Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
16./17.11.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Aufgabe 4.1. Schreiben Sie eine Funktion `kgV`, die zu zwei gegebenen natürlichen Zahlen $a, b \in \mathbb{N}$ das kleinste gemeinsame Vielfache berechnet und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahlen $a, b \in \mathbb{N}$ einliest und das dazugehörige kleinste gemeinsame Vielfache ausgibt. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `kgV.c` in das Verzeichnis `serie04`.

Aufgabe 4.2. Schreiben Sie eine `void`-Funktion `printUnique`, die von einem gegebenem Vektor $x \in \mathbb{R}^n$ alle Elemente, die nur einmal im Vektor vorkommen, am Bildschirm ausgibt. Ferner schreibe man ein aufrufendes Hauptprogramm, das den Vektor x einliest und die Funktion aufruft. Die Länge des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `printUnique` ist für beliebige Länge n zu programmieren. Überprüfen Sie mittels `assert`, dass $n \in \mathbb{N}$. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `printUnique.c` in das Verzeichnis `serie04`.

Aufgabe 4.3. Schreiben Sie eine `void`-Funktion `kreuz`, die für ein $n \in \{1, 2, \dots, 9\}$ ein $2n - 1$ Zeilen großes "X" am Bildschirm ausgibt. Für $n = 6$ sollen Sie beispielsweise folgenden Output am Bildschirm erhalten:

```
1          1
 2        2
  3      3
   4    4
    5  5
     66
    5  5
   4    4
  3      3
 2        2
1          1
```

Überprüfen Sie mittels `assert`, dass $n \in \{1, 2, \dots, 9\}$. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem der Parameter n eingelesen wird und die Funktion `kreuz` aufgerufen wird. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `kreuz.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 $n \in \mathbb{N}$, $a_i \in \{0, 1, \dots, 9\}$, $0 < i \leq n$, $a_n \neq 0$, ist eine Armstrong-Zahl, wenn $a_1^n + a_2^n + \ldots + a_n^n = a$. Die ersten Armstrong-Zahlen sind beispielsweise $1, 2, \dots, 8, 9, 153, 370, \dots$
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 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 4.6. 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`.

Aufgabe 4.7. Die Cosinus-Funktion hat die Darstellung $\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$. Wir betrachten die Partialsummen

$$C_n(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{(2k)!}.$$

Schreiben Sie eine *nicht-rekursive* Funktion `cos_new`, die für gegebene $x \in \mathbb{R}$ und $\tau > 0$ den Wert $C_n(x)$ zurückliefert, sobald

$$|C_n(x) - C_{n-1}(x)|/|C_n(x)| \le \tau \quad \text{oder} \quad |C_n(x)| \le \tau$$

gilt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem $x \in \mathbb{R}$ und $\tau > 0$ eingelesen werden. Neben dem berechneten Wert $C_n(x)$ sollen auch der korrekte Wert $\cos(x)$ und der absolute Fehler $|C_n(x) - \cos(x)|$ ausgegeben werden sowie der relative Fehler $|C_n(x) - \cos(x)|/|\cos(x)|$ im Fall $\cos(x) \neq 0$. Schreiben Sie die Funktion möglichst so, dass diese mit einer Schleife auskommt und dass x^{2k} und $(2k)!$ möglichst kostensparend realisiert werden. Man vermeide also insbesondere (vor- oder selbst implementierte) Funktionen zur Berechnung der Potenz oder der Faktoriellen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `cos.c` in das Verzeichnis `serie04`.

Aufgabe 4.8. Für $x > 0$ konvergiert die Folge

$$x_1 := \frac{1}{2}(1 + x), \quad x_{n+1} := \frac{1}{2}\Big(x_n + \frac{x}{x_n}\Big) \quad \text{für } n \ge 1$$

gegen $\sqrt{x}$. Schreiben Sie eine *nicht-rekursive* Funktion `sqrt_new`, die für gegebene $x > 0$ und $\tau > 0$ als Ergebnis das erste Folgenglied $y = x_n$ zurückgibt, für das gilt

$$\frac{|x_n - x_{n+1}|}{|x_n|} \le \tau \quad \text{oder} \quad |x_n| \le \tau.$$

Überprüfen Sie mittels `assert`, dass $x > 0$. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem x eingelesen und neben der Approximation x_n von $\sqrt{x}$ auch der exakte Wert sowie der absolute Fehler $|x_n - \sqrt{x}|$ ausgegeben werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `sqrt_new.c` in das Verzeichnis `serie04`.