Dirk Praetorius,
Marcus Page

Sommersemester 2012
26.03.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Aufgabe 4.1*. (Pointer, Arrays) Schreiben Sie eine `void`-Funktion `meansum`, die zu einem gegebenen Vektor $x \in \mathbb{R}^n$ sowohl die Summe $s = \sum_{i=1}^n x_i$, als auch das harmonische Mittel

$$x_{\text{harm.}} := \frac{n}{\sum_{i=1}^n \frac{1}{x_i}}$$

berechnet. Schreiben Sie außerden ein aufrufendes Hauptprogramm in dem der Vektor x eingelesen und s und $x_{\text{harm.}}$ ausgegeben werden. In der Funktion `meansum` selbst soll keine Ausgabe erfolgen. Speichern Sie den Source-Code unter `meansum.c` in das Verzeichnis `serie04`.

Aufgabe 4.2*. (Gleitkommadarstellung, Schleifen) Schreiben Sie eine Funktion `float2dec`, die für eine gegebene Mantissenlänge $M \in \mathbb{N}$, Ziffern $a_1, \dots, a_M \in \{0, 1\}$ und einen Exponenten $e \in \mathbb{Z}$ den Dezimalwert $x = \big(\sum_{k=1}^M a_k 2^{-k}\big) 2^e$ berechnet und zurückgibt. Die Mantissenlänge M soll hierbei eine Konstante im Hauptprogramm sein. Schreiben Sie ein aufrufendes Hauptprogramm, in dem a_j und e eingelesen und der Wert x ausgegeben wird. Realisieren die Potenzen 2^{-k} möglichst rechenökonomisch. Speichern Sie den Source-Code unter `float2dec.c` in das Verzeichnis `serie04`.

Aufgabe 4.3*. (Gleitkommadarstellung, Schleifen) Schreiben Sie eine Funktion `dec2float`, die für eine gegebene Dezimalzahl $x \in \mathbb{R}_{>0}$ und eine Mantissenlänge $M \in \mathbb{N}$ die Ziffern $a_1, \dots, a_M \in \{0, 1\}$ und den Exponenten $e \in \mathbb{Z}$ der normalisierten Gleitkommadarstellung (d.h. $a_1 = 1$) berechnet und am Bildschirm ausgibt. Schreiben Sie ein aufrufendes Hauptprogramm, in dem x eingelesen wird. Um Ihre Funktion zu verifizieren, können Sie Aufgabe 4.2 verwenden. Speichern Sie den Source-Code unter `dec2float.c` in das Verzeichnis `serie04`.
Hinweis: Durch die feste Mantissenlänge $M \in \mathbb{N}$ kann x evtl. nur näherungsweise wiedergegeben werden.

Aufgabe 4.4*. (Arrays, Schleifen) Gegeben sei ein Polynom $p(x) = \sum_{j=0}^n a_j x^j$ in Form seines Koeffizientenvektors $a = (a_0, \dots, a_n) \in \mathbb{R}^{n+1}$. Schreiben Sie eine Funktion `evalpolynomial`, die für gegebenen Koeffizientenvektor a und Auswertungspunkt x den Funktionswert $p(x)$ berechnet. Die Funktion `pow` zur Berechnung von x^j soll *nicht* verwendet werden. Schreiben Sie eine Funktion, die möglichst nur *eine* Schleife verwendet. Der Grad $n \in \mathbb{N}$ des Polynoms soll eine Konstante im Hauptprogramm sein, die Funktion `evalpolynomial` soll aber beliebigen Grad zulassen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Koeffizienten a_j sowie der Auswertungspunkt x eingelesen werden und $p(x)$ ausgegeben wird. Speichern Sie den Source-Code unter `evalPolynomial.m` in das Verzeichnis `serie04`.

Aufgabe 4.5. (Arrays, Schleifen) Schreiben Sie eine Funktion `evalDiffPoly`, die für ein gegebenes Polynom p, eine Ableitungsordnung $k \in \mathbb{N}_0$ und einen Punkt $x \in \mathbb{R}$ den Funktionswert $p^{(k)}(x)$ zurückgibt. Dabei soll das Polynom $p^{(k)}$ nicht explizit gebildet und gespeichert werden. Das Polynom p ist, wie zuvor, anhand seines Koeffizientenvektors gegeben. Der Grad $n \in \mathbb{N}$ des Polynoms soll wieder eine Konstante im Hauptprogramm sein, die Funktion `evalDiffPoly` soll jedoch für beliebige Grade implementiert werden.

Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem das Polynom p, die Ableitungsordnung k und der Punkt x eingelesen werden und $p^{(k)}(x)$ ausgegeben wird. Berücksichtigen Sie explizit den Fall, dass $k > grad(p)$ ist. Speichern Sie den Source-Code unter `evalDiffPoly.c` in das Verzeichnis `serie04`. **Hinweis:** Machen Sie sich zunächst mathematisch (auf einem Zettel) klar, was Sie eigentlich implementieren wollen.

Aufgabe 4.6. (Gleitkommadarstellung) In der Vorlesung haben Sie gelernt, dass es zu jeder Zahl $x \in \mathbb{R}$

- ein Vorzeichen $\sigma \in \{\pm 1\}$,
- Ziffern $a_j \in \{0, 1\}$ und
- einen Exponenten $e \in \mathbb{Z}$

gibt, so dass

$$x = \sigma\big(\sum_{k=1}^{\infty} a_k 2^{-k}\big)2^e$$

gilt (Folie 95). Machen Sie sich den Beweis dieser Aussage klar und geben Sie ihn in eigenen Worten wieder. Diese Darstellung ist im Allgemeinen nicht eindeutig. Sie kann aber in einem Gleitkommazahlensystem $\mathbb{F}(2, M, e_{\min}, e_{\max})$ durch feste Mantissenlänge $M \in \mathbb{N}$ und Normierung $a_1 = 1$ eindeutig gemacht werden. Warum? Beweisen Sie diese Aussage. Was wissen Sie über das implizite erste Bit?

Aufgabe 4.7. (Gleitkommadarstellung, Zahlensystem) In der Vorlesung haben Sie ferner gelernt, dass Brüche deren Nenner keine Zweierpotenz ist, in der Darstellung aus Aufgabe 4.6 nicht exakt wiedergegeben werden können. Wieso ist das der Fall? Was versteht man unter Über- und Unterlauf? Was sind *numerisch schlecht gestellte Probleme*? Was bedeuten diese Probleme für die Implementierung?

Aufgabe 4.8. (Pointer) Schreiben Sie eine Funktion `smallerBigger`, die zu einem gegebenen Vektor $x \in \mathbb{R}^n$ die Anzahl derjenigen Einträge $x_i \geq 0$ und die Anzahl der Einträge $x_i < 0$ zurückliefert. Die Länge des Vektors soll auch hier eine Konstante im Hauptprogramm sein, während die Funktion selbst für beliebige Längen zu implementieren ist. Schreiben Sie ferner ein aufrufendes Hauptprogramm in dem x eingelesen und die entsprechenden Anzahlen ausgegeben werden. Speichern Sie den Source-Code unter `smallerBigger.c` in das Verzeichnis `serie04`.

Aufgabe 4.9. (Pointer) Was ist der Unterschied und der Zusammenhang zwischen einer Variable und einem Pointer? Was könnten Vor- und Nachteile dieser Konstrukte sein?
Schreiben Sie eine Funktion `swap`, welche die Werte zweier Zahlen a und b vertauscht. Warum funktioniert das folgende Vorgehen nicht?

```
void swap(double x, double y)
{
   double tmp;
   tmp = x;
   x = y;
   y = tmp;
}
```

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

Aufgabe 4.10. (Schleifen) Für eine gewisse Anzahl an Schritten gilt die Darstellung

$$\begin{aligned} 1 \cdot 9 + 2 &= 11 \\ 12 \cdot 9 + 3 &= 111 \\ 123 \cdot 9 + 4 &= 1111 \\ &\vdots \end{aligned}$$

Schreiben Sie ein Programm `mathFun` welches die obigen Gleichheiten verifiziert und den Schritt $i \in \mathbb{N}$ berechnet, ab dem die Darstellung nichtmehr gilt. Hierzu sollen die Terme Schritt für Schritt berechnet und am Bildschirm ausgegeben werden. In Schritt 3, soll die Ausgabe also etwa das Format

```
123*9+4 = 1111
```

haben. Falls keine Gleichheit mehr gilt, so ist eine entsprechende Meldung auszugeben. Speichern Sie den Source-Code unter `mathFun.c` in das Verzeichnis `serie04`.
Hinweis: Machen Sie sich zunächst mathematisch klar, was hier zu implementieren ist.