Die lvalue-Referenz-Übergabe

```
void f(int& x)        // x wird als lvalue-Referenz des Aufrufer-Arguments erstellt
     // oder void f(int &x) oder void f(int & x)
{
     x = 5;
}
```

Diese Funktion übergibt das `int`-Argument `x` per Referenz (erkennbar am `&`-Zeichen zwischen Typangabe und Argumentname). Seit C++11 nennt man diese Art der Referenzen **lvalue**-Referenzen (denn C++11 definiert zusätzlich **rvalue**-Referenzen und **forwarding** Referenzen, erkennbar am doppelten Ampersand `&&`):

In diesem Fall kann die Funktion eine übergebene Variable des <u>Aufrufers</u> ändern. Man verwendet diese Art der Übergabe **NUR**, **wenn man das Argument auch wirklich ändern will.** C++ übergibt hier in Wirklichkeit die Adresse des Aufrufer-Arguments, sodass die Funktion hier immer auf das Original zugreift.

Was darf man per `lvalue`-Referenz übergeben (am Beispiel: `int f(int&);`)
nichtkonstante Variable vom richtigen Typ `int i; f(i);`

Was darf man **nicht** per `lvalue`-Referenz übergeben:
alles andere (Konstante, konstante Variablen, andere Typen)

Die Übergabe **einer Konstanten, einer konstanten Variablen, einer Variablen von einem anderen Typ** (auch wenn es eine passende Typumwandlung gibt!) **oder das Ergebnis eines Ausdrucks** ist in diesem Fall **ein Fehler**:

```
int x = 1; f(x);          // korrekt. x hat danach den Wert 5.
const int c = -1; f(c);   // Fehler: Konstante Var. per lv.-Referenz übergeben
f(1);                     // Fehler: Konstante per lv.-Referenz übergeben
double y = 1.5; f(y);     // Fehler: falscher Typ per lv-Referenz übergeben
f(x + 0);                 // Fehler: Rechenergebnis per lv-Referenz übergeben
f(abs(x));                // Fehler: Rechenergebnis per lv-Referenz übergeben
```