Solutions to Lars's Übung 1

3. April 2017

1 Aufgabe 1:Debugging

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

#include <stdio.h>

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

For debugging your code the easiest thing you can do is try to compile it, and read the compilers error messages. When you try to compile this example you will get a lot of error messages. We just show only the first of them:

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

Usually it is a good practice to start the debugging with the first error message, because the later ones could be just induced by the first one. This happens also in this simple example. The compiler shows that something goes wrong in line 1. Actually this is the wrong useage of comments. They should be within `/*` and `*/`. So the corrected version is:

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

#include <stdio.h>

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

When you try to compile this one, this would not work as well. But fortunately we have much less error messeages:

```
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;
```

The warning message tells you that the compiler was not able to find the definition of your function **`pritnf`** in the standard libraries. This kind of warnings occur when you have made a typo in the name of your function. Here it is also the case: you should use **`printf`**:

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

#include <stdio.h>

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

Almost done, only one error message remained:

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

This is also a very common error. It says that before the 8-th line the instruction was not closed. In `C` after every instruction you should use **`;`**. This will show to the compiler the end of your instruction, not the new line. Now we actually finished (with respect to the compiler).

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

#include <stdio.h>

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

Now our code will be compiled and we can let it run. The output will the the famous `Hello Welt` message. In the final code, one can complain about the type of the return value of **`main`**. This has to be an integer (type **`int`**), but actually the checking of this returned value is beyond the scope of this introductory kurs.

2 Aufgabe 2: Simple code

Implement the following function in `C` (n is an integer)

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

First you should find out the type your input parameter and the type of your output. As the exercise says the input should be a variable of `int` type. The return value should also be `int`, because the function `f` returns an integer for every possible n-s. Now comes the rest, the actual implementation of `f`. As the definition of the function suggests, you should use the `if` statement in the implementation.

```c
#include<stdio.h>
int f(int n){
   if ( n % 2 == 0 )   //When n is even
     return n/2;
   else                //n is not even: i.e. it should be odd
     return (n+1)/2;
}
int main(){
   int n=3;
   int fn=f(n);
   printf("n=%d\tf(n)=%d\n", n, fn);
}
```

fn.c

For function call you should use the funtion name and in brackets you should enumerate its arguments separated by commas. Here the function `f` has only one argument. In order to store the return value a variable should stay on the left hand side of the function call.

3 Aufgabe 4

Algorithmus 1

Input: Integer non-negative number $c \in \mathbb{N}$
Output: Either `YES` or `NO`.

```
1: set $n := 2$.
2: if $n > \sqrt{c}$ then
3:     return YES
4: end if
5: if $n$ teilt $c$ then
6:     return NO
7: end if
8: set $n := n + 1$
9: goto 2
```

One first has to look at the input and output parameters. Here the input is a natural number ($c \in \mathbb{N}$) and the output is a binary one (one can think of (`yes` , `no`), or (1,0))). From this we can conclude that the Algorithm test some property of the natural numbers. Lets look it in detail.

In the first row we initialize an integer variable(n) with initial value equal to 2. The we check whether the square root of c is smaller than two. These will be the case for the following c-s:

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

So these numbers `share` the property tested by the algorithm, what about the other ones?. We check whether they can be divided by two (without remainder). So the numbers

$$4, 6, 8, 10, 12, 14, 16, \dots \tag{2}$$

`do not` share the property asked by the algorithm. For the other possible c-s we make a next round: Increase n by one, and continue from the second row. Now it should be clear, that the property we are looking after is whether the number is prime or not.

In general: Starting from $n = 2$ we check whether c can be divided with any numbers smaller than its square root[1] (with zero remainder). If not, it is obviously a prime.

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
```

This algorithm assign to its two integer input parameters an another integer number. Let start again with checking what happens in a simple case. For example: $a = 12$, $b = 9$. First we check which is the greatest of the two, that is a in the present case. Then we subtract the smaller number (now b) from the larger one (now a). After this first iteration we have $a = 3, b = 9$. We basically do these two operation until one of the numbers (either a or b) gets zero. The history of a, b values is the following:

1. $a = 12, b = 9$
2. $a = 3, b = 9$

[1]For a natural number its largest possible divisor is its square root

3. $a = 3, b = 6$
4. $a = 3, b = 3$
5. $a = 3, b = 0$

So the result will be 3 which is actually the greatest common factor (GCR) of the two input numbers. The algorithm is also called Euclid's Algorithm. In each iteration we check which is the greatest of the two numbers, and replace the larger one with the subtraction of the two. We stop, when any of the two numbers get zero, and give back the other one. To decrease the number of iterations we can replace the subtraction in line 8 and 10 with the modulo operator.

Algorithmus 3

```
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
```

The whole algorithm is based on the following identity:

$$GCF\,(a, b) = GCF\,(a\%b, b)\,, \tag{3}$$

where it was assumed that $a \geq b$. As soon as one of the elements in eq. 3 becomes zero, we can simply read the greatest common factor, it is the other element.

4 Aufgabe 5

Algorithmus 4

```
Input: Reelle Zahl $a \in \mathbb{R}_{\geq 0}$
Output: Reine reelle Zahl $x \in \mathbb{R}$
set $x := 2$ und $y := 1$.
if $|x - y| \leq 10^{-10}$ then
    return $x$
end if
set $x := y$
set $y := \frac{1}{2} \cdot \left(x + \frac{a}{x}\right)$
goto 2
```

This will be the most difficult one, because here one maps a real number to another one, and with simply tries it is hard to guess the actual form of the mapping. What's happening, is that you have a function f (a mapping) with the following rules:

$$f(x) = \frac{1}{2}\left(x + \frac{a}{x}\right) \tag{4}$$

and you do the following iterations:

1. $x = y; y = f(x)$
2. $x = y; y = f(x)$

We do this till

$$|x - f(x)| \leq 10^{-10} \tag{5}$$

In a mathematical sense we are looking for the fixpoint values of the function f. With the present form of f we can easily calculate the fixpoint x by solving an algebraic equation:

$$\frac{1}{2} \cdot \left(x + \frac{a}{x}\right) = x \tag{6}$$

which results to:

$$x = \sqrt{a} \tag{7}$$

Therefore this algorithm for every real positive a gives back its square root. Advanced question: How should we modify the code in order to work also for non-positive a-s? In order to work for a general complex number?