Dirk Praetorius,
Thomas Führer,
Michael Feischl

Wintersemester 2013/14
05.11.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1. Gegeben sei ein Polynom $p(x) = \sum_{j=0}^{n} a_j x^j$ in Form seines Koeffizientenvektors $a = (a_0, \dots, a_n) \in \mathbb{R}^{n+1}$. Schreiben Sie eine Funktion `evalpolynomial`, die für gegebenen Koeffizientenvektor a und Auswertungspunkt x den Funktionswert $p(x)$ berechnet. Die Funktion `pow` zur Berechnung von x^j soll *nicht* verwendet werden. Schreiben Sie eine Funktion, die möglichst nur *eine* Schleife verwendet. Der Grad $n \in \mathbb{N}$ des Polynoms soll eine Konstante im Hauptprogramm sein, die Funktion `evalpolynomial` soll aber beliebigen Grad zulassen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Koeffizienten a_j sowie der Auswertungspunkt x eingelesen werden und $p(x)$ ausgegeben wird. Speichern Sie den Source-Code unter `evalPolynomial.c` in das Verzeichnis `serie06`.

Aufgabe 6.2. Das Produkt $r = pq$ zweier Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{j=0}^{n} b_j x^j$ ist wieder ein Polynom. Schreiben Sie eine Funktion `prodPoly`, die das Produktpolynom r berechnet. Überlegen Sie sich zunächst, welchen Grad das Polynom r hat und wie sich die Koeffizienten berechnen lassen. Ein Polynom $p(x) = \sum_{j=0}^{m} a_j x^j$ kann dabei als Vektor $a = (a_0, a_1, \dots, a_m) \in \mathbb{R}^{m+1}$ gespeichert werden. Der Einfachheit halber können Sie annehmen, dass $n = m$ gilt. Der Polynomgrad n soll eine fixe Konstante im Hauptprogramm sein. Schreiben Sie ein aufrufendes Hauptprogramm, in dem p und q eingelesen und $r = pq$ ausgegeben wird. Speichern Sie den Source-Code unter `prodPoly.c` in das Verzeichnis `serie06`.

Aufgabe 6.3. Schreiben Sie eine `void`-Funktion `diffpol` welche die Koeffizienten der Ableitung eines gegebenen Polynoms

$$p(x) = \sum_{j=0}^{n} a_j x^j \quad \text{mit} \quad p'(x) = \sum_{j=0}^{n-1} b_j x^j$$

berechnet. Welche Eingabeparameter benötigen Sie hierzu? Der Koeffizientenvektor soll schließlich ausgegeben werden. Testen sie ihren Code in einem Hauptprogramm. Dabei soll die Länge des Polynoms p eine Konstante in ihrem Hauptprogramm sein. Die Funktion `diffpol` soll aber für beliebiges n geschrieben werden. Speichern Sie den Source-Code unter `diffpol.c` in das Verzeichnis `serie06`.

Aufgabe 6.4. Die Frobeniusnorm einer Matrix $A \in \mathbb{R}^{M \times N}$ ist durch

$$\|A\|_F^2 := \sum_{j=1}^{M} \sum_{k=1}^{N} A_{jk}^2$$

definiert. Schreiben Sie eine Funktion `frobeniusnorm`, die für gegebene Matrix A und gegebene Größen $M, N \in \mathbb{N}$ die Frobeniusnorm berechnet und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem A eingelesen wird und $\|A\|_F^2$ ausgegeben wird. Die Größen M, N sollen dabei feste Konstanten im Hauptprogramm sein, die Funktion `frobeniusnorm` soll aber für beliebiges M, N programmiert werden. Die Matrix A soll dabei spaltenweise in einem Array gespeichert werden. Speichern Sie den Source-Code unter `frobeniusnorm.c` in das Verzeichnis `serie06`.

Aufgabe 6.5. Man erweitere *Selection Sort* aus der Vorlesung um einen Parameter `type`, sodass die Funktion einen Vektor $x \in \mathbb{R}^n$ wahlweise aufsteigend (`type = 1`) oder absteigend (`type = -1`) sortiert. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $x \in \mathbb{R}^n$ und die Sortierrichtung eingegeben werden und der sortierte Vektor ausgegeben wird. Die Länge n des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `sortvector` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `selectionsort.c` in das Verzeichnis `serie06`.

Aufgabe 6.6. Implementieren Sie den Quicksort-Algorithmus, um einen Vektor $x \in \mathbb{R}^n$ zu sortieren: Quicksort wählt willkürlich ein Pivotelement aus der zu sortierenden Liste x, z.B. x_1. Dann zerlegt man die Liste in zwei Teillisten: $x^{(<)}$ enthält alle Elemente $< x_1$, $x^{(\geq)}$ enthält alle Elemente $\geq x_1$. Diese Teillisten werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Es besteht (in der Reihenfolge) aus der unteren Liste $x^{(<)}$, dem Pivotelement und der oberen Liste $x^{(\geq)}$. — Eine direkte Implementation dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, geht man wie folgt vor: Beginnend mit $j = 2$ sucht man ein Element $x_j \geq x_1$, d.h. x_j gehört zu $x^{(\geq)}$. Ferner sucht man beginnend bei $k = n$ ein Element $x_k < x_1$, d.h. x_k gehört zu $x^{(<)}$. In diesem Fall vertauscht man x_j und x_k. Wenn sich die Zähler j und k treffen, liegt die Liste x bereits in der Form $(x^{(<)}, x_1, x^{(\geq)})$ vor. Es müssen nun nur noch die Teillisten sortiert werden. Speichern Sie den Source-Code unter `quicksort.c` in das Verzeichnis `serie06`.

Aufgabe 6.7. Schreiben Sie eine Funktion `exponential`, die den Funktionswert $\exp(x)$ approximativ berechnet: Dazu berechnen Sie die Partialsumme

$$S_N(x) := \sum_{j=0}^{N} \frac{x^j}{j!},$$

wobei die Summationsgrenze $N \in \mathbb{N}$ durch das Kriterium

$$\left|\frac{x^{N+1}}{(N+1)!}\right| \leq \left|\frac{x^N}{N!}\right| \leq \varepsilon$$

für eine gegebene Toleranz $\varepsilon > 0$ bestimmt werde. Intern realisiere man die Berechnung der Summationsglieder $x^j/j!$ möglichst rechenökonomisch. Vergleichen Sie den Fehler $|S_N(x) - \exp(x)|$ für verschiedene Wahlen von $\varepsilon > 0$ und Auswertungspunkten $x \in \mathbb{R}$. Speichern Sie den Source-Code unter `exponential.c` in das Verzeichnis `serie06`.

Aufgabe 6.8. Welchen Aufwand hat die Auswertung des Polynoms aus Aufgabe 6.1? Welchen Aufwand hat die Berechnung der Frobeniusnorm aus Aufgabe 6.4?