusetzen? Wann ergibt der Einsatz von Interrupts mehr Sinn?

Aufgabe 1.3: C-Datentypen (0,5+0,5+0,5+0,5+0,5+1,5 = 4 Punkte)

Gegeben seien folgende Deklarationen:

```
char data[] = "BuK Zero";
int A[2][3] = {{3, 4, 2}, {2, 3, 2}};
char* z;
int i = 1;
int* pi;
```

Welche der folgenden Operationen sind möglich? Falls eine Operation nicht möglich ist: warum nicht? Falls sie möglich ist: was bewirkt sie? Gehen Sie davon aus, dass eine Operation das Ergebnis der darauf folgenden Operationen beeinflusst, wenn sie möglich ist.

a) `z = data + 4;`

b) `*(data + 4) = 'H';`

c) `data[2] = 'S';`

d) `pi = &i;`

e) `&i = pi;`

f) `z = &data[*A[1]] - *pi;`

Aufgabe 1.4: Zahlentypen und Ausgabe (1+2 = 3 Punkte)

Die Programmiersprache C bietet Zahlentypen mit unterschiedlichen Größen, Wertebereichen und Präzisionen an, z.B.:

	Typ	Größe	Wertebereich			Beispiele
Ganzzahlig	`signed char`	8 Bit	-127	...	127	39, -12, 'a'
	`unsigned char`	8 Bit	0	...	255	
	`(signed) short`	16 Bit	-32767	...	32767	
	`unsigned short`	16 Bit	0	...	65535	
	`(signed) long`	32 Bit	-2147483647	...	2147483647	
	`unsigned long`	32 Bit	0	...	4294967295	
Fließkomma	`float`	32 Bit	-1×10^{38}	...	1×10^{38}	-2.0, -1.3e-12
	`double`	64 Bit	-1.8×10^{308}	...	1.8×10^{308}	

Beachten Sie, dass sich die Wertebereiche für einen bestimmten Compiler auf einem bestimmten Betriebssystem von den hier angegebenen unterscheiden können.

Die meisten Datentypen können gemischt in Berechnungen oder Zuweisungen verwendet werden. Dazu gibt es in C eine automatische (implizite) Typkonvertierung. Man kann also z.B. folgende Zuweisung machen:

```
int i=3;
float f=2.5;
...
f = i;
```

Implizite Typkonvertierungen stellen eine erhebliche Fehlerquelle dar. Man sollte daher die Konvertierung explizit anweisen, z.B. durch durch sogenanntes *casting* :

```
f = (float)i;
```

a) Übersetzen und testen Sie das Programm `umrechnung.c`, welches Sie zusammen mit dem Übungsblatt im Lernraum finden. Überprüfen Sie die Umrechnung von Kilometern in Meilen nach der genäherten Formel

$$d[km] = \frac{1609}{1000} \cdot d[mi]$$

und beseitigen Sie etwaige Fehler im Programm. Geben Sie das korrigierte Programm ab.

b) Modifizieren Sie das Programm so, dass keine Benutzereingabe erwartet wird, sondern eine Meilen-Kilometer–Umrechnungstabelle ausgegeben wird. Es sollen die Distanzen $0, 2, 4, 6, \ldots, 20$ Meilen umgerechnet und tabellarisch ausgegeben werden. Die Tabelle kann einfach gehalten sein, es reicht eine formatierte Ausgabe `Meilenwert | Kilometerwert` mit einer solchen Angabe pro Zeile, allerdings sollen alle Trennzeichen (`|`) untereinander stehen. Wie auch in Teil a) dieser Aufgabe soll der Kilometerwert mit zwei Nachkommastellen ausgegeben werden.

Aufgabe 1.5: Einführung in C: Funktionen (1+2+1+2 = 6 Punkte)

Ziel dieser Aufgabe ist es, Sie mit den verschiedenen Arten von Variablenvereinbarungen, dem Gültigkeitsbereich von Variablen und der Parameterübergabe bei Funktionsaufrufen vertraut zu machen.

Grundsätzlich unterscheidet man zwischen globalen und lokalen Variablen. Globale Variablen sind im gesamten Programm gültig. Lokale Variablen können hingegen nur in der Funktion benutzt werden, in der sie deklariert wurden. Globale und lokale Variablen können gleiche Namen haben. Bei Namenskonflikten hat die lokale Variable Vorrang.

Die Parameterübergabe an Funktionen erfolgt nach dem **call-by-value**-Prinzip. Im Funktionskopf werden Variablen deklariert. Diese erhalten bei Ausführung jeweils den Wert zugewiesen, der beim Funktionsaufruf angegeben wurde. Die Variablen können dann zwar im Körper der Funktion modifiziert werden, jedoch wird der neue Wert nicht an die aufrufende Funktion zurückgegeben. Die Parameterrückgabe muss entweder explizit über den Ergebniswert der Funktion (mittels `return`) oder implizit über Speicheradressen erfolgen. Im zweiten Fall wird nicht der Wert einer Variable, sondern die Adresse ihrer Speicherstelle an die Funktion übergeben. Alle Änderungen, die an der adressierten Speicherstelle vorgenommen werden, sind dann automatisch auch in der aufrufenden Funktion sichtbar.

a) Compilieren Sie den Quellcode `variablen.c`, den Sie zusammen mit diesem Übungsblatt im Lernraum finden. Starten Sie das Programm und erklären Sie alle Ausgaben. Wieso haben die Variablen `a` und `b` zu unterschiedlichen Zeitpunkten unterschiedliche Werte?

b) Im folgenden Liedtext hat sich ein Fehler eingeschlichen. Das letzte Zeichen einer jeden Zeile muss durch `y` ersetzt werden. Laden Sie sich den Quellcode `lyrics.c` herunter (ebenfalls im Lernraum) und fügen Sie die entsprechende Lücke in der Funktion `correct_last_character` aus.
Hinweis: Das Ende eines Strings wird durch `\0` gekennzeichnet. Die Aufgabe dient dazu den Umgang mit Zeigern zu lernen. Bitte verwenden Sie keine Indexierungen. Benutzen Sie `char *c = text+1;` anstelle von `char *c = &text[1];`.

c) Compilieren Sie nun den Quelltext zum Programm `sort.c` (ebenfalls zusammen mit dem Übungsblatt im Lernraum) und starten Sie es. Das Programm hat die Aufgabe, ein Array von Integerzahlen aufsteigend zu sortieren. Gleichzeitig soll die kleinste Zahl in der Variable `min` gespeichert werden. Finden und korrigieren Sie die Programmierfehler. Lassen Sie dabei die Funktionsvereinbarung (Signatur) von

`exchange()` unverändert. Reichen Sie als Lösung zur Aufgabe die korrigierte Version des Quellcodes ein.

Hinweis: auf einige Programmierfehler wird Sie der Compiler durch Fehlermeldungen und Warnungen hinweisen. Bitte nehmen Sie stets auch Warnungen ernst, denn auch diese können zu ernsthaften Fehlern bei der Programmausführung führen! Weitere Programmierfehler werden Sie allerdings selbst suchen müssen, da sie in der Programmlogik stecken.

d) Globale Variablen sollten so sparsam wie möglich verwendet werden, da sie unerwünschte Nebeneffekte hervorrufen können, evtl. Speicherplatz verschwenden und häufig eine Fehlerquelle darstellen. Verändern Sie den Quelltext von `sort.c` noch einmal derart, dass keine globalen Variablen mehr benötigt werden. Sie können dazu auch die Funktionsvereinbarung (Signatur) von `exchange()` modifizieren. Reichen Sie als Lösung zur Aufgabe auch hier die korrigierte Version des Quellcodes ein.

Aufgabe 1.6: C-Speicherverwaltung (1+1+1,5 = 3,5 Punkte)

a) Was ist ein Buffer Overflow? Welche Fehler können hierbei entstehen?

b) Olaf Oberklug hat ein C-Programm geschrieben, dass seinen Server vor unberechtigten Zugriff schützen soll. Wenn der Benutzer das richtige Passwort angibt, erhält er Root-Zugriff auf das System. Er hat den Quellcode `password_checker.c` ausgiebig getestet und ist mit seiner Arbeit zufrieden. Doch was hat er übersehen? Kann eine andere Person Zugriff ohne das Passwort erhalten?

c) Korrigiere Olafs Fehler! Diskutiere deinen Ansatz, um solche Fehler zu vermeiden.