Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
27.11.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Die Aufgabe 9.3 und 9.8 sollen in C++ implementiert werden. Die Programmierteile der restlichen Aufgaben sollen in C realisiert werden.

Aufgabe 9.1*. Schreiben Sie eine Struktur `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. Schreiben Sie die Funktionen, um mit dieser Struktur arbeiten zu können (`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Dabei hängen insbesondere die Funktionen `getMatrixEntry` und `setMatrixEntry` vom Matrixtyp (Dreiecksmatrix!) ab.

Aufgabe 9.2*. Nicht jede Matrix $A \in \mathbb{R}^{n \times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n,$$

$$\ell_{ki} = \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n,$$

$$\ell_{ii} = 1 \quad \text{für } i = 1, \dots, n,$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n \times n}$ sind Null. Schreiben Sie eine Funktion `computeLU`, die die LU-Zerlegung von A berechnet und zurückgibt. Dazu überlege man, in welcher Reihenfolge man die Einträge von L und U berechnen

muss, damit die angegebenen Formeln wohldefiniert sind (d.h. alles was benötigt wird, ist bereits zuvor berechnet worden). Die Matrizen sollen in der Struktur aus Aufgabe 9.1 gespeichert werden. Speichern Sie den Source-Code unter `computeLU.c` in das Verzeichnis `serie09`.

Aufgabe 9.3*. *(C++)* In Aufgabe 7.4 haben Sie schwachbesetzte Matrizen kennengelernt. Schreiben Sie eine Struktur `SMatrix` die zur Speicherung schwach besetzter Matrizen $S \in \mathbb{R}^{n\times n}$ dient. Implementieren Sie Funktionen `newSMatrix, delSMatrix` zum Allokieren und Freigeben des Speichers einer schwachbesetzten Matrix und eine Funktion `getSMatrixN`, welche die Größe $n \in \mathbb{N}$ zurückgibt. In dieser Aufgabe sollen Sie den `vector` Container in C++ für alle auftretenden Vektoren benützen.

Aufgabe 9.4*. Recherchieren Sie im WWW wie man auf Dateien (im ASCII-Format) zugreifen und wie man diese manipulieren kann. Dazu können Sie z.B. auch die VO-Unterlagen aus den letzten Semestern zu Rate ziehen.
Schreiben Sie dann eine Funktion `saveMatrix` welche eine vollbesetzte Matrix $A \in \mathbb{R}^{n\times n}$ in eine ASCII-Datei speichert. Ist die Matrix vom Typ `'L'` oder `'U'` so sollen an den entsprechenden Stellen Null-Einträge geschrieben werden. In der Datei werden also nur vollbesetzte Matrizen abgespeichert. Schreiben Sie weiters 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 9.1.
Schreiben Sie ferner eine Funktion `isUmatrix` bzw. `isLmatrix`, die eine zuvor mittels `loadMatrix` vollbesetzte Matrix A daraufhin überprüft, ob es sich um eine obere bzw. untere Dreiecksmatrix handelt. Ist dies der Fall, so soll der Typ von `'F'` auf `'U'` bzw. `'L'` geändert, und die Matrix entsprechend effizient abgespeichert werden. Achten Sie hierbei darauf, den Speicher entsprechend neu zu allokieren. Speichern Sie den Source-Code unter `saveAndLoadMatrix.c` in das Verzeichnis `serie09`.

Aufgabe 9.5. Das Produkt $C = AB$ zweier oberer bzw. unterer Dreiecksmatrizen $A, B \in \mathbb{R}^{n\times n}$ ist wieder eine obere bzw. untere 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 `matmult`, die die Produktmatrix $C = AB$ zweier beliebiger quadratischer Matrizen $A, B \in \mathbb{R}^{n\times n}$ berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von C berechnet werden, falls es sich bei A, B um obere bzw. untere Dreiecksmatrizen handelt. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. Sie müssen die anfangs hergeleitete Formel verwenden. Zur Speicherung der Matrizen benutzen Sie die Struktur aus Aufgabe 9.1. Sind also A, B zwei Matrizen vom Typ `'L'`, dann soll auch C vom Typ `'L'` sein. Hat eine der Matrizen A, B den Typ `'F'` dann hat auch C den Typ `'F'`. Speichern Sie den Source-Code unter `matmult.c` in das Verzeichnis `serie09`.

Aufgabe 9.6. 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 $(a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$ 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 9.7. Sie kennen die Funktion `main` mit der Signatur `main()`. Man kann der Funktion `main` auch zusätzliche Eingabeparameter in der Kommandozeile übergeben. Die Signatur lautet dabei

```
int main (int argc, char *argv[]).
```

Finden Sie heraus was es damit auf sich hat und schreiben Sie ein Programm, dem beliebig viele Zahlen übergeben werden. Ihr Programm soll anschließend den Mittelwert all dieser Zahlen am Bildschirm ausgeben. Speichern Sie den Source-Code unter `mean.c` in das Verzeichnis `serie09`. Der Mittelwert der Zahlen $1, 2, 3$ soll also mittels Kommandozeilenaufruf

```
mean 1 2 3
```

bestimmt werden können.
Finden Sie weiters heraus welche Unterschiede zwischen der `main` Funktion in C und C++ besteht.

Aufgabe 9.8. *(C++, Strings)* Schreiben Sie ein Programm, welches ein Wort einliest und überprüft ob es sich bei dem eingegeben Wort um ein Palindrom handelt. Ein Palindrom ist ein Wort, welches von vorne und hinten gelesen gleich lautet, z.B.: Anna, Otto, Reliefpfeiler. Verwenden Sie in dem Beispiel den Datentyp `string`. Speichern Sie den Source-Code unter `palindrom.cpp` in das Verzeichnis `serie09`.