#define, Debug, Release

Mit dem `#define`-Befehl lassen sich in C++ Konstanten oder Makros definieren, z.B.

```
1 #define PI 3.1415926
2 #define MIN(a,b) ((a) < (b) ? (a) : (b))
```

liefert das Minimum von zwei Zahlen. (Ich würde allerdings hier wegen einer Typprüfung eine Funktion bevorzugen).

Da man auch mittels Compiler Konstanten definieren kann (z.B. -DDEBUG definiert die Konstante DEBUG), lässt sich Code unterschiedlich compilieren, jenachdem, ob oder wie Konstanten definiert sind, z.B.

```
1 #ifndef DEBUG
2  clog.rdbuf(NULL);
3 #endif
```

deaktiviert alle clog-Ausgaben (`clog << „Error“ << endl;`), wenn DEBUG definiert ist.

Die Standard-Funktion `assert(bool)` bietet einen logischen Test und führt zum Programmabbruch, wenn die logische Bedingung nicht erfüllt ist, z.B.

```
1 void FilterMean(Image& dst, const Image& src, int filterSize=3){
2   assert(filterSize>=3 && filterSize <100);
3   ...
4 }
```

Durch Definition von NDEBUG lässt ich diese Funktion deaktivieren.

```
1 #include <assert.h>
2 #ifndef DEBUG
3 #define NDEBUG
4 #endif
```

Siehe http://www.cplusplus.com/reference/cassert/assert/

RGB ↔ GRAY

Implementieren Sie folgende Funktionen:

```
1 /** Transform from Gray to RGB */
2 void gray2rgb(const Image& gray, Image& rgb);
3
4 /** Transform from RGB to Gray */
5 void rgb2gray(const Image& rgb, Image& gray, int mode=0);
6
7 /** Split RGB image into separate channels */
8 void splitRGB(const Image& rgb, Image& r, Image& g, Image &b);
```

Optional

Es gibt prinzipiell vier verschiedene Möglichkeiten, aus einem RGB-Wert einen Grauwert x zu berechnen:

1. $x = \frac{1}{3}(r + g + b)$

2. $x = \max(r, g, b)$
3. $x = 0,299 \cdot r + 0,587 \cdot g + 0,114 \cdot b$
4. $x = g$

Implementieren Sie alle Möglichkeiten