Dirk Praetorius,
Marcus Page,
Thomas Führer

Sommersemester 2013
30.04.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. Schreiben Sie einen Strukturdatentyp `cdouble`, in dem Realteil a und Imaginärteil b einer komplexen Zahl $a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Schreiben Sie Funktionen `cdouble* newCDouble(double a, double b)`, `delCDouble` sowie die vier Zugriffsfunktionen `setCDoubleReal`, `getCDoubleReal`, `setCDoubleImag` sowie `getCDoubleImag`. Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, in das Verzeichnis `serie07`.

Aufgabe 7.2. Schreiben Sie Funktionen `cadd, csub, cmul`, die die Addition, die Subtraktion und die Multiplikation für komplexe Zahlen $a+bi \in \mathbb{C}$ realisieren. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 7.1, und benutzen Sie beim Strukturzugriff nur die entsprechenden Zugriffsfunktionen. Schreiben Sie ein aufrufendes Hauptprogramm, in dem zwei komplexe Zahlen $w, z \in \mathbb{C}$ eingelesen werden und $w+z$, $w - z$, $w \cdot z$ sowie w/z ausgegeben werden. Speichern Sie den Source-Code unter `carithmetik.c` in das Verzeichnis `serie07`.
Bonus: Begründen Sie, warum die geschachtelte Verwendung obiger Funktionen vermieden werden sollte und geben Sie eine saubere Lösung zu nachfolgendem Code an:

```
cdouble* c1 = newCDouble(1,2);
cdouble* c2 = newCDouble(3,4);
cdouble* c3 = cadd(cmul(c1,c1),c2);
c1 = delCDouble(c1);
c2 = delCDouble(c2);
c3 = delCDouble(c3);
```

Aufgabe 7.3. Schreiben Sie einen Strukturdatentyp `squareMatrix` zur Speicherungen quadratischer Matrizen $A \in \mathbb{R}^{n\times n}$. Hierbei sollen die Einträge der Matrix spaltenweise als `double*`, sowie die Größe $n \in \mathbb{N}$ abgespeichert werden. Der Eintrag A_{ij} werde also an der Stelle `[i+j*n]` gespeichert. Orientieren Sie sich hierbei an den Folien 175 ff. der Vorlesung. Schreiben Sie außerdem alle nötigen Funktionen um mit dieser Struktur arbeiten zu können, d.h. implementieren Sie `newSquareMatrix, delSquareMatrix, getSquareMatrixDimension, getSquareMatrixEntry` und `setSquareMatrixEntry`. Speichern Sie den Source-Code unter `squareMatrix.c` in das Verzeichnis `serie07`..

Aufgabe 7.4. Eine Matrix $A \in \mathbb{R}^{n\times n}$ ist symmetrisch, falls $A_{jk} = A_{kj}$ für alle $j, k = 1, \dots, n$ gilt. Schreiben Sie eine Funktion `issymmetric`, die eine Matrix A auf Symmetrie überprüft (Rückgabewert `1` bei Symmetrie und `0` bei Nicht-Symmetrie). Schreiben Sie ein aufrufendes Hauptprogramm, in dem A eingelesen wird und ausgegeben wird, ob A symmetrisch ist oder nicht. Speichern Sie den Source-Code unter `issymmetric.c` in das Verzeichnis `serie07`.

Aufgabe 7.5. Und weil der gute Koarl am Übungstag seinen 236er feiern würde: Gegeben seien eine Matrix $A \in \mathbb{R}^{n\times n}$ und eine rechte Seite $b \in \mathbb{R}^n$. Ziel ist es ein Verfahren zu erstellen, mit dem das Gleichungssystem $Ax = b$ gelöst werden kann. Hierzu verwenden wir das *Gauß'schen Eliminationsverfahren*. Dies ist das Verfahren, dass man unbewusst verwendet, wenn man ein lineares Gleichungssystem händisch löst:

- 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 7.8.

Im ersten Eliminationsschritt zieht man von jeder anderen Zeile ein geeignetes Vielfaches der ersten Zeile

ab und erhält dadurch eine Matrix der Form

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

Mit dem gleichen Vielfachen multipliziert man auch den entsprechenden Eintrag des Vektors. Im zweiten Eliminationsschritt zieht man nun geeignete Vielfache der zweiten Zeile von den übrigen Zeilen ab (und von b) 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. 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. Schreiben Sie eine Funktion `void gauss(squareMatrix* A, Vector* b)` die obiges Verfahren realisiert und die modifizierten Werte von A und b „inplace“ speichert, d.h. der Orginalspeicher von A und b soll überschrieben werden.
Hinweis: Eine Struktur zum Umgang mit Vektoren steht auf der Homepage zum Download bereit. Speichern Sie den Source-Code unter `gauss.c` in das Verzeichnis `serie07`.

Aufgabe 7.6. Eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ hat höchstens $\frac{n(n+1)}{2} = \sum_{j=1}^{n} j$ nicht-triviale Einträge. Schreiben Sie eine Struktur `matrixU`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten U_{ij} in einem dynamischen Vektor der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Schreiben Sie die entsprechenden Zugriffsfunktionen (`newMatrixU`, `delMatrixU`, `getMatrixUDimension`, `getMatrixUij`, `setMatrixUij`), und überlegen Sie sich zuvor, an welcher Stelle u_ℓ im dynamischen Vektor ein Eintrag U_{ij} gespeichert werden soll. Tipp: Speichern Sie U spaltenweise.

Aufgabe 7.7. Schreiben Sie eine Funktion `int isUpperTriangular(squareMatrix* mat)` die überprüft ob es sich bei der übergebenen Matrix um eine obere Dreiecksmatrix handelt, d.h. ob `mat` von der Form

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

ist. Falls `mat` eine obere Dreiecksmatrix ist, so werde 1 zurückgegeben, ansonsten 0.
Schreiben Sie weiters eine Funktion `matrixU* convert(squareMatrix* mat)`, die eine gegebene quadratische Matrix `mat` (speichereffizient) als obere Dreiecksmatrix „abspeichert“, falls es sich um eine obere Dreiecksmatrix handelt. Überflüssige Nullen sollen also nicht mehr gespeichert werden. Die Dreiecksmatrix werde dann als `matrixU*` zurückgeliefert. Falls es sich bei `mat` nicht um eine obere Dreiecksmatrix handelt, so werde `NULL` zurückgegeben. Speichern Sie den Source-Code unter `convertSquareToUpper.c` in das Verzeichnis `serie07`.

Aufgabe 7.8. Es sei $U \in \mathbb{R}^{n\times n}$ eine obere Dreiecksmatrix mit $U_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebener rechter Seite $b \in \mathbb{R}^n$ existiert dann ein eindeutiger Vektor $x \in \mathbb{R}^n$ mit $Ux = b$. Leiten Sie, ausgehend von der Formel für die Matrix-Vektor-Multiplikation, eine Formel für x her. Schreiben Sie sich dazu das Matrix-Vektor-Produkt $b = Ux$ komponentenweise für b_j mit $j = 1, \dots, n$ als Summe, und überlegen Sie, wie die spezielle Gestalt von U die Laufindizes der Summe vereinfacht. Schreiben Sie eine Funktion `solveU`, die für gegebenes U und b den Vektor x berechnet und zurückgibt. Die Matrix U soll dabei in der Struktur aus Aufgabe 7.6 gespeichert werden, die Vektoren $b, x \in \mathbb{R}^n$ in der Struktur aus der Vorlesung.