Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
21./22.12.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Erweitern Sie die Klasse `Fraction` aus der Vorlesung (Folie 230) um die `public` Methode `void kuerzen()`, die die gekürzte Darstellung des Bruchs `zaehler/nenner` bestimmt. Dazu benütze man den euklidischen Algorithmus aus der Vorlesung. Weiters implementiere man eine Methode `setWert(string wert)`, welche eine beliebige Gleitkommazahl in einen Bruch umwandelt. Die Zahl ist dabei als String gegeben. Um diese Methode zu erstellen können Sie wie folgt vorgehen. Zunächst sucht man in dem String nach dem Dezimalpunkt und zählt die Nachkommastellen. Dann löscht man den Dezimalpunkt aus dem String heraus. Den String, welcher nun eine natürliche Zahl repräsentiert, kann nun mittels der Funktion `atoi` in eine `int` Variable umgewandelt werden. Diese Zahl benützt man als Zähler. Als Nenner verwende man 10^p wobei $p \in \mathbb{N}$ die Anzahl der Nachkommastellen sei. Rufen Sie dann `kuerzen()` auf. Überladen Sie schließlich die Methode `setWert` geeignet, um auch `setWert(n)` für n vom Typ `int` in sinnvoller Weise ausführen zu können. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `bruch.{hpp,cpp}` in das Verzeichnis `serie09`.
Hinweis: Mit der Methode `find` der Klasse `string` können Sie nach einem bestimmten Zeichen im String suchen, z.B.: `int pos = wert.find('.')` gibt die Stelle des Dezimalpunkts im String `wert` an. Mit `wert.erase(pos,k)` werden ab der Stelle `pos` die `k` darauf folgenden Zeichen im String gelöscht. Die Funktion `atoi` aus der Standardbibliothek `cstdlib` konvertiert einen gegebenen String (im C-Stil) in eine `int` Variable. Um die Zeichenkette eines Strings zu erhalten kann man die Methode `c_str()` der Klasse `string` benutzen.

Aufgabe 9.2. Schreiben Sie eine Klasse `University`. Diese soll neben den Feldern `numStudents`, `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. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `University.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.3. Für die Personalabteilung der Universität ist es sehr mühsam, Studenten immer nur einzeln hinzuzufügen oder aus dem System zu löschen. Überladen Sie die Methoden `graduate` und `newStudent` der Klasse `University` aus Aufgabe 9.2 so, 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 `numStudents = 0`, `city = noWhere` und `name = noName` eingetragen werden. Erweitern Sie die Klasse zusätzlich um eine `plot` -Routine, welche sämtliche Daten von `University` ausgibt. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `University.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.4. Schreiben Sie eine Klasse `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `'F'`) auch untere (Typ `'L'`) und obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \qquad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

als obere bzw. untere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$ bzw. $\ell_{jk} = 0$ für $j < k$. Eine vollbesetzte Matrix werde im Fortran-Format spaltenweise als dynamischer Vektor der Länge $n \cdot n$ gespeichert. Dreiecksmatrizen sollen in einem Vektor der Länge $\sum_{j=1}^{n} j = n(n+1)/2$ gespeichert werden. Implementieren Sie folgende Funktionalitäten:

- Standardkonstruktor, der eine 0×0 Matrix vom Typ `'F'` anlegt
- Konstruktor, bei dem der Typ und die Dimension mit übergeben werden kann
- Destruktor
- `get` und `set`-Methoden für die Matrix-Einträge, den Typ und die Dimension

Dabei hängen insbesondere die `get` und `set`-Methoden für die Matrixeinträge vom Matrixtyp (Dreiecksmatrix!) ab. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `matrix.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.5. Erweitern Sie die Klasse `Matrix` aus Aufgabe 9.4 um

- eine Methode `scanMatrix(char typ, int n)`, die den Typ, sowie die Matrix $A \in \mathbb{R}^{n \times n}$ dem Typ entsprechend von der Tastatur einliest,
- eine Methode `printMatrix()`, die die Matrix am Bildschirm ausgibt,
- eine Methode `columnsumnorm()`, die die Spaltensummennorm

$$\|A\| = \max_{k=0,\ldots,n-1} \sum_{j=0}^{n-1} |a_{jk}|$$

  berechnet und zurückgibt,

- eine Methode `rowsumnorm()`, die die Zeilensummennorm

$$\|A\| = \max_{j=0,\ldots,n-1} \sum_{k=0}^{n-1} |a_{jk}|$$

  berechnet und zurückgibt.

Beachten Sie, dass die Methoden bei unteren bzw. oberen Dreiecksmatrizen nur auf Koeffizienten a_{jk} bzw. a_{kj} für $0 \leq k \leq j \leq n-1$ zugreifen können. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `matrix2.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.6. Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ mit $U_{jj} \neq 0$ für alle $j = 0, \ldots, n-1$. Zu gegebenem $b \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Ux = b$. Leiten Sie eine Formel her, um die Lösung $x \in \mathbb{R}^n$ von $Ux = b$ zu berechnen, indem Sie die Formel des Matrix-Vektor-Produkts mithilfe der Dreiecksstruktur von U vereinfachen. Implementieren Sie eine Funktion, um für eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ und einen Vektor $b \in \mathbb{R}^n$ das System $Ux = b$ zu lösen. U ist dabei vom Typ `Matrix` aus Aufgabe 9.4 und b ist dabei vom Typ `Vector` aus der Vorlesung (vgl. Folien 249 ff.). Stellen Sie mittels `assert` sicher, dass die Dimensionen passen und dass $U_{jj} \neq 0$ für alle j gilt. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `solveMatrixU.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.7. Wir betrachten die Klasse `Matrix` aus Aufgabe 9.4 und die Klasse `Vector` aus der Vorlesung (vgl. Folien 249 ff.). Implementieren Sie für die Klasse `Matrix` die Methode `solve`, welche das lineare Gleichungssystem $Ax = b$ mit dem *Gauß'schen Eliminationsverfahren* löst. Gegeben seien eine Matrix $A \in \mathbb{R}^{n \times n}$ und eine rechte Seite $b \in \mathbb{R}^n$:

- Zunächst bringt man die Matrix A auf obere Dreiecksform, indem man die Unbekannten eliminiert. Gleichzeitig modifiziert man die rechte Seite b.

- Das entstandene Gleichungssystem mit oberer Dreiecksmatrix A löst man mit Aufgabe 9.6.

Im ersten Eliminationsschritt zieht man geeignete Vielfache der ersten Zeile von den übrigen Zeilen ab und erhält dadurch eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Im zweiten Eliminationsschritt zieht man nun geeignete Vielfache der zweiten Zeile von den übrigen Zeilen ab und erhält eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3} & \dots & a_{nn} \end{pmatrix}.$$

Nach $n-1$ Eliminationsschritten erhält man also eine obere Dreiecksmatrix A. Stellen Sie sich mittels `assert` sicher, dass $a_{kk} \neq 0$ im k-ten Eliminationsschritt gilt. Berücksichtigen Sie, dass auch die rechte Seite $b \in \mathbb{R}^n$ geeignet modifiziert werden muss. Lösen Sie das System $Ax = b$ mit oberer Dreiecksmatrix A mit Hilfe von Aufgabe 9.6. Welchen Aufwand hat Ihre Implementierung des Gauß'schen Eliminationsverfahren und warum? Machen Sie sich das Vorgehen zunächst an einem Beispiel mit $A \in \mathbb{R}^{2\times 2}$ sowie $A \in \mathbb{R}^{3\times 3}$ klar. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `gauss.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.8. Das Gauß'sche Eliminationsverfahren aus Aufgabe 9.7 scheitert, falls im k-ten Schritt $a_{kk} = 0$ gilt, auch wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung x besitzt. Deshalb kann man das Verfahren um eine sogenannte *Pivot-Suche* erweitern:

- Im k-ten Schritt wählt man aus $a_{kk}, \dots, a_{nk}$ das betragsgrößte Element a_{pk}.
- Dann vertauscht man die k-te und die p-te Zeile von A (und b).
- Schließlich führt man den Eliminationsschritt aus wie zuvor.

Implementieren Sie für die Klasse `Matrix` aus Aufgabe 9.4 die Methode `gausspivot`, die die Lösung von $Ax = b$ wie angegeben berechnet. (Man kann übrigens mathematisch beweisen, dass das Gauss-Verfahren mit Pivot-Suche genau dann durchführbar ist, wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung besitzt. Einen Beweis dazu sehen Sie in der Vorlesung zur Numerischen Mathematik.) Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `gausspivot.cpp` in das Verzeichnis `serie09`.