DELPHI 05

Sortier- und Suchalgorithmen

Das Speicherarray - Eine Spielwiese für Algorithmen

Als Informatiklehrer, der schon jahrelang dieses Fach an einer AHS-Oberstufe unterrichtet, bin ich der Meinung, dass das Erlernen des Programmierens als Werkzeug zur Problemlösung wichtig und unverzichtbar ist. Dabei darf der Unterricht nicht stehenbleiben bei der mehr oder weniger geschickten Verwendung von visuellen Dialogobjekten, sondern soll an einfachen Beispielen algorithmisches Denken vermitteln. Von zentraler Bedeutung sind dabei Sortier- und Suchverfahren. Nach meinen Erfahrungen stellen indizierte Speicherarrays eine sehr geeignete Datenstruktur zur systematischen Entwicklung solcher Algorithmen dar. Ein Speicherarray ist nichts anderes als ein Kasten mit Schubladen, welche fortlaufend mit eindeutigen Nummern versehen sind. Dadurch wird ein gezielter Zugriff auf die Daten in den Schubladen möglich. Alle Schubladen enthalten Daten vom gleichen Datentyp.

Als Programmiersprache habe ich mich vor Jahren für PASCAL im Betriebssystem MSDOS und in der heutigen Zeit für die Nachfolgersprache DELPHI im Betriebssystem WINDOWS entschieden. Hauptgrund dafür ist die gut strukturierte und daher leicht erfassbare Syntax der Sprache. Sie eignet sich vorzüglich sowohl für Anfänger als auch für professionelle Programmierer. Im vorliegenden Artikel werden einfache Sortier- und Suchverfahren besprochen, die sich alle auf Datenbestände im Hauptspeicher beziehen. Als Daten werden ganze Zufallszahlen verwendet, welche in einem Array abgespeichert sind. Beispielhaft sollen folgende elf Algorithmen besprochen werden.

[5.01] Zufallszahlen ohne Wiederholung
[5.02] Zufallszahlen mit Wiederholung
[5.03] Sortieren durch direktes Austauschen
[5.04] Sortieren durch direktes Einfügen
[5.05] Sequentielles Suchen
[5.06] Binäres Suchen
[5.07] Elemente einfügen
[5.08] Elemente löschen
[5.09] Sortieren mittels Quick-Sort
[5.10] Geprüfte Eingabe von Zahlen
[5.11] Formatierte Ausgabe von Zahlen

Neben den numerischen Datentypen Integer und Real ist die Zeichenkette (String) ein weiterer sehr wichtiger Datentyp. Diese besteht aus aufeinander folgenden Bytes, welche als ANSI-Zeichen (Char bzw. Character) interpretiert werden und auf die über einen Index zugegriffen werden kann. Im Grund ist also eine Zeichenkette nichts anderes als ein Speicherarray, in dessen Schubladen Zeichen stehen.

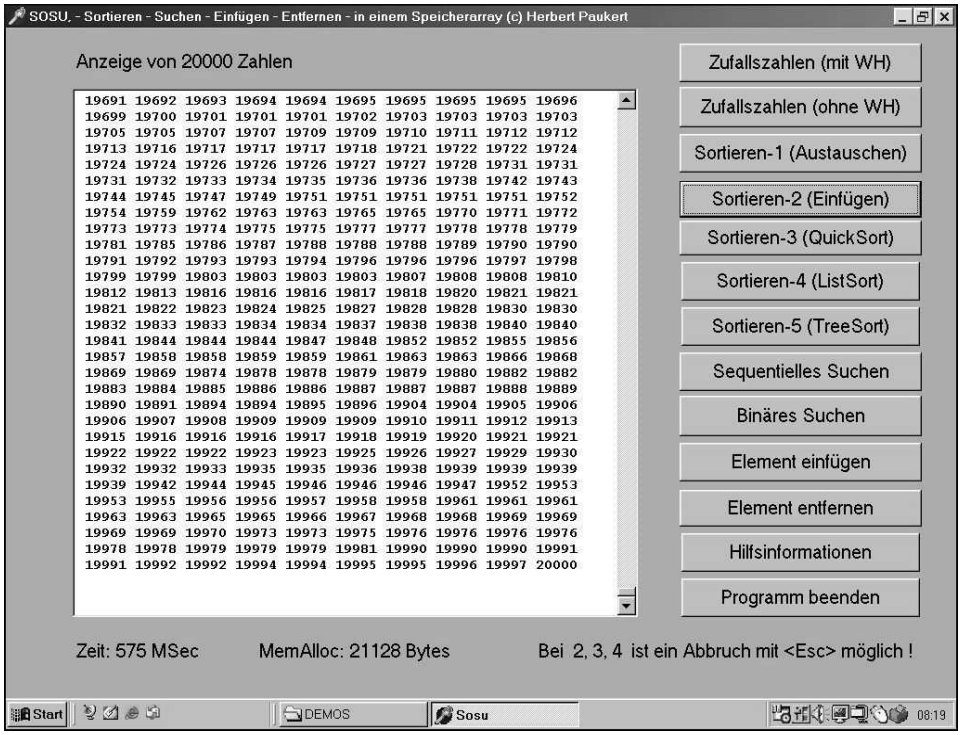
Solche Zeichenketten sind die zentralen Bestandteile jeder Textverarbeitung. Als Behälterobjekte dienen so genannte Memo-Felder oder Richedit-Komponenten, mit deren Hilfe die einzelnen Textzeilen visuell dargestellt werden. Jede Textzeile ist ein String. Der gesamte Text kann seinerseits ebenfalls als ein Speicherarray aufgefasst werden, in dessen Schubladen die einzelnen Textzeilen stehen und auf die über einen Index zugegriffen werden kann. Das Modell des Arrays für die Textverarbeitung ist ein sehr anschauliches und nützliches, wiewohl nicht verschwiegen werden soll, dass der Text meistens speicherintern als nullterminierte zeigerverkettete Liste repräsentiert ist.

In einer Textverarbeitung werden Strings bearbeitet: Sortieren, Suchen, Ersetzen, Einfügen, Löschen, usw.. Beispielhaft sollen folgende sechs Algorithmen besprochen werden:

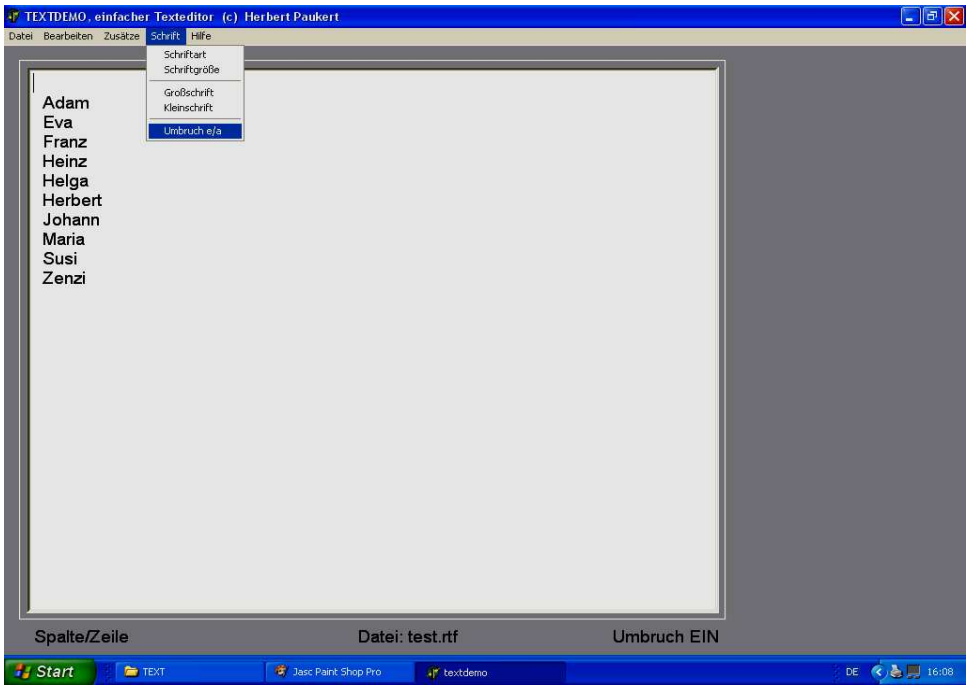
[6.01] Löschen mehrfacher Zeichen
[6.02] Suchen von Textteilen
[6.03] Suchen und Ersetzen
[6.04] Extraktion von numerischen Daten
[6.05] Statistische Datenauswertung
[6.06] Zusätzliche Routinen
[6.07] Ein einfacher Texteditor

Zur Demonstration der oben angeführten Algorithmen habe ich zwei Programme verfasst, welche auf einem Datenträger dem Leser zur freien Verfügung stehen.

Das Programm **SOSU.EXE** demonstriert anschaulich die verschiedenen Verfahren des Sortierens und Suchens. Es können dabei bis zu 100 000 zufällig erzeugte Zahlen verwendet werden. Bei allen Verfahren wird zusätzlich die benötigte Zeit gemessen.



Das Programm **TEXTDEMO.EXE** ist eine kleine Textverarbeitung und demonstriert anschaulich die Verfahren zur Verarbeitung von Strings. Zusätzlich kann der gesamte Text auf eine externe Datei abgespeichert und auch wieder in den Speicher geladen werden. Die weiterführenden Programme **TEXTPRO.EXE** und **PAUTEX.EXE** sind Textverarbeitungen mit zusätzlichen Funktionen.



[5] Suchen und Sortieren von Zufallszahlen

Alle vorgestellten Verfahren werden als Unterprogramme (Prozeduren oder Funktionen) definiert. Vorausgesetzt wird dabei, dass ein globaler Datentyp ***Bereich*** zur Speicherung von ganzen Zahlen im Hauptspeicher des Computers festgelegt ist.

```
const Max = 100000;           { Maximale Elementanzahl im Bereich }
type  Bereich = array[1..Max] of Integer;
var   Z   : Bereich;          { Bereich von höchstens Max ganzen Zahlen }
      Anz : Integer;          { Jeweilige Anzahl der Elemente (Anz ≤ Max) }
```

Soll beispielsweise eine Prozedur ***SORT*** den Bereich sortieren, dann wird im Prozedurenkopf der Zahlenbereich als Variablenparameter (call by reference) und die Elementanzahl als Werteparameter (call by value) übergeben. Das ist deswegen so, weil beim Sortieren die Anordnung der Zahlen im Bereich verändert wird, nicht aber ihre Anzahl. Die Kopfzeile der Prozedur hat dann folgende Form:
procedure SORT(var Z: Bereich; Anz: Integer);

[5.01] Erzeugung von Zufallszahlen mit Wiederholung

Der Zahlenbereich *Z* soll mit *Anz* ganzzahligen Zufallszahlen belegt werden. Dabei können sich die Zahlen wiederholen. Es muss zuerst unbedingt der interne Zufallsgenerator durch den Befehl *Randomize* initialisiert werden. Dann liefert der Funktionsaufruf *X := Random* eine zufällige reelle Zahl zwischen *0* und *1 (0 ≤ X < 1)*. Die Anweisung *X := Random(N)* hingegen liefert eine zufällige ganze Zahl zwischen *0* und *N (0 ≤ X < N)*.

```
procedure Zuf1(var Z: Bereich; Anz: Integer);
{ Erzeugung von Zufallszahlen MIT Wiederholung von 1 bis Anz }
var I: Integer;
begin
  for I := 1 to Anz do Z[I] := Random(Anz) + 1;
end;
```

[5.02] Erzeugung von Zufallszahlen ohne Wiederholung

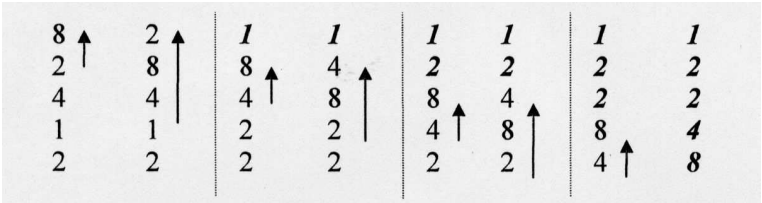
Die Aufgabenstellung ist wie bei [5.01]. Es dürfen sich aber jetzt die erzeugten Zufallszahlen **nicht** wiederholen. Am Beginn wird die erste Zufallszahl *Z[1]* erzeugt. Dann erfolgt die schrittweise Bestimmung der restlichen Zahlen. Dabei muss jede neue Zufallszahl mit allen vorher erzeugten Zahlen verglichen werden. Nur wenn sie mit keiner davon übereinstimmt, wird sie in den Bereich übernommen und zur nächsten Zahlen-Erzeugung fortgeschritten.

```
procedure Zuf2(var Z: Bereich; Anz: Integer);
{ Erzeugung von Zufallszahlen OHNE Wiederholung von 1 bis Anz }
var I,J: Integer;
    Gleich: Boolean;
begin
  Z[1] := Random(Anz) + 1;
  for I := 2 to Anz do
    repeat
      Z[I] := Random(Anz) + 1;
      Gleich := False;
      for J := 1 to (I-1) do
        if Z[I] = Z[J] then begin
           Gleich := True;
           break;
        end;
    until not Gleich;
end;
```

[5.03] Sortieren durch direktes Austauschen

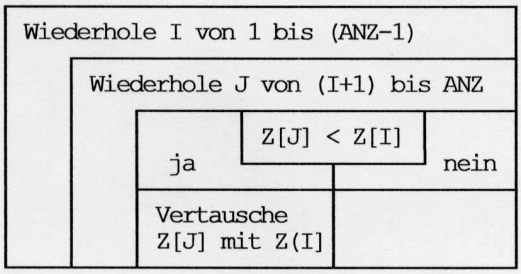
Sortieren und Durchsuchen von Datenbeständen im Hauptspeicher des Computers zählen zu den Grundaufgaben der EDV. Zuerst soll ein einfaches Sortierverfahren (Sortieren durch Austauschen) besprochen werden. Im nächsten Abschnitt wird ein anderer Algorithmus (Sortieren durch Einfügen) erläutert. Neben diesen beiden Verfahren gibt es noch einige andere Techniken, welche wesentlich schneller, aber auch wesentlich komplexer sind (z.B. der Quicksort-Algorithmus).

In aufsteigender Folge wird jedes Datenelement mit dem ersten verglichen, und wenn es kleiner als dieses ist, dann wird es mit diesem vertauscht. Wenn der ganze Datenbereich durchlaufen ist, steht das kleinste Element an erster Stelle. In einem zweiten Durchlauf wird das zweitkleinste Element an die zweite Stelle befördert, in einem dritten Durchlauf das drittkleinste Element an die dritte Stelle usw. Am Ende aller Durchläufe ist der Bereich sortiert. Zur Illustration sollen fünf Zahlen (8, 2, 4, 1, 2) sortiert werden:



Zur Sortierung dieser 5 Zahlen wurden also in 4 Durchläufen genau 7 Vertauschungen vorgenommen.

Struktogramm:

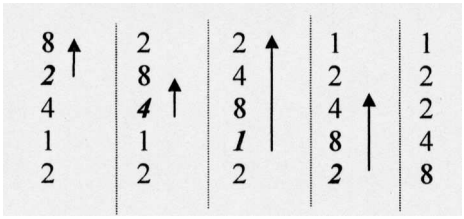



```
procedure Sort1(var Z: Bereich; Anz: Integer);
{ Sortieren durch Austauschen }
var X: Integer;
    I,J: Integer;
begin
  for I := 1 to Anz-1 do
    for J := I+1 to Anz do
      if Z[J] < Z[I] then begin
         X := Z[I]; Z[I] := Z[J]; Z[J] := X;
      end;
end;
```

[5.04] Sortieren durch direktes Einfügen

Will man ein neues Datenelement in einen bereits sortierten Bereich richtig einordnen, dann vergleicht man es, beginnend beim letzten Bereichselement, in absteigender Folge mit den einzelnen Elementen des Bereiches. Wenn das neue Element kleiner als ein Bereichselement ist, dann wird das Bereichselement um eine Stelle nach hinten verschoben, wodurch sein alter Platz im Bereich frei wird. Dieses Vergleichen und Verschieben wird so lange fortgesetzt, bis man auf ein Bereichselement trifft, das seinerseits kleiner als das neue Datenelement ist. Dann wird das neue Element an den zuletzt frei gewordenen Platz gestellt und ist somit richtig eingefügt.

Nimmt man nun keine neuen Datenelemente, sondern in aufsteigender Folge die Bereichselemente selbst und fügt sie in die Menge der vorangehenden Bereichselemente richtig ein, dann wird der gesamte Bereich schrittweise sortiert. Je größere Teilbereiche bereits sortiert vorliegen, umso weniger Daten müssen verschoben werden und umso schneller verläuft die Sortierung. Zur Illustration sollen wieder die fünf Zahlen (8, 2, 4, 1, 2) sortiert werden:

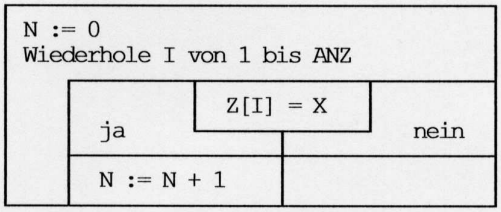



```
procedure Sort2(var Z: Bereich; Anz: Integer);
{ Sortieren durch Einfügen }
var X: Integer;
    I,J: Integer;
begin
  for I := 2 to Anz do begin
    X := Z[I];
    J := I - 1;
    while (X < Z[J]) and (J > 0) do begin
      Z[J+1] := Z[J];
      J := J - 1;
    end;
    Z[J+1] := X;
  end;
end;
```

[5.05] Sequentielles Suchen

Beim sequentiellen Suchen eines gegebenen Datenelementes *X* wird der Bereich *Z*, beginnend am Anfang, schrittweise (*I = 1*, ..., *Anz*) durchsucht. Bei Gleichheit von gesuchtem Element *X* mit dem jeweilig erreichten Bereichselement *Z[I]* wird in einer eigenen Zählvariablen *N* mitgezählt. Am Ende des Durchlaufes weiß man dann, wie oft das gesuchte Element im Datenbereich vorkommt. Beim sequentiellen Suchen ist es unwichtig, ob der Bereich bereits sortiert vorliegt oder nicht.

Struktogramm:

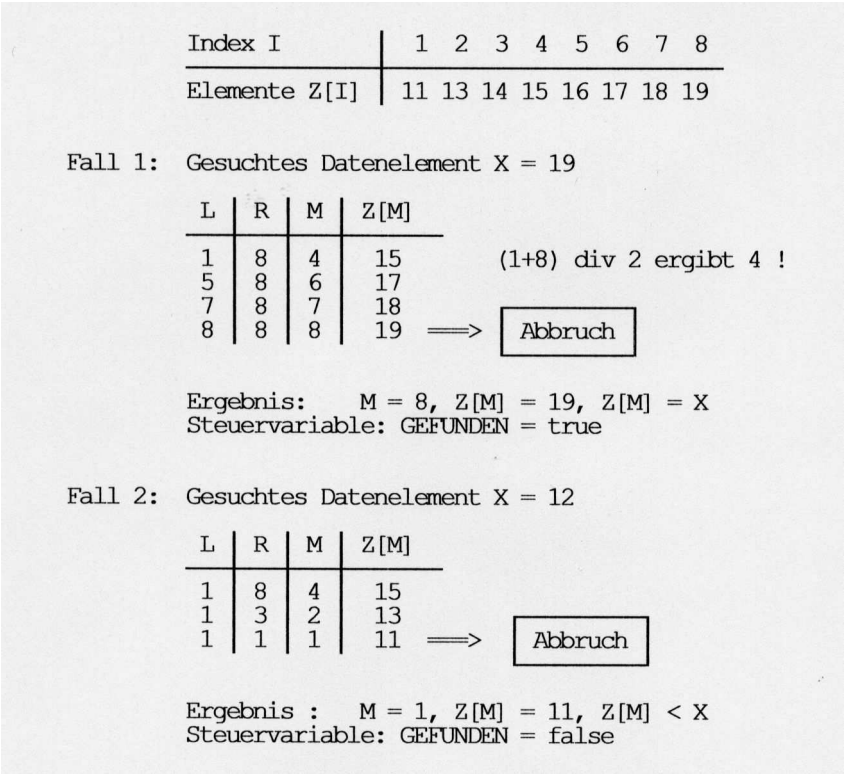



```
function Such1(var Z: Bereich; Anz: Integer; X: Integer): Integer;
{ Sequentielles Suchen nach dem Element X im Datenbereich Z }
{ Rückgabe der Anzahl der Fundstellen des Elementes         }
var I,N : Integer;
begin
  N := 0;
  for I := 1 to Anz do
    if Z[I] = X then N := N + 1;
  Result := N;
end;
```

[5.06] Binäres Suchen

Beim binären Suchen muss der Datenbestand bereits sortiert sein. Man zerlegt den Bereich *Z* durch Halbierung in zwei Teile und überprüft, ob das gesuchte Datenelement *X* im linken oder im rechten Teilbereich liegt. Diesen Teil zerlegt man wieder und wiederholt die fortgesetzte Intervallhalbierung so lange, bis der in Frage kommende Teilbereich auf ein einziges Element zusammenschrumpft. Entweder ist dieses Element das gesuchte Datenelement, oder der Suchvorgang war nicht erfolgreich. Die Variablen *L* und *R* bezeichnen die Indizes der Intervallränder und *M* ihr arithmetisches Mittel.

Beispiel: Gegeben ist die sortierte Zahlenmenge (11, 13, 14, 15, 16, 17, 18, 19).



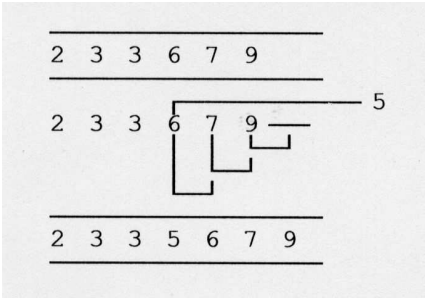

```
function Such2(var Z: Bereich; Anz: Integer; X: Integer): Integer;
{ Binäres Suchen nach dem Element X im Datenbereich Z }
{ Rückgabe der Position der Fundstelle des Elementes  }
var L,R,M: Integer;
    Gefunden: Boolean;
begin
  L := 1;
  R := Anz;
  Gefunden := FALSE;
  while (L <= R) and not Gefunden do begin
    M := (L+R) div 2;
    if X < Z[M] then R := M - 1;
    if X > Z[M] then L := M + 1;
    if X = Z[M] then Gefunden := True;
  end;
  if not Gefunden then M := 0;
  Result := M;
end;
```

Die obige Funktion liefert den Wert 0, wenn das gesuchte Element *X* im Bereich *Z* nicht gefunden wurde. Andernfalls wird die Bereichsposition des gefundenen Elementes übergeben.

[5.07] Elemente einfügen

Vorausgesetzt wird ein bereits sortierter Datenbereich *Z*. Will man ein neues Datenelement *X* in diesen Bereich richtig einordnen, dann vergleicht man es beginnend beim letzten Bereichselement in absteigender Folge mit den einzelnen Elementen des Bereiches. Wenn das neue Element kleiner als ein Bereichselement ist, dann wird das Bereichselement um eine Stelle nach hinten verschoben, wodurch sein alter Platz im Bereich frei wird. Dieses Vergleichen und Verschieben wird so lange fortgesetzt, bis man auf ein Bereichselement trifft, das seinerseits kleiner als das neue Datenelement ist. Dann wird das neue Element an den zuletzt frei gewordenen Platz gestellt und ist somit richtig eingefügt. Die unten stehende Funktion liefert zusätzlich noch die Position des eingefügten Elementes. Wichtig ist zu erwähnen, dass nach der Einfügung die Anzahlvariable *Anz* um Eins erhöht wird.

Beispiel: Die Zahl 5 soll in den Bereich (2, 3, 3, 6, 7, 9) eingefügt werden.

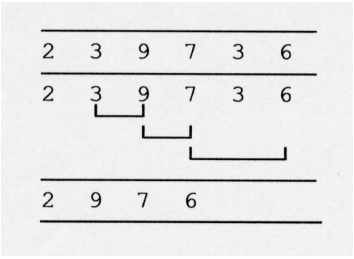



```
function Einfuegen(var Z: Bereich; var Anz: Integer; X: Integer): Integer;
{ Element X einfügen in den sortierten Datenbereich Z }
{ Rückgabe der Position des eingefügten Elementes     }
var I: Integer;
begin
  I := Anz;
  while (X < Z[I]) and (I > 0) do begin
    Z[I+1] := Z[I];
    I := I - 1;
  end;
  Z[I+1] := X;
  Result := I + 1;
  Anz := Anz + 1;
end;
```

[5.08] Elemente entfernen

Das Entfernen eines Datenelementes *X* kann in sortierten als auch in nicht sortierten Bereichen *Z* erfolgen. Im unten stehenden Verfahren wird der gesamte Bereich durchlaufen und jedes Bereichselement *Z[I]* mit dem Datenelement *X* verglichen. Nur bei einer Ungleichheit wird das Element im Bereich behalten. Auf diese Weise wird der ganze Bereich *Z* ohne Datenelement X neu aufgebaut. Die unten stehende Funktion liefert zusätzlich noch die Anzahl der Elemententfernungen. Wichtig ist zu erwähnen, dass nach den Entfernungen die Anzahlvariable *Anz* entsprechend verringert wird.

Beispiel: Die Zahl 3 soll aus dem Bereich (2, 3, 9, 7, 3, 6) entfernt werden.



```
function Entfernen(var Z: Bereich; var Anz: Integer; X: Integer): Integer;
{ Element X entfernen aus dem Datenbereich Z  }
{ Rückgabe der Anzahl der entfernten Elemente }
var I,J: Integer;
begin
  J := 0;
  for I := 1 to Anz do
    if X <> Z[I] then begin
       J := J + 1;
       Z[J] := Z[I];
    end;
  Result := Anz - J;
  Anz := J;
end;
```

[5.09] Sortieren mittels Quicksort

In einem unsortierten Feld wird ein ***Bezugselement*** ausgewählt und die Feldelemente so umgruppiert, dass ***links*** vom Bezugselement alle ***kleineren*** und ***rechts*** alle ***grösseren*** Elemente liegen. Dieser Prozess wird nun ***rekursiv*** in den beiden ***Teilfeldern*** wiederholt, bis jedes erzeugte Teilfeld nur mehr aus ***einem*** Element besteht. Danach ist das gesamte Feld sortiert.

Die Umgruppierung des Feldes in zwei Teilfelder wird nach folgendem Verfahren durchgeführt:

Als Bezugselement wird das Element in der Feldmitte genommen. Es wird vom linken Rand aufsteigend nach einem Feldelement gesucht, welches größer als das Bezugselement ist. In gleicher Weise wird vom rechten Rand absteigend nach einem Feldelement gesucht, welches kleiner als das Bezugselement ist. Diese beiden Elemente werden nun ausgetauscht und die Suche solange fortgesetzt, bis sich beide Suchläufe überschneiden. Danach befinden sich links vom Bezugselement die kleineren und rechts davon nur mehr die größeren Feldelemente.

```
procedure QSORT(var Z: Bereich; LO, HI: Integer);
{ Quick-Sort des Datenbereiches Z zwischen den }
{ Indexgrenzen LO und HI                       }

  procedure SORT(L,R: Integer);
  { Rekursive Quick-Sort-Routine }
  var I,J,M,X: Integer;
  begin
    I := L; J := R;
    M := Z[(L+R) div 2];
    repeat
      while M < Z[J] do J := J - 1;
      while Z[I] < M do I := I + 1;
      if I <= J then begin
         X := Z[I]; Z[I] := Z[J]; Z[J] := X;
         I := I + 1; J := J - 1;
      end;
    until I > J;
    if L < J then SORT(L,J);
    if I < R then SORT(I,R);
  end;

begin
  SORT(LO,HI);
end;
```

[5.10] Geprüfte Eingabe von Zahlen

Eine Integer-Zahl soll mit Hilfe einer visuellen Edit-Komponente als Dialogobjekt eingetastet und mit einer geeigneten Funktion geprüft und übergeben werden. Dazu wird der Text der Edit-Komponente auf der Stringvariablen *T* zwischengespeichert und der String *T* in eine Zahl *X* umgewandelt. Dann erfolgt die Prüfung, ob *X* zwischen den Grenzen *UG* und *OG* liegt.

```
function Dateneingabe(ED: TEdit; UG,OG: Integer): Integer;
{ Geprüfte Dateneingabe mittels Edit-Komponente ED }
var S, T : String;
    Code : Integer;
    X : Integer;
begin
  S := ' ACHTUNG: Nur Zahlen zwischen ' + IntToStr(UG) +
       ' und ' + IntToStr(OG) + ' eingeben ! ';
  T := ED.Text;
  Val(T,X,Code);
  if (Code <> 0) or (X < UG) or (X > OG) then ShowMessage(S);
  Result := X;
end;
```

Die Systemroutine *VAL* leistet die Umwandlung eines Strings in eine Zahl - sie wird im nachfolgenden Kapitel näher beschrieben. Der Befehl *IntToStr* hingegen wandelt eine Zahl unformatiert in einen String um. Die Systemroutine *ShowMessage(S)* zeigt die Zeichenkette *S* in einer Messagebox am Bildschirm an und wartet auf die Betätigung der <Enter>-Taste.

[5.11] Formatierte Ausgabe von Zahlen

Die einzelnen Zahlen eines Speicherarrays sollen in den Zeilen eines Textes so ausgegeben werden, dass in jeder Zeile genau *N* Zahlen nebeneinander, durch jeweils ein Blank getrennt, stehen. Als Behälterobjekt zur visuellen Darstellung des ganzen Textes kann ein Memo-Feld oder eine Richedit-Komponente dienen. Insgesamt werden dabei *ANZ* Zahlen ausgegeben.

```
procedure Datenausgabe(RE: TRichedit; var Z: Bereich; Anz,N: Integer);
{ Formatierte Datenausgabe in den Textzeilen einer Richedit-Komponente RE}
var S,T : String;
      I : Integer;
begin
  RE.Clear;
  S := ' ';
  for I := 1 to Anz do begin
    Str(Z[I]:6,T);
    S := S + T + ' ';
    if (I mod N) = 0 then begin
       RE.Lines.Add(S);
       S := ' ';
    end;
  end;
  RE.Lines.Add(S);
end;
```

Die Systemroutinen der Stringverkettung (*S + T*) und der Umwandlung einer Zahl in einen String mittels *STR* werden im nachfolgenden Kapitel beschrieben. Die Routine *MOD* bestimmt den ganzzahligen Rest einer Division. Der Befehl *Richedit.Clear* löscht den ganzen Text in der Richedit-Komponente, der Befehl *Richedit.Lines.Add(S)* hängt den String *S* als neue Zeile an den Text.

Damit die einzelnen Zahlen spaltenweise genau untereinander stehen, sollte als Schrift in der Richedit-Komponente eine nicht proportionale Schrift (Courier New) gewählt werden.

[6] Verarbeitung von Zeichenketten

Alle vorgestellten Verfahren werden als Unterprogramme (Prozeduren oder Funktionen) definiert. Vorausgesetzt wird dabei, dass mehrere Strings *A*, *B* und *S* als globale Variable im Hauptspeicher definiert sind. Außerdem muss ein globales Speicherarray vom Datentyp ***Bereich*** zur Aufnahme von numerischen Daten im Hauptspeicher des Computers festgelegt sein.

```
const Max  = 1000;           { Maximale Elementanzahl im Bereich }
type  Bereich = array[1..Max] of Real;
var   Z    : Bereich;        { Bereich von höchstens Max reellen Zahlen }
      Anz  : Integer;        { Jeweilige Anzahl der Zahlen (Anz ≤ Max)  }
      A,B,S: String;         { Drei Zeichenketten }
```

Soll beispielsweise eine Funktion ***REPLACE*** den Quellstring *S* nach dem Suchstring *A* absuchen und diesen an allen Fundstellen durch den Ersatzstring *B* ersetzen und zusätzlich die Anzahl der Ersetzungen ermitteln, dann wird im Funktionenkopf der Quellstring als Variablenparameter (call by reference) und Such- und Ersatzstring als Werteparameter (call by value) übergeben. Das ist deswegen so, weil beim Ersetzen der Quellstring verändert wird, nicht aber Such- und Ersatzstring. Als Wert der Funktion muss eine ganze Zahl definiert sein. Die Kopfzeile der Funktion hat dann folgende Form:
function REPLACE(var S: String; A,B: String): Integer;

In den nachfolgenden Algorithmen werden mehrere Routinen zur Stringverarbeitung benutzt, welche das Compilersystem automatisch zur Verfügung stellt:

function Length(S: String): Integer;	*Liefert die Länge des Strings S.*
function Pos(T,S: String): Integer;	*Liefert die erste Position von String T im String S.*
function Trim(S: String): String;	*Entfernt führende und nachfolgende Leerzeichen.*
function Uppercase(S: String): String;	*Umwandlung in Goßbuchstaben.*
function Lowercase(S: String): String;	*Umwandlung in Kleinbuchstaben.*
function Concat(S,T: String); String;	*Verkettung von S und T (auch mittels R := S + T)*
function Copy(S: String; P,L: Integer): String;	*Liefert von dem String S jenen Teilstring, der an der Position P beginnt und L Zeichen lang ist.*
function Delete(var S: String; P,L: Integer);	*Löscht im String S ab Position P genau L Zeichen.*
function Insert(T: String; var S: String; P: Integer);	*Fügt T in S an der Position P ein.*
function Val(S: String; var Z: Real; var P: Integer);	*Wandelt den String S in eine reelle Zahl Z um und liefert für P entweder 0 oder eine Fehlernummer > 0. Bei einem Fehler ist Z immer Null.*
procedure Str(Z:n:d: Real; var S: String);	*Wandelt die reelle Zahl Z in einen String S um - mit n Stellen und d Dezimalstellen.*
function StrToInt(S: String): Integer;	*Ungeprüfte Umwandlung von String in Zahl.*
function IntToStr(N: Integer): String;	*Unformatierte Umwandlung von Zahl in String.*
function UpCase(C: Char): Char;	*Umwandlung des Zeichens C in Goßbuchstaben.*
function Ord(C: Char): Integer;	*Liefert den ANSI-Code des Zeichens C.*
function Chr(N: Integer): Char;	*Liefert das Zeichen C mit dem ANSI-Code C. (auch mittels C := #N).*

[6.01] Löschen von mehrfachen Zeichen

In einem Quellstring *S* kommt das Zeichen (Character) *C* an verschiedenen Positionen mehrfach aufeinanderfolgend vor. Die Aufgabe besteht darin, dass an all diesen Positionen das Zeichen schließlich nur EINMAL steht.

Beispiel:

Mehrfach vorkommendes Zeichen *C*: ' ' (Leerzeichen)
Quellstring vor der Verarbeitung *S*: 'Man lernt nicht für die Schule, sondern für das Leben.'
Quellstring nach der Verarbeitung *S*: 'Man lernt nicht für die Schule, sondern für das Leben.'

Die gestellte Aufgabe wird dadurch gelöst, dass man den Quellstring *S* schrittweise durchläuft und jedes seiner Zeichen mit dem Zeichen *C* vergleicht. Fällt der Vergleich wahr aus und stimmt im Quellstring das nachfolgende Zeichen auch mit *C* überein, dann muss es gelöscht werden. Dazu wird die Systemroutine *DELETE* verwendet. Die Stringlänge wird mit der Systemroutine *LENGTH* abgefragt.

```
function RemoveMultipleChar(S: String; C: Char): String;
{ Entfernt mehrfache Zeichen C aus einem String S }
const Marker: Char = #1;
var   I : Integer;
begin
  S := S + Marker;
  I := 0;
  repeat
    I := I + 1;
    if (S[I] = C) and (S[I+1] = S[I]) then begin
       Delete(S,I,1);
       I := I - 1;
    end;
  until S[I] = Marker;
end;
```

[6.02] Suchen von Textteilen

Ein Quellstring *S* soll auf das Vorkommen eines Suchstrings *A* durchsucht werden und die Anzahl der Fundstellen soll ermittelt werden. Dazu muss zuerst im Quellstring *S* das erste Vorkommen des Suchstrings *A* gesucht werden. Die entsprechende Position liefert die Systemroutine *POS*. Dann wird an dieser Position im Quellstring *S* der Suchstring *A* gelöscht. Dieses Verfahren wird in einer Schleife so lange wiederholt, bis der Suchstring im Quellstring nicht mehr vorkommt. Bei jeder Fundstelle wird ein Zähler um Eins erhöht. Die Suche soll unabhängig von Großschrift und Kleinschrift erfolgen (case-insensitiv). Das Ganze erfolgt natürlich nicht mit den originalen Zeichenketten *S, A* sondern mit lokalen Kopien *S1*, *A1* davon.

```
function SearchString(var S: String; A: String): Integer;
{ Sucht im Text S den Suchstring A und liefert die Anzahl der Fundstellen. }
var S1,A1 : String;
    L,P,N : Integer;
begin
  S1 := UpperCase(S);
  A1 := UpperCase(A);
  L := Length(A1);
  N := 0;
  repeat
    P := Pos(A1,S1);
    if (P > 0) then begin
       N := N + 1;
       Delete(S1,P,L);
    end;
   until (P = 0);
   Result := N;
end;
```

[6.03] Suchen und Ersetzen

Ein Quellstring *S* soll auf das Vorkommen eines Suchstrings *A* durchsucht und dieser an allen Fundstellen durch den Ersatzstring *B* ersetzt werden. Zusätzlich wird die Anzahl der Fundstellen ermittelt. Suchen und Ersetzen soll unabhängig von Groß- oder Kleinschrift erfolgen (case-insensitiv). Ein Problem kann sich dann ergeben, wenn der Suchstring *A* ein Teil des Ersatzstrings *B* ist.

Um das Problem zu lösen, kann folgendes zweistufiges Verfahren angewendet werden: In einer ersten Wiederholungsschleife wird der Suchstring *A* durch ein nicht druckbares Zeichen (z.B. ein Zeichen mit dem ANSI-Code #1) ersetzt. In einer zweiten Wiederholungsschleife wird dann dieses Zeichen durch den Ersatzstring *B* ersetzt. Die Systemroutinen *DELETE* und *INSERT* werden dabei benutzt.

```
function ReplaceString(var S: String; A,B: String): Integer;
{ sucht im Text S den Suchstring A und ersetzt ihn }
{ durch den Ersatzstring B und liefert die Anzahl. }
const Marker : Char = #1;
var   S1,A1  : String;
      L,P,N  : Integer;
begin
  S1 := UpperCase(S);
  A1 := UpperCase(A);
  L  := Length(A1);
  N  := 0;
  repeat
    P := Pos(A1,S1);
    if (P > 0) then begin
       N := N + 1;
       Delete(S1,P,L); Insert(Marker,S1,P);
       Delete(S,P,L); Insert(Marker,S,P);
    end;
   until (P = 0);
   repeat
     P := Pos(Marker,S);
     if (P > 0) then begin
        Delete(S,P,1); Insert(B,S,P);
     end;
   until (P = 0);
   Result := N;
end;
```

[6.04] Extraktion von numerischen Daten

In einem Quellstring *S* kommen aufeinanderfolgende Zahlen vor, welche durch Beistriche getrennt sind (CSV, Comma Separated Values). Diese Zahlen sollen herausgebrochen und in ein Speicherarray abgelegt werden. In der ersten Schublade mit dem Index 0 wird entweder die Anzahl der extrahierten Zahlenwerte oder Null gespeichert werden. Null wird dann gespeichert, wenn bei einer Umwandlung einer Zeichenkette in eine Zahl ein Fehler aufgetreten ist, d.h. der Text zwischen zwei Kommas keine Zahl darstellt.

Die Aufgabe wird derart gelöst, dass in einer Schleife das Separatorzeichen (in unserem Fall das Komma) solange im Quellstring *S* gesucht wird bis es nicht mehr vorkommt. Der jeweilige Text zwischen zwei Fundstellen des Separators wird mit der Systemroutine *COPY* herausgebrochen und mittels *VAL* in eine Zahl umgewandelt. Diese wird dann in die jeweils nächste Schublade des Arrays gespeichert. Tritt bei einer Zahlenumwandlung ein Fehler auf, dann wird das in einer Hilfsvariablen festgehalten, andernfalls wird eine Zählvariable um Eins erhöht. Ist man am Ende des Quellstrings *S* angelangt, dann wird in seine erste Schublade mit dem Index 0 entweder die Anzahl der gespeicherten Zahlenwerte oder Null abgelegt. Die Daten sind reelle Zahlen und bei Dezimalzahlen muss ein Dezimalpunkt geschrieben werden.

Beispiel: *S* := '2.15,-0.725,31,9.6,-18,Herbert,-67.561';

Der String *S* enthält sieben Komma Separated Values, wovon sechs Werte numerisch sind und ein Wert nicht numerisch ist.

```
procedure ExtractValues(S: String; var Z: Bereich);
{ Komma Separated Values (CSV) aus einem String S extrahieren }
{ und in das Array Z speichern. Die Anzahl aller extrahierter }
{ Dezimalzahlen steht dann in Z[0]. }
const SEP : String = ',';
var   T   : String;
      X   : Real;
      N,P,Code : Integer;
      Error    : Boolean;
begin
  if Copy(S,Length(S),1) <> SEP then S := S + SEP;
  Error := False;
  N := 0;
  repeat
    P := Pos(SEP,S);
    if (P > 0) then begin
       N := N + 1;
       T := Trim(Copy(S,1,P-1));
       Val(T,X,Code);
       Z[N] := X;
       if (Code <> 0) then Error := True;
       S := Copy(S,P+1,Length(S));
    end;
  until (P = 0);
  if Error then Z[0] := 0
           else Z[0] := N;
end;
```

[6.05] Statistische Datenauswertung

In einer Textverarbeitung wird jener Textteil markiert, der aufeinanderfolgende numerische Daten enthält, welche durch Beistriche getrennt sind (CSV). Zuerst werden wie in [6.04] aus dem markierten Text die reellen Zahlen in die Schubladen eines Speicherarrays transferiert. In der ersten Schublade mit dem Index 0 steht die Anzahl *ANZ* aller extrahierter Daten. Nun kann das Zahlenarray statistisch ausgewertet werden, indem mit Hilfe einer Wiederholungsschleife die kleinste Zahl *MIN*, die größte Zahl *MAX*, die Summe aller Zahlen *SUM*, die Summe aller Zahlenquadrate *QSUM*, der Mittelwert *MWT* und die Streuung *STG* der Daten berechnet werden.

Die Ausgabe der ermittelten statistischen Kennwerte erfolgt in angehängten Zeilen in einer Richedit-Komponente des Formulars.

```
procedure Statis(RE: TRichedit; var Z: Bereich);
{ Statistische Datenauswertung eines Arrays Z mit reellen Zahlen. }
{ Die Datenanzahl steht in Z[0]. Die Ausgabe der statistischen    }
{ Kennwerte erfolgt in einer Richedit-Komponente des Formulares.  }
var  N,I : Integer;
     X, Sum,QSum,Min,Max,Mwt,Stg : Real;
     S: String;
begin
  N := Round(Z[0]);
  if (N = 0) then begin
     ShowMessage(' Auswertungs-Fehler ');
     Exit;
  end;
  Sum := 0; QSum := 0; Mwt := 0; Stg  := 0;
  Min := Z[1]; Max := Z[1];
```

```
  for I := 1 to N do begin
    X := Z[I];
    Sum  := Sum + X;
    QSum := QSum + X * X;
    if X < Min then Min := X;
    if X > Max then Max := X;
  end;
  Mwt := Sum / N;
  Stg := Sqrt((QSum / N) - (Mwt * Mwt));

  with RE do begin
    Lines.Add(' ');
    Str(N,S); Lines.Add('Anzahl = ' + Trim(S));
    Str(Min:10:2,S); Lines.Add('Minimum = ' + Trim(S));
    Str(Max:10:2,S); Lines.Add('Maximum = ' + Trim(S));
    Str(Sum:10:2,S); Lines.Add('Summe = ' + Trim(S));
    Str(Mwt:10:2,S); Lines.Add('Mittelwert = ' + Trim(S));
    Str(Stg:10:2,S); Lines.Add('Streuung = ' + Trim(S));
    Lines.Add(' ');
  end;
end;
```

[6.06] Zusätzliche Routinen

Die Auswahl von Textdateien zum Laden oder Speichern erfolgt mit der Hilfe so genannter visueller Dialogboxen (***OpenDialog, SaveDialog***). Das Laden einer ausgewählten Textdatei in eine Richedit-Komponente übernimmt der Befehl ***Richedit.LoadFromFile(S)***, das Speichern wird mit Hilfe des Befehls ***Richedit.SaveToFile(S)*** durchgeführt. Dabei enthält der String S den Namen der Datei. Der Textausdruck erfolgt mittels ***Richedit.Print('')***. Der Transfer von markiertem Text zu und von der Zwischenablage erfolgt mit Hilfe der Befehle ***Richedit.CutToClipboard***, ***Richedit.CopyToClipboard*** und ***Richedit.PasteFromClipboard***. Der Befehl ***Application.Terminate*** beendet das Programm. Alle diese Objekte mit ihren Eigenschaften und Methoden werden natürlich von DELPHI zur Verfügung gestellt und können vom Programmierer in geeigneter Weise in einem Formular verwendet werden.

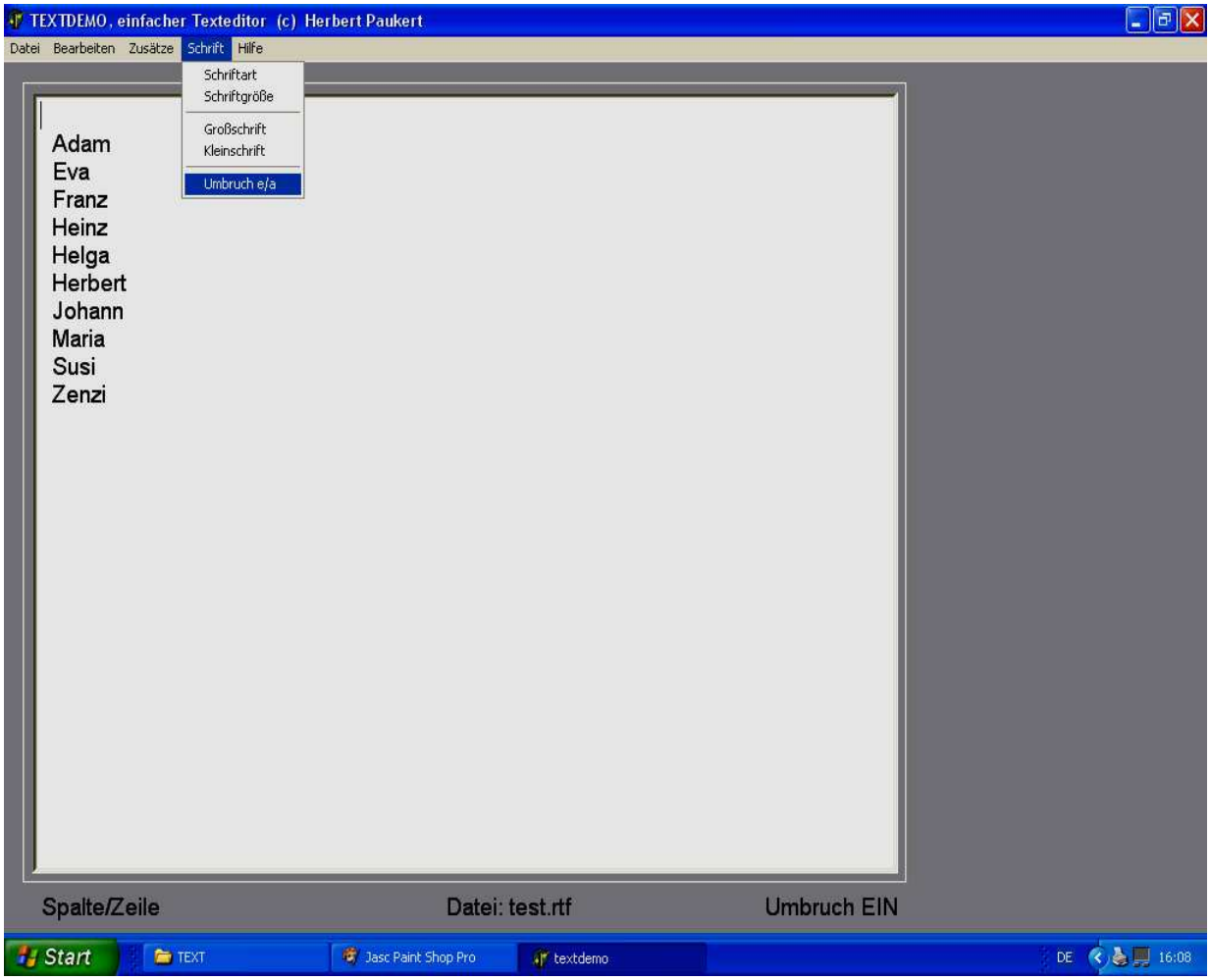
Ich hoffe, dass es mir mit diesem Artikel über Algorithmen gelungen ist, den Leser (Schüler, Lehrer) zu motivieren, selbständige Programmroutinen zu entwickeln. Das Unterprogramm ***ExtractValues*** beispielsweise ist ein sehr nützliches Werkzeug, um auf einfache und bequeme Weise Zahlenwerte einzugeben und diese dann weiter zu verarbeiten. In der Mathematik könnten sie als Koeffizienten von Polynomfunktionen oder von linearen Gleichungssystemen dienen. In den Naturwissenschaften könnten sie Messwerte darstellen, welche in spezifischer Art und Weise dann ausgewertet werden. Diese Liste ließe sich noch beliebig fortsetzen.

Das Programm ***TEXTDEMO.EXE*** ist ein einfacher Texteditor, welcher die oben dargestellten Textverwaltungs-Routinen enthält. Auf den folgenden Seiten ist das komplette Listing des Programms ausgedruckt.

Das Programm ist mit einem Hauptmenü ausgestattet und das globale Suchen und Ersetzen von Textstellen in einer Richedit-Komponente erfolgt mit den Routinen ***SearchAllText*** und ***ReplaceAllText***.

Dabei wird die Funktion ***Richedit.FindText(SText, StartPos, EndPos, SType)*** verwendet, welche die erste Fundstelle vom Suchtext ***SText*** als Integerzahl liefert. Gesucht wird von ***StartPos*** bis ***EndPos***, und ***SType*** ist ein Parameter vom Typ ***TSearchTypes***, welcher die Art und Weise des Suchens spezifiziert. Wird **[]** für ***STyp*** genommen, so wird unabhängig von Groß- und Kleinschreibung gesucht. Wenn die Suche erfolglos ist, dann wird die Zahl -1 zurückgeliefert.

[6.07] Ein einfacher Texteditor (TEXTDEMO)




```
unit textdemo_u;
// Demo für ausgewählte Textverarbeitungs-Routinen (c) Herbert Paukert

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Clipbrd,
     Controls, Forms, Dialogs, StdCtrls, Math, Printers, ExtCtrls, ComCtrls,
     Menus;

type
  TForm1 = class(TForm)
    RichEdit1: TRichEdit;
    Memo1: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Bevel1: TBevel;
    FontDialog1: TFontDialog;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
```

```
    MainMenu1: TMainMenu;
      Datei1: TMenuItem;
      Neu1: TMenuItem;
      ffnen1: TMenuItem;
      Speichern1: TMenuItem;
      Drucken1: TMenuItem;
      Ende1: TMenuItem;
    Bearbeiten1: TMenuItem;
      AllesMarkieren1: TMenuItem;
      Ausschneiden1: TMenuItem;
      Kopieren1: TMenuItem;
      Einfgen1: TMenuItem;
      N1: TMenuItem;
      Suchen1: TMenuItem;
      Ersetzen1: TMenuItem;
    Zustze1: TMenuItem;
      SortierenText1: TMenuItem;
      SortierenZahl1: TMenuItem;
      Statistik1: TMenuItem;
    Schrift1: TMenuItem;
      Schriftart1: TMenuItem;
      Schriftgre1: TMenuItem;
      N2: TMenuItem;
      Groschrift1: TMenuItem;
      Kleinschrift1: TMenuItem;
      N3: TMenuItem;
      Umbruchea1: TMenuItem;
    Hilfe1: TMenuItem;

    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);

    procedure Neu1Click(Sender: TObject);
    procedure ffnen1Click(Sender: TObject);
    procedure Speichern1Click(Sender: TObject);
    procedure Drucken1Click(Sender: TObject);
    procedure Ende1Click(Sender: TObject);
    procedure AllesMarkieren1Click(Sender: TObject);
    procedure Ausschneiden1Click(Sender: TObject);
    procedure Kopieren1Click(Sender: TObject);
    procedure Einfgen1Click(Sender: TObject);
    procedure Suchen1Click(Sender: TObject);
    procedure Ersetzen1Click(Sender: TObject);
    procedure SortierenText1Click(Sender: TObject);
    procedure SortierenZahl1Click(Sender: TObject);
    procedure Statistik1Click(Sender: TObject);
    procedure Schriftart1Click(Sender: TObject);
    procedure Schriftgre1Click(Sender: TObject);
    procedure Groschrift1Click(Sender: TObject);
    procedure Kleinschrift1Click(Sender: TObject);
    procedure Umbruchea1Click(Sender: TObject);
    procedure Hilfe1Click(Sender: TObject);

  private { Private declarations }
  public  { Public declarations }
end;

var  Form1: TForm1;
```

```
implementation
{$R *.dfm}

type  ZFeld  = array[0..1000] of Real;          // Zahlenfeld
var   Verz,FName,Ext: String;                   // Ordner,Datei,Extension
      MyList: TStringList;                      // Stringliste
      ZF    : ZFeld;                            // Zahlenfelder
      L4W   : Integer;                          // originale Label4.Width

procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
const SW: Integer = 1024;
      SH: Integer = 768;
      FS: Integer = 96;
      FL: Integer = 120;
var   X,Y,K: Integer;
      V0,V : Real;
begin
  with F do begin
    Scaled := True;
    X  := Screen.Width;
    Y  := Screen.Height;
    K  := Font.PixelsPerInch;
    V0 := SH / SW;
    V  := Y / X;
    if V < V0 then ScaleBy(Y,SH)
              else ScaleBy(X,SW);
    if (K <> FS)  then ScaleBy(FS,K);
    WindowState := wsMaximized;
  end;
end;

function FillString(N: Integer; C: Char): String;
// Erzeugt einen String aus N Zeichen C
var S : String;
    I : Integer;
begin
  S := '';
  For I := 1 to N do S := S + C;
  Result := S;
end;

function SearchAllText(RE: TRichEdit; S: String): Integer;
// Sucht überall in RE den String S und übergibt die Anzahl der Fundstellen
var E,L,P,N: Integer;
begin
  E := Length(S);
  L := Length(RE.Text);
  N := 0; P := 0;
  repeat
    P := RE.FindText(S,P,L,[]);     // Interne Suchroutine von RichEdit
    if P <> -1 then begin
       N := N + 1;
       P := P + E;
    end;
  until (P = -1);
  Result := N;
end;
```

```
function ReplaceAllText(RE: TRichEdit; S,T: String): Integer;
// Sucht überall in RE den String S und ersetzt ihn durch den String T
var E,F,L,P,N: Integer;
begin
  E := Length(S);
  F := Length(T);
  L := Length(RE.Text);
  N := 0;
  P := 0;
  repeat
    P := RE.FindText(S,P,L,[]);     // Interne Suchroutine von RichEdit
    if P <> -1 then begin
       N := N + 1;
       RE.SelStart  := P;
       RE.SelLength := E;
       RE.SelText   := T;
       P := P + F;
    end;
  until (P = -1);
  Result := N;
end;

procedure ExtractValues(S,SEP: String; var ZF: ZFeld);
// CSV-Liste aus einen String S extrahieren und in das Array ZF speichern
// Die Anzahl der extrahierten Dezimalzahlen steht dann in ZF[0]
var T : String;
    Z : Real;
    N,P,Code : Integer;
    Error    : Boolean;
begin
  if S[Length(S)] <> SEP then S := S + SEP;
  Error := False;
  N := 0;
  Repeat
    P := Pos(SEP,S);
    if P > 0 then begin
       N := N + 1;
       T := Trim(Copy(S,1,P-1));
       Val(T,Z,Code);
       ZF[N] := Z;
       if Code <> 0 then Error := TRUE;
       S := Copy(S,P+1,Length(S));
    end;
  Until P = 0;
  if Error then ZF[0] := 0 else ZF[0] := N;
end;

procedure Statis;
// Statistische Datenauswertung - Kenndaten
var  N,I: Integer;
     X, Sum,QSum,Min,Max,Mwt,Stg: Real;
     SEP,S: String;
begin
with Form1.RichEdit1 do begin
  SEP := ',';
  N := SearchAllText(Form1.RichEdit1,SEP);
  if N = 0 then SEP := #13;
```

```
  S := SelText;
  ExtractValues(S,SEP,ZF);
  N := Round(ZF[0]);
  if (N = 0) then begin
     ShowMessage('Auswertungs-Fehler !');
     Exit;
  end;
  Sum := 0; QSum := 0;
  Mwt := 0; Stg := 0;
  Min := ZF[1]; Max := ZF[1];
  For I := 1 to N do begin
      X := ZF[I];
      Sum := Sum + X;
      QSum := QSum + X * X;
      if X < Min then Min := X;
      if X > Max then Max := X;
  end;
  Mwt := Sum / N;
  Stg := Sqrt((QSum / N) - (Mwt * Mwt));
  Lines.Add(' ');
  Lines.Add('------------------------------');
  Lines.Add('Anzahl = ' + IntToStr(N));
  Str(Min:10:2,S); Lines.Add('Minimum = ' + Trim(S));
  Str(Max:10:2,S); Lines.Add('Maximum = ' + Trim(S));
  Str(Sum:10:2,S); Lines.Add('Summe = ' + Trim(S));
  Str(Mwt:10:2,S); Lines.Add('Mittelwert = ' + Trim(S));
  Str(Stg:10:2,S); Lines.Add('Streuung = ' + Trim(S));
  Lines.Add('------------------------------');
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  GetDir(0,Verz);
  Form1.Color := RGB(150,160,180);
  FitForm(Form1);
end;

procedure TForm1.FormActivate(Sender: TObject);
// Initialisierungen
var S: String;
    k: Integer;
begin
  if Font.PixelsPerInch = 120 then k := 130 else k := 170;
  Label4.Font.Name  := 'Arial';
  Label4.Font.Size  := 12;
  Label4.Font.Style := [];
  L4W := Label4.Width div 2;
  Label4.Caption    := FillString(k,#32) + ']< A4-Rand';
  RichEdit1.Width := Label4.Width - L4W;
  RichEdit1.Left := (Screen.Width - RichEdit1.Width) div 2;
  RichEdit1.Top  := (Screen.Height - RichEdit1.Height) div 4;
  Label1.Left := RichEdit1.Left;
  Label2.Left := RichEdit1.Left + RichEdit1.Width div 2;
  Label3.Left := RichEdit1.Left + RichEdit1.Width - Label3.Width;
  Label4.Left := RichEdit1.Left;
```

```
  Label1.Top  := RichEdit1.Top + RichEdit1.Height + 12;
  Label2.Top  := Label1.Top;
  Label3.Top  := Label1.Top;
  Label4.Top  := RichEdit1.Top - Label4.Height - 12;
  Bevel1.Left := RichEdit1.Left - 6;
  Bevel1.Top  := RichEdit1.Top  - 6;
  Bevel1.Width  := RichEdit1.Width + 12;
  Bevel1.Height := RichEdit1.Height + 12;
  Memo1.Left  := RichEdit1.Left + 8;
  Memo1.Top   := RichEdit1.Top  + 8;
end;

procedure TForm1.Neu1Click(Sender: TObject);
// Neuer Text
begin
  RichEdit1.Clear;
  RichEdit1.SetFocus;
end;

procedure TForm1.ffnen1Click(Sender: TObject);
// Bestehende Textdatei lesen
begin
with Form1 do begin
  With OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'RTF-Textdateien (*.rtf)|*.rtf|' +
                 'ANSI-Textdateien (*.txt)|*.txt|' +
                 'Alle Dateien (*.*)|*.*';
       DefaultExt := 'rtf';
       Options    := [ofFileMustExist];
       FileName   := '';
       if Execute then begin
          Verz  := Trim(ExtractFilePath(FileName));
          Ext   := Trim(LowerCase(ExtractFileExt(FileName)));
          FName := Trim(ExtractFileName(FileName));
          if (Ext = '.rtf') then RichEdit1.PlainText := False
                            else RichEdit1.PlainText := True;
          Try
            RichEdit1.Clear;
            RichEdit1.Lines.LoadFromFile(FileName);
          Except
            MessageBox(0,'Daten-Fehler','Problem',16);
          end;
       end;
  end;
  Label2.Caption := 'Datei: ' + FName;
end;
end;

procedure TForm1.Speichern1Click(Sender: TObject);
// Aktuellen Text speichern
begin
With Form1 do begin
  With SaveDialog1 do begin
       InitialDir := Verz;
       Filter := 'RTF-Textdateien (*.rtf)|*.rtf|' +
                 'ANSI-Textdateien (*.txt)|*.txt|' +
                 'Alle Dateien (*.*)|*.*';
```

```
      if (FName = '') or (Pos('.',FName) = 0) then DefaultExt := 'txt'
                                              else DefaultExt := 'rtf';
      FilterIndex := OpenDialog1.FilterIndex;
      Options    := [ofOverwritePrompt];
      FileName   := FName;
      if Execute then begin
         Verz  := Trim(ExtractFilePath(FileName));
         Ext   := Trim(LowerCase(ExtractFileExt(FileName)));
         FName := Trim(ExtractFileName(FileName));
         if (Ext = '.rtf') then RichEdit1.PlainText := False
                           else RichEdit1.PlainText := True;
         Try
            RichEdit1.Lines.SaveToFile(FileName);
         Except
            MessageBox(0,'Daten-Fehler','Problem',16);
         end;
      end;
  end;
  Label2.Caption := 'Datei: ' + FName;
end;
end;

procedure TForm1.Drucken1Click(Sender: TObject);
begin
  RichEdit1.Print('');
  RichEdit1.SetFocus;
end;

procedure TForm1.Ende1Click(Sender: TObject);
// Programm beenden
begin
  Form1.Close;
end;

procedure TForm1.AllesMarkieren1Click(Sender: TObject);
// Bearbeiten - Alles Markieren
begin
  RichEdit1.SelectAll;
end;

procedure TForm1.Ausschneiden1Click(Sender: TObject);
// Bearbeiten - Ausschneiden
begin
  RichEdit1.CutToClipBoard;
end;

procedure TForm1.Kopieren1Click(Sender: TObject);
// Bearbeiten - Kopieren
begin
  RichEdit1.CopyToClipBoard;
end;

procedure TForm1.Einfgen1Click(Sender: TObject);
// Bearbeiten - Einfügen
begin
  RichEdit1.PasteFromClipboard;
end;
```

```
procedure TForm1.Suchen1Click(Sender: TObject);
// Text suchen
var S: String;
    N: Integer;
begin
  S := InputBox('Suchtext eingeben','','');
  if (S = '') then begin
     ShowMessage('Falsche Texteingabe !');
     Exit;
  end;
  N := SearchAllText(RichEdit1,S);
  ShowMessage(IntToStr(N) + ' Fundstellen !');
end;

procedure TForm1.Ersetzen1Click(Sender: TObject);
// Text ersetzen
var S,T: String;
    P,N: Integer;
begin
  S := InputBox('Suchtext und Ersatztext','durch "/" getrennt eingeben' + #13,'');
  P := Pos('/',S);
  if (S = '') or (P = 0) then begin
     ShowMessage('Falsche Texteingabe !'); Exit;
  end;
  T := Copy(S,P+1,Length(S));
  S := Copy(S,1,P-1);
  N := ReplaceAllText(RichEdit1,S,T);
  ShowMessage(IntToStr(N) + ' Ersetzungen !');
end;

procedure TForm1.SortierenText1Click(Sender: TObject);
// Sortieren (Text)
begin
  if MessageBox(0,'Zeilen als Texte sortieren ?','Frage',36) = 6 then begin
     MyList.Assign(RichEdit1.Lines);
     MyList.Sort;
     RichEdit1.Lines.Assign(MyList);
     RichEdit1.SetFocus;
  end;
end;

procedure TForm1.SortierenZahl1Click(Sender: TObject);
// Sortieren (Zahl)
var I,J,N,Code1,Code2 : Integer;
    A,B : String;
    X,Y : Real;
begin
  if MessageBox(0,'Zeilen als Zahlen sortieren ?','Frage',36) = 6 then begin
     MyList.Assign(RichEdit1.Lines);
     N := MyList.Count;
     For I := 0 to N-2 do begin
         For J := I+1 to N-1 do begin
             A := MyList[I]; Val(A,X,Code1);
             B := MyList[J]; Val(B,Y,Code2);
             if (Code1 > 0) or (Code2 > 0) then begin
                ShowMessage(' Daten sind nicht numerisch !');  Exit;
             end;
```

```
            if Y < X then begin
               Mylist[I] := B;
               MyList[J] := A;
            end;
         end;
     end;
     RichEdit1.Lines.Assign(MyList);
     RichEdit1.SelStart := 0;
  end;
end;

procedure TForm1.Statistik1Click(Sender: TObject);
// Einfache statistische Datenauswertung
begin
 Statis;
 RichEdit1.SetFocus;
end;

procedure TForm1.Schriftart1Click(Sender: TObject);
// Schriftart
begin
  FontDialog1.Font.Assign(RichEdit1.SelAttributes);
  if FontDialog1.Execute then
     RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  RichEdit1.SetFocus;
end;

procedure TForm1.Schriftgre1Click(Sender: TObject);
// Schriftgröße
var S: String;
    Y,Err : Integer;
begin
  S := InputBox('Schriftgröße (6 - 36)','','14');
  Val(S,Y,Err);
  if (Err <> 0) or (Y < 6) or (Y > 36) then Y := 14;
  RichEdit1.SelAttributes.Size := Y;
end;

procedure TForm1.Groschrift1Click(Sender: TObject);
// Großschrift
var S : String;
begin
  if RichEdit1.SelLength = 0 then Exit;
  S := UpperCase(RichEdit1.SelText);
  RichEdit1.SelLength := Length(S);
  RichEdit1.SelText := S;
end;

procedure TForm1.Kleinschrift1Click(Sender: TObject);
// Kleinschrift
var S : String;
begin
  if RichEdit1.SelLength = 0 then Exit;
  S := LowerCase(RichEdit1.SelText);
  RichEdit1.SelLength := Length(S);
  RichEdit1.SelText := S;
end;
```

```
procedure TForm1.Umbruchea1Click(Sender: TObject);
// Zeilenumbruch EIN/AUS
var N: Integer;
begin
  RichEdit1.WordWrap := not RichEdit1.WordWrap;
  if RichEdit1.WordWrap then begin
     Label3.Caption  := 'Umbruch EIN'
  end
  else begin
     Label3.Caption  := 'Umbruch AUS';
  end;
  RichEdit1.SetFocus;
end;

procedure TForm1.Hilfe1Click(Sender: TObject);
begin
   Memo1.Visible := Not Memo1.Visible;
end;

end.
```