

## Übungen zur Vorlesung Einführung in das Programmieren für TM

### Serie 7

Die Aufgaben mit Stern (\*) sind bis zur nächsten Übung vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung und mir als zusätzliche Grundlage für den Prüfungsstoff in den schriftlichen Tests. Kopieren Sie bitte den Source-Code in ein Unterverzeichnis `serie07` Ihres Home-Verzeichnisses. In den folgenden Aufgaben sollen **rekursive Funktionen** geübt werden.

**Aufgabe 61\*.** Unter einer **rekursiven Funktion** versteht man eine Funktion, die sich selber aufruft, zusammen mit einer Abbruchbedingung. Das folgende Beispiel ist Ihnen allen bekannt: Für  $n \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}$  definieren wir  $f(0) = 1$  und  $f(n) := n \cdot f(n-1)$  für  $n \geq 1$ . Um welche Funktion handelt es sich? Man schreibe sich dazu beispielsweise alle Faktoren von  $f(5)$  hin:  $f(5) = 5 \cdot f(4) = 5 \cdot 4 \cdot f(3) \dots$ . Implementieren Sie diese rekursive Funktion und schreiben Sie ein aufrufendes Hauptprogramm, das  $n \in \mathbb{N}_0$  einliest und  $f(n)$  ausgibt. Wählen Sie einen geeigneten Namen für den Source-Code und speichern Sie ihn ins Verzeichnis `serie07`.

**Aufgabe 62\*.** Die Fibonacci-Folge ist definiert durch  $x_0 := 0$ ,  $x_1 := 1$  und  $x_{n+1} := x_n + x_{n-1}$ . Man schreibe eine rekursive Funktion `fibonacci`, die zu gegebenem Index  $n$  das Folgenglied  $x_n$  zurückgibt. Ferner schreibe man ein aufrufendes Hauptprogramm, das den Index  $n$  einliest und  $x_n$  ausgibt. Speichern Sie den Source-Code unter `fibonacci.c` in das Verzeichnis `serie07`. Warum ist die induktive Implementierung über eine Zählschleife effizienter als diese rekursive Implementierung?

**Aufgabe 63\*.** Man schreibe eine Funktion `merge`, die zwei aufsteigend sortierte Felder  $a \in \mathbb{R}^m$  und  $b \in \mathbb{R}^n$  so vereinigt, dass das resultierende Feld  $c \in \mathbb{R}^{m+n}$  ebenfalls aufsteigend sortiert ist, z.B. soll  $a = (1, 3, 3, 4, 7)$  und  $b = (1, 2, 3, 8)$  als Ergebnis  $c = (1, 1, 2, 3, 3, 3, 4, 7, 8)$  liefern. Schreiben Sie die Funktion so, dass neben dem Base-Pointer des Vektors  $c$  die Längen  $m, n \in \mathbb{N}$  übergeben werden. Bei Übergabe gelte  $c_j = a_j$  für  $j = 0, \dots, m-1$  und  $c_j = b_{j-m}$  für  $j = m, \dots, m+n-1$ , d.h. bei Eingabe gilt  $c = (a, b)$ . Der Vektor  $c$  soll dann geeignet überschrieben werden. In der Funktion darf temporärer Speicher der Länge  $m+n$  angelegt werden. Schreiben Sie ein aufrufendes Hauptprogramm, in dem  $m, n \in \mathbb{N}$  sowie  $a \in \mathbb{R}^m$  und  $b \in \mathbb{R}^n$  eingelesen werden und  $c \in \mathbb{R}^{m+n}$  ausgegeben wird. Die Eingabe des Benutzers soll dabei auf Korrektheit überprüft und ggf. mit Fehlermeldung abgebrochen werden. Speichern Sie den Source-Code unter `merge` ins Verzeichnis `serie07`.

**Aufgabe 64\*.** Man schreibe eine rekursive Funktion `mergesort`, die ein Feld  $c \in \mathbb{R}^n$  aufsteigend sortiert und das sortierte Feld zurückgibt:

- Hat  $c$  Länge  $\leq 2$ , so wird das Feld  $c$  explizit sortiert.
- Hat  $c$  Länge  $> 2$ , halbiere man  $c$  in zwei Teilfelder  $a$  und  $b$ , rufe `mergesort` für  $a$  und  $b$  auf und vereinige die sortierten Teilfelder mittels `merge` aus Aufgabe 63.

Für die Implementierung beachten Sie, dass für  $k \in \mathbb{N}$  der Basepointer für den durch  $b_j := c_{j+k}$  definierten Teilvektor  $b \in \mathbb{R}^{n-k}$  durch  $c+k$  gegeben wird (sog. *Pointer-Arithmetik*). Schreiben Sie ein aufrufendes Hauptprogramm, in dem  $n$  und  $c \in \mathbb{R}^n$  eingelesen werden und der sortierte Vektor  $c$  ausgegeben wird. Speichern Sie den Source-Code unter `mergesort` ins Verzeichnis `serie07`. Was ist der Vorteil von *Mergesort* gegenüber *Bubblesort*?

**Aufgabe 65.** Man schreibe eine rekursive Funktion `binomial`, die den Binomialkoeffizienten  $\binom{n}{k}$  berechnet. Dazu wende man das Additionstheorem  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ . Ferner schreibe man eine Funktion `binomial2`, die den Binomialkoeffizienten mittels  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$  berechnet. Dazu ist eine rekursive Funktion `faktorielle` zu entwickeln, die zu gegebenem  $n \in \mathbb{N}$  die Faktorielle  $n!$  berechnet. Man schreibe ein aufrufendes Hauptprogramm, in dem  $k, n \in \mathbb{N}_0$  mit  $k \leq n$  eingelesen und  $\binom{n}{k}$ , berechnet auf beide Weisen, ausgegeben werden.

**Aufgabe 66.** Gegeben sei eine stetige Funktion  $f : [a, b] \rightarrow \mathbb{R}$  auf einem Intervall  $[a, b]$ . Es gelte

$$f(a) \cdot f(b) \leq 0.$$

Dann hat  $f$  eine Nullstelle  $x_0$ , die im Folgenden mittels Bisektion approximiert werden soll: Man definiert  $c := (a + b)/2$  als Intervallmittelpunkt. Aufgrund der Voraussetzung gilt

$$f(a) \cdot f(c) \leq 0 \quad \text{oder} \quad f(c) \cdot f(b) \leq 0.$$

Im Fall  $f(a) \cdot f(c) \leq 0$  ruft man den Bisektionsalgorithmus für das Intervall  $[a, c]$  auf, anderenfalls für das Intervall  $[c, b]$ . Als Abbruchbedingung verwende man  $|b - a| \leq \varepsilon$ . Wegen  $x_0 \in [a, b]$  ist dann beispielsweise  $c$  (oder auch  $a$  und  $b$ ) eine Approximation der Nullstelle bis auf einen Fehler  $\varepsilon$ . Der Funktion `bisektion` werden also die Parameter  $a, b \in \mathbb{R}$  und  $\varepsilon > 0$  übergeben. Im Fall  $f(a) \cdot f(b) > 0$  soll abgebrochen werden. Man realisiere die Übergabe von  $a$  und  $b$  mittels Call by Reference, sodass der Return-Value angibt, ob beim Bisektionsverfahren ein Fehler (Rückgabewert `-1`) aufgetreten ist oder nicht (Rückgabewert `0`). Bei jedem Funktionsaufruf sollen  $a, b, |b - a|$  und  $f(c)$  ausgegeben werden. Als Testfunktion verwende man  $f(x) = x^2 + \exp(x) - 2$  auf  $[0, \infty)$ , die man als eigene C-Funktion realisiere. Man schreibe ferner ein Hauptprogramm, das  $b, \varepsilon > 0$  einliest und, sofern kein Fehler aufgetreten ist, das Intervall  $[a, b]$  ausgibt, in dem eine Nullstelle liegt.

**Aufgabe 67.** Implementieren Sie den Quicksort-Algorithmus, um einen Vektor  $x \in \mathbb{R}^n$  zu sortieren: Quicksort wählt willkürlich ein Pivotelement aus der zu sortierenden Liste  $x$ , z.B.  $x_1$ . Dann zerlegt man die Liste in zwei Teillisten:  $x^{(<)}$  enthält alle Elemente  $< x_1$ ,  $x^{(\geq)}$  enthält alle Elemente  $\geq x_1$ . Diese Teillisten werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Es besteht (in der Reihenfolge) aus der unteren Liste  $x^{(<)}$ , dem Pivotelement und der oberen Liste  $x^{(\geq)}$ . — Eine direkte Implementation dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, geht man wie folgt vor: Beginnend mit  $j = 2$  sucht man ein Element  $x_j \geq x_1$ , d.h.  $x_j$  gehört zu  $x^{(\geq)}$ . Ferner sucht man beginnend bei  $k = n$  ein Element  $x_k < x_1$ , d.h.  $x_k$  gehört zu  $x^{(<)}$ . In diesem Fall vertauscht man  $x_j$  und  $x_k$ . Wenn sich die Zähler  $j$  und  $k$  treffen, liegt die Liste  $x$  bereits in der Form  $(x^{(<)}, x_1, x^{(\geq)})$  vor. Es müssen nun nur noch die Teillisten sortiert werden.

**Aufgabe 68.** Das Integral  $\int_a^b f dx$  einer stetigen Funktion  $f$  wird üblicherweise durch sogenannte Quadraturformeln berechnet. Die *Trapezregel*  $I_1(f, a, b)$  sowie die *Simpson-Regel*  $I_2(f, a, b)$  sind beispielsweise durch

$$I_1(f, a, b) := \frac{b-a}{2} (f(a) + f(b)) \quad \text{und} \quad I_2(f, a, b) := \frac{b-a}{6} \left( f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$$

gegeben. Schreiben Sie eine rekursive Funktion `adquad`, die das Integral  $\int_a^b f dx$  mittels Simpson-Regel  $I_2(f, a, b)$  und iterierter Intervallhalbierung

$$\int_a^b f dx = \int_a^{(a+b)/2} f dx + \int_{(a+b)/2}^b f dx \approx I_2(f, a, (a+b)/2) + I_2(f, (a+b)/2, b)$$

berechnet. Dabei soll der Integrationsbereich jeweils halbiert werden, falls

$$|I_1(f, a, b) - I_2(f, a, b)| > \varepsilon \max\{|I_1(f, a, b)|, |I_2(f, a, b)|\} \quad \text{und} \quad |b - a| > h_{\min}$$

ist. Der Funktion sollen neben  $a$  und  $b$  auch die Parameter  $\varepsilon$  und  $h_{\min}$  übergeben werden. Die Funktion  $f$  implementiere man als eigene C-Funktion. Zum Test verwende man das Integral  $\int_0^x \exp(t) dt = \exp(x) - 1$  bzw. das Integral  $\int_0^x \log(t) dt = x(\log(x) - 1)$ .

**Aufgabe 69.** Gegeben Sei ein sortierter Vektor  $x \in \mathbb{R}^n$ . Man schreibe eine Funktion `findbisection(x,y)`, die einen Index  $i$  zurückgibt, für den  $x_i = y$  gilt. Falls  $y$  nicht in  $x$  vorkommt, werde `0` zurückgegeben. Naive Realisierung führt auf Aufwand  $\mathcal{O}(n)$  im worst case, d.h. man durchsucht den Vektor von vorne nach hinten und das gesuchte  $y$  steht entweder an der letzten Stelle in  $x$  bzw. kommt überhaupt nicht in  $x$  vor. Wie kann man den Bisektionsalgorithmus aus Aufgabe 66 geeignet modifizieren, um einen Algorithmus mit worst case Aufwand  $\mathcal{O}(\log(n))$  zu erhalten?

**Aufgabe 70.** Man schreibe eine (rekursive) Funktion `papierschnitt`, die alle Möglichkeiten visualisiert, wie ein Papierbogen der ganzzahligen Länge `laenge` in Papierbahnen der Länge 1 und 2 geschnitten werden kann. — D.h. man stelle eine natürliche Zahl  $n = \text{laenge}$  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 `laenge=4` gibt es beispielsweise 5 Möglichkeiten:

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