Dirk Praetorius
Michele Ruggeri
Bernhard Stiftner

Sommersemester 2016
18.04.2016

**Übungen zur Vorlesung**
**Einführung in das Programmieren für TM**

**Serie 6**

**Aufgabe 6.1.** 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 Variablen `x` und `y` 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.c` in das Verzeichnis `serie06`.

**Aufgabe 6.2.** Die Funktion `squareVec` soll alle Einträge eines Vektors $x \in \mathbb{R}^n$ quadrieren, d.h. aus $(-1, 2, 0)$ soll $(1, 4, 0)$ werden. Der Vektor soll dabei als Pointer übergeben werden.

```
#include <stdio.h>

int squareVec(double vec, int n) {
  int j=0;
  for(j=1, j<dim; --j) {
    *vec[j] = &vec[j] * &vec[j];
  }
  return vec;
}

main() {
  double vec[3] = {-1.0,2.0,0.0};
  int j=0;

  squareVec(vec,3);
  for(j=0; j<3; ++j) {
    printf("vec[%d] = %f ",j,vec[j]);
  }
  printf("\n");
}
```

Ändern Sie nur die Funktion `squareVec`, so dass die `main`-Funktion das richtige Ergebnis ausgibt. Wie viele Fehler finden Sie? Welchen Aufwand hat Ihre korrigierte Funktion `squareVec`?

**Aufgabe 6.3.** Gegeben seien die Summen

$$a_N := \sum_{n=0}^{N} \frac{1}{(n+1)^2} \quad \text{und} \quad b_M := \sum_{m=0}^{M} \sum_{k=0}^{m} \frac{1}{(k+1)^2 (m-k+1)^2}.$$

Schreiben Sei zwei Funktionen, welche für gegebene $N$ bzw. $M \in \mathbb{N}$ die Zeit messen um $a_N$ bzw. $b_M$ zu berechnen. Stellen Sie mittels `assert` sicher, dass $M, N \geq 0$. Wie groß ist der Aufwand bei der Berechnung

von $a_N$ bzw. $b_M$? Z.B.: Falls die Funktion für $N = 10^3$ eine Laufzeit von 3 Sekunden hat, welche Laufzeit erwarten Sie aufgrund des Aufwands für $N = 10^4$? Schreiben Sie ferner ein Hauptprogramm, welches für verschiedene Werte von $N$ bzw. $M$ die Ergebnisse in Form einer Tabelle am Bildschirm ausgibt. Entsprechen die Resultate Ihren Erwartungen? Wie haben Sie Ihr Programm getestet? Speichern Sie den Source-Code unter `zeitmessung.c` in das Verzeichnis `serie06`.

**Aufgabe 6.4.** Um aus experimentellen Daten das asymptotische Verhalten zu bestimmen, kann man wie folgt vorgehen. Gegeben sei eine Funktion $f : \mathbb{N} \to \mathbb{R}$ und Datenpunkte $(N_i, f(N_i))$ für $i = 0, \dots M$. Angenommen $f = CN^\alpha$ mit einer Konstante $C > 0$ und $\alpha \in \mathbb{R}$. Aus den Datenpunkten lassen sich dann durch

$$\alpha_i = \frac{\log(f(N_i)) - \log(f(N_{i-1}))}{\log(N_i) - \log(N_{i-1})} \quad \text{für } i = 1, \dots, M$$

experimentelle Raten bestimmen. Überlegen Sie sich wie man auf obige Formel kommt. Schreiben Sie eine Funktion, welches aus experimentellen Daten die Raten bestimmt und ausgibt. Testen Sie ihre Funktion mit der Berechnung der beiden Summen der Aufgabe 6.3, d.h. $f(N)$ ist also die Zeit zur Berechnung der Summe $a_N$ bzw. $b_N$. Schreiben Sie ein Hauptprogramm, welches Ihre Ergebnisse in Form einer Tabelle ausgibt. Stimmen die Raten mit Ihren Erwartungen überein? Wie haben Sie Ihr Programm getestet? Speichern Sie den Source-Code unter `raten.c` in das Verzeichnis `serie06`.

**Aufgabe 6.5.** 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 $x^{(<)}$ und $x^{(\geq)}$ und das Pivotelement $x_1$: $x^{(<)}$ enthält dabei alle Elemente $< x_1$, $x^{(\geq)}$ enthält nur Elemente $\geq x_1$. $x^{(<)}$ und $x^{(\geq)}$ werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Eine direkte Implementierung dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, gehen Sie nun folgendermaßen 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 $x$ in der Form $(x_1, x^{(<)}, x^{(\geq)})$ vor. Mit einer weiteren Vertauschung erreicht man sofort die Gestalt $(x^{(<)}, x_1, x^{(\geq)})$. Es müssen nur noch $x^{(<)}$ und $x^{(\geq)}$ rekursiv sortiert werden. Schreiben darüberhinaus ein `main`-Programm, in dem Sie den Vektor $x$ und und die Länge $n$ einlesen und Quicksort aufrufen. Testen Sie Ihren Code entsprechend! Wie groß ist der Aufwand ihrer Funktion? Speichern Sie den Source-Code unter `quicksort.c` in das Verzeichnis `serie06`.

**Aufgabe 6.6.** Schreiben Sie 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. Dabei soll ausgenutzt werden, dass die Felder $a$ und $b$ bereits sortiert sind. Schreiben Sie die Funktion so, das 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. 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. Testen Sie ihr Programm mit entsprechenden Beispielen! Speichern Sie den Source-Code unter `merge.m` in das Verzeichnis `serie06`.

**Aufgabe 6.7.** Schreiben Sie eine rekursive Funktion `mergesort`, die ein Feld $a$ aufsteigend sortiert und das sortierte Feld zurückgibt. Gehen Sie dabei nach folgender Strategie vor:

- Hat $a$ Länge $\leq 2$, so wird das Feld $a$ explizit sortiert.
- Hat $a$ Länge $> 2$, halbiert man $a$ in zwei Teilfelder $b$ und $c$. Man ruft rekursiv `mergesort` für $b$ und $c$ auf und vereinigt die beiden sortierten Teilfelder wieder zu einem sortierten Feld.

Überlegen Sie sich, wie Sie die bereits sortierten Teilfelder $b$ und $c$ im zweiten Schritt vereinigen können! Machen Sie sich das gesamte Vorgehen anhand des Beispiels $a = (1, 3, 5, 2, 7, 1, 1, 3)$ klar. Schreiben Sie darüber hinaus ein Hauptprogramm, in dem Sie das Feld $a$ und die Länge einlesen, mit `mergesort` und sortiert ausgeben. Testen Sie das Programm entsprechend! Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie06`.

**Aufgabe 6.8.** Schreiben Sie eine Funktion `void unique(double * x, int * n)`, die einen Vektor $x \in \mathbb{R}^n$ aufsteigend sortiert, doppelte Einträge streicht und den Vektor in gekürzter Form zurückgibt. Die Funktion soll also beispielsweise den Vektor $x = (4, 3, 5, 1, 4, 3, 4) \in \mathbb{R}^7$ durch den Vektor $x = (1, 3, 4, 5) \in \mathbb{R}^4$ überschreiben. Die Länge $n$ der Vektoren ist dynamisch zu realisieren. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $n$ und $x \in \mathbb{R}^n$ eingelesen werden und das Ergebnis der Funktion `unique` ausgegeben wird. Testen Sie Ihre Implementierung mit passenden Beispielen! Speichern Sie den Source-Code unter `unique.c` in das Verzeichnis `serie06`.