Übungsblatt 1

Abgabe: 13. Mai 2021

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: Bash: Einfacher Einstieg (1 Punkt)

Geben Sie einen Kommandozeilen-Befehl an, mit dem in einer über Standard-Eingabe gelesenen Zeichenkette die erste (und nur die erste!) '1' durch eine '2' ersetzt wird. Beispiel (`mein-befehl` ist der hier zu definierende Befehl):

```
$ echo "BuS 1021: Abgabe der 1. Uebung am 13.5."| mein-befehl
BuS 2021: Abgabe der 1. Uebung am 13.5.
```

Aufgabe 1.3: Bash systemnah (0.5+2+1+1.5 = 5 Punkte)

a) Beschreiben Sie knapp, was ein Systemcall (Systemaufruf, Syscall) ist.

b) Beschreiben Sie in je einem Satz, was die folgenden vier wichtigen Syscalls tun:

`exec, ioctl, mmap, brk`

Tipp: Die zweite Section der man-Pages beschreibt Syscalls, siehe: `man man`

c) Wozu dient das Programm `strace`? Beschreiben Sie seine Funktion.

Tipp: `strace` hat eine man-page!

d) Wir wollen `strace` nutzen, um das Core Util[1] `ls` zu analysieren. Nutzen Sie dabei die Option `-C`. Da dies viel Output erzeugt, schränken wir unsere Betrachtung außerdem noch mit `-e trace=stat,lstat,fstat,open,openat` auf die Syscalls `stat`,`lstat`,`fstat` (zwei Varianten von `stat`) und `open` bzw. die Variante `openat` ein. Analysieren Sie die folgenden Kommandos:

```
ls /etc
ls -la /etc
```

Vergleichen Sie die Ausgabe von `strace` bei Anwendung auf die beiden Kommandos. Was fällt Ihnen in Bezug auf die Anzahl und Art der auftretenden Systemaufrufe auf? Erklären Sie den Sachverhalt kurz.

Aufgabe 1.4: Bash: Textverarbeitung (0.5+0.5+2+1 = 4 Punkte)

Im Folgenden wird die Textbearbeitung mittels bash betrachtet. Beantworten Sie dazu die folgenden Fragen. Alle diese Probleme können und sollen als "Einzeiler"-Shell-Skripte durch das Kombinieren verschiedener Kommandozeilenprogramme, aber ohne Schleifen und bash-Variablen gelöst werden!

a) Schreiben Sie ein Skript, welches zu allen Dateien (und Verzeichnissen) im aktuellen Verzeichnis die Dateigröße und den Dateinamen (und auch nur diese Angaben) ausgibt. Die Ausgabe braucht nicht schön formatiert zu sein, beide Angaben können einfach durch ein Leerzeichen getrennt ausgegeben werden.

b) Modifizieren Sie die Ausgabe aus dem vorherigen Aufgabenteil so, dass Dateigröße und Dateiname in umgekehrter Reihenfolge ausgegeben werden (also "Dateigröße Dateiname" wird zu "Dateiname Dateigröße").

c) Bei der Anmeldung zu den Übungen konnte man einen String als "Teamname" festlegen. Personen mit gleichem String wurden bevorzugt der gleichen Übungsgruppe zugeordnet. Im Lernraum finden Sie eine Datei `teamnamen.txt`, die in jeder Zeile einen Teamnamen enthält, wie er von einem Studierenden vergeben worden sein könnte. Schreiben Sie ein Skript, das ausgibt, wie viele Dreiergruppen sich aus den Teamnamen ergeben. Wie müssten Sie ihr Skript ändern, um die Anzahl der Zweiergruppen auszugeben? Wie viele Einergruppen (Teamname, der nur einmal auftaucht) gibt es? Wie viele Studierende haben keinen Teamnamen (Leerzeile) angegeben?

Hinweis: `uniq`

d) Schreiben Sie ein Skript, das mit einem Aufruf die Anzahl der Einer-, Zweier- und Dreiergruppen, wie sie im vorherigen Aufgabenteil definiert wurden, ausgibt. Dabei ist sowohl eine Ausgabe der Form "<Gruppengröße> <Anzahl>" (oder umgekehrt) in drei Zeilen erlaubt, als auch die reine Ausgabe der Anzahlen in aufsteigender Gruppengröße.

Hinweis: Die Anzahl der Studierenden ohne Teampräferenz ("Nullergruppen") darf ausgeben werden, muss aber nicht.

[1] `http://www.gnu.org/software/coreutils/coreutils.html`

Aufgabe 1.5: C-Datentypen (0.5+0.5+0.5+0.5+0.5+0.5 = 3 Punkte)

Gegeben seien folgende Deklarationen:

```
char zeile[] = "E  war einmal in einem Land vor unserer Zeit"; // 44 Buchstaben
char* z;
int i = 0;
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. Ungültige Operationen sollen hingegen für die weiteren Operationen ignoriert werden.

a) `zeile[1] = 's';`

b) `zeile++;`

c) `pi = &i;`

d) `*pi = 42;`

e) `&i = pi;`

f) `z = &zeile[43]-i;`

Aufgabe 1.6: 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 Grad Celsius in Fahrenheit nach der genäherten Formel

$$t[F] = \frac{9}{5} \cdot t[C] + 32$$

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 Celsius-Fahrenheit–Umrechnungstabelle ausgegeben wird. Es sollen die Temperaturen $0, 2, 4, 6, \ldots, 20$ Grad Celsius umgerechnet und tabellarisch ausgegeben werden. Hierfür soll im C-Programm eine Schleife verwendet werden. Die Tabelle kann einfach gehalten sein, es reicht eine formatierte Ausgabe `Celsiuswert | Fahrenheitwert` mit einer solchen Angabe pro Zeile, allerdings sollen alle Trennzeichen (`|`) untereinander stehen. Wie auch in Teil a) dieser Aufgabe soll der Fahrenheitwert mit zwei Nachkommastellen ausgegeben werden.

Aufgabe 1.7: Einführung in C: Funktionen (1+1+2 = 4 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) Kompilieren 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) Kompilieren Sie nun den Quellcode zum Programm `sort.c` (ebenfalls zusammen mit dem Übungsblatt im Lernraum) und starten Sie es. Das Programm hat die Aufgabe, ein Array von Integerzahlen absteigend zu sortieren. Gleichzeitig soll die größte Zahl in der Variable `max` 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. Der korrigierte Quellcode darf beim Kompilieren keine Fehler oder Warnungen ausgeben (mit `gcc -Wall`).

c) 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 Quellcode 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, die ebenfalls beim Kompilieren keine Fehler oder Warnungen ausgeben darf (mit `gcc -Wall`).