```c
// Nebeneffekte!

// globale Variable
int zaehler = 0;

int square (int x) {
    zaehler = zaehler+1;
    if (zaehler == 42)
        printf ("42. Aufruf!\n")
    return x*x;
}


int main() {
   // zaehler = 0

   int x = 5;
   int y = 3;
   // int result = square (x+y);
   int result = square(x) + 2*x*y + square(y);

   // zaehler = 2
}
```

```
Beispiele
=========
(( Das ist keine Haskell-Syntax! ))

apply ( f, x ) = f (x)
double (x) = 2*x
sum (x,y) = x+y
sum ({x,y}) = x+y
apply sum {x,y}
apply2 ( f, x, y ) = f (x,y)
apply ( double, 5 ) = 2*5 = 10


Aufgabe 1
=========

f x = x+1
f = \ x -> x+1

f x y = sqrt (x^2+y^2)
f = \ x y -> sqrt (x^2+y^2)

apply f x = f x
apply = \ f x -> f x

multiapply f x n = if n == 0 then x else multiapply f (f x) (n-1)

multiapply f x 0 = x
multiapply f x 1 = f x           = f(x)
multiapply f x 2 = f (f x)       = f(f(x))
multiapply f x 3 = f (f (f x))   = f(f(f(x)))
....

multiapply = \ f x n -> if n == 0 then x else multiapply f (f x) (n-1)


Aufgabe 2
=========

[f, g, h] x =  h(g(f(x)))  ??

id x = x

compose funcs x = foldl (.) id (reverse funcs) x


Aufgabe 3
=========

func = \x xs -> xs ++ [x]
func x xs = xs ++ [x]

Bsp.: func 3 [4,5,6] = [4,5,6] ++ [3] = [4,5,6,3]

myrev = foldr (\x xs -> xs ++ [x]) []

myrev [1,2,3]
= foldr func  [] [1,2,3]
= 1 `func` (foldr func [] [2,3])
= 1 `func` (2 `func` (foldr func [] [3]))
= 1 `func` (2 `func` (3 `func` (foldr func [] [])))
= 1 `func` (2 `func` (3 `func` [])
= 1 `func` (2 `func` [3])
= 1 `func` [3,2]
= [3,2,1]
```

FuLP
06.03.23

Aufg. 2

```
compose funcs x =
    foldl (.) id (reverse funcs) x
```

```
reverse [1,2,3] = [3,2,1]
reverse [double, inc, halb, dec] = [dec, halb, inc, double]
```

```
    foldl (.) id [dec, halb, inc, double]
= foldl (.) (id . dec) [halb, inc, double]
= foldl (.) ((id . dec) . halb) [inc, double]
= foldl (.) ((id.dec). halb). inc) [double]
= foldl (.) (((id.dec).halb). inc). double) [ ]
    (((id.dec). halb). inc). double
    (f.g) x = f (g x)
```

```
foldl (+) 7 [3,4,5]
          ((7+3)+4)+5
```

```
summe list
    = foldl (+) 0 list
produkt list
    = foldl (*) 1 list
```