Lösungen zur Übung 1

16. März 2018

1 Aufgabe 1

```
 * Hello World Program.
 * (c) 2015 Clelia und Johannes */

#include <stdio.h>

double main () {
    pritnf ("Hallo Welt\n")
    return 0;
}
```

Um Fehler einfach zu erkennen, kann man zunächst versuchen die Quelltextdatei zu kompilieren. Das bedeutet, dass man zuerst den oberen Text in eine Datei schreibt (name *beispiel*1.*c*) und versucht, mit

```
gcc -Wall -Wpedantic -std=c99 beispiel1.c -o beispiel1.exe
```

zu übersetzten. In diesem Fall werden sehr viele Fehler ausgegeben. Die folgende Nachricht ist die erste von mehreren Fehlermeldungen:

```
hello_world.c:1:10: error: expected '=', ',', ';', 'asm' or '__attribute__' before '
    World'
  * Hello World Program.
```

Praktisch müssen wir uns zuerst um die erste Fehlermeldung kümmern. Die Fehler sind meistens nicht unabhängig voneinander: Wenn wir den ersten Fehler korrigiert haben, kann es sein, dass nachfolgende Fehler gleich mit korrigiert wurden. Das passiert auch in unserem Beispiel. Laut der Fehlermeldung ist etwas in der ersten Zeile schief gegangen. Der Kommentar in der ersten Zeile sollte mit `/*` beginnen. Danach sieht unser Beispiel so aus:

```
/* Hello World Program.
 * (c) 2015 Clelia und Johannes */

#include <stdio.h>

double main () {
```

```
7     pritnf ("Hallo Welt\n")
8     return 0;
9 }
```

Wenn wir jetzt den Quelltext übersetzen, haben wir nur noch ein paar Fehlermeldungen. Die nächste Fehlermeldung ist:

```
hello_world.c: In function 'main':
hello_world.c:7:5: warning: implicit declaration of function 'pritnf' [-Wimplicit-
    function-declaration]
     pritnf ("Hallo Welt\n")
     ^
hello_world.c:8:5: error: expected ';' before 'return'
     return 0;
```

Wir bemerken jetzt dass der C kompiler zwei verschiedenen Typen von Fehlermeldungen gibt:

- Errors: Mit diesem Typ von Fehlermeldungen kann unsere Quelldatei nicht übersetzt werden. Wir müssen alle *Errors* korrigieren.
- Warnings: Diese sind nur Warnungen. Das Programm kann ausgeführt werden. Aber Achtung! Wir müssen auch Warnungen beachten. Warnungen teilen uns mit, dass das Programm unter Umständen nicht korrekt funktionieren wird.

Laut der Warnung, kann der Compiler die Funktionsdeklaration von `pritnf` nicht finden. Der Tippfehler ist schnell korrigiert, der richtige Name der Funktion lautet:

`printf();`

und unser Programm sieht jetzt so aus:

```
/* Hello World Program.
 * (c) 2015 Clelia und Johannes */

#include <stdio.h>

double main () {
    printf ("Hallo Welt\n")
    return 0;
}
```

Jetzt verbleibt nur noch zwei Fehlermeldungen:

```
hello_world.c:8:5: error: expected ';' before 'return'
     return 0;
```

Das ist auch ein typischer Fehler. Jeder Befehl muss mit einem ";"beendet werden.

Des weiteren muss der Rückgabetyp der `main` Funktion eines Programms eine ganze Zahl sein, also

```c
/* Hello World Program.
 * (c) 2015 Clelia und Johannes */

#include <stdio.h>

int main () {
    printf ("Hallo Welt\n");
    return 0;
}
```

2 Aufgabe 3

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

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

Die Funktion f hat ein Argument (n) und einen Rückgabewert $(f(n))$. Zuerst müssen die Datentypen festgelegt werden. Laut Anweisung sind sowohl die Eingabe, als auch die Ausgabe ganzzahlig. Somit ist die Funktiondeklaration die folgende:

```c
int f(int );
```

und eine mögliche Implementation sieht wie folgt aus:

```c
#include<stdio.h>
int f(int n){
   if ( n % 2 == 0 )  // n gerade
     return n/2;
   else                // n ungerade
     return (n+1)/2;
}
int main(){
   int n=3;
   int fn=f(n);
   printf("n=%d\tf(n)=%d\n", n, fn);
   printf("n=%d\tf(n)=%d\n", 4, fn(4));
}
```

fn.c

In der 10-ten Zeile rufen wir die Funktion auf und weisen den Rückgabewert der Variable `fn` zu, danach geben wir sowohl den Wert von `n` als auch den Funktionswert aus. In der darauffolgenden Zeile sieht man, dass wir die Funktion auch direkt innerhalb der Argumentenliste von `printf` aufrufen können.

3 Aufgabe 4

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

Man muss zuerst die Argumente und den Rückgabewert betrachten. Das Argument ist eine ganze Zahl, $c > 0$, und der Rückgabewert ist entweder Ja(1) oder Nein(0). Der Algorithmus überprüft eine Eigenschaft der natürlichen Zahlen.

In der ersten Zeile initialisieren wir eine Variable (in einem C-Quelltext wäre diese vom Datentyp *int*) mit dem Wert 2. Danach überprüfen wir ob n (im ersten Fall $n = 2$) größer als $\sqrt{c}$ ist oder nicht. Für die Eingaben:

$$c = 1, 2, 3 \tag{1}$$

ist der Rückgabewert "Ja".

Ist dies nicht der Fall, läuft der Algorithmus in Zeile 5 damit weiter, dass überprüft wird, ob c duch n geteilt werden kann. Wenn ja, dann ist der Rückgabewert "Nein".

Trifft keine der Bedinungen zu, wird der Zähler n um 1 inkrementiert und der Algorithmus läuft in Zeile 2 wieder an. Jetzt ist es klar, dass es sich hierbei um einen Primzahlentest handelt.

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

Dieser Algorithmus erhält als Eingabe zwei ganze Zahlen, gibt eine dritte Zahl zurück.

Beispielhaft setzen wir $a = 12$, $b = 9$. Nur die Bedingung in Zeile 7 trifft zu, und a wird auf $12 - 9 = 3$ gesetzt. In der folgenden Iteration greift die zweite Bedingung und es wird $b = 9 - 3 = 6$ gesetzt. Der Algorithmus iteriert weiter, bis eine der beiden Zahlen Null ist und gibt die andere Zahl zurück:

1. $a = 12, b = 9$
2. $a = 3, b = 9$
3. $a = 3, b = 6$
4. $a = 3, b = 3$
5. $a = 3, b = 0$

Das Ergebnis wird 3 sein. Wie man leicht nachprüfen kann, berechnet dieser Algorithmus den größten gemeinsamen Teiler. Man erhält eine effizientere Version dieses Algorithmus, indem man in den Zeilen 8 und 10 den Modulo-operator verwendet:

if $a > b$ **then**
  set $a = a \,\%\, b$
else
  set $b = b \,\%\, a$
end if

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** $y := \frac{1}{2} \cdot \left(x + \frac{a}{x}\right)$
7: **goto** 2

In dieser Aufgabe kann man nicht durch Ausprobieren den Algorithmus erkennen.

Der Algorithmus erhält als Eingabe a eine reelle Zahl (größer oder gleich Null) und hat als Rückgabewert ebenfalls eine reelle Zahl. Die Iteration beginnt mit $x = 2$ und $y = 1$. Es wird dabei überprüft, ob der Betrag von $(x - y)$ kleiner ist als 10^{-10}. In der sechsten Zeile ist die Zuweisung

$$y = \frac{1}{2}\left(x + \frac{a}{x}\right) \tag{2}$$

und der Algorithmus führt in jeder Iteration die beiden Befehle

set $x := y$
set $y := \frac{1}{2} \cdot \left(x + \frac{a}{x}\right)$

aus, bis die Bedingung erfüllt ist. Der Algorithmus sucht also iterativ nach x, so dass

$$\left| x - \frac{1}{2}\left(x + \frac{a}{x}\right) \right| = 0\,,$$

bis auf eine Genauigkeit von 10^{-10} gilt. Diese Bedingung ist erfüllt für eine Näherung von $x = \sqrt{a}$ und der Algorithmus berechnet also die Wurzel von a.