Dirk Praetorius,
Michele Ruggeri

Wintersemester 2015/16
09.-10.12.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Schreiben Sie einen Strukturdatentyp `polynomial` zur Speicherung von Polynomen, die bezüglich der Monombasis dargestellt sind, d.h. $p(x) = \sum_{j=0}^{n} a_j x^j$. Es ist also der Grad $n \in \mathbb{N}_0$ sowie der Koeffizientenvektor $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$ zu speichern. Schreiben Sie alle nötigen Funktionen, um mit dieser Struktur arbeiten zu können (`newPoly`, `delPoly`, `getPolyDegree`, `getPolyCoefficient`, `setPolyCoefficient`). Speichern Sie den Source-Code unter `polynomial.c` in das Verzeichnis `serie09`.

Aufgabe 9.2. Die Summe $r = p + q$ zweier Polynome p, q ist wieder ein Polynom. Schreiben Sie eine Funktion `addPolynomials`, die die Summe r berechnet. Zur Speicherung verwende man die Struktur aus Aufgabe 9.1. Zum Test schreibe man eine Funktion, die zwei Polynome einliest und deren Summe ausgibt. Speichern Sie den Source-Code unter `addPolynomials.c` in das Verzeichnis `serie09`.

Aufgabe 9.3. Das Produkt $r = pq$ zweier Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{j=0}^{n} b_j x^j$ ist wieder ein Polynom. Schreiben Sie eine Funktion `prodPoly`, die das Produktpolynom r berechnet und in der Struktur aus Aufgabe 9.1 speichert. Überlegen Sie sich zunächst, welchen Grad das Polynom r hat und wie sich die Koeffizienten berechnen lassen. Schreiben Sie ein aufrufendes Hauptprogramm, in dem p und q eingelesen und $r = pq$ ausgegeben wird. Testen Sie Ihren Code an einem geeigneten Beispiel. Speichern Sie den Source-Code unter `prodPoly.c` in das Verzeichnis `serie09`.

Aufgabe 9.4. Die k-te Ableitung $p^{(k)}$ eines Polynoms p ist wieder ein Polynom. Schreiben Sie eine Funktion `differentiatePolynomial`, die zu gegebenem p und $k \in \mathbb{N}$ die Ableitung $p^{(k)}$ berechnet. Zur Speicherung verwende man die Struktur aus Aufgabe 9.1. Schreiben Sie ein Hauptprogramm, das p und k einliest und $p^{(k)}$ ausgibt. Testen Sie Ihren Code an einem geeigneten Beispiel. Speichern Sie den Source-Code unter `differentiatePolynomial.c` in das Verzeichnis `serie09`.

Aufgabe 9.5. Schreiben Sie eine Struktur `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `0`) 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. Schreiben Sie die Funktionen, um mit dieser Struktur arbeiten zu können (`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Speichern Sie den Source-Code unter `matrix.c` in das Verzeichnis `serie09`. Dabei hängen insbesondere die Funktionen `getMatrixEntry` und `setMatrixEntry` vom Matrixtyp (Dreiecksmatrix!) ab.

Aufgabe 9.6. Schreiben Sie eine Funktion `spaltensummennorm`, die die Spaltensummennorm

$$\|A\|_S := \max_{j=1,\dots,n} \sum_{i=1}^{n} |A_{ij}|$$

einer Matrix $A \in \mathbb{R}^{n \times n}$ zurückgibt. Die Matrix A sei dabei in der Datenstruktur aus Aufgabe 9.5 gespeichert, und etwaige Struktur (Dreiecksmatrix!) von A soll ausgenutzt werden. Speichern Sie den Source-Code unter `spaltensummennorm` in das Verzeichnis `serie09`. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 9.7. 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. Speichern Sie den Source-Code unter `University.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.8. Erstellen Sie eine Klasse `SparKonto` mit den Variablen `kontonummer`, `guthaben` und `zinssatz`. Ferner sollen noch die `get` und `set`-Methoden für die Variablen `zinssatz` und `kontonummer` implementiert werden. Um das Guthaben zu ändern schreiben Sie die Methoden `abheben` und `einzahlen`. Beachten Sie, dass Sie bei einem Sparkonto nicht ins Minus gehen können. Der Zinssatz und die Kontonummer dürfen natürlich auch nicht negativ werden. Schließlich implementiere man noch die Methode `berechneGuthaben`. Speichern Sie den Source-Code unter `sparkonto.cpp` in das Verzeichnis `serie09`.