Praxisorientierte Einführung in C++
Aufgabenblatt 10

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

26. Juni 2014

Aufgabe1: Klassen-Templates

Ausgangsbasis für diese Aufgabe stellt folgendes sehr einfach gehaltenes `SimplePoint2D`-struct dar:

```
struct SimplePoint2D{
  int x;
  int y;
};
```

Anmerkung: Alle Methoden und Konstruktoren können `inline` implementiert werden. Es sei denn, eine nicht-`inline`-Implementierung wird explizit erwünscht.

a. (2 Punkte) Verallgmeinern Sie das `struct SimplePoint2D` durch Implementation eines Klassen-Templates

```
template<class T>struct Point2D;
```

, so dass anstelle von `int`s beliebige Datentypen verwendet werden können.

b. (5 Punkte) Implementieren Sie für das `Point2D`-Template folgende Methoden, Konstruktoren und Operatoren:

```
template<class T> Point2D<T>::Point2D(T x=T(0), T y = T(0));
```

```
template<class T> T &Point2D::x();
```

```
template<class T> T &Point2D::y();
```

```
template<class T> const T &Point2D::x() const;
```

```
template<class T> const T &Point2D::y() const;
```

```
template<class T> std::ostream &operator<<(std::ostream &s,const Point2D<T> &p);
```

Hinweis: Beachten Sie, dass die Default-Argumente `T(0)` des Konstruktors nur Typen zulässt, für die ein Konstruktor mit 0-Argument existiert. Der ostream-Operator ist natürlich nicht als Methode, sondern als globale Funktion zu überladen.

c. (3 Punkte) Fügen Sie der Klasse einen impliziten cast-operator nach `bool` hinzu. Signatur:

```
template<class T> Point2D<T>::operator bool() const;
```

Implementieren sie diesen nicht `inline`, sondern in einer separaten .cpp-Datei. Instanzieren Sie den Operator explizit für die Typen `Point2D<int>` und `Point2D<float>`. Falls Sie sich nicht sicher sind, ob Sie die richtig instanziert haben, sollten Sie sich ein kleines Test-Programm schreiben, in welchem dieser Operator tatsächlich verwendet wird. Die Implementation des `bool`-cast-Operators soll `false` zurückgeben, falls `(x==0) && (y==0)` gilt.

d. (5 Punkte) Erweitern Sie das Template

```
template<class T>struct Point2D;
```

zu einem Template

```
template<class T, int N>struct FixedVector;
```

welches zusätzlich auch über die Dimensionsanzahl des Punktes abstrahiert. Reimplementieren Sie die Funktionen von

```
template<class T>struct Point2D;
```

in geeigneter Art und Weise: Die Zugriffsoperatoren `x()` und `y()` müssen hierbei durch den Index-Operator ersetzt werden:

```
template<class T, int N> T &FixedVector<T,N>::operator[](int index);
```

und

```
template<class T, int N> const T &FixedVector<T,N>::operator[](int index) const;
```

Diese sollten selbstverständlich auch implementiert werden. Den Cast-Operator können Sie aber dieses Mal inline definieren.

e. (3 Punkte) Implementieren Sie den Multiplikationsoperator als Memberfunktion. Dieser soll das Skalarprodukt zweier gleichlanger Vektoren (mit gleichem Typ T) realisieren. Welche Signatur ist sinnvoll?

f. (5 Punkte) Implementieren Sie eine Template-Memberfunktion `concatenate`, die zwei Vektoren unterschiedlicher Länge (aber mit Elementen vom selben Typ) konkateniert. Beachten Sie, dass diese selbst noch mal ein Template ist. Es soll also möglich sein, z.B. einen `FixedVector<float,3>` mit einem `FixedVector<float,7>` zu einem `FixedVector<float,10>` zu konkatenieren. Es soll aber nicht möglich sein, einen `FixedVector<float,3>` mit einem `FixedVector<double,7>` zu konkatenieren.