Abgabefrist Übungsaufgabe 0

Die Lösung muss abhängig von der Tafelübung abgegeben werden, an der ihr teilnehmt:

- **W1** – Übung U0 in KW16 (15.04. - 18.04.): Abgabe bis **Donnerstag, den 25.04.2023 10:00**
- **W2** – Übung U0 in KW17 (22.04. - 25.04.): Abgabe bis **Montag, den 29.04.2023 10:00**

In darauffolgenden Tafelübungen werden teilweise einzelne abgegebene Lösungen besprochen, teilweise auch ein Lösungsvorschlag aus dem Tutorenteam.

Allgemeine Hinweise zu den BS-Übungen

- Die Aufgaben sind *in Dreiergruppen* zu bearbeiten. Der Lösungsweg und die Programmierung sind gemeinsam zu erarbeiten.
- Die Gruppenmitglieder **sollten nach Möglichkeit** gemeinsam an der gleichen Tafelübung teilnehmen. Es sind aber auch Abgaben mit Studierenden aus **verschiedenen Übungsgruppen** zulässig. Es gilt dabei immer die **früheste Abgabefrist**. Die Lösung wird jeweils komplett bewertet und den Gruppenmitgliedern gleichermaßen angerechnet.
- 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 korrigiert wurde, kann die korrigierte Lösung ebenfalls in AsSESS eingesehen werden.

[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

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 ersten C-Programms mit der Programmiersprache *C* vertraut machen.

0.1 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. Was macht der Befehl "`rm -rf /`"? **(/!\ nicht ausführen /!\)** Erklärt die einzelnen Optionen und Parameter.
2. Erklärt den Unterschied zwischen den Befehlen `man 1 kill` und `man 3 kill`
3. Gebt eine Möglichkeit an, die Anzahl der Zeilen in der Datei `datei.txt`, in denen das Wort „quiesel“ vorkommt, auf der UNIX-Konsole auszugeben. Verwendet dazu nur einen einzigen Befehl.
4. Wozu braucht man beim Kompilieren mit dem gcc die Option `-o`?
5. Gebt einen kürzeren Befehl (sprich: weniger Zeichen) an, aus dem Verzeichnis `~/foo/bar` in das Verzeichnis `~/foo` zu wechseln, als `cd ~/foo`. Was bewirkt hier „~“?

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

Betrachtet das folgende C-Programm:

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

double luftreibung;
double g = 9.81;  // Gravitationskonstante
double luftdichte = 1.225;  // Luft bei 15 Grad

double endgeschwindigkeit(double masse, double flaeche) {
    /* Berechne die maximale Fallgeschwindigkeit eines Objektes */
    return sqrt((2 * masse * g) / (luftdichte * flaeche * luftreibung));
}

int main(void) {
    luftreibung = 0.47;  // Koeffizient einer Kugel

    double v;  // Endgeschwindigkeit
    v = endgeschwindigkeit(5.0, 0.5);

    printf("%lf\n", v);  // Ausgabe
    return 0;
}
```

1. In welchen Bereichen des Speicherlayouts (Segmente) befinden sich die Funktion `endgeschwindigkeit()`, die Variablen `luftreibung`, `luftdichte`, `g` und `v`, sowie die Parameter der Funktion `endgeschwindigkeit()`?
2. Warum befinden sich die Variablen `luftreibung` und `luftdichte` 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.

0.3 Programmierung in C (3 Punkte)

Wir wollen an dieser Stelle ein Programm implementieren, welches alle Primzahlen ≤ 20 berechnet und diese anschließend in der Unix-Konsole ausgibt.

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

Ein Beispielaufruf sähe ungefähr folgendermaßen aus:

```
ab@mars:~$ ./prim
2,3,5,7,11,13,17,19,
```

Tipp: Eine Zahl ist eine Primzahl, wenn es keine kleinere Zahl (größer als 1) gibt, durch die sie ohne Rest teilbar ist. Ihr benötigt zum Lösen dieser Aufgabe *keine* Arrays.

Die Implementierung soll in der Datei `prim.c` abgegeben werden.

Zusatzaufgabe 0: Kommandozeilenparameter (1 Punkte)

Erweitert eure Implementierung so weit, dass es möglich ist, eine gewünschte Obergrenze für Primzahlen als Argument an eure Gammafunktion zu übergeben. An dieser Stelle müsst ihr keine Fehlerbehandlung implementieren, außer für den Fall, dass nicht genau ein Argument übergeben wurde. Diese kann an dieser Stelle durch **`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 `prim_extra.c` abgegeben werden.

Beispiel für Programmaufrufe:

```
ab@mars:~$ ./prim_extra 5
2,3,5,

ab@mars:~$ ./prim_extra 40
2,3,5,7,11,13,17,19,23,29,31,37,

ab@mars:~$ ./prim_extra
Fehler: Keine Obergrenze für Primzahlen übergeben
```

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 prim prim.c`
  Falls sie zur Entwicklung stattdessen einen C++ Compiler nutzen möchten, achten sie bitte darauf, dass ihre Abgabe nur C Standards nutzt.
  Weitere (nicht zwingend zu verwendende) Compilerflags, die dafür sorgen, dass man sich näher an die Standards hält, sind: `-Wpedantic -Werror`