7 Einfache Ein- und Ausgaben Bildschirm, Tastatur und Drucker Vermischtes

Gliederung

In diesem Kapitel werden einfache Ausgaben auf Bildschirm, Eingaben von der Tastatur und Ausgaben auf den Drucker behandelt. Ein- und Ausgaben auf Diskette/Platte werden im Kapitel "Dateien" behandelt. Für die Behandlung der Bildschirm-Graphik und für die Tastatur-Programmierung sind ebenfalls separate Kapitel vorgesehen.

Für die Beispiele in diesem Kapitel muß ein Vorgriff auf einige Datentypen gemacht werden. Nachstehend eine Auflistung, die z.T. vereinfacht ist. Auf die vollständige Behandlung im Kapitel *Datentypen* wird verwiesen.

Datentyp	Definition, Bereich, Erklärungen
`Integer`	Ganzzahlen im Bereich -32768..32767
`Word`	Ganzzahlen im Bereich 0..65535
`Byte`	Ganzzahlen im Bereich 0..255
`Real`	Kommazahlen im Bereich -1.7E+38..1.7E+38. Darstellung in der Gleitkomma-Schreibweise (wissenschaftliche Schreibweise, Standard) oder in der Fixkomma-Schreibweise
`Char`	Einzelnes Zeichen (character). 256 verschiedene Zeichen, von *#0* bis *#255*
`string`	Zeichenkette mit maximal 255 Zeichen. Anmerkung: Ab Turbo-Pascal 7.0 können mit der neuen Unit Strings und den darin enthaltenen Deklarationen nullterminerte Strings miteiner Länge von bis zu 2^{32} = 65535 Zeichen verwaltet werden. In Delphi ist die Stringlänge nur noch durch die Speicherkapazität beschränkt.
`Boolean`	Kann nur die vordefinierten Wahrheitswerte *True* oder *False* annehmen

7.1 Die Standardprozeduren Write und WriteLn

Mit den Standardprozeduren *Write* und *WriteLn* werden Daten auf den Bildschirm ausgegeben. Der Unterschied zwischen *Write* und *WriteLn* besteht lediglich darin, daß der Cursor bei *Write* nach der Ausgabe des letzten Zeichens in der Zeile stehen bleibt, wogegen bei *WriteLn* (Write Line) der Cursor nach der Ausgabe des letzten Zeichens auf den Anfang der nächsten Bildschirmzeile gesetzt wird.

Formate:

```
Write(  a1 [, a2, ..., an])
WriteLn(a1 [, a2, ..., an])
WriteLn
```

a1, a2, ... an Ausdrücke

Die eckigen Klammern sind *nicht* einzugeben, da sie in der Formatbeschreibung lediglich Symbole für Optionen darstellen. Ebenso nicht die Punkte, die beliebige Wiederholungen symbolisieren. Siehe Kap. 4.6, in dem auch der Begriff "Ausdruck" erklärt ist.

Wenn *keine* Ausgabeliste angegeben ist (3. Format), dann wird mit *WriteLn* eine Leerzeile gedruckt bzw. eine noch nicht abgeschlossene Zeile abgeschlossen.

Bei den folgenden Beispielen wird angenommen, daß *x*, *y* und *z* numerische Variablen sind, wogegen *s* eine Stringvariable sein soll.

Beispiele:

```
Write(x, y, z);              { Cursor bleibt in der Zeile }
WriteLn('Der Wert von x: ', x); { Nach Ausgabe neue Zeile }
Write('Der Funktionswert: ', x + 3*Sin(z)/2 - 7);
WriteLn('Der Name: ', s);    { Nach Ausgabe neue Zeile    }
```

Real-Typen (Kommazahlen) werden standardmäßig in Gleitkomma-Schreibweise mit einer Schreibbreite von 17 Zeichen (Datentyp Real) oder 23 Zeichen (Datentyp Double) ausgegeben. Mit Hilfe einer Formatierung können Real-Typen aber auch in Fixkomma-Schreibweise ausgegeben werden. Die Anzahl der Nachkommastellen kann gewählt werden. Im Gegensatz zu Integerdaten wird bei Realdaten für das positive Vorzeichen ein Leerzeichen gedruckt.

Die Zahl `47.11` in Gleitkomma-Schreibweise (unterhalb der Abzählleiste):

`12345678901234567890123`	Nur Abzählleiste
`4.7110000000E+01`	Ohne Coprozessor, Datentyp Real
`4.71099999999860E+0001`	Mit Coprozessor, Datentyp Real
`4.71100000000000E+0001`	Mit Coprozessor, Datentyp Double

Zur Formatierung der Ausgabe:

Für die Formatierung der Ausgabe können die Schreibbreiten aller Ausdrücke und bei Real-Typen in der Fixkomma-Darstellung zusätzlich auch noch die Anzahl der Dezimalstellen eingegeben werden.

Für die beiden Formatier-Parameter *Schreibbreite* und *Anzahl der Dezimalstellen* sind Integer-Ausdrücke zulässig. Die Werte müssen >= 1 sein. Die beiden Parameter werden mit Doppelpunkt an den betreffenden Ausdruck der Ausgabeliste angehängt.

Die *Schreibbreite* wird immer ab der letzten Schreibstelle gezählt; bei neuen Zeilen ab Zeilenanfang.

Für das folgende Beispiel sei mit *x* ein Integer-Variable, mit *y* eine Real-Variable und mit *s* eine String-Variable angenommen.

Beispiel: `WriteLn(x:25, y:12:6, s:20);`

Die Zahlenwert von *x* wird rechtsbündig in ein Schreibfeld von 25 Stellen gedruckt, dann wird der Wert von *y* rechtsbündig in ein anschließendes Schreibfeld von 12 Stellen gedruckt, davon sind 6 Stellen Nachkommastellen, somit verbleiben für Vorkommastellen und Vorzeichen noch 5 Stellen. In ein anschließendes Schreibfeld von 20 Stellen wird der Wert des Strings *s* rechtsbündig gedruckt.

Wenn die *Schreibbreite* größer ist als die Anzahl der auszugebenden Zeichen, wird in das Feld rechtsbündig geschrieben.

Wenn die Anzahl der Zeichen größer ist als die vorgegebene Schreibbreite, dann werden zwar dennoch alle Zeichen ausgegeben, die Formatierung stimmt dann aber nicht mehr. Eine Warnung erfolgt nicht.

Wenn bei Real-Typen die Anzahl der Nachkommastellen fehlt, dann wird in Gleitkomma-Schreibweise ausgegeben.

In den folgenden Beispielen wird u.a. die Kreiszahl *Pi* in verschiedenen Formaten ausgegeben. Die Kreiszahl *Pi* wird durch die gleichnamige Turbo-Pascal-Standardfunktion dargestellt. In allen neun Beispielen wird zur Verdeutlichung der Formatierung ein vorangestellter und ein nachstehender Senkrechtstrich gedruckt. Anmerkung: Der Senkrechtstrich wird mit Alt+179 dargestellt.

```
program Pas07011;  { Demo Formatierung der Ausgabe }
begin
  WriteLn('1:12345678901234567890|');
  WriteLn('2:',  Pi, '|');
  WriteLn('3:', -Pi:8, -0.4711, '|');
  WriteLn('4:',  Pi:8:3, '|');
  WriteLn('5:',  4711,  4711, '|');
  WriteLn('6:', -4711, -4711, '|');
  WriteLn('7:', 'Anton Huber':15, '|');
  WriteLn('8:', 'Anton Huber':4, '|');
  WriteLn('9:12345678901234567890|');
end.
```

```
1:12345678901234567890|
2: 3.1415926536E+00|
3:-3.1E+00-4.7110000000E-01|
4:   3.142|
5:47114711|
6:-4711-4711|
7:    Anton Huber|
8:Anton Huber|
9:12345678901234567890|
```

Die Bildschirmausgabe des Programms "Pas07011.PAS"

Man beachte, daß bei fehlender Formatierung lückenlos hintereinander geschrieben wird. Lediglich bei Real-Typen wird für das nicht-abgedruckte positive Vorzeichen ein Leerzeichen gedruckt. Bei positiven Integer-Typen entfällt auch dieses Leerzeichen, wie Zeile 5 zeigt. Die Zeile 8 zeigt, daß trotz falscher Schreibbreite alle Zeichen ausgedruckt werden.

7.2 Die Standardprozeduren Read und ReadLn

Mit den beiden Standardprozeduren werden Daten von der Tastatur eingelesen und auf die im Aufruf genannten Variablen zugewiesen. Die eingegebenen Daten werden auf dem Bildschirm angezeigt. Die Eingabe ist mit der Eingabetaste *Enter (Return)* abzuschließen.

Der Unterschied zwischen den beiden Prozeduren besteht lediglich darin, daß der Cursor bei *Read* nach dem Einlesen der Daten in der Bildschirm-Eingabezeile verbleibt, wogegen er mit *ReadLn* (Read Line) nach dem Einlesen der Daten auf den Anfang der nächsten Bildschirmzeile gesetzt wird, egal wieviele Daten die Eingabezeile noch enthält. Wenn die Eingabezeile mehr Daten enthält als `Read` verlangt, dann werden beim nächsten *Read*-Aufruf diese Daten gelesen. Bei *ReadLn* ist dies nicht der Fall.

Formate:
```
Read(  v1 [, v2, ..., vn])
ReadLn(v1 [, v2, ..., vn])
ReadLn                                  Wartet auf Taste Enter
```
v1, v2, ..., vn Variablen

Die Variablen können beliebige Zeichenketten-Variablen (Typ String oder Char) oder numerische Variablen (alle Integer- und Realtypen) sein. Eine Mischung ist zulässig. Allerdings dürfen nach einer Zeichenketten-Variablen keine weiteren Variablen stehen. Somit können in einer Read-Anweisung auch keine zwei Strings eingelesen werden. Führende und nachstehende Leerzeichen werden bei Strings übernommen. Bei numerischen Eingaben werden Leerzeichen (ein oder mehrere) und Tabulatorsprünge lediglich als Trennzeichen zwischen mehreren Daten interpretiert. **Das Komma darf somit bei der Eingabe nicht als Trennzeichen benutzt werden.**

Beispiel: `ReadLn(x);`
Die Tastatureingabe wird auf die (deklarierte) Variable `x` zugewiesen.

Für ein benutzerfreundliches Programm ist es unbedingt erforderlich, daß vor *Read* auf dem Bildschirm ein Hinweis über die folgende Eingabe erscheint. Dazu nimmt man zweckmäßigerweise die Prozedur *Write*, wenn der Cursor nach dem Hinweistext stehen bleiben soll.

Beispiel:
```
Write('Eingabe Rechnungsbetrag und Name: ');
ReadLn(Betrag, Name);
```

Demo-Programm:

```
program Pas07021;    { Demo Read und ReadLn }
var
  a, b, c, d: Integer;
  s:          string[10];

begin
  WriteLn('Geben Sie 6 Zahlen wie in der folgenden Zeile ein: ');
  WriteLn('1 2 3 4 5 6');
```

```
  Read(a, b, c, d);
  WriteLn(a:2, b:2, c:2, d:2);  { Die Ausgabe:  1 2 3 4  }

  WriteLn('Geben Sie 6 Zahlen wie in der folgenden Zeile ein: ');
  WriteLn('1 2 3 4 5 6');
  ReadLn(a, b, c, d);
  WriteLn(a:2, b:2, c:2, d:2);  { Die Ausgabe:  5 6 1 2
                                  da noch 2 Daten von Read frei !! }

  WriteLn('Geben Sie die folgende Zeile ein:');
  WriteLn('4711 Anton Huber');
  ReadLn(a, s);
  WriteLn(a, s);  { Die Ausgabe: 4711 Anton Hub
                    da der String s auf 10 Zeichen deklariert ist
                    und das Leerzeichen vor dem "Anton" mitzählt.  }
end.
```

```
Geben Sie 6 Zahlen wie in der folgenden Zeile ein:
1 2 3 4 5 6
1 2 3 4 5 6
 1 2 3 4
Geben Sie 6 Zahlen wie in der folgenden Zeile ein:
1 2 3 4 5 6
1 2 3 4 5 6
 5 6 1 2
Geben Sie die folgende Zeile ein:
4711 Anton Huber
4711 Anton Huber
4711 Anton Hub
```

Die Bildschirmausgabe des vorstehenden Programms

7.3 Die Standardfunktion ReadKey

Mit der Funktion *ReadKey* wird *ein* Zeichen ohne abschließendes *Return* von der Tastatur gelesen. Das Ergebnis dieser Funktion hat den Datentyp *Char* (character). Genau genommen wird mit *ReadKey* der Pufferspeicher der Tastatur gelesen. Wenn der Tastaturpuffer leer ist, dann wartet das Programm auf einen Tastendruck. Siehe auch Funktion *KeyPressed*.

ReadKey benötigt die Unit *CRT*.

Format: `ReadKey`

Im Gegensatz zu *den Prozeduren Read* und *ReadLn* wird das Zeichen nicht auf dem Bildschirm angezeigt. Sollte das gewünscht sein, ist das Zeichen mit *Write* oder *WriteLn* auszugeben, was in den meisten Fällen sinnvoll ist.

Beispiel:
```
Write('Drücken Sie eine Taste: ');
WriteLn(ReadKey);
```

Vorgriff: Spezielle Tasten (z.B. die Cursortasten und die Funktionstasten und auch Tastenkombinationen mit der Alt-Taste) liefern einen sogenannten Doppelcode. Der erste Code eines Doppelcodes ist immer das Null-Byte, *Chr(0)*, *#0*. Der zweite Code ist der Scan-Code der betreffenden Taste. Doppelcodes werden im Kapitel "Programmierung der Tastatur" behandelt.

Mit *ReadKey* kann man Tastatureingaben absichern. Im folgenden Demo-Programm wird ein Vorgriff auf die repeat/until-Schleifenkonstruktion gemacht.

```
program Pas07031;  { Demo Tasteneinzug mit ReadKey  }

uses
  CRT;             { Unit CRT für ReadKey notwendig }

var
  Zeichen: Char;

begin
  Write('Drücken Sie die Taste <j>: ');
  repeat                { Wiederhole solange, bis .....  }
    Zeichen := ReadKey;
  until (Zeichen = 'j'); { bis Taste 'j' gedrückt wird  }
  WriteLn(Zeichen);
end.
```

7.4 Die Standardfunktion KeyPressed

Die Standardfunktion *KeyPressed* stellt fest, ob eine Taste gedrückt wurde, genauer: ob sich im Tastaturpuffer noch ein Zeichen befindet. Das Ergebnis der Funktion hat den Datentyp *Boolean* und kann somit nur *True* oder *False* sein. *KeyPressed* liest aber nicht das Zeichen aus dem Tastaturpuffer. Dazu dient die Standardfunktion *ReadKey*.

KeyPressed benötigt die Unit *CRT*.

Format: `KeyPressed`

Es werden nur Tasten geprüft, die lesbare Zeichen erzeugen; Tasten wie *Umsch*, *Strg*, *Num*, Funktionstasten usw. somit nicht alleine sondern nur in Verbindung mit einer anderen Taste, die ein lesbares Zeichen ergibt.

Bei den folgenden Demo-Programmen wird wieder ein Vorgriff auf die repeat/until-Schleifenkonstruktion gemacht.

```
program Pas07041;    { Demo Funktion KeyPressed }
uses
  CRT;     { Unit CRT für Funktion KeyPressed notwendig }
begin
  ClrScr;
  Write('Ich warte auf einen Tastendruck ');
  Write('oder bis zum Stromausfall . ... ');
  repeat
  until KeyPressed;
end.
```

Zum Leeren des Tastaturpuffers:

Bei längeren Programmläufen kann es vorkommen, daß der Anwender mit der Tastatur "spielt", d.h. versehentlich Zeichen im Tastaturpuffer gespeichert werden. Bei einem späteren *ReadKey* wird ein gespeichertes Zeichen ohne Warten ausgelesen. Unter Umständen kann das Programm einen völlig ungewollten Ablauf nehmen. Deshalb sollte man in diesen Fällen den Tastaturpuffer leeren und erst dann das gewünschte Zeichen einziehen. Das folgende Programmschema zeigt die Vorgehensweise unter Vorwegnahme der while- und repeat/until-Schleifen und noch einiger nicht erklärter Begriffe:

```
...
uses
  CRT;    { wegen ReadKey und KeyPressed }

var
  Ch: Char;
...
begin
  ...   { Hier sei ein langer Programmlauf ... }
  ...   { ... und der Anwender "spielt" mit der Tastatur }

  while KeyPressed do
    Ch := ReadKey;  { Zuerst Tastaturpuffer leeren ...}

  Write('Eingabe (j/n): ');

  repeat
    Ch := ReadKey; { ... dann erst Zeichen einziehen }
  until (Ch = 'j') or (Ch = 'n');

  if Ch = 'j'
     then ....
     else ....;
  ...
end.
```

7.5 Die Ausgabe auf den Drucker

Für die Druckerausgabe wird die Unit *PRINTER* benötigt. Ansonsten werden die gleichen Prozeduren wie für die Bildschirmausgabe benutzt, also *Write* oder *WriteLn*, die lediglich mit der Drucker-Dateivariablen "***Lst***" zu ergänzen sind. Die Eigenschaften

der beiden Prozeduren bezüglich Zeilenvorschub und Formatierung gelten auch für den Drucker. Das folgende Beispiel zeigt die Ausgabe auf den Drucker:

```
program Pas07051;          { Demo Drucker-Test }
uses
  PRINTER;                 { Unit PRINTER für Druckerausgaben }
begin
  WriteLn(Lst, 'Anton Huber'); { Dateivariable »Lst« für Drucker }
  WriteLn(Lst);               { Leerzeile auf Drucker }
end.
```

Ein Seitenvorschub wird auf dem Drucker mit

`Write(Lst, #12);` oder mit `Write(Lst, Chr(12));`

erreicht. Mit *#12* wird das Zeichen Nr 12 eingegeben. Bei diesem Zeichen handelt es sich um das Steuerzeichen Nr. 12 nach Ascii (Vorgriff), das für *Form Feed* (FF) bestimmt ist.

7.6 Die Standardprozedur ClrScr

Die Prozedur *ClrScr* (Clear Screen) löscht den Bildschirm und setzt den Cursor in die linke obere Ecke des Bildschirm-Fensters.

ClrScr benötigt die Unit *CRT*.

Format: `ClrScr`

Diese Prozedur hat keine Parameter.

Turbo-Pascal speichert Bildschirmausgaben. Wenn man die früheren Bildschirmausgaben nicht sehen möchte, dann den Bildschirm vor der ersten Ausgabe mit *ClrScr* löschen.

7.7 Die Standardprozedur GotoXY

Die Prozedur *GotoXY* versetzt den Cursor in die Spalte `X` der Zeile `Y`.

GotoXY benötigt die Unit *CRT*.

Format: `GotoXY(`*`spalte`*`, `*`zeile`*`)`

`spalte`	Ausdruck für Spaltenposition, Datentyp Byte
`zeile`	Ausdruck für Zeilenposition, Datentyp Byte

Für das Standard-Textfenster gilt: *`spalte`*`: 1..80`, *`zeile`*`: 1..25`

Wenn auch nur einer der beiden Ausdrücke ungültig ist, dann wird der Cursor nicht versetzt.

Die Prozedur *GotoXY* arbeitet relativ zum gesetzten Textfenster, siehe auch Standardprozedur *Window*.

Im folgenden Demo-Programm wird die Standardfunktion *Random* vorweggenommen.

```
program Pas07071;     { Demo Cursorpositionierung mit GotoXY       }

uses
  CRT;                { Unit CRT für ClrScr, GotoXY und KeyPressed }

const
  SpMin = 10; SpMax = 70;
  ZeMin =  5; ZeMax = 22;

var
  Sp, Ze: Byte;

begin
  ClrScr;            { Bildschirm löschen }
  GotoXY(SpMin, ZeMin - 2);     { Cursor in Spalte und Zeile }
  WriteLn('Demo: Cursorpositionierung');
  GotoXY(SpMin, ZeMin - 1);
  WriteLn('Ende mit beliebigem Tastendruck');

  repeat
    Sp := SpMin + Random(SpMax - SpMin + 1); { Standardfunkt. Random }
    Ze := SpMin + Random(ZeMax - ZeMin + 1); { liefert Zufallszahlen }
    GotoXY(Sp, Ze);
    Write('*');
  until KeyPressed;
end.
```

7.8 Die Standardfunktionen WhereX und WhereY

Die Standardfunktion *WhereX* liefert die momentane Spaltenposition des Cursors, *WhereY* liefert dagegen die momentane Zeilenposition. Das Ergebnis hat den Datentyp Byte. Wenn mit der Standardfunktion *Window* ein Textfenster gesetzt ist, dann sind die Ergebnisse von *WhereX* und *WhereY* Relativ-Koordinaten zum gesetzten Textfenster.

WhereX und *WhereY* benötigen die Unit *CRT*.

Format:
```
WhereX
WhereY
```

```
program Pas07081;    { Demo WhereX und WhereY }
uses
  CRT;
begin
  ClrScr;
  Write('*****');
  GotoXY(WhereX - 3, WhereY);  { Cursor in der gleichen Zeile
                                 um 3 Spalten zurück          }
  WriteLn('!');     { und das mittlere Zeichen überschreiben  }
  ReadLn;
end.
```

7.9 Die Standardprozedur Window

Die Standardprozedur *Window* (nicht zu verwechseln mit *Windows*) definiert einen Ausschnitt des Bildschirms als Textfenster.

Window benötigt die Unit *CRT*.

Format: `Window(x1, y1, x2, y2)`

`x1, y1, x2, y2`		Ausdrücke mit Datentyp Byte
`x1`	Erste Bildschirmspalte,	Minimalwert 1
`y1`	Erste Bildschirmzeile,	Minimalwert 1
`x2`	Letzte Bildschirmspalte,	Maximalwert 80
`y2`	Letzte Bildschirmzeile,	Maximalwert 25

Alle Koordinaten von *GotoXY*, *WhereX* und *WhereY* sind Relativ-Koordinaten zum zuletzt mit *Window* gesetzten Fenster.

Beim Aufruf von *Window* wird der Cursor in die linke obere Ecke des Fensters versetzt. Die Standardprozedur *ClrScr* wirkt nur für das aktuelle Fenster.

7.10 Die Standardprozeduren HighVideo, LowVideo und NormVideo

Mit den Prozeduren *HighVideo* wird die Bildschirmintensität für alle folgenden Ausgaben erhöht, so daß die Zeichen heller als normal erscheinen. Mit *LowVideo* erscheinen die Zeichen dunkler als normal. Mit *NormVideo* wird die normale Intensität (Standard) eingestellt. Bei Farbbildschirmen sind die Prozeduren *TextColor* und *TextBackground* interessanter, siehe nächsten Punkt.

HighVideo, *LowVideo* und *NormVideo* benötigen die Unit *CRT*.

Formate: `HighVideo`
`LowVideo`
`NormVideo` (wie HighVideo)

```
program Pas07101;    { Demo Bildschirmintensitäten }
uses
  CRT;               { Unit CRT für Bildschirmintensitäten }
begin
  ClrScr;
             WriteLn('Das ist normale Intensität.');
  HighVideo; WriteLn('Das ist hohe Intensität.');
  LowVideo;  WriteLn('Das ist geringe Intensität.');
  NormVideo; WriteLn('Das ist wieder normale Intensität.');
  ReadLn;
end.
```

7.11 Die Standardprozeduren TextMode, TextColor und TextBackground

Mit diesen Prozeduren können der Textmodus (Schwarz/Weiß- und Farb-Bildschirme), die Farben und Attribute des Textes (Farbbildschirm-Textfarbe, Farbbildschirm-Hintergrundfarbe), blinkende Zeichen,) eingestellt werden. Diese Prozeduren benötigen die Unit *CRT*.

Der Textmodus, die Vordergrund- und die Hintergrundfarben werden sind durch Zahlen festgelegt (Textmodus Datentyp Word, Bildschirmfarben Datentyp Byte). An Stelle der Zahlen können jedoch auch die Bezeichner benutzt werden, die in der Unit *CRT* für diese Zwecke vordefiniert sind. Siehe folgende Demo-Programme.

```
program Pas07111;  { Demo Standardprozedur "TextMode" }
uses
  CRT;
begin
  { Hinweise: - Es kann immer nur  e i n  Text-Mode wirksam sein.
                Die nichtzutreffenden in diesem Programm mit
                Kommentarklammern ausschalten.
              - Der Text-Mode kann sein: 0, 1, 2, 3, 7 oder 256.
                Statt dieser Zahlen (Datentyp Word) können auch
                die in der Unit "CRT" vordefinierten Konstanten-
                bezeichner "BW40", "Co80", "BW40", "Co80", "Mono"
                und "Font8x8" benutzt werden. Alles Steinzeit!
              - Ab Win95 mit "Alt+Enter" auf Vollbild umschalten.
  }
 (*
  TextMode(BW40);    { 40 Zeichen in einer Zeile, 25 Zeilen }
  WriteLn('BW40 = 0: CGA, 40 * 25, monochrom');

  TextMode(Co40);
  WriteLn('Co40 = 1: CGA, 40 * 25, farbig');
```

```
  TextMode(BW80);    { 80 Zeichen in einer Zeile, 25 Zeilen }
  WriteLn('BW80 = 2: CGA, 80 * 25, monochrom');

  TextMode(Co80);
  WriteLn('Co80 = 3: CGA, 80 * 25, farbig');

  TextMode(Mono);
  WriteLn('Mono = 7: CGA, 80 * 25, monochrom, Monochrom-Adapter');
 *)
  TextMode(Font8x8 + Co80);  { Font8x8 nur mit einer Addition }
  WriteLn('Font8x8 = 256: 50 Zeilen VGA (43 Zeilen EGA)');

  ReadLn;
end.
```

```
program Pas07112;  { Demo Standardprozeduren "TextColor"
                     und "TextBackground" }
uses
  CRT;
var
  Vordergrund,
  Hintergrund: Byte;

begin
  repeat
    ClrScr;
    GotoXY(10,  1); Write('Demo Bildschirmfarben. Ende mit 0 0');
    GotoXY(10,  3); Write('Vordergrund             Hintergrund  ');
    GotoXY(10,  4); Write('------------------      -------------');
    GotoXY(10,  5); Write(' 0 = Black              0 = Black    ');
    GotoXY(10,  6); Write(' 1 = Blue               1 = Blue     ');
    GotoXY(10,  7); Write(' 2 = Green              2 = Green    ');
    GotoXY(10,  8); Write(' 3 = Cyan               3 = Cyan     ');
    GotoXY(10,  9); Write(' 4 = Red                4 = Red      ');
    GotoXY(10, 10); Write(' 5 = Magenta            5 = Magenta  ');
    GotoXY(10, 11); Write(' 6 = Brown              6 = Brown    ');
    GotoXY(10, 12); Write(' 7 = LightGray          7 = LightGray');
    GotoXY(10, 13); Write(' 8 = DarktGray          -------------');
    GotoXY(10, 14); Write(' 9 = LightBlue   ');
    GotoXY(10, 15); Write('10 = LightGreen  ');
    GotoXY(10, 16); Write('11 = LightCyan   ');
    GotoXY(10, 17); Write('12 = LightRed    ');
    GotoXY(10, 18); Write('13 = LightMagenta');
    GotoXY(10, 19); Write('14 = Yellow      ');
    GotoXY(10, 20); Write('15 = White       ');
    GotoXY(10, 21); Write('128= Blink       ');
    GotoXY(10, 22); Write('-----------------');
```

```
    GotoXY(10, 23);  ReadLn(Vordergrund); { Der Einfachheit halber }
    GotoXY(34, 14);  ReadLn(Hintergrund); { ohne Fehlerprüfung     }

    TextColor(Vordergrund);
    TextBackground(Hintergrund);

  until (Vordergrund = Black) and (Hintergrund = Black);
end.
```

7.12 Die Standardprozeduren ClrEoL, DelLine und InsLine

- Mit *ClrEoL* (Clear End of Line) werden alle Zeichen ab der momentanen Cursorposition bis zum Zeilenende gelöscht.
- Mit *DelLine* (Delete Line) wird die Zeile, in der sich der Cursor befindet, gelöscht. Alle darunter liegenden Zeilen werden um eine Zeile nach oben geschoben.
- Mit *InsLine* (Insert Line) wird die Zeile, in der der Cursor steht und alle nachfolgenden Zeilen nach unten geschoben und an Stelle der alten Cursorzeile eine Leerzeile eingefügt. Mit einer **trickhaften Anwendung von *InsLine*** gelingt es, in der **25. Bildchirmzeile auch die 80. Bildschirmspalte zu bedrucken**, ohne daß nach der Ausabe des 80. Zeichens der Bildschirm gescrollt wird. Man schreibt den Text zuächst mittels *GotoXY* in die 24. Zeile und "schiebt" diese Zeile dann mit *InsLine* in die 25. Zeile.

Alle drei Prozeduren benötigen die Unit *CRT*.

Formate:
```
ClrEoL
DelLine
InsLine
```

Wenn mit *Window* ein Fenster definiert wurde, dann beziehen sich die Aktionen aller drei Prozeduren auf das aktuelle Fenster, sonst auf den gesamten Bildschirm.

7.13 Die Standardprozedur Delay

Die Prozedur *Delay* bewirkt eine Verzögerung der Programmausführung für eine wählbare Zeit (Zeitschleife).

Delay benötigt die Unit *CRT*

Format: `Delay(`*zeit*`)`

zeit Ausdruck. Datentyp Word,
Zeit in Millisekunden (0..65535)

```
program Pas07131;    { Demo Delay }

uses
  CRT;

const
  Piep = #7;     { Steuerzeichen #7: Beep, Bell, Pfeifton }

begin
  ClrScr;
  WriteLn(Piep, 'Das Programm ist in ca. 5 Sekunden beendet');
  Delay(5000);   { 5000 Millisekunden warten }
  WriteLn(Piep);
end.
```

7.14 Die Standardprozeduren Sound und NoSound

Mit der Prozedur *Sound* wird auf dem eingebauten Lautsprecher ein Ton mit einer wählbaren Frequenz ausgegeben. Der Ton wird solange erzeugt, bis er mit der Standardprozedur *NoSound* wieder abgestellt wird.

Beide Prozeduren benötigen die Unit *CRT*.

Format: `Sound(`*`frequenz`*`)`

`frequenz`	Ausdruck, Datentyp Word Frequenz in Hertz, 0..65535, aber durch Lautsprecher unten und oben unbegrenzt.

`NoSound`

```
program Pas07141;  { Demo Sound, Vorgriff Funktion Random }
uses
  CRT;
const
  fMin =   40;
  fMmax = 4000;
begin
  ClrScr;
  WriteLn('Den Ohrenschmaus mit einem Tastendruck beenden');
  Sound(440);        { Kammerton a (440 Hertz )  }
  Delay(2000);       { mit ca. 2 Sekunden Dauer  }
  repeat
    Sound(fMin + Random(fMax - fMin + 1));  { Zufallstöne }
    Delay(200);      { Dauer ca. 0.2 Sekunden }
  until KeyPressed;
  NoSound;
end.
```

7.15 Die Standardprozeduren SetDate, SetTime, GetDate und GetTime

Diese Prozeduren haben zwar nicht unmittelbar mit Ein- und Ausgaben zu tun, sollen aber dennoch an dieser Stelle abgehandelt werden.

- Mit *SetDate* kann das Datum des Systems gesetzt werden, ähnlich wie mit *DATE* im Betriebssystem.
- Mit *SetTime* kann die Uhrzeit des Systems gesetzt werden, ähnlich wie mit *TIME* im Betriebssystem.
- Mit *GetDate* wird das aktuelle Datum an Variablen geliefert.
- Mit *GetTime* wird die aktuelle Uhrzeit an Variablen geliefert.

Alle vier Prozeduren benötigen die Unit *DOS*. Alle Parameter und Variablen müssen den Datentyp Word haben und müssen bei *SetDate* und *SetTime* in später angegebenen Gültigkeitsbereich liegen.

Format: `SetDate(jahr, monat, tag, wochentag)`

jahr	Numerischer Ausdruck. Gültig: 1980..2099
monat	Numerischer Ausdruck. Gültig: 1..12
tag	Numerischer Ausdruck. Gültig: 1..31
wochentag	Numerischer Ausdruck. Gültig: 0..6, wobei 0 für Sonntag steht.

Format: `SetTime(hh, mm, ss, ss100)`

hh	Numerischer Ausdruck für Stunden. Gültig: 0..23
mm	Numerischer Ausdruck für Minuten. Gültig: 0..59
ss	Numerischer Ausdruck für Sekunden. Gültig: 0..59
ss100	Numerischer Ausdruck für Hunderstel-Sekunden. Gültig: 0..99

Format: `GetDate(jahr, monat, tag, wochentag)`

jahr	Variable. Ergebnis: 1980..2099
monat	Variable. Ergebnis: 1..12
tag	Variable. Ergebnis: 1..31
wochentag	Variable. Ergebnis: 0..6, wobei: 0 = Sonntag

Format: `GetTime(hh, mm, ss, ss100)`

hh	Variable für Stunden.	Ergebnis: 0..23
mm	Variable für Minuten.	Ergebnis: 0..59
ss	Variable für Sekunden.	Ergebnis: 0..59
ss100	Variable für Hunderstel-Sekunden.	Ergebnis: 0..99

Die Parameter bei *SetDate* und *SetTime* werden im Normalfall als Konstanten eingegeben. Wenn auch nur *ein* Parameter außerhalb des Gültigkeitsbereiches liegt, wird die Prozedur nicht ausgeführt.

```
program Pas07151;  { Demo Datum und Uhrzeit }

uses
  CRT, DOS;     { Unit CRT für Delay, Unit DOS für Datum und Zeit }

const
  Dauer = 1000;

var
  Jahr, Monat, Tag, Wochentag,
  Stunden, Minuten, Sekunden, Sekunden_100: Word;
  Zeit_1, Zeit_2, Zeit:                     Real;

begin
  ClrScr;
  GetDate(Jahr, Monat, Tag, Wochentag);
  Writeln('Das Jahr:      ', Jahr);
  Writeln('Der Monat:     ', Monat);
  Writeln('Der Tag:       ', Tag);
  Writeln('Der Wochentag: ', Wochentag, '   (0 = Sonntag)');

  GetTime(Stunden, Minuten, Sekunden, Sekunden_100);
  Zeit_1 := Stunden*3600.0 + Minuten*60 + Sekunden + Sekunden_100/100;
  Delay(Dauer);
  GetTime(Stunden, Minuten, Sekunden, Sekunden_100);
  Zeit_2 := Stunden*3600.0 + Minuten*60 + Sekunden + Sekunden_100/100;
  Zeit   := Zeit_2 - Zeit_1;

  Writeln('Die Zeitschleife Delay mit ', Dauer, ' Millisekunden');
  Writeln('hat', Zeit:6:3, ' Sekunden gedauert.');
  ReadLn;
 end.
```

Die Ausgabe am Dienstag, den 14. März 2000:

```
Das Jahr:      2000
Der Monat:     3
Der Tag:       14
Der Wochentag: 2   (0 = Sonntag)
Die Zeitschleife Delay mit 1000 Millisekunden
hat 0.980 Sekunden gedauert.
```

7.16 Die Standardprozedur Halt

Die Prozedur *Halt* bricht die Programmausführung ab. Die Prozedur ist i.a. nur in Verbindung mit einer Bedingung, z.B. bei einem nicht behebbaren Fehlerfall) oder für Programmtestzwecke sinnvoll. Optional kann ein Exitcode übergeben werden. Für Tests sollte man aber statt *Halt* die Möglichkeiten des integrierten Debuggers (Setzen von

Abbruchpunkten oder schrittweise Ausführung, Variablenabfrage an den Haltepunkten mit *Strg+F4*, siehe Kap. 5.7) nutzen.

Format: `Halt[(`*`exitcode`*`)]`

Beispiele:		
	`Halt;`	Kein Exitcode = 0
	`Halt(0);`	Exitcode 0
	`Halt(11);`	Exitcode 11

Der optionale Exitcode ist ein Ausdruck mit den Datentyp *Word*, man sollte aber nur Werte aus dem Byte-Bereich 0..255 verwenden. Wenn der Exitcode fehlt, wird standardmäßig 0 angenommen, also: `Halt(0)`.

Der Exitcode ist dann von Bedeutung, wenn das compilierte Pascal-Programm innerhalb eines Batch-Programms aufgerufen wird; siehe DV II, Kap. 1: Betriebssystem MS-DOS. Bei fehlerfreier Ausführung übergibt man sinnvollerweise den Exitcode 0. Ansonsten hat der Programmierer des Batch-Programms auf die Festlegungen des Programmieres des Pascal-Programms einzugehen. Im Batch-Programm kann mit *errorlevel* auf den Exitcode zugegriffen werden.