Dirk Praetorius,
Marcus Page

Wintersemester 2011
30. Okt. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

Aufgabe 3.1*. 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) $\texttt{prim} = (2, \dots, \texttt{nmax}) \in \mathbb{N}^{\texttt{nmax}-1}$ 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.

Die Ausgabe soll am Ende gekürzt erfolgen. Geben Sie außerdem die Anzahl der gefundenen Primzahlen mit aus. Realisieren Sie das Streichen geeignet, z.B. indem Sie die entsprechenden Einträge auf 0 setzen. Vergleichen Sie dieses Vorgehen mit dem Primzahlentest des letzten Aufgabenzettels. Speichern Sie den Source-Code unter `eratosthenes.c` in das Verzeichnis `serie03`.

Aufgabe 3.2*. In vielen mathematischen Bibliotheken werden Matrizen $A \in \mathbb{R}^{m \times n}$ spaltenweise gespeichert, d.h. in Form eines Vektors $a \in \mathbb{R}^{mn}$, wobei $a_{j+km} = A_{jk}$ gilt, wenn die Indizierung (wie in C üblich) bei 0 beginnt. Schreiben Sie eine `void`-Funktion `mvmultiplication`, die die Matrix-Vektor-Multiplikation einer spaltenweise gespeicherten Matrix $A \in \mathbb{R}^{m \times n}$ mit einem Vektor $x \in \mathbb{R}^n$ realisiert und den Ergebnisvektor $b = Ax \in \mathbb{R}^m$ ausgibt. Die Dimensionen $m, n \in \mathbb{N}$ können Konstanten im Hauptprogramm sein, müssen aber als Parameter an die Funktion übergeben werden. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem A und x eingelesen werden `mvmultiplication` aufgerufen wird. Speichern Sie den Source-Code unter `mvmultiplaction.c` in das Verzeichnis `serie03`.

Aufgabe 3.3*. *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. $x_1 \leq x_2 \leq \dots \leq x_n$, 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 `serie03`.

Aufgabe 3.4*. In der Programmierung benötigt man häufig Zufallszahlen. Machen Sie sich im Internet über deren Einsatz in C schlau. Schreiben Sie nun eine `void`-Funktion `fillArray`, die ein aufsteigend sortiertes Array `x` der Länge 5 generiert. Hierbei soll komplett zufällig vorgegangen werden, d.h. Sie generieren so lange Zufallszahlen bis für jeden Index i `x[i+1]` $\geq$ `x[i]` gilt. Die Funktion `fillArray` soll

`void` sein, d.h. das Array soll nicht zurückgegeben werden, sondern nur ausgegeben. Speichern Sie den Source-Code unter `fillArray.c` in das Verzeichnis `serie03`.

Aufgabe 3.5. 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 wie in Aufgabe 3.2 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 `serie03`.

Aufgabe 3.6. Man erweitere *MinSort* aus der Vorlesung (S. 76-77) 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 `serie03`.

Aufgabe 3.7. 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. Speichern Sie den Source-Code unter `diffpol.c` in das Verzeichnis `serie03`.

Aufgabe 3.8. Schreiben Sie eine Funktion `power`, die für gegebene reelle Zahlen $x > 1$ und $C > 0$ die kleinste Zahl $n \in \mathbb{N}$ berechnet mit $x^n > C$. Dabei sollen die Funktionen `log` und `pow` nicht verwendet werden. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem x und C eingelesen werden und n ausgegeben wird.

Aufgabe 3.9. Die Summe $r = p + q$ zweier Polynome p, q ist wieder ein Polynom. Schreiben Sie eine Funktion `addPolynomials`, die die Summe r berechnet. Zum Test schreibe man eine Funktion, die zwei Polynome einliest und deren Summe ausgibt. Sie können für diese Aufgabe von fester Polynomlänge (gleicher Grad) ausgehen. Speichern Sie den Source-Code unter `addPolynomials.c` in das Verzeichnis `serie03`.

Aufgabe 3.10. Schreiben Sie eine `void`-Funktion `plotVec2`. Auch diese soll die Aufgabe erfüllen einen Vektor `x` $\in \mathbb{N}^5$ am Bildschirm auszugeben. Dieses mal soll die Ausgabe allerdings grafisch sein, d.h. der Vektor soll über seinen Indizes geplotted werden. Implementieren Sie die Funktion für die feste Länge 5, wobei die Einträge von `x` zwischen 0 und 7 liegen können. Für die Eingabe `x = (2, 5, 1, 3, 7)` soll die Ausgabe nun wie folgt aussehen:

```
7                 x
6
5     x
4
3           x
2  x
1        x
   1  2  3  4  5.
```

Da sie in der Bildschirmausgabe nicht mehr nach oben springen können, könnte es sinnvoll sein, den Vektor zuvor zu sortieren. Die Originalindizierung müssen Sie sich hierbei merken. Speichern Sie den Source-Code unter `plotVec2.c` in das Verzeichnis `serie03`. Eventuell hilft folgender Hinweis: Zeichenketten werden in C durch `char` Arrays realisiert (mehr dazu erfahren Sie später in der Vorlesung) und können mittels `strcat` aneinandergefügt werden.