Abgabefrist Übungsaufgabe 0

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

- **W1** – eigene Übung U0 am 20./21.04.: Abgabe bis **Donnerstag, den 29.04.2021 08:00**
- **W2** – eigene Übung U0 am 27./28.04.: Abgabe bis **Dienstag, den 04.05.2021 08:00**

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 (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. 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.

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.

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] 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://sys.cs.tu-dortmund.de/Teaching/SS2021/BS/Uebung/materialien.html`

1. Mit welchem Parameter zeigt `ls` den Inhalt eines Verzeichnisses im Listenformat an?[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 in den beiden Manpages beschrieben wird.
3. Mit welchem UNIX-Kommando kann man Dateien und Ordner umbenennen? Für welchen Zweck kann man dieses Kommando noch verwenden?

(Fortsetzung der Theoriefragen am Ende der Programmieraufgabe)

Programmierung in C (5 Punkte)

Schreibt ein C-Programm `fak.c`, das in einer *rekursiven* Funktion `int fak(int n)` die *n*-te Fakultät berechnet. Ein Aufruf `fak(7)` soll dementsprechend 5040 zurückgegeben. Der Rückgabewert soll anschließend ausgegeben werden, so dass folgende Ausgabe erscheint:

```
Die Fakultät von 7 lautet: 5040.
```

Die Ausgabe soll mittels **`printf(3)`** erfolgen. Die Folien zur C-Einführung können euch bei der Umsetzung helfen.

Legt darüber hinaus eine Reihe von unterschiedlichen Variablen an:

- verschiedene Datentypen
- global initialisierte Variablen
- global uninitialisierte Variablen
- lokale Variablen

Lasst die Adressen dieser Variablen im Hauptspeicher in der `main()` ausgeben. Die Adresse kann mit **`printf(3)`** ausgegeben werden:

```
printf("Adresse von foo ist %p\n", &foo);
```

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

1. Probiert verschiedene Werte (mehrere Zehner bis hin zu mehreren Tausend) für den Parameter *n* aus:
   a) Warum stimmt das Ergebnis ab einem bestimmten Wert für *n* nicht mehr? Wie groß ist *n* in eurem Fall?
   b) Warum läuft das Programm ab einem bestimmten Wert nicht mehr bis zum Ende? Wie groß ist *n* in eurem Fall?
2. Warum wird die Adresse einer lokalen Variablen in der rekursiven Funktion *immer kleiner*, wenn die Funktion immer weiter „rekursiv absteigt“?
3. Warum liegt eine globale `int`-Variable an einer völlig anderen Adresse?

[5] Lest im Zweifelsfall in der manpage nach: **`man ls`**

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

Erweitert eure Implementierung so, dass es möglich ist, die Zahl *n* als Kommandozeilenparameter an euer Programm 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 Nummer der Fakultät jetzt variabel ist.

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

Die erweiterte Version soll als `fak_extended.c` abgegeben werden.

Beispiele für Programmaufrufe:

```
al@ios:~$ ./fak_extended 5
Die Fakultät von 5 lautet: 120

al@ios:~$ ./fak_extended 7
Die Fakultät von 7 lautet: 5040

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

al@ios:~$ ./fak_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 fak fak.c`
  Alternativ *könnt* ihr die Programme auch in C++ schreiben, der Compiler ist dazu z.B. mit folgenden Parametern aufzurufen:
  `g++ -Wall -o fak fak.c`
  Weitere (nicht zwingend zu verwendende) Compilerflags, die dafür sorgen, dass man sich näher an die Standards hält, sind: `-Wpedantic -Werror`