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 nach Möglichkeit** gemeinsam an der gleichen Tafelübung teilnehmen. Es sind aber auch Abgaben mit Studierenden aus **verschiedenen Übungsgruppen** zulässig. 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 geraden Kalenderwochen) oder Dienstag nachdem das nächste Blatt erschienen ist (Übungsgruppen in ungeraden Kalenderwochen) jeweils bis 08 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 BS-Entwicklungsumgebung[4] vertraut. Öffnet ein Terminal-Fenster und experimentiert mit den in der Tafelübung vorgestellten UNIX-Kommandos.

1. Mit welchem Parameter listet `ls` auch Dateien in Unterverzeichnissen? [5]
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
[4] Weitere Informationen: https://ess.cs.tu-dortmund.de/DE/Teaching/SS2020/BS/Downloads/
[5] Lest im Zweifelsfall in der manpage nach: `man ls`

3. Mit welchem UNIX-Kommando kann man Dateien löschen? Mit welchem Parameter löscht man einen kompletten Verzeichnisbaum?

(Fortsetzung der Theoriefragen am Ende der Programmieraufgabe)

Programmierung in C (5 Punkte)

Schreibt ein einfaches C-Programm (`rek-sum.c`), das in einer *rekursiven* Funktion `int sum_n(int n)` die Summe der Zahlen von 1 bis *n* berechnet. Ein Aufruf `sum_n(5)` soll dementsprechend 15 zurückgegeben. Der Rückgabewert soll anschließend ausgegeben werden, so dass folgende Ausgabe erscheint: `Summe von 1 bis 5: 15`. Die Ausgabe soll mittels **`printf(3)`** erfolgen. Schaut euch auch die Folien zur C-Einführung an.

Legt darüber hinaus eine Reihe von unterschiedlichen Variablen (verschiedene Typen; global, lokal; initialisiert, uninitialisiert) an und lasst deren Adresse im Hauptspeicher in der `main()` ausgeben. Die Adresse könnt ihr mit **`printf(3)`** ausgeben: `printf("Adresse von foo ist %p\n", &foo);`

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

1. Probiert verschiedene (auch große!) Werte für den Parameter *n aus.* Warum läuft das Programm ab einem bestimmten Wert nicht mehr bis zum Ende? Wie groß ist *n* in eurem Fall?
2. Welchen Abstand (in Bytes) haben die Adressen zweier nacheinander in `main()` angelegter Variablen vom Typ double? Erklärt außerdem, wie dieser Abstand zustande kommt.
3. Warum liegt eine globale `int`-Variable an einer völlig anderen Adresse?
4. Warum wird die Adresse einer lokalen Variablen in der rekursiven Funktion *immer kleiner*, wenn die Funktion immer weiter „rekursiv absteigt“?

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 mit einem Wert < 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. Bedenkt bitte auch, dass die Ausgabe, wie sie in der vorherigen Aufgabe vorgegeben ist, angepasst werden muss, da die Obergrenze der Summe jetzt variabel ist.

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 `rek_sum_extended.c` abgegeben werden.

Beispiele für Programmaufrufe:

```
al@ios:~$ ./rek_sum_extended 5
Summe von 1 bis 5: 15

al@ios:~$ ./rek_sum_extended 20
Summe von 1 bis 20: 210

al@ios:~$ ./rek_sum_extended -23
Fehler: Negative Zahl als Obergrenze angegeben

al@ios:~$ ./rek_sum_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 rek_sum rek_sum.c`
  Alternativ *könnt* ihr die Programme auch in C++ schreiben, der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `g++ -Wall -o rek_sum rek_sum.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 07. Mai 08:00 (Übung in gerader Kalenderwoche), bzw. Dienstag, den 12. Mai 08:00 (Übung in ungerader Kalenderwoche)