Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
9./10.11.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

Aufgabe 3.1. Schreiben Sie eine `void`-Funktion `teiler`, die für eine gegebene Zahl $x \in \mathbb{N} := \{1, 2, 3, \dots\}$ ausgibt, ob diese durch 2, durch 3 oder durch 6 teilbar ist. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Integer x einliest und `teiler` aufruft. Speichern Sie den Source-Code unter `teiler.c` in das Verzeichnis `serie03`.

Aufgabe 3.2. Schreiben Sie eine `void`-Funktion `kurvendiskussion`, die für eine quadratische Funktion $p(x) = a + bx + cx^2$ mit Koeffizienten $a, b, c \in \mathbb{R}$ eine Kurvendiskussion durchführt. Wenn vorhanden, berechne man das Extremum (und Art) und die Nullstellen. Anderenfalls gebe man aus, dass die Funktion kein Extremum bzw. keine Nullstelle besitzt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Parameter a, b, c einliest und die Funktion aufruft. Speichern Sie den Source-Code unter `kurvendiskussion.c` in das Verzeichnis `serie03`.

Aufgabe 3.3. Schreiben Sie eine `void`-Funktion `geld`, die für einen übergebenen Geldbetrag $n \in \mathbb{N}$ die minimale Anzahl an Scheinen (500 €, 100 €, 50 €, 20 €, 10 €, 5 €) bzw. Münzen (2 €, 1 €) berechnet, die zusammen genau den Wert n haben, und am Bildschirm ausgibt, wie viele jeweils notwendig sind. Für $n = 351$ soll man beispielsweise folgenden Output erhalten

```
3 x 100 EUR
1 x 50 EUR
1 x 1 EUR
```

Schreiben Sie weiters ein Hauptprogramm, in dem der Wert $n \in \mathbb{N}$ eingelesen und die Funktion aufgerufen wird. Speichern Sie den Source-Code unter `geld.c` in das Verzeichnis `serie03`.

Aufgabe 3.4. Schreiben Sie ein Programm, das einen statischen Vektor x der Länge 1000 anlegt. Für die Koeffizienten $x[i]$ soll gelten, dass $x[i] = i$ für alle $i \in \{0, 1, \dots, 999\}$. Anschließend soll der Vektor am Bildschirm ausgegeben werden. Sie dürfen keine `for`-Schleifen verwenden. Speichern Sie den Source-Code unter `array.c` in das Verzeichnis `serie03`.
Hinweis: Schreiben Sie Funktionen `createVector` und `printVector`, die im Hauptprogramm aufgerufen werden.

Aufgabe 3.5. Schreiben Sie eine rekursive Funktion `double powN(double x, int n)`, welche x^n für einen ganzzahligen Exponenten $n \in \mathbb{Z}$ berechnet. Es gilt $x^0 = 1$ für alle $x \in \mathbb{R} \setminus \{0\}$ und für $n < 0$ gilt $x^n = (1/x)^{-n}$. Weiters gilt $0^n = 0$ für $n > 0$. Die Potenz 0^n ist für $n \leq 0$ nicht definiert. Die Funktion soll in diesem Fall den Wert `0.0/0.0` zurückgeben. Für diese Aufgabe dürfen Sie die Funktion `pow` aus der Mathematikbibliothek nicht verwenden. Speichern Sie den Source-Code unter `powN.c` in das Verzeichnis `serie03`.

Aufgabe 3.6. Eine (möglicherweise nicht die beste) Art die Zahl π anzunähern liefert die als *Leibniz-Reihe* bekannte Formel

$$\pi = \sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1}.$$

Die n-te Partialsumme

$$P(n) = \frac{4(-1)^n}{2n+1} + P(n-1)$$

können wir also als rekursive Funktion auffassen, für die $\lim_{n\to\infty} P(n) = \pi$ gilt. Implementieren Sie eine Funktion `double P(int n)` die obige Funktionalität realisiert. Schreiben Sie auch ein Hauptprogramm, das n über die Tastatur einliest und das obige Partialsumme berechnet und ausgibt. Speichern Sie den Source-Code unter `piRekursiv.c` in das Verzeichnis `serie03`.

Aufgabe 3.7. Nach einer alten Legende gibt es in Hanoi einen Tempel, in dem sich folgende rituelle Anordnung befindet: Es handelt sich um 3 Pfosten und um 64 goldene Scheiben, die in der Mitte ein Loch haben, so dass sie auf die Pfosten gestapelt werden können. Die 64 Scheiben haben paarweise verschiedenen Durchmesser. Als der Tempel erbaut wurde, wurden diese Scheiben der Größe nach aufsteigend auf den ersten Pfosten gestapelt. Die Aufgabe der Priester in diesem Tempel besteht darin, diese Scheiben systematisch unter Zuhilfenahme des zweiten Pfostens so umzustapeln, dass sich diese am Schluss in derselben Anordnung wie zu Beginn auf dem dritten Pfosten befinden. Dabei sind folgende Regeln zu beachten:

- Die Scheiben dürfen nur einzeln verschoben werden.
- In jedem Schritt wird die oberste Scheibe eines Stapels auf einen der anderen Stapel gesetzt. D.h. es kann jeweils nur die erste Scheibe bewegt werden.
- Eine Scheibe darf nie auf einer anderen Scheibe kleineren Durchmessers zu liegen kommen.

Das Ende der Welt, so sagt die Legende, ist durch denjenigen Zeitpunkt vorherbestimmt, an dem die Priester diese Aufgabe erfüllt haben werden.
Sei n die Anzahl der Scheiben ($n = 64$ in der Legende). Ein rekursiver Algorithmus, der dieses Problem löst, lässt sich wie folgt formulieren: Um die obersten $m \leq n$ auf Pfosten i befindlichen Scheiben auf Pfosten j zu verschieben, verschiebt man

1. die obersten $m-1$ Scheiben von Pfosten i auf Pfosten $k \notin \{i, j\}$,
2. die grösste der besagten m Scheiben von Pfosten i auf Pfosten j,
3. und schliesslich die $m-1$ in Schritt 1 auf Pfosten k verschobenen Scheiben auf Pfosten j.

Die Wahl $m = n$, $i = 1$ und $j = 3$ löst das Problem. Schreiben Sie eine rekursive Funktion `void hanoi(int m, int i, int j)` die diesen Algorithmus implementiert. Jeder einzelne Schritt soll dabei am Display protokolliert werden. Zum Beispiel:

```
Eine Scheibe wandert vom 2. zum 3. Pfosten.
```

Schreiben Sie auch ein Hauptprogramm, das n über die Tastatur einliest und die Liste der Schritte ausgibt. Zum Testen verwende man $n \ll 64$, z.B. $n = 3, 4, 5$. Speichern Sie den Source-Code unter `hanoi.c` in das Verzeichnis `serie03`.

Aufgabe 3.8. Wiederholen Sie die Begriffe *Lifetime & Scope*. Was gibt folgendes Programm aus und erklären Sie warum?

```
1  #include <stdio.h>
2
3  int max(int,int);
4
5  main() {
6    int x = 1;
7    int y = 2;
8    int z = 3;
9
10   printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
11
12   {
13     x = 100;
14     y = 4;
15     int z = max(x,y);
16     printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
17
18     {
19       x = y;
20       int y = 200;
21
22       printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
23     }
24     printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
25   }
26   printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
27 }
28
29 int max(int x, int y) {
30   if(x>=y) {
31     return x;
32   }
33   else {
34     return y;
35   }
36 }
```

Zeichnen Sie einen Zeitstrahl, wo sie die Lifetime und den Scope der Variablen `x,y,z` auftragen. Kennzeichnen Sie die einzelnen Blöcke bzw. Funktionen.