Praxisorientierte Einführung in C++
Aufgabenblatt 5

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

22. Mai 2014

Aufgabe1: Klassen und Dynamische Speicherverwaltung

a. (3 Punkte) Schreiben Sie Header ("image.h") und Quellcodedatei ("image.cpp") für eine Klasse `Image`, die ein Graustufenbild repräsentiert. Benutzen Sie folgende Listings als Ausgangspunkt:

```
// image.h
#ifndef IMAGE_H
#define IMAGE_H

typedef unsigned int uint;
typedef unsigned char uchar;

struct Image {
  Image(uint width, uint height);
  ~Image();

  uint getWidth() const;
  uint getHeight() const;

  protected:
    uint m_width;
    uint m_height;

    uchar *m_imageData;
};

#endif
```

```
// image.cpp
#include "image.h"

uint Image::getWidth() const {
  return m_width;
}

uint Image::getHeight() const {
  return m_height;
}
```

Implementieren Sie Konstruktor und Destruktor. Beachten Sie hierbei, daß im Konstruktor `Image()` das Element `m_imageData` initialisiert, und genug Speicher für ein Bild der Größe width x height reserviert werden muss. Entsprechend muss dieser Speicher im Destruktor wieder freigegeben werden. Initialisieren Sie die Elemente `m_width` und `m_height` in der Initialisierungsliste des Konstruktors. Berücksichtigen

Sie den Sonderfall, dass eine oder beide Dimensionen null sind.

Bearbeitungshinweis: Die Image-Klasse soll in den folgenden Aufgaben sukzessive erweitert werden. Hierbei sollen alle Funktionen, Methoden und Operatoren **nicht** inline deklariert und implementiert werden. Deklarieren Sie also alle Funktionen im Header-File und implementieren Sie sie im der zugehörigen Quellcode-Datei. Es sollen also für Aufgabe 1 nur **drei** Dateien angefertigt und abgegeben werden: "image.h", "image.cpp" und für die letzte Aufgabe: "main.cpp"!

b. (3 Punkte) Implementieren Sie die folgenden Methoden:

```
const uchar &pixel(const uint &x, const uint &y) const;
```

und

```
uchar &pixel(const uint &x, const uint &y);
```

Die Methoden sollen Referenzen für den Pixel an der Stelle (x, y) im Bild zurückliefern.
Da für die Bilddaten nur ein einziger linearer und zusammenhängender Speicherbereich existiert (referenziert durch `m_imageData`), muss das zweidimensionale Bild auch irgendwie linear im Speicher abgelegt werden. Dies wird i.d.R. so implementiert, dass das Bild Zeile für Zeile nacheinander im dem entspr. Speicherblock abgelegt wird.

Beispiel:
Sei X ein 2x2-Bild mit Komponenten X_{11}, X_{12}, X_{21}, X_{22}, wobei der erste Index für die Zeile und der zweite Index für die Spalte steht, dann werden diese vier Elemente auch in der selben Reihenfolge im Speicher abgelegt: (also X_{11}, X_{12}, X_{21}, X_{22}).

c. (2 Punkte) Implementieren Sie einen Copy-Constructor für die Klasse `Image`, der eine "tiefe" Kopie erstellt ("tief" in dem Sinne, daß neuer Speicher reserviert wird, und das Bild Wert für Wert kopiert wird, so daß am Ende beide `Image`-Objekte vollkommen unabhängig sind). Zur Erinnerung: In diesem Fall hätte ein üblicher Copy-Constructor folgende Signatur:

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

d. (2 Punkte) Implementieren Sie nun noch die Methode mit der Signatur

```
void print() const;
```

Die Methode soll das Bild als Zahlenmatrix auf der Standardausgabe anzeigen. Achten Sie hierbei darauf, dass kleine Zahlen mit führenden Freizeichen verlängert werden, so dass auch die Spalten eines Bildes zueinander ausgerichtet sind.

e. (3 Punkte) Schreiben Sie ein kleines Programm, welches Ihre Bildklasse testet. Hierfür soll ein 16×16 großes Bild erstellt werden, welches dann mit einem 8×8-Schachbrettmuster gefüllt wird (schwarz: 0, weiß: 255, oben links ist weiß). Anschließend soll das Bild mit dem Copy-Konstruktor tief kopiert, und dann mittels `print` angezeigt werden.

```
void print() const;
```

Die Methode soll das Bild als Zahlenmatrix auf der Standardausgabe anzeigen. Achten Sie hierbei darauf, dass kleine Zahlen mit führenden Freizeichen verlängert werden, so dass auch die Spalten eines Bildes zueinander ausgerichtet sind.

Aufgabe2: Verkettete Listen

Endlich haben wir alle Sprachprimitive zusammen um verkettete Listen zu implementieren!

a. (10 Punkte) Implementieren Sie alle Methoden der `List`-Klasse die in folgendem Beispiel nicht implementiert sind.

```
#include <stdexcept>
#include <iostream>
#include <string>

struct Node{
  Node(Node *next, std::string value):
    next(next),value(value){}
  Node *next;
  std::string value;
};

class List{
  Node *first; // Erstes Element, 0 falls die Liste leer ist
  int len;     // Laenge der liste
  Node *nthNode(int index);   // Hilfsfunktion: O(index)

public:
  // Default-Konstruktor (Laenge 0): O(1)
  List():first(0),len(0){}

  // Copy-Konstruktor: O(other.len)
  List(const List &other);

  // Zuweisungs-Operator O(len+other.len)
  List &operator=(const List &other){
    clear();
    if(!other.len) return *this;
    Node *it = first = new Node(0,other.first->value);
    for(Node *n = other.first->next; n ; n = n->next){
      it = it->next = new Node(0,n->value);
    }

    len = other.len;
    return *this;
  }

  // Destruktor (gibt den Speicher von allen Nodes frei): O(len)
  ~List();

  // Haengt der Liste ein Element hinten an: O(len)
  void push_back(std::string value);

  // Fuegt am Anfang der Liste ein Element ein: O(1)
  void push_front(std::string value);

  // gibt eine Referenz auf das index-te Element zurueck: O(index)
  std::string &at(int index);

  // Entfernt alle Elemente: O(len)
  void clear();

  // Zeigt alle Elemente an: hier: O(len*len)
  void show(){
    std::cout << "List[" << len << "]:{";
      for(int i=0;i<len;++i){
        std::cout << at(i) << (i==len-1?'}':',');
    }
    std::cout << std::endl;
  }
```

```
};
```

Anmerkungen:

- Exception-Handling wurde zwar noch nicht besprochen, wird hier aber evtl. benötigt. Wenn der Header `stdexcept` eingebunden wurde, kann z.B. mittels `throw std::runtime_error("geht nicht");` eine Exception geworfen werden, was eine Fehlermeldung ausgibt und das Programm sauber beendet. Das ist zwar nur die halbe Wahrheit, aber für diese Aufgabe sollte das erst mal ausreichen.
- Studieren Sie die Implementierung des vorgegebenen Zuweisungs-Operators um zu verstehen wie die Liste intern aufgebaut ist.
- Die Hilfsfunktion `nthNode` muss nicht unbedingt implementiert werden. Wir denken allerdings, dass diese Funktion an mehreren Stellen hilfreich sein könnte.
- Machen Sie sich über die Komplexität der Funktionen Gedanken. Die Implementation von `show` ist bspw. suboptimal. Die angestrebten Komplexitäts-Klassen sind in der *Big O-Notation*[1] mit angegeben.
- Auch wenn sie nicht alle Funktionen hinbekommen, können sie Teilpunkte für diese Aufgabe ergattern.
- Wenn Ihre Implementationen richtig sind, sollte folgende `main`-Funktion eine sinnvolle Ausgabe erzeugen.

```
int main(){
  List l;
  l.push_back("bla");
  l.push_back("blub");
  l.push_front("blubbler");
  l.show();

  List(l).show();
}
```

- Denken sie daran, dass der Speicher für alle `Node`-Instanzen mittels `delete` freigegeben werden muss.

[1] google