Dirk Praetorius,
Marcus Page

Wintersemester 2011
18. Nov. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1*. Schreiben Sie eine Struktur `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `'F'`) auch obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix}$$

als obere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$. Eine vollbesetzte Matrix werde als dynamische Matrix der Größe $n \times n$ gespeichert. Dreiecksmatrizen sollen entsprechend effizient gespeichert werden (orientieren Sie sich hierbei an Aufgabe 5.1). 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 `serie06`. Achten Sie darauf, dass viele dieser Funktionen vom Matrixtyp abhängen.

Aufgabe 6.2*. Schreiben Sie eine Funktion `loadMatrix`, die eine $n \times n$-Matrix A aus einer ASCII Datei ausliest. und entsprechend abspeichert. Die Größe n soll hierbei ein Input-Parameter der Funktion sein. Verwenden Sie die Struktur aus Aufgabe 6.1. Schreiben Sie ferner eine Funktion `isUmatrix`, die eine vollbesetzte Matrix A daraufhin checkt, ob es sich um eine obere Dreiecksmatrix handelt. Ist dies der Fall, so soll der Typ von `'F'` auf `'U'` geändert, und die Matrix entsprechend effizient abgespeichert werden. Achten Sie hierbei darauf, den Speicher entsprechend neu zu allokieren. Speichern Sie den Source-Code unter `loadNcheck.c` in das Verzeichnis `serie06`.

Aufgabe 6.3*. Was versteht man und 'Aufwand' eines Algorithmus. Erklären Sie diesen Begriff formal. Welchen Aufwand hat das Lösen eines Systems mit oberer Dreiecksmatrix (Aufgabe 5.2)?

Aufgabe 6.4*. 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 `serie06`.

Aufgabe 6.5. 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 6.4. 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 `serie06`.

Aufgabe 6.6. Schreiben Sie eine Funktion `savepolynomial`, dass ein Polynom in Form seines Koeffizientenvektors in eine Datei speichert. Ferner schreibe man ein aufrufendes Hauptprogramm, in dem ein Polynom sowie ein Dateiname von der Tastatur eingelesen werden und das Polynom in die angegebene Datei gespeichert wird. Als Datenformat für das Polynom verwende man die Struktur aus Aufgabe 6.4. Speichern Sie den Source-Code unter `savepoly.c` in das Verzeichnis `serie06`.

Aufgabe 6.7. Eine Variante zur Berechnung einer Nullstelle einer Funktion $f : [a, b] \to \mathbb{R}$ ist das *Newton-Verfahren*. Ausgehend von einem Startwert x_0 definiert man induktiv eine Folge (x_n) durch

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Man realisiere das Newton-Verfahren in einer Funktion `newton`, wobei die Iteration abgebrochen wird, falls entweder

$$|f'(x_n)| \leq \tau,$$

$$|f(x_n)| \leq \tau$$

oder

$$|x_n - x_{n-1}| \leq \tau$$

gilt. Im ersten Fall gebe man zusätzlich eine Warnung aus, dass das numerische Ergebnis vermutlich falsch ist (warum?). Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` arbeiten, die als eigenständige Funktion implementiert wird. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x_0 und $\tau > 0$ eingelesen und x_n ausgegeben wird. Speichern Sie den Source-Code unter `newton.c` in das Verzeichnis `serie06`.

Aufgabe 6.8. Wie kann man das Newton-Verfahren aus Aufgabe 6.7 dazu verwenden um $\sqrt{2}$ zu berechnen.

Aufgabe 6.9. Eine weitere Möglichkeit eine Nullstelle einer Funktion zu finden bietet das *Bisektionsverfahren*. Gegeben sei hierzu eine stetige Funktion $f : [a, b] \to \mathbb{R}$, d.h. eine Funktion ohne Sprünge. Es gelte

$$f(a) \cdot f(b) \leq 0.$$

Dann hat f eine Nullstelle z_0, die im Folgenden mittels Bisektion (= Intervallhalbierung) approximiert werden soll: Der Bisektionsalgorithmus arbeitet wie folgt: In jedem Bisektionsschritt definiert man $c := (a + b)/2$ als Intervallmittelpunkt. Aufgrund der Voraussetzung gilt

$$f(a) \cdot f(c) \leq 0 \quad \text{oder} \quad f(c) \cdot f(b) \leq 0.$$

Im Fall $f(a) \cdot f(c) \leq 0$ liegt eine Nullstelle im Intervall $[a, c]$, und man ersetzt daher b durch c. Im Fall $f(c) \cdot f(b) \leq 0$ liegt eine Nullstelle im Intervall $[c, b]$, und man ersetzt daher a durch c. In beiden Fällen geht man also vom Intervall $[a, b]$ zu einem Teilintervall halber Länge über. — Machen Sie sich das Verfahren zunächst anhand des Beispiels $f : [0, 3] \to \mathbb{R}$, $f(x) = x-1$ klar. — Schreiben Sie eine Funktion `bisection`, die als Parameter a, b und eine Toleranz $\tau > 0$ übernimmt und den Bisektionsschritt wiederholt, bis man vom Startintervall $[a, b]$ zu einem Intervall $[a, b]$ mit Länge $|b - a| \leq \tau$ übergegangen ist. In diesem Fall gebe man a zurück. Es gilt dann $|a - z_0| \leq |a - b| \leq \tau$, d.h. a ist eine Approximation einer Nullstelle z_0 bis auf eine Genauigkeit von τ. Verwenden Sie auch dieses Verfahren um $\sqrt{2}$ zu berechnen. Schreiben Sie

ferner ein Hauptprogramm, das $b, \tau > 0$ einliest und die Approximation von z_0 ausgibt. Vergleichen Sie die Ergebnisse mit Aufgabe 6.8. Speichern Sie den Source-Code unter `bisection.c` in das Verzeichnis `serie06`.

Aufgabe 6.10. Das Produkt $U = AB$ zweier oberer Dreiecksmatrizen $A, B \in \mathbb{R}^{n \times n}$ ist wieder eine obere Dreiecksmatrix. Beweisen Sie diese Aussage zunächst mathematisch, indem Sie sich die Formel für das Matrix-Matrix-Produkt hinschreiben und mittels der Voraussetzung an A und B die Indizes vereinfachen. Schreiben Sie eine Funktion `matrixmatrixU`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von U, d.h. U_{jk} für $j \leq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. Sie müssen die anfangs hergeleitete Formel verwenden. Speichern Sie den Source-Code unter `matrixmatrixU.c` in das Verzeichnis `serie06`.