Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
16.10.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

Aufgabe 3.1*. Schreiben Sie eine Funktion `geometricMean`, die von einem gegebenem Vektor $x \in \mathbb{R}^n_{\geq 0}$ den geometrischen Mittelwert

$$\overline{x}_{\text{geom}} = \sqrt[n]{\prod_{j=1}^{n} x_j}$$

berechnet und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor $x \in \mathbb{R}^n$ einliest und den geometrischen Mittelwert ausgibt. Die Länge $n \in \mathbb{N}$ des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `geometricMean` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `geometricMean.c` in das Verzeichnis `serie03`.

Aufgabe 3.2*. Die Fibonacci-Folge ist definiert durch $x_0 := 0$, $x_1 := 1$ und $x_{n+1} := x_n + x_{n-1}$. Schreiben Sie eine nicht-rekursive Funktion `fibonacci(k)`, die zu gegebenem Index k das Folgenglied x_k berechnet und zurückgibt. Schreiben Sie ferner ein Hauptprogramm, das k von der Tastatur einliest und x_k am Bildschirm ausgibt. Speichern Sie den Source-Code unter `fibonacci.c` in das Verzeichnis `serie03`. Vergleichen Sie diese nicht-rekursive Funktion mit der rekursiven Funktion `fibonacciRek` aus der letzten Übung.

Aufgabe 3.3*. 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 3.4*. Schreiben Sie eine Funktion `pol2cart`, die gegebene Polarkoordinaten (r, φ) in kartesische Koordinaten umrechnet. Verwenden Sie hierbei die mathematische Bibliothek. Speichern Sie den Source-Code unter `pol2cart.c` in das Verzeichnis `serie03`.

Aufgabe 3.5. Sie legen ihr Kapital bei ihrer Hausbank zu einem fixen Jahreszinssatz an. Schreiben Sie eine Funktion `endkapital` welches aus den Werten Laufzeit $n \in \mathbb{N}$, Jahreszinssatz p (in Prozent %), und Startkapital $x \in \mathbb{R}_{\geq 0}$ das Endkapital nach n Jahren zurückgibt. Dabei soll die Funktion ihren Kontostand in folgender Form

```
Jahr    Kapital
====    =======
 0      1000.00
 1      1010.00
 2      1020.10
 3      1030.30
..      .......
10      1104.62
```

ausgeben. (Bei diesem Beispiel wurde ein Zinssatz von $p = 1$ gewählt.) Schreiben Sie ferner eine Funktion `laufzeit` welche berechnet wie lange Sie mindestens Ihr Startkapital x bei einem Zinssatz p anlegen müssen um ein Endkapital von mindestens $x_{\max}$ zu haben. Die Funktion soll x, p, und $x_{\max}$ als Eingabe erhalten. Weiters schreiben Sie ein Hauptprogramm welches die beiden Funktionen testet. Wie lange müssen Sie warten um Euromillionär zu werden, wenn Sie $x = 1000$ Euro bei einem Fixzinssatz von $p = 4$ anlegen? Speichern Sie den Source-Code unter `kapital.c` in das Verzeichnis `serie03`.

Aufgabe 3.6. Schreiben Sie eine `void` Funktion `plotRectangle`, die zu gegebenen Seitenlängen $a, b \in \mathbb{N}$ ein Rechteck auf dem Bildschirm ausgibt. Das Rechteck soll entsprechend den a und b skaliert werden, so dass wir bei `plotRectangle(3,4)` etwa folgende Ausgabe erhalten:

```
xxxx
xxxx
xxxx
```

Die Eingabe `plotRectangle(2,10)` würde diese Ausgabe liefern:

```
xxxxxxxxxx
xxxxxxxxxx
```

Schreiben Sie außerdem ein aufrufendes Hauptprogramm, welches die Seitenlängen a und b von der Tastatur einliest. Speichern Sie den Source-Code unter `plotRectangle.c` in das Verzeichnis `serie03`.

Aufgabe 3.7. Implementieren Sie folgendes Computerspiel. Der Computer merke sich eine zufällige Zahl zwischen 0 und 15. Sie haben maximal drei Versuche um die richtige Zahl zu erraten. Geben Sie beim ersten oder zweiten Versuch eine falsche Zahl an, soll der Computer Ihnen mitteilen, ob die angegebene Zahl größer oder kleiner als die gesuchte Zahl ist. Wenn man auch beim dritten Versuch daneben liegt, soll die richtige Zahl angezeigt werden. Zufallszahlen zwischen 0 und 15 können Sie folgendermaßen erzeugen: Zunächst binden Sie die Headerdateien `stdlib.h` und `time.h` in Ihr Programm ein. Danach können Sie in einer beliebigen Funktion mit

```
srand( (unsigned) time(NULL) );
zufallszahl = (int) (16.0*rand()/(RAND_MAX+1.0));
```

eine Zufallszahl zwischen 0 und 15 generieren. Die Variable `zufallszahl` ist dabei vom Typ `int`. Speichern Sie den Source-Code unter `spiel.c` in das Verzeichnis `serie03`.

Aufgabe 3.8. 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 $n = 64$ 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.
- 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.
Ein rekursiver Algorithmus, der dieses Problem löst, lässt sich wie folgt formulieren: Um die obersten m 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.

Man beginnt mit $m = n$, $i = 1$ und $j = 3$. Man entwickle eine rekursive Funktion, die diesen Algorithmus implementiert. Jeder einzelne Schritt soll dabei am Display protokolliert werden. Zum Beispiel:

```
Scheibe 1 wandert vom 2. zum 3. Pfosten
```

Zum Testen verwende man $n \ll 64$, z.B. $n = 3, 4, 5$. Speichern Sie den Source-Code unter `hanoi.c` in das Verzeichnis `serie03`.