Dirk Praetorius
Michele Ruggeri

Sommersemester 2015
21.04.2015

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

**Serie 5**

**Aufgabe 5.1.** *Bubble-Sort* ist ein ineffizienter, aber kurzer Sortier-Algorithmus: Man vergleicht aufsteigend jedes Element eines Arrays $x_j$ mit seinem Nachfolger $x_{j+1}$ und - falls notwendig - vertauscht die beiden. Nach dem ersten Durchlauf muß zumindest das letzte Element bereits am richtigen Platz sein. Der nächste Durchlauf muß also nur noch bis zur vorletzten Stelle gehen, usw. Wie viele geschachtelte Schleifen braucht dieses Vorgehen? Schreiben Sie eine Funktion `bubblesort`, die ein gegebenes Array $x \in \mathbb{R}^n$ mittels Bubble-Sort aufsteigend sortiert, d.h. $x_1 \leq x_2 \leq \cdots \leq x_n$, und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor $x$ einliest und in sortierter Reihenfolge ausgibt. Die Länge des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `bubblesort` ist für beliebige Länge $n$ zu programmieren. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie05`.

**Aufgabe 5.2.** 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 $j = 0, \ldots, m-1$ und $c_j = b_{j-m}$ für $j = m, \ldots, m+n-1$, 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. Speichern Sie den Source-Code unter `merge.m` in das Verzeichnis `serie05`.

**Aufgabe 5.3.** 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 vereinige die sortierten Teilfelder mittels `merge` aus Aufgabe 5.2.

Machen Sie sich das Vorgehen anhand des Beispiels $a = (1, 3, 5, 2, 7, 1, 1, 3)$ klar. Testen Sie das Programm entsprechend.
*Bemerkung:* Falls $a$ die Länge $2n+1$ für $n \geq 1$ hat, dann teilt man $a$ in die zwei Teilfelder $b$ und $c$, wobei $b$ die Länge $n + 1$ und $c$ die Länge $n$ hat. Für diese Aufgabe können Sie *Pointer-Arithmetik* verwenden, d.h. falls `a` ein Feld und `p` ein Pointer ist, welcher die Adresse von `a[k]` enthält, dann ist `p+n` die Adresse von `a[k+n]` (d.h. `*(p+n)` stimmt mit `a[k+n]` überein). Man beachte, dass `a` der Base-Pointer ist, der die Adresse von `a[0]` enthält.

**Aufgabe 5.4.** 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 ein Programm, welches für verschiedene Werte von $N$ bzw. $M$ die Zeit misst um $a_N$ bzw. $b_M$ zu berechnen. Geben Sie anschließend die Ergebnisse in Form einer Tabelle am Bildschirm aus. Entsprechen die Resultate Ihren Erwartungen? Speichern Sie den Source-Code unter `zeitmessung.c` in das Verzeichnis `serie05`. *Hinweis:* Überlegen Sie sich wie groß der Aufwand bei der Berechnung von $a_N$ bzw. $b_M$ ist.

**Aufgabe 5.5.** Die Sinus-Funktion hat die Reihendarstellung

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Wir betrachten die Partialsummen

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

Schreiben Sie eine Funktion `sin_`, die für gegebene $x \in \mathbb{R}$ und $\varepsilon > 0$ den Wert $S_n(x)$ zurückliefert, sobald

$$|S_n(x) - S_{n-1}(x)|/|S_n(x)| \leq \varepsilon \quad \text{oder} \quad |S_n(x)| \leq \varepsilon$$

gilt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem $x \in \mathbb{R}$ und $\varepsilon > 0$ eingelesen werden. Neben dem berechneten Wert $S_n(x)$ sollen auch der korrekte Wert $\sin(x)$ und der absolute Fehler $|S_n(x) - \sin(x)|$ ausgegeben werden sowie der relative Fehler $|S_n(x) - \sin(x)|/|\sin(x)|$ im Fall $\sin(x) \neq 0$. Speichern Sie den Source-Code unter `sin.c` in das Verzeichnis `serie05`.

**Aufgabe 5.6.** Eine Variante zur Berechnung einer Nullstelle einer Funktion $f : [a, b] \to \mathbb{R}$ ist das *Newton-Verfahren*. Ausgehend von einem Startwert $x_0$ definiert man induktiv eine Folge $(x_n)_{n\in\mathbb{N}}$ durch

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

Man realisiere das Newton-Verfahren in einer Funktion `newton`, wobei die Iteration abgebrochen wird, falls entweder

$$|f'(x_n)| \leq \tau$$

oder

$$|f(x_n)| \leq \tau \quad \text{und} \quad |x_n - x_{n-1}| \leq \begin{cases} \tau & \text{für } |x_n| \leq \tau, \\ \tau|x_n| & \text{sonst} \end{cases}$$

gilt. Im ersten Fall gebe man zusätzlich eine Warnung aus, dass das numerische Ergebnis vermutlich falsch ist. Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` und Ableitung `double fstrich(double x)` arbeiten. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $x_0$ eingelesen und $x_n$ ausgegeben wird.Speichern Sie den Source-Code unter `newton.c` in das Verzeichnis `serie05`.

**Aufgabe 5.7.** Genauso wie der Inhalt von Variablen elementaren Datentyps kann auch der Inhalt eines Pointers mittels `printf` ausgegeben werden. Man verwendet hier `%p` als Platzhalter für Adressen. Die Ausgabe dafür erfolgt systemabhängig meist in Hexadezimaldarstellung. Schreiben Sie eine Funktion `void charPointerAbstand(char* anfangsadresse, char* endadresse)`, welche folgende drei Werte tabelliert:

- Anfangsadresse
- Endadresse
- Abstand (Differenz) der beiden Adressen (Platzhalter im `printf` beachten!)

Da Arrays zusammenhängend im Speicher liegen, entspricht der Abstand zweier aufeinanderfolgender Elemente genau dem Speicherverbrauch des entsprechenden Datentyps. Testen Sie Ihre Funktion für einen `char`-Array `c[2]` mit den beiden Aufrufen:

```
charPointerAbstand(&c[0],&c[1]);
charPointerAbstand(c,c+1);
```

Schreiben Sie nun nach obiger Manier eine Funktion `void doublePointerAbstand(double* anfangsadresse, double* endadresse)`, testen diese mit einem `double`-Array und vergleichen die unterschiedlichen Ergebnisse.
*Optional:* Finden Sie heraus, wieviel Speicher die Typen `short`, `int` und `long` auf dem Übungsserver verbrauchen.

**Aufgabe 5.8.** Die Funktion `squareVector` 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`?