Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
23.10.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Aufgabe 4.1*. *Bubble-Sort* ist ein ineffizienter, aber kurzer Sortier-Algorithmus: Man vergleicht aufsteigend jedes Element eines Arrays x_j mit seinem Nachfolger x_{j+1} und - falls notwendig - vertauscht die beiden. Nach dem ersten Durchlauf muss zumindest das letzte Element bereits am richtigen Platz sein. Der nächste Durchlauf muss also nur noch bis zur vorletzten Stelle gehen, usw. Wie viele geschachtelte Schleifen braucht dieses Vorgehen? Schreiben Sie eine `void`-Funktion `bubblesort`, die ein gegebenes Array $x \in \mathbb{R}^n$ mittels Bubble-Sort aufsteigend sortiert, d.h. $x_1 \leq x_2 \leq \cdots \leq x_n$, und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor x einliest und in sortierter Reihenfolge ausgibt. Die Länge des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `bubblesort` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie04`.

Aufgabe 4.2*. Für $p \in [1, \infty)$ ist die ℓ_p-Norm auf $\mathbb{R}^n$ definiert durch

$$\|x\|_p := \Big(\sum_{j=1}^{n} |x_j|^p \Big)^{1/p}.$$

Schreiben Sie eine Funktion `pnorm`, die einen Vektor $x \in \mathbb{R}^5$ (d.h. 5 reelle Zahlen) sowie $p \in [1, \infty)$ übernimmt und $\|x\|_p$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm in dem x und p eingelesen werden und $\|x\|_p$ ausgegeben wird. Die p-te Wurzel $y^{1/p}$ einer Zahl $y > 0$ können Sie mit Hilfe der Funktion `pow` aus der Mathematik-Bibliothek berechnen. Die dritte Wurzel aus einer Zahl $y > 0$ können Sie beispielsweise mit `pow(y,1.0/3.0)` berechnen (Achtung auf eventuelles Type-Casting). Testen Sie Ihr Programm mit verschiedenen Werten. Speichern Sie den Source-Code unter `pnorm.c` in das Verzeichnis `serie04`.

Aufgabe 4.3*. Schreiben Sie eine Funktion `eratosthenes`, die das *Sieb des Eratosthenes* realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl `nmax` errechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Der Algorithmus sieht folgendermaßen aus:

- Man legt eine Liste (Vektor) $\texttt{prim} = (2, \ldots, \texttt{nmax}) \in \mathbb{N}^{\texttt{nmax}-1}$ an.
- Man streicht aus der Liste alle Vielfachen der ersten Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Am Ende sollen alle Primzahlen von $2, \ldots, \texttt{nmax}$ ausgegeben werden. Geben Sie außerdem die Anzahl der gefundenen Primzahlen mit aus. Realisieren Sie das Streichen indem Sie die entsprechenden Einträge auf 0 setzen. Die Zahl `nmax` soll eine feste Konstante im Hauptprogramm sein. Speichern Sie den Source-Code unter `eratosthenes.c` in das Verzeichnis `serie04`.

Aufgabe 4.4*. Schreiben Sie eine `void`-Funktion `matmult`, die eine Matrix-Matrix-Multiplikation realisiert. Die beiden Matrizen $A \in \mathbb{R}^{a\times b}$ und $B \in \mathbb{R}^{b\times c}$ sollen hierbei inklusive der Größen $a, b, c \in \mathbb{N}$ übergeben werden. Die Dimensionen sollen wieder Konstanten im Hauptprogramm sein. Verwenden Sie eine spaltenweise Speicherung der Matrizen. Das Ergebnis $C = AB$ soll schließlich ausgegeben werden. Es kann sinnvoll sein, die Ausgabe in eine eigene Funktion auszulagern. Speichern Sie den Source-Code unter `matmult.c` in das Verzeichnis `serie04`.

Aufgabe 4.5. Die Quotientenfolge $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ zur Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$,

$$a_0 := 1, \quad a_1 := 1, \quad a_n := a_{n-1} + a_{n-2} \quad \text{für } n \geq 2,$$

konvergiert gegen den goldenen Schnitt $(1 + \sqrt{5})/2$. Insbesondere konvergiert die Differenz

$$b_n := \frac{a_{n+1}}{a_n} - \frac{a_n}{a_{n-1}}$$

gegen Null. Schreiben Sie eine Funktion `cauchy`, die zu gegebenem $k \in \mathbb{N}$ die kleinste Zahl $n \in \mathbb{N}$ mit $|b_n| \leq 1/k$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahl $k \in \mathbb{N}$ einliest und den zugehörigen Index $n \in \mathbb{N}$ ausgibt. Zur Berechnung der Glieder der Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$ können Sie die Funktion aus Aufgabe 3.2 verwenden. Speichern Sie den Source-Code unter `cauchy.c` in das Verzeichnis `serie04`.

Aufgabe 4.6. 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 $a = (a_0, a_1, \ldots, a_m) \in \mathbb{R}^{m+1}$ 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 `serie04`.

Aufgabe 4.7. Man erweitere *MinSort* aus der Vorlesung (Folie 71-72) 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 `serie04`.

Aufgabe 4.8. Schreiben Sie Funktionen

```
nk = binomial(n,k)
```

die den Binomialkoeffizienten $\binom{n}{k}$ berechnen. Dies läßt sich auf verschiedene Weisen realisieren:

- direkt in der Form $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ unter Verwendung einer Funktion für die Faktorielle (siehe VO),
- in gekürzter Form $\binom{n}{k} = \frac{n\cdot(n-1)\cdots(n-k+1)}{k\cdot(k-1)\cdots 1}$ mittels geeigneter Schleifen,
- mittels einer rekursiven Funktion, die das Additionstheorem $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ benutzt.

Realisieren Sie alle 3 Varianten und zusätzlich ein Hauptprogramm, das die Zahlen n und k über die Tastatur einliest und den Binomialkoeffizien berechnet und ausgibt. Speichern Sie den Source-Code unter `binomial.c` in das Verzeichnis `serie04`.