Dirk Praetorius,
Gregor Gantner

Sommersemester 2018
10.04.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Aufgabe 4.1. Nach einer alten Legende gibt es in Hanoi einen Tempel, in dem sich folgende rituelle Anordnung befindet: Es handelt sich um 3 Pfosten und um 64 goldene Scheiben, die in der Mitte ein Loch haben, so dass sie auf die Pfosten gestapelt werden können. Die 64 Scheiben haben paarweise verschiedenen Durchmesser. Als der Tempel erbaut wurde, wurden diese Scheiben der Größe nach aufsteigend auf den ersten Pfosten gestapelt. Die Aufgabe der Priester in diesem Tempel besteht darin, diese Scheiben systematisch unter Zuhilfenahme des zweiten Pfostens so umzustapeln, dass sich diese am Schluss in derselben Anordnung wie zu Beginn auf dem dritten Pfosten befinden. Dabei sind folgende Regeln zu beachten:

- Die Scheiben dürfen nur einzeln verschoben werden.
- In jedem Schritt wird die oberste Scheibe eines Stapels auf einen der anderen Stapel gesetzt. D.h. es kann jeweils nur die erste Scheibe bewegt werden.
- Eine Scheibe darf nie auf einer anderen Scheibe kleineren Durchmessers zu liegen kommen.

Das Ende der Welt, so sagt die Legende, ist durch denjenigen Zeitpunkt vorherbestimmt, an dem die Priester diese Aufgabe erfüllt haben werden.
Sei n die Anzahl der Scheiben ($n = 64$ in der Legende). Ein rekursiver Algorithmus, der dieses Problem löst, lässt sich wie folgt formulieren: Um die obersten $m \leq n$ auf Pfosten i befindlichen Scheiben auf Pfosten j zu verschieben, verschiebt man

1. die obersten $m-1$ Scheiben von Pfosten i auf Pfosten $k \notin \{i, j\}$,
2. die grösste der besagten m Scheiben von Pfosten i auf Pfosten j,
3. und schliesslich die $m-1$ in Schritt 1 auf Pfosten k verschobenen Scheiben auf Pfosten j.

Die Wahl $m = n$, $i = 1$ und $j = 3$ löst das Problem. Schreiben Sie eine rekursive Funktion `void hanoi(int m, int i, int j)` die diesen Algorithmus implementiert. Jeder einzelne Schritt soll dabei am Display protokolliert werden. Zum Beispiel:

```
Eine Scheibe wandert vom 2. zum 3. Pfosten.
```

Schreiben Sie auch ein Hauptprogramm, das n über die Tastatur einliest und die Liste der Schritte ausgibt. Zum Testen verwende man $n \ll 64$, z.B. $n = 3, 4, 5$. Speichern Sie den Source-Code unter `hanoi.c` in das Verzeichnis `serie04`.

Aufgabe 4.2. Schreiben Sie eine rekursive Funktion `papierschnitt`, die alle Möglichkeiten visualisiert, wie ein Papierbogen der ganzzahligen Länge n in Papierbahnen der Länge 1 und 2 geschnitten werden kann. D.h. man stelle eine natürliche Zahl n auf alle möglichen Weisen als Summe $n = \sum_{j=1}^{k} \sigma_j$ mit Summanden $\sigma_j \in \{1, 2\}$ dar. Dabei soll die Reihenfolge beachtet werden. Für $n = 4$ gibt es beispielsweise 5 Möglichkeiten:

- $4 = 2 + 2$
- $4 = 2 + 1 + 1$
- $4 = 1 + 2 + 1$
- $4 = 1 + 1 + 2$

- $4 = 1 + 1 + 1 + 1$

Schreiben Sie ein aufrufendes Hauptprogramm, in dem n eingelesen wird und alle Möglichkeiten ausgegeben werden. Speichern Sie den Source-Code unter `papierschnitt.m` in das Verzeichnis `serie04`.

Aufgabe 4.3. Schreiben Sie die Funktion `int = binomial(int n, int k, int type)` die den Binomialkoeffizienten $\binom{n}{k}$ berechnet und zurückgibt. 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 (`type=1`),
- in gekürzter Form $\binom{n}{k} = \frac{n \cdot (n-1) \cdots (n-k+1)}{k \cdot (k-1) \cdots 1}$ mittels geeigneter Schleifen (`type=2`),
- mittels einer rekursiven Funktion, die das Additionstheorem $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ benutzt (`type=3`).

Realisieren Sie alle drei 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`.

Aufgabe 4.4. Eine natürliche Zahl $a \in \mathbb{N}$ ist eine Armstrong-Zahl, genau dann, wenn die Summe ihrer Ziffern, jeweils potenziert mit der Stellenanzahl der Zahl, wieder die Zahl selbst ergibt, d.h., $a := 10^{n-1} \cdot a_n + 10^{n-2} \cdot a_{n-1} + \ldots + 10^0 \cdot a_1$ mit $n \in \mathbb{N}$, $a_i \in \{0, 1, \ldots, 9\}$, $0 < i \leq n$, $a_n \neq 0$, ist eine Armstrong-Zahl, wenn $a_1^n + a_2^n + \ldots + a_n^n = a$. Die ersten Armstrong-Zahlen sind beispielsweise $1, 2, \ldots, 8, 9, 153, 370, \ldots$ Schreiben Sie eine `void`-Funktion `armstrong`, die eine untere Schranke $b \in \mathbb{N}$ und eine obere Schranke $c \in \mathbb{N}$ mit $b \leq c$ erhält und sämtliche Armstrong-Zahlen innerhalb dieser Schranken am Bildschirm ausgibt. Überprüfen Sie mittels `assert`, dass $b \leq c$. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Schranken eingelesen werden und die Funktion `armstrong` aufgerufen wird. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `armstrong.c` in das Verzeichnis `serie04`.

Aufgabe 4.5. Schreiben Sie ein main-Programm, das $n \in \mathbb{N}$ über die Tastatur einliest und die ersten n Stufen des Pascal'schen Dreiecks ausgibt: Jede Zeile dieses Schemas beginnt und endet mit 1. Die restlichen Zahlen werden als Summe nebeneinanderstehender Zahlen der vorhergegangenen Zeile gebildet. Für $n = 5$ sieht das Pascal'sche Dreieck dann folgendermaßen aus:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

Siehe auch:

`http://de.wikipedia.org/wiki/Pascalsches_Dreieck`

Speichern Sie den Source-Code unter `pascal.c` in das Verzeichnis `serie04`.

Aufgabe 4.6. Schreiben Sie eine Funktion `energie`, die für einen gegebenen Vektor $x \in \mathbb{R}^n$ die Energie $e = \sum_{j=1}^{n} x_j^2$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor x einliest und die Energie ausgibt. Die Dimension $n \in \mathbb{N}$ soll eine Konstante im Hauptprogramm sein, die Funktion `energy` ist für beliebige Dimension zu programmieren. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `energie.c` in das Verzeichnis `serie04`.

Aufgabe 4.7. Die Frobeniusnorm einer Matrix $A \in \mathbb{R}^{m \times n}$ ist durch

$$\|A\|_F := \Big(\sum_{j=1}^{m} \sum_{k=1}^{n} A_{jk}^2 \Big)^{1/2}$$

definiert. Schreiben Sie eine Funktion `frobeniusnorm`, die für gegebene Matrix A und gegebene Dimensionen $m, n \in \mathbb{N}$ die Frobeniusnorm berechnet und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Zeilen- und Spaltendimensionen $m, n \in \mathbb{N}$ und A eingelesen werden und

$\|A\|_F$ ausgegeben wird. Die Matrix A soll spaltenweise gespeichert werden. Die Dimensionen $m, n \in \mathbb{N}$ sollen zwei Konstanten im Hauptprogramm sein, die Funktion `frobeniusnorm` ist aber für beliebige Dimensionen zu programmieren. Speichern Sie den Source-Code unter `frobeniusnorm.c` in das Verzeichnis `serie04`.

Aufgabe 4.8. Gegeben sei eine 10-stellige Zahlenfolge x (statisches Array vom Typ `int`) und eine 3-stellige Zahlenkombination y (Array vom Typ `int`), die jeweils von der Tastatur eingelesen werden. Man schreibe eine Funktion `check`, die die beiden Arrays bekommt und überprüft, ob die Zahlenkombination y in der Zahlenfolge x vorkommt (Rückgabewert 1), oder nicht (Rückgabewert 0). Zusätzlich schreibe man ein aufrufendes Hauptprogramm, in dem die Felder x und y eingelesen werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `check.c` in das Verzeichnis `serie04`.