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

Fachbereich Design Informatik Medien
Studiengang Bachelor Allgemeine Informatik

Programmieren 3 AI – Praktikum

Übungsblatt 5

1 Punkt

Sebastian Flothow

2011-11-14

Abnahme: 2011-11-21
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. 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

Bei dieser Aufgabe soll ein binärer Suchbaum für Ganzzahlen implementiert werden. Achten Sie dabei insbesondere darauf, dass belegter Speicher auch wieder freigegeben wird; stellen Sie dazu sicher, dass jedes Objekt zu jedem Zeitpunkt genau einen Eigentümer hat, der für die Freigabe des Objekts zuständig ist.

Schreiben Sie eine Klasse `TreeNode`, die drei Felder hat: eine Ganzzahl mit dem Wert des Knotens, einen Pointer auf den linken Kindknoten sowie einen Pointer auf den rechten Kindknoten. Ein Knoten ist Eigentümer seiner Kindknoten. Die Klasse soll folgende öffentliche Funktionalität bieten:

- Konstruktor `TreeNode(int n)`: Initialisiert den Knoten so, dass er den Wert `n` und keine Kinder hat
- Destruktor `~TreeNode()`: Gibt die Kindknoten frei
- Methode `void insert(TreeNode* node)`: Fügt `node` als Kindknoten korrekt sortiert ein (dazu ist evtl. ein rekursiver Aufruf nötig). Der Aufruf dieser Methode überträgt das Eigentum an `node` vom Aufrufer an den Aufgerufenen.
- Methode `int count() const`: Gibt die Anzahl der Knoten im Baum zurück
- Methode `int height() const`: Gibt die Höhe des Baums zurück
- Methode `int sum() const`: Gibt die Summe aller Einträge des Baums zurück
- Methode `int min() const`: Gibt den kleinsten Wert des Baums zurück
- Methode `std::string toString() const`: Gibt einen String mit allen Einträgen in der durch Inorder-Traversierung entstehenden Reihenfolge zurück (d.h. die Werte sind sortiert)

Schreiben Sie zum Testen eine `main`-Funktion, die einige Werte als Knoten in einen Baum einfügt, den Baum als String ausgibt und den Baum wieder löscht. Für die Erzeugung der Knoten ist `new` zu verwenden.

Aufgabe 2

Schreiben Sie eine Klasse `Point`, die die x- und y-Koordinate eines Punkts im zweidimensionalen Raum speichert. Versehen Sie die Klasse mit einem Konstruktor, der diese beiden Parameter entgegennimmt.

Schreiben Sie eine Klasse `Shape`, die als einzige Eigenschaft einen Mittelpunkt hat, dargestellt als `Point`. Die Klasse soll einen *nichtöffentlichen* Konstruktor mit einem `Point` als Parameter haben. Sehen Sie Methoden für die Berechnung von Fläche und Umfang vor; diese können hier einfach 0 zurückgeben. Gestalten Sie die Klasse vererbungsgeeignet, d.h. der Destruktor und alle Methoden sollen virtuell sein.

Leiten Sie von `Shape` die Klasse `Rect` ab, die ein Rechteck beschreibt. Diese soll als zusätzliche Eigenschaften Höhe und Breite des Rechtecks enthalten. Leiten Sie von `Rect` die Klasse `Square` ab, die ein Quadrat beschreibt. Leiten Sie von `Shape` die Klasse `Ellipse` ab, die eine Ellipse beschreibt. Diese soll als zusätzliche Eigenschaften die Länge der beiden Halbachsen enthalten. Leiten Sie von `Ellipse` die Klasse `Circle` ab, die einen Kreis beschreibt. Leiten Sie von `Shape` die Klasse `EquiTri` ab, die ein gleichseitiges Dreieck beschreibt. Diese soll als zusätzliche Eigenschaften die Seitenlänge enthalten.

Statten Sie alle abgeleiteten Klassen mit geeigneten öffentlichen Konstruktoren aus. Überschreiben Sie die beiden Methoden dort wo es sinnvoll ist, so dass für alle Objekte korrekte Werte berechnet werden.

Testen Sie ihre Implementation mit einer `main`-Funktion mit einem Array von Shape-Pointern. Erzeugen Sie einige Objekte der verschiedenen Typen per `new`, speichern Sie

die Pointer im Array und lassen Sie dann in einer Schleife für alle Objekte Fläche und Umfang berechnen und ausgeben. Löschen Sie danach alle angelegten Objekte wieder.