

# Datentypen in Pascal

## 1. Einfache Datentypen

Variablen eines einfachen Typs können jeweils nur einen Wert annehmen. Turbo-Pascal stellt 6 einfache Datentypen zur Verfügung.

**real** : Kommazahlen  
**integer** : ganze Zahl zwischen -32768 und 32767  
**byte** : ganze Zahl zwischen 0 und 255  
**char** : einzelnes Zeichen  
**string** : eine Zeichenkette mit bis zu 255 Zeichen  
**boolean** : Wahrheitswerte true oder false

## 2. Benutzerdefinierte Datentypen

Durch eine TYPE-Vereinbarung kann der Benutzer seine eigenen Datentypen definieren. Man kann entweder

- vordefinierte Datentypen umbenennen
- alle erlaubten Werte eines Datentyps aufzählen

Beispiele:

- a) TYPE Ganzzahl = integer;  
VAR a,b,c : Ganzzahl;
- b) TYPE Tage = (Mo,Di,Mi,Do,Fr,Sa,So);  
VAR t : Tage;  
mögliche Anweisung:  
for t := Mo to Fr do ...

## 3. Felder (ARRAY)

Bei vielen Aufgaben in der Datenverarbeitung besteht die Notwendigkeit, eine größere Anzahl von Daten des gleichen Typs zu bearbeiten.

Beispiel 1: Der Umsatz, den eine Firma in den einzelnen Monaten eines Jahres gemacht hat, eingelesen und bearbeitet werden, z.B. prozentual oder als Balkendiagramm dargestellt werden.

```
TYPE Umsatztyp = ARRAY[1..12] of real;  
VAR Umsatz : umsatztyp;
```

Durch diese Vereinbarung wird ein ganzes Feld mit 12 Fächern definiert. In jedes Fach kann eine Kommazahl gespeichert werden.

Die einzelnen Elemente des Feldes UMSATZ sind UMSATZ[1], UMSATZ[2], UMSATZ[3] usw.

Um alle Elemente eines Feldes einzulesen oder auszugeben, bieten sich die Wiederholungen mit for ... to ... do ... an.

```

for i := 1 to 12 do
  begin write('  Wert = '); readln(UMSATZ[i]);
  end;

```

Die Variable i heißt auch Index und nimmt nacheinander die Werte von 1 bis 12 an.

Beispiel 2: Der Umgang mit Feldern soll an einem einfachen Beispiel demonstriert werden. Das Feld soll zunächst 15 ganze Zahlen aufnehmen können. Damit Schreibarbeit gespart wird, wird das Feld mit Zufallszahlen gefüllt, es soll auf dem Bildschirm ausgegeben werden, dann der Größe nach sortiert und zum Schluss auch noch auf der Festplatte als Datei gespeichert werden.

```

program Feld_demo;
uses crt,dos;

const max = 15; {Anzahl der Feldelemente}

type feldtyp = array[1..max] of integer;
var feld: feldtyp;
    datei : file of feldtyp;

```

Die Variable Feld wird als globale Variable vereinbart, alle Prozeduren benutzen diese Variable. Zunächst werden zufällige Zahlen in das Feld geschrieben.

```

procedure feldfuellen;
{füllt Feld mit Zufallszahlen}
var i : integer;
begin for i := 1 to max do
  feld[i] := random(100)+10;
end; {eingabe}

```

Die Prozeduren werden jeweils im Hauptprogramm aufgerufen. Zur Kontrolle werden alle Feldelemente auf dem Bildschirm ausgegeben.

```

procedure feldausgeben;
var i: integer;
begin clrscr;
  writeln('  So sieht das Feld jetzt aus:');
  for i := 1 to max do
    writeln(i:6,feld[i]:8);
  readln;
end;

```

Ein wichtiger Algorithmus ist das Sortieren des Feldes nach der Größe. Der einfachste Algorithmus ist das Sortieren durch wiederholtes Vertauschen benachbarter Elemente. Das größte Element wandert dabei wie Luftblasen in einer Flüssigkeit nach oben (bubble sort). Zunächst wird eine Prozedur zum Vertauschen zweier Zahlen benötigt. Die einfache Anweisungsfolge  $a := b$ ;  $b := a$ ; ist ungeeignet, weil danach in beiden Variablen der Wert von b steht. Es wird daher eine Hilfsvariable c benötigt.

```

procedure tausche(var a,b:integer);
var c: integer;
begin c := a; a := b; b := c;
end;

procedure feldsortieren;
var sortiert : boolean;
    i : integer;

begin writeln('    Das Feld wird jetzt sortiert.');
```

```

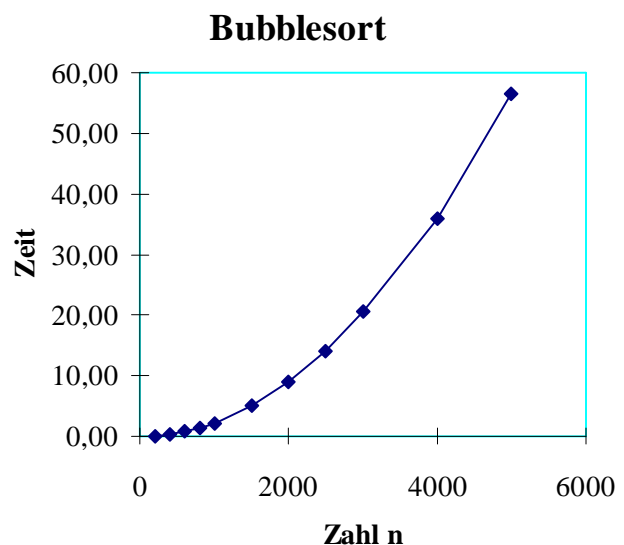
    repeat sortiert := true;
        for i := 1 to max-1 do
            if feld[i] > feld[i+1] then
                begin tausche(feld[i],feld[i+1]);
                    sortiert := false;
                end;
            until sortiert=true;
readln;
end;

```

Wichtig ist die bool'sche Variable **sortiert**. Man nimmt vor jedem Durchlauf durch das Feld an, dass es bereits sortiert ist. Sobald eine Vertauschung erfolgt, wird diese Variable auf false gesetzt, man weiß dann, dass das Feld noch nicht sortiert ist.

Dieses Sortierverfahren ist einfach zu verstehen, arbeitet aber bei großen Feldern immer langsamer. Die folgende Tabelle zeigt, wie lange das Sortieren bei verschiedenen großen Feldern dauert, gemessen auf einem 486er mit 33 Mhz.

Zahl n	Zeit in s
200,00	0,07
400,00	0,33
600,00	0,77
800,00	1,43
1000,00	2,20
1500,00	5,00
2000,00	8,96
2500,00	13,90
3000,00	20,50
4000,00	35,84
5000,00	56,60



Die Darstellung in einem Diagramm ergibt eine Parabel. Die Zeit zum Sortieren ist proportional zum Quadrat der Zahl der Feldelemente. Für sehr große Felder werden daher schnellere Sortierverfahren benötigt.

## Speichern und Laden des Feldes als Datei

In der Praxis sollen die mühevoll eingegebenen Werte eines Feldes nach Beendigung des Programms nicht verloren sein. Sie werden daher als Datei auf der Festplatte dauerhaft abgespeichert. Dazu wird eine Dateivariablen definiert, die Variablen vom Feldtyp aufnehmen kann.

```
var datei : file of feldtyp;
```

Für solche Dateien stehen einige Prozeduren in Pascal zur Verfügung.

- |                               |   |
|-------------------------------|---|
| 1. ASSIGN(Datei, 'Feld.dat'); | ordnet der Dateivariablen einen Dateinamen zu |
| 2. REWRITE(Datei);            | öffnet die Datei zum Schreiben                |
| 3. RESET(Datei);              | öffnet die Datei zum Lesen                    |
| 4. WRITE(Datei, feld);        | schreibt die Variable feld in die Datei       |
| 5. READ(Datei, feld);         | liest die Variable feld aus der Datei         |
| 6. CLOSE(Datei);              | schließt die Datei                            |

In unserem Beispiel wird jeweils das ganze Feld als eine einzige Variable in der Datei gespeichert.

```
procedure feldspeichern;  
begin writeln('    Das Feld wird jetzt gespeichert.');
```

```
    assign(datei, 'feld.dat');  
    rewrite(datei);  
    write(datei, feld);  
    close(datei);  
    writeln('    Feld erfolgreich abgespeichert!');
```

```
    readln;  
end; {speichern}
```

```
procedure feldladen;  
begin writeln('    Das Feld wird jetzt geladen');
```

```
    assign(datei, 'feld.dat');  
    reset(datei);  
    read(datei, feld);  
    close(datei);  
    writeln('    Tabelle erfolgreich geladen!');
```

```
    readln;  
end; {laden}
```

Sind in einem Feld viele Zahlen abgespeichert, so werden diese oft als Diagramm dargestellt. In der Praxis benutzt man heute meist ein Tabellenkalkulationsprogramm, aber auch mit Turbo-Pascal sind unsere 15 Zahlenwerte schnell als Balkendiagramm dargestellt. Um Schreibarbeit zu sparen, kopiert man sich die eigenen Grafikprozeduren aus einem früheren Programm.

```

procedure grafikein;
var k, treiber,
modus:integer;
begin clrscr;
    treiber:=detect;
initgraph(treiber,modus,'');
end;

```

```

procedure grafikaus;
begin
repeat until readkey <> '';
    closegraph;
end;

```

```

procedure schreibe(x,y:integer;t:string);
begin moveto(x,y);
    settextstyle(1,0,2);
    outtext(t);
end;

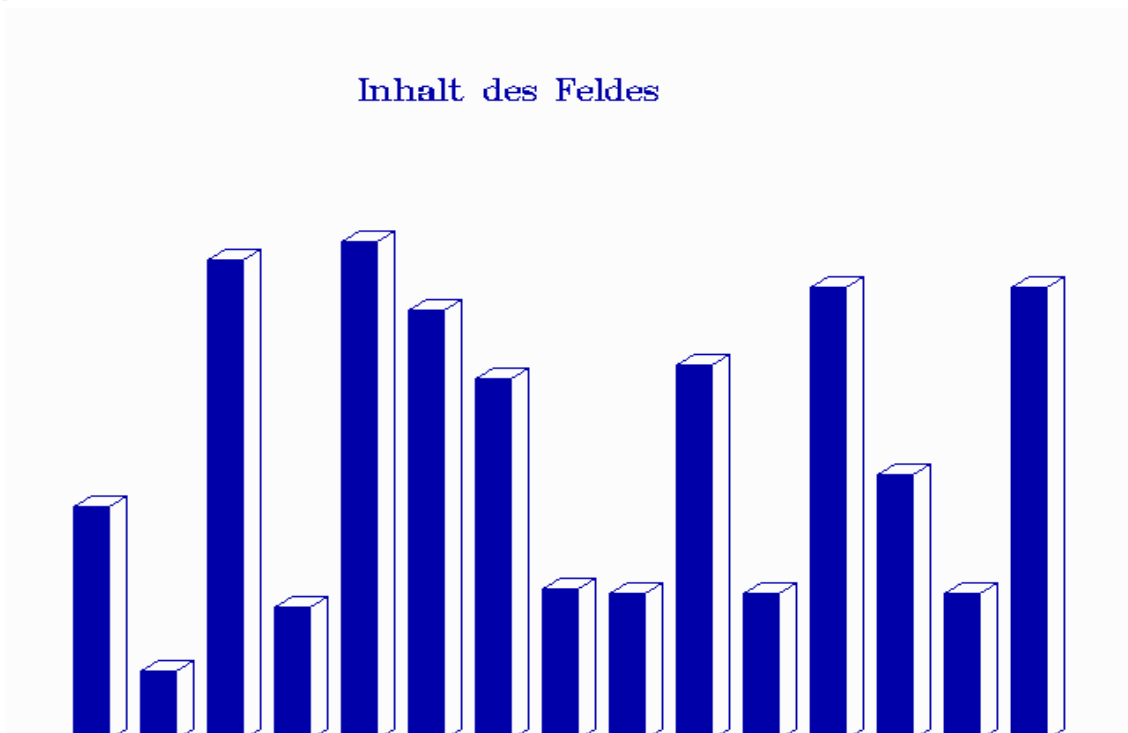
```

Die folgende kleine Prozedur erstellt dann aus den 15 Zahlen des Feldes ein Balkendiagramm.

```

procedure felddiagramm;
var x,y: integer;
begin grafikein;
    setbkcolor(white); setcolor(blue);
    setfillstyle(1,blue);
    schreibe(200,40,'Inhalt des Feldes');
    line(0,479,640,479);
    for x := 1 to 15 do
    begin y := 3*feld[x];
        bar(38*x, 480, 38*x+20,480-y);
        {bar3d(38*x,480,38*x+20,480-y,10,true);
        für räumliche Balken}
    end;
grafikaus;
end;

```



#### 4. Zeichenkette (STRING)

Zur Aufnahme von Zeichenketten, also Worten und Texten, stellt Pascal den vordefinierten Datentyp STRING zur Verfügung. Dieser Datentyp stellt einen strukturierten Datentyp dar, bei dem man auf jede Komponente einzeln zugreifen kann.

```
TYPE string = ARRAY[1..255] of char;
var wort : string;
```

##### Elementare Operationen mit einer Zeichenkette

- ganze Zeichenkette einlesen: `readln(wort);`
- den 5. Buchstaben schreiben: `write(wort[5]);`
- Zeichenketten addieren: `wort := wort + 'txt';`
- Länge der Zeichenkette `laenge := length(wort);`

Es gibt weitere Prozeduren zum Einfügen von Text in eine Zeichenkette (INSERT) oder zum Löschen von Teilen der Zeichenkette (DELETE).

Gelegentlich wird die Umwandlung einer Zahl in eine Zeichenkette benötigt, etwa um Zahlen in der hochauflösenden Grafik auszugeben: `STR(5600, zahlstring);`

Beispiel: Jeder kennt das bekannte Kinderlied „Drei Chinesen mit dem Kontrabass“, bei dem in den einzelnen Strophen alle Vokale durch einen einzigen ersetzt werden:

*Dree Chenesen met dem Kentrebess seßen eef der Streße  
end erzählten sech wes, kem dee Pelzee, fregt wes est denn des?  
dree Chenesen met dem Kentrebess.*

```
program string_demo;
uses crt;
var s1,s2,s3 : string;
    k, l : byte;
    vokal : char;
begin clrscr;
  writeln('  Lustiges Wortspiel mit den drei Chninesen');
  write('  Wie sollen alle Vokale ersetzt werden? ');
  readln(vokal);
  s1:='Drei Chinesen mit dem Kontrabass saßen auf der Straße';
  s2 := 'und erzählten sich was, kam die Polizei, fragt was ist
        denn das?';
  s3 := 'drei Chinesen mit dem Kontrabass.';
  for k := 1 to 255 do
  begin
    if s1[k] in ['e','i','o','a','u'] then s1[k] := vokal;
    if s2[k] in ['e','i','o','a','u'] then s2[k] := vokal;
    if s3[k] in ['e','i','o','a','u'] then s3[k] := vokal;
  end;
  writeln;
  writeln(s1);  writeln(s2);  writeln(s3);
  readln;
end.
```

## 5. Kryptologie - Verschlüsselung von Informationen

Die modernen Methoden der Informationsübermittlung haben das Problem der Datensicherheit entstehen lassen. Man denke nur an elektronische Post oder die Abwicklung von Bankgeschäften über das Internet. Daher werden vertrauliche Informationen verschlüsselt, das heißt so umgeformt, dass sie nur vom rechtmäßigen Empfänger wieder entschlüsselt werden können, für andere Personen aber unverständlich bleiben.

Die Geschichte der Kryptologie, d.h. der Lehre von Verschlüsselungsverfahren, reicht bis ins Altertum zurück. Im 2. Weltkrieg wurden die Funksprüche der deutschen Wehrmacht durch eine komplizierte Maschine, die ENIGMA genannt wurde, verschlüsselt. Es gelang aber einer Gruppe britischer Mathematiker, viele der verschlüsselten Nachrichten zu „knacken“.

Beispiel 1: Beim sog. „Cäsar-Verfahren“ wird jeder Buchstabe durch seinen s-ten Nachfolger im Alphabet ersetzt. Ist  $s = 2$ , so wird aus „LEBACH“ das Wort „NGDCEJ“. Leider sind Texte, die nach diesem Verfahren verschlüsselt wurden, sehr leicht von Unbefugten zu entschlüsseln. Man benutzt dazu eine Tabelle der relativen Häufigkeiten der einzelnen Buchstaben. In der deutschen Sprache sind folgende Buchstaben am häufigsten:

‘e’ : 13,4 % - ‘n’ : 8,0 % - ‘i’ : 6,2 % - ‘r’ : 5,8 %

Ist aufgrund der relativen Häufigkeit erst einmal das ‘e’ im verschlüsselten Text erkannt, so kennt man natürlich auch die Zahl  $s$  und das Verschlüsselungsverfahren ist bekannt. Das einfache Cäsar-Verfahren wird daher heute nicht mehr benutzt.

Beispiel 2: Blaise de Vigenère, ein französischer Kryptologe aus dem 16. Jahrhundert, erschwerte Cäsars Methode, indem jeder Buchstabe durch mehrere Geheimbuchstaben ersetzt wird. So kann die Zahl  $s$  etwa kontinuierlich von -5 bis 5 laufen.

Ist nur ein Satz, also eine Variable vom typ string zu verschlüsseln und zu entschlüsseln, so erledigen die beiden folgenden Prozeduren diese Aufgaben.

```
procedure verschluesseln(var wort : string);
begin for k := 1 to length(wort) do
  begin n := ord(wort[k]);
        n := n + ( k mod 10 - 5 );
        wort[k] := chr(n);
  end;
  writeln(' ',wort);
end;
```

```
procedure entschluesseln(var wort:string);
begin for k := 1 to length(wort) do
  begin n := ord(wort[k]);
        n := n - ( k mod 10 - 5 );
        wort[k] := chr(n);
  end;
  writeln(' ',wort);
end;
```

Leider ist auch dieses Verfahren nicht sicher genug, so dass heute wesentlich bessere Methoden zur Verschlüsselung benötigt werden.

## 6. Verbundtyp (record)

In einem Feld kann man zwar viele Daten abspeichern, die aber alle vom gleichen Typ sein müssen. Oft gehören aber gewisse Daten zusammen, die von höchst unterschiedlichem Typ sind.

Beispiel 1: Ein Firma will zu jedem Kunden folgende Informationen abspeichern:

- Kundennummer
- Vorname
- Name
- bisheriger Umsatz
- Stammkunde (ja / nein)

Diese verschiedenen Informationen lassen sich in einer Variablen vom Typ **record** (Verbundtyp) zusammenfassen.

```
type kundentyp = record
    kundennummer : integer;
    vorname : string[20];
    name : string[20];
    umsatz : real;
    stammkunde : boolean;
end;
var kunde : kundentyp;
    kundenfeld : array[1..80] of kundentyp;
```

Jede Komponente des Feldes kundenfeld ist wiederum eine Verbundvariable und wird als Datensatz bezeichnet.

Der Zugriff auf die einzelnen Komponenten kann auf zwei Arten erfolgen.

### 1. direkt mit Separationspunkt

```
readln(kunde.vorname);
writeln(kunde.umsatz);
```

### 2. mit der with-Anweisung

```
with kunde do begin
    writeln(vorname);
    readln(umsatz);
end;
```

Übung: Entwirf einen Datentyp und eine Variable für folgende Informationen

- Kontonummer
- Kontoinhaber
- Kontostand
- Kreditlimit

## Beispiel 2: Eine Datei zur Verwaltung von CD's

Ein menügesteuertes Programm soll Informationen zu mehreren CD's verwalten, abspeichern und sortieren. Außerdem sollen sich CD's über Suchbegriffe finden lassen. Wir vereinbaren folgende Datentypen:

```
program cdverwaltung;
uses crt;

type cdtyp = record
    Interpret: string[20];
    Titel: string[40];
    Musikrichtung: string[20];
    Spieldauer: integer;
    jahr: integer;
end;

var cd : array[1..100] of cdtyp;
    wahl : byte;
    index : integer;
    datei : file of cdtyp;
```

Das **Hauptprogramm** besteht lediglich aus einem Auswahlmenü, die einzelnen Aufgaben werden in Prozeduren verlagert. So hat man schnell ein lauffähiges Programm, das nach und nach erweitert werden kann.

```
begin repeat
    textbackground(blue); textcolor(yellow);
    clrscr;
    writeln('    Menüprogramm zum Verwalten von CDs');
    writeln('    =====');
    writeln('        Version 1.01 - April 1998');
    writeln;
    writeln('    1 : Eine neue CD eingeben');
    writeln('    2 : Liste aller CDs');
    writeln('    3 : CD-Liste speichern');
    writeln('    4 : CD-Liste laden');
    writeln('    5 : Sortieren nach Interpret');
    writeln('    6 : CD suchen');
    writeln('    0 : E N D E');
    writeln;
    write('    Bitte wählen! ');
    readln(wahl);
    if wahl = 1 then eingabe;
    if wahl = 2 then ausgabe;
    if wahl = 3 then speichern;
    if wahl = 4 then laden;
    if wahl = 5 then sort;
    if wahl = 6 then suchen;
    until wahl = 0;
end.
```

## Dateneingabe und Datenausgabe

Die Ausgabe eines einzelnen Datensatzes kann man in eine Prozedur verlagern. Dabei wird überprüft, ob der Datensatz nicht leer ist (`if cd[index].seitenzahl > 0`).

```
procedure eingabe;
begin clrscr;
  write('  Indexnummer der neuen CD (1 - 100): ');
  readln(index);
  with cd[index] do
  begin write('  Interpret : '); readln(interpret);
        write('  Titel      : '); readln(titel);
        write('  Richtung  : '); readln(musikrichtung);
        write('  Spieldauer: '); readln(spieldauer);
        write('  Jahr   : '); readln(jahr);
  end;
end;
```

Die Ausgabe mehrerer Bücher erfolgt in einer repeat-Wiederholung. Die Endbedingung `until readkey = #27` bedeutet, dass die Ausgabe der Bücherdatei durch die ESC-Taste abgebrochen werden kann. Die Prozedur `schreibe` gibt nur einen Datensatz aus.

```
procedure schreibe(index:integer);
begin with cd[index] do
  if cd[index].spieldauer > 0 then
  begin
    writeln(index);
    writeln('  Interpret : ',interpret);
    writeln('  Titel      : ',Titel);
    writeln('  Richtung  : ',musikrichtung);
    writeln(spieldauer:6,' Minuten');
    writeln(jahr:6,' erschienen');
    writeln;
  end;
end;
```

```
procedure ausgabe;
begin clrscr;
  writeln('  Ausgabe der CD-Liste - Abbruch mit ESC');
  writeln;
  index := 1;
  repeat with cd[index] do
  if spieldauer > 0 then
    schreibe(index);
    index := index + 1;
  until (readkey = #27) or (index>100)
or(cd[index].spieldauer=0);
  writeln('  Fertig! Weiter mit RETURN!');
  readln;
end;
```

## Speichern und Laden der Datensätze

Die mühevoll eingegebenen Informationen sollen natürlich auf der Festplatte oder einer Diskette abgespeichert und von dort geladen werden.

```
procedure speichern;
begin clrscr;
  writeln('    CD-Datei wird jetzt gespeichert');
  delay(1000);
  assign(datei,'a:cd.dat');
  rewrite(datei);
  for index := 1 to 100 do
    write(datei,cd[index]);
  close(datei);
end;

procedure laden;
begin clrscr;
  writeln('    CD-Datei wird jetzt geladen');
  delay(1000);
  assign(datei,'a:cd.dat');
  reset(datei);
  for index := 1 to 100 do
    read(datei,cd[index]);
  close(datei);
end;
```

## Sortieren nach Interpreten

Die einzelnen Datensätze können nach jeder Komponente sortiert werden. Oft werden CD'S nach Interpret oder Komponist sortiert.

Da zunächst viele Datensätze leer sind, würden diese leeren Datensätze nach vorne sortiert werden, was wenig sinnvoll ist. Daher wird zunächst die Anzahl der vorhandenen Datensätze ermittelt.

```
function cdzahl:integer;
var k : integer;
begin k := 0;
  repeat k := k+1
  until cd[k].spieldauer < 1;
  cdzahl := k;
end;
```

Dann werden nur die ersten cdzahl Datensätze in den Sortiervorgang eingebunden. Das Sortieren erfolgt wieder nach dem einfachen Verfahren des „bubble sort“.

```
procedure sort;
var sortiert: boolean;
  hilf : cdtyp;
  k : integer;
begin repeat
  sortiert := true;
  for k := 1 to cdzahl do
    if cd[k].interpret > cd[k+1].interpret then
      begin sortiert := false;
        hilf := cd[k];
        cd[k] := cd[k+1];
        cd[k+1] := hilf;
      end;
  until sortiert;
end;
```

```

        cd[k+1] := hilf;
    end; {then}
until sortiert;
clrscr;
writeln('    Das Feld wurde nach Interpreten sortiert. ');
delay(1000);
end; {sort}

```

## Suchen von CD's

Besonders bei sehr vielen Datensätzen wünscht man sich, eine bestimmte CD gezielt über ein Stichwort zu finden. Da man oft nur Bruchstücke des Titels weiß, soll auch ein Teil des Titels oder des Interpreten zum Finden genügen.

So soll beispielsweise die CD „Dire straits: on every street“ mit den Stichworten „straits“ oder „street“ gefunden werden!

Es gibt in Turbo-Pascal eine leistungsstarke Funktion, die ein Stichwort in einem Suchtext sucht und die Position dieses Stichwortes als Funktionswert liefert. Die Vereinbarung dieser Funktion lautet:

```
function pos(stichwort:string; suchtext:string):integer;
```

Ist das Stichwort nicht enthalten, so ist der Funktionswert 0. Genau diese Eigenschaft der Funktion pos kann man daher zur Suche ausnutzen.

```

procedure suchen;
var suchtext: string;
begin clrscr;
    writeln('    Suchen nach Interpret oder Titel');
    writeln;
    write('    Bitte Suchbegriff eingeben: ');
    readln(suchtext);
    for index := 1 to cdzahl do
        if (pos(suchtext,cd[index].interpret) > 0) or
            (pos(suchtext,cd[index].titel) > 0) then
            begin schreibe(index);
                readln;
            end;
    writeln('    Fertig!');
    readln;
end;

```

Dieses Programm ist ein typisches Beispiel für ein sog. Dateiprogramm. In solchen Programmen wird nicht gerechnet, sondern es werden nur sehr viele Informationen hin und her bewegt oder getauscht und es wird nach Informationen gesucht. Das obige Programmbeispiel ließe sich leicht abändern, um beispielsweise Bücher, Weinflaschen oder Kunden zu verwalten.

Heute werden solche Aufgaben meistens von Datenbankprogrammen (DBASE, ACCESS, mit Einschränkungen auch EXCEL) erledigt. Diese Programme lassen sich sehr flexibel an die jeweilige Anwendung anpassen.

Auch ein Tabellenkalkulationsprogramm kann zur Datenverwaltung eingesetzt werden. Eine Zeile entspricht einem Datensatz, jede Spalte entspricht einer Komponente.