Dirk Praetorius
Michele Ruggeri

Wintersemester 2014/15
25.11.2014

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Schreiben Sie eine Funktion `checkoccurrence`, die einen String s und einen Buchstaben b übernimmt und zurückgibt wie oft b in s vorkommt. Dabei zähle man sowohl das Vorkommen als Großbuchstabe als auch das Vorkommen als Kleinbuchstabe. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem s und b eingelesen und checkoccurrence aufgerufen werden. Speichern Sie den Source-Code unter `checkoccurrence.c` in das Verzeichnis `serie09`.

Aufgabe 9.2. Für eine konvergente Folge $(x_n)_{n\in\mathbb{N}}$ mit Grenzwert x spricht man von Konvergenzordnung $p \geq 1$, falls es eine Konstante $c > 0$ gibt mit $|x_n - x| \leq c\,|x_{n-1} - x|^p$ für alle $n \in \mathbb{N}$. Mit dem Ansatz

$$|x_{n+2} - x| = c|x_{n+1} - x|^p \quad \text{und} \quad |x_{n+1} - x| = c|x_n - x|^p \quad \text{für } n \in \mathbb{N}$$

kann man für fixiertes n die Unbekannten p und c bestimmen. Elementare Rechnung zeigt dann

$$p = \frac{\log(|x_{n+2} - x|/|x_{n+1} - x|)}{\log(|x_{n+1} - x|/|x_n - x|)} \quad \text{und} \quad c = \frac{|x_{n+2} - x|}{|x_{n+1} - x|^p},$$

d.h. aus den Gleichungen für die Fehler $|x_{n+2} - x|$ und $|x_{n+1} - x|$ berechnet man die Unbekannten $p_n := p$ und $c_n := c$. Leiten Sie diese Gleichheiten her! Schreiben Sie eine Funktion `convorder`, die für eine gegebene Folge $(x_n)_{n=1}^N$ und einen Grenzwert x die experimentelle Konvergenzrate $p, c \in \mathbb{R}^{N-2}$ berechnet und zurückgibt. — Üblicherweise ist x unbekannt und man hat nur eine Folge $(x_n)_{n=1}^N$ von Approximationen. In diesem Fall kann man die Funktion `convorder` mit der Teilfolge $(x_n)_{n=1}^{N-1}$ und $x := x_N$ verwenden. Speichern Sie den Source-Code unter `convorder.c` in das Verzeichnis `serie09`.

Aufgabe 9.3. 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`). Schreiben Sie außerdem eine `double` Funktion `evalPoly`, die ein Polynom an einer gegebenen Stelle auswertet und den Wert zurückgibt. Speichern Sie den Source-Code unter `polynomial.c` in das Verzeichnis `serie09`.

Aufgabe 9.4. 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 9.3. 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 `serie09`.

Aufgabe 9.5. Die k-te Ableitung $p^{(k)}$ eines Polynoms p ist wieder ein Polynom. Schreiben Sie eine Funktion `differentiatePolynomial`, die zu gegebenem p und $k \in \mathbb{N}$ die Ableitung $p^{(k)}$ berechnet. Zur Speicherung verwende man die Struktur aus Aufgabe 9.3. Zum Test schreibe man ein Hauptprogramm, das p und k einliest und $p^{(k)}$ ausgibt. Speichern Sie den Source-Code unter `differentiatePolynomial.c` in das Verzeichnis `serie09`.

Aufgabe 9.6. 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 `serie09`..

Aufgabe 9.7. 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 9.6 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 9.8. Eine Matrix $A \in \mathbb{R}^{n \times n}$ ist symmetrisch, falls $A_{jk} = A_{kj}$ für alle $j, k = 1, \ldots, n$ 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 9.6. 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 `serie09`.