Dirk Praetorius,
Michele Ruggeri

Wintersemester 2015/16
25.-27.11.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. 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;
}
```

Aufgabe 7.2. 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 7.3. 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 7.4. Für eine differenzierbare Funktion $f : [a, b] \to \mathbb{R}$ kann man die Ableitung $f'(x)$ in einem festen Punkt $x \in \mathbb{R}$ durch den einseitigen Differenzenquotienten

$$\Phi(h) := \frac{f(x+h) - f(x)}{h} \quad \text{für } h > 0$$

approximieren. Schreiben Sie eine Funktion `double* diff(double x, double h0, double tau, int* n)`, die für $h_n := 2^{-n} h_0$ die Folge der $\Phi(h_n)$ berechnet, bis gilt

$$|\Phi(h_n) - \Phi(h_{n+1})| \leq \begin{cases} \tau & \text{falls } |\Phi(h_n)| \leq \tau, \textit{oder} \\ \tau\,|\Phi(h_n)| & \text{anderenfalls.} \end{cases}$$

Die Funktion soll mit einer beliebigen reellwertigen Funktion `double f(double x)` arbeiten. Die Funktion liefere in diesem Fall die vollständige Folge $(\Phi(h_0), \ldots, \Phi(h_n))$ der Iterierten zurück. Beachten Sie, dass auch die Länge des Vektors „zurückgegeben“ werden muss. Speichern Sie den Source-Code unter `diff.c` in das Verzeichnis `serie07`.

Aufgabe 7.5. 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 `serie07`.

Aufgabe 7.6. Das Newton-Verfahren aus Aufgabe 7.5 benötigt neben der Funktion `f` auch eine Funktion `fstrich`, die die Ableitung f' der Funktion f auswertet. Alternativ kann man $f'(x_k)$ durch den Differenzenquotienten $\Phi_h(x_k)$ aus Aufgabe 7.4 ersetzen. Realisieren Sie dieses Vorgehen indem Sie eine Funktion `newton2(x0,h0,tau)` schreiben, die zur Approximation der Abeitung $f'(x_k)$ das Ergebnis von `diff(xk,h0,tau,n)` verwendet. Speichern Sie den Source-Code unter `newton2.c` in das Verzeichnis `serie07`.

Aufgabe 7.7. 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 `serie07`.

Aufgabe 7.8. 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 ≤ 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. Testen Sie das Programm entsprechend! Schreiben Sie darüber hinaus ein Hauptprogramm, in dem Sie das Feld a einlesen, mit `mergesort` und sortiert ausgeben. Die Länge von a soll dabei ein Konstante im Hauptprogramm sein, die Funktion `mergesort` soll aber für beliebige Längen funktioneren. Speichern Sie den Source-Code unter `mergesort.c` in das Verzeichnis `serie07`.