Dirk Praetorius,
Thomas Führer,
Michael Feischl

Wintersemester 2013/14
11.12.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1. 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 11.2. 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 11.1 gespeichert werden, die Vektoren $b, x \in \mathbb{R}^n$ in der Struktur aus der Vorlesung.

Aufgabe 11.3. 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. 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 `serie11`..

Aufgabe 11.4. 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 `serie11`.

Aufgabe 11.5. 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

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

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

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

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). Speichern Sie den Source-Code unter `computeLU.c` in das Verzeichnis `serie11`.

Aufgabe 11.6. Was tut die folgende Funktion `func` bei Übergabe der Matrix

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 \\ 1 & 2 & 0 & 2 \\ 17 & 4 & 4 & 1 \end{pmatrix}?$$

A sei hierbei in der Struktur aus Aufgabe 11.3 gespeichert. Geben Sie tabellarisch wieder, welchen Wert die Variablen zum angegebenen Zeitpunkt haben. Welche Funktionalität wird durch `func` bereitgestellt? Was ist an dieser Lösung ineffizient realisiert und wie könnte man das effizienter gestalten?

```
int func(squareMatrix* mat) {
   double foo = 0;
   int mp, dp, tf;
   mp = 1;
   for (dp = 0; dp < getMatrixDim(mat); ++dp) {
      for (tf = dp+1; tf < getMatrixDim(mat); ++tf) {
         foo = getMatrixEntry(mat,dp,tf);
         if ( foo != 0 ) {
            mp = 0;
         }
      /* WERT DER VARIABLEN ZU DIESEM ZEITPUNKT */
      }
   }
   return mp;
}
```

Aufgabe 11.7. Schreiben Sie Funktionen `cadd, csub, cmul, cdiv`, die die Addition, die Subtraktion, die Multiplikation und die Division für komplexe Zahlen $a + bi \in \mathbb{C}$ realisieren. Weiters soll eine Funktion `double cnorm(cdouble* c)` programmiert werden, die das Betragsquadrat $|a + ib|^2 := a^2 + b^2$ zurückliefert. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 10.7, 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 `serie11`.
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 11.8. Schreiben Sie eine Struktur `CPoly` zur Speicherung von Polynomen mit komplexwertigen Koeffizienten, die bezüglich der Monombasis dargestellt sind, d.h. $p(x) = \sum_{j=0}^{n} a_j x^j$. Es sind also der Grad $n \in \mathbb{N}_0$ sowie der Koeffizientenvektor $(a_0, \ldots, a_n) \in \mathbb{C}^{n+1}$ zu speichern. Verwenden Sie für die Darstellung der komplexwertigen Koeffizienten den Strukturdatentyp aus Aufgabe 10.7. Schreiben Sie die ferner die nötigen Zugriffsfunktionen `newCPoly`, `delCPoly`, `getCPolyDegree`, `getCPolyCoefficient` und `setCPolyCoefficient`. Speichern Sie den Source-Code unter `cpoly.c` in das Verzeichnis `serie11`.