

Aufgabe 1. Installiere einen Compiler auf deinem Computer und kompiliere ein Hallo-Welt-Programm. Unter Linux geht dies einfach mit dem Paketmanager. Wir empfehlen dringend, entweder Linux in einer virtuellen Maschine zu nutzen oder direkt auf dem eigenen Rechner zu installieren.

Wenn dies nicht möglich ist, kann man unter Windows auch Cygwin nutzen. Informationen dazu und einen Download-Link für Cygwin findest du auf der Webseite des C-Kurses von Lars Wallenborn und Jesko Hüttenhain:

<http://ah-effect.net/wp-content/uploads/2016/09/cygwin.zip>

Für Mac OS bietet Apple mit XCode die notwendigen Entwicklungstools an, wie z.B. hier beschrieben:

<https://www.proggen.org/doku.php?id=c:compiler:macos>

Es ist vorteilhaft, den so installierten Compiler zunächst in der Konsole zu nutzen und die Programme erstmal mit einem einfachen Texteditor zu verfassen.

Aufgabe 2. In dem folgenden Hallo-Welt-Programm befinden sich 4 Fehler. Finde sie alle.

```
1  * Hello World Program.
2  * (c) 2015 Clelia und Johannes */
3
4  #include <stdio.h>
5
6  double main () {
7      pritnf ("Hallo Welt\n")
8      return 0;
9  }
```

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

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

Aufgabe 4. Wenn man sich überlegt, wie man eine numerische Aufgabe in einem Programm umsetzen möchte, ist es oft hilfreich, zunächst sogenannten Pseudocode zu schreiben. Dabei abstrahiert man weg von der Programmiersprache, in der man die Lösung implementiert und überlegt sich, welche elementaren Schritte notwendig sind, um das Problem zu lösen.

Zunächst sollte man sich im Klaren sein, welchen Input man an das Programm übergibt. Auch sollte man wissen, welchen Output man erwartet. Schlussendlich benötigt man eine kompakte Abbildung der Kontrollstrukturen des verwendeten Programmierparadigmas. In unserem Fall handelt es sich um eine *imperative*, *strukturierte* und *prozedurale* Programmiersprache, in der Zustände in Variablen gespeichert werden und mithilfe von Kontrollstrukturen der Programmfluss gesteuert wird.

Diesen abstrakten Pseudocode versteht man am besten anhand eines Beispiels. Es gibt leider keine wirklichen Standards, was die Syntax von Pseudocode angeht. Wir verwenden daher die gleiche Syntax, wie sie auch in den Journalen des IEEE genutzt wird.

Was machen folgende Algorithmen?

Algorithmus 1

Input: Ganze Zahl $c \in \mathbb{N}$

Output: Entweder Ja oder Nein.

```
1: set  $n := 2$ .
2: if  $n > \sqrt{c}$  then
3:   return Ja
4: end if
5: if  $n$  teilt  $c$  then
6:   return Nein
7: end if
8: set  $n := n + 1$ 
9: goto 2
```

Algorithmus 2

Input: Ganze Zahlen $a, b \in \mathbb{N}$

Output: Eine ganze Zahl $k \in \mathbb{N}$

```
1: if  $a = 0$  then
2:   return  $b$ 
3: end if
4: if  $b = 0$  then
5:   return  $a$ 
6: end if
7: if  $a > b$  then
8:   set  $a = a - b$ 
9: else
10:  set  $b = b - a$ 
11: end if
12: goto 4
```

Algorithmus 3

Input: Reelle Zahl $a \in \mathbb{R}_{\geq 0}$

Output: Reine reelle Zahl $x \in \mathbb{R}$

```
1: set  $x := 2$  und  $y := 1$ .
2: if  $|x - y| \leq 10^{-10}$  then
3:   return  $x$ 
4: end if
5: set  $x := y$ 
6: set  $y := \frac{1}{2} \cdot (x + \frac{a}{x})$ 
7: goto 2
```

Aufgabe 5. 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 $2^{(3^3)}$ ausgegeben wird. Erlaubt sind aber nur folgende Zeichen:

x + - * / = () ;

Zeilenumbrüche und Leerzeichen sollen genutzt werden, um das Programm möglichst übersichtlich zu gestalten. Die Benutzung von Klammern kann ebenfalls genutzt werden, um die Lesbarkeit zu erhöhen.