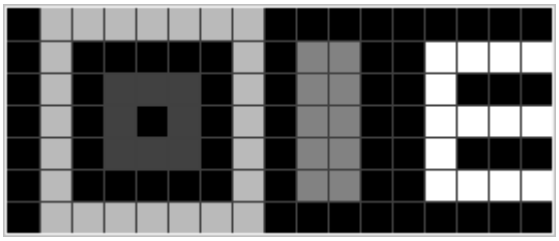
1 PGM-Bilder

In dieser Aufgabe lernen Sie die Verwendung von File-Streams.

In der nachfolgenden Abbildung sehen Sie ein Beispiel für eine PGM Bilddatei im ASCII-Format:

image_ascii.pgm

```
P2
# comment
17 7
63
0 46 46 46 46 46 46 46  0  0  0  0  0  0  0  0  0
0 46  0  0  0  0  0 46  0 32 32  0  0 63 63 63 63
0 46  0 16 16 16  0 46  0 32 32  0  0 63  0  0  0
0 46  0 16  0 16  0 46  0 32 32  0  0 63 63 63 63
0 46  0 16 16 16  0 46  0 32 32  0  0 63  0  0  0
0 46  0  0  0  0  0 46  0 32 32  0  0 63 63 63 63
0 46 46 46 46 46 46 46  0  0  0  0  0  0  0  0  0
```



PGM-Dateien können direkt in CLion betrachtet werden. Wenn Sie sie bearbeiten möchten, ändern Sie die Dateiendung auf `.txt`.

Die Datei besteht aus zwei Teilen: im oberen Teil ist der Bild-Header aufgeführt und im unteren Teil die Bildmatrix.

Der Bild-Header ist bei PGM immer im ASCII-Format, unabhängig davon ob es sich um PGM-ASCII oder PGM-Binary handelt. Nur die Bildmatrix ist bei PGM-Binary im Binärformat abgespeichert. Dadurch braucht das PGM-Binary üblicherweise viel weniger Speicherplatz und kann schneller gelesen und geschrieben werden.

Der Bild-Header muss ganz zu Beginn den Formatbezeichner P2 (für PGM-ASCII) bzw. P5 (für PGM-Binary) enthalten. Header-Zeilen die mit einem # beginnen werden als Kommentare überlesen. Nach dem Formatbezeichner müssen die Bildbreite gefolgt von der Bildhöhe in Pixel stehen. Die letzte Header-Information enthält den positiven Maximalwert, welcher in der Bildmatrix vorkommen darf.

Nach jeder Zeile steht ein `"\n"` Charakter. Auch nach der letzten Zeile.

Die genaue Spezifikation finden Sie auf sourceforge.net.

1.1 Aufgabe

Implementieren Sie die Klasse PGM, welche eine Bilddatei im Format PGM-ASCII einlesen und das gleiche Bild im Format PGM-Binary abspeichern kann. Verwenden Sie diese Klasse in einer eigenen App. Folgend ein Vorschlag für das Interface dieser Klasse:

pgm.h

```
#pragma once

#include <cstddef>

#include <vector>
#include <string>

class PGM {
 public:
  bool ReadASCII(const std::string& filename);
  bool WriteBinary(const std::string& filename);

 private:
  size_t width_;
  size_t height_;
  int32_t max_value_;
  std::vector<uint8_t> data_;
};
```

1.2 Lösung

pgm.cpp

```
#include "pgm.h"

#include <fstream>
#include <sstream>

bool ReadUntilRealToken(std::ifstream* stream, std::string* token) {
  do {
    std::getline(*stream, *token);
    if (stream->good() == false) {
      return false;
    }
  } while ((*token)[0] == '#');
  return true;
}

```

```
16 bool PGM::ReadASCII(const std::string& filename) {
17   std::string token;
18   std::ifstream file(filename);
19
20   if (file.is_open() == false) {
21     return false;
22   }
23
24   // read magic
25   if (ReadUntilRealToken(&file, &token) == false) {
26     return false;
27   }
28
29   if (token != "P2") {
30     return false;
31   }
32
33   // read width/height
34   if (ReadUntilRealToken(&file, &token) == false) {
35     return false;
36   }
37
38   std::stringstream size_stream(token);
39   size_stream >> width_ >> height_;
40   if (width_ <= 0 || height_ <= 0) {
41     return false;
42   }
43
44   // read max value
45   if (ReadUntilRealToken(&file, &token) == false) {
46     return false;
47   }
48
49   std::stringstream max_value_stream(token);
50   max_value_stream >> max_value_;
51   if (max_value_ <= 0 || max_value_ > 255) {
52     return false;
53   }
54
55   // allocate image buffer
56   const size_t size = height_ * width_;
57   data_.resize(size);
58
59   // read data
60   int32_t value;
61   for (size_t i = 0; i < size; ++i) {
62     file >> value;
63     data_[i] = static_cast<uint8_t>(value);
```

```
64    }
65
66    return true;
67  }
68
69  bool PGM::WriteBinary(const std::string& filename) {
70    if (data_.empty()) {
71      return false;
72    }
73
74    std::ofstream file(filename, std::ofstream::binary);
75
76    // write magic
77    file << "P5" << std::endl;
78
79    // write width and height and 0x0A as end of line (because of binary mode)
80    file << width_ << ' ' << height_ << std::endl;
81
82    // write max value
83    file << max_value_ << std::endl;
84
85    // write data
86    file.write(reinterpret_cast<char*>(data_.data()), static_cast<ptrdiff_t>(
          width_ * height_));
87
88    return true;
89  }
```

main.cpp

```
1   #include <iostream>
2
3   #include "pgm.h"
4
5   int main() {
6     PGM image;
7
8     if (image.ReadASCII("/tmp/image_ascii.pgm") == false) {
9       std::cerr << "read failed" << std::endl;
10      return EXIT_FAILURE;
11    }
12
13    if (image.WriteBinary("/tmp/image_binary.pgm") == false) {
14      std::cerr << "write failed" << std::endl;
15      return EXIT_FAILURE;
16    }
17
18    std::cout << "conversion finished" << std::endl;
19    return EXIT_SUCCESS;
```

```
20  }
```