Ex. 7 Task 1

Für diese Aufgabe müssen zunächst zwei kleine Begrifflichkeiten geklärt werden:

semantisch: korrekt in der Logik, macht Sinn (in dieser Aufgabe: wird Konstanz eingehalten?)
⇒ Problem nur in 2.!

Laufzeit-gültig: semantisch gültig + gültig wenn während dem Programm die Funktionsaufrufe gültig sind.
⇒ Bei 4. Problem

Weiter sei noch zu erwähnen, was bspw. `int& foo(...)` bedeutet: return by reference. Des return der Funktion wird also als Referenz zurückgegeben. WICHTIG: Der Funktionsaufruf ist daher auch ein L-WERT!

Bsp:

```
#include <iostream>
#include <ctime>

using namespace std;

double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0};

double& setValues( int i ) {
   return vals[i];   // return a reference to the ith element
}

// main function to call above defined function.
int main () {

   cout << "Value before change" << endl;
   for ( int i = 0; i < 5; i++ ) {
      cout << "vals[" << i << "] = ";
      cout << vals[i] << endl;
   }
   setValues(1) = 20.23; // change 2nd element
   setValues(3) = 70.8;  // change 4th element

   cout << "Value after change" << endl;
   for ( int i = 0; i < 5; i++ ) {
      cout << "vals[" << i << "] = ";
      cout << vals[i] << endl;
   }
   return 0;
}
```

global

return by reference

L-value

nun zu Task 1

Da vor allem 4. ein Problem war überlassen wir die anderen dem Leser (LOL :'))

4.
```
int& foo (int i) {
        return ++i;
}
```

semantisch kohrent (keine const-Dinge zu beachten)
Laufzeit-gültig?

Sei im int main der Ausdruck

```
int n = 3;
foo (n) = 4;
```

Würde das Sinn machen? Nein, denn n wird nicht als Referenz gegeben und foo returned eine Referenz auf eine temporäre Variable.
Falls n als Referenz übergeben wird, wäre der Ausdruck oben äquivalent zu

```
++i = 4;
```
✓

und dies ist möglich, da ++i ein L-Wert ist.
(im Beispiel unten auf Seite 1 ist vals ein globaler Vektor. Daher ist die Funktion überhaupt möglich.)