T E X T

Textverarbeitung

© Herbert Paukert

[01] Routinen zur Stringverarbeitung

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;        { Aktuelle 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).*

[02] Textwiederholungen entfernen

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

Beispiel:

Mehrmals 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 Routine *DELETE* verwendet. Das Stringende wird mit einem Spezialzeichen markiert.

```
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;
  Result := Copy(S,1,Length(S)-1);
end;
```

[03] Suchen von Texten

Ein Quelltext *S* soll **erstens** auf das Vorkommen eines Suchstrings *A* ab der Startposition *Start* durchsucht und die Position *P* der Fundstelle ermittelt werden. Der Parameter *Flag* gibt an, ob die Suche unabhängig von Groß-/Kleinschrift (Case-insensitiv, *Flag = 0*) oder davon abhängig (Case-sensitiv, *Flag = 1*) erfolgen soll. Die entsprechende Position *P* liefert die Systemroutine *POS*. Wenn dabei 0 herauskommt, dann wurde nichts gefunden.

In der **zweiten** Routine wird die Anzahl der Fundstellen des Suchstrings *A* im ganzen Quelltext *S* ermittelt. Dabei wird in einer Wiederholungsschleife der Quelltext *S* so lange durchlaufen, bis der Suchstring *A* nicht mehr gefunden wird.

```
function SearchFirst(S,A: String; Start,Flag: Integer): Integer;
{ Sucht im Text S den Suchstring A ab Start und liefert die Fundposition }
var P : Integer;
begin
  S := Copy(S,Start+1,Length(S));
  if Flag = 0 then begin
     S := UpperCase(S);
     A := UpperCase(A);
  end;
  P := Pos(A,S);
  if P = 0 then Result := 0
           else Result := Start + P;
end;

function SearchAll(S,A: String; Flag: Integer): Integer;
{ String A im ganzen Text S suchen und die Anzahl der Fundstellen liefern }
var P,N: Integer;
begin
  P := 0;
  N := 0;
  repeat
    P := SearchFirst(S,A,P,Flag);
    if (P > 0) then begin
       N := N + 1;
       P := P + Length(A);
     end;
   until (P = 0);
   Result := N;
end;
```

[04] Suchen und Ersetzen von Texten

Ein Quellstring *S* soll erstens auf das Vorkommen eines Suchstrings *A* ab der Startposition *Start* durchsucht und dieser an der ersten Fundstelle durch den Ersatzstring *B* ersetzt werden. Die Position der Fundstelle wird zurückgeliefert. Der Parameter *Flag* gibt an, ob die Suche unabhängig von Groß-/Kleinschrift (Case-insensitiv, *Flag = 0*) oder davon abhängig (Case-sensitiv, *Flag = 1*) erfolgen soll. Die entsprechende Position liefert die Systemroutine *POS*. Wenn dabei 0 herauskommt, dann wurde nichts gefunden.

In der letzten Routine wird die Anzahl der Fundstellen bzw. Ersetzungen im ganzen Quelltext ermittelt. Es wird in einer Wiederholungsschleife der Quelltext *S* so lange durchlaufen, bis der Suchstring *A* nicht mehr gefunden wird. An jeder Fundstelle wird zuerst der Suchstring *A* gelöscht und dann der Ersatzstring *B* eingesetzt. Dabei werden die Systemroutinen *DELETE* und *INSERT* benutzt.

```
function SearchFirst(S,A: String; Start,Flag: Integer): Integer;
{ Sucht im Text S den Suchstring A ab Start und liefert die Fundposition }
var P : Integer;
begin
  S := Copy(S,Start+1,Length(S));
  if Flag = 0 then begin
     S := UpperCase(S);
     A := UpperCase(A);
  end;
  P := Pos(A,S);
  if P = 0 then Result := 0
           else Result := Start + P;
end;

function ReplaceFirst(var S: String; A,B:String; Start,Flag: Integer): Integer;
{ String A im Text S suchen und durch String B ersetzen }
{ und die erste Fundstelle ab Position Start liefern }
var P : Integer;
begin
  P := SearchFirst(S,A,P,Flag);
  if P > 0 then begin
     Delete(S,P,Length(A));
     Insert(B,S,P);
  end;
  Result := P;
end;

function ReplaceAll(var S: String; A,B:String; Flag: Integer): Integer;
{ String A im ganzen Text S suchen und durch String B ersetzen }
{ und die Anzahl N der Ersetzungen liefern }
var P,N : Integer;
begin
  P := 0;
  N := 0;
  repeat
    P := SearchFirst(S,A,P,Flag);
    if P > 0 then begin
       N := N + 1;
       Delete(S,P,Length(A));
       Insert(B,S,P);
       P := P + Length(B);
    end;
  until (P = 0);
  Result := N;
end;
```

[05] Extraktion von numerischen Daten

In einem Quellstring *S* kommen aufeinander folgende 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 soll 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. ein Text zwischen zwei Kommas keine Zahl darstellt.

Die Aufgabe wird derart gelöst, dass in einer Schleife das Separatorzeichen (in unserem Fall das Komma) so lange 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 abgespeichert. Tritt bei einer Zahlenumwandlung ein Fehler auf, dann wird das in der Hilfsvariablen *Error* 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* in obigem Beispiel enthält sieben Komma Separated Values, wovon sechs Werte numerisch sind und ein Wert nicht numerisch ist.

In der nachfolgenden Routine muss das Separatorzeichen *SEP* als Parameter übergeben werden. Bei CSV-Listen wird *SEP* ein Komma '**,**' zugewiesen.

```
procedure ExtractValues(S: String; SEP: Char; var Z: Bereich);
{ Durch den Separator SEP getrennte Zahlenwerte aus einem String S extrahieren }
{ und im Array Z speichern. Z[0] enthält die Anzahl aller extrahierter Werte.  }
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;
```

Das Unterprogramm ***ExtractValues*** ist ein sehr nützliches Werkzeug, um auf einfache und bequeme Weise Zahlenwerte einzugeben und diese dann weiterzuverarbeiten. In der Mathematik könnten sie beispielsweise als Koeffizienten von Polynomfunktionen oder von linearen Gleichungssystemen dienen. In den Naturwissenschaften könnten sie Messwerte darstellen, welche dann in spezifischer Art und Weise ausgewertet werden. Diese Liste ließe sich noch beliebig fortsetzen.

[06] Statistische Datenauswertung

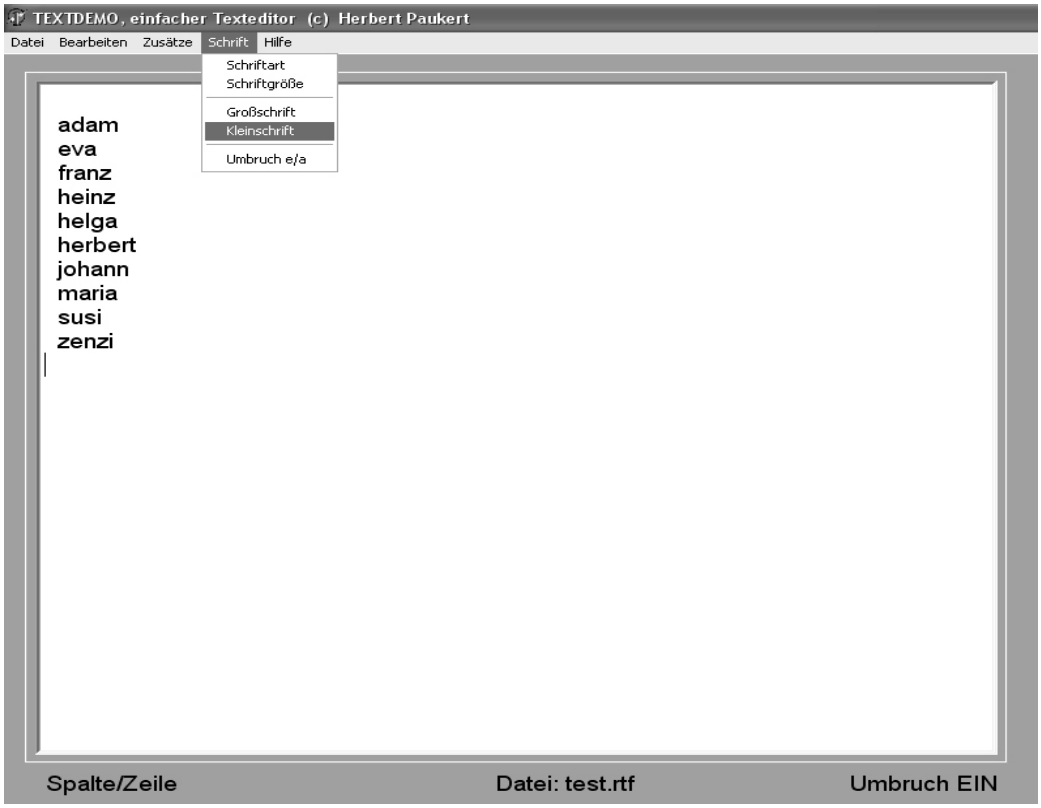
In einer Textverarbeitung wird jener Textteil markiert, der aufeinander folgende numerische Daten enthält, welche durch Beistriche getrennt sind (CSV). Zuerst werden wie im vorangehenden Buchabschnitt aus dem markierten Text die reellen Zahlen in die Schubladen eines Speicherarrays *Z* transferiert. In der ersten Schublade mit dem Index Null *Z[0]* steht die Anzahl *N* 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;
```

[07] Die Textverabeitung *TEXTPRO*



Das Programm ***TEXTPRO*** ist eine Textverarbeitung, die mit einem ***Menü*** verwaltet wird. Die zentrale mehrzeilige Editierkomponente ist dabei vom Typ ***TRichEdit***.

Datei

Neu	Neuen Text anlegen
Öffnen	Neuen Text aus Datei laden
Speichern	Text in Datei speichern
Drucken	Text ausdrucken
Ende	Programm beenden

Bearbeiten

Alles Markieren	Gesamten Text markieren
Ausschneiden	Markierten Text in die Zwischenablage ausschneiden
Kopieren	Markierten Text in die Zwischenablage kopieren
Einfügen	Text aus der Zwischenablage holen und einfügen
Suchen	Text suchen
Ersetzen	Text suchen und ersetzen

Zusätze

Sortieren (Text)	Zeilen als Text sortieren
Sortieren (Zahl)	Zeilen als Zahlenwerte sortieren
Statistik	Markierten Text statistisch auswerten
Komprimieren	Mehrfache Leerzeilen und Leerzeichen entfernen
Gehe zur Zeile	Zu einer bestimmten Zeile springen

Schrift

Schriftart	Schriftart und Textauszeichnung setzen
Schriftgröße	Zeichengröße der Schrift setzen
Hochstellen	Markierten Text hochstellen
Tiefstellen	Markierten Text tiefstellen
Normalstellen	Markierten Text normalstellen
Umbruch e/a	Zeilenumbruch ein- oder ausschalten

Hilfe ein/aus

Im Folgenden werden einige wichtige Routinen aus dem Quellcode von ***TEXTPRO*** beschrieben:

[1] Zu einer bestimmten Zeile (Y) und Spalte (X) im Text von *RichEdit* springen.

```
procedure GotoXY(RE: TRichEdit; X,Y: Integer);
// Positioniert den Cursor an die Spalte X und Zeile Y
var I,J,N: Integer;
begin
  if Y < 1 then Y := 1;
  if Y > RE.Lines.Count then Y := RE.Lines.Count;
  N := 0;
  For I := 0 to Y-2 do begin
      if RE.Lines[I] = '' then J := 2
      else J := Length(RE.Lines[I]) + 2;
      N := N + J;
  end;
  N := N + X - 1;
  RE.SelStart  := N;
  RE.SelLength := 0;
  SendMessage(RE.Handle,EM_SCROLLCARET,0,0);
end;
```

Weil *RE.Lines.Count* die Zeilenanzahl angibt und jede Zeile mit einem zweistelligen *EndOfLine*-Code (#13#10) schließt, müssen bis zur vorletzten Zeile pro gezählte Zeile genau zwei Zeichen zum Zeichenzähler *N* addiert werden. Am Ende werden dann noch die Spalten addiert. Diese Anzahl aller Textzeichen wird dann *RE.SelStart* zugewiesen, wodurch der Cursor automatisch an der gewünschten Textstelle positioniert ist.

Der Befehl *SendMessage(RE.Handle,EM_SCROLLCARET,0,0)* ermöglicht dabei, dass ein langer Text automatisch gescrollt wird.

[2] Suchen und Ersetzen von Texten in *RichEdit*.
(Weitersuchen mit der Taste <F3> mit Hilfe der Funktion *SearchFirst*)

```
var QUELLE,SUCH,ERSATZ: String;
    Posi: Integer;

function SearchFirst(S,A: String; Start,Flag: Integer): Integer;
{ Sucht im Text S den Suchstring A ab Start und liefert die Fundposition }
var P : Integer;
begin
  S := Copy(S,Start+1,Length(S));
  if Flag = 0 then begin
     S := UpperCase(S);
     A := UpperCase(A);
  end;
  P := Pos(A,S);
  if P = 0 then Result := 0
           else Result := Start + P;
end;
```

Mit Hilfe eines Schalters kann dann folgende Prozedur aufgerufen werden:

```
procedure Suchen;
begin
  QUELLE := Form1.RichEdit1.Text;
  SUCH   := Form1.Edit1.Text;              // Suchtext im ersten Edit-Feld
  ERSATZ := Form1.Edit2.Text;              // Ersatztext im zweiten Edit-Feld
  Posi := SearchFirst(QUELLE,SUCH,0,0);
  Form1.RichEdit1.SetFocus;
end;
```

Weitersuchen und Ersetzen werden dann mit folgender Prozedur ausgeführt:

```
procedure TForm1.RichEdit1.KeyUp(Sender: TObject; var Key: Word;
                                 Shift: TShiftState;
// Text ersetzen und mit der Taste <F3> weitersuchen
begin
  if (key = vk_F3) then begin
     if (Posi = 0) then Exit;
     QUELLE := RichEdit1.Text;
     RichEdit1.SelStart  := Posi-1;
     RichEdit1.SelLength := Length(SUCH);
     RichEdit1.Seltext   := ERSATZ;
     Posi := Posi + Length(ERSATZ);
     Posi := SearchFirst(QUELLE,SUCH,Posi,0);
  end;
end;
```

In der Textverabeitung ***TEXTPRO*** wird zum Suchen ein anderer Weg beschritten. Dort wird die Delphi-Dialogkomponente ***FindDialog*** verwendet. In ihre Methode ***FindDialog.Find*** wird dann die entsprechende Routine zum Suchen von Texten in einer ***RichEdit***-Komponente geschrieben.

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.

Über einen eigenen Schalter wird dann die Methode ***FindDialog.Execute*** aufgerufen.

Zum Suchen und Ersetzen wird die Delphi-Dialogkomponente ***ReplaceDialog*** verwendet. Die ausführliche Programmierung findet man am Ende dieses Buchteiles im Listing des Quellcodes von ***TEXTPRO***.

[3] Die Textzeilen von *RichEdit* alphabetisch sortieren.

Dazu muss eine Stringliste *MyList* am Programmanfang mittels *MyList := TStringList.Create* im Speicher erzeugt werden. Diese dient zur Aufnahme **aller** Zeilen von *RichEdit*. Dann kann mit der Methode *MyList.Sort* die Liste sortiert werden. Den Abschluss bildet der Rücktransfer der Liste nach *RichEdit*. Dabei gehen natürlich in *RichEdit* alle Textauszeichnungen verloren.

```
procedure Sortieren;
begin
  MyList.Assign(Form1.RichEdit1.Lines);
  MyList.Sort;
  Form1.RichEdit1.Lines.Assign(MyList);
end;
```

[4] Einen markierten Text aus *RichEdit* zeilenweise in eine Stringliste *MyList* kopieren.

Dazu muss eine Stringliste *MyList* am Programmanfang mittels *MyList := TStringList.Create* im Speicher erzeugt werden. Diese dient dann zur Aufnahme der **markierten** Zeilen von *RichEdit*.

```
procedure EditToList(RE:TRichEdit; var LI: TStringList);
{ Transfer markierter Zeilen aus RichEdit in eine Stringliste }
const SEP = #13#10;
var T,S : String;
    P,L : Integer;
begin
  MyList.Clear;
  T := RE.SelText;
  L := Length(SEP);
  if RE.SelLength = 0 then Exit;
  if Copy(T,Length(T)+1-L,L) <> SEP then T := T + SEP;
```

```
  repeat
    P := Pos(SEP,T);
    if P > 0 then begin
       S := Trim(Copy(T,1,P-1));
       LI.Add(S);
       T := Copy(T,P+L,Length(T));
    end;
  until P = 0;
end;
```

[5] Durch Kommas separierte Zahlenwerte aus den Zeilen einer *RichEdit*-Komponente mit Hilfe einer Stringliste *MyList* in eine zweidimensionale Matrix *Mat* kopieren.

Dazu muss eine Stringliste *MyList* am Programmanfang mittels *MyList := TStringList.Create* im Speicher erzeugt werden. Der Befehl *EditToList(RichEdit1,MyList)* kopiert dann die markierten Zeilen von *RichEdit* in die Liste. Zuletzt kann die Prozedur ***ListToMatrix****(MyList,Mat)* ausgeführt werden. Die Anzahl aller transferierter Daten *Sum* steht in dem Matrixfeld *Mat[0,0]*, die Anzahl aller Zeilen *N* steht in *Mat[0,1]*, die Anzahlen der Daten pro Zeile *Anz* stehen in den ersten Feldern *Mat[Y,0]* der jeweiligen Zeilen *Y*. Die komma-separierten Daten der Zeile *Y* befinden sich in den Feldern *Mat[Y,1]*, *Mat[Y,2]*, *Mat[Y,3]* ... *Mat[Y,Anz]*, wobei der Zeilenindex *Y* von *1* bis *N* geht.

```
type Matrix = array[0..1000,0..1000] of Real;
var  MyList : TStringList;
     Mat : Matrix;
begin
  MyList := TStringList.Create;
  EditToList(RichEdit1,MyList);
end;

procedure ListToMatrix(LI: TStringList; var MA: Matrix);
{ CSV-Datentransfer von Listenzeilen in ein Zahlenarray }
const SEP = ',';
var   Zeile: Array[0..1000] of Real;
      N,Sum,Anz,X,Y: Integer;
begin
  N := LI.Count;
  Sum := 0;
  for Y := 0 to N-1 do begin
      ExtractValues(LI[Y],SEP,Zeile);   // Siehe Buchabschnitt [05]
      Anz := Round(Zeile[0]);           // Anzahl der Daten pro Zeile
      MA[Y+1,0] := Anz;
      For X := 1 to Anz do MA[Y+1,X] := Zeile[X];
      Sum := Sum + Anz;
  end;
  MA[0,0] := Sum;                       // Anzahl aller Daten
  MA[0,1] := N;                         // Anzahl der Zeilen
end;
```

[6] Einen Text aus einer Datei einlesen, entweder im TXT-Format oder im RTF-Format.

```
procedure TextRead(RE: TRichEdit);
// Textdatei nach RichEdit laden unter Verwendung einer OpenDialog-Box
// Verz, Ext und FName sind bereits definierte Stringvariable
begin
  GetDir(0,Verz);           // Aktuelles Verzeichnis auf Verz speichern
  with Form1.OpenDialog1 do begin
       InitialDir := Verz;
       Filter := 'ANSI-Textdateien (*.txt)|*.txt|' +
                 'RTF-Textdateien (*.rtf)|*.rtf|' +
                 'Alle Dateien (*.*)|*.*';
       DefaultExt := 'txt';
       Options    := [ofFileMustExist];
       FileName   := '';
```

```
      if Execute then begin
         Verz  := Trim(ExtractFilePath(FileName));
         Ext   := Trim(LowerCase(ExtractFileExt(FileName)));
         FName := Trim(ExtractFileName(FileName));
         if Ext = '.rtf' then RE.PlainText := False
                         else RE.PlainText := True;
         try
           RE.Lines.LoadFromFile(FileName);
         except
           MessageBox(0,'Daten-Fehler','Problem',16);
         end;
      end;
  end;
end;
```

[7] Einen Text in eine Datei schreiben, entweder im TXT-Format oder im RTF-Format.

```
procedure TextWrite(RE: TRichEdit);
// Text in eine Datei speichern unter Verwendung einer SaveDialog-Box
// Verz, Ext und FName sind bereits definierte Stringvariable
begin
  GetDir(0,Verz);           // Aktuelles Verzeichnis auf Verz speichern
  with Form1.SaveDialog1 do begin
       InitialDir := Verz;
       Filter := 'ANSI-Textdateien (*.txt)|*.txt|' +
                 'RTF-Textdateien (*.rtf)|*.rtf|' +
                 'Alle Dateien (*.*)|*.*';
       DefaultExt := 'txt';
       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 RE.PlainText := False
                          else RE.PlainText := True;
          try
             RE.Lines.SaveToFile(FileName);
          except
             MessageBox(0,'Daten-Fehler','Problem',16);
          end;
       end;
  end;
end;
```

[8] Schrift und Schriftattribute in *RichEdit* ändern.

```
var REF : TFont;                          // Universeller Schrift-Typ
    REA : TTextAttributes;                // Schriftattribut-Typ für RichEdit

begin
  REF := Form1.RichEdit1.Font;            // Definierte Schrift speichern
  REA := Form1.RichEdit1.DefAttributes;   // Definierte Attribute speichern
end;

procedure TForm1.Schriftart1Click(Sender: TObject);
// Schriftart ändern unter Verwendung einer FontDialog-Box
// Menüeintrag "Schrift - Schriftart"
begin
  FontDialog1.Font.Assign(RichEdit1.SelAttributes);
  if FontDialog1.Execute then
     RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  RichEdit1.SetFocus;
end;
```

Am Programmanfang werden die eingestellte Schrift und ihre Schriftattribute von *RichEdit* auf zwei entsprechende Variable *REF* und *REA* gespeichert. Im Event-Handler des Menü-Eintrages "*Schrift – Schriftart*" wird zunächst die aktuelle Schriftart von *RichEdit* der *FontDialog*-Box übermittelt. Daraufhin kann dort eine neue Schriftart eingestellt und diese der aktuellen Schriftart von *RichEdit* zugewiesen werden. Die Schriftzuweisungen erfolgen mit der Methode *Assign*.

Die hier verwendeten Objekttypen ***TFont*** und ***TTextAttribut*** sind in ihren Eigenschaften (Typ, Größe, Farbe und Stil) und Methoden nahezu identisch. Mit der Methode *Assign* können sie einander zugewiesen werden. *TFont* bezeichnet ein universelles Schriftobjekt, *TTextAttributes* hingegen bezieht sich nur auf den Text von *RichEdit*. Die Eigenschaft ***DefAttributes*** von *RichEdit* bestimmt die Schriftart für den ganzen Text, die Eigenschaft ***SelAttributes*** hingegen bestimmt die Schriftart des markierten bzw. aktuellen Textes.

[9] Mit den Tasten <Strg> und <Enter> das Zeichen ¶ in den Text einfügen. (Beim Ausdrucken wird dann genau dort ein Seitenvorschub durchgeführt)

```
procedure TForm1.RichEdit1.KeyUp(Sender: TObject; var Key: Word;
                                 Shift: TShiftState;
// Mit den Tasten <Strg> und <Enter> das Zeichen ¶ in den Text einfügen.
// REF ist eine Font-Variable, in der die Standardschrift gespeichert wurde.
const PiChar = #182;                // Code für das Zeichen "PI"
      EOL = #13#10;                 // Code für Zeilenende
begin
  if (key = 13) and (Shift = [ssCtrl]) then begin
     FontDialog1.Font.Assign(RichEdit1.SelAttributes);
     RichEdit1.SelAttributes.Assign(REF);
     RichEdit1.SelText := PiChar + EOL;
     RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  end;
end;
```

[10] Einen Text formatiert ausdrucken (WYSIWYG = "What You See Is What You Get").

Die RichEdit-Komponente von Delphi enthält eine Vielzahl von mächtigen Eigenschaften und Methoden zur Textverarbeitung. Jedoch fehlen auch einige wünschenswerte Funktionen.

Erstens kann für den Ausdruck des Textes (***print***) kein manueller Seitenvorschub, keine Kopfzeile und auch kein linker Rand gesetzt werden. Im Programm **TEXTPRO** ist dieses Problem mit Hilfe der Routine ***procedure RTFPrint(RE1,RE2: TRichEdit)*** gelöst.

Zweitens gibt es keine Möglichkeit einen markierten Textteil hochzustellen oder tiefzustellen.

[11] Hochstellen und Tiefstellen von Textteilen

(a) Zuerst werden mit Hilfe der Textverarbeitung WORD zwei Textdateien im RTF-Format angelegt. Die erste Datei ***hoch.rtf*** enthält nur die drei Zeichen .^{A}. in einer Zeile, wobei das A hochgestellt ist. Die zweite Datei ***tief.rtf*** enthält nur die drei Zeichen .$_{A}$. in einer Zeile, wobei das A tiefgestellt ist.

(b) Aus diesen beiden Dateien wird mit dem Borland Ressourcen Compiler ***brcc32.exe*** eine so genannte Ressourcen-Datei ***hochtief.res*** erzeugt. Dazu muss zunächst mit einem Editor die Textdatei ***hochtief.rc*** angelegt werden, welche folgende zwei Zeilen enthält:

```
101 rcdata loadoncall discardable "hoch.rtf"
102 rcdata loadoncall discardable "tief.rtf"
```

Dann wird mit einem Editor eine Batch-Datei ***hochtief.bat*** angelegt, welche folgende zwei Zeilen enthält und bei ihrer Ausführung die gewünschte Ressourcen-Datei ***hochtief.res*** erzeugt.

```
brcc32.exe  hochtief.rc
pause
```

(c) Im Programm **TEXTPRO** wird die Ressourcen-Datei folgendermaßen eingebunden:

```
implementation
{$R  *.dfm}
{$R  hochtief.res}
```

Nach diesen Schritten kann zuletzt durch folgende Menü-Routine das Hochstellen von markierten Texten verwirklicht werden:

```
procedure TForm1.Hochstellen1Click(Sender: TObject);
// Hochschrift
var L,G,N: Integer;
    TA: TTextAttributes;
    TS: TFontStyles;
    C : TColor;
    S : String;
begin
  L  := RichEdit1.SelLength;
  if (L = 0) or (Trim(RichEdit1.SelText) = '') then Exit;
  TA := RichEdit1.SelAttributes;
  TA0 := TA;
  TS := TA.Style;
  S  := TA.Name;
  N  := TA.Size;
  C  := TA.Color;
  RichEdit2.Clear;
  ResStream := TResourceStream.CreateFromID(HInstance,101,RT_RCDATA);
  RichEdit2.Lines.LoadFromStream(ResStream);
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := 1;
  RichEdit2.SelText := RichEdit1.SelText;
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := L;
  RichEdit2.SelAttributes.Name  := S;
  RichEdit2.SelAttributes.Style := TS;
  RichEdit2.SelAttributes.Size  := N;
  RichEdit2.SelAttributes.Color := C;
  RichEdit2.CopyToClipboard;
  RichEdit1.PasteFromClipboard;
  RichEdit1.SelStart := RichEdit1.SelStart - L;
  RichEdit1.SelLength := L;
end;
```

In der Routine kommen zwei globale Variable ***TA0*** und ***ResStream*** vor.

```
var   ResStream: TResourceStream;  // Ressourcen-Stream
      TA0: TTextAttributes;        // Text-Attribut-Variable
```

Die Variable ***TA0*** dient zur Speicherung der aktuellen Schriftattribute vor dem Hochstellen des Textes und wird dann beim Normalstellen des Textes verwendet. Die Variable ***ResStream*** dient dem Zugriff auf die zwei Dateien (Instanzen) in der Ressourcen-Datei, sodass diese dann in eine RichEdit-Komponente geladen werden können.

unit textpro_u;

```
// Textverarbeitung (c) Herbert Paukert

interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Clipbrd,
     Controls, Forms, Dialogs, StdCtrls, Math, Printers, ExtCtrls, ComCtrls,
     Menus, mparse_u;
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    RichEdit1: TRichEdit;
    RichEdit2: TRichEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Bevel1: TBevel;

    FontDialog1: TFontDialog;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FindDialog1: TFindDialog;
    ReplaceDialog1: TReplaceDialog;

    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;
      Komprimieren1: TMenuItem;
      GehezurZeile1: TMenuItem;
    Schrift1: TMenuItem;
      Schriftart1: TMenuItem;
      Schriftgre1: TMenuItem;
      N2: TMenuItem;
      Groschrift1: TMenuItem;
      Kleinschrift1: TMenuItem;
      N3: TMenuItem;
      Hochstellen1: TMenuItem;
      iefstellen1: TMenuItem;
      Normalstellen1: TMenuItem;
      N4: TMenuItem;
      Umbruchea1: TMenuItem;
    Hilfe1: TMenuItem;
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Memo1DblClick(Sender: TObject);
    procedure RichEdit1MouseDown(Sender: TObject; Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
    procedure RichEdit1KeyUp(Sender: TObject; var Key: Word;
                             Shift: TShiftState);
    procedure FindDialog1Find(Sender: TObject);
    procedure ReplaceDialog1Find(Sender: TObject);
    procedure ReplaceDialog1Replace(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 Komprimieren1Click(Sender: TObject);
    procedure GehezurZeile1Click(Sender: TObject);
    procedure Schriftart1Click(Sender: TObject);
    procedure Schriftgre1Click(Sender: TObject);
    procedure Groschrift1Click(Sender: TObject);
    procedure Kleinschrift1Click(Sender: TObject);
    procedure Hochstellen1Click(Sender: TObject);
    procedure iefstellen1Click(Sender: TObject);
    procedure Normalstellen1Click(Sender: TObject);
    procedure Umbruchea1Click(Sender: TObject);
    procedure Hilfe1Click(Sender: TObject);
  private { Private declarations }
  public  { Public declarations }
end;

var  Form1: TForm1;

implementation
{$R *.dfm}
{$R hochtief.RES}

type  ZFeld  = array[0..1000] of Real;          // Zahlenfeld
      Matrix = array[0..1000,0..1000] of Real;  // Zahlenmatrix

const Rand     : Integer = 10;                  // Textrand
      PiChar   : String = #182;                 // PI-Symbol
      FormFeed : String = #182#13#10;           // PI-Symbol + LineFeed
      FontNorm : Integer = 14;                  // Standard-Schriftgröße

var   ResStream: TResourceStream;               // Ressourcen-Stream
      TA0: TTextAttributes;                     // Text-Attribut-Variable
      REA: TTextAttributes;
      REF: TFont;                               // Text-Schrift-Variable
```

```
      L4W: Integer;                          // originale Label4.Width
      Verz,FName,Ext: String;                // Ordner,Datei,Extension

      MyList : TStringList;                  // Stringliste
      ZF     : ZFeld;                        // Zahlenfelder
      Mat    : Matrix;                       // Zahlenmatrix
      Error  : Boolean;                      // Fehlervariable

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;

procedure SetMargin(RE: TRichEdit; L,R,T,B : Integer);
// Linken, Rechten, Oberen und Unteren Rand setzen
var Rect :TRect;
begin
  SendMessage(RE.Handle,EM_GETRECT,0,LongInt(@Rect));
  Rect.Left   := Rect.Left + L;
  Rect.Right  := Rect.Right - R;
  Rect.Top    := Rect.Top + T;
  Rect.Bottom := Rect.Bottom - B;
  SendMessage(RE.Handle,EM_SETRECT,0,LongInt(@Rect));
end;

procedure ShowColRow(RE: TRichEdit; L: TLabel);
// Anzeigen der aktuellen Spalte und Zeile
begin
   L.Caption := 'Spalte/Zeile: ' + IntToStr(RE.CaretPos.X+1) + ' / '
                                 + IntToStr(RE.CaretPos.Y+1);
end;

procedure GotoXY(RE: TRichEdit; X,Y: Integer);
// Positioniert den Cursor an die Spalte X und Zeile Y
var I,J,N: Integer;
begin
  if Y < 1 then Y := 1;
  if Y > RE.Lines.Count then Y := RE.Lines.Count;
  N := 0;
```

```
  For I := 0 to Y-2 do begin
      if RE.Lines[I] = '' then J := 2
      else J := Length(RE.Lines[I]) + 2;
      N := N + J;
  end;
  N := N + X - 1;
  RE.SelStart  := N;
  RE.SelLength := 0;
  SendMessage(RE.Handle,EM_SCROLLCARET,0,0);
end;

function SearchFirst(S,A: String; Start,Flag: Integer): Integer;
// sucht im Text S den Suchstring A und liefert die
// Position der ersten Fundstelle ab Startposition.
// Case-Insensitiv: Flag = 0, Case-Sensitiv: Flag = 1
var P : Integer;
begin
  S := Copy(S,Start+1,Length(S));
  if Flag = 0 then begin
     S := UpperCase(S);
     A := UpperCase(A);
  end;
  P := Pos(A,S);
  if P = 0 then Result := 0
           else Result := Start + p;
end;

function ReplaceAllText(RE: TRichEdit; A,B: String; Flag: Integer): Integer;
// String A in Text S suchen und durch Text B ersetzen
// und die Anzahl N der Ersetzungen liefern
var S   : String;
    P,N : Integer;
begin
  N := 0;
  P := 0;
  repeat
    S := RE.Text;
    P := SearchFirst(S,A,P,Flag);
    if P > 0 then begin
       N := N + 1;
       RE.SelStart  := P - 1;
       RE.SelLength := Length(A);
       RE.SelText   := B;
       P := P + Length(B);
    end;
  until (P = 0);
  Result := N;
end;


procedure PrintMemo(M: TMemo);
// Memotext ausdrucken
const  LR = '    ';
       FF = '.';
var    S : String;
       N : Integer;
       Datei: TextFile;
```

```
begin
  AssignPrn(Datei);
  {$I-} Rewrite(Datei); {$I+}
  if IOResult <> 0 then begin
     Messagebox(0,' KEIN Druckerzugriff ! ','Problem',16);
     Exit;
  end;
  Printer.Canvas.Font := M.Font;
  Printer.Canvas.Font.Size := 11;
  Writeln(Datei,LR);
  Writeln(Datei,LR);
  For N := 0 to M.Lines.Count-1 do begin
      S := M.Lines[N];
      if Pos(FF,Trim(S)) = 1 then begin
         Writeln(Datei,#12);
         Writeln(Datei,LR);
      end
      else Writeln(Datei,LR + S);
  end;
  CloseFile(Datei);
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 RemoveBlank(S: String): String;
// Entfernt alle Blanks aus einem String
begin
  While Pos(#32,S) > 0 do Delete(S,Pos(#32,S),1);
  Result := S;
end;

function RemoveMultipleChar(S: String; CH: Char): String;
// Entfernt mehrfache Zeichen CH aus einem String S
const Dummy = #1;
var   I : Integer;
begin
  S := S + Dummy;
  I := 0;
  repeat
    I := I + 1;
    if (S[I] = CH) and (S[I+1] = S[I]) then begin
        Delete(S,I,1);
        I := I - 1;
    end;
  until S[I] = Dummy;
  Result := Copy(S,1,Length(S)-1);
end;
```

```
procedure KompressLines(RE: TRichEdit; B: Boolean);
//  Entfernt alle mehrfachen Leerzeilen aus dem Text,
//  bei B = TRUE werden auch alle mehrfachen Blanks entfernt
var N,I : Integer;
    S: String;
begin
  with RE do begin
     N := Lines.Count-1;
     I := -1;
     repeat
       I := I + 1;
       if (Trim(Lines[I]) = '') and (Trim(Lines[I+1]) = '') then begin
          Lines.Delete(I);
          I := I-1;
          N := N-1;
       end;
     until I > N;
     if B then begin
        N := Lines.Count-1;
        For I := 0 to N do begin
            S := Trim(Lines[I]);
            if (S <> '') then S := RemoveMultipleChar(S,#32);
            Lines[I] := S;
        end;
     end;
  end;
end;

procedure EOLAppend(var RE: TRichEdit);
// An jede Zeile ein <ENTER> anfügen
const EOL : String = #13#10;
var   I,K,N,ANZ  : Integer;
begin
  with RE do begin
     SelStart := Length(Text);
     SelText  := EOL;
     N := Length(Text);
     K := 0; I := 0;
     repeat
       SelStart := I;
       if (CaretPos.X = 0) and (CaretPos.Y <> 0) then begin
          K := K + 1;
       end;
      I := I + 1;
     until I >= N;
     ANZ := K;
     K := 0; I := 0;
     repeat
       SelStart := I;
       if (CaretPos.X = 0) and (CaretPos.Y <> 0) then begin
           K := K + 1;
           SelText := EOL;
           I := I + 2;
       end;
       I := I + 1;
     until K >= ANZ;
  end;
end;
```

```
procedure TextFlow(RE: TRichEdit);
// Erzeugt einen Fließtext
const EOL: String = #13#10;
var S,T: String;
    N  : Integer;
begin
  with RE do begin
     S := EOL;
     T := '~';
     RE.SelStart := 0;
     N := ReplaceAllText(RE,S,T,0);
     S := EOL;
     T := '';
     RE.SelStart := 0;
     N := ReplaceAllText(RE,S,T,0);
     S := '~~';
     T := EOL+EOL;
     RE.SelStart := 0;
     N := ReplaceAllText(RE,S,T,0);
     S := '~';
     T := ' ';
     RE.SelStart := 0;
     N := ReplaceAllText(RE,S,T,0);
   end;
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 EditToList(RE:TRichEdit; var LI: TStringList);
// Transfer markierter Zeilen aus RichEdit in eine Stringliste
const SEP = #13#10;
var T,S : String;
    P,L : Integer;
begin
  L := Length(SEP);
  if RE.SelLength = 0 then Exit;
  MyList.Clear;
  T := RE.SelText;
  if Copy(T,Length(T)+1-L,L) <> SEP then T := T + SEP;
  repeat
    P := Pos(SEP,T);
    if P > 0 then begin
       S := Trim(Copy(T,1,P-1));
       LI.Add(S);
       T := Copy(T,P+L,Length(T));
    end;
  until P = 0;
end;

procedure ListToMatrix(LI: TStringList; var MA: Matrix);
// CSV-Datentransfer von Listenzeilen in ein Zahlenarray
const SEP = ',';
var   Zeile: ZFeld;
      N,Sum,Anz,X,Y: Integer;
begin
  N := LI.Count;
  Sum := 0;
  For Y := 0 to N-1 do begin
      ExtractValues(LI[Y],SEP,Zeile);
      Anz := Round(Zeile[0]);        // Anzahl der Daten pro Zeile
      MA[Y+1,0] := Anz;
      For X := 1 to Anz do MA[Y+1,X] := Zeile[X];
      Sum := Sum + Anz;
  end;
  MA[0,0] := Sum;                    // Anzahl aller Daten
  MA[0,1] := N;                      // Anzahl der Zeilen
end;

procedure Statis;
// Eindimensionale statistische Datenauswertung
var  NN,A,N,Anz,X,Y: Integer;
     Sum,QSum,Min,Max,Mwt,Stg: Real;
     S: String;
begin
with Form1.RichEdit1 do begin
  Error := False;
  EditToList(Form1.RichEdit1,MyList);
  ListToMatrix(MyList,Mat);
  SelStart := SelStart + SelLength;
  NN := CaretPos.Y;
  A := Round(Mat[0,0]);
  N := Round(Mat[0,1]);
  Anz := 0;
  For Y := 1 to N do Anz := Anz + Round(Mat[Y,0]);
```

```
  if Anz <> A then begin
     ShowMessage('Auswertungs-Fehler !');
     Error := True;
     Exit;
  end;
  Sum := 0; QSum := 0; Mwt := 0; Stg := 0;
  Min := Mat[1,1]; Max := Min;
  For Y := 1 to N do begin
      Anz := Round(Mat[Y,0]);
       For X := 1 to Anz do begin
           Sum  := Sum + Mat[Y,X];
           QSum := QSum + Mat[Y,X]*Mat[Y,X];
           if Mat[Y,X] < Min then Min := Mat[Y,X];
           if Mat[Y,X] > Max then Max := Mat[Y,X];
       end;
  end;
  Lines.Add(' ');
  Lines.Insert(NN+1,'-----------------------------');
  Lines.Insert(NN+2,'Anzahl = ' + IntToStr(A));
  Str(Min:10:2,S);
  Lines.Insert(NN+3,'Minimum = ' + Trim(S));
  Str(Max:10:2,S);
  Lines.Insert(NN+4,'Maximum = ' + Trim(S));
  Str((Max-Min):10:2,S);
  Lines.Insert(NN+5,'Schwankung = ' + Trim(S));
  Str(Sum:10:2,S);
  Lines.Insert(NN+6,'Summe = ' + Trim(S));
  Mwt := Sum / A; Str(Mwt:10:2,S);
  Lines.Insert(NN+7,'Mittelwert = ' + Trim(S));
  Stg := Sqrt(QSum / A - Mwt * Mwt); Str(Stg:10:2,S);
  Lines.Insert(NN+8,'Streuung = ' + Trim(S));
  Lines.Insert(NN+9,'-----------------------------');
end;
end;

procedure InsertTab(RE: TRichEdit; N: Integer);
// Fügt am Anfang der Zeilen von RE genau N Tabulaturen TABS ein
const TAB : String = #9;
      LF  : String = #13#10;
var   L,P : Integer;
      LR  : String;
begin
  L := Length(RE.Text);
  LR := TAB;
  if N = 2 then LR := TAB + TAB;
  P := 0;
  repeat
    P := RE.FindText(LF,P,L,[]);
    if P <> -1 then begin
       RE.SelStart := P + 2;
       RE.SelText := LR;
       P := P + 3;
       if N = 2 then P := P + 4;
    end;
  until (P = -1);
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;

procedure RTFPrint(var RE1,RE2 : TRichEdit);
// Aktuellen Text im RTF-Format ausdrucken
// REF = Standardfont, REA = Standardattribute von RE1
// NumFlag = Steuerung für Seitennummern
const SL = #32;
      Blank        : String = #32;
      TAB          : String = #9;
      LineFeed     : String = #13#10;
      PiChar       : String = #182;
      FormFeed     : String = #182#13#10;
var   SEP,K,T,LR   : String;
      P,P0,I,N,RES : Integer;
      LEN,ANZ,NUM  : Integer;
      AllFlag      : Boolean;
      TopMargin    : Integer;
      LeftMargin   : Integer;
      NumFlag      : Boolean;
      PANZ         : Integer;
begin
  NumFlag := True;
  LeftMargin := 1;
  TopMargin  := 1;
  PANZ := SearchAllText(RE1,SEP);
  Clipboard.Clear;
  RE2.ReadOnly := False;
  RE2.Clear;
  RE2.Font := REF;
  RE2.DefAttributes := REA;
  SEP := FormFeed;
  LEN := Length(SEP);
  K   := '<' + FName + ': Seite ';
  NUM := 1;
  with RE1 do begin
    if (SelLength = 0) then begin
       P  := 0;
       P0 := 0;
       SelectAll;
       AllFlag := True;
    end
```

```
else begin
   P  := SelStart;
   P0 := SelStart;
   AllFlag := False;
end;
ANZ  := SelLength;
NUM := 1;
if Not AllFlag then begin
     CopyToClipboard;
     RE2.Clear;
     RE2.Font := REF ;
     RE2.DefAttributes := REA;
     RE2.Lines.Add(Blank);
     T := K + IntToStr(NUM) + '>';
     if NumFlag then RE2.Lines.Add(T);
     For i := 1 to TopMargin do RE2.Lines.Add(Blank);
     RE2.PasteFromClipboard;
     InsertTab(RE2,LeftMargin);
     RE2.Print('');
     Clipboard.Clear;
     RE2.ReadOnly := True;
     Exit;
end;
PANZ := SearchAllText(RE1,SEP);
repeat
  RES := ANZ - P;
  P := FindText(SEP,P,ANZ,[]);
  if (P <> -1) or (NUM = 1) then begin
     N := P - 1 - P0;
     SelLength := N;
     CopyToClipboard;
     RE2.Clear;
     RE2.Font := REF ;
     RE2.DefAttributes := REA;
     RE2.Lines.Add(Blank);
     T := K + IntToStr(NUM) + '>';
     if NumFlag then RE2.Lines.Add(T);
     For i := 1 to TopMargin do RE2.Lines.Add(Blank);
     RE2.PasteFromClipboard;
     P  := P + LEN;
     P0 := P;
     SelStart := P;
     NUM := NUM + 1;
     InsertTab(RE2,LeftMargin);
     RE2.Print('');
  end;
until (P = -1);

if PANZ >  0 then begin
   SelStart  := P0;
   SelLength := RES;
   CopyToClipboard;
   RE2.Clear;
   RE2.Font := REF ;
   RE2.DefAttributes := REA;
   RE2.Lines.Add(Blank);
   T := K + IntToStr(NUM) + '>';
   if NumFlag then RE2.Lines.Add(T);
```

```
      For i := 1 to TopMargin do RE2.Lines.Add(Blank);
      RE2.PasteFromClipboard;
      InsertTab(RE2,LeftMargin);
      RE2.Print('');
    end;
    SelStart := 0;
  end;
  Clipboard.Clear;
  RE2.ReadOnly := True;
end;

procedure TForm1.FormCreate(Sender: TObject);
// Initialisierungen
begin
  MyList := TStringList.Create;
  GetDir(0,Verz);
  Form1.Color := RGB(150,160,180);
  FitForm(Form1);
  if Screen.Height = 768 then
     if (Font.PixelsPerInch = 96) then RichEdit1.Font.Size := FontNorm;
  if (Screen.Height > 768) and (Screen.Height <= 1024) then
     if (Font.PixelsPerInch = 96) then RichEdit1.Font.Size := FontNorm + 2;
  if Screen.Height > 1024 then
     if (Font.PixelsPerInch = 96) then RichEdit1.Font.Size := FontNorm + 4;
  if (Font.PixelsPerInch > 96) then RichEdit1.Font.Size := RichEdit1.Font.Size-2;
  SetMargin(RichEdit1,Rand,Rand,Rand,Rand);
  REF := RichEdit1.Font;
  FontDialog1.Font := REF;
  REA := RichEdit1.DefAttributes;
  TA0 := REA;
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.Memo1DblClick(Sender: TObject);
// Hilfstext ausdrucken
begin
  PrintMemo(Memo1);
end;

procedure TForm1.RichEdit1MouseDown(Sender: TObject; Button: TMouseButton;
                                    Shift: TShiftState; X, Y: Integer);
// Aktuelle Spalte und Zeile anzeigen
begin
  ShowColRow(RichEdit1,Label1);
end;

procedure TForm1.RichEdit1KeyUp(Sender: TObject; var Key: Word;
                                Shift: TShiftState);
var S,T: String;
    P,N: Integer;
    Z  : Real;
begin
  ShowColRow(RichEdit1,Label1);       // Aktuelle Spalte und Zeile anzeigen
                                      // Markierung für Seitenvorschub einfügen
  if (key = 13) and (Shift = [ssCtrl]) then begin
     FontDialog1.Font.Assign(RichEdit1.SelAttributes);
     RichEdit1.SelAttributes.Assign(REF);
     RichEdit1.SelText := FormFeed;
     RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  end;
  if (key = vk_F1) then begin        // Aktuelle Textzeile mathematisch auswerten
      N := RichEdit1.CaretPos.Y;
      T := RichEdit1.Lines[N];
      P := Pos('=',T);
      S := RemoveBlank(Copy(T,1,P-1));
      if (P = 0) or (S='') then begin
          ShowMessage('Auswertungs-Fehler !');
          Exit;
      end;
      Z := Parse(S,ATT);
      if (ATT<1) or (ATT>3) then begin
          ShowMessage('Auswertungs-Fehler !');
          Exit;
      end;
      Str(Z:10:2,T);
      T := S + ' = ' + Trim(T);
      RichEdit1.Lines[N] := T;
  end;
  if (Key = vk_F2) then begin        // ANSI-Codes aller Textzeichen auflisten
     For N := 1 to 255 do begin
       T := Chr(N);
       if (N = 9) or (N = 10) or (N = 13) then T := #32;
       S := T + #9 + IntToStr(N);
       Richedit1.Lines.Add(S);
     end;
  end;
end;
```

```
procedure TForm1.FindDialog1Find(Sender: TObject);
// Text schrittweise in RichEdit suchen
var FoundAt, StartPos, ToEnd: Integer;
    SType: TSearchTypes;
begin
  with RichEdit1 do begin
    SType := [];
    if (frMatchCase) in FindDialog1.Options then SType := SType + [stMatchCase];
    if (frWholeWord) in FindDialog1.Options then SType := SType + [stWholeWord];
    if SelLength <> 0 then StartPos := SelStart + SelLength
                      else StartPos := SelStart;
    ToEnd := Length(Text) - StartPos;
    FoundAt := FindText(FindDialog1.FindText, StartPos, ToEnd, SType);
    if FoundAt <> -1 then begin
       SetFocus;
       SelStart  := FoundAt;
       SelLength := Length(FindDialog1.FindText);
       SendMessage(Handle,EM_SCROLLCARET,0,0);
    end;
    SetFocus;
  end;
end;

procedure TForm1.ReplaceDialog1Find(Sender: TObject);
// Text schrittweise in RichEdit suchen
var FoundAt, StartPos, ToEnd: Integer;
    SType: TSearchTypes;
begin
with RichEdit1 do begin
  SType := [];
  if (frMatchCase) in ReplaceDialog1.Options then SType := SType + [stMatchCase];
  if (frWholeWord) in ReplaceDialog1.Options then SType := SType + [stWholeWord];
  if SelLength <> 0 then StartPos := SelStart + SelLength
                    else StartPos := SelStart;
  ToEnd := Length(Text) - StartPos;
  FoundAt := FindText(ReplaceDialog1.FindText, StartPos, ToEnd, SType);
  if FoundAt <> -1 then begin
       SetFocus;
       SelStart  := FoundAt;
       SelLength := Length(ReplaceDialog1.FindText);
       SendMessage(Handle,EM_SCROLLCARET,0,0);
  end;
end;
end;

procedure DoReplace;
// Text schrittweise in RichEdit ersetzen
var FoundAt, StartPos, ToEnd: Integer;
    SType: TSearchTypes;
begin
with Form1 do begin
with RichEdit1 do begin
  SType := [];
  if (frMatchCase) in ReplaceDialog1.Options then SType := SType + [stMatchCase];
  if (frWholeWord) in ReplaceDialog1.Options then SType := SType + [stWholeWord];
  if SelLength <> 0 then StartPos := SelStart + SelLength
                    else StartPos := SelStart;
  ToEnd := Length(Text) - StartPos;
```

```
  FoundAt := FindText(ReplaceDialog1.FindText, StartPos, ToEnd, SType);
  if FoundAt <> -1 then begin
       SetFocus;
       SelStart  := FoundAt;
       SelLength := Length(ReplaceDialog1.FindText);
       SelText   := ReplaceDialog1.ReplaceText;
  end;
end;
end;
end;

procedure DoReplaceAll;
// Text global in RichEdit ersetzen
var FoundAt, StartPos, ToEnd: Integer;
    SType: TSearchTypes;
begin
with Form1 do begin
with RichEdit1 do begin
  SType := [];
  if (frMatchCase) in ReplaceDialog1.Options then SType := SType + [stMatchCase];
  if (frWholeWord) in ReplaceDialog1.Options then SType := SType + [stWholeWord];
  StartPos := 0;
  repeat
       ToEnd := Length(Text) - StartPos;
       FoundAt := FindText(ReplaceDialog1.FindText, StartPos, ToEnd, SType);
       if FoundAt <> -1 then begin
          SelStart  := FoundAt;
          SelLength := Length(ReplaceDialog1.FindText);
          SelText   := ReplaceDialog1.ReplaceText;
          StartPos  := FoundAt + Length(ReplaceDialog1.ReplaceText);
       end;
  until FoundAt = -1;
end;
end;
end;

procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
// Texte ersetzen
begin
  if (frReplace) in ReplaceDialog1.Options then DoReplace
  else if (frReplaceAll) in ReplaceDialog1.Options then DoReplaceAll;
  RichEdit1.SetFocus;
end;

procedure TForm1.Neu1Click(Sender: TObject);
// Neuer Text
begin
  RichEdit1.Clear;
  RichEdit1.SetFocus;
  ShowColRow(RichEdit1,Label1);
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 begin
             RichEdit1.PlainText := False;
             if MessageBox(0,'OHNE Anzeige der internen RTF-Steuercodes ? ',
                           'Eingabe',36) = 6 then RichEdit1.PlainText := False
                                             else RichEdit1.PlainText := True;
          end
          else
             RichEdit1.PlainText := True;
          Try
            RichEdit1.Clear;
            RichEdit1.Lines.LoadFromFile(FileName);
          Except
            MessageBox(0,'Daten-Fehler','Problem',16);
          end;
       end;
  end;
  Label2.Caption := 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 := FName;
end;
end;

procedure TForm1.Drucken1Click(Sender: TObject);
// Text ausdrucken (WYSIWYG)
begin
  RTFPrint(RichEdit1,RichEdit2);
  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;
  RichEdit1.SetFocus;
end;

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

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

procedure TForm1.Suchen1Click(Sender: TObject);
// Text suchen
begin
  FindDialog1.Position := Point(RichEdit1.Width div 2,RichEdit1.Height div 2);
  FindDialog1.Execute;
end;
```

```
procedure TForm1.Ersetzen1Click(Sender: TObject);
// Text ersetzen
begin
  ReplaceDialog1.Position := Point(RichEdit1.Width div 2,RichEdit1.Height div 2);
  ReplaceDialog1.Execute;
end;

procedure TForm1.SortierenText1Click(Sender: TObject);
// Textzeilen sortieren (als Strings)
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);
// Textzeilen sortieren (als Zahlen)
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
 if RichEdit1.SelLength = 0 then begin
    ShowMessage('Kein Zahlenbereich markiert ! ');
    Exit;
 end;
 Statis;
 if Error then ShowMessage('Datenfehler !');
 RichEdit1.SetFocus;
end;
```

```
procedure TForm1.Komprimieren1Click(Sender: TObject);
// Textzeilen komprimieren
begin
  if MessageBox(0,'Alle Zeilen komprimieren ?','Frage',36) = 6 then begin
     if MessageBox(0,'Alle mehrfachen Blanks entfernen ?','Frage',36) = 6 then
        KompressLines(RichEdit1,True)
     else
        KompressLines(RichEdit1,False);
  end;
end;

procedure TForm1.GehezurZeile1Click(Sender: TObject);
// Gehe zu Zeile
var S: String;
    X,Y,Err : Integer;
begin
  S := InputBox('Gehe zur Zeilennummer','','1');
  Val(S,Y,Err);
  if (Err <> 0) or (Y < 1) then Y := 1;
  X := 1;
  GotoXY(RichEdit1,X,Y);
  ShowColRow(RichEdit1,Label1);
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.Hochstellen1Click(Sender: TObject);
// Hochschrift
var L,G,N: Integer;
    TA: TTextAttributes;
    TS: TFontStyles;
    C : TColor;
    S : String;
begin
  L  := RichEdit1.SelLength;
  TA := RichEdit1.SelAttributes;
  TA0 := TA;
  TS := TA.Style;
  S  := TA.Name;
  N  := TA.Size;
  C  := TA.Color;
  if (L = 0) or (Trim(RichEdit1.SelText) = '') then Exit;
  RichEdit2.Clear;
  ResStream := TResourceStream.CreateFromID(HInstance,101,RT_RCDATA);
  RichEdit2.Lines.LoadFromStream(ResStream);
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := 1;
  RichEdit2.SelText := RichEdit1.SelText;
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := L;
  RichEdit2.SelAttributes.Name  := S;
  RichEdit2.SelAttributes.Style := TS;
  RichEdit2.SelAttributes.Size  := N;
  RichEdit2.SelAttributes.Color := C;
  RichEdit2.CopyToClipboard;
  RichEdit1.PasteFromClipboard;
  RichEdit1.SelStart := RichEdit1.SelStart - L;
  RichEdit1.SelLength := L;
end;

procedure TForm1.iefstellen1Click(Sender: TObject);
// Tiefschrift
var L,G,N: Integer;
    TA: TTextAttributes;
    TS: TFontStyles;
    C : TColor;
    S : String;
begin
  L  := RichEdit1.SelLength;
  TA := RichEdit1.SelAttributes;
  TA0 := TA;
  TS := TA.Style;
  S  := TA.Name;
  N  := TA.Size;
  C  := TA.Color;
  if (L = 0) or (Trim(RichEdit1.SelText) = '') then Exit;
```

```
  RichEdit2.Clear;
  ResStream := TResourceStream.CreateFromID(HInstance,102,RT_RCDATA);
  RichEdit2.Lines.LoadFromStream(ResStream);
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := 1;
  RichEdit2.SelText := RichEdit1.SelText;
  RichEdit2.SelStart  := 1;
  RichEdit2.SelLength := L;
  RichEdit2.SelAttributes.Name  := S;
  RichEdit2.SelAttributes.Style := TS;
  RichEdit2.SelAttributes.Size  := N;
  RichEdit2.SelAttributes.Color := C;
  RichEdit2.CopyToClipboard;
  RichEdit1.PasteFromClipboard;
  RichEdit1.SelStart := RichEdit1.SelStart - L;
  RichEdit1.SelLength := L;
end;

procedure TForm1.Normalstellen1Click(Sender: TObject);
// Normalschrift
var L: Integer;
begin
  L := RichEdit1.SelLength;
  RichEdit2.Clear;
  RichEdit2.SelText := RichEdit1.SelText;
  RichEdit2.SelStart  := 0;
  RichEdit2.SelLength := L;
  RichEdit2.SelAttributes.Name  := TA0.Name;
  RichEdit2.SelAttributes.Style := TA0.Style;
  RichEdit2.SelAttributes.Size  := TA0.Size;
  RichEdit2.SelAttributes.Color := TA0.Color;
  RichEdit2.CopyToClipboard;
  RichEdit1.PasteFromClipboard;
  RichEdit1.SelStart := RichEdit1.SelStart - L;
  RichEdit1.SelLength := L;
end;

procedure TForm1.Umbruchea1Click(Sender: TObject);
// Zeilenumbruch EIN/AUS
var N: Integer;
begin
  if RichEdit1.WordWrap then begin
     Screen.Cursor := crHourglass;
     N := MessageBox(0,'Zeilenende mit Zeilensprung-Einfügung ?','Frage',36);
     if (N = 6) then begin
         RichEdit1.Visible := False;
         EOLAppend(RichEdit1);
         KompressLines(RichEdit1,True);
         RichEdit1.Visible := True;
     end;
     RichEdit1.WordWrap := False;
     Setmargin(Form1.RichEdit1,-Rand,-Rand,-Rand,-Rand);
     Setmargin(Form1.RichEdit1,Rand,Rand,Rand,Rand);
     Label3.Caption := 'Umbruch AUS';
     Screen.Cursor := crDefault;
     RichEdit1.SetFocus;
     Exit;
  end;
```

```
  if NOT RichEdit1.WordWrap then begin
     Screen.Cursor := crHourglass;
     N := MessageBox(0,'Zeilenende mit Zeilensprung-Löschung ?','Frage', 36);
     if (N = 6) then begin
        RichEdit1.Visible := False;
        TextFlow(RichEdit1);
        RichEdit1.Visible := True;
     end;
     RichEdit1.WordWrap := True;
     SetMargin(RichEdit1,-Rand,-Rand,-Rand,-Rand);
     SetMargin(RichEdit1,Rand,Rand,Rand,Rand);
     Label3.Caption  := 'Umbruch EIN';
     Screen.Cursor := crDefault;
     RichEdit1.SetFocus;
  end;
end;

procedure TForm1.Hilfe1Click(Sender: TObject);
// Hilfstext ein- oder ausblenden
begin
  Memo1.Visible := Not Memo1.Visible;
end;

end.
```