Eindimensionale Felder[1]

Der Computer dient dazu große Datenmengen zu verarbeiten. Mit unseren bisher zur Verfügung stehenden Mitteln ist dies jedoch kaum möglich. Sollen z.B. 100 reelle Zahlenwerte eingegeben und verarbeitet werden, so müssten diese alle einzeln deklariert werden:

```
var z1, z2, ... , z100: real;
```

Dabei müssten anhand der Punkte natürlich alle fehlenden Variablen ausgeschrieben werden. Mit dem Datentyp *Feld* ist es nun möglich, große Datenmengen **gleichen Typs** zu verarbeiten. Ein Nachteil ist allerdings, dass die Anzahl der Elemente des Feldes schon bei der Deklaration angegeben werden muss. Das Feld ist damit ein sogenannter statischer Datentyp. Es ist nicht möglich, während des Programmablaufs die Anzahl der Feldelemente zu verändern. Dies ist erst bei dynamischen Datentypen wie z.B. *Listen* möglich. Es ist ebenfalls nicht möglich in einem Feld Elemente mit unterschiedlichen Datentypen zu speichern. Dazu muss man sogenannte *Records* verwenden.

Ein Beispiel

Der Umgang mit Feldern soll an einem Beispiel aus der Mathematik aufgezeigt werden.:
$P(p_1; p_2; p_3)$ ist die in der Mathematik übliche Darstellungsweise eines Punktes des dreidimensionalen Anschauungsraumes. Offensichtlich werden unter dem Namen P mehrere Zahlen des gleichen Typs zusammengefasst. Die Zahlen p_1 bis p_3 heißen Komponenten und sind jeweils mit einem Index versehen. Dies wollen wir nun auf die Informatik übertragen:
P bezeichnet man als **eindimensionales Feld** (engl. Array), dessen **Komponenten** vom Typ `real` sind und dessen **Indexbereich** die Menge {1; 2; 3} ist.
Die Deklaration sieht folgendermaßen aus:

```
type punkt = array[1..3] of real;
var  p: punkt;
```

Erläuterungen:

- `type` zeigt an, dass ein neuer Datentyp deklariert wurde. Die Deklaration eines Datentyps muss hinter der Konstantendeklaration und vor der Variablendeklaration stehen. Hinter `type` steht der Bezeichner. Anschließend steht ein Gleichheitszeichen.
- Hinter `array` steht in eckigen Klammern der Indexbereich. Er wird durch Anfangszahl und Endzahl des Bereichs und zwei dazwischenstehende Punkte gekennzeichnet.
- Hinter `of` steht der Datentyp der einzelnen Komponenten.
- Die einzelnen Komponenten der Variablen p vom Typ Punkt werden durch p[i] mit $1 \le i \le 3$ angesprochen. Beachten Sie bitte, dass jedes Element des Feldes durch Angabe seines Index direkt angesprochen werden kann. Man spricht hierbei von einem wahlfreien Zugriff (engl. random access) auf die gespeicherten Elemente (Vergleich: Musik-CD, DVD, Arbeitsspeicher [RAM] eines PC, Festplatte). Im Gegensatz dazu erfolgt z.B. in Listen der Zugriff auf die gespeicherten Elemente seriell, d.h. die Liste muss vom Kopf angefangen bis zum betreffenden Element durchwandert werden. Erst dann hat man Zugriff auf das gewünschte Element (Vergleich: Musikkassette, Videokassette, Streamer).
- Die Ein – und Ausgabe sowie die Berechnung erfolgt bei Feldern fast immer über Schleifen.

Aufgabe1:

Nach Eingabe der Punkte $P(p_1; p_2; p_3)$ und $Q(q_1; q_2; q_3)$ sollen die Differenzen $q_i - p_i$ für $1 \le i \le 3$ ausgegeben werden. Nutzen Sie dazu das schon vorgegebene Delphi – Projekt!

1 inhaltlich entnommen aus: Junger, u.a.: *Informatik*, Diesterweg und Sauerländer, 1988

```
procedure Tform1.Button1Click(Sender: Tobject);
type punkt = array[1..3] of double;
var
  d, p, q: punkt;
  i: integer;
begin
  // Eingabe
  p[1]:= StrToFloat(Edit1.Text);
  p[2]:= StrToFloat(Edit2.Text);
  p[3]:= StrToFloat(Edit3.Text);
  q[1]:= StrToFloat(Edit4.Text);
  q[2]:= StrToFloat(Edit5.Text);
  q[3]:= StrToFloat(Edit6.Text);
  // Berechnung der Differenz
  for i := 1 to 3 do d[i] := q[i]-p[i];
  // Ausgabe
  Edit7.Text := FloatToStrF(d[1],ffFixed,6,2);
  Edit8.Text := FloatToStrF(d[2],ffFixed,6,2);
  Edit9.Text := FloatToStrF(d[3],ffFixed,6,2);
end;
```

Aufgabe 2:

Erweitern Sie das Programm dahingehend, dass die errechneten Differenzen $q_i - p_i$ zur Berechnung des Abstandes der beiden Punkte P und Q benutzt werden!

Abstandsformel: $\mathrm{PQ} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2}$

Aufgabe 3:

Suchen Sie die Fehler in den folgenden Programmteilen! Es sind jeweils zwei Fehler!

1)
```
var       f: feld;
          i: integer;
type   feld: array[-2..4] of real;
```

2)
```
type   name: array[1..10] of real;
var    nam = name;
```

3)
```
const    n = 1; m = 12;
type  wort = array[m..n] of char;
       v, w: wort
```