Dirk Praetorius,
Marcus Page

Wintersemester 2011
6. Dez. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 8

Aufgabe 8.1*. *[Klassendefinition]* Schreiben Sie eine Klasse `university`. Diese soll neben den Feldern `anzStudents, city` und `name` die Methoden `graduate` und `newStudent` haben. Wird `graduate` aufgerufen, so verringert sich die Anzahl der Studenten um 1, wohingegen `newStudent` die Anzahl um 1 erhöht. Alle Datenfelder sollen als `private` deklariert sein. Sie müssen sich also zusätzlich `get-` und `set-`Methoden schreiben. Speichern Sie den Source-Code unter `university.cpp` in das Verzeichnis `serie08`.

Aufgabe 8.2*. *[Methoden überladen]* Für die Personalabteilung der Universität ist es sehr mühsam, Studenten immer nur einzeln hinzuzufügen oder aus dem System zu löschen. Überladen Sie daher die Methoden `graduate` und `newStudent` dahingehend, dass die Anzahl der abschließenden bzw. neu hinzukommenden Studenten mit übergeben werden kann. Schreiben Sie außerdem Konstruktoren die Ihre Universität mit sinnvollen Daten befüllen. Wird das Objekt nicht direkt initialisiert, so soll `anzStudents = 0, city = noWhere` und `name = noName` eingetragen werden. Erweitern Sie die Klasse zusätzlich um eine `plot`-Routine, welche sämtliche Daten über die Standardausgabe ausgibt. Speichern Sie den Source-Code unter `university2.cpp` in das Verzeichnis `serie08`.

Aufgabe 8.3*. *[Klassendefinition]* Schreiben Sie eine Klasse `myRealVector`, die einen Vektor $x \in \mathbb{R}^n$ speichert. Die eigentlichen Werte sollen in einem `private`-Feld `entries` gespeichert werden. Verwenden Sie hierzu den Container `vector`. Die Länge des Vektors soll *nicht* explizit gespeichert werden. Schreiben Sie `get-` und `set`-Methoden um sowohl Länge, also auch Einträge des Vektors manipulieren zu können. Stellen Sie hierbei sicher, dass die Länge des Vektors nicht negativ werden kann. Schreiben Sie einen Konstruktor, dem Sie die Länge $n \in \mathbb{N}$ und eine Zahl $y \in \mathbb{R}$ übergeben und der dann den konstanten y-Vektor der Länge n erzeugt. Schreiben Sie außerdem eine Funktion `plotVec` die den Vektor über die Standardausgabe ausgibt. Speichern Sie den Source-Code unter `myRealVector.cpp` in das Verzeichnis `serie08`.
Tip: Der `vector`-Container lässt sich mit `vector<double> vec(size, a)` so initialisieren, dass direkt ein Vektor der Länge `size` mit den Einträgen $a \in \mathbb{R}$ erzeugt wird.

Aufgabe 8.4*. *[Methoden überladen, Standardwerte]* In Aufgabe 7.3 haben Sie die ℓ^2-Norm für Vektoren $x \in \mathbb{R}^n$ kennengelernt und implementiert. Es gibt jedoch noch andere Normen, wie etwa die ℓ^1-Norm

$$\|x\|_{\ell^1} := \sum_{i=0}^{n-1} |x_i|.$$

Erweitern Sie Ihre Vektor Klasse aus Aufgabe 8.3 um eine öffentliche `getNorm` Routine, die einen Integer `t` $\in \{1, 2\}$ übernimmt und abhängig davon die ℓ^1 oder die ℓ^2-Norm zurückliefert. Falls keine Zahl mit übergeben wird, so soll standardmäßig die ℓ^2-Norm berechnet werden. Sie kennen zwei verschiedene Möglichkeiten dies zu erreichen. Welche? Erklären Sie die Unterschiede. Speichern Sie den Source-Code unter `myRealvector2.cpp` in das Verzeichnis `serie08`.

Aufgabe 8.5. *[Zugriffsspezifizierer]* Wie lauten die verschiedenen Modi der Zugriffskontrolle? Was sind die Unterschiede? Was sind die Vorteile dieses kontrollierten Zugriffs?

Aufgabe 8.6. *[Klassendefinition]* Schreiben Sie eine `matrix`-Klasse, die analog zu Ihrer Vektor-Klasse aus den obigen Aufgaben aufgebaut ist. Die Daten sollen hierbei in einer dynamischen Matrix vom Typ `vector <vector <double>>` gespeichert werden. Schreiben Sie außerdem Methoden `getSize, setSize` sowie `getEntry` und `setEntry`. Analog zu oben soll außerdem ein Kontruktor geschrieben werden, der eine mit Nullen befüllte Matrix beliebiger Größe erzeugt. Speichern Sie den Source-Code unter `myMatrix.cpp` in das Verzeichnis `serie08`.

Aufgabe 8.7. *[Matrix-Matrix-Multiplikation]* Erweitern Sie die Klasse `matrix` aus der vorherigen Aufgabe um eine Methode `multiply`, die eine Matrix übernimmt und mit der aktuellen Matrix multipliziert. Rückgabewert soll wieder `matrix` sein. Für die Berechnung von $A = B * C$ soll also `A = B.multiply(C)` aufgerufen werden. Die Rückgabematrix A muss also innherhalb von `multiply` erstellt und befüllt werden.

Aufgabe 8.8. *[Methoden überladen]* Überladen Sie die Methode `multiply` von `matrix` so, dass auch ein Vektor übergeben werden kann. Der Rückgabwert muss in diesem Fall natürlich auch ein Vektor sein. Denken Sie daran, dass Sie die Matrix-Vektor-Multiplikation auf dem letzten Übungsblatt schon einmal implementiert haben. In guter C++ Manier sollten Sie diesen Code wiederverwenden.

Aufgabe 8.9. *[Matrixnorm]* Die Zeilensummennorm einer Matrix haben $M \in \mathbb{R}^{n \times n}$ haben Sie bereits mehrfach implementiert. In dieser Aufgabe soll zusätzlich die Spaltensummennorm implementiert werden, welche analog zur Zeilensummennorm als

$$\|A\| := \max_{j=0\ldots n-1} \sum_{i=0}^{m-1} |A_{ij}|$$

definiert ist. Erweitern Sie dann Ihre Matrix-Klasse um eine `getNorm`-Methode, die — abhängig davon ob 1 oder 2 mit übergeben wird — die Zeilen- bzw. Spaltensummennorm zurückliefert.

Aufgabe 8.10. *[Iteratoren]* In C++ können Vektoren mittels sogenannter Iteratoren bequem durchlaufen werden. Finden Sie heraus, wie das funktioniert. Ändern Sie schließlich die `plot`-Routine in Ihrer Vektorklasse so ab, dass der Vektor mit Hilfe von Iteratoren durchlaufen wird.