Praxisorientierte Einführung in C++
Aufgabenblatt 6

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

28. Mai 2014

Aufgabe1: Operatoren

Verwenden Sie folgende Implementation der `Image`-Klasse vom letzten Übungsblatt. Die Datei `image.h` können Sie zusätzlich auf der Vorlesungswebseite herunterladen.

```
#ifndef IMAGE_H
#define IMAGE_H
#include <algorithm>
#include <fstream>
#include <string>
typedef unsigned int uint ;
typedef unsigned char uchar ;
class Image {
  uint width, height;
  uchar * data;
  public :
  inline Image():
    width(0),height(0),data(0){}
  inline Image (uint width, uint height):
    width(width),height(height),
    data(getDim() ? new uchar[getDim()] : 0){
    std::fill(data,data+getDim(),0);
  }
  inline Image(const Image &other):
    width(other.width),height(other.height),
   data(getDim() ? new uchar[getDim()] : 0){
    std::copy(other.data,other.data+getDim(),data);
  }
  inline ~Image (){  if(data) delete [] data; }
  inline uint getDim() const { return width*height; }
  inline uint getWidth () const { return width; }
  inline uint getHeight () const { return height; }
  inline uchar &operator()(int x, int y) { return data[x+width*y]; }
  inline const uchar &operator()(int x, int y) const { return data[x+width*y]; }
  static void saveImage(const Image *image,  const std::string &filename){
    if(!image || !image->getDim()) return;
    std::ofstream s(filename.c_str());
    s << "P2" << std::endl << image->width << " "
      << image->height << std::endl << 255 << std::endl;

    for(uint y=0;y<image->height;++y){
      for(uint x=0;x<image->width;++x){
        s << (int) (*image)(x,y) << " ";
      }
      s << std::endl;
```

```
    }
  }
};
# endif
```

a. (3 Punkte) Implementieren Sei den Zuweisungsoperator

```
Image &operator=(const Image &other);
```

der eine elementweise Zuweisung durchführt. Falls die Dimensionen der Bilder nicht übereinstimmen soll die Dimension des `rvalue`-Bildes übernommen werden. Hier muss also ggf. der Speicher des `lvalue`-Bildes freigegeben und neu reserviert werden.

b. (3 Punkte) Implementieren Sie die Operatoren

```
Image operator+(const Image &other) const;
```

```
Image operator-(const Image &other) const;
```

```
Image &operator+=(const Image&other);
```

und

```
Image &operator-=(const Image &other);
```

die die arithmetischen Operationen pixelweise durchführen. Falls die Bilder in ihrer Größe nicht kompatibel sind, soll das Ergebnisbild von `A+B` mit der Größe `max(A.width,B.width)` × `max(A.height,B.height)` erzeugt werden. Fehlende Pixel sollen mit dem Wert 0 angenommen werden:
Beispiel

```
A = 1 1 1 1 1
    1 1 1 1 1

B = 2 2
    2 2
    2 2

A+B = 3 3 1 1 1
      3 3 1 1 1
      2 2 0 0 0
```

Die Bilder sollen also in diesen Fall an der oberen Linken Ecke übereinander gelegt werden.
Beachten Sie die unterschiedliche Semantik der Operatoren, welche sich auch im unterschielichen Rückgabetyp wiederspiegelt. Die Operatoren `+=` und `-=` verändern das `lvalue`-Bild und geben dieses als Referenz zurück. Die Operatoren `+` und `-` hingegen erzeugen ein neues Bild für das Ergebnis, welches zurückgegeben wird. Das `lvalue`-Bild wird hierbei nicht verändert, weshalb diese Operatoren `const` sind.
`Hinweis:` Im standard-Header `<algorithm>`, welcher in `image.h` bereits eingebunden wird, finden sie das Funktions-Template `std::transform` welches Ihnen bei der Implementierung der Operatoren viel Arbeit abnehmen kann. Die Verwendung von `std::transform` ist aber optional.

Aufgabe2: Laden und Speichern von Bildern

Wie z.B. in Java können in C++ Funktionen in einer Klasse auch als *statisch* deklariert werden. Hierfür wird das Schlüsselwort `static` verwendet. Statische Funktionen arbeiten nicht auf Objekt-Instanzen, sondern sie haben *globalen* Charakter und müssen durch Voranstellen des Klassennamens und dem `::`-Operator aufgerufen werden. Beispiel:

```
#include "image.h"
int main(){
  Image I(256,256);
  for(uint x=0;x<256;++x){
    for(uint y=0;y<256;++y){
      I(x,y) = x;
    }
  }
  Image::saveImage(&I,"image.pgm");
}
```

Der Klassenscope ist hier also *quasi* der Namensraum der Funktion. Statische Funktionen werden i.d.R. immer dann verwendet, wenn globale Funktionen semantisch zu einer Objektklasse gehören.

a. (6 Punkte) Fügen Sie Ihrer Image-Klasse die statische Funktion mit der Signatur

```
static Image *load(const std::string &filename);
```

hinzu. Die Funktion soll in der Lage sein, Bilder vom Typ Plain-PGM[1] einzulesen .
Das PGM Format ist ein Spezialfall des PPM-Formats(Portable Pixmap). Eine Plain-PGM-Datei ist folgendermaßen Aufgebaut:

```
P2
# Kommentarzeilen die alle mit einer Raute beginnen
# ...
BreiteDesBildes HoeheDesBildes
MaximalerGrauwert
PixelDaten...
```

`P2` ist der sog. *Magic-Code*, welcher darüber informiert, welcher Subtyp des allgemeinen PPM-Formats vorliegt (Hier P2 für Plain-PGM: Die Pixeldaten sind in diesem Fall nicht binär, sondern ASCII-Text). Andere Magic-Codes sollen nicht geladen werden können; in diesem Fall soll die Funktion einen null-Pointer zurückliefern. `HoeheDesBildes` und `BreiteDesBildes` müssen verwendet werden um die Bild-Dimension zu bestimmen. Die Zusatzinformation `MaximalerGrauwert` soll beim Lesen ignoriert werden. Jede Zeile in `PixelDaten` entspricht den Grauwerten einer Bildzeile[2]. Hier ein Beispiel:

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Diese Datei (`feep.pgm`) steht ebenfalls auf der Vorlesungswebseite zum Download bereit.Zum Lesen aus einer Datei kann die C++-Klasse `ifstream` aus den Header `<fstream>` verwendet werden. Hierbei kann `std::ifstream` fast vollständig analog zu `std::cin` benutzt werden, wie folgendes Beispiel zeigt:

```
void foo(const char *filename){
  std::ifstream s(filename);

  // lesen des naechsten Tokens mit dem '>>'-Operator
  // also analog zu std::cin
  int i; float f; unsigned char c;
  s >> i; // konvertiert das naechste Token in einen int
  s >> f; // s.o.
  s >> c; // s.o.
```

[1] siehe `http://en.wikipedia.org/wiki/Portable_Gray_Map`

[2] Eigentlich können an beliebigen Stellen Kommentarzeilen (beginnend mit '#') vorkommen. Der Einfachheit halber sollen diese allerdings nur, wir oben beschrieben, zwischen Magic-Code und Dimensionsangaben vorkommen dürfen

```
}
```

Zum Einlesen einer ganzen Zeile aus einer Datei kann die Funktion `std::getline(std::istream&,std::string&)` verwendet werden (verfügbar durch Einbinden des Headers `<string>`):

```
std::string nextLine;
getline(s,nextLine); // nextLine enthaelt nun die naechste Zeile
```

Strings können ebenfalls in einen Stream 'konvertiert' werden; hierfür wird die Klasse `std::stringstream` (Header `<sstream>`) verwendet:

```
std::stringstream strstr(nextLine);
while(strstr.good()){
  string t;
  strstr >> t;
  std::cout << "Naechstes Token von nextLine ist:" << t << std::endl;
}
```

Aufgabe3: Verwenden der Bildklasse

a. (3 Punkte) Implementieren Sie eine Anwendung (`main`-Funktion in "main.cpp"), die drei Dateinamen als Kommandozeilenparameter erwartet (Bild A, Bild B, und Bild C). Das Programm soll die Bilder A und B einlesen, pixelweise addieren (C=A+B) und das Ergebnisbild C abspeichern. Achten Sie hierbei darauf, dass Opereratoren **ausschliesslich** auf Objekten/Referenzen angewandt werden können; nicht auf Pointern:

```
  Image *A = ...;
  Image *B = ...;

  Image *C = A+B; // Absolut falsch: hier werden
                  // Speicheradressen addiert
  Image C = (*A) + (*B); // GUT
```