M A T H E

Zehn ausgewählte Mathematik-Programme

© Herbert Paukert

ZEHN MATHEMATIK-PROGRAMME

[01] Berechnung mathematischer Ausdrücke (*mathe*)

In der Mathematik wird eine Funktion meistens durch einen algebraischen Ausdruck (Term) definiert. Dieser kann als Zeichenkette (String) aufgefasst werden, welche Zahlen, Variablen, Operatoren, Klammern und mathematische Funktionssymbole enthält. Im nachfolgenden Beispiel wird als Potenzoperator das Symbol ^ verwendet.

Beispiel: *F(X) = X^2 + 10 * (X – 3)*

Will man die Funktion berechnen, so muss für ***X*** ein konstanter Zahlenwert eingesetzt werden und sodann entsprechend den Regeln der Algebra die Berechnung durchgeführt werden. Dabei sind die Vorrangregeln der Rechenoperationen (z.B. Punkt- vor Strichrechnungen) zu berücksichtigen.

Beispiel: *X = 4*
*F(4) = 4^2 + 10 * (4 – 3) = 26*

Unter einem **Formelparser** versteht man eine Programmroutine, welcher der Term einer mathematischen Funktion als Stringparameter übergeben wird. Außerdem muss dem Funktionsargument der gewünschte Zahlenwert zugewiesen werden. In der Routine selbst wird dann der entsprechende Funktionswert berechnet und zurückgeliefert. Zusätzlich müssen Auswertungsfehler im Funktionsterm erkannt und in einer Fehlervariablen festgehalten werden.

Die Programmierung eines solchen Formelparsers ist eine sehr umfangreiche und auch schwierige Aufgabe und soll hier nicht näher erklärt werden. Der Autor stellt den Lesern seines Buches im Anhang den kompletten Quellcode eines universellen Formelparsers in einer eigenständigen Unit zur Verfügung. Im Folgenden soll die Verwendung dieser Unit beschrieben und ein Demonstrationsprogramm "***mathe***" dazu vorgestellt werden. Das Listing der gesamten Unit findet der interessierte Leser als Anhang am Ende dieses Textes.

Die Unit, welche den Formelparser enthält, heißt "***mparse_u.pas***". In der Unit werden 26 globale reelle Variable ***A_, B_, ... Z_*** definiert. Ihnen entsprechen die Buchstaben ***A, B, ... Z*** (bzw. ***a ... z***). Diese können als Argumente im Funktionsterm verwendet werden. Die ganzzahlige Variable ***ATT*** speichert etwaig aufretende Fehler bei der Funktionsberechnung (Syntaxfehler, Division durch Null, Wurzeln aus negativen Zahlen usw.). Die Parserroutine selbst ist als Funktion programmiert:

function Parse(S: String; var ATT: Integer): Real;

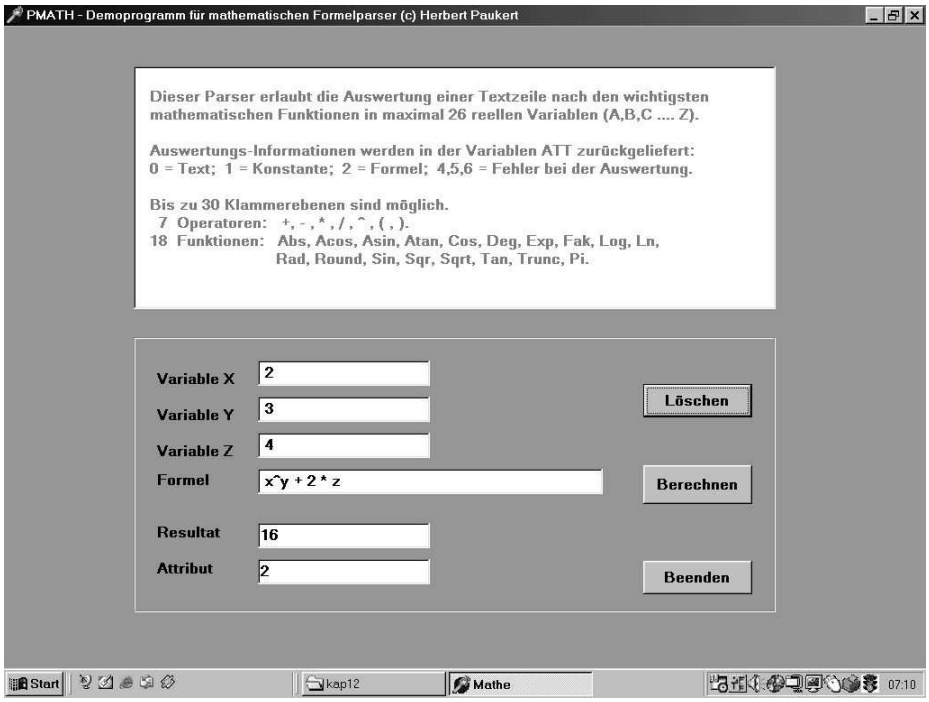
Der erste Parameter ***S*** ist der mathematische Funktionsterm, der zweite Parameter ***ATT*** ist die Fehlervariable und zurückgeliefert wird der Funktionswert. Voraussetzung ist, dass zuvor die im Term verwendeten Argumentvariablen mit Zahlenwerten belegt worden sind.

```
Beispiel:   X_  :=  4;                          // bzw. mparse_u.X_ := 4;
            S   :=  'X^2 + 10 * (X – 3)';
            W   :=  Parse(S,ATT);
```

Als Ergebnis erhalten die reelle Variable ***W*** den Wert ***26*** und die Fehlervariable ***ATT*** den Wert ***2***.

Im Funktionsterm können 7 Operatoren +, -, *, /, ^, (,) und 18 mathematische Funktionssymbole verwendet werden: *ABS, ACOS, ASIN, ATAN, COS, DEG, EXP, FAK, LOG, LN, RAD, ROUND, SIN, SQR, SQRT, TAN, TRUNC und PI.* Außerdem sind bis zu 30 Klammerebenen möglich. Die Variable ***ATT*** enthält den Code etwaiger Auswertungsfehler: ***0*** = der Term besteht nur aus einem einfachen Text, ***1*** = der Term besteht nur aus einer Zahl, ***2*** = der Term besteht aus einer auswertbaren Formel (Normalfall). Die Codes ***4, 5*** oder ***6*** besagen, dass ein Auswertungsfehler aufgetreten ist. In diesem Fall wird der Funktionswert immer auf Null gesetzt.

Das Programm "***mathe***" demonstriert die Anwendung des Formelparsers. Wichtig ist zu erwähnen, dass die Parserunit ***mparse_u.pas*** in die Projektunit ***mathe_u.pas*** mittels ***uses*** eingebunden wird. Die 26 globalen Variablen ***A_, B_, ... Z_*** der Unit sind deswegen mit einem Unterstrich versehen, damit es zu keinen Neu-Definitionen von bereits bestehenden gleichnamigen Variablen kommt. Dadurch können etwaige Fehler bei Wertebelegungen vermieden werden. Die Abbildung zeigt das Programmformular und dann folgt das Programmlisting. Der über zwölf A4-Seiten umfassende Quellcode der Parserunit ***mparse_u.pas*** befindet sich auf der Begleit-CD.




```
unit mathe_u;
{ Demonstration des Formelparsers mparse_u.pas (c) H.Paukert }

interface
uses  Windows, Messages, SysUtils, Classes, Graphics,
      Controls, Forms, Dialogs, StdCtrls, ExtCtrls,
      mparse_u;
type
  TForm1 = class(TForm)
    Memo1: TMemo;                          //  Informationsfeld
    Label1: TLabel;                        //  Verschiedene Beschriftungen
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit1: TEdit;                          //  Eingabe von X
    Edit2: TEdit;                          //  Eingabe von Y
    Edit3: TEdit;                          //  Eingabe von Z
    Edit4: TEdit;                          //  Eingabe des Funktionsterms S
    Edit5: TEdit;                          //  Ausgabe des Funktionswertes W
    Edit6: TEdit;                          //  Ausgabe der Fehlervariablen ATT
    Button1: TButton;                      //  Programm beginnen
    Button2: TButton;                      //  Funktion berechnen
    Button3: TButton;                      //  Programm beenden
    Bevel1: TBevel;                        //  Rahmen um die Eingabefelder
    procedure FormActivate(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    private { Private declarations }
   public  { Public declarations }
  end;
```

```
var  Form1: TForm1;

implementation
{$R *.DFM}

var S : String;
    W : Real;
    X,Y,Z : Real;

procedure FitForm(F :TForm);
{ Anpassung des Formulares an die jeweilige Monitorauflösung }
begin
  with F do begin
    if (Screen.Width<>1024) then ScaleBy(Screen.Width,1024);
    if (Font.PixelsPerInch<>120) then ScaleBy(120,Font.PixelsPerInch);
    WindowState := wsMaximized;
  end;
end;

procedure TForm1.FormActivate(Sender: TObject);
{ Initialisierungen }
begin
  FitForm(Form1);
  Color := RGB(140,160,150);
  Button1.SetFocus;
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
{ Mit der <Enter>-Taste weiter zur nächsten Eingabe }
begin
  Edit5.Text := '';
  Edit6.Text := '';
  if (Sender = Edit1) and (Key = #13) then Edit2.SetFocus;
  if (Sender = Edit2) and (Key = #13) then Edit3.SetFocus;
  if (Sender = Edit3) and (Key = #13) then Edit4.SetFocus;
  if (Sender = Edit4) and (Key = #13) then Button2.SetFocus;
end;

procedure TForm1.Button1Click(Sender: TObject);
{ Eingabefelder löschen }
begin
  Edit1.Text := ''; Edit2.Text := ''; Edit3.Text := '';
  Edit4.Text := ''; Edit5.Text := ''; Edit6.Text := '';
  Edit1.SetFocus;
end;

procedure TForm1.Button2Click(Sender: TObject);
{ mathematische Formel berechnen }
var T : String;
    Code : Integer;
begin
  T := Edit1.Text; val(T,X,Code); X_ := X;    // bzw. mparse_u.X_ := X;
  T := Edit2.Text; val(T,Y,Code); Y_ := Y;    // bzw. mparse_u.Y_ := Y;
  T := Edit3.Text; val(T,Z,Code); Z_ := Z;    // bzw. mparse_u.Z_ := Z;
  S := Edit4.Text;
  if Trim(S) = '' then Exit;
  W := Parse(S,ATT);
  Edit5.Text := FloatToStr(W);
  Edit6.Text := IntToStr(ATT);
  Button1.SetFocus;
end;

procedure TForm1.Button3Click(Sender: TObject);
// Programm beenden
begin
  Application.Terminate;
end;

end.
```

[02] Reelle Nullstellen von Funktionen (*null*)

Gegeben ist die allgemeine Gleichung F(X) = 0. Eine reelle Lösung dieser Gleichung entspricht einer Nullstelle der Funktion F(X). Wird angenommen, dass die Funktion stetig auf dem Intervall [L,R] ist und dass das Vorzeichen der Funktionswerte in den Intervallrändern verschieden ist, dann muss die Funktion in dem Intervall eine Nullstelle besitzen (d.h. die X-Achse schneiden).

Das Intervall [L,R] wird nun halbiert und somit in zwei Teilintervalle zerlegt: [L,M] und [M,R] mit M = (L+R)/2. Man überprüft nun wieder das Vorzeichen der Funktion in den Randpunkten und entscheidet sich für jenes Teilintervall, wo ungleiche Vorzeichen vorliegen. Dieses Intervall wird wieder halbiert und das Verfahren in gleicher Weise fortgesetzt (Verfahren der fortgesetzten Intervallhalbierung, auch BISEKTION genannt). Dadurch erhält man eine Folge ineinander geschachtelter Intervalle, deren innerster Punkt die gesuchte Nullstelle ist. Das Verfahren wird dann abgebrochen, wenn der Funktionswert der Intervallmitte F(M) unter eine vorgegebene Genauigkeit GEN gesunken ist: abs(F(M)) < GEN.

Das Programm beginnt mit der Eingabe des Funktionsterms S als String und setzt fort mit der Eingabe eines linken Startwertes A, eines rechten Endwertes B und der Nullstellengenauigkeit GEN. Außerdem muss noch die Anzahl N der Tabellierungsschritte eingegeben werden.

Ein Tabellierungsschritt hat daher die Breite D := (B - A) / N. Dann wird, beginnend mit A, das Argument der Funktion so lange um die Breite D erhöht, bis ein Intervall [L,R] erreicht ist, wo die Funktion ihr Vorzeichen wechselt. Ein Vorzeichenwechsel liegt vor, wenn das Produkt der Funktionswerte in den beiden Intervallrändern negativ ist: F(L) * F(R) < 0. In diesem Intervall wird dann das iterative Verfahren der BISEKTION durchgeführt. Durch Prüfung der Bedingung abs(F(L)) < GEN bzw. abs(F(R)) < GEN werden jene Nullstellen sofort erfasst, die direkt auf den Randpunkten eines Teilintervalles liegen.

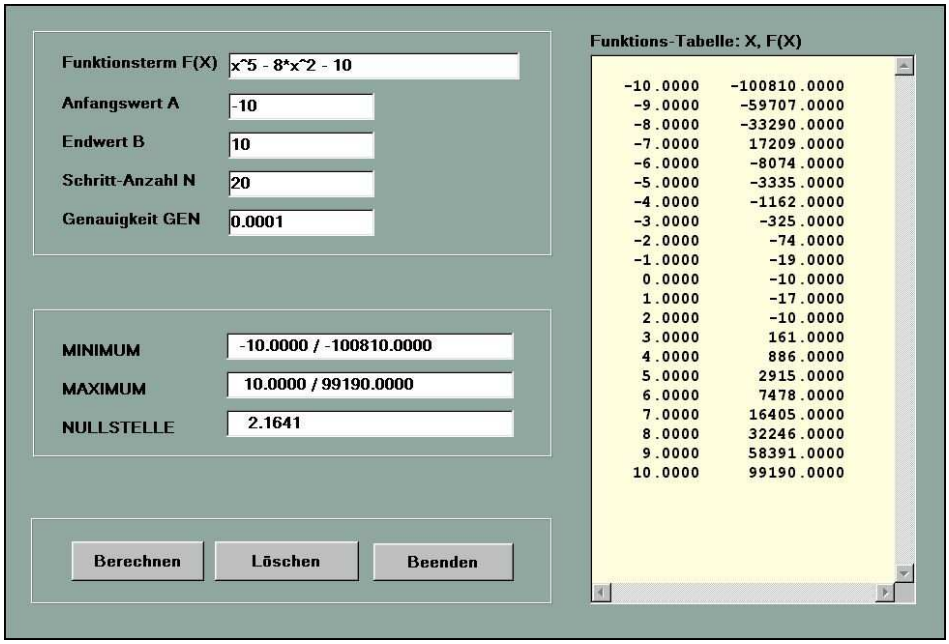
Das zentrale Funktionsunterprogramm ***F(S,X)*** liefert für den Funktionsterm S und das Argument X den zugehörigen Funktionswert. Zur Auswertung des Funktionsterms wird der Formelparser aus der Unit ***mparse_u.pas*** verwendet. Der Wert der globalen Variablen ERROR hängt von der Parservariablen ATT ab. Er ist nur dann *True*, wenn ein Fehler bei der Funktionsberechnung aufgetreten ist (beispielsweise Division durch Null oder Wurzel aus negativen Zahlen).

Das Funktionsunterprogramm ***Nuls(S,A,B,N,GEN)*** sucht zwischen Startwert A und Endwert B mit der Genauigkeit GEN eine reelle Nullstelle. Dabei sind S der Funktionsterm und N die Anzahl der Tabellierungsschritte. Wird eine reelle Nullstelle gefunden, so wird die globale Variable ERROR auf *False* gesetzt und der Wert der Nullstelle zurückgeliefert. Andernfalls erhält ERROR den Wert *True* und die Nullstelle den Wert 0. Auf die Funktion selbst wird über die Routine ***F(S,X)*** zugegriffen, welche ihrerseits den Formelparser aufruft.

Die zusätzliche Prozedur ***MiniMax(S,A,B,N)*** tabelliert eine Funktion auf dem Intervall [A,B] und ermittelt den kleinsten und größten Funktionswert. Diese Werte werden in den globalen Variablen XU,YU und XO,YO gespeichert. Argumente und Werte der Funktion werden in einem Memofeld ausgegeben. Der Parameter S ist der Funktionsterm und der Parameter N gibt die Anzahl der Tabellierungsschritte der Funktion an. Auf die Funktion selbst wird über die Routine ***F(S,X)*** zugegriffen, welche ihrerseits den Formelparser aufruft.

Zwei Nachteile dieses Verfahrens seien nicht verschwiegen: (1) Wenn die Kurve die X-Achse nur berührt und nicht schneidet, wird die Nullstelle nicht sicher erfasst. (2) Wenn die Anzahl der Tabellierungsschritte N ungünstig gewählt wird, dann können Vorzeichenwechsel und Nullstelle übersehen werden.

Die Abbildung zeigt das Programmformular und dann folgt das Programmlisting der wichtigsten Routinen.




```
var  A, B  : Real;                    // Anfangs- und Endwert
     N     : Integer;                 // Anzahl der Tabellierungsschritte
     GEN   : Real;                    // Nullstellen-Genauigkeit
     XU,YU : Real;                    // Funktions-Minimum
     XO,YO : Real;                    // Funktions-Maximum
     XNULL : REAL;                    // Reelle Nullstelle
     ERROR : Boolean;                 // Fehlervariable

function F(S:string; X:real): Real;
// Funktionsberechnung, S = Funktionsterm, X = Argument
begin
  ERROR := False;
  mparse_u.X_ := X; F := Parse(S,ATT);
  if (ATT>3) then ERROR := True;
end;

procedure MiniMax(S:string; A,B:real; N:integer);
// Tabellierung einer Funktion umd Ermittlung von ihrem
// Minimum und Maximum zwischen A und B. N = Schrittanzahl,
// S = Funktionsterm. Es wird die Funktion F(S,X) verwendet.
var FIRST : Boolean;
    X,Y,D : Real;
    U,V   : String;
begin
  if (B < A) then begin
     D := A; A := B; B := D;
  end;
  D := (B-A)/N;
  FIRST := True;
  X := A;
  repeat                              // Beginn der Tabellierung
    Y := F(S,X);
    if not ERROR then begin
       if FIRST then begin
          XU := X; YU := Y; XO := X; YO := Y; FIRST := False;
       end;
       if not FIRST then begin
          if Y < YU then begin XU := X; YU := Y; end;
          if Y > YO then begin XO := X; YO := Y; end;
       end;
       Str(X:10:4,U); Str(Y:10:4,V);
       Form1.Memo1.Lines.Add(#32 + U + #9 + V);
    end;
    X := X + D;
  until X > B;
end;
```

```
function Nuls(S:String; A,B:Real; N:Integer; Gen:Real): Real;
// Reelle Nullstellen zwischen A und B mit Genauigkeit Gen,
// S = Funktionsterm, N = Anzahl der Tabellierungsschritte.
// Die Funktion F(S,X) wird dabei verwendet. Abbruch mit <Esc>.
var L,R,M,YL,YR,YM,D: Real;
begin
  ERROR := False;
  if (B < A) then begin
    D := A; A := B; B := D;
  end;
  D := (B-A)/N;
  R := A;
  repeat                              // Beginn der Tabellierung
    L := R; YL := F(S,L);
    if Not ERROR and (Abs(YL) < Gen) then begin
       Nuls := L;
       Exit;
    end;
    R := L + D; YR := F(S,R);
    if (YL*YR <= 0) then begin        // Wenn Vorzeichenwechsel, dann Bisektion
       repeat
         if (getAsyncKeyState(vk_Escape) <> 0 ) then begin
            ERROR := True;
            Nuls  := 0;
            Exit;
         end;
         M := (L+R)/2; YM := F(S,M);
         if (YL*YM <= 0) then R := M
                         else L := M;
       until (Abs(YM) < Gen);
       if Not ERROR and (Abs(YM) < Gen) then begin
          Nuls := M;
          Exit;
       end;
    end;
  until R >= B;
  ERROR := True;
end;
```

[03] Numerisches Differenzieren von Funktionen (*diff*)

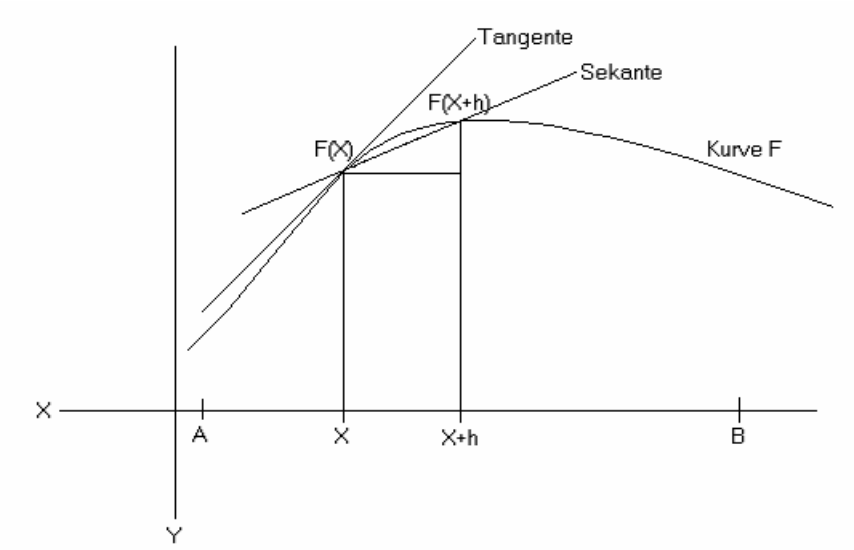
Will man in einem Punkt P(x/F(x)) einer gegebenen Funktion F(x) den Differentialquotienten ermitteln, dann lässt sich diese wichtige mathematische Aufgabe auf zwei Arten lösen.

Bei der ersten Methode muss zu jeder gegebenen Funktion die Ableitungsfunktion explizit ermittelt und in einem eigenen Funktionsunterprogramm deklariert werden. Sicherlich eine eher unbefriedigende Lösung, weil jedesmal das Programm umgeschrieben und außerdem die genaue Kenntnis der Regeln der Differentialrechnung vorausgesetzt werden muss.

Bei der zweiten Methode, dem numerischen Differenzieren, wird nach einem allgemeinen Verfahren ein Näherungswert für den Differentialquotienten in dem Punkt P(x/F(x)) berechnet. Zur Auswertung der Funktion wird der Formelparser aus der Unit ***mparse_u.pas*** verwendet.

Es existieren mehrere numerische Näherungsverfahren zur Ermittlung des Differentialquotienten. Im Folgenden soll die einfachste Annäherung verwendet werden. Der Differentialquotient gibt den Anstieg der Kurventangente im Punkt P(x/F(x)) an. Wählen wir nun in seiner unmittelbaren Umgebung einen zweiten Kurvenpunkt Q(x+h/F(x+h)), wobei h ein beliebig kleiner Zahlenwert ist. Die Sekante, welche durch die Punkte P und Q verläuft, wird als Näherung für die Tangente im Punkt P genommen. Diese Näherung ist um so besser, je kleiner der Zahlenwert h ist (z.B. h = 0.001), also je näher der Punkt Q an den Punkt P heranrückt. Für die Sekantensteigung k gilt folgende Beziehung, wobei α der Richtungswinkel der Sekante ist:

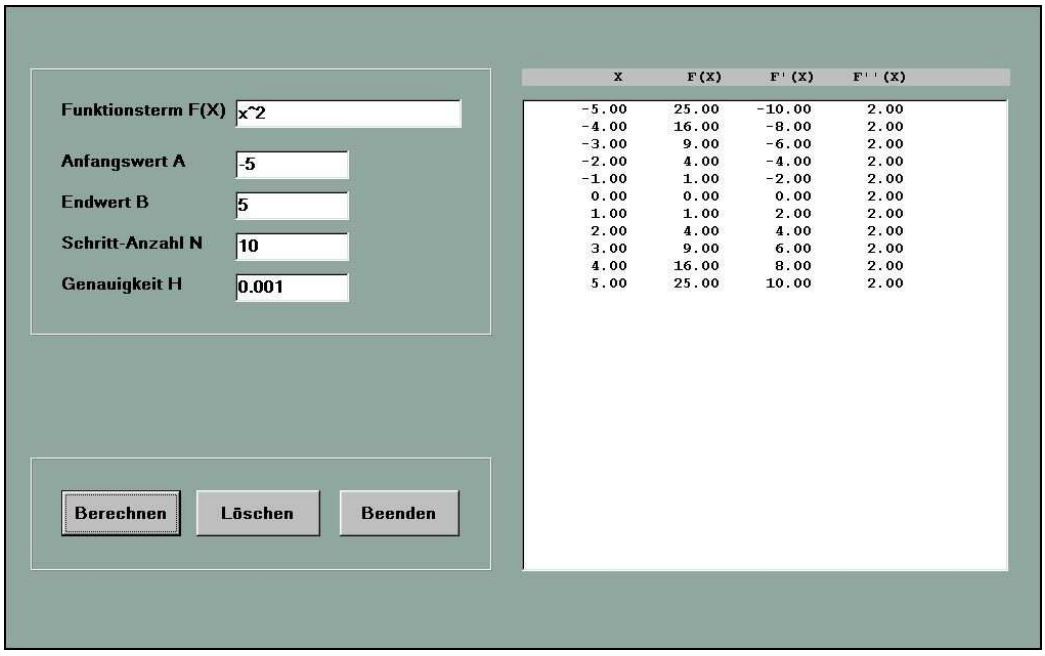
$$k = \tan(\alpha) = (F(x+h) - F(x)) / h$$



In dem Funktionsunterprogramm ***F1(S,X,h)*** wird obiger Sekantenanstieg k als Näherungswert für den Differentialquotienten verwendet. Dabei sind der Parameter S der Funktionsterm, X das Argument der Funktion und h die kleine Intervallbreite. Auf die Funktion selbst wird über die Routine ***F(S,X)*** zugegriffen, welche ihrerseits den Formelparser aufruft.

In dem Funktionsunterprogramm ***F2(S,X,h)*** wird nach dem gleichen Näherungsverfahren der zweite Differentialquotient der Funktion berechnet. Dabei wird im Programmcode der Routine die originale Funktion ***F(S,X)*** durch ***F1(S,X,h)*** ersetzt. Auf diese Art und Weise können auch höhere Ableitungen gebildet werden.

In der Prozedur ***Tabelle(S,A,B,N)*** erfolgt eine Tabellierung der Funktion. Dabei sind S der Funktionsterm, A der linke Startwert, B der rechte Endwert und N die Anzahl der Tabellierungsschritte. In einer Listbox werden die Werte *X, F(X), F1(X) und F2(X)* tabellenförmig ausgegeben.




```
var A,B   : Real;                  //  Anfangs- und Endwert
    N     : Integer;               //  Anzahl der Tabellierungsschritte
    h     : Real;                  //  Genauigkeit
    ERROR : Boolean;               //  Fehlervariable
```

```
function F(S:string; X:Real): Real;
// Funktionsberechnung, S = Funktionsterm, X = Argument
begin
  ERROR := False;
  mparse_u.X_ := X; F := Parse(S,ATT);
  if (ATT>3) then ERROR := True;
end;

function F1(S: String; X,h: Real): Real;
// Ermittelt näherungsweise die erste Ableitung
var Y,X1,Y1,K : Real;
begin
  Y := F(S,X);
  If Not ERROR then begin
     X1 := X + h;
     Y1 := F(S,X1);
     if Not ERROR then begin
        try
          K := (Y1-Y) / h;
          F1 := K;
        except
          ERROR := True;
          F1 := 0;
        end;
     end;
  end;
end;

function F2(S: String; X,h: Real): Real;
// Ermittelt näherungsweise die zweite Ableitung
var Y,X1,Y1,K : Real;
begin
  Y := F1(S,X,h);
  If Not ERROR then begin
     X1 := X + h;
     Y1 := F1(S,X1,h);
     if Not ERROR then begin
        try
          K := (Y1-Y) / h;
          F2 := K;
        except
          ERROR := True;
          F2 := 0;
        end;
     end;
  end;
end;

{
function F1(S: String; X,h: Real): Real;
// Alternatives numerisches Differenzieren durch Kurvenapproximation
// mit Hilfe einer Polynomfunktion 4. Grades in einer Umgebung von X.
// Ermittlung der ersten Ableitung.
begin
  try
    ERROR := False;
    F1 := (F(S,X-2*h)-8*F(S,X-h)+8*F(S,X+h)-F(S,X+2*h)) / (12*h);
  except
    ERROR := True;
    F1 := 0;
  end;
end;

function F2(S: String; X,h: Real): Real;
// Alternatives numerisches Differenzieren durch Kurvenapproximation
// mit Hilfe einer Polynomfunktion 4. Grades in einer Umgebung von X
// Ermittlung der zweiten Ableitung.
begin
  F2 := (F1(S,X-2*h,h)-8*F1(S,X-h,h)+8*F1(S,X+h,h)-F1(S,X+2*h,h)) / (12*h);
end;
}
```

```
procedure Tabelle(S: String; A,B: Real; N: Integer);
// Funktion tabellieren und Argument X, Funktionswert F(X),
// erste Ableitung F'(X) und zweite Ableitung F''(X) ausgeben.
// A = Startwert, B = Endwert, N = Tabellierungsschritte.
const W = 10;
      K = 2;
var   TX,T0,T1,T2,Zeile: String;
      D: Real;                    // Schrittweite der Tabellierung
      X,Y  : Real;                // Argument und Wert der Funktion
      D1,D2: Real;                // Erste und zweite Ableitung
begin
  Form1.ListBox1.Items.Clear;
  if (B < A) then begin
     D := A; A := B; B := D;
  end;
  D := (B-A)/N;
  X := A;
  repeat
    T0 := '  ........'; T1 := T0; T2 := T0;
    str(X:W:K,TX);
    Y  := F(S,X);
    if Not ERROR then begin
      str(Y:W:K,T0);
      D1 := F1(S,X,h);
      if not ERROR then begin
        str(D1:W:K,T1);
        D2 := F2(S,X,h);
        if not ERROR then str(D2:W:K,T2);
      end;
    end;
    Zeile := ' ' + TX + T0 + T1 + T2;
    Form1.ListBox1.Items.Add(Zeile);
    X := X + D;
  until X > B;
end;
```

[04] Numerisches Integrieren von Funktionen (*integ*)

Gegeben ist eine Funktion F(X), die auf dem Intervall [A,B] stetig ist. Es soll das bestimmte Integral der Funktion auf dem Intervall berechnet werden. Als Näherungsverfahren wird die Trapezformel verwendet. Dabei wird das Intervall [A,B] in N gleich lange Teilintervalle zerlegt, und in jedem dieser Teilintervalle wird die Funktionskurve durch eine Sehne ersetzt. Die Fläche unter der Kurve kann dann durch ein Trapez angenähert werden und die Summe dieser Trapezflächen ergibt einen Näherungswert für das bestimmte Integral. Das Verfahren ist umso genauer, je größer die Anzahl N der Teilintervalle ist. Für eine solche Trapezfläche gilt folgende Formel:

Trapezfläche = D * (F(X) + F(X+D)) / 2

Dabei ist D die Breite eines Tabellierungsschrittes (D = (B-A)/N) und X ist der jeweils erreichte Tabellierungspunkt (X = A + i*D mit i = 0,1,2, ... N-1). Für die Summe der Trapezflächen gilt die Formel:

$$\int_A^B F(X)dx \approx \mathrm{D} * (\mathrm{F(A)} + \mathrm{F(B)} + 2 * \sum_{\mathrm{i=1}}^{\mathrm{N-1}} \mathrm{F(A + i * D)}) \,/\, 2$$

Dieses Verfahren versagt, wenn innerhalb des Integrationsintervalls [A,B] ein Vorzeichenwechsel der Funktion stattfindet, weil die Funktion dann auf der einen Seite der X-Achse ein anderes Vorzeichen hat als auf der gegenüberliegenden Seite. Damit liefert obige Formel einen falschen Wert. Die schrittweise Berechnung des bestimmten Integralwertes erfolgt im Funktionsunterprogramm ***Trapez(S,A,B,N).*** Dabei sind S der Funktionsterm, A und B die Integralgrenzen und N die Anzahl der Tabellierungsschritte. Der Integralwert wird zurückgeliefert. Auf die Funktion selbst wird über die Routine ***F(S,X)*** zugegriffen, welche ihrerseits den Formelparser aufruft.

```
var A, B  : Real;                    // Anfangs- und Endwert
    N     : Integer;                 // Anzahl der Tabellierungsschritte
    GEN   : Real;                    // Nullstellen-Genauigkeit
    ERROR : Boolean;                 // Fehlervariable

function F(S:string; X:Real): Real;
// Funktionsberechnung, S = Funktionsterm, X = Argument
begin
  ERROR := False;
  mparse_u.X_ := X; F := Parse(S,ATT);
  if (ATT>3) then ERROR := True;
end;

function TRAPEZ(S: String; A,B: Real; N: integer): Real;
// Integralberechnung mittels Trapezformel, S = Funktionsterm,
// A,B = Integralgrenzen, N = Anzahl der Tabellierungsschritte
var D,SUM: Real;
    i: Integer;
begin
  TRAPEZ := 0;
  if (B < A) then begin
     D := A; A := B; B := D;
  end;
  D := (B-A)/N;
  SUM := F(S,A) + F(S,B);
  if ERROR then Exit;
  for i := 1 to (N-1) do begin
    SUM := SUM + 2 * F(S,A+i*D); if ERROR then Exit;
  end;
  TRAPEZ := D * SUM / 2;
end;

function SIMPSON(S: String; A,B: Real; N: Integer): Real;
// Alternatives numerisches Integrieren durch Kurvenapproximation
// mit Hilfe einer Polynomfunktion 2. Grades in einer Umgebung von X
var W,D,SUM: Real;
    i: Integer;
begin
  SIMPSON := 0;
  if Odd(N) then N := N + 1;
  if (B < A) then begin
     D := A; A := B; B := D;
  end;
  D := (B-A)/N;
  SUM := F(S,A); if ERROR then Exit;
  W := F(S,B);   if ERROR then Exit;
  for i := 1 to (N-1) do begin
    W := F(S,A+i*D); if ERROR then Exit;
    if Odd(i) then SUM := SUM + 4*W
              else SUM := SUM + 2*W;
  end;
  SIMPSON := SUM * D / 3;
end;
```

[05] Grafische Darstellung von Funktionen (funk)

Das Programm "***funk***" dient der Tabellierung und grafischen Darstellung von Funktionen. Zuerst wird der Funktionsterm F(X) als Strings S eingegeben, welcher in dem Funktionsunterprogramm ***F(S,X)*** mit Hilfe des Parsers der Unit ***mparse_u*** ausgewertet wird. Dabei ist S der jeweilige Funktionsterm und X das Argument. Die erste Ableitung der Funktion an der Stelle X wird mit Hilfe des Unterprogramms ***F1(S,X)*** ermittelt (siehe dazu die Buchabschnitte [01] und [03]).

Die Eingabe einer Funktionsformel S erfolgt im Eingabefeld einer ComboBox. Mit der Taste <F3> können dort mehrere Funktionen eingefügt und mit einem Mausklick auch wieder aktualisiert werden. Außerdem besteht die Möglichkeit, die Funktionsformeln in einer Textdatei zu speichern und von dieser zu laden (z.B. "funk.txt").

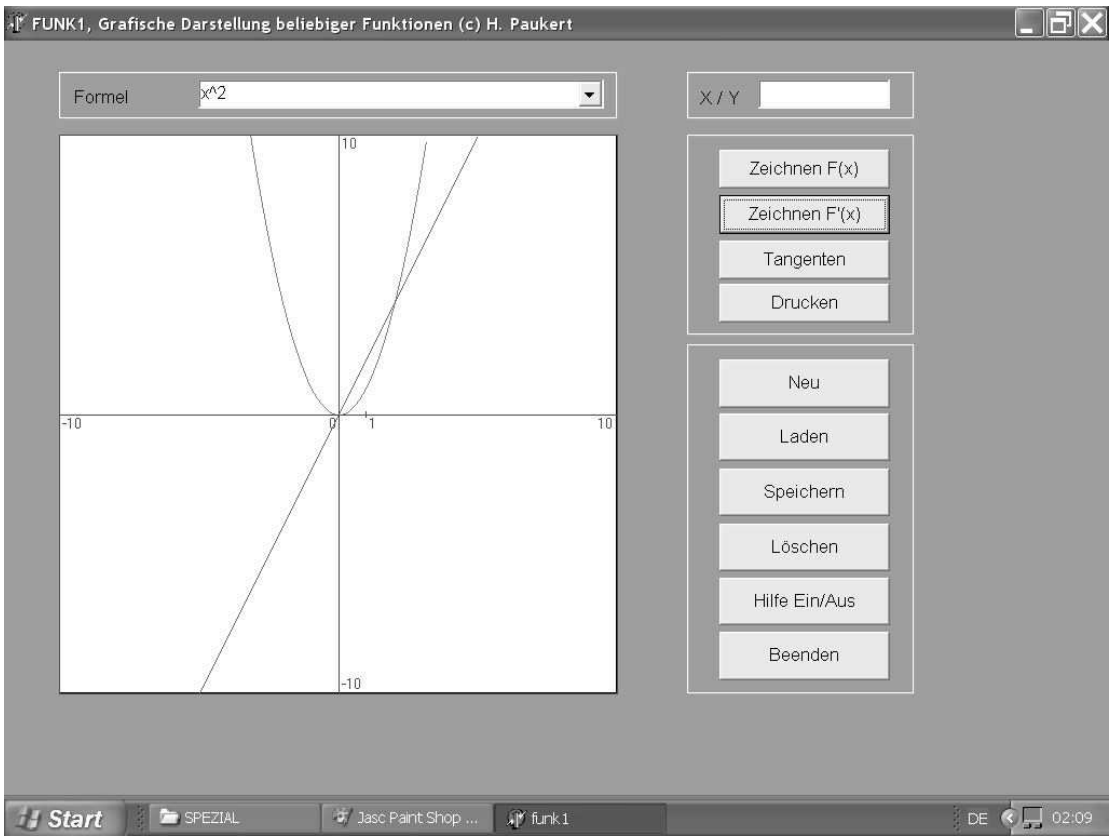
Zur grafischen Darstellung muss ein Weltbereich auf einen Bildbereich abgebildet werden. Der Sachverhalt vereinfacht sich wesentlich, wenn man als Weltbereich ein Quadrat mit der halben Seitenlänge ***g*** nimmt, also mit der Diagonale [*-g/+g*] – [*+g/-g*] begrenzt. Der Bildbereich ist durch die eingestellten Abmessungen der *Image*-Komponente [*0,0*] – [*Xmax,Ymax*] gegeben. Dabei ist *Xmax* die Bildbreite (*Image.Width*) und *Ymax* die Bildhöhe (*Image.Height*). Die Transformation eines Weltbereichs auf einen Bildbereich ist in „Grafik“ [03] ausführlich beschrieben.

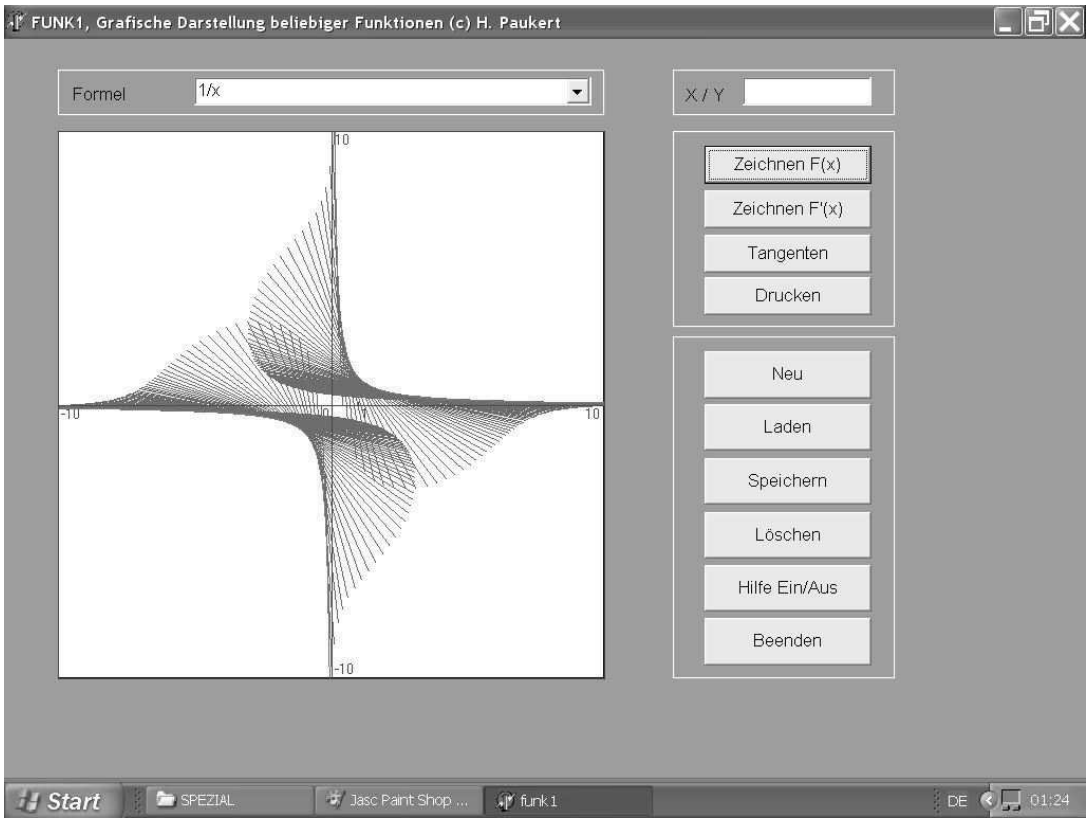
Ein Weltpunkt ***W*** ist durch den reellzahligen Record-Typ *WPunkt* definiert. Ein Bildpunkt ***B*** ist durch den ganzzahligen Record-Typ *TPoint* deklariert. Die Prozedur ***WeltZuBild*** berechnet zu gegebenen Weltkoordinaten *(W.x,W.y)* die zugehörigen Bildkoordinaten *(B.x,B.y).* Die Prozedur ***BildZuWelt*** leistet genau das Umgekehrte.

Die eigentliche grafische Darstellung einer mathematischen Funktion erfolgt im Unterprogramm ***Kurve(S,Wahl),*** wobei *S* der Funktionsterm ist und der Parameter *Wahl* angibt, ob die Funktion (*Wahl = 0*) oder ihre erste Ableitung (*Wahl = 1*) gezeichnet werden soll. In der Routine werden in einer Wiederholungsschleife zuerst die Weltkoordinaten und dann die Bildkoordinaten von jedem Funktionspunkt ermittelt. In der Schleife wird das gegebene Argumentintervall [*-g,+g*] in *Xmax* Schritten durchlaufen. Um das grafische Schaubild schöner zu gestalten, werden benachbarte Bildpunkte durch eine gerade Linie verbunden. Liegt der Punkt außerhalb des Weltbereichs, dann wird er nicht gezeichnet. Auf die Funktion selbst wird über die Routine ***F(S,X)*** zugegriffen, welche ihrerseits den Formelparser aufruft.

Die Achsen des Koordinatensystems werden in der Routine ***KoordinatenAchsen(g)*** gezeichnet. Dabei ist g die halbe Breite des Weltbereiches. Über einen Schaltknopf kann der Parameter g frei gewählt werden, jedoch muss er geradzahlig sein und zwischen 1 und 101 liegen.

Als besonderer Zusatz kann mit der Maus ein beliebiger Bildpunkt auf der Zeichenebene angeklickt werden, worauf eine Anzeige der Weltkoordinaten erfolgt. So ist es möglich, interessante Kurvenbereiche MANUELL abzutasten. Weiters können mit der Routine ***Tangenten(S,n)*** genau n Tangenten entlang der Kurve S gleiten. Schließlich besteht noch die Möglichkeit, die gezeichnete Kurve skaliert auszudrucken, d.h., die Größe des Druckbilds kann individuell eingestellt werden.






```
unit funk1_u;
// Grafische Darstellung beliebiger Funktionen (c) H.Paukert

interface
uses  Windows, Messages, SysUtils, Classes, Graphics, Math,
      Controls, Forms, Dialogs, Printers, StdCtrls, ExtCtrls,
      mparse_u;
type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Button10: TButton;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    ComboBox1: TComboBox;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;


    procedure FormActivate(Sender: TObject);
    procedure Image1MouseDown(Sender: TObject; Button: TMouseButton;
                              Shift: TShiftState; X, Y: Integer);
    procedure ComboBox1KeyUp(Sender: TObject; var Key: Word;
                             Shift: TShiftState);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
```

```
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
  private { Private-Deklarationen }
  public  { Public-Deklarationen }
end;

var Form1: TForm1;

implementation
{$R *.DFM}

type  WPunkt = record                  // Selbstdefinierter Typ
         X : Real;                     // eines Weltpunktes
         Y : Real;
      end;
      BPunkt = TPoint;                 // eines Bildpunktes

const g    : Integer = 10;             // Halbe Breite des Weltbereiches
var   M    : BPunkt;                   // Koordinatenursprung
      B    : BPunkt;                   // Ein Bildpunkt
      W    : WPunkt;                   // Ein Weltpunkt
      Xmax : Integer;                  // Bildbreite (Image.Width)
      Ymax : Integer;                  // Bildhöhe (Image.Height)
      FName: String;                   // Dateiname
      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;
      Z: Real;
begin
  with F do begin
    X := Screen.Width;
    Y := Screen.Height;
    K := Font.PixelsPerInch;
    Scaled := True;
    Z := Y/X;
    if (Z >= 0.75) then ScaleBy(X,SW) else ScaleBy(Y,SH);
    if (K <> FS) then ScaleBy(FS,K);
    WindowState := wsMaximized;
  end;
end;

procedure WeltZuBild(W: WPunkt; var B: BPunkt);
{ Rechnet die übergebenen Weltkoordinaten in  }
{ Bildkoordinaten um.                         }
begin
  B.X := Round((W.X + g) * Xmax / (2*g));
  B.Y := Round(Ymax - (W.Y + g) * Ymax / (2*g));
end;

procedure BildZuWelt(B: BPunkt; var W: WPunkt);
{ Rechnet die übergebenen Bildkoordinaten  }
{ in Weltkoordinaten um.                   }
begin
  W.X := (2*g*B.X - g*Xmax) / Xmax;
  W.Y := (-2*g*B.Y + g*Ymax) / Ymax;
end;

procedure KoordinatenAnzeigen(W: WPunkt);
{ Zeigt die Koordinaten eines Weltpunkte W an }
var Sx,Sy,S : String;
begin
  str(W.X:6:2,Sx); str(W.Y:6:2,Sy);
  S := ' ' + Trim(Sx) + ' / ' + Trim(Sy) + ' ';
  Form1.Edit1.Text := S;
end;
```

```
procedure Strecke(A,B: WPunkt);
{ Strecke durch A und B zeichnen (im Weltbereich [-g,+g]) }
var C,D: BPunkt;
begin
  WeltZuBild(A,C);
  WeltZuBild(B,D);
  Form1.Image1.Canvas.Pen.Color := clGreen;
  Form1.Image1.Canvas.MoveTo(C.X,C.Y);
  Form1.Image1.Canvas.LineTo(D.X,D.Y);
  Form1.Image1.Canvas.Pen.Color := clBlack;
end;

procedure Gerade(A,B: WPunkt);
{ Gerade durch A und B zeichnen (im Weltbereich [-g,+g]) }
var k,d: Real;
    P,Q: WPunkt;
    P1,Q1: BPunkt;
begin
  if B.X = A.X then begin
     P.X := A.X;
     P.Y := -g;
     Q.X := A.X;
     Q.Y := -g;
  end
  else begin
     if B.y = A.y then begin
       P.x := -g;
       P.y := A.y;
       Q.x := g;
       Q.y := A.y;
     end
     else begin
       k := (B.Y-A.Y) / (B.X-A.X);
       d := A.Y - k * A.X;
       P.X := -(g+d)/k;
       P.Y := -g;
       Q.X := (g-d)/k;
       Q.Y := g;
     end;
  end;
  WeltZuBild(P,P1);
  WeltZuBild(Q,Q1);
  Form1.Image1.Canvas.Pen.Color := clGreen;
  Form1.Image1.Canvas.MoveTo(P1.X,P1.Y);
  Form1.Image1.Canvas.LineTo(Q1.X,Q1.Y);
  Form1.Image1.Canvas.Pen.Color := clBlack;
end;

function F(S: String; X: Real): Real;
{ Funktionswert von S an der Stelle X berechnen }
const FLim: Real = 1E16;   // Funktions-Grenze
var   Y: Real;
begin
  Error := False;
  mparse_u.X_ := X;
  Y := Parse(S,ATT);
  if Y > FLim then Y := FLim;
  if Y < -FLim then Y := -FLim;
  Result := Y;
  if (ATT > 3) then Error := True;
end;

function F1(S: String; X: Real): Real;
{ Erste Ableitung: Numerisches Differenzieren durch Kurvenapproximation }
{ mit Hilfe einer Polynomfunktion 4. Grades in einer Umgebung (h) von X }
const h: Real = 0.001;   // Genauigkeit der Ableitung
begin
  Result := (F(S,X-2*h)-8*F(S,X-h)+8*F(S,X+h)-F(S,X+2*h)) / (12*h);
end;

function F2(S: String; X: Real): Real;
{ Zweite Ableitung: numerisches Differenzieren durch Kurvenapproximation }
{ mit Hilfe einer Polynomfunktion 4. Grades in einer Umgebung (h) von X }
const h: Real = 0.001;   // Genauigkeit der Ableitung
begin
  Result := (F1(S,X-2*h)-8*F1(S,X-h)+8*F1(S,X+h)-F1(S,X+2*h)) / (12*h);
end;
```

```
procedure Kurve(S: String; Wahl: Integer);
{ Zeichnet bei Wahl = 0 das Schaubild einer Funktion mit der Formel S }
{ Zeichnet bei Wahl = 1 das Schaubild der ersten Ableitungs-Funktion  }
{ im Weltbereich [-g,+g] }
var  Anfa, Ende, Diff : Real;
     A,B : BPunkt;
     W   : WPunkt;
begin
   with Form1.Image1.Canvas do begin
      if Wahl = 0 then Pen.Color := clRed
                  else Pen.Color := clGreen;
      Anfa := -g;
      Ende := g;
      Diff := (Ende-Anfa) / Xmax;
      W.X  := Anfa;
      if Wahl = 0 then W.Y := F(S,W.X)
                  else W.Y := F1(S,W.X);
      WeltZuBild(W,B);
      A := B;
      repeat
        W.X := W.X + Diff;
        if Wahl = 0 then W.Y := F(S,W.X)
                    else W.Y := F1(S,W.X);
        WeltZuBild(W,B);
        MoveTo(A.X,A.Y);
        if ((abs(W.Y)) > g) then MoveTo(B.X,B.Y)
                            else LineTo(B.X,B.Y);
        A := B;
      until W.X >= Ende;
      Pen.Color := clBlue;
      MoveTo(0,M.Y); LineTo(Xmax,M.Y);
      MoveTo(M.X,0); LineTo(M.X,Ymax);
      Rectangle(0,0,Xmax,Ymax);
      Pen.Color := clBlack;
   end;
end;

procedure Tangente(S: String; n: Integer);
{ Zeichnet n Tangenten entlang der Funktionskurve S }
{ im Weltbereich [-g,+g] }
var  Anfa,Ende,Diff,k,wi,d,x,y: Real;
     W,U,V: WPunkt;
     lim: Boolean;
begin
   lim := True;
   if n < 0 then begin
      n := - n;
      lim := False;
   end;
   with Form1.Image1.Canvas do begin
      Pen.Width := 1;
      Pen.Color := clBlack;
      d := g div 2;
      Anfa := -g;
      Ende := g;
      Diff := (Ende-Anfa) / n;
      W.X  := Anfa;
      W.Y := F(S,W.X);
      k   := F1(S,W.X);
      wi  := arctan(k);
      x   := d*cos(wi);
      y   := d*sin(wi);
      U.X := Anfa - x;  U.Y := W.Y - y;
      V.X := Anfa + x;  V.Y := W.Y + y;
      if lim then Strecke(U,V) else Gerade(U,V);
      repeat
        W.X := W.X + Diff;
        W.Y := F(S,W.X);
        k   := F1(S,W.X);
        wi  := arctan(k);
        x   := d*cos(wi);
        y   := d*sin(wi);
        U.X := W.X - x;  U.Y := W.Y - y;
        V.X := W.X + x;  V.Y := W.Y + y;
        if (abs(V.Y) < 2*g) then begin
           if lim then Strecke(U,V) else Gerade(U,V);
        end;
      until W.X >= Ende;
```

```
      Pen.Color := clBlue;
      MoveTo(0,M.Y); LineTo(Xmax,M.Y);
      MoveTo(M.X,0); LineTo(M.X,Ymax);
      Pen.Color := clBlack;
      Pen.Width := 1;
   end;
end;

procedure FormImagePrint(FO: TForm; IM: TImage);
{ Nur ein Image "IM" eines Formulars "FO" ausdrucken }
type VA = array[0..999] of Boolean;
var  I: Integer;
     V: VA;
     C: TColor;
begin
  For I := 0 to FO.ControlCount-1 do begin
      V[I] := FO.Controls[I].Visible;
      if V[I] then FO.Controls[I].Visible := False;
  end;
  IM.Visible := True;
  C := FO.Color;
  FO.Color := clWhite;
  Form1.PrintScale := poPrintToFit;
  Printer.Orientation := poLandscape;
  Form1.Print;
  Printer.Orientation := poPortrait;
  FO.Color := C;
  For I := 0 to Form1.ControlCount-1 do begin
      if V[I] then Form1.Controls[I].Visible := True;
  end;
end;

procedure PrintImage(IM: TImage; f: Real);
{ Ausdruck eines Images mit Skalierungsfaktor f (A4-Format mit f = 1) }
var Breite, Hoehe: Integer;
    Faktor: Real;
    Bereich: TRect;
    Bild: TBitMap;
begin
  with IM do begin
    if f = 0 then Exit;
    f := abs(f);
    Bild := TBitMap.Create;
    Bild.Width  := ClientWidth;
    Bild.Height := ClientHeight;
    Bild.PixelFormat := pf24Bit;
    Bild.Canvas.CopyRect(Rect(0,0,ClientWidth,ClientHeight),
                         Canvas,Rect(0,0,ClientWidth,ClientHeight));
    Faktor := ClientHeight / ClientWidth;
    Printer.BeginDoc;
      Breite := Round(Printer.Canvas.ClipRect.Right * f);
      Hoehe  := Round(Breite * Faktor);
      if Hoehe > Printer.Canvas.ClipRect.Bottom then begin
         Hoehe   := Round(Printer.Canvas.ClipRect.Bottom * f);
         Breite  := Round(Hoehe / Faktor);
      end;
      Bereich:= Rect(0,0,Breite,Hoehe);
      Printer.Canvas.StretchDraw(Bereich,Bild);
    Printer.EndDoc;
    Bild.Free;
  end;
end;

procedure KoordinatenAchsen(g: Integer);
{ Koordinaten-Achsen zeichnen }
var S0,S1,S2: String;
    dx: Integer;
begin
  S0 := '0';
  S1 := ' ' + IntToStr(g) + ' ';
  S2 := ' ' + IntToStr(-g) + ' ';
  dx := Xmax div (2*g);
  with Form1.Image1.Canvas do begin
    Font.Color:= clBlue;
    Pen.Color := clBlack;
    Pen.Width := 2;
    Pen.Style := psSolid;
    Brush.Color := clWhite;
```

```
    Brush.Style := bsSolid;
    Rectangle(0,0,Xmax,Ymax);
    Brush.Style := bsClear;
    Pen.Color := clBlue;
    Pen.Width := 1;
    MoveTo(0,M.Y); LineTo(Xmax,M.Y);
    MoveTo(M.X,0); LineTo(M.X,Ymax);
    TextOut(M.X-TextWidth(S0)-2,M.Y,S0);
    TextOut(M.X,0,S1);
    TextOut(M.X,Ymax-TextHeight(S2)-2,S2);
    TextOut(0,M.Y,S2);
    TextOut(Xmax-TextWidth(S1)-2,M.Y,S1);
    MoveTo(M.X+dx,M.Y-3); LineTo(M.X+dx,M.Y+3);
    TextOut(M.X+dx,M.Y,' 1 ');
    Pen.Color := clBlack;
    Font.Color:= clBlack;
  end;
end;

procedure TForm1.FormActivate(Sender: TObject);
{ Initialisierungen }
begin
  Error := False;
  Fname := '';
  FitForm(Form1);
  Color := RGB(140,160,180);
  Xmax := Image1.ClientWidth; Ymax := Image1.ClientHeight;
  M.X  := Round(Xmax/2); M.Y  := Round(Ymax/2);
  g    := 10;
  KoordinatenAchsen(g);
  ComboBox1.SetFocus;
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
{ Umrechnung des mit der linken Maustaste angeklickten  }
{ Bildpunktes in Weltkoordinaten. }
begin
  if Button = mbLeft then begin
     B.X := X; B.Y := Y;
     BildZuWelt(B,W);
     KoordinatenAnzeigen(W);
  end;
end;

procedure TForm1.ComboBox1KeyUp(Sender: TObject; var Key: Word;
                                Shift: TShiftState);
{ Dateneingabe in der Combobox mit Enter-Taste }
begin
  if Key = 13 then begin
     ComboBox1.Items.Add(ComboBox1.Text);
     ComboBox1.Text := '';
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
// Funktion Zeichnen
var S : String;
begin
  S := Trim(ComboBox1.Text);
  if S = '' then begin
     ShowMessage('Funktionsformel eingeben !'); ComboBox1.SetFocus; Exit;
  end;
  Kurve(S,0);
end;

procedure TForm1.Button2Click(Sender: TObject);
// Erste Ableitungs-Funktion zeichnen
var S: String;
begin
  S := Trim(ComboBox1.Text);
  if S = '' then begin
     ShowMessage('Funktionsformel eingeben !'); ComboBox1.SetFocus; Exit;
  end;
  Kurve(S,1);
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
// Tangenten-Scharen
var R,S,T: String;
    n,Code: Integer;
begin
  S := Trim(ComboBox1.Text);
  if S = '' then begin
     ShowMessage('Funktionsformel eingeben !');
     ComboBox1.SetFocus;
     Exit;
  end;
  R := 'Anzahl der gleitenden Tangenten (N)';
  T := 'N = 10 ... 1000, bei N < 0 unbegrenzte Längen';
  T := InputBox(R,T,'100');
  Val(T,n,Code);
  if (n > -10) and (n < 10) then n := 100;
  if (n < -1000) or (n > 1000) then n := 100;
  Tangente(S,n);
end;

procedure TForm1.Button4Click(Sender: TObject);
// ComboBox und Image löschen
var s : String;
    Code : Integer;
begin
  s := InputBox('','Eingabe der geraden halben Weltbreite 2 ... 100','10');
  val(s,g,Code);
  if (g<2) or (g>100) or odd(g) then g := 10;
  KoordinatenAchsen(g);
  Edit1.Clear;
  ComboBox1.Items.Clear;
  ComboBox1.Text := '';
  ComboBox1.SetFocus;
end;

procedure TForm1.Button5Click(Sender: TObject);
// Funktionsformeln aus Texdatei laden
begin
  with OpenDialog1 do begin
    InitialDir := GetCurrentDir;
    Filter     := 'Textdatei (*.txt)|*.txt';
    DefaultExt := 'txt';
    Options    := [ofFileMustExist];
    FileName   := 'funk';
    if OpenDialog1.Execute then FName := FileName
                           else Exit;
  end;
  ComboBox1.Items.LoadFromFile(FName);
  ComboBox1.SetFocus;
end;

procedure TForm1.Button6Click(Sender: TObject);
// Funktionsformeln in Textdatei speichern
begin
  with SaveDialog1 do begin
    InitialDir := GetCurrentDir;
    Filter     := 'Textdatei (*.txt)|*.txt';
    DefaultExt := 'txt';
    Options    := [ofOverwritePrompt];
    FileName   := FName;
    if Execute then FName := FileName else Exit;
  end;
  ComboBox1.Items.SaveToFile(FName);
  ComboBox1.SetFocus;
end;

procedure TForm1.Button7Click(Sender: TObject);
// Image löschen
var s : String;
    Code : Integer;
begin
  s := InputBox('','Eingabe der geraden halben Weltbreite 2 ... 100','10');
  val(s,g,Code);
  if (g<2) or (g>100) or odd(g) then g := 10;
  KoordinatenAchsen(g);
  Edit1.Clear;
  ComboBox1.SetFocus;
end;
```

```
procedure TForm1.Button8Click(Sender: TObject);
// Hilfetext Ein/Ausblenden
begin
  Memo1.Visible := NOT Memo1.Visible;
  ComboBox1.SetFocus;
end;


procedure TForm1.Button9Click(Sender: TObject);
// Programm beenden
begin
  Application.Terminate;
end;


procedure TForm1.Button10Click(Sender: TObject);
// Das Image ausdrucken
var s,t : String;
    k : Real;
    Code : Integer;
begin
  t := 'k = 1 für ganze Druckseite, k = 0 für Abbruch';
  s := InputBox('Druckskalierung k (0.1 bis 10) eingeben',t,'1.00');
  s := Trim(s);
  val(s,k,Code);
  if (Code <> 0) or (k < 0.1) or (k > 10) then k := 0;
  PrintImage(Form1.Image1,k);
  ComboBox1.SetFocus;
end;

end.
```

[06] Lineare Gleichungssysteme (lingl)

Gegeben ist ein lineares Gleichungssystem mit höchstens 10 Variablen. Zur Lösung des Systems wird die Methode der schrittweisen Elimination verwendet. Das Programm gliedert sich grundsätzlich in drei Teile:

(1) Eingabe

Die Koeffizienten der Gleichungen werden zunächst, durch Kommas getrennt, als Textzeilen eines RichEdit-Feldes eingegeben. Diese Textzeilen müssen nun mit der Maus markiert werden. In dem markierten Bereich ersetzt das Unterprogramm ***ReplaceAllString*** alle Codes für einen Zeilensprung (*EOL* = #13#10) durch den Code für ein Komma. Dadurch liegen alle Koeffizienten des Systems, hintereinander durch Kommas getrennt, in einem String. Durch die Anzahl *Anz* der Zeilensprünge ist die Anzahl der Gleichungen und somit auch die Zahl der Variablen automatisch bestimmt. Nun werden die Koeffizienten der Gleichungen mit dem Unterprogramm ***ExtractValues*** aus dem String extrahiert und in ein eindimensionales Array von reellen Zahlen *B[i]* kopiert (Datentyp *TKoeff*). Daraus werden sie in ein zweidimensionales *(Anz+1) x Anz*-Array von reellen Zahlen *A[i,j]* kopiert (Datentyp *TMatrix*), welches der eigentlichen Koeffizientenmatrix entspricht. Dabei sind *i* der Spaltenindex von *1* bis (*Anz+1*) und *j* der Zeilenindex von *1* bis *Anz*.

(2) Elimination und Berechnung

Die einzelnen Variablen werden mit Hilfe entsprechender Äquivalenzumformungen schrittweise aus den Gleichungen eliminiert, bis die Systemmatrix auf Halbdiagonalform gebracht ist. Beginnend mit der letzten Gleichung werden dann die einzelnen Lösungen bestimmt und in dem eindimensionalen Array *X[i]* (Datentyp *TLoes*) gespeichert. Dies geschieht in der Routine ***Linglei***. Ist das Gleichungssystem nicht lösbar, dann wird die Fehlervariable *Error* auf *False* gesetzt.

(3) Ausgabe

Zuletzt werden die berechneten Lösungen *X[i]* im RichEdit-Feld ausgegeben und sogar eine Probe durchgeführt, indem die berechneten linken Gleichungsseiten $\sum A[i,j]*X[i]$ den entsprechenden rechten Koeffizienten *A[Anz+1,j]* gegenübergestellt werden ($1 \le i \le Anz$ und $1 \le j \le Anz$).

Ein Beispiel mit *Anz* = 3 soll diesen Algorithmus demonstrieren.

```
a11*x1 + a21*x2 + a31*x3 = a41                    (I)
a12*x1 + a22*x2 + a32*x3 = a42                   (II)
a13*x1 + a23*x2 + a33*x3 = a43                  (III)
```

Beispiel:

```
2*x1 + 6*x2 - 2*x3 = 8          (I)
3*x1 - 9*x2 + 3*x3 = 6         (II)
4*x1 - 4*x2 - 2*x3 = 4        (III)
```

Elimination:

```
1.Schritt:   Elimination von x1 aus (II):
             (I) * 3/2 bzw. (I) * a12/a11
             und dann (II-I).

             Elimination von x1 aus (III):
             (I) * 4/2 bzw. (I) * a13/a11
             und dann (III-I).

             Das ergibt folgendes vereinfachte System:

             2*x1 + 6*x2 - 2*x3 =   8       (I)
                  -18*x2 + 6*x3 =  -6      (II)
                  -16*x2 + 2*x3 = -12     (III)

2. Schritt:  Elimination von x2 aus (III):
             (II) * (-16)/(-18) bzw. (II) * a23/a22
             und dann (III-II).

             Das ergibt folgendes vereinfachte System:

             2*x1 + 6*x2 - 2*x3 =   8       (I)
                  -18*x2 + 6*x3 =  -6      (II)
                         -10*x3 = -20     (III)
```

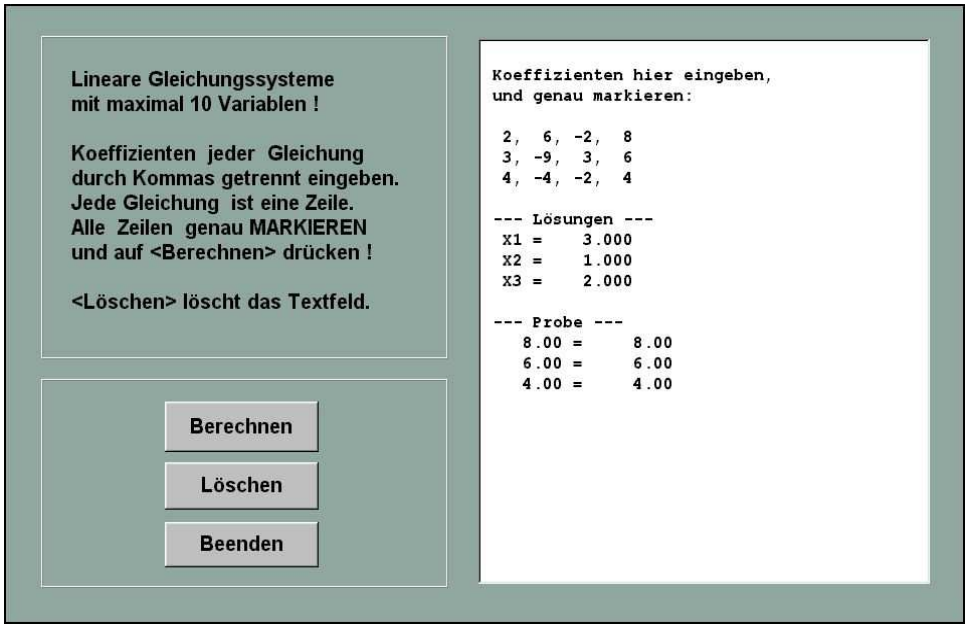
Berechnung:

```
1. Schritt:  Berechnung von x3 aus (III)
             x3 = (-20)/(-10) = 2
             bzw. x3 = a43/a33

2. Schritt:  Einsetzen von x3 in (II) und Berechnung von x2
             x2 = (-6-6*2)/(-18) = 1
             bzw. x2 = (a42-a32*x3)/a22

3. Schritt:  Einsetzen von x2,x3 in (I) und Berechnung von x1
             x1 = (8-(6*1-2*2))/2 = 3
             bzw. x1 = (a41-(a21*x2+a31*x3))/a11
```

Ergebnis: `x1 = 3,  x2 = 1,  x3 = 2`



unit lingl_u;

```
// Lingl, Lineare Gleichungssysteme lösen (c) H.Paukert

interface

uses Windows, Messages, SysUtils, Classes, Graphics,
     Controls, Forms, Dialogs, StdCtrls, ExtCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    RichEdit1: TRichEdit;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    private { Private declarations }
    public  { Public declarations }
  end;

var Form1: TForm1;

implementation
{$R *.DFM}

const EOL = #13#10;                          //  Code für Zeilensprung
      SEP = ',';                             //  Separatorzeichen (Komma)
type  TKoeff  = array[0..110] OF Real;       //  Koeffizienten-Vektor
      TMatrix = array[1..11,1..10] OF Real;  //  Gleichungs-Matrix (11 x 10)
      TLoes   = array[1..10] OF Real;        //  Lösungs-Vektor
var   B : TKoeff;                            //  Koeffizienten-Vektor
      A : TMatrix;                           //  Gleichungs-Matrix
      C : TMatrix;                           //  Kopie der Matrix
      X : TLoes;                             //  Lösungs-Vektor
      ANZ  : Integer;                        //  Anzahl der Unbekannten
      ERROR: Boolean;                        //  Globale Fehlervariable

procedure FitForm(F :TForm);
// Anpassung des Formulares an die Monitorauflösung
begin
  with F do begin
    if (Screen.Width<>1024) then ScaleBy(Screen.Width,1024);
    if (Font.PixelsPerInch<>120) then ScaleBy(120,Font.PixelsPerInch);
    WindowState := wsMaximized;
  end;
end;
```

```
function RemoveAllBlank(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 ReplaceAllString(var S: String; A,B: String): Integer;
// Ersetzt im String S den Teilstring A durch den String B.
// Zurückgeliefert wird die Anzahl der Ersetzungen.
Const Dummy = #1;
var   S1,A1 : String;
      L,P,N : Integer;
begin
  N := 0;
  S1 := UpperCase(S);
  A1 := UpperCase(A);
  L := Length(A1);
  repeat
     P := Pos(A1,S1);
     if P > 0 then begin
        N := N + 1;
        Delete(S1,P,L); Insert(Dummy,S1,P);
        Delete(S,P,L);  Insert(Dummy,S,P);
     end;
  until (P = 0);
  repeat
     P := Pos(Dummy,S);
     if P > 0 then begin
        Delete(S,P,1); Insert(B,S,P);
     end;
  until (P = 0);
  Result := N;
end;

procedure ExtractValues(S,SEP: String; var ZF: TKoeff);
// Durch den Separator getrennte Zahlenwerte aus einem String S extrahieren
// und in das Array ZF speichern. Die Anzahl der Zahlen 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;
  N := 0; Error := False;
  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 LINGLEI(var A: TMatrix; var X: TLoes; N:Integer);
// Lösung eines linearen Gleichungsystems in N Variablen
// A = Koeffizienten-Matrix, X = Lösungs-Vektor
var I,J,K,M: Integer;
          S: Real;
begin
  ERROR := False;

  { AUFSUCHEN DES GRÖSSTEN ANFANGS-KOEFFIZIENTEN }
  for I:= 1 to N-1 do begin
      M:= I;
      for K:= I+1 to N do
          if Abs(A[K,I]) > Abs(A[M,I]) then M:= K;

  { DIE I-TE ZEILE MIT DER M-TEN ZEILE TAUSCHEN }
      if I<>M then for J:= I to N+1 do begin
                       S:= A[I,J];
                       A[I,J]:= A[M,J];
                       A[M,J]:= S;
                   end;
```

```
  { ELIMINATION VON X[I] AUS DER K-TEN GLEICHUNG }
      for K:= I+1 to N do begin
          if A[I,I] = 0 then begin ERROR := True; Exit; end;
          S:= A[K,I]/A[I,I];
          for J:= I+1 to N+1 do A[K,J]:= A[K,J]-A[I,J]*S
      end;
   end;

  { EIGENTLICHE BERECHNUNG DER LÖSUNGEN }
   for I:= N DownTo 1 do begin
       S:= A[I,N+1];
       for J:= I+1 to N do S:= S-A[I,J]*X[J];
       if A[I,I]=0 then begin ERROR := True; Exit; end;
       X[I]:= S/A[I,I];
   end;
end;


procedure TForm1.FormActivate(Sender: TObject);
// Initialisierungen
begin
  FitForm(Form1);
  Color := RGB(150,170,160);
  Label1.Caption := 'Lineare Gleichungssysteme'+#13+
                    'mit maximal 10 Variablen!'+#13#13+
                    'Koeffizienten jeder Gleichung'+#13+
                    'durch Kommas getrennt eingeben.'+#13+
                    'Jede Gleichung ist eine Zeile.'+#13+
                    'Alle Zeilen genau MARKIEREN'+#13+
                    'und auf <Berechnen> drücken!'+#13#13+
                    '<Löschen> löscht das Textfeld.';
end;


procedure TForm1.Button1Click(Sender: TObject);
// Koeffizienten eingeben, Lösungen berechnen und ausgeben
var EIN,AUS : String;
    Z : Real;
    n,i,j,k : Integer;
begin
  EIN := RichEdit1.SelText;
  EIN := RemoveAllBlank(EIN);
  if EIN = '' then Exit;

  ANZ := ReplaceAllString(EIN,EOL,SEP);          //  Zeilensprung => Komma
  ExtractValues(EIN,SEP,B);                      //  Koeffizienten ermitteln
  n := Trunc(B[0]);
  if (ANZ > 10) or (n <> ANZ*(ANZ+1)) then begin
    RichEdit1.Lines.Add(EOL + ' EINGABE fehlerhaft!');
    Exit;
  end;

  k := 0;                                        //  Matrix erzeugen
  for i := 1 to ANZ do begin
      for j := 1 to ANZ + 1 do begin
          k := k + 1;
          A[i,j] := B[k];
      end;
  end;
  C := A;                                        //  Matrix-Kopie bilden

  RichEdit1.Lines.Add(EOL + ' --- Lösungen --- ');

  LINGLEI(A,X,ANZ);                              //  Gleichungssystem lösen

  if ERROR then begin
    RichEdit1.Lines.Add(EOL + ' SYSTEM unlösbar!');
    Exit;
  end;

  for j := 1 to ANZ do begin                     //  Lösungen ausgeben
    EIN := '  X' + IntToStr(j) + ' = ';
    str(X[j]:8:3,AUS);
    AUS := EIN + AUS;
    RichEdit1.Lines.Add(AUS);
  end;
```

```
  RichEdit1.Lines.Add(EOL + ' --- Probe --- ');  // Probe durchführen
  for i := 1 to ANZ do begin
     Z := 0;
     for j := 1 to ANZ do Z := Z + X[j]*C[i,j];
     Str(Z:8:2,AUS); Str(C[i,ANZ+1]:8:2,EIN);
     AUS := AUS + ' = ' + EIN;
     RichEdit1.Lines.Add(AUS);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
// Textfeld löschen und Cursor platzieren
begin
   RichEdit1.Clear;
   ichEdit1.Lines.Add(EOL + ' Koeffizienten hier eingeben' + EOL);
   RichEdit1.SetFocus;
end;

procedure TForm1.Button3Click(Sender: TObject);
{ Programm beenden }
begin
  Application.Terminate;
end;

end.
```

[07] Erzeugung von Primzahlen (*prim*)

Eine Primzahl ist eine natürliche Zahl Z, die außer sich selbst und 1 keine Teiler T besitzt. Unsere Aufgabe ist es nun, in aufsteigender Folge alle Primzahlen bis zu einer vorgegebenen Grenzzahl (GRENZE) zu finden und sie außerdem zu zählen (Zähler N).

Ausgenommen 2 müssen alle Primzahlen ungerade sein. Man wird daher in einer äußeren Schleife alle ungeraden Zahlen Z von 3 bis GRENZE durchlaufen und bei jeder einen Teilertest durchführen. Ist Z unteilbar, dann handelt es sich um eine Primzahl.

Der Teilertest einer ungeraden Zahl Z besteht einfach darin, dass man in einer inneren Schleife alle ungeraden Zahlen T von 3 bis W durchläuft und überprüft, ob T ein Teiler von Z ist. T teilt Z, wenn (Z - int(Z/T)*T) = 0 bzw. (Z mod T) = 0 gilt. Hat man dabei die Wurzel W von Z erreicht und keinen Teiler gefunden, dann muss Z eine Primzahl sein. Diese Überprüfung der Teilbarkeit von der Zahl Z erfolgt in der Routine ***Teilbar(Z).***

Die gefundenen Primzahlen Z werden in der Routine ***Ausgabe(Z)*** in einem RichEdit-Feld nebeneinander ausgegeben. Außerdem kann das Programm mit der Taste <Esc> abgebrochen werden.

Beispiel: Teilertest von 67
W = 8.2
(2,3,5,7) teilen 67 nicht ==> 67 ist Primzahl

```
function Teilbar(z: Integer): Boolean;
// Teilbarkeitstest einer ungeraden Zahl z > 3
var t,r : Integer;
    w   : Real;
begin
  w := Sqrt(z);
  t := 1;
  repeat
    t := t + 2;
    r := z mod t;
  until (r = 0) or (t > w);
  if r = 0 then Result := True else Result := False;
end;
```

```
procedure Ausgabe(z: Integer);
// Formatierte Zahlenausgabe
const Dez = 7;
      Num = 10;
var   s,t : String;
      zei : Integer;
      len : Integer;
begin
  with Form1.Richedit1 do begin
    Str(z:Dez,t);
    zei := Lines.Count;
    s   := Lines[zei-1];
    len := Length(s);
    if len < dez * num then begin
       s := s + t;
       Lines[zei-1] := s;
    end
    else Lines.Add(t);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
// Primzahlen erzeugen
var Grenze, Anzahl: Integer;
    s: String;
    z, Code: Integer;
    BreakFlag: Boolean;
begin
  s := Edit1.Text;
  Val(S,Grenze,Code);
  if (Code <> 0) or (Grenze < 3) or (Grenze > 1000000) then begin
     ShowMessage('Eingabefehler!');
     Edit1.Clear;
     Edit1.SetFocus;
     Exit;
  end;
  BreakFlag := False;
  Label2.Caption := '';
  RichEdit1.Clear;
  RichEdit1.Lines.Add('');
  RichEdit1.Lines.Add('');
  z := 2; Ausgabe(z);
  z := 3; Ausgabe(z);
  Anzahl := 2;
  repeat
    if not teilbar(z) then begin
       Ausgabe(z);
       Anzahl := Anzahl + 1;
    end;
    if (GetAsyncKeyState(vk_escape) <> 0) then begin
       BreakFlag := True;
       Break;
    end;
    z := z + 2;
  until z > Grenze;
  if BreakFlag then s := 'abgebrochen!' else  s := IntToStr(Anzahl);
  Label2.Caption := 'Primzahlen: ' + s;
end;
```

[08] Primfaktorenzerlegung einer Zahl (*pfak*)

Ein Primfaktor T einer gegebenen Zahl Z ist ein Teiler der Zahl, der selbst unteilbar, also genau eine Primzahl ist.

Die Prozedur ***Teile(Z,T)*** überprüft, ob T ein Teiler von Z ist. Wenn ja, dann wird T als Primfaktor ausgewiesen und als neue Zahl Z der Quotient von Z durch T genommen. Das geht so lange, bis T nicht mehr als Faktor in Z vorkommt.

Die Prozedur ***Teile(Z,T)*** wird zunächst mit T = 2 durchgeführt und dann mit allen ungeraden Zahlen bis zur Quadratwurzel W von Z. Am Ende dieser Schleife sind alle Primfaktoren von Z erfasst worden.

```
var Erg: String;    // globale Variable für das Ergebnis

procedure Teile(var z: Integer; t: Integer);
// Teiler t von z prüfen
begin
  while ((z mod t) = 0) and (z > 1) do begin
    z  := z div t;
    Erg := Erg + ' * ' + IntToStr(t);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
// Primfaktoren berechnen
var s : String;
    z,t,Code: Integer;
    w : Real;
begin
  Erg := '1';
  s := Edit1.Text;
  Val(s,z,Code);
  if (Code <> 0) or (z < 0) or (z > 1000000) then begin
     ShowMessage('Eingabefehler!');
     Edit1.Clear;
     Edit1.SetFocus;
     Exit;
  end;
  w := Sqrt(z);
  t := 2; Teile(z,t);
  t := 3;
  repeat
    Teile(z,t);
    t := t + 2;
  until t > w;
  Erg := Erg + ' * ' + IntToStr(z);
  Edit2.Text := Erg;
end;
```

[09] Verschiedene Zahlensysteme (*zsys*)

Die Funktion ***DezToSys(X,B)*** wandelt eine ganze Zahl X aus dem Zehner-System in eine Zahl aus dem Zahlensystem mit der Basis B um. Die konvertierte Zahl wird als Zeichenkette geliefert. Die Basis B muss zwischen 1 und 37 liegen.

Die Funktion ***SysToDez(S,B)*** wandelt eine als Zeichenkette S eingegebene Zahl aus dem Zahlensystem mit der Basis B in eine Zahl aus dem Zehner-System um. Die rekonvertierte Zahl ist dann vom Typ *Integer*. Die Basis B muss zwischen 1 und 37 liegen.

```
function DezToSys(X,B: Integer): String;
// Konvertierung der Zahl X von Dez zu Sys mit Basis B (2 bis 36)
var S: String;
    R: Integer;
begin
  if (B<2) or (B>36) then begin Result := '0'; Exit; end;
  S := '';
  repeat
    R := X mod B;
    if R < 10 then S := Chr(R+48) + S
              else S := Chr(R+55) + S;
    X := X - R;
    X := X div B;
  until X < B;
```

```
  if X < 10 then S := Chr(X+48) + S
            else S := Chr(X+55) + S;
  Result := S;
end;

function SysToDez(S: String; B: Integer): Integer;
// Rekonvertierung der Zahl S von Sys zu Dez mit Basis B (2 bis 36)
var I,K,Z : Integer;
begin
  if (B<2) or (B>36) then begin Result := 0; Exit; end;
  Z := 0;
  for I := 1 to Length(S) do begin
    K := Ord(S[I]);
    if (K>=0)  and (K<=57)  then K := K - 48;
    if (K>=65) and (K<=90)  then K := K - 55;
    if (K>=97) and (K<=122) then K := K - 87;
    if (K>=0)  and (K<B) then Z := Z*B + K
                         else begin Result := 0; Exit; end;
  end;
  Result := Z;
end;
```

[10] Näherungsweise Berechnung der Zahl "PI" (*pizahl*)

Gegeben sei ein Kreis mit dem Radius r = 1 (Einheitskreis). Diesem wird ein regelmäßiges Vieleck mit der Seitenanzahl N und der Seitenlänge a1 eingeschrieben. Den Umfang dieses N-Ecks erhält man mit der Formel *U := N * a1*. Dividiert man den Umfang des Vielecks durch den Kreisdurchmessers (2 im Einheitskreis), so erhält man einen Näherungswert *p1 := N * a1 / 2* für die Zahl PI.

Aus der Seite a1 des regelmäßigen N-Ecks im Einheitskreis kann mit Hilfe der pythagoräischen Lehrsätze die Seite a2 des regelmäßigen 2N-Ecks berechnet werden. Als Ergebnis dieser Berechnungen erhält man folgende Formel:

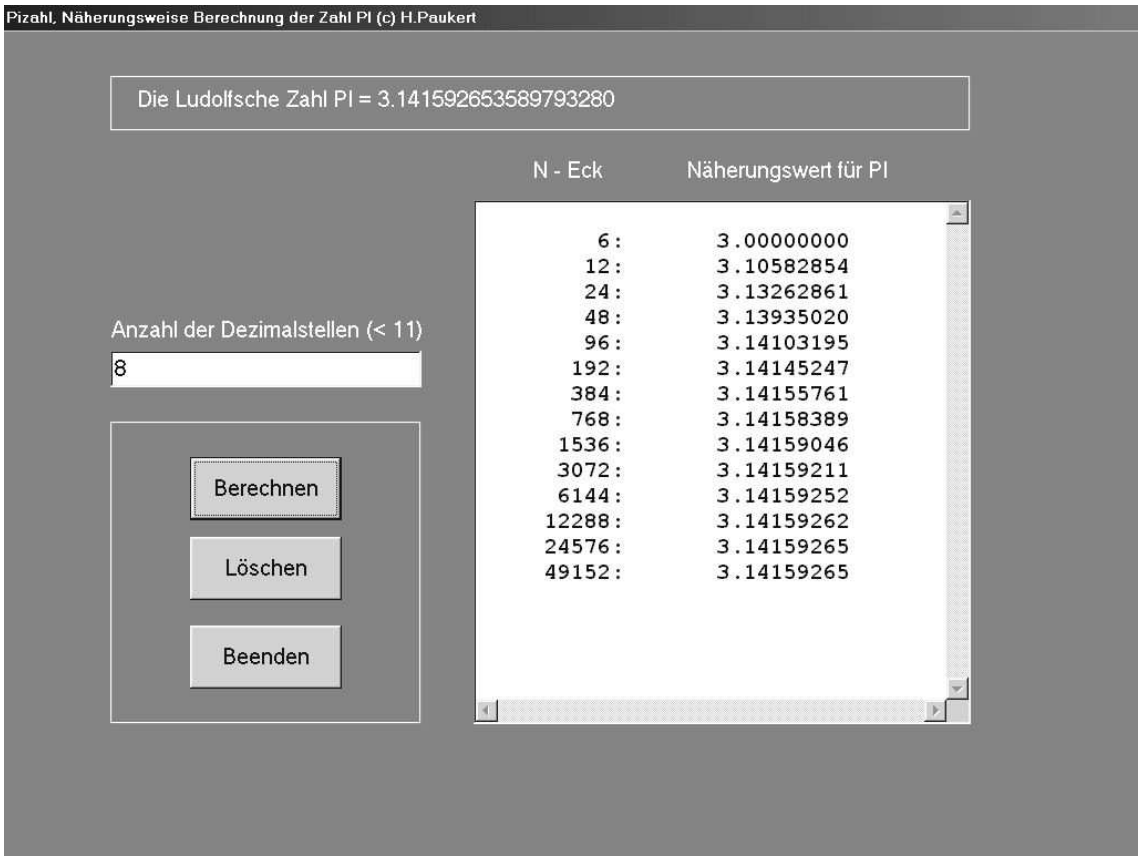
$$a2 := \sqrt{2 - \sqrt{4 - a1^2}}$$

Das Programm "***pizahl***" beginnt mit dem regelmäßigen Sechseck (N := 6, a1 := 1, p1 := 3) im Einheitskreis. In einer Wiederholungsschleife berechnet man schrittweise die Seite a2 des regelmäßigen 2N-Ecks und daraus den Näherungswert für die Zahl PI. Die Schleife ist dann zu Ende, wenn der Unterschied (diff) zwischen dem neuen Näherungswert p2 und dem alten Näherungswert p1 kleiner als eine vorgegebene Genauigkeit GEN ist. Diese Genauigkeit kann vorher mit 1 bis 10 Dezimalstellen festgesetzt werden. Sie wird in einem Edit-Feld eingegeben. Die Ausgabe der Werte für PI erfolgt in einem Memo-Feld.

```
procedure TForm1.Button1Click(Sender: TObject);
// Berechnen
var Code  : Integer;                    // Fehlervariable
    Dezi  : Integer;                    // Anzahl der Dezimalstellen
    N     : Integer;                    // Anzahl der Eckpunkte
    a1,p1 : Extended;                   // alte Werte für Seiten und PI
    a2,p2 : Extended;                   // neue Werte für Seiten und PI
    Gen   : Extended;                   // gegebene Genauigkeit
    Diff  : Extended;                   // aktuelle Genauigkeit
    S,T   : String;                     // Hilfsvariable
begin
   S  := Edit1.Text;                    // Eingabe der Dezimalstellen
   Val(S,Dezi,Code);
   if (Code <> 0) or ( Dezi < 1) or (Dezi > 10) then begin
      Edit1.Clear;
      Edit1.SetFocus;
      Exit;
   end;
```

```
   Memo1.Clear;
   Memo1.Lines.Add(' ');
   Gen:= Power(10,-Dezi);                  // Rechengenauigkeit
   N  := 6;                                // Rekursionsanfang
   a1 := 1;
   p1 := N * a1 / 2;
   str(N:10,S);
   str(p1:12:Dezi,T);
   S := S + ':     ' + T;
   Memo1.Lines.Add(S);
   repeat                                  // Wiederholungsschleife
      N  := 2 * N;
      a2 := Sqrt(2-sqrt(4-a1*a1));         // Rekursionsformel
      p2 := N * a2 / 2;
      str(N:10,S);
      str(p2:12:Dezi,T);
      S := S + ':     ' + T;
      Memo1.Lines.Add(S);
      Diff := Abs(p2-p1);
      a1 := a2;                            // Umspeichern der neuen Werte
      p1 := p2;                            // auf die alten Werte
    until Diff < Gen;
end;
```

Die unten stehende Abbildung zeigt das Formular von dem Programm "***pizahl***".



[11] Anhang: Der Mathematikparser "*mparse_u.pas*".

```
unit mparse_u;

{
  Mathematik-Parser (c) Herbert Paukert.
  Dieser PARSER erlaubt die Analyse einer Textzeile nach allen wichtigen
  Funktionen mit maximal 26 reellen Variablen, die global mit A_, ... ,Z_
  bezeichnet sind. Nicht zulässige Argumentbereiche werden erkannt.
  Es wird die mathematische Formel mit den Variablenwerten berechnet und
  der Formelwert zurück geliefert. Etwaige Fehler bei der Formelauswertung
  stehen in der globalen Variablen ATT.

  Fehlercodes: 0 = TEXT; 1 = KONSTANTE; 2,3 = FORMEL; 4,5,6 = FEHLER

  OPERATOREN ( 7):  +, -, *, /, ^, (, )
  FUNKTIONEN (18):  ABS, ACOS, ASIN, ATAN, COS, DEG, EXP, FAK, LOG, LN,
                    RAD, ROUND, SIN, SQR, SQRT, TAN, TRUNC, PI

  Die Schreibweise der Formel folgt der normalen algebraischen Notation
  und es sind bis zu 30 geschachtelte Klammerebenen möglich.
}

INTERFACE
{ ENTHÄLT EINE GLOBALE FUNKTION, 26 GLOBALE ARGUMENT-VARIABLE UND EINE
  FEHLERVARIABLE ATT }

CONST NULGEN = 1E-12;       { NULL-WERTE-GENAUIGKEIT }
      INTLIM = 1E18;        { INTEGER-ZAHLEN-LIMIT   }
      DEZLIM = 1E300;       { REAL-ZAHLEN-LIMIT      }
VAR   ATT : INTEGER;
      A_,B_,C_,D_,E_,F_,G_,H_,I_,J_,K_,L_,M_,
      N_,O_,P_,Q_,R_,S_,T_,U_,V_,W_,X_,Y_,Z_: Real;

FUNCTION PARSE(S: STRING; VAR ATT: INTEGER): REAL;
{ ANALYSIERT DEN STRING S - LIEFERT DEN WERT DES ANALYSIERTEN STRINGS UND
  DES ATTRIBUTES ATT: TXT = 0, CONSTANT = 1, FORMULA = 2,3, ERROR = 4,5,6 }

IMPLEMENTATION

CONST  TXT     = 0;
       VALUE   = 1;
       FORMULA = 2;

CONST
  EXPLIMIT = 450;
  SQRLIMIT = 1E150;
  MAXEXPLEN= 3;
  MAXINPUT = 159;
  PARSERSTACKSIZE = 30;
  MSGSTACKERROR = ' STACK-ÜBERLAUF BEI DER FORMELAUSWERTUNG ! ';
  LETTERS: SET OF CHAR = ['a'..'z','A'..'Z'];

CONST
  PLUS = 0;
  MINUS = 1;
  TIMES = 2;
  DIVIDE = 3;
  EXPO = 4;
  COLON = 5;
  OPAREN = 6;
  CPAREN = 7;
  NUM = 8;
  VART = 9;
  FUNC = 10;
  EOL = 11;
  BAD = 12;
  MAXFUNCNAMELEN = 5;
  VARNUM : WORD = 0;

TYPE
  TOKENREC = RECORD
    STATE : BYTE;
    CASE BYTE OF
      0 : (VALUE : REAL);
      1 : (VARNUM: WORD);
      2 : (FUNCNAME : STRING[MAXFUNCNAMELEN]);
  END;
  ISTRING = STRING[MAXINPUT];
```

```
VAR
  STACK : ARRAY [1..PARSERSTACKSIZE] OF TOKENREC;
  CURTOKEN : TOKENREC;
  STACKTOP, TOKENTYPE : WORD;
  MATHERROR, TOKENERROR, ISFORMULA : BOOLEAN;
  INPUT : ISTRING;
  HELP  : REAL;

FUNCTION BLANKOFF(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  SUBSTRING(VAR S: STRING; A,B: STRING): INTEGER;
{ SUCHT UND ERSETZT EINEN TEILSTRING, ÜBERGIBT DIE ANZAHL }
CONST C = #1;
VAR   L,P,N : INTEGER;
BEGIN
   L := LENGTH(A);
   P := 1; N := 0;
   WHILE P<>0 DO
     BEGIN
     P := POS(A,S);
     IF P>0 THEN
        BEGIN
          INC(N);
          DELETE(S,P,L);
          INSERT(C,S,P);
        END;
     END;
   P := 1;
   WHILE P<>0 DO
     BEGIN
     P := POS(C,S);
     IF P>0 THEN
        BEGIN
          DELETE(S,P,1);
          INSERT(B,S,P);
        END;
     END;
   SUBSTRING := N;
END;

FUNCTION UPPERCASE(S : STRING) : STRING;
CONST PI : STRING = '3.14159265358979';
VAR COUNTER : WORD;
BEGIN
  FOR COUNTER := 1 TO LENGTH(S) DO
      S[COUNTER] := UPCASE(S[COUNTER]);
  COUNTER := SUBSTRING(S,'PI',PI);
  UPPERCASE := S;
END;

FUNCTION ISFUNC(S : STRING) : BOOLEAN;
{ PRÜFT, OB DER BEGINN DES INPUT-STRINGS EINE ZULŽSSIGE FUNKTION IST. }
VAR
  LEN : WORD;
BEGIN
  LEN := LENGTH(S);
  IF POS(S, INPUT) = 1 THEN
  BEGIN
    CURTOKEN.FUNCNAME := COPY(INPUT, 1, LEN);
    DELETE(INPUT, 1, LEN);
    ISFUNC := TRUE;
  END
  ELSE
    ISFUNC := FALSE;
END;

FUNCTION ISVAR(S : STRING; VAR VARNUM: WORD): BOOLEAN;
{ PRÜFT, OB DER BEGINN DES INPUT-STRINGS EINE ZULŽSSIGE VARIABLE (X,Y,Z) IST}
BEGIN
  S := UPPERCASE(S);
  IF POS(S,INPUT)=1 THEN
     BEGIN
       ISVAR := TRUE;
       IF (S='A') THEN VARNUM := 1;
       IF (S='B') THEN VARNUM := 2;
       IF (S='C') THEN VARNUM := 3;
       IF (S='D') THEN VARNUM := 4;
       IF (S='E') THEN VARNUM := 5;
```

```
       IF (S='F') THEN VARNUM := 6;
       IF (S='G') THEN VARNUM := 7;
       IF (S='H') THEN VARNUM := 8;
       IF (S='I') THEN VARNUM := 9;
       IF (S='J') THEN VARNUM := 10;
       IF (S='K') THEN VARNUM := 11;
       IF (S='L') THEN VARNUM := 12;
       IF (S='M') THEN VARNUM := 13;
       IF (S='N') THEN VARNUM := 14;
       IF (S='O') THEN VARNUM := 15;
       IF (S='P') THEN VARNUM := 16;
       IF (S='Q') THEN VARNUM := 17;
       IF (S='R') THEN VARNUM := 18;
       IF (S='S') THEN VARNUM := 19;
       IF (S='T') THEN VARNUM := 20;
       IF (S='U') THEN VARNUM := 21;
       IF (S='V') THEN VARNUM := 22;
       IF (S='W') THEN VARNUM := 23;
       IF (S='X') THEN VARNUM := 24;
       IF (S='Y') THEN VARNUM := 25;
       IF (S='Z') THEN VARNUM := 26;
     END
     ELSE
       ISVAR := FALSE;
END;

FUNCTION NEXTTOKEN : WORD;
{ HOLT DEN NÄCHSTEN TOKEN VOM INPUT-STRING }
VAR
  NUMSTRING : STRING[159];
  LEN, NUMLEN, CHECK : INTEGER;
  DECIMAL : BOOLEAN;
BEGIN
  IF INPUT = '' THEN
  BEGIN
    NEXTTOKEN := EOL;
    EXIT;
  END;
  WHILE (INPUT <> '') AND (INPUT[1] = ' ') DO
    DELETE(INPUT, 1, 1);
  IF INPUT[1] IN ['0'..'9', '.'] THEN
  BEGIN
    NUMSTRING := '';
    LEN := 1;
    DECIMAL := FALSE;
    WHILE (LEN <= LENGTH(INPUT)) AND
          ((INPUT[LEN] IN ['0'..'9']) OR
           ((INPUT[LEN] = '.') AND (NOT DECIMAL))) DO
    BEGIN
      NUMSTRING := NUMSTRING + INPUT[LEN];
      IF INPUT[1] = '.' THEN
        DECIMAL := TRUE;
      INC(LEN);
    END;
    IF (LEN = 2) AND (INPUT[1] = '.') THEN
    BEGIN
      NEXTTOKEN := BAD;
      EXIT;
    END;

    IF (LEN <= LENGTH(INPUT)) AND (INPUT[LEN] = 'E') THEN
    BEGIN
      NUMSTRING := NUMSTRING + 'E';
      INC(LEN);
      IF INPUT[LEN] IN ['+', '-'] THEN
      BEGIN
        NUMSTRING := NUMSTRING + INPUT[LEN];
        INC(LEN);
      END;
      NUMLEN := 1;
      WHILE (LEN <= LENGTH(INPUT)) AND (INPUT[LEN] IN ['0'..'9']) AND
            (NUMLEN <= MAXEXPLEN) DO
      BEGIN
        NUMSTRING := NUMSTRING + INPUT[LEN];
        INC(NUMLEN);
        INC(LEN);
      END;
    END;
    IF NUMSTRING[1] = '.' THEN
      NUMSTRING := '0' + NUMSTRING;
    VAL(NUMSTRING, CURTOKEN.VALUE, CHECK);
    IF CHECK <> 0 THEN
      MATHERROR := TRUE;
```

```
    NEXTTOKEN := NUM;
    DELETE(INPUT, 1, LENGTH(NUMSTRING));
    EXIT;
  END
  ELSE IF INPUT[1] IN LETTERS THEN
  BEGIN
    IF ISFUNC('DEG') OR
       ISFUNC('RAD') OR
       ISFUNC('TAN') OR
       ISFUNC('ASIN') OR
       ISFUNC('ACOS') OR
       ISFUNC('LOG') OR
       ISFUNC('FAK') OR
       ISFUNC('ABS') OR
       ISFUNC('ATAN') OR
       ISFUNC('SWI') OR
       ISFUNC('COS') OR
       ISFUNC('EXP') OR
       ISFUNC('LN') OR
       ISFUNC('ROUND') OR
       ISFUNC('SIN') OR
       ISFUNC('SQRT') OR
       ISFUNC('SQR') OR
       ISFUNC('TRUNC') THEN
    BEGIN
      NEXTTOKEN := FUNC;
      EXIT;
    END;
    IF ISVAR('A',CURTOKEN.VARNUM) OR
       ISVAR('B',CURTOKEN.VARNUM) OR
       ISVAR('C',CURTOKEN.VARNUM) OR
       ISVAR('D',CURTOKEN.VARNUM) OR
       ISVAR('E',CURTOKEN.VARNUM) OR
       ISVAR('F',CURTOKEN.VARNUM) OR
       ISVAR('G',CURTOKEN.VARNUM) OR
       ISVAR('H',CURTOKEN.VARNUM) OR
       ISVAR('I',CURTOKEN.VARNUM) OR
       ISVAR('J',CURTOKEN.VARNUM) OR
       ISVAR('K',CURTOKEN.VARNUM) OR
       ISVAR('L',CURTOKEN.VARNUM) OR
       ISVAR('M',CURTOKEN.VARNUM) OR
       ISVAR('N',CURTOKEN.VARNUM) OR
       ISVAR('O',CURTOKEN.VARNUM) OR
       ISVAR('P',CURTOKEN.VARNUM) OR
       ISVAR('Q',CURTOKEN.VARNUM) OR
       ISVAR('R',CURTOKEN.VARNUM) OR
       ISVAR('S',CURTOKEN.VARNUM) OR
       ISVAR('T',CURTOKEN.VARNUM) OR
       ISVAR('U',CURTOKEN.VARNUM) OR
       ISVAR('V',CURTOKEN.VARNUM) OR
       ISVAR('W',CURTOKEN.VARNUM) OR
       ISVAR('X',CURTOKEN.VARNUM) OR
       ISVAR('Y',CURTOKEN.VARNUM) OR
       ISVAR('Z',CURTOKEN.VARNUM) THEN
    BEGIN
      DELETE(INPUT, 1, 1);
      ISFORMULA := TRUE;
      NEXTTOKEN := VART;
      EXIT;
    END
    ELSE BEGIN
      NEXTTOKEN := BAD;
      EXIT;
    END;
  END
  ELSE BEGIN
    CASE INPUT[1] OF
      '+' : NEXTTOKEN := PLUS;
      '-' : NEXTTOKEN := MINUS;
      '*' : NEXTTOKEN := TIMES;
      '/' : NEXTTOKEN := DIVIDE;
      '^' : NEXTTOKEN := EXPO;
      ':' : NEXTTOKEN := COLON;
      '(' : NEXTTOKEN := OPAREN;
      ')' : NEXTTOKEN := CPAREN;
      ELSE
        NEXTTOKEN := BAD;
    END;
    DELETE(INPUT, 1, 1);
    EXIT;
  END; { CASE }
END;
```

```
PROCEDURE PUSH(TOKEN : TOKENREC);
{ STAPELT EINEN NEUEN TOKEN AUF DEN STACK }
BEGIN
  IF STACKTOP = PARSERSTACKSIZE THEN
  BEGIN
  WRITELN;
  WRITELN(MSGSTACKERROR);
  WRITELN;
  TOKENERROR := TRUE;
  END
  ELSE BEGIN
    INC(STACKTOP);
    STACK[STACKTOP] := TOKEN;
  END;
END;

PROCEDURE POP(VAR TOKEN : TOKENREC);
{ HOLT DEN TOP-TOKEN VOM STACK }
BEGIN
  TOKEN := STACK[STACKTOP];
  DEC(STACKTOP);
END;

FUNCTION GOTOSTATE(PRODUCTION : WORD) : WORD;
{ DEFINIERT DEN NEUEN STATUS BASIEREND AUF DER GERADE FERTIGEN PRODUKTION
  UND AUF DEM JEWEILIGEN TOP-STATUS }
VAR
  STATE : WORD;
BEGIN
  GOTOSTATE := 0;
  STATE := STACK[STACKTOP].STATE;
  IF (PRODUCTION <= 3) THEN
  BEGIN
    CASE STATE OF
      0 : GOTOSTATE := 1;
      9 : GOTOSTATE := 19;
      20 : GOTOSTATE := 28;
    END; { CASE }
  END
  ELSE IF PRODUCTION <= 6 THEN
  BEGIN
    CASE STATE OF
      0, 9, 20 : GOTOSTATE := 2;
      12 : GOTOSTATE := 21;
      13 : GOTOSTATE := 22;
    END; { CASE }
  END
  ELSE IF PRODUCTION <= 8 THEN
  BEGIN
    CASE STATE OF
      0, 9, 12, 13, 20 : GOTOSTATE := 3;
      14 : GOTOSTATE := 23;
      15 : GOTOSTATE := 24;
      16 : GOTOSTATE := 25;
    END; { CASE }
  END
  ELSE IF PRODUCTION <= 10 THEN
  BEGIN
    CASE STATE OF
      0, 9, 12..16, 20 : GOTOSTATE := 4;
    END; { CASE }
  END
  ELSE IF PRODUCTION <= 12 THEN
  BEGIN
    CASE STATE OF
      0, 9, 12..16, 20 : GOTOSTATE := 6;
      5 : GOTOSTATE := 17;
    END; { CASE }
  END
  ELSE BEGIN
    CASE STATE OF
      0, 5, 9, 12..16, 20 : GOTOSTATE := 8;
    END; { CASE }
  END;
END;

FUNCTION VARVALUE(VARNUM : WORD): REAL;
BEGIN
  CASE VARNUM OF
    1 : VARVALUE := A_;
    2 : VARVALUE := B_;
    3 : VARVALUE := C_;
    4 : VARVALUE := D_;
```

```
    5 : VARVALUE := E_;
    6 : VARVALUE := F_;
    7 : VARVALUE := G_;
    8 : VARVALUE := H_;
    9 : VARVALUE := I_;
   10 : VARVALUE := J_;
   11 : VARVALUE := K_;
   12 : VARVALUE := L_;
   13 : VARVALUE := M_;
   14 : VARVALUE := N_;
   15 : VARVALUE := O_;
   16 : VARVALUE := P_;
   17 : VARVALUE := Q_;
   18 : VARVALUE := R_;
   19 : VARVALUE := S_;
   20 : VARVALUE := T_;
   21 : VARVALUE := U_;
   22 : VARVALUE := V_;
   23 : VARVALUE := W_;
   24 : VARVALUE := X_;
   25 : VARVALUE := Y_;
   26 : VARVALUE := Z_;
  END;
END;

PROCEDURE SHIFT(STATE : WORD);
{ SHIFTET EIN TOKEN AUF DEN STACK }
BEGIN
  CURTOKEN.STATE := STATE;
  PUSH(CURTOKEN);
  TOKENTYPE := NEXTTOKEN;
END;

PROCEDURE REDUCE(REDUCTION : WORD);
{ KOMPLETTIERT EINE REDUKTION }
VAR
  TOKEN1, TOKEN2 : TOKENREC;
  TOKENSIGN, N : INTEGER;
  TEMP: INTEGER;
BEGIN
  CASE REDUCTION OF
    1 : BEGIN
      POP(TOKEN1);
      POP(TOKEN2);
      POP(TOKEN2);
      CURTOKEN.VALUE := TOKEN1.VALUE + TOKEN2.VALUE;
    END;
    2 : BEGIN
      POP(TOKEN1);
      POP(TOKEN2);
      POP(TOKEN2);
      CURTOKEN.VALUE := TOKEN2.VALUE - TOKEN1.VALUE;
    END;
    4 : BEGIN
      POP(TOKEN1);
      POP(TOKEN2);
      POP(TOKEN2);
      CURTOKEN.VALUE := TOKEN1.VALUE * TOKEN2.VALUE;
    END;
    5 : BEGIN
      POP(TOKEN1);
      POP(TOKEN2);
      POP(TOKEN2);
      IF TOKEN1.VALUE = 0 THEN
        MATHERROR := TRUE
      ELSE
        CURTOKEN.VALUE := TOKEN2.VALUE / TOKEN1.VALUE;
    END;
    7 : BEGIN
      TOKENSIGN := 1;
      POP(TOKEN1);
      POP(TOKEN2);
      POP(TOKEN2);
      IF TOKEN2.VALUE < 0 THEN BEGIN
         TOKEN2.VALUE := -TOKEN2.VALUE;
         TOKENSIGN := -1;
         END;
      IF (TOKENSIGN=-1) AND (FRAC(TOKEN1.VALUE)=0) THEN
         IF (ODD(TRUNC(TOKEN1.VALUE))) THEN TOKENSIGN:=-1
                                       ELSE TOKENSIGN:=1;
      IF (TOKENSIGN=-1) AND (FRAC(TOKEN1.VALUE)>0) THEN TOKENSIGN:=0;
      IF (TOKENSIGN=-1) AND (TOKEN1.VALUE < 0) AND
         (FRAC(TOKEN1.VALUE)>0) THEN TOKENSIGN:=0;
```

```
  IF (TOKEN2.VALUE = 0) THEN CURTOKEN.VALUE:=0;
  IF (TOKEN2.VALUE = 0) AND (TOKEN1.VALUE = 0) THEN CURTOKEN.VALUE := 1;
  IF (TOKEN2.VALUE = 0) AND (TOKEN1.VALUE < 0) THEN MATHERROR := TRUE;
  IF TOKEN2.VALUE <> 0 THEN
     IF (TOKEN1.VALUE * LN(ABS(TOKEN2.VALUE)) < -EXPLIMIT) OR
        (TOKEN1.VALUE * LN(ABS(TOKEN2.VALUE)) > EXPLIMIT) OR
        (TOKENSIGN=0) THEN
     MATHERROR := TRUE
     ELSE
     CURTOKEN.VALUE := TOKENSIGN*EXP(TOKEN1.VALUE * LN(TOKEN2.VALUE));
END;
9 : BEGIN
  POP(TOKEN1);
  POP(TOKEN2);
  CURTOKEN.VALUE := -TOKEN1.VALUE;
END;
11 : ;
13 : BEGIN
  POP(CURTOKEN);
  CURTOKEN.VALUE := VARVALUE(CURTOKEN.VARNUM);
END;
14 : BEGIN
  POP( TOKEN1);
  POP(CURTOKEN);
  POP(TOKEN1);
END;
16 : BEGIN
  POP(TOKEN1);
  POP(CURTOKEN);
  POP(TOKEN1);
  POP(TOKEN1);

  IF ABS(CURTOKEN.VALUE) < NULGEN THEN CURTOKEN.VALUE := 0;

  IF TOKEN1.FUNCNAME = 'DEG' THEN
     CURTOKEN.VALUE := CURTOKEN.VALUE*180/PI

  ELSE IF TOKEN1.FUNCNAME = 'RAD' THEN
     CURTOKEN.VALUE := CURTOKEN.VALUE*PI/180

  ELSE IF TOKEN1.FUNCNAME = 'TAN' THEN
    IF ABS(COS(CURTOKEN.VALUE*PI/180)) < NULGEN THEN MATHERROR := TRUE
    ELSE BEGIN
      CURTOKEN.VALUE := CURTOKEN.VALUE * PI / 180;
      CURTOKEN.VALUE := SIN(CURTOKEN.VALUE)/COS(CURTOKEN.VALUE);
    END
  ELSE IF TOKEN1.FUNCNAME = 'ACOS' THEN
    IF (ABS(CURTOKEN.VALUE)>1) THEN MATHERROR := TRUE
    ELSE BEGIN
      IF (ABS(CURTOKEN.VALUE)<1) and (CURTOKEN.VALUE<>0) THEN BEGIN
         CURTOKEN.VALUE := ARCTAN(SQRT(1-SQR(CURTOKEN.VALUE))/CURTOKEN.VALUE);
         CURTOKEN.VALUE := CURTOKEN.VALUE * 180 / PI;
         IF CURTOKEN.VALUE < 0 THEN CURTOKEN.VALUE := 180 + CURTOKEN.VALUE;
      END;
      IF CURTOKEN.VALUE = 0 THEN CURTOKEN.VALUE :=  90;
      IF CURTOKEN.VALUE = 1 THEN CURTOKEN.VALUE :=   0;
      IF CURTOKEN.VALUE =-1 THEN CURTOKEN.VALUE := 180;
    END

  ELSE IF TOKEN1.FUNCNAME = 'ASIN' THEN
    IF (ABS(CURTOKEN.VALUE)>1) THEN MATHERROR := TRUE
    ELSE BEGIN
         IF (ABS(CURTOKEN.VALUE)<1) and (CURTOKEN.VALUE<>0) THEN BEGIN
            CURTOKEN.VALUE:=ARCTAN(CURTOKEN.VALUE/SQRT(1-SQR(CURTOKEN.VALUE)));
            CURTOKEN.VALUE:=CURTOKEN.VALUE * 180 / PI;
         END;
         IF CURTOKEN.VALUE = 0 THEN CURTOKEN.VALUE :=   0;
         IF CURTOKEN.VALUE = 1 THEN CURTOKEN.VALUE :=  90;
         IF CURTOKEN.VALUE =-1 THEN CURTOKEN.VALUE := -90;
    END

  ELSE IF TOKEN1.FUNCNAME = 'LOG' THEN
    IF CURTOKEN.VALUE<=0 THEN MATHERROR := TRUE
    ELSE CURTOKEN.VALUE := LN(CURTOKEN.VALUE)/LN(10)

  ELSE IF TOKEN1.FUNCNAME = 'FAK' THEN
    IF (CURTOKEN.VALUE>150) OR (CURTOKEN.VALUE<0)
       { OR (FRAC(CURTOKEN.VALUE)<>0) } THEN MATHERROR := TRUE
    ELSE IF CURTOKEN.VALUE = 0 THEN CURTOKEN.VALUE := 1
         ELSE BEGIN
         HELP  := 1;
         TEMP  := TRUNC(ABS(CURTOKEN.VALUE));
         FOR N := 1 TO TEMP DO HELP := HELP*N;
```

```
            CURTOKEN.VALUE := HELP;
            END

      ELSE IF TOKEN1.FUNCNAME = 'ABS' THEN
        CURTOKEN.VALUE := ABS(CURTOKEN.VALUE)

      ELSE IF TOKEN1.FUNCNAME = 'ATAN' THEN BEGIN
        CURTOKEN.VALUE := ARCTAN(CURTOKEN.VALUE);
        CURTOKEN.VALUE := CURTOKEN.VALUE * 180 / PI;
        IF CURTOKEN.VALUE < 0 THEN CURTOKEN.VALUE := 180 + CURTOKEN.VALUE;
      END

      ELSE IF TOKEN1.FUNCNAME = 'COS' THEN BEGIN
        CURTOKEN.VALUE := CURTOKEN.VALUE * PI / 180;
        CURTOKEN.VALUE := COS(CURTOKEN.VALUE);
        IF ABS(CURTOKEN.VALUE) < NULGEN THEN CURTOKEN.VALUE := 0
      END

      ELSE IF TOKEN1.FUNCNAME = 'EXP' THEN
      BEGIN
        IF (CURTOKEN.VALUE < -EXPLIMIT) OR (CURTOKEN.VALUE > EXPLIMIT) THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := EXP(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'LN' THEN
      BEGIN
        IF CURTOKEN.VALUE <= 0 THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := LN(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'ROUND' THEN
      BEGIN
        IF (CURTOKEN.VALUE < -INTLIM) OR (CURTOKEN.VALUE > INTLIM) THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := ROUND(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'SIN' THEN BEGIN
        CURTOKEN.VALUE := CURTOKEN.VALUE * PI / 180;
        CURTOKEN.VALUE := SIN(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'SQRT' THEN
      BEGIN
        IF ABS(CURTOKEN.VALUE) < NULGEN THEN CURTOKEN.VALUE := 0;
        IF CURTOKEN.VALUE < 0 THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := SQRT(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'SQR' THEN
      BEGIN
        IF (CURTOKEN.VALUE < -SQRLIMIT) OR (CURTOKEN.VALUE > SQRLIMIT) THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := SQR(CURTOKEN.VALUE);
      END

      ELSE IF TOKEN1.FUNCNAME = 'TRUNC' THEN
      BEGIN
        IF (CURTOKEN.VALUE < -INTLIM) OR (CURTOKEN.VALUE > INTLIM) THEN
          MATHERROR := TRUE
        ELSE
          CURTOKEN.VALUE := TRUNC(CURTOKEN.VALUE);
      END;
    END;
    3, 6, 8, 10, 12, 15 : POP(CURTOKEN);
  END; { CASE }
  CURTOKEN.STATE := GOTOSTATE(REDUCTION);
  PUSH(CURTOKEN);
END;
```

```
FUNCTION PARSE;
{ EIGENTLICHE KERNROUTINE }
VAR
  FIRSTTOKEN : TOKENREC;
  ACCEPTED   : BOOLEAN;
  NNN        : INTEGER;
  ERG        : REAL;
BEGIN
  ACCEPTED := FALSE;
  TOKENERROR := FALSE;
  MATHERROR := FALSE;
  ISFORMULA := FALSE;
  S := UPPERCASE(BLANKOFF(S));
  NNN := SUBSTRING(S,'(-','((0-1)*');
  NNN := SUBSTRING(S,'(+','((0+1)*');
  IF S[1] = '-' THEN S := '(-1)*' + COPY(S,2,LENGTH(S));
  IF S[1] = '+' THEN S := COPY(S,2,LENGTH(S));
  INPUT := S;
  STACKTOP := 0;
  FIRSTTOKEN.STATE := 0;
  FIRSTTOKEN.VALUE := 0;
  PUSH(FIRSTTOKEN);
  TOKENTYPE := NEXTTOKEN;
  REPEAT
    CASE STACK[STACKTOP].STATE OF
      0, 9, 12..16, 20 : BEGIN
        IF TOKENTYPE = NUM THEN
          SHIFT(10)
        ELSE IF TOKENTYPE = VART THEN
          SHIFT(7)
        ELSE IF TOKENTYPE = FUNC THEN
          SHIFT(11)
        ELSE IF TOKENTYPE = MINUS THEN
          SHIFT(5)
        ELSE IF TOKENTYPE = OPAREN THEN
          SHIFT(9)
        ELSE
          TOKENERROR := TRUE;
      END;
      1 : BEGIN
        IF TOKENTYPE = EOL THEN
          ACCEPTED := TRUE
        ELSE IF TOKENTYPE = PLUS THEN
          SHIFT(12)
        ELSE IF TOKENTYPE = MINUS THEN
          SHIFT(13)
        ELSE
          TOKENERROR := TRUE;
      END;
      2 : BEGIN
        IF TOKENTYPE = TIMES THEN
          SHIFT(14)
        ELSE IF TOKENTYPE = DIVIDE THEN
          SHIFT(15)
        ELSE
          REDUCE(3);
      END;
      3 : REDUCE(6);
      4 : BEGIN
       IF TOKENTYPE = EXPO THEN
         SHIFT(16)
       ELSE
         REDUCE(8);
      END;
      5 : BEGIN
        IF TOKENTYPE = NUM THEN
          SHIFT(10)
        ELSE IF TOKENTYPE = VART THEN
          SHIFT(7)
        ELSE IF TOKENTYPE = FUNC THEN
          SHIFT(11)
        ELSE IF TOKENTYPE = OPAREN THEN
          SHIFT(9)
        ELSE
          TOKENERROR := TRUE;
      END;
      6 : REDUCE(10);
      7 : BEGIN
        IF TOKENTYPE = COLON THEN
          SHIFT(18)
        ELSE
          REDUCE(13);
      END;
```

```
      8 : REDUCE(12);
      10 : REDUCE(15);
      11 : BEGIN
        IF TOKENTYPE = OPAREN THEN
          SHIFT(20)
        ELSE
          TOKENERROR := TRUE;
      END;
      17 : REDUCE(9);
      18 : BEGIN
        IF TOKENTYPE = VART THEN
          SHIFT(26)
        ELSE
          TOKENERROR := TRUE;
      END;

      19 : BEGIN
        IF TOKENTYPE = PLUS THEN
          SHIFT(12)
        ELSE IF TOKENTYPE = MINUS THEN
          SHIFT(13)
        ELSE IF TOKENTYPE = CPAREN THEN
          SHIFT(27)
        ELSE
          TOKENERROR := TRUE;
      END;
      21 : BEGIN
        IF TOKENTYPE = TIMES THEN
          SHIFT(14)
        ELSE IF TOKENTYPE = DIVIDE THEN
          SHIFT(15)
        ELSE
          REDUCE(1);
      END;
      22 : BEGIN
        IF TOKENTYPE = TIMES THEN
          SHIFT(14)
        ELSE IF TOKENTYPE = DIVIDE THEN
          SHIFT(15)
        ELSE
          REDUCE(2);
      END;
      23 : REDUCE(4);
      24 : REDUCE(5);
      25 : REDUCE(7);
      26 : REDUCE(11);
      27 : REDUCE(14);
      28 : BEGIN
        IF TOKENTYPE = PLUS THEN
          SHIFT(12)
        ELSE IF TOKENTYPE = MINUS THEN
          SHIFT(13)
        ELSE IF TOKENTYPE = CPAREN THEN
          SHIFT(29)
        ELSE
          TOKENERROR := TRUE;
      END;
      29 : REDUCE(16);
    END; { CASE }
  UNTIL ACCEPTED OR TOKENERROR;
  IF TOKENERROR THEN
  BEGIN
    ATT := TXT;
    PARSE := 0;
    EXIT;
  END;
  IF ISFORMULA THEN
    ATT := FORMULA
  ELSE
    ATT := VALUE;
  IF MATHERROR THEN
  BEGIN
    INC(ATT, 4);
    PARSE := 0;
    EXIT;
  END;
  ERG := STACK[STACKTOP].VALUE;
  IF ABS(ERG) < NULGEN THEN ERG := 0;
  PARSE := ERG;
END;

BEGIN
END.
```