Dirk Praetorius,
Michele Ruggeri

Wintersemester 2016/17
20.10.2016

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

Aufgabe 3.1. Schreiben Sie eine Funktion `vektorprodukt`, die zu gegebenen Vektoren $\mathbf{u} = (a, b, c)^T$ und $\mathbf{v} = (x, y, z)^T$ das Vektorprodukt $\mathbf{w} = \mathbf{u} \times \mathbf{v}$ mit

$$w_1 = bz - cy$$
$$w_2 = cx - az$$
$$w_3 = ay - bx$$

berechnet und ausgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Vektoren $\mathbf{u}, \mathbf{v}$ eingelesen und die Funktion aufgerufen werden. Speichern Sie den Source-Code unter `vektorprodukt.c` in das Verzeichnis `serie03`.

Aufgabe 3.2. Schreiben Sie eine `void`-Funktion `dreieck`, die für gegebene Seitenlängen $a, b, c \in \mathbb{R}$ mit $a, b, c \geq 0$ feststellt, ob es sich bei dem zugehörigen Dreieck um ein allgemeines, gleichschenkeliges, gleichseitiges, rechtwinkeliges, eindimensional „entartetes" oder um ein „unmögliches" Dreieck handelt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem a, b und c eingelesen werden und die Funktion aufgerufen wird. Speichern Sie den Source-Code unter `dreieck.c` in das Verzeichnis `serie03`.

Aufgabe 3.3. Schreiben Sie eine rekursive Funktion `binomialRek`, die den Binomialkoeffizienten $\binom{n}{k}$ berechnet. Verwenden Sie dazu das Additionstheorem

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad \text{für } 1 \leq k < n$$

mit $\binom{n}{0} = 1 = \binom{n}{n}$ für $n \in \mathbb{N}_0$. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $k, n \in \mathbb{N}_0$ mit $k \leq n$ eingelesen und $\binom{n}{k}$, berechnet und ausgeben werden. Speichern Sie den Source-Code unter `binomialRek.c` in das Verzeichnis `serie03`.

Aufgabe 3.4. Schreiben Sie die Funktion `binomialdirect` die den Binomialkoeffizienten $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ auf direkte Weise berechnet. Verwenden Sie zur Berechnung von $n!$ die Funktion `int = factorial(int n)` aus der Vorlesung. Vergleichen Sie für verschiedene (n, k) diese Implementierung mit der aus Aufgabe 3.3. Was fällt Ihnen dabei auf? Speichern Sie den Source-Code unter `binomialdirect.c` in das Verzeichnis `serie03`.

Aufgabe 3.5. Die Fibonacci-Folge ist definiert durch $x_0 := 0$, $x_1 := 1$ und $x_{n+1} = x_n + x_{n-1}$. Schreiben Sie eine rekursive Funktion `fibonacciRek`, die zu gegebenem Index n das Folgenglied x_n zurückgibt. Speichern Sie den Source-Code unter `fibonacciRek.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 aud 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`.
Fakultativ (1 bonus-Punkt): Es lässt sich zeigen, dass eine optimale Lösung des Problems $2^n - 1$ Schritte benötigt. Ist die obige Rekursion optimal?

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

```
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     int x = 100;
14     y = 2;
15     z = max(x,y);
16     printf("(x,y,z) = (%d,%d,%d)\n",x,y,z);
17
18     {
```

```
19        int z = y;
20        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.