Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Fachbereich Design Informatik Medien
Studiengang Bachelor Allgemeine Informatik

Programmieren 3 AI – Praktikum

Übungsblatt 6

1,5 + 0,5 Punkte

Sebastian Flothow

2011-11-21

Abnahme: 2011-11-28
Zur Abnahme ist die persönliche Anwesenheit erforderlich. Bearbeiten Sie die Aufgaben so, dass Sie nicht nur funktionierenden Code vorzuweisen haben, sondern diesen auch erläutern und Fragen dazu beantworten können. Zusätzlich sind die Lösungen am Tag der Abnahme per Email an `sebastian@flothow.de` einzusenden; dies dient u.a. der Prüfung auf eventuelle Plagiate.

Es muss für jede Aufgabe eine getrennte Lösung in einem separaten Verzeichnis vorliegen. Statten Sie jedes Abgabeverzeichnis mit einem Makefile aus. Dieses muss ein Default-Target haben, dass Ihr Programm mit `g++ -Wall -Wextra -pedantic` kompiliert und linkt, sowie das Target „`clean`“, dass die erzeugten Dateien wieder entfernt. Rufen Sie vor dem Packen `make clean` auf (d.h. die eingesandten Archive sollen nur Quelltexte enthalten, keine Kompilate).

Senden Sie Ihre Abgabe als tar.gz-Archiv(e). Diese müssen so gepackt sein, dass jedes Archiv genau ein Verzeichnis enthält, dessen Name (bis auf das Suffix) mit dem des Archivs übereinstimmt; ein solches Archiv kann durch einen Aufruf der Form `tar -czf` *`verzeichnis`*`.tar.gz` *`verzeichnis`* erzeugt werden.

Beim Kompilieren der Programme mit `g++ -Wall -Wextra -pedantic` dürfen keine Fehler oder Warnungen auftreten.

Aufgabe 1

Kopieren Sie Ihre Quelltexte von Blatt 5 Aufgabe 2 und führen Sie die folgenden Änderungen daran aus. Vergewissern Sie sich nach jedem Schritt dass Ihr Programm kompilierbar ist und korrekt läuft. Schreiben Sie dazu geeigneten Code, der die hinzugekommene Funktionalität testet.

1. Markieren Sie in der Klasse `Shape` die Methoden für die Berechnung von Fläche und Umfang als rein virtuell (abstrakt).

2. Fügen Sie der Klasse `Shape` eine rein virtuelle Methode booleschen Rückgabetyps hinzu, mit der geprüft werden kann, ob ein gegebener Punkt innerhalb der Form liegt. Diese Methode soll genau einen Parameter vom Typ `Point` haben. Implementieren Sie die Funktion in den abgeleiteten Klassen so, dass sie genau dann *true* zurückgibt, wenn der übergebene `Point` innerhalb der jeweiligen Form liegt. Berücksichtigen Sie dabei sowohl die Abmessungen der Form als auch die Position ihres Mittelpunkts.

3. Kopieren Sie das Interface `Movable` (Listing 1, auch als Datei von der Veranstaltungsseite herunterladbar) in Ihr Arbeitsverzeichnis. Lassen Sie `Shape` davon erben und implementieren Sie das Interface.

4. Schreiben Sie das Interface `Scalable`. Dieses soll eine Skalierungsmethode definieren, die als einzigen Parameter einen Skalierungsfaktor vom Typ `double` akzeptiert. Lassen Sie `Shape` davon erben und implementieren Sie das Interface so dass bei Aufruf dieser Methode für eine Form alle Abmessungen mit dem übergebenen Faktor skaliert werden.

Hinweis zu 2.: Nehmen Sie beim Rechteck an, dass seine Kanten parallel zu den Koordinatenachsen liegen. Nehmen Sie beim gleichseitigen Dreieck an dass eine Spitze nach oben weist und die gegenüberliegende Seite parallel zur x-Achse liegt. Nehmen Sie bei der Ellipse an, dass Ihre Halbachsen parallel zu den Koordinatenachsen liegen. Seien a, b die Längen der Halbachsen und $x_M|y_M$ der Mittelpunkt der Ellipse, dann liegt der Punkt $x|y$ innerhalb der Ellipse genau dann wenn

$$\frac{(x - x_M)^2}{a^2} + \frac{(y - y_M)^2}{b^2} \leq 1$$

Aufgabe 2

Laden Sie das Archiv `Rational.tar.gz` von der Veranstaltungsseite herunter und entpacken Sie es. Ändern Sie die Klasse `Rational` so, dass stets sichergestellt ist, dass der Nenner nicht negativ ist.

Die Klasse überlädt bereits die Multiplikationsoperatoren (`*`, `*=`); überladen Sie zusätzlich die Operatoren für die übrigen Grundrechenarten (`+`, `+=`, `-`, `-=`, `/`, `/=`). Die resultierenden Brüche müssen nicht gekürzt sein, der Nenner darf jedoch nicht negativ werden.

Überladen Sie weiterhin die Vergleichsoperatoren (`==`, `!=`, `<`, `<=`, `>`, `>=`). Verwenden Sie dabei auschließlich Ganzzahlarithmetik; Sie können sich zunutze machen dass

$$\frac{a}{b} \diamond \frac{c}{d} \Leftrightarrow ad \diamond cb$$

für $b, d \geq 0$ und $\diamond \in \{=, \neq, <, \leq, >, \geq\}$.

Erweitern Sie die `main`-Funktion um Tests für die neuen Operatoren.

Aufgabe 3 (optional, 1/2 Punkt)

Die Klasse `Rational` überlädt bereits den Ausgabeoperator (`<<`); überladen Sie zusätzlich den Eingabeoperator (`>>`). Der Eingabeoperator soll das vom Ausgabeoperator erzeugte Format einlesen können, das Testprogramm `iotest.cpp` (Listing 2, auch als Datei von der Veranstaltungsseite herunterladbar) muss also fehlerfrei laufen.[1]

Achten Sie auf sorgfältige Fehlerprüfung bei den Eingabeoperationen. Falls ein `Rational`-spezifischer Fehler auftritt (z.B. inkorrekte Syntax der Eingabe) sollten Sie vermittels der Methode `setstate` das *failbit* des Eingabestroms setzen, damit der Aufrufer den Fehler erkennen kann.

[1] Assertions können durch `#define NDEBUG` bzw. die Compiler-Option `-DNDEBUG` deaktiviert werden – das ist hier aber natürlich nicht zulässig.

Listing 1: movable.hpp

```
#ifndef _MOVABLE_HPP_
#define _MOVABLE_HPP_

#include "point.hpp"

/**
 * Interface fuer bewegliche Objekte.
 */
class Movable {

        /**
         * Verschiebt das Objekt
         *
         * @param p Punkt, um dessen Koordinaten verschoben
            werden soll (d.h. p wird als Vektor aufgefasst)
         */
        virtual void move(Point p) = 0;
};

#endif
```

Listing 2: iotest.cpp

```
#include <sstream>
#include <cassert>
#include "rational.hpp"

int main() {
        Rational x(17, 23);
        Rational y;

        std::stringstream strstr;
        strstr << x;
        strstr >> y;
        assert(x == y);

        return 0;
}
```