Programmiertechnik II

Sommersemester 2015 – Aufgabenblatt 7

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Anmerkungen zu Aufgabenblatt 7

Der zu verwendende Programmrahmen steht als Zip-Archiv `uebung7.zip` im Moodle unter der Adresse `https://moodle.hpi3d.de/course/view.php?id=75` zum Download zur Verfügung. Die Aufgaben sollen in den benannten und im Programmrahmen zumeist gekennzeichneten Bereichen implementiert werden. Bevor Sie mit der Lösung beginnen, lesen Sie das gesamte Aufgabenblatt und überprüfen Sie, ob der Programmrahmen auf Ihrem System fehlerfrei kompiliert, die Datensätze enthalten sind und sich alle Programme ausführen lassen. Wir wünschen viel Spaß und Erfolg bei der Bearbeitung der Aufgaben!

Aufgabe 7.1: Elementare Klasse Vector3d (6 Punkte)

Ausgangsbasis bilden eine nicht objektorientierte Implementierung einer Datenstruktur und zugehörige Funktionen für 3D-Vektoren (`vector.cpp`). Überarbeiten Sie die gegebene Implementierung, indem Sie eine elementare C++-Klasse für 3D-Vektoren entwickeln, die die Datenstruktur und die Funktionen als Klassenelemente formuliert und erweitert:

a) Überführen Sie `struct Vector3d` in eine Klasse `class Vector3d`. Die Datenfelder der Struktur (`x_,y_,z_`) werden zu nicht-öffentlichen, gekapselten Attributen der Klasse. Ermöglichen Sie den Zugriff auf die Koordinaten eines 3D-Vektors in syntaktisch kompakter Form durch den Subscription-Operator. Fügen Sie entsprechende Konstruktoren ein. `Vector3d`-Objekte sollen auch durch einen Konstruktor erzeugt werden können, der eine Initialisierungsliste verarbeiten kann.

b) Formulieren Sie die bisherigen Funktionen `length`, `normalize`, `cross` und `dot` als öffentliche Funktionen der Klasse. Formulieren Sie die bisherigen Funktionen `plus`, `minus`, `times` und `divide` als öffentliche Operatoren der Klasse; überladen Sie dazu die Operatoren `+`, `-`, `*`, `/` sowie `==` und `!=` (Überprüfung auf Wertgleichheit, nicht Identität). Vervollständigen Sie die Operationen der Klasse durch zusätzliche globale Operatoren, z. B. `Vector3d operator*(double, Vector3d)`.

c) Definieren Sie einen Stream-Ausgabeoperator im Format `(x, y, z)` analog zur bisherigen Funktion `print()`.

Im Programmrahmen ist eine auskommentierte Testfunktion `test()` implementiert, die von Ihrer Klassenimplementierung erfüllt werden soll. Kommentieren Sie die Funktion wieder ein, um ihre Implementierung zu testen.

Aufgabe 7.2: Modellierung und Implementierung von Geldbeträgen (4 Punkte)

Modellieren und implementieren Sie eine Klasse `Amount` in `amount.cpp`, die Geldbeträge, z.B. für Waren oder Leistungen, repräsentiert. Ein Geldbetrag umfasst einen Nettobetrag, eine Mehrwertsteuer (gem. eines festlegbaren Prozentsatz) und einen Bruttobetrag sowie die zugrundeliegende Währung und eine Bezeichnung (Text). Nettobetrag, Mehrwertsteuerbetrag, Mehrwertsteuersatz und Bruttobetrag sind stets konsistent. In der Klasse sollen die zwei Währungen Euro (EUR) und Dollar (USD), als `enum`-Klassenelemente eingetragen werden. Überlegen Sie sich sinnvolle Attribute und Methoden der Klasse `Amount` und implementieren Sie diese. Vervollständigen Sie zudem die Funktion `test()` die alle Methoden automatisch testen soll.

Anforderungen: a) Die Konstruktoren soll eine gewisse Bequemlichkeit bieten. b) Geldbeträge sollen auch automatisch konvertierbar sein in die jeweils andere Währung. c) Der Wechselkurs soll über einen festen Umwandlungsfaktor definiert sein. d) Die Default-Währung ist Euro (EUR).

Aufgabe 7.3: Verrechnungskonto mit Protokoll (4 Punkte)

Die gegebene Klasse `Account` in `account.cpp` repräsentiert allgemeine Verrechnungskonten (vgl. Vorlesung). Die im Programmrahmen angegebene Implementierung dieser Klasse ist bislang nicht ausgelegt, um durch Vererbung spezialisiert zu werden.

Überarbeiten Sie diese Klasse und entwickeln Sie eine abgeleitete Klasse `LoggedAccount`. Die Klasse `LoggedAccount` repräsentiert Verrechnungskonten, die zusätzlich jede Buchung (`debit`, `credit`) und die Änderung des Überziehungsrahmens (`setLimit`) protokolliert, d.h. jedes Konto führt eine interne Liste, die die Kontovorgänge verzeichnet. Als Protokolleintrag reicht ein Schlüsselwort ("credit", "debit", "limit change") und der zugehörige Betrag. Auch soll jederzeit (also auch direkt nach Konstruktion eines LoggedAccount-Objekts) in dieser Transaktionsliste der Ausgangssaldo ("initial balance") als erster Eintrag und der aktuelle Saldo ("current balance") als letzter Eintrag stehen; dazwischen sind die oben genannten Transaktionen (`credit`, `debit`, `setLimit`) aufzuführen. Ein LoggedAccount-Objekt unterhält eigenständig sein Protokoll, das auch abgefragt werden kann.

Zusatzaufgabe: Erzeuger-Verbraucher-Muster (1 Klausurpunkt)

Das Erzeuger-Verbraucher-Muster (Producer-Consumer Pattern) unterscheidet zwei Kategorien von Threads: Ein 'Producer' ist verantwortlich für die Herstellung von Objekten (d.h. Konstruktion, Initialisierung und Bereitstellung), wohingegen ein 'Consumer' die hergestellten Objekte abruft und verwertet. Sowohl Producer wie Consumer arbeiten in eigenen Threads, um die Ausführung der Teilprozesse zu entkoppeln und zu skalieren.

In der Aufgabe `producerconsumer.cpp` soll der Producer Thread (`main`-Methode) dynamisch Objekte vom Typ `Item` auf dem Heap anlegen und in einer Warteschlange vorhalten. `Item`-Objekte besitzen ein eindeutigen Identifikator (id_), kennen ihren Entstehungszeitpunkt (creation_); jedes Item kann zum aktuellen Zeitpunkt seine bisherige Lebenszeit (lifeTime()) berechnen. Die Klasse hat einen globalen Objektzähler (items_), um am Ende des Programms feststellen zu können, ob ein Memory-Leak vorliegt. Der Consumer-Thread (`Consumer::work`-Methode) ruft diese Objekte von dort ab, verarbeitet sie und dealloziiert sie nach der Verarbeitung. Die Verarbeitung besteht in dieser Aufgabe darin, dass jedes Item im Consumer Thread eine maximale Lebenszeit (z.B. 4.000 Millisekunden) besitzt; ist diese überschritten, dann gilt das Item als verarbeitet.

Vervollständigen Sie den Programmrahmen, in dem Sie die Funktionalität für Producer und Consumer implementieren. Beachten Sie dabei das Speichermanagement für Heap-Objekte. Bei erfolgreicher Entfernung der Items ist die letzte Ausgabe auf der Konsole: `0 objects left in memory`

Allgemeine Hinweise zur Bearbeitung und Abgabe

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen sind nicht vorgesehen. Je Gruppe ist nur eine Abgabe notwendig.
- Bitte reichen Sie Ihre Lösungen bis **Montag, den 13. Juli um 09:00 Uhr** ein.
- Zur Prüfungszulassung muss ein Aufgabenblatt zumindest bearbeitet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bewertet werden. Aufgaben mit Klausurpunkten werden 1:1 berücksichtigt.
- Die Aufgaben können auf allen wesentlichen Plattformen (Windows, Linux, OS X) bearbeitet werden. Lesen Sie bei Fragen zum Kompilieren und Ausführen bitte genau die den Archiven jeweils beiligende README.TXT. Bestehen weitere Fragen und Probleme, kontaktieren Sie einen der Betreuer direkt oder nutzen Sie das Forum im Moodle.
- Die Durchsicht zur Bewertung kann sowohl auf Windows, Linux oder OS X erfolgen. Es wird folglich eine **plattformunabhängige Lösung** der Aufgaben erwartet.
- Archivieren (Zip) Sie zur Abgabe Ihren bearbeiteten Programmrahmen und ergänzen Sie Ihre Matrikelnummer(n) im Bezeichner des Zip-Archivs entsprechend im folgenden Format: **`uebung7_matrikNr1_matrikNr2.zip`**. Beachten Sie, dass dabei nur die vollständigen Quelltexte, die CMake-Konfiguration sowie eventuelle Zusatzdaten gepackt werden (alles was im gegebenen Programmrahmen entsprechend vorhanden ist). Senden Sie uns keinesfalls Kompilate und temporäre Dateien (*.obj, *.pdb, *.ilk, *.ncb etc.). Testen Sie vor dem Verschicken, ob die Projekte aus den Zipdateien fehlerfrei kompiliert und ausgeführt werden können.
- Reichen Sie Ihr Zip-Archiv im Moodle ein: `https://moodle.hpi3d.de/course/view.php?id=75`.