Dirk Praetorius,
Thomas Führer

Sommersemester 2014
05.05.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 8

Aufgabe 8.1. 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 `serie08`..

Aufgabe 8.2. 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 8.1 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 8.3. 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). Benutzen Sie die Struktur aus Aufgabe 8.1. 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 `serie08`.

Aufgabe 8.4. 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 `serie08`.

Aufgabe 8.5. 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 8.4, 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 `serie08`.
Bonus: Begründen Sie, warum die geschachtelte Verwendung obiger Funktionen vermieden werden sollte und geben Sie eine saubere Lösung zu nachfolgendem Code an (es handelt sich dabei um ein Problem mit dynamischem Speicher):

```
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 8.6. 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 8.4. Schreiben Sie die ferner die nötigen Zugriffsfunktionen `newCPoly`, `delCPoly`, `getCPolyDegree`, `getCPolyCoefficient` und `setCPolyCoefficient`. Speichern Sie den Source-Code unter `cpoly.c` in das Verzeichnis `serie08`.

Aufgabe 8.7. Schreiben Sie eine Funktion `addCpolynomials`, die die Summe $r = p+q$ zweier komplexer Polynome p und q (auch unterschiedlichen Grades) berechnet und zurückgibt. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 8.6. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem zwei Polynome p, q eingelesen und die Summe $r = p + q$ ausgegeben werden. Speichern Sie den Source-Code unter `addcpoly.c` in das Verzeichnis `serie08`.

Aufgabe 8.8. Schreiben Sie eine Struktur `CMatrix`, zur Speicherung von $(m \times n)$-Matrizen $A \in \mathbb{C}^{m \times n}$ mit komplexwertigen Koeffizienten. Benutzen Sie hierzu den Strukturdatentyp `cdouble` aus Aufgabe 8.4. Ferner schreibe man die zugehörigen Funktionen `newCMatrix`, `delCMatrix`, `getCMatrixM`, `getCMatrixN`, `getCMatrixCoeff`, `setCMatrixCoeff`.