Allgemeine Hinweise zu den BS-Übungen

- Die Aufgaben sind *in Dreiergruppen* zu bearbeiten (Aufgabe 0 in Ausnahmefällen noch allein oder zu zweit). Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die Gruppenmitglieder sollten gemeinsam an der gleichen Tafelübung teilnehmen. Die Lösung wird jeweils komplett bewertet und den Gruppenmitgliedern gleichermaßen angerechnet.
- Die Übungsaufgaben müssen abhängig von der Tafelübung entweder bis zum Donnerstag bevor das nächste Blatt erscheint (Übungsgruppen in ungeraden Kalenderwochen) oder Dienstag nachdem das nächste Blatt erschienen ist (Übungsgruppen in geraden Kalenderwochen) jeweils bis 10 Uhr abgegeben werden. In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch eine Musterlösung.
- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die Zusatzaufgaben sind ein Stück schwerer als die „normalen“ Aufgaben und geben zusätzliche Punkte.
- Die Aufgaben sind über AsSESS (`https://ess.cs.tu-dortmund.de/ASSESS/`) abzugeben. Dort gibt *ein* Gruppenmitglied die erforderlichen Dateien ab und nennt dabei die anderen beteiligten Gruppenmitglieder (Matrikelnummer, Vor- und Nachname erforderlich!). Namen und Anzahl der abzugebenden C-Quellcodedateien[2] variieren und stehen in der jeweiligen Aufgabenstellung; Theoriefragen sind grundsätzlich in der Datei `antworten.txt`[3] zu beantworten. Bis zum Abgabetermin kann eine Aufgabe beliebig oft abgegeben werden – es gilt die letzte, vor dem Abgabetermin vorgenommene Abgabe.
- Sobald eine Abgabe von den Betreuern korrigiert wurde, kann die korrigierte Lösung ebenfalls im AsSESS eingesehen werden.

Aufgabe 0: Erste Schritte in C (10 Punkte)

Das Lernziel dieser Aufgabe ist der Umgang mit der UNIX-Systemumgebung und dem C-Compiler. Darüber hinaus sollt ihr euch durch das Schreiben eines einfachen C-Programms mit dieser Programmiersprache vertraut machen.

Theoriefragen: Systemumgebung (5 Punkte)

Macht euch zunächst mit der Systemumgebung – im IRB-Pool (am besten in der Rechnerübung!), in der auf der Veranstaltungswebseite zur Verfügung gestellten virtuellen Maschine oder in einer eigenen Linux-Installation zu Hause – vertraut. Öffnet ein Terminal-Fenster und experimentiert mit den in der Tafelübung vorgestellten UNIX-Kommandos.

1. Erklärt in eigenen Worten, welche Rolle die Umgebungsvariable **`PATH`** spielt.
2. Erklärt in eigenen Worten, wofür man das UNIX-Kommando `man` verwendet. Erklärt den Unterschied zwischen den Befehlen **`man 1 printf`** und **`man 3 printf`** und was hier beschrieben wird.

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.
[2] codiert in UTF-8
[3] reine Textdatei, codiert in UTF-8

3. Gebt ein Kommando an, um im aktuellen Verzeichnis alle Dateien nach dem Text „todo“ zu durchsuchen. Dabei soll jedes Vorkommen mit der Zeilennummer auf der Konsole ausgegeben werden.
4. Gebt den Befehl an, um die Source-Datei **`datei.c`** mit dem gcc zu kompilieren und dadurch die ausführbare Datei **`exec.out`** zu erzeugen.
5. Mit welchem UNIX-Kommando kann man Dateien und Ordner umbenennen? Für welchen Zweck kann man dieses Kommando noch verwenden?

Theoriefragen (Fortsetzung): Variablen in C (2 Punkte)

Betrachtet das folgende C-Programm:

```
#include <stdio.h>
#include <math.h>

const double G = 6.67384E-11; /* Gravitationskonstante */
double mErde = 5.972E24; /* Masse Erde */
double distanz;

/* Berechnet Kraft zwischen zwei Objekten */
double force(double mass1, double mass2, double distance) {
   double d2 = pow(distance, 2);
   return G * mass1 * mass2 / d2;
}

int main(void) {
   distanz = 3.844E8; /* Distanz Erde zu Mond */
   double f = force(mErde, 7.349E22, distanz); /* Kraft Erde zu Mond */
   printf("Kraft Erde zu Mond: %e\n", f);
   return 0;
}
```

- In welchen Bereichen des Speicherlayouts (Segmente) befinden sich die Funktionen `main()` und `force()`, die Variablen `G`, `mErde`, `distanz`, `d2`, und `f`, sowie die Parameter der Funktion `force()`?
- Warum befinden sich die Variablen `mErde` und `distanz` in unterschiedlichen Segmenten?

Tipp: Wollt ihr obigen Code compilieren, müsst ihr noch das Argument „`-lm`“ (der erste Buchstabe ist ein kleines L, kein großes i) anfügen, damit gegen die math-Bibliothek gelinkt wird. Das Compilieren ist jedoch für diese Aufgabe nicht notwendig.

Programmierung in C (3 Punkte)

Manche natürlichen Zahlen lassen sich als Summe zweier Quadratzahlen aufschreiben, z.B. gilt $17 = 1^2 + 4^2$. Schreibt ein Programm, welches alle solche Zahlen ≤ 100 übersichtlich auf der Konsole ausgibt. Dabei sollen jeweils die Zahl als auch die zwei Basen ausgegeben werden. Als Einschränkung darf jede Zahl nur einmal ausgegeben werden und es interessieren uns nur Zahlen mit zwei unterschiedlichen Basen > 0. Die kleinste gesuchte Zahl ist demnach $5 = 1^2 + 2^2$.

Die Ausgabe soll mittels **`printf(3)`** erfolgen.

Ein Beispielaufruf inkl. Ausgabe könnte folgendermaßen aussehen:

```
titan@Unidesk:~$ ./qzahlen
5: 1, 2
10: 1, 3
13: 2, 3
...
```

Die Implementierung soll in der Datei qzahlen.c abgegeben werden.

Zusatzaufgabe 0: Programmierung in C - Extended (2 Sonderpunkte)

Erweitert eure Implementierung, so dass es möglich ist, eine gewünschte Obergrenze als Argument an eure main-Funktion zu übergeben. An dieser Stelle soll eine einfache Fehlerbehandlung Eingaben < 1 und den Fall abfangen, dass nicht genau ein Argument übergeben wurde. Die Fehlerbehandlung kann an dieser Stelle durch eine Ausgabe mit **printf(3)** geschehen.

Schaut euch für die Bearbeitung dieser Aufgabe die Bedeutung von argc und argv sowie die Funktion **atoi(3)** genauer an.

Die veränderte Version soll als qzahlen_extended.c abgegeben werden.

Beispiele für Programmaufrufe:

```
titan@Unidesk:~$ ./qzahlen_extended 5
5: 1, 2

titan@Unidesk:~$ ./qzahlen_extended 18
5: 1, 2
10: 1, 3
13: 2, 3
17: 1, 4

titan@Unidesk:~$ ./qzahlen_extended -23
Fehler: Negative Zahl als Obergrenze angegeben

titan@Unidesk:~$ ./qzahlen_extended
Fehler: Keine Obergrenze angegeben
```

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Die Programme sollen sich mit dem gcc auf den Linux-Rechnern im IRB-Pool übersetzen lassen. Der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `gcc -std=c11 -Wall -o qzahlen qzahlen.c -lm`
  Alternativ könnt ihr die Programme auch in C++ schreiben, der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `g++ -Wall -o qzahlen qzahlen.c`
  Weitere (nicht zwingend zu verwendende) Compilerflags, die dafür sorgen, dass man sich näher an die Standards hält, sind: `-Wpedantic -Werror`

Abgabe: bis spätestens Donnerstag, den 18. April 10:00 (Übung in ungerader Kalenderwoche), bzw. Dienstag, den 23. April 10:00 (Übung in gerader Kalenderwoche).