12 Strukturierter Datentyp Array

Gliederung

12.1 Ein- und mehrdimensionale Arrays 2

12.2 Einleitendes Programmbeispiel 3

12.3 Beispiel: Quadratische Matrix 4

12.4 Beispiel: Statistik I, Mittelwert und Standardabweichung 5

12.5 Beispiel: Statistik II, Regression und Korrelation 6
Korrelation, Grafikdarstellung

12.6 Beispiel: Statistik III, Approximation Gauß-Normalverteilung 12

12.7 Beispiel: Array-Übergabe 17

12.8 Beispiel: Offene Arrays 18

12.9 Beispiel: Primzahlensieb des Erathostenes 20

12.10 Deklaration von Arrays als typisierte Konstanten 21

12.11 Die vordefinierten Arrays Mem, MemW und MemL 22

12.1 Ein- und mehrdimensionale Arrays

Zweck: Zusammenfassung von mehreren Daten mit gleichem Datentyp unter einem Namen (Bezeichner), wobei die einzelnen Elemente (Komponenten) über den Index angesprochen werden. Arrays können eindimensional oder mehrdimensional sein. Eindimensionale Arrays nennt man häufig Vektoren oder Listen. Für zweidimensionale Arrays werden häufig die Begriffe Matrix oder Tabellen benutzt.

Eindimensionale Arrays werden wie folgt deklariert:

```
var
  Arraybezeichner: array[IndexMin..IndexMax] of Datentyp;
```

Die beiden Folgepunkte "`..`" dürfen nicht mit Leerzeichen getrennt werden.

Auch:

```
var
  Arraybezeichner:  array[Indexbereich] of Datentyp;
```

IndexMin, *IndexMax* bzw. selbstdefinierter Aufzählungs- oder Teilbereichstyp *Indexbereich*: Konstanten mit ordinalem Datentyp (Integer, Byte, Word, ShortInt, Char, Boolean, selbstdefiniert) und maximal 65535 Elementen. Der *Datentyp* des Arrays selbst kann beliebig sein (unstrukturiert, strukturiert oder dynamisch).

Aber: Eine (statische) Datenstruktur darf wegen der MS-DOS-Beschränkungen aber nicht mehr als 64 KByte umfassen. Ein Real-Array darf z.B. nicht mehr als 10810 Komponenten enthalten, da ein Real-Element in Turbo-Pascal sechs Byte belegt. Größere Arrays müssen deshalb mit Zeigern dynamisch verwaltet werden, siehe Kap. 19.

Beispiel für einen eindimensionalen Array:

```
...
var
  Vektor: array[1..99] of Integer;
...
begin
  ...
  Vektor[27] := 4711;
  ...
  Write(Vektor[27]);
  ...
end.
```

Beispiel für die Deklaration eines zweidimensionalen `Arrays`:

```
var
  Arraybezeichner: array[i1..i2] of array[j1..j2] of Datentyp;
```

oder anschaulicher:

```
var
  Arraybezeichner: array[i1..i2, j1..j2] of Datentyp;
```

Bei Matrizen muß zuerst der Zeilenindex und dann der Spaltenindex aufgeführt werden. Bei weiteren Dimensionen verfährt man analog.

Beispiel einen zweidimensionalen Array:

```
...
var
  Matrix: array[1..9, 3..13] of Real;
...
begin
  ...
  Matrix[5, 12] := 47.11;   { Zeile 5, Spalte 12 }
  ...
  Write(Matrix[5, 12]);
  ...
end.
```

12.2 Einleitendes Programmbeispiel

```
program Pas12021;  { Kapitel 12, Arrays,  kha }
                   { Einleitendes Beispiel. }
uses               { Integer-Array eindimensional }
  CRT;

const
  N = 4;
var
  x: array[1..n] of Integer;
  i: Integer;

begin
  ClrScr;
  x[1] := 4711;
  x[2] := 4712;
  x[3] := 4713;
  x[4] := 4714;
  for i := 1 to n     do
    Write(' ', x[i]);         { ¦ 4711 4712 4713 4714 }
  WriteLn;
  for i := n downto 1 do
    Write(' ', x[i]);         { ¦ 4714 4713 4712 4711 }
  Writeln;
  Writeln(' ', x[4] - x[1]);  { ¦ 3                   }
  repeat
  until ReadKey <> '';
end.
```

12.3 Beispiel: Quadratische Matrix

```
program Pas12031;    { Kapitel 12, Array,  kha }
  { Quadratische Matrix mit Integer-Elementen.
    Die gegebene quadratische Matrix ist auszugeben. Dabei soll
    gleichzeitig die Summe der Hauptdiagonal-Elemente ermittelt
    werden.
    Nach diesen Operationen ist die Matrix in "kopfgestellter" Form
    auszugeben,  d.h. höchste Zeile mit höchster Spalte zuerst.
    Für das Ansprechen eines Elementes in einem zweidimensionalen
    Array kann man sich folgender Eselsbrücke bedienen:
                         (Z)eilen  (z)uerst,
                         (S)palten (s)päter.
  }
uses
  CRT;
const
  n = 4;
var
  Zeile,
  Spalte:   Integer;
  SummeHDE: Integer;   { Summe der Hauptdiagonal-Elemente }
  x:        array[1..n, 1..n] of Integer;
  {oder:
    x: array[1..n] of array[1..n] of Integer;
  }
begin
  ClrScr;
  x[1, 1] := 7;   x[1, 2] :=  3;   x[1, 3] :=  4;   x[1, 4] := 4;
  x[2, 1] := 2;   x[2, 2] := -4;   x[2, 3] :=  5;   x[2, 4] := 7;
  x[3, 1] := 1;   x[3, 2] :=  0;   x[3, 3] := -2;   x[3, 4] := 8;
  x[4, 1] := 8;   x[4, 2] :=  2;   x[4, 3] :=  3;   x[4, 4] := 2;

  SummeHDE := 0;  { Summe der Hauptdiagonalelemente }
  WriteLn('Original-Matrix:');
  for Zeile := 1 to n do
    begin
      for Spalte := 1 to n do
        begin
          Write(x[Zeile, Spalte]:4);
          if Zeile = Spalte
             then SummeHDE := SummeHDE + x[Zeile, Spalte];
        end;
      WriteLn;
    end;
  WriteLn('Summe der HDE: ', SummeHDE);
  WriteLn;
  WriteLn('Die "kopfgestellte" Matrix:');
  for Zeile := n downto 1 do
    begin
      for Spalte := n downto 1 do
        Write(x[Zeile, Spalte]:4);
      WriteLn;
    end;
```

```
  { Die Bildschirm-Ausgabe:                }
  { -----------------------                }
  {                  ¦Original-Matrix:     }
  {  für Zeile = 1: ¦   7   3   4   4      }
  {  für Zeile = 2: ¦   2  -4   5   7      }
  {  für Zeile = 3: ¦   1   0  -2   8      }
  {  für Zeile = 4: ¦   8   2   3   2      }
  {                  ¦Summe der HDE: 3     }
  {                  ¦                     }
  {                  ¦Gestürzte Matrix:    }
  {  für Zeile = 4: ¦   2   3   2   8      }
  {  für Zeile = 3: ¦   8  -2   0   1      }
  {  für Zeile = 2: ¦   7   5  -4   2      }
  {  für Zeile = 1: ¦   4   4   3   7      }
  repeat
  until ReadKey <> '';
end.
```

12.4 Beispiel: Statistik I, Mittelwert und Standardabweichung

Die Berechnung erfolgt nach folgenden Formeln:

$$\overline{x} = \frac{\sum x_i}{n} \qquad \sigma = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{n}}{n-1}}$$

```
program Pas12041;   { Kapitel 12, Array, kha }
  {
    Statisik I: Mittelwert xQuer und Standardabweichung Sigma.
    Lösung mit eindimensionalem Array nach folgenden Formeln:

    Anmerkung: Es wird hier so getan, als ob die Daten in einem
    späteren Programmteil noch benötigt würden. Ansonsten wäre
    der Gebrauch eines Arrays nicht notwendig.
  }
uses
  CRT;
const
  n = 5;   { Für Demo nur 5 Meßwerte. }
           { Der Statistiker muß wegschauen ! }
var
  i:            1..n;
  Messwert:     array[1..n] of Real;
  SummeXi,
  SummeXiXi,
  xQuer, Sigma: Real;

begin
```

```
  ClrScr;
  Messwert[1] := 2.124;   { Bei ernsthafter Anwendung Daten mit }
  Messwert[2] := 1.987;   { "ReadLn" von Tastatur oder Datei    }
  Messwert[3] := 2.027;   { einlesen. Hier nur Demo!            }
  Messwert[4] := 1.945;
  Messwert[5] := 1.937;

  SummeXi   := 0;         { Summe der Meßwerte              }
  SummeXiXi := 0;         { Summe der Quadrate der Meßwerte }

  for i := 1 to n  do
    begin
      WriteLn('Meßwert Nr. ', i, ': ', Messwert[i]:6:3);
      SummeXi   := SummeXi   + Messwert[i];
      SummeXiXi := SummeXiXi + Sqr(Messwert[i]);
    end;

  xQuer := SummeXi/n;
  Sigma := Sqrt( (SummeXiXi - Sqr(SummeXi)/n) / (n - 1) );

  WriteLn('Mittelwert:         ', xQuer:6:3);   { ¦ ....  2.004 }
  WriteLn('Standardabweichung: ', Sigma:6:3);   { ¦ ....  0.076 }
  repeat
  until ReadKey <> '';
end.
```

12.5 Beispiel: Statistik II
Lineare Regression und Korrelation

Die Berechnung erfolgt nach folgenden Formeln:

$$Q_x = \sum x_i^2 - \frac{(\sum x_i)^2}{n}$$

$$Q_y = \sum y_i^2 - \frac{(\sum y_i)^2}{n}$$

$$Q_{xy} = \sum (x_i \cdot y_i) - \frac{\sum x_i \cdot \sum y_i}{n}$$

$$r = \frac{Q_{xy}}{\sqrt{(Q_x \cdot Q_y)}} \qquad b = \frac{Q_{xy}}{Q_x} \qquad a = \frac{\sum y_i - b \cdot \sum x_i}{n}$$

Korrelationsgerade: `y = bx + a`

`b` = Steigung, `a` = Achsenabschnitt

```
program Pas12051; { Kapitel 12, Arrays,  kha }
  {
   Statistik II: Lineare Regression, Korrelation
   Korrelationsgerade:  y = b*x + a
   In der Statistik:    b = Steigung,  a = Achsenabschnitt
```

```
  }
uses
  CRT;
const
  n = 10;      { Auch mit 10 Meßwerten macht der Statistiker noch    }
               { keine Statistik. Aber hier gehts ja um was anderes! }
var
  i:             1..n;
  X, Y:          array[1..n] of Real;
  SummeXi,
  SummeYi,
  SummeXiXi,
  SummeYiYi,
  SummeXiYi,
  Qx, Qy, Qxy,
  r, a, b:       Real;
begin
  ClrScr;

  { Für ernsthafte Anwendungen Daten mit "ReadLn" einlesen }
  X[1]  :=  1.234;      Y[1]  :=  1.785;
  X[2]  :=  5.324;      Y[2]  :=  4.789;
  X[3]  :=  3.245;      Y[3]  :=  2.964;
  X[4]  :=  7.456;      Y[4]  :=  8.684;
  X[5]  :=  9.345;      Y[5]  :=  7.456;
  X[6]  :=  1.235;      Y[6]  :=  0.836;
  X[7]  := -1.234;      Y[7]  := -2.245;
  X[8]  :=  5.234;      Y[8]  :=  6.371;
  X[9]  :=  1.010;      Y[9]  :=  0.954;
  X[10] :=  6.260;      Y[10] :=  5.920;

  SummeXi   := 0.0;       { Summenspeicher X[i]      }
  SummeYi   := 0.0;       { Summenspeicher Y[i]      }
  SummeXiXi := 0.0;       { Summenspeicher X[i]*X[i] }
  SummeYiYi := 0.0;       { Summenspeicher Y[i]*Y[i] }
  SummeXiYi := 0.0;       { Summenspeicher X[i]*Y[i] }

  for i := 1  to n do
    begin
      SummeXi   :=  SummeXi   + X[i];
      SummeYi   :=  SummeYi   + Y[i];
      SummeXiXi :=  SummeXiXi + X[i]*X[i];
      SummeYiYi :=  SummeYiYi + Y[i]*Y[i];
      SummeXiYi :=  SummeXiYi + X[i]*Y[i];
    end;

  Qx  := SummeXiXi - SummeXi*SummeXi/n;
  Qy  := SummeYiYi - SummeYi*SummeYi/n;
  Qxy := SummeXiYi - SummeXi*SummeYi/n;

  r := Qxy/Sqrt(Qx*Qy);               { Korrelationskoeffizient }
  b := Qxy/Qx;                        { Steigung                }
  a := (SummeYi - b*SummeXi)/n;       { Achsenabschnitt         }

  WriteLn;
  WriteLn('Korrelationskoeffizient  r = ', r:10:5, '   ',
          'Bestimmtheitsmaß Sqr(r) = ', Sqr(r):7:5);
  WriteLn('Korrelation              y =  a  +  b*x');
  WriteLn('Achsenabschnitt          a = ', a:10:5);
  WriteLn('Steigung                 b = ', b:10:5);
```

```
  {  Die Bildschirm-Ausgabe:  }
  {  ----------------------  }
  {  |...  0.96169           }
  {  |...                    }
  {  |... -0.12782           }
  {  |...  0.99190           }
  repeat
  until ReadKey <> '';
end.
```

Das nächste Programm demonstriert die Bildschirmdarstellung einer angenommenen Meßreihe unter Vorwegnahme von Grafikroutinen der Unit GRAPH. Es dient auch zum Üben im Schätzen des Korrelationskoeffizienten.

```
program Pas12052;  { Kapitel 12, Arrays (Statistik)    }
                   { Demo Korrelation. Vorgriff Grafik }
uses               { Dr. K. Haller, 77170498           }
  CRT, GRAPH;
const
  nMax = 1000;         { max. Anzahl Meßwertpaare }
  xMin =  0.0;         { Anfangswert für x        }
  xMax =  9.0;         { Endwert für x            }
  yMin = xMin;
  yMax = xMax - 1.0;
  a0   =  2.0;         { Achsenabschnitt vorerst  }
  b0   =  0.4;         { Steigung vorerst         }
  DeltaYMaxMax = 2.5;  { Grenzwert für maximale Y-Abweichnung }
type
  ArrayTyp = array[1..nMax] of Real;
var
  n:         Word;
  X, Y:      Arraytyp;
  r, a, b:   Real;
  DeltaYMax: Real;
procedure WriteXY(Sp, Ze: Byte; Meldung: string);
begin
  GotoXY(Sp, Ze);
  Write(Meldung);
end;
procedure Grafik(X, Y:       Arraytyp;
                 n:          Word;
                 xMin, xMax,
                 yMin, yMax: Real);
const
  x0 = 100;       { Nullpunkt x. VGA 640 * 480 }
  y0 = 450;       { Nullpunkt y. VGA 640 * 480 }
  mx =  40;       { Maßstab   x. VGA 640 * 480 }
  my =  1.0 * mx; { Maßstab   y.               }
var
  Grafiktreiber,
  Grafikmodus:     Integer;
  i:               Word;
```

```
  rStr, aStr, bStr,
  nStr, iStr,
  DeltaYMaxStr,
  BSMStr:           string;   { B = BSM = Bestimmheitsmaß = r * r }
begin
  Grafiktreiber := Detect;
  InitGraph(Grafiktreiber, Grafikmodus, 'C:\BP\BGI');  { anpassen }
  for i := Round(xMin) to Round(xMax) do     { y-Linien }
    Line(x0 + Round(i * mx), y0,
         x0 + Round(i * mx), y0 - Round(yMax * my));
  for i := Round(yMin) to Round(yMax) do     { x-Linien }
    Line(x0,                    y0 - Round(i * my),
         x0 + Round(xMax * mx), y0 - Round(i * my));

  SetColor(LightCyan);
  SetLineStyle(SolidLn, 0, 3);  { 3 = dreifache Liniendicke }
  Line(x0,                   y0 - Round(a*my), { Regressionsgerade }
       x0 + Round(xMax*mx), y0 - Round((a + b*xMax)*my));
  SetLineStyle(SolidLn, 0, 1);  { 1 = Standardliniendicke }
  SetColor(White);

  for i := Round(xMin) to Round(xMax) do      { x-Achsenbechriftung }
    begin
      Str(i, iStr);
      OutTextXY(x0 + Round(i * mx) - 3, y0 + 4, iStr);
    end;

  for i := Round(yMin) to Round(yMax) do      { y-Achsenbechriftung }
    begin
      Str(i, iStr);
      OutTextXY(x0 - 10, y0 - Round(i * my) - 3, iStr);
    end;

  SetColor(Yellow);

  for i := 1 to n do                          { Meßpunkte }
    Arc(x0 + Round(mx*X[i]), y0 - Round(my*Y[i]), 0, 360, 2);
  SetColor(White);  { Standard }

  Str(r:8:4, rStr);
  Str(a:8:4, aStr);
  Str(b:8:4, bStr);
  Str(n:8,   nStr);
  Str(DeltaYMax:8:4, DeltaYMaxStr);
  Str(r*r:6:4, BSMStr);

  OutTextXY( x0, y0 - 428, 'Korrelation             y = ' +
                           ' a + b*x');
  OutTextXY( x0, y0 - 414, 'Achsenabschnitt         a = ' +
                            aStr);
  OutTextXY( x0, y0 - 400, 'Steigung                b = ' +
                            bStr);
  OutTextXY( x0, y0 - 386, 'Anzahl                  n = ' +
                            nStr);
```

```
  OutTextXY( x0, y0 - 372, 'Kennwert für max.   DeltaY = ' +
                           DeltaYMaxStr);
  OutTextXY(500, y0 -  80, 'Korrelations-' );
  OutTextXY(500, y0 -  68, 'koeffizient'   );
  OutTextXY(500, y0 -  50, 'Dr. K. Haller' );
  OutTextXY(500, y0 -  38, 'FH München, DR');

  SetColor(Yellow);
  OutTextXY(x0, y0 - 442, 'Ausgabe Korrelationskoeffizient ' +
                          'nach beliebigem Tastendruck ... ');
  repeat
  until ReadKey <> '';

  SetColor(Black);  { Grafiktext wieder löschen durch Überschreiben }
                    { mit der (schwarzen) Hintergrundfarbe          }
  OutTextXY(x0, y0 - 442, 'Ausgabe Korrelationskoeffizient ' +
                          'nach beliebigem Tastendruck ... ');

  SetColor(Yellow);

  OutTextXY(x0, y0 - 442, 'Korrelationskoeffizient  r = ' + rStr +
                          '   (B = ' + BSMStr + ')');
  SetColor(Yellow);
  OutTextXY(x0, y0 +  20, 'Weiter mit beliebigem Tastendruck ... ');

  repeat
  until ReadKey <> '';
  CloseGraph;
end;

procedure Einzug_n_und_DeltaYMax(var n:         Word;
                                 var DeltaYMax: Real);
var
  Fehlercode:      Integer;
  Spalte, Zeile:   Byte;
  nStr, nMaxStr:   string;
  DeltaYMaxStr,
  DeltaYMaxMaxStr: string;

begin
  TextBackground(Blue);
  TextColor(Yellow);
  ClrScr;
  Str(nMax, nMaxStr);
  Str(DeltaYMaxMax:4:2, DeltaYMaxMaxStr);
  WriteXY(10, 2, 'Programm zum Schätzen des Korrelations' +
                 'koeffizienten. kha');
  WriteXY(10, 3, '-------------------------------------' +
                 '-----------------');
  TextColor(White);
  WriteXY(10, 5, 'Eingabe n (0, 2 ... ' + nMaxStr +
                 ',  Ende mit 0):  ');
  Spalte := WhereX;
  Zeile  := WhereY;
  WriteXY(10, 6, 'Kennwert für max. Delta-Y (0 ... ' +
                  DeltaYMaxMaxStr + '):  ');
  repeat
    GotoXY(Spalte, Zeile);
    ClrEoL;
    ReadLn(nStr);
```

```
    Val(nStr, n, Fehlercode);
    if Fehlercode = 0
       then if (n < 0) or (n > nMax) or (n = 1)
               then Fehlercode := 4711;
  until Fehlercode = 0;

  if n = 0
     then Exit;  { >>>>>>>>>>>>>>>>>>>> }

  Inc(Zeile);
  repeat
    GotoXY(Spalte, Zeile);
    ClrEoL;
    ReadLn(DeltaYMaxStr);
    Val(DeltaYMaxStr, DeltaYMax, Fehlercode);
    if Fehlercode = 0
       then if (DeltaYMax < 0) or (DeltaYMax > DeltaYMaxMax)
               then Fehlercode := 4711;
  until Fehlercode = 0;
end;

                                      { Strukturierten Datentyp nur  }
procedure Datengenerierung(var X, Y:  Arraytyp; { mit eigenem Daten- }
                           n:         Word;     { typ übergeben!!!   }
                           DeltaYMax: Real;     { Ausnahme: "string" }
                           a0, b0:    Real);    { ohne Längenbegrenz.}
var
  i:      Word;
  DeltaY: Real;
begin
  { Randomize; }  { Für Vergleichbarkeit besser kein "Randomize" }
  for i := 1 to n do
    begin
      X[i]   := xMin + (xMax - xMin)*Random;
      DeltaY := DeltaYMax*(2*Random - 1);
      Y[i]   := a0 + b0*X[i] + DeltaY;
    end;
end;

procedure Statistik(X, Y:        Arraytyp;
                    n:           Word;
                    var r, b, a: Real);
var
  SummeXi,
  SummeYi,
  SummeXiXi,
  SummeYiYi,
  SummeXiYi,
  Qx, Qy, Qxy: Real;
  i:           Word;

begin
  SummeXi   := 0;       { Summenspeicher X[i]      }
  SummeYi   := 0;       { Summenspeicher Y[i]      }
  SummeXiXi := 0;       { Summenspeicher X[i]*X[i] }
  SummeYiYi := 0;       { Summenspeicher Y[i]*Y[i] }
  SummeXiYi := 0;       { Summenspeicher X[i]*Y[i] }

  for i := 1  to n do
```

```
    begin
      SummeXi   :=  SummeXi   + X[i];
      SummeYi   :=  SummeYi   + Y[i];
      SummeXiXi :=  SummeXiXi + X[i]*X[i];
      SummeYiYi :=  SummeYiYi + Y[i]*Y[i];
      SummeXiYi :=  SummeXiYi + X[i]*Y[i];
    end;

  Qx  := SummeXiXi - SummeXi*SummeXi/n;
  Qy  := SummeYiYi - SummeYi*SummeYi/n;
  Qxy := SummeXiYi - SummeXi*SummeYi/n;
  r   := Qxy/Sqrt(Qx*Qy);               { Korrelationskoeffizient }
  b   := Qxy/Qx;                        { Steigung                }
  a   := (SummeYi - b*SummeXi)/n;       { Achsenabschnitt         }
end;

begin  { ----------------- Hauptprogramm -------------------------- }
  repeat
    Einzug_n_und_DeltaYMax(n, DeltaYMax);
    if n <>  0 then
       begin
         Datengenerierung(X, Y, n, DeltaYMax, a0, b0);
         Statistik(X, Y, n, r, b, a);
         Grafik(X, Y, n, xMin, xMax, yMin, yMax);
       end;
  until n = 0;
end.
```

12.6 Beispiel: Statistik III Approximation Gauß-Normalverteilung

Das nächste Programm zeigt keine Anwendung des Datentyps "array", sondern zur Abrundung der Statistik-Anwendungsbeispiele die Approximation der Gaußschen Fehlerintegrals ø(z) der Normalverteilung φ(z). Bekanntlich kann dieses Integral nicht geschlossen dargestellt werden und somit auch nicht die inverse Funktion $z = z(ø)$. Es werden deshalb die Approximationsformeln von Abramowitz/Stegun, "Handbook of Mathematical Functions", New York, Dover, 1965, verwendet. Von dem vergriffenen Original gibt es einen Reprint im vollständigem Umfang und einen um unwesentliche Zahlentabellen verkürzten Reprint "Pocketbook Of Mathematical Functions". Beide Reprints sind erschienen im Verlag Harri Deutsch, Thun, Frankfurt/M, 1984. Im Pocketbook sind die Intergrations-Approximationsformeln und die zugehörigen Konstanten auf Seite 408 unter Formel 26.2.17 für $ø = f(z)$) und auf Seite 409 unter Formel 26.2.23 für die Umkehrung (Inversion) $ø = f(z)$ angegeben.

Die Formel für die Gauß-Normalverteilung mit $\bar{x}$ = Mittelwert und σ = Standardabweichung:

(1) $$\varphi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-[(x-\bar{x})/\sigma]^2/2}$$

Mit $z = (x - \bar{x}) / \sigma$ und $\sigma = 1$ erhält man die normierte Darstellung:

(2) $$\varphi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$ normierte Wahrscheinlichkeitsdichte

Die normierte Wahrscheinlichkeit bzw. normierte Verteilungsfunktion stellt sich dar als:

(3) $$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-z^2/z} dz$$

Das Integral in (3) läßt sich bekanntlich nicht lösen und somit gibt es auch keine Umkehrung (Inverse) $z = f(\Phi)$ als geschlossene Lösung.

Für die praktische DV-Statistik-Anwendung ist man auf Approximationsformeln angewiesen (wenn man nicht in jedem Einzelfall numerisch integrieren möchte), z.B. auf die der genannten Autoren.

Sie lauten:

(4) $$\Phi(z) \approx 1 - \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \cdot (b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5) \quad \text{mit} \quad t = \frac{1}{1 + p \cdot z}, \quad 0 \le z < \infty$$

Die Approximationskonstanten für (4):

p = 0,231641900
b_1 = 0,319381530
b_2 = -0,356563782
b_3 = 1,781477937
b_4 = -1,821255978
b_5 = 1,330274429

Der Absolutwert des Fehlers von (4) wird von den Autoren mit $< 7.5 \cdot 10^{-8}$ angegeben.

Für die Umkehrung (Inverse) geben die Autoren an:

(5) $$z(\Phi) \approx t - \frac{c_0 + c_1 t + c_2 t^2}{1 + d_1 t + d_2 t^2 + d_3 t^3} \quad \text{mit} \quad t = \sqrt{\ln \frac{1}{\Phi^2}}, \qquad 0 < \Phi \le 0.5$$

Die Approximationskonstanten für (5):

c_0 = 2,515517 d_1 = 1,432788
c_1 = 0,802853 d_2 = 0,189269
c_2 = 0,010328 d_3 = 0,001308

Der Absolutwert des Fehlers von (5) wird von den Autoren mit $< 4.5 \cdot 10^{-4}$ angegeben.

Es folgt das Demo-Programm:

```
{$N+,E+ Compilerschalter für Coprozessor oder Emulation }
program Pas12061; { Statistik III, Gauß-Normalverteilung -----------+}
```

```
                  { 27240593,  Dr. K. Haller, FH München Stg DR     ¦}
      { Normierte Zufallsgröße z = (x - xQuer) / Sigma              ¦}
      { xQuer = Mittelwert,  Sigma = Standardabweichung             ¦}
      { Approximationen Phi(z) und z(Phi) nach Abramowitz, Stegun:  ¦}
      { "Handbook of Mathematical Functions", New York, Dover, 1965 ¦}
      { Reprint 1984: "Pocketbook of Mathematical Functions"        ¦}
      { Verlag Harri Deutsch, Frankfurt/M, Seiten 408, 409 und 435. ¦}
      {-------------------------------------------------------------+}
uses
  CRT;

type  { Wenn Koprozessor oder Emulation, Typ "Double", sonst "Real" }
  TReal = Double;

const { Folgende Konstanten wegen "Bereichsüberschreitung"   }
  zMax1 = 150.70;  { Mit Coprozessor "150.70", sonst "13.26" }
  zMax3 = 106.56;  { Mit Coprozessor "106.56", sonst  "9.38" }

var
  z, PhiKlein, PhiGross: TReal;
  Ch:                    Char;

function PhiGross_z(z, zMax1: TReal): TReal; { ---------------------+}
const  { Approximationskonstanten nach Abramowitz/Stegun            ¦}
       { Reprint: Seite 408, Formel 26.2.17.  |Fehler| < 7.5E-8     ¦}
       { PhiGross(z): Integral von -unendlich bis z der Gauß-Kurve  ¦}
  p  = 0.231641900;  b1 =  0.319381530;  b2 = -0.356563782;        {¦}
  b3 = 1.781477937;  b4 = -1.821255978;  b5 =  1.330274429;        {¦}
var                                                                {¦}
  t: TReal;                                                        {¦}
begin                                                              {¦}
  if z < -zMax1 then begin PhiGross_z := 0.0; Exit; end; { >>>>>>>>>¦}
  if z > +zMax1 then begin PhiGross_z := 1.0; Exit; end; { >>>>>>>>>¦}
  t  := 1/(1 + p*Abs(z));                                          {¦}
  t  := ((((b5*t + b4)*t + b3)*t + b2)*t + b1)*t;                  {¦}
  if z < 0.0                                                       {¦}
     then PhiGross_z :=        t * Exp(-Sqr(z)/2.0) / Sqrt(2*Pi)   {¦}
     else PhiGross_z := 1.0 -  t * Exp(-Sqr(z)/2.0) / Sqrt(2*Pi);  {¦}
end; { -------------------------------------------------------------+}

function z_PhiGross(Phi, zMax1: TReal): TReal; { -------------------+}
const  { Approximationskonstanten nach Abramowitz/Stegun (invers)   ¦}
       { Reprint: Seite 409, Formel 26.2.23. |Fehler| < 4.5E-4      ¦}
       { Phi(z): Integral von -unendlich bis z der Gauß-Kurve       ¦}
  c0 = 2.515517;  c1 = 0.802853;  c2 = 0.010328;                   {¦}
  d1 = 1.432788;  d2 = 0.189269;  d3 = 0.001308;                   {¦}
  PhiMin = 1.0E-16; { Mit Coprozessor "1E-16", sonst "1E-12" }     {¦}
var                                                                {¦}
  t:                  TReal;                                       {¦}
  PhiGroesserEinHalb: Boolean;                                     {¦}
begin                                              { +- Dummy }    {¦}
  if  Phi        < PhiMin then begin z_PhiGross := -999.99; Exit; end;
  if (1.0 - Phi) < PhiMin then begin z_PhiGross := +999.99; Exit; end;
  if Phi >= 0.5                                                    {¦}
     then PhiGroesserEinHalb := True                               {¦}
     else begin                                                    {¦}
            PhiGroesserEinHalb := False;                           {¦}
            Phi := 1.0 - Phi;                                      {¦}
```

```
        end;                                                          {¦}
  t := Sqrt(Ln(1/Sqr(1 - Phi)));                                      {¦}
  t := t - (( c2*t + c1)*t + c0) / (((d3*t + d2)*t + d1)*t + 1);      {¦}
  if PhiGroesserEinHalb                                               {¦}
     then z_PhiGross :=  t                                            {¦}
     else z_PhiGross := -t;                                           {¦}
end; { ------------------------------------------------------------------+}

procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
  Write(Meldung);
end;

begin { ---------------- Hauptprogramm ----------------------------+}
  TextBackground(Blue);

  repeat
    TextColor(Yellow);
    ClrScr;
    WriteXY(6,  2, 'Gauß-Normalverteilung mit Integral-' +
                   'Approximation nach Abramowitz/Stegun');
    WriteXY(6,  3, '------------------------------------' +
                   '--------------------------------');
    TextColor(LightGreen);
    WriteXY(6,  4, 'z:          Normierte Zufallsgröße z = '         +
                    '(x - xQuer)/Sigma');
    WriteXY(6,  5, 'xQuer:      Mittelwert,     Sigma: Standard'     +
                   'abweichung');
    WriteXY(6,  6, 'Phi = f(z): (Groß-Phi) Normierte Wahrscheinlich' +
                   'keit (Verteilungs-');
    WriteXY(6,  7, '            funktion) = Approximation des '      +
                   'Integrals der normierten');
    WriteXY(6,  8, '            Verteilungsdichte (Klein-Phi) von '  +
                   '-unendlich bis z von');
    WriteXY(6,  9, '                                phi = Exp(-z*z/2)' +
                   '/Sqrt(2*Pi)');
    WriteXY(6, 10, '            zMax = ±150.70 (mit Coprozessor), '  +
                   '= ±13.26 (ohne Coproz.)');
    WriteXY(6, 11, 'z = f(Phi): Inverse Approximation');
    TextColor(Yellow);
    WriteXY(6, 12, '------------------------------------' +
                   '---------------- kha FHM DR ----');

    WriteXY(18, 13, '1   Phi = f(z)   Integral-Approximation       ');
    WriteXY(18, 14, '2   z   = f(Phi) Integral-Approximation invers');
    WriteXY(18, 15, '3   phi = f(z)   Gauß-Kurve                   ');
    WriteXY(18, 16, '4   z   = f(phi) Gauß-Kurve invers            ');
    WriteXY(18, 17, 'Esc Ende                                      ');
    WriteXY(18, 18, '----------------------------------------------');
    GotoXY( 18, WhereY + 1);
    repeat
      Ch := ReadKey;
      if Ch = #27 then Exit; { >>>>>>>>>>>> }
    until Ch in ['1'..'4'];
    WriteLn(Ch); WriteLn;

    case Ch of
```

```
    '1': begin
           WriteLn('     ------------------ Wahrscheinlichkeit ' +
                   'Phi = f(z) ----------------------');
           repeat
             Write('     Eingabe z (± zMax siehe oben, ' +
                   'Ende mit 0): ');
             ReadLn(z);
             GotoXY(59, WhereY - 1);
             WriteLn('Phi(z) = ', PhiGross_z(z, zMax1):9:6);
           until z = 0.0;
           repeat
           until ReadKey <> '';
         end;
    '2': begin
           WriteLn('     ------------- normierte Zufallsgröße ' +
                   'z = f(Phi) ----------------------');
           repeat
             repeat
               Write('     Eingabe Phi (0.0 ... 1.0, Ende mit 0): ');
               ReadLn(PhiGross);
             until (PhiGross >= 0.0) and (PhiGross <= 1.0);
             GotoXY(57, WhereY - 1);
             WriteLn('z(Phi) = ', z_PhiGross(PhiGross, zMax1):11:6);
           until (PhiGross = 0.0);
           repeat
           until ReadKey <> '';
         end;
    '3': begin
           WriteLn('     ------------------ Gauß-Kurve, ' +
                   'phi = f(z) -----------------------------');
           repeat
             Write('     Eingabe z (Ende mit 0): ');
             ReadLn(z);
             GotoXY(59, WhereY - 1);
             if Abs(z) < zMax3
                then PhiKlein := Exp(-Sqr(z)/1) / Sqrt(2*Pi)
                else PhiKlein := 0.0;
             WriteLn('phi(z) = ', PhiKlein:9:6);
           until z = 0.0;
           repeat
           until ReadKey <> '';
         end;
    '4': begin
           WriteLn('     ------------------ Gauß-Kurve invers, ' +
                   'z = f(phi) ----------------------');
           repeat
             repeat
               Write('     Eingabe phi (0.0 ...', 1/Sqrt(2*Pi):8:5,
                     ', Ende mit 0): ');
               ReadLn(PhiKlein);
             until (PhiKlein >= 0.0) and (PhiKlein < 1/Sqrt(2*Pi));
             if PhiKlein > 0.0 then
                begin
                  GotoXY(59, WhereY - 1);
                  WriteLn('z(phi) = ±',
```

```
                          Sqrt(-2*Ln(Sqrt(2*Pi)*PhiKlein)):9:6 );
              end;
          until (PhiKlein = 0.0);
        end;
    end;
  until False;  { Pseudo-Endlosschleife }
end.
```

12.7 Beispiel: Array-Übergabe

```
program Pas12071;  { Kapitel 12, Arrays: Demo Array-Übergabe     }
                   { "Offene Array-Grenzen" siehe "Pas12081.PAS" }
uses
  CRT;

const
  i1 = 1;
  i2 = 6;

type
  Arraytyp = array[i1..i2] of Real;

var
  Array1,
  Array2:  Arraytyp;
  i:       Integer;

procedure Demo_Array_Uebergabe(Array3: Arraytyp;
                               Index1, Index2: Integer);
var
  i: Integer;
begin
  Write('Array in Prozedur: ');

  for i := Index1 to Index2 do
    Write(Array3[i]:6:2);

  WriteLn;
end;


begin  { Hauptprogramm }
  ClrScr;

  Array1[1] := 47.11; Array1[2] := 47.12; Array1[3] := 47.13;
  Array1[4] := 47.14; Array1[5] := 47.15; Array1[6] := 47.16;

  Write('Array1:            ');

  for i := i1 to i2 do
    Write(Array1[i]:6:2);

  WriteLn;

  Array2 := Array1;  { <--- Zuweisung eines Arrays an einen anderen }

  Write('Array2:            ');
```

```
  for i := i1 to i2 do
    Write(Array2[i]:6:2);

  WriteLn;

  Demo_Array_Uebergabe(Array1, i1, i2);

  { Alle Bildschirmausgaben zusammen:                    }

  { ¦Array1:             47.11 47.12 47.13 47.14 47.15 47.16 }
  { ¦Array2:             47.11 47.12 47.13 47.14 47.15 47.16 }
  { ¦Array in Prozedur:  47.11 47.12 47.13 47.14 47.15 47.16 }

  repeat
  until ReadKey <> '';
end.
```

12.8 Array-Übergabe mit offenen Arrays

Mit dem Compilerschalter {$P+} oder über den Menüpunkt "Option/Compiler/Offene Array-Grenzen" können in neueren Turbo-Pascal-Versionen sog. "Offene Arrays" an Prozeduren oder Funktionen übergeben werden. Damit können an die gleiche Routine nacheinander verschieden große Arrays übergeben werden, ohne daß man die Deklaration bei der Routine auf den größten der vorkommenden Arrays abstimmen muß. Allerdings läuft der Array-Index in der Routine immer von 0 bis n - 1, wenn n die Anzahl der Array-Elemente ist. Der untere Index kann mit der Standardfunktion "Low(Arraybezeichner)", der obere mit "High(Arraybezeichner)" ermittelt werden. Wendet man "Low(..)" bzw. "High(..)" im Hauptprogramm an, erhält man die deklarierten Indizies. Da "Low(..)" in der Routine immer 0 liefert, kann man sich den Aufruf dieser Funktion sparen und gleich den Wert 0 annehmen. Da man Strings als "arrays of Char" auffassen kann, kann man in gleicher Weise auch "Offene Strings" an Routinen übergeben, wie das Demo-Programm zusätzlich noch zeigt. Für diese Zwecke kann man vorteilthaft den (neuen) Datentypbezeichner "OpenString" einsetzen. Details dazu siehe Kap. 14.

Kleine Einschränkung: Offene Arrays können innerhalb der Routine nur elementweise verarbeitet werden, z-B. mit for-Schleifen. Eine direkte Zuweisung eines Arrays an einen anderen ist bei offenen Arrays in der Routine nicht möglich.

```
{$P+ Compilerschalter für "Offene Arrays }
program Pas12081;  { Demo "Offene Arrays" ab Turbo-Pascal 7.0 }
                   { 47170492, kha       }
uses
  CRT;
var
  x:  array[2..4] of Byte;  { "normale" Array-Deklaration }
  y:  array[1..2] of Byte;  { dto. }
  s1: string[50];           { "normale" String-Deklaration }
```

```
  s2: string[4];            { dto. }
procedure Test_OffeneArrays(z: array of Byte; { offener Array  }
                            s: OpenString);   { offener String }
var                         { "OpenString" ist vordefiniert    }
  i, iMin, iMax: Byte;
begin
  TextColor(White);
  WriteLn('In der Routine mit offenen Arrays:');
  iMin := Low(z);
  iMax := High(z);
  Writeln('iMin von z: ', iMin, ', iMin von s: ', Low(s) );
  WriteLn('iMax von z: ', iMax, ', iMax von s: ', High(s));
  for i := iMin to iMax do
    WriteLn('i = ', i, ',   z[i] = ', z[i]);
  WriteLn('s: ', s);
  TextColor(Yellow);
end;

begin
  TextBackground(Blue); TextColor(Yellow); ClrScr;
  x[2] :=  1; x[3] :=  2;  x[4] := 3;
  y[1] := 11; y[2] := 12;

  WriteLn('Die Funktionen "Low" und "High" im Hauptprogramm:');
  WriteLn('Low(x): ', Low(x), ', High(x): ', High(x));
  Writeln;

  Test_OffeneArrays(x, 'Anton Huber');

  WriteLn;

  Test_OffeneArrays(y, 'Gabi');
  repeat
  until ReadKey <> '';
end.
```

12.9 Beispiel: Primzahlensieb des Erathostenes

```
program Pas12091;   { ---- Primzahlen-Sieb des Erathostenes --------}
      { Inhaltlich identisch mit "Prim_TP.PAS", K. Haller, 57170497 }
      { Dieses Primzahl-Programm dient häufig für den Geschwindig-
        keitsvergleich(Benchmark) von Programmiersprachen oder
        Rechner-Systemen. Der Benchmark liegt in vergleichbarer
        Programmiertechnik für folgende Sprach-Versionen vor:
        ¦ Turbo-C      "Prim-TC.C"       ¦ Turbo-Basic  "Prim-TB.BAS"
        ¦ Turbo-Pascal "Prim-TP.PAS"     ¦ Quick-Basic  "Prim-QB.BAS"
        ¦ Quick-Pascal "Prim-QP.PAS"
        ¦ GWBASIC      "Prim-GW.BAS" <-- nur Interpreter-Version
        Man achte bei den Sprach-Versionen auf den Wert von "Endwert".
      }
uses
  CRT, DOS;
const
  Endwert    = 60000;   { Alle Primzahlen von 1 bis "Endwert" }
  AusgabeMax =  2000;   { aber nur bis "AusgabeMax" ausgeben  }
var
```

```
  PrimzahlFlag:       array[1..Endwert] of Boolean;
  i, k:               Word;
  hh, mm, ss, ss100:  Word;
  Zeit:               Real;
begin
  TextBackground(Blue); TextColor(Yellow); ClrScr;
  GotoXY(20, 1);
  Write('Sieb des Erathostenes. Primzahlen bis ', Endwert);
  GotoXY(20, 2);
  Write('------- Version in Turbo-Pascal -----------');
  GotoXY(20, 3); Write('Bitte warten .... ');
  GetTime(hh, mm, ss, ss100);
  Zeit := hh*3600.0 + mm*60.0 + ss + ss100/100;
{ +---------- Sieb des Erathostenes ----------------------------+ }
  for i := 1 to Endwert do
    PrimzahlFlag[i] := True; { Array auf "True" initialisieren }
  for i := 2 to Endwert div 2 do       { Die Primzahl "1"   }
    if PrimzahlFlag[i]                 { wird ausgeklammert }
       then for k := 2 to (Endwert - i) div i + 1 do
              PrimzahlFlag[k*i] := False;
    { Erathostenes: "Wenn die Zahl i eine Primzahl ist,       }
    { dann sind die Vielfachen von i  k e i n e  Primzahlen". }
{ +-------------------------------------------------------------+ }
  GetTime(hh, mm, ss, ss100);
  Zeit := hh*3600.0 + mm*60.0 + ss + ss100/100 - Zeit;
  GotoXY(20, 3);
  Write('Primzahlen bis ', Endwert, '. Zeit:', Zeit:4:1, ' s');
  GotoXY(20, 4);
  WriteLn('Aus Platzgründen Ausgabe nur bis ', AusgabeMax);
  TextColor(LightCyan);
                                  { Die Rechenzeiten bis 30000 }
  for i := 1 to AusgabeMax do     { mit i80386, 16 MHz:        }
    if PrimzahlFlag[i]            { ¦ Turbo-C        ca. 0.7 s }
       then Write(i:5);           { ¦ Turbo-Pascal   ca. 0.4 s }
                                  { ¦ Quick-Pascal   ca. 0.4 s }
                                  { ¦ Turbo-Basic    ca. 0.5 s }
                                  { ¦ Quick-Basic    ca. 3.1 s }
                                  { ¦ GWBASIC:       ca. 80 s  }
    { Beim Pentium mit 166 MHz ergeben sich auch bei n = 60000 }
    { praktisch nichtmeßbare Zeiten < 0.1 s.                   }
  TextColor(Yellow);
  Write(#13#10, ' Ende mit beliebigem Tastendruck ... ');
  repeat;
  until ReadKey <> '';
end.
```

12.10 Deklaration von Arrays als typisierte Konstanten

Das folgende Programm demonstriert die Deklaration von Arrays als typisierte Konstanten. Man beachte, daß typisierte Konstanten nichts anderes sind als Variablen, die

bei der Deklaration initialisiert werden. Bei zwei- und mehrdimensionalen Arrays achte man auf die Abfolge Zeile/Spalte.

```
program Pas12101; { Arrays können auch als typisierte Konstanten }
      { deklariert werden, nicht aber als "echte" Konstanten.    }
      { Wiederholung: Typisierte Konstanten sind initialisierte  }
      { Variablen.}
uses
  CRT;
const   { Arrays als typisierte Konstanten }
  Vektor: array[1..5]       of Byte = (0, 2, 4, 6, 8);
  Matrix: array[1..3, 1..4] of Byte = ( (1, 2, 3, 4),    { 1. Zeile }
                                        (5, 6, 7, 8),    { 2. Zeile }
                                        (9, 0, 1, 2) );  { 3. Zeile }
var
  i, j: Byte;

begin
  TextBackground(Blue);
  TextColor(Yellow); ClrScr;

  for i := 1 to 5 do
    Write(Vektor[i]:4);
  WriteLn; WriteLn;

  TextColor(White);
  for i := 1 to 5 do
    begin
      Vektor[i] := Vektor[i] + 100;
      Write(Vektor[i]:4);
    end;
  WriteLn; WriteLn;

  TextColor(LightCyan);
  for i := 1 to 3 do
    begin
      for j := 1 to 4 do
        Write(Matrix[i, j]:4);
      WriteLn;
    end;
  WriteLn;

  TextColor(LightGray);
  for i := 1 to 3 do
    begin
      for j := 1 to 4 do
        begin
          Matrix[i, j] := Matrix[i, j] + 100;
          Write(Matrix[i, j]:4);
        end;
      WriteLn;
    end;
  repeat
  until KeyPressed;
end.
```

12.11 Vordefinierte Arrays: Mem, MemW, MemL

Turbo-Pascal stellt die (vordefinierten Pseudo-) Arrays *Mem*, *MemW* und *MemL* zur Verfügung, um Speicherzellen zu lesen und auch zu beschreiben; letzteres natürlich nicht für ROM-Speicher. Die (Anfangs-) Adresse der Speicherzellen muß im üblichen Intel-Format "Segment:Offset" angegeben werden, meistens bedient man sich der Hex-Notation.

- Mit `Mem[Segment:Offset]` wird ein Byte gelesen/geschrieben. Datentyp: Byte
- Mit `MemW[Segment:Offset]` werden zwei Byte gelesen/geschrieben. Datentyp: Word
- Mit `MemL[Segment:Offset]` werden vier Byte gelesen/geschrieben. Datentyp: LongInt

Im Kapitel "Systemnahe Programmierung" wird näher darauf eingegangen. An dieser Stelle lediglich ein kleines Demo-Programm, das das Tastaturstatusbyte des Rechners liest und überschreibt. Dieses Statusbyte wird in der Segmentadresse $0040 und die Offsetadresse $0017 gespeichert. Diese Speicheradresse liegt im RAM; die Berechnung der physischen Adresse ergibt: $16*(4*16^1 + 0*16^0) + 1*16^1 + 7*16^0 = 1047$.

```
program Pas12111;  { Kapitel 12.11: Array. Vordefinierte Arrays }
uses               { "Mem":  Byte,    1 Byte }
  CRT;             { "MemW": Word,    2 Byte }
begin              { "MemL": LongInt, 4 Byte }
  ClrScr;
  Write('Achten Sie auf die Leuchtdiode "Num": ');
  repeat
    Mem[$0040:$0017] := Mem[$0040:$0017] xor 32;   { 32 = 2^5 }
    Delay(100);     { "NumLock", Bit 5 im Tastatur-Statusbyte }
  until KeyPressed;
end.
```