Dirk Praetorius,
Marcus Page

Wintersemester 2011
21. Okt. 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 2

Aufgabe 2.1*. Schreiben Sie eine Funktion `evalpol`, die zu gegebenen Koeffizienten $a_1, a_2, a_3, a_4 \in \mathbb{R}$ und einer Auswertungsstelle $x \in \mathbb{R}$ den Wert des zugehörigen Polynoms

$$p(x) := \sum_{i=1}^{4} a_i x^i$$

zurückliefert. Verwenden Sie hierzu den Befehl `pow` aus der mathematischen Bibliothek. Schreiben Sie ferner ein aufrufendes Hauptprogramm, welches a_1, a_2, a_3, a_4 und x einliest und `evalpol` aufruft. Speichern Sie den Source-Code unter `evalpol.c` in das Verzeichnis `serie02`.

Aufgabe 2.2*. Schreiben Sie eine `void`-Funktion `vielfache(k,nmax)`, die alle ganzzahligen Vielfachen der Zahl $k \in \mathbb{N}$, die $\leq n_{\max} \in \mathbb{N}$ sind, am Bildschirm ausgibt. Die Ausgabe erfolge zeilenweise in der Form

```
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
```

beispielsweise für den Fall $k = 5$ und $n_{\max} = 19$. Ferner schreibe man ein Hauptprogramm, das die Daten k und n von der Tastatur einliest und `vielfache(k,nmax)` aufruft. Speichern Sie den Source-Code unter `vielfache.c` in das Verzeichnis `serie02`.

Aufgabe 2.3*. Schreiben Sie eine Funktion `isSorted`, der ein Vektor x der Länge n mit Einträgen $x_i \in \mathbb{N}$ übergeben wird. Die Funktion soll nun untersuchen, ob x sortiert ist und abängig von der Art der Sortierung folgende Ausgabe (Rückgabewert) liefern:

- -1, falls x unsortiert ist,
- 0, falls alle Einträge von x gleich sind,
- 1, falls x aufsteigend sortiert ist,
- 2, falls x absteigend sortiert ist.

Machen Sie sich vor der Implementierung Gedanken darüber, wie Sie die einzelnen Abfragen schachteln wollen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, welches x von der Tastatur einliest. Die Länge n soll eine Konstante im Hauptprogramm sein. Die Funktion selbst soll jedoch für variable Längen geschrieben werden, wobei die Länge als Parameter übergeben wird. Speichern Sie den Source-Code unter `isSorted.c` in das Verzeichnis `serie02`.

Aufgabe 2.4*. Schreiben Sie eine `void` Funktion `plotVec`, deren Aufgabe es ist, einen gegebenen Vektor am Bildschirm auszugeben. Für einen Vektor $x = (3, 2, 6, 19, 3, 12)$ soll die Ausgabe das folgende Format haben:

$x1 = 3$
$x2 = 2$
$x3 = 6$
$x4 = 19$
$x5 = 3$
$x6 = 12.$

Der Funktion soll nun der Vektor x und dessen Länge $n \in \mathbb{N}$ übergeben werden. Die Länge n soll in einem aufrufenden Hauptprogramm wiederum konstant sein. Speichern Sie den Source-Code unter `plotVec.c` in das Verzeichnis `serie02`.

Aufgabe 2.5. Schreiben Sie eine Funktion `vecadd`, der zwei Vektoren $x \in \mathbb{R}^a$ und $y \in \mathbb{R}^b$ und deren Längen a und b übergeben werden. Sind beide Vektoren gleich lang, so soll die Funktion die Summe $z = x + y$, d.h.

$$z_i = x_i + y_i \quad \text{für jedes } i = 1 \dots a$$

ausgeben (verwenden Sie `plotVec` aus Aufgabe 2.4) und falls $a \neq b$ eine entsprechende Meldung. Schreiben Sie außerdem ein aufrufendes Hauptprogramm, in dem die Vektoren x und y von der Tastatur eingelesen werden. Die Längen a und b sollen wieder Konstanten im Hauptprogramm sein. Speichern Sie den Source-Code unter `vecadd.c` in das Verzeichnis `serie02`.

Aufgabe 2.6. Schreiben Sie einen Primzahlentest. Dieser soll (relativ ineffizient) wie folgt arbeiten: Sie lesen eine Zahl $p \in \mathbb{N}$ von der Tastatur ein. Ihr Programm soll nun für alle Zahlen $a \leq p$ prüfen, ob

$$p \mod a \not\equiv 0$$

gilt. Ist dies für alle $a \leq p$ der Fall, so ist p prim und eine entsprechende Meldung wird ausgegeben. Ansonsten soll der kleinste Teiler zurückgegeben werden. Speichern Sie den Source-Code unter `isPrime.c` in das Verzeichnis `serie02`. Wieso ist dieser Test nicht effizient? Was würden Sie verbessern?

Aufgabe 2.7. 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 `serie02`.

Aufgabe 2.8. Schreiben Sie eine Funktion `mean`, die von einem gegebenem Vektor $x \in \mathbb{N}^n$ natürlicher Zahlen den Mittelwert $\frac{1}{n}\sum_{j=1}^{n} x_j$ berechnet und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor $x \in \mathbb{N}^n$ einliest und den Mittelwert ausgibt. Die Länge $n \in \mathbb{N}$ des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `mean` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `mean.c` in das Verzeichnis `serie02`.

Aufgabe 2.9. 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 $x \in \mathbb{R}^n$, dessen Länge n 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 Dimension $n \in \mathbb{N}$ soll eine Konstante im Hauptprogramm sein, die Funktion `pnorm` soll aber beliebige Dimension zulassen. Testen Sie Ihr Programm mit verschiedenen Werten für p bei festem Vektor x. Was beobachten Sie für $p \to \infty$? Speichern Sie den Source-Code unter `pnorm.c` in das Verzeichnis `serie02`.

Aufgabe 2.10. 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 `serie02`.