Allgemeine Informationen

Die Unterlagen sowie alle Aufgaben und Hilfen finden Sie auf der Web-Seite des C-Vorkurses.

`https://https://precampus.uni-bonn.de/`

Wer sich mit `git` auskennen, kann gerne das Repo clonen.

Erklärung des Schwierigkeitsgrad:

▲ leicht
▲▲ mittel
▲▲▲ schwer

Virtuelle Maschine

Wir stellen für den Kurs eine virtuelle Maschine (`VirtualBox`) zur Verfügung. Um diese zu nutzen, müssen Sie VirtualBox aus dem Internet herunterladen oder vom USB-Stick kopieren und installieren. Auf der virtuellen Maschine sind alle wichtigen Programme installiert, die Sie benötigen um gut in den C-Vorkurs zu starten. Auf der virtuellen Maschine befindet sich ein Linux System, da wir in unserem Vorkurs die Beispiele und Aufgaben mit Linux demonstrieren.

Zum installieren der `VirtualBox` einfach den Anweisungen folgen. Danach muss die virtuelle Maschine eingebunden werden. Über

`Gruppe->Maschine hinzufügen`

kann dann die gewünschte Maschine erstellt werden. Ein Doppelklick auf das Symbol links startet die VM.

Aufgabe 1. ▲ Getting started!

Starten Sie die virtuelle Maschine mit VirtualBox. Auf dem Desktop angekommen, drücken Sie die Windows-Taste und geben Sie anschließend *Terminal* ein bestätigen Sie mit Enter. Vor sich sehen Sie nun ein Linux Terminal, die universelle Schnittstelle zu dem Linuxsystem.

Das Prinzip dieses Terminals ist einfach wie effektiv: Sie geben ein Kommando ein, gefolgt von Argumenten und Optionen und bestätigen mit Enter. Geben Sie zum Beispiel *pwd* ein, um zu sehen in welchem Ordner Sie sich gerade befinden. Sie sollten sich im sogenannten home-Directory befinden, das Linuxpendant zu Eigene Dateien auf Windows.
Ihre Aufgabe besteht nun darin, ein Hallo-Welt Programm zu erstellen, zu kompilieren und auszuführen. Der dazu nötige Quellcode hier:

```c
#include <stdio.h>
int main(void) {
  puts("Hallo,Welt!");
  return 0;
}
```

Tipp: Eine Liste mit den wichtigen Kommandozeilenbefehlen finden Sie im Github-Repository. Als kleine Übersicht hilft die folgende Tabelle:

Befehl	**Effekt**
ls	Listet den Inhalt des derzeitigen Verzeichnisses auf.
mkdir <name>	Erstellt einen Ordner mit dem angegebenen Namen
cd <ordner>	Wechselt in den angegebenen Ordner
cp [-rf] <quelle><ziel>	Kopiert die Datei / Ordner von der Quelle zum Ziel
mv [-rf] <quelle><ziel>	Verschiebt die Datei / Ordner von der Quelle zum Ziel
rm [-rf] <ordner/datei>	Löscht eine Datei / einen Ordner
clang -o <ausgabedatei> <quelldatei>	Kompiliert eine Quelltext Datei

Aufgabe 2. ▲ ▲ Was machen folgende vier Algorithmen?

Algorithmisch Denken

Als kleine Einstimmung auf die kommenden zwei Wochen und um mit dem algorithmischen Denken warm zu werden, sind im folgenden vier Algorithmen in Pseudocode[1] aufgelistet. Ihre Aufgabe ist es herauszufinden, was diese Algorithmen tun.

Algorithmus 1

Eingabe ist eine ganze Zahl $c \in \mathbb{N}$. Ausgabe ist entweder »Ja« oder »Nein«. Schritte:

- Setze n auf 2.
- Wiederhole ...
  - Falls $n > \sqrt{c}$, so gib »Ja« aus und beende das Programm.
  - Falls n ein Teiler von c ist, so gib »Nein« aus und beende das Programm.
  - Erhöhe n um eins.

Algorithmus 2

Eingabe sind die ganzen Zahlen $a, b \in \mathbb{N}$. Ausgabe ist eine ganze Zahl $k \in \mathbb{N}$. Schritte:

- Wiederhole ...
  - Falls $a = 0$ ist, gib b aus und beende das Programm.
  - Falls $b = 0$ ist, gib a aus und beende das Programm.
  - Falls $a > b$ ist, setze a auf $a - b$.
  - Falls obige Bedingung nicht erfüllt ist, setze b auf $b - a$.

[1] siehe wikipedia.org/wiki/Pseudocode

Algorithmus 3

Eingabe ist eine reelle nicht-negative Zahl $a \in \mathbb{R}_0^+$. Ausgabe ist eine reelle Zahl $x \in \mathbb{R}$.
Schritte:

- Setze x auf 2 und y auf 1.
- Wiederhole ...
  - Falls $|x - y| \leq 10^{-10}$ ist, gib x aus und beende das Programm.
  - Setze x auf y.
  - Setze y auf $\frac{1}{2}(x + \frac{a}{x})$.

Algorithmus 4

Eingabe ist eine Zahl $z \in \mathbb{Z}$ und eine Zahl $a \in \mathbb{N}$. Ausgabe ist eine ganze Zahl $k \in \mathbb{Z}$.
Schritte:

- Sei $k = 0$.
- Falls $a = 0$ gib 0 zurück.
- Solange $a > 1$:
  - Falls a ungerade ist, erniedrige a um eins und addiere z zu k.
  - Ansonsten halbiere a und verdopple z.
- Gebe $k + z$ zurück.

Aufgabe 3. ▲ Schreibe ein Programm, dass den Wert der folgenden Funktion ausgibt (für eine fest in den Quellcode geschriebene `int`-Variable):

$$f(n) = \begin{cases} \frac{n}{2} & \text{wenn } n \text{ gerade} \\ \frac{n+1}{2} & \text{wenn } n \text{ ungerade} \end{cases}$$

Und das geht natürlich nur mit Wissen aus der Vorlesung.

Aufgabe 4. ▲ ▲ Wettbewerb: Gegeben ist folgender Programmrumpf:

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      int x = 2;
4      /* dein Code hier */
5      printf("%i\n", x);
6      return 0;
7  }
```

Füge an der markierten Stelle C-Code ein, sodass der Wert von $2^{\left(3^3\right)}$ ausgegeben wird. Erlaubt sind aber nur die Zeichen

```
x   +   -   *   /   =   ;
```

Versuche durch geschicktes Multiplizieren und Nutzung von Variablen mit möglichst wenig Zeichen auszukommen. Zeilenumbrüche und Leerzeichen können nach belieben verwendet werden, da sie vom Compiler ignoriert werden.

Diese Art ein Programm zu schreiben nennt man »Code Golf« und ist ein witziger Zeitvertreib.

Aufgabe 5. ▲ Finde die Fehler:

Umgang mit Compiler-Fehlern

Als Anfänger ist es besonders wichtig die verschiedenen Compiler-Fehler zu interpretieren. Zu diesem Zweck ist das folgende Programmbeispiel gelistet. Das Programm hat insgesamt 5 Fehler. Dazu bitte den Programmcode in den Editor kopieren und mit `clang / gcc` compilieren.

```
1         include <stdlib.h>
2         #include <stdio.h>
3
4         int do-addition(int a, int b)
5             return a+b;
6         }
7
8         int do-subtraction(int a, int b){
9             return a-b
10        }
11
12        int main(int argc, char **argv) {
13            int a = 1;
14            int b = 2;
15
16            sum = do-addition(a,b);
17            int erg = do-subsubsection(a,a;
18            return 0;
19        }
```

Es wird ein Teil der Aufgaben immer den darauf folgenden Tag erklärt.