Was bei der Durchsicht der zweiten Aufgabe auffiel.

Gut oder schlecht?

```
Levenshtein& Levenshtein::operator=(const Levenshtein& v2) {
if (this != &v2) {
   delete[] mpi_mat;
   mpi_mat = nullptr;
   initLevenshtein(v2.getMstr1(), v2.getMstr2());
}
return *this;
}
Levenshtein::Levenshtein(const Levenshtein& val) {
   initLevenshtein(val.getMstr1(), val.getMstr2());
}
```

Was ist am gelben Teil problematisch?

Gut oder schlecht? (2)

```
void Levenshtein::initLevenshtein(const std::vector<std::string>& astr1,
        const std::vector<std::string>& astr2) {
  mstr1 = astr1; mstr2 = astr2;
  mi_zCnt_m1 = static_cast<int> (mstr1.size());
  mi_spCnt_n2 = static_cast<int> (mstr2.size());
  mpi_mat = new int[(mi_zCnt_m1 + 1) * (mi_spCnt_n2 + 1)];

if (mi_zCnt_m1 != 0 && mi_spCnt_n2 != 0) {
   Set(0, 0, 0);
   for (int i = 1; i <= mi_zCnt_m1; i++)
      Set(i, 0, i);
   for (int j = 1; j <= mi_spCnt_n2; j++)
      Set(0, j, j);
  for (int i = 1; i <= mi_zCnt_m1; i++) {
      for (int j = 1; j <= mi_spCnt_n2; j++) {
         Set(i, j, std::min({ Get(i - 1, j) + 1, Get(i, j - 1) + 1,
              Get(i - 1, j - 1) + (mstr1[i - 1] == mstr2[j - 1] ? 0 : 1) }));
}}}}
```

Gut oder Schlecht?

```
class Levenshtein {
public:
// Standardkonstruktor
   Levenshtein() = default;

} …
```

```
Levenshtein::Levenshtein(const Levenshtein& levenshtein) {
mstr1 = levenshtein.mstr1;
mstr2 = levenshtein.mstr2;
mi_zCnt_m1 = levenshtein.mi_zCnt_m1;
mi_spCnt_n2 = levenshtein.mi_spCnt_n2;
int mpi_mat_size = (mi_zCnt_m1 + 1) * (mi_spCnt_n2 + 1);
mpi_mat = new int[mpi_mat_size];
for (int i = 0; i < mpi_mat_size; i++)
        mpi_mat[i] = levenshtein.mpi_mat[i];
   }
```

Warum geht der folgende Test also schief?

```
void MacheNix(Levenshtein ldCopy)
{
   return;
}
bool DefConstructorTest()
{
   Levenshtein defLd;
   MacheNix(defLd);
   return true;
}
```

Verbesserung

```
Levenshtein::Levenshtein(const Levenshtein& levenshtein) {
mstr1 = levenshtein.mstr1;
mstr2 = levenshtein.mstr2;
mi_zCnt_m1 = levenshtein.mi_zCnt_m1;
mi_spCnt_n2 = levenshtein.mi_spCnt_n2;
if (levenshtein.mpi_mat == nullptr){
   mpi_mat = nullptr;
}
else {
   int mpi_mat_size = (mi_zCnt_m1 + 1) * (mi_spCnt_n2 + 1);
   mpi_mat = new int[mpi_mat_size];
   for (int i = 0; i < mpi_mat_size; i++)
        mpi_mat[i] = levenshtein.mpi_mat[i];
   } }
```

```
Levenshtein::Levenshtein() {
    mpi_mat = nullptr;
    mi_zCnt_m1 = 0; mi_spCnt_n2 = 0;
}
```

Wo ist es unschön?

```
#include <algorithm>
#ifndef LEVENSHTEIN_H
#define LEVENSHTEIN_H
class Levenshtein {

private:
   vector<string> mstr1; // Erster Vektor von Strings
   vector<string> mstr2; // Zweiter Vektor von Strings
   int mi_zCnt_m1; // Groesse des ersten Vektors
   int mi_spCnt_n2; // Groesse des zweiten Vektors
   int* mpi_mat; // Dynamisches Array als Int Pointer

public:
   // Konstruktor - Levenshtein-Distanz zwischen zwei
       Vektoren von Strings berechnen
    Levenshtein(
        const vector<string>& astr1,
        const vector<string>& astr2);
```

Wo ist es unschön?

```
#include <algorithm>   // gehört auch in den Include-Wächter
#ifndef LEVENSHTEIN_H
#define LEVENSHTEIN_H
class Levenshtein {

private:
   vector<string> mstr1; // includes von vector und string fehlen
                         // using namespace std; fehlt
```

Was bei der Durchsicht der zweiten Aufgabe früher so auffiel.

Clicker-“Abstimmung“

```
bool testResize() {
    try {
        ds::MyVector<int> v(5);
        for (int i = 0; i < 20; ++i)
            v.push_back(i);
        for (int i = 0; i < 20; ++i)  {
            if (v.at(i) != i)
                return false;
        }
    }
    catch (std::exception &e)     {
        return false;
    }
    return true;
}
```

Warum gibt es hier eine Compiler Warnung?
Hinweis: Im catch-Bereich

Clicker-“Abstimmung“

```
bool testResize() {
    try {
        ds::MyVector<int> v(5);
        for (int i = 0; i < 20; ++i)
            v.push_back(i);
        for (int i = 0; i < 20; ++i)  {
            if (v.at(i) != i)
                return false;
        }
    }
    catch (std::exception &e) {
        return false;
    }
    return true;
}
```

Warum gibt es hier eine Compiler Warnung?
Hinweis: Im catch-Bereich

Variable e wurde nicht verwendet.

Clicker-“Abstimmung“

```
bool testResize() {
    try {
        ds::MyVector<int> v(5);
        for (int i = 0; i < 20; ++i)
            v.push_back(i);
        for (int i = 0; i < 20; ++i)  {
            if (v.at(i) != i)
                return false;
        }
    }
    catch (const std::exception& ) {
        return false;
    }
    return true;
}
```

Warum gibt es hier eine Compiler Warnung?
Hinweis: Im catch-Bereich

e einfach weglassen

Lösung 2: Auf dem Heap anlegen

```
int readLine(
      std::ifstream &file_handle,
      std::stringstream &nextLine,
      const int MAX_LINE_LENGTH) {
         char* p_readLine = new char[MAX_LINE_LENGTH];
         p_readLine[0] = '\0';
         while (strlen(readLine) <= 0 &&
           file_handle.good())
           file_handle.getline(p_readLine, MAX_LINE_LENGTH);
   …



      delete[] p_readLine;
```

Was ein guter Compiler so alles erkennt

```
void f(unsigned int i) {
   char a[20];
   char j;
   if (i <= 20) {
      j = a[i];
   }
}
```

Was könnte falsch sein?

a[20] gibt es nicht
a[i] ist nicht initialisiert

```
#define MAX 25
void f() {
   char ar[MAX];
   // code ...
   ar[MAX] = '\0';
}
```

Was könnte falsch sein?

a[25] gibt es nicht

Was ein guter Compiler so alles erkennt

```
typedef int V_BaseType;
void Vektor::ResizeCopy(int dim) {
   V_BaseType* tmp=new V_BaseType[dim];
   for (int j = 0; j < dimension; ++j) {
      tmp[j] = daten[j];
   }
   delete[] daten;
   daten = tmp;
   dimension = dim;
}
```

Was könnte falsch sein?

Dimension könnte größer dim sein.

Was ein guter Compiler so alles erkennt; Verbesserung

```
#include <algorithm>
typedef int V_BaseType;
void Vektor::ResizeCopy(int dim) {
   V_BaseType* tmp=new V_BaseType[dim];
   for (int j = 0; j < min(dim,dimension); ++j) {
      tmp[j] = daten[j];
   }
   delete[] daten;
   daten = tmp;
   dimension = dim;
}
```