4 Allgemeines zu Pascal und Turbo-Pascal

Einleitende Programmbeispiele in Turbo-Pascal
Aufbau eines Turbo-Pascal-Programms
Reservierte Wörter. Bezeichner. Formatbeschreibung
Programmbeispiel Zahlenumwandlung
Register der Turbo-Pascal-Begriffe

Gliederung

4.1 Allgemeines zu Pascal und Turbo-Pascal 2

4.2 Einleitende Programmbeispiele in Turbo-Pascal 3

4.3 Aufbau eines Pascal-Programms 7

4.4 Reservierte Wörter 8

4.5 Selbstdefinierte Bezeichner und Standardbezeichner 9

4.6 Die Begriffe Konstante, Variable und Ausdruck 10

4.7 Zur Formatbeschreibung 11

4.8 Programmbeispiel Zahlenumwandlung 12

4.9 Zur Gestaltung von Pascal-Quelltexten 14

4.10 Register der Turbo-Pascal-Begriffe 19

4.1 Allgemeines zu Pascal und Turbo-Pascal

Wie bereits erwähnt, wurde die Sprache Pascal um 1970 von Prof. Dr. NikolausWirth an der Eidgenössischen Technischen Hochschule Zürich (ETH Zürich) als Ausbildungssprache entwickelt. Das Ziel war, die Studenten zu einem neuen Programmierstil, dem strukturierten Programmieren, anzuleiten. Pascal war somit nicht als Brotsprache konzipiert. Dennoch hat sich die Sprache relativ schnell durchgesetzt.

Neben einem zwischenzeitlich definierten Standard-Pascal haben sich viele Pascal-Dialekte herausgebildet. Um 1970 gab es noch keine Mikrocomputer im heutigen Sinne. Der Befehlsvorrat von Standard-Pascal orientiert sich mehr an dem früheren Stand der Rechentechnik (Batch-Betrieb, Eingabe mit Lochkarten), so daß Standard-Pascal mehr als Referenz dient als zur praktischen Anwendung. Mit den aufkommenden Mikrocomputern hat Basic eine starke Verbreitung erlangt. Diese Sprache stand auch bei Mikrocomputern mit kleinem Arbeitspeicher zur Verfügung, war in den meisten Fällen fest im ROM abgelegt, war leicht erlernbar und einfach in der Anwendung, zumindest im Vergleich mit den damaligen Pascal-Compilern.

Der große Durchbruch von Pascal im Bereich der Mikrocomputer begann um etwa 1984, als das amerikanische Software-Haus Borland ein neuartiges Pascal-Entwicklungssystem, das Turbo-Pascal, herausbrachte. Der Quelltext, der Editor, der Compiler und das Compilat stehen bei Turbo-Pascal immer im Arbeitsspeicher. Wenn beim Compilieren ein Fehler entdeckt wird, wird sofort der Editor aktiviert und der Quelltext mit einer Fehlermarkierung angezeigt. Mit Turbo-Pascal kann man fast genau so interaktiv wie bei einem Interpreter-Basic Programme entwickeln. Optional kann das compilierte Programm auch auf der Platte/Diskette gespeichert werden. Das so compilierte Programm hat die Extension .EXE und kann außerhalb der Turbo-Pascal-Entwicklungssystems und ohne ein zusätzliches Laufzeit-Modul benutzt werden. Der Erfolg von Turbo-Pascal hat zu vielen Nachahmungen geführt.

Die Versionen von Turbo-Pascal: 1.0, 2.0, 3.0, 4.0 (4.0 erstmal mit kompletter Entwicklungumgebung, Integrated Development Enviroment, IDE), 5.0, 5.5, 6.0 (6.0 erst mit Mausbedienung des Editors) und 7.0 (1993). Zusammen mit Version 7.0 erschien die Windows-Version Borland Pascal 7.0. Mit dieser großen Version war es möglich, Pascal-Programme für den DOS-Real-Mode, den DOS-Protected-Mode (wichtig für große Programme) und für Windows zu entwickeln. Die große Version wurde wegen der Schwierigkeiten mit der Windows-Programmierung ein Flop. Wesentlich besser sieht es mit der 1995 erschienen reinen Windows-Version **Delphi** aus, mit der – ähnlich wie mit **Visual Basic** (VB) von Microsoft – Windows-Anwendungen in weiten Teilen visuell entwickelt werden können. Es bleibt abzuwarten, ob Delphi den Vorsprung von VB aufholen kann. Delphi setzt fast den gesamten Sprachumfang von Turbo-Pascal

vorraus. DOS-Anwendungen können mit Delphi nicht entwickelt werden, es sei denn, man mißbraucht Delphi nur als Editor.

Turbo-Pascal besitzt gegenüber Standard-Pascal sehr viele Erweiterungen. Allerdings darf nicht übersehen werden, daß Turbo-Pascal auf das Betriebssystem MS-DOS und somit auf die Intel-Prozessoren 8088, 8086, 80x86 und Pentium abgestimmt ist. Die Nutzung der Spracherweiterungen erschwert die Übertragung des Programms in der Quellcode-Form auf Rechnersysteme mit anderen Prozessoren und Betriebssystemen. In der Regel muß der Quellcode geändert werden.

Im Rahmen der Ausbildung im Studiengang Druckereitechnik werden die Erweiterungen von Turbo-Pascal im sinnvoll erscheinenden Umfang genutzt. Auf die Unterschiede zu Standard-Pascal wird nur in Sonderfällen hingewiesen.

Hinweis: Das Turbo-Pascal-System enthält eine **Menü-Version** von Turbo-Pascal (Dateiname Turbo.EXE), als auch eine **Kommandozeilen-Version** (Dateiname TPC.EXE). Beide Versionen sind inhaltlich identisch, unterscheiden sich in der äußeren Erscheinungsform gänzlich. Die Menü-Version bedient sich einer anwenderfreundlichen Entwicklungsumgebung, die Kommandozeilen-Version ähnelt in der Handhabung mehr den klassischen Compilern. Die Kommandozeilen-Version wird z.B. für die Entwicklung größerer Programme gebraucht, die Programm-Module aus anderen Programmiersprachen enthalten. Die Kommandozeilen-Version wird hier nicht behandelt; auf die Turbo-Pascal-Handbücher wird verwiesen.

4.2 Einleitende Programmbeispiele in Turbo-Pascal

1. Beispiel:

```
program Hallo;
begin
  WriteLn('Hallo Welt,');
  WriteLn('das ist das erste Programm von Anton Huber');
end.
```

Dieses Programm benutzt die Pascal-Standardprozedur `WriteLn` und schreibt die in Hochkommata eingeschlossene Zeichenfolgen (Zeichenkette, engl. string) auf den Bildschirm. Man beachte den Punkt nach **`end`**. Er zeigt den Compiler das Programmende an. Man achte auch auf die Semikolons, die Programmanweisungen abschließen. Die hier fettgedruckten Wörter **`program`**, **`begin`** und **`end`** sind sog. reservierte Wörter.

Nach der Eingabe des Quelltextes und dem Start mit **Strg**+**F9** (Taste Strg gedrückt halten und dann Funktionstaste F9 drücken) zeigt der Bildschirm folgendes an:

```
Hallo Welt,
das ist das erste Programm von Anton Huber
```

Die Anzeige verschwindet so schnell wieder und das System kehrt zum Editor zurück, daß die Anzeige auch vom schnellsten Auge nicht wahrgenomen wird. Mit **Alt+F5** (Taste **Alt** gedrückt halten und dann die Funktionstaste **F5** drücken) kann der (DOS-) Ausgabebildschirm wieder angezeigt werden. Mit nochmaligen Alt+F5 kehrt man zum Editor zurück. Später werden andere Methoden gezeigt, um das lästige Umschalten zu umgehen.

2. Beispiel:

```
begin end.
```

Das kürzeste Pascal-Programm der Welt! Es tut gar nichts und ist denoch korrekt. Es zeigt, daß Pascal formatfrei ist und daß der Programmkopf mit dem reserviertenWort **`program`** in Turbo-Pascal optional ist. Der Verfasser ist der Meinung, daß ein Programmkopf schon angegeben werden sollte. In Standard-Pascal wäre noch mehr zu tun.

3. Beispiel:

```
program Test3;

var
  x: Integer;
  y: Real;

begin
  x := 4711;
  y := 47.11;
  WriteLn(x, y);
end.
```

Dieses Programm schreibt die Ganzzahl (Integerzahl) 4711 und die Kommazahl (Realzahl) 47.11 nebeneinander auf den Bildschirm. Es zeigt, daß in Pascal alle Variablen deklariert werden müssen und zwar nach Datentypen getrennt. Die Belegung einer Variablen mit einem Wert erfolgt von rechts nach links. Das Symbol für die Zuweisung (Zuweisungsoperator "**`:=`**") besteht aus den Zeichen Doppelpunkt und Gleichheits-

zeichen. Dazwischen darf kein Leerzeichen (engl. blank, space) stehen. Davor und danach dürfen Leerzeichen stehen. Der besseren Lesbarkeit halber sollte man Leerzeichen setzen. Regel: Wo ein Leerzeichen stehen darf, können beliebig viele stehen. Hinweis: In den Programmiersprachen Basic und C wird das Gleichheitszeichen alleine als Zuweisungssymbol benutzt. Die Formatfreiheit von Pascal benutzt man dazu, durch sinnvoll angeordnete Leerzeichen und Leerzeilen eine optisch-logische Struktur in den Quelltext zu bringen.

4. Beispiel:

```
program Test4;

var
  x:    Integer;
  Name: string[20];  { Zeichenkette mit max. 20 Zeichen }

begin
  x    := 4711;
  x    := x + 1;     { Zuweisung von rechts nach links  }
  Name := 'Anton Huber.';   { Auch hier eine  Zuweisung }
  Write(Name, ' Der Wert von x: ', x);
end.
```

Die Ausgabe des Programms: `Anton Huber. Der Wert von x: 4712`

In das Programm können erklärende Kommentare geschrieben werden. Sie sind in geschweifte Klammern zu setzen. Kommentare haben keinen Einfluß auf den Programmablauf, sie werden beim Compilieren ignoriert. Als Ersatzzeichen für die geschweiften Klammern können auch runde Klammern mit einem nachfolgenden bzw. vorausgehenden Stern benutzt werden.

Kommentarklammern: **{ }** oder Ersatzzeichen **(* *)**

Kommentare können über beliebig viele Zeilen gehen. Nach dem Kommentarbeginn dürfen Anweisungen stehen; sie werden nicht ausgeführt**. Kommentarklammern müssen immer paarweise auftreten und dürfen nicht verschachtelt werden.** Bei der ersten schließenden Klammer wird der Kommentar als beendet betrachtet; ein Verstoß gegen die Vorgabe führt in der Regel zu irreführenden Fehlermeldungen des Compilers und deshalb zu langwieriger Fehlersuche. **Einzige Ausnahme** ist die Kommentarklammerung mit den Ersatz-zeichen; in dieser Klammerung dürfen geschweifte Klammern enthalten sein.

Hinweis: In **Basic** und in **PostScript** kennt man nur ein einleitendes Kommentarsymbol. Nach diesem Symbol wird nur der Rest der Zeile als Kommentar betrachtet.

In den bisherigen Programmbeispielen wurden die reservierten Wörter **program, var, string, begin** und **end** benutzt. Zur besseren Erkennung werden die reservierten Wörter im Skriptum durch fettgedruckte Kleinbuchstaben hervorgehoben. Auf dem Bildschirm erscheinen sie in gelber Schrift. Die Farbe ist aber einstell- und somit veränderbar. Die Schreibweise groß/klein der reservierten Wörter und auch aller anderen Bezeichner (Bezeichner: Namen von Prozeduren, Funktionen, Units, Datentypen, Konstanten und Variablen) ist in Pascal frei. Nur die Schreibweise der Zeichenketten (strings) ist verbindlich. Die Variablen x und X sind somit in Pascal (wie auch in den meisten BASIC-Dialekten) identisch. Hinweis: Für die Programmiersprache C und für Post-Script trifft dies nicht zu.

Es ist in Pascal üblich, Bezeichner, vor allem wenn sie aus mehreren Zeichen bestehen, mit einem großen Anfangsbuchstaben beginnen zu lassen. Bei langen Bezeichnern und zusammengesetzten Wörtern scheue man sich nicht, den Anfangsbuchstabe der Einzel-wörter groß zu schreiben. Insbesondere sollte man bei Prozeduren und Funktionen so verfahren. Wählbare Bezeichner sollten selbsterklärend sein. Für Mehrwertsteuer sollte man z.B. **»Mehrwertsteuer«** oder **»MwSt«** als Bezeichner benutzen und nicht etwa **»x«.**

Beispiele für die Schreibweise von frei gewählten Bezeichner:
`x, Phi, iMin, iMax, MwSt, FlaechendeckungsgradMurrayDavies`

5. Beispiel: Die folgenden beiden Programmvarianten haben gleiche Wirkung:

```
program Test5a;

var
  x: Integer;
  y: Real;

begin
  x := 3;
  y := 0.3;   { führende Null in Pascal anschreiben !! }
  Write(x, y);
end.
```

```
PrOgRam tEST5B;
vAr X   :INteGeR;y:ReAL;bEgin x
:=3;Y      :=0.3;wRitE    (x,
y);eND.
```

Diese beiden Varianten demonstrieren nochmals die Formatfreiheit von Pascal. Es versteht sich von selbst, daß die erste Variante vorzuziehen ist. Systematische Schreibweise der Bezeichner, Leerzeichen, Einrückungen, Leerzeilen und sinnvoll gewählte Bezeichner tragen viel zur Lesbarkeit des Programms bei.

4.3 Der Aufbau eines Pascal-Programms

Ein Pascal-Programm besteht grundsätzlich aus einem Deklarationsteil und einem Ausführungsteil (Hauptprogramm). In Standard-Pascal muß zwingend der folgende Aufbau eingehalten werden. Turbo-Pascal ist liberaler und läßt in gewissen Grenzen eine andere Reihenfolge der Deklarationen und auch Wiederholungen zu. Optional heißt, daß die so gekennzeichneten Deklarationen nur bei bestimmten Situationen notwendig sind.

`program` Beginn des Deklarationteils mit der Deklaration des Programmkopfes. In Turbo-Pascal ist der Programmkopf optional. Der wählbare Programmname hat nur interne Bedeutung und wird *nicht* als Quelltext-Dateiname interpretiert.

`uses` Deklaration von sog. Units. Optional. Units sind bereits compilierte Programm-Module, von Borland oder eigene (Kap. 23). Die Deklaration muß aber unmittelbar nach dem Programmkopf stehen. Die relevanten Standard-Units von Turbo-Pascal:

- **`CRT`** Deklarationen für Bildschirm (Cathode Ray Tube) und Tastatur
- **`PRINTER`** Deklarationen für den Drucker
- **`DOS`** Deklarationen für Betriebssystem-Funktionen
- **`GRAPH`** Deklarationen für Bildschirm-Graphik

`label` Deklaration von Labels (Sprungmarken). Optional. In Turbo-Pascal sind nur Sprünge innerhalb des Blockes erlaubt, in dem das Label deklariert ist. Die Labels bestehen aus einem Label-Namen, dem ein Doppelpunkt folgt. Sie werden mit »**`goto`** *`labelname`*« angesprungen. Der Label-Name kann in Turbo-Pascal aus Zahlen im Bereich 0..9999 oder aus beliebigen alphanumerischen Bezeichnern bis maximal 63 Zeichen bestehen. Von besonderen Situationen abgesehen (z.B. vorzeitiger Programmmabbruch wegen einer Fehlersituation) sollte man in Pascal *keine* GOTO-Sprünge programmieren. **Beispiel für Labelanwendung:**

```
....
label VorzeitigerAbbruch;
....
goto  VorzeitigerAbbruch;
....
VorzeitigerAbbruch:
....
```

`const` Deklaration und Belegung von Konstanten und typisierten Konstanten. Optional.

`type` Deklaration von eigenen Datentypen. Optional.

`var` Deklaration von Variablen mit Angabe des Datentyps. Optional. **Wichtig**: Mit der Deklaration sind die Variablen noch *nicht* initialisiert und besitzen somit undefi-

	nierte Werte, mal so und mal so. Der Compiler erkennt diesen folgenschweren Fehler nicht!
`procedure`	Deklaration der Prozeduren (Unterprogramme). Optional. Der interne Aufbau der Prozeduren und der Funktionen ist analog zu dem des Pascal-Programms selbst, d.h. es können innerhalb der Prozeduren und Funktionen lokale Labels, Konstanten, Datentypen, Variablen, Prozeduren und Funktionen deklariert werden. Im Gegensatz zum Hauptprogramm werden Prozeduren und Funktionen mit »**`end;`**« statt mit »**`end.`**« beendet. Es können beliebig viele Prozeduren und Funktionen deklariert werden.
`function`	Deklaration der Funktionen. Optional. Im Gegensatz zu Prozeduren liefern Funktionen immer einen Wert zuück. Weitere Details siehe Punkt vorher.
`begin`	Beginn des Ausführungsteils (Hauptprogramm).
`end.`	Ende des Ausführungsteils. Man beachte den Punkt!

Es ist empfehlenswert, bei einem Pascal-Programm-Listing nach **`uses`**, **`const`**, **`type`** und **`var`** zuerst das am Ende stehende Hauptprogramm zu lesen und dann erst die **`procedure`** und **`function`**.

An jeder Stelle des Programms können Kommentare geschrieben werden. Sie sind in gechweifte Klammern `{ }` zu setzen. Ersatzzeichen: `(* *)`.

Das spätere Demo-Programm "Pas04081.PAS" im Kap. 4.8 zeigt den typischen Aufbau eines Pascal-Programms mit den Deklarationen **`program`**, **`uses`**, **`const`**, **`type`**, **`var`**, **`procedure`**, **`function`** und dem anschließenden Ausführungsteil (Hauptprogramm, main).

4.4 Die reservierten Wörter in Turbo-Pascal

Im Gegensatz zu Standardbezeichnern wie z.B. `Real`, `Integer`, `Write`, `Sin`, `Log` usw. dürfen reservierte Wörter nicht umdefiniert oder für eigene Bezeichner benutzt werden.

Der folgende Kasten enthält alle reservierten Wörter von Turbo-Pascal. Wie bei allen Pascal-Bezeichnern ist Groß-/Kleinschreibung auch bei reservierten Wörtern beliebig. Es hat sich aber durchgesetzt, reservierte Wörter in Kleinschreibung darzustellen. Im Druck werden reservierte Wörter vorteilhaft durch Fettdruck hervorgehoben.

absolute	and	array	begin	case
const	div	do	downto	else
end	external	file	for	forward
function	goto	if	implementation	in
inline	interface	interrupt	label	mod
nil	not	of	or	packed
procedure	program	record	repeat	set
shl	shr	string	then	to
type	unit	until	uses	var
while	with	xor		

4.5 Selbstdefinierte Bezeichner / Standardbezeichner

Alle Datentypen, Konstanten, Variablen, Prozeduren, Funktionen, Units und der Programmname werden durch Bezeichner identifiziert. Die Bezeichner können beliebig lang sein, aber nur die ersten 63 Zeichen sind signifikant, und das ist mehr als genug!

Die zulässigen Zeichen: **`A..Z`**, (bzw. **`a..z`**), **`0..9`** und der Unterstrich _.

Das erste Zeichen darf kein Ziffernzeichen sein. Groß-/Kleinschreibung ist beliebig. Nicht zulässig sind u.a. das Leerzeichen, der Bindestrich, die Umlaute, das Scharf-S und vor allem der Punkt, da dieser für den Datentyp Record reserviert ist.

Beispiele für zulässige Bezeichner:

1) `BetragOhneMehrwertsteuer`
2) `betragohnemehrwertsteuer`
3) `Betrag_ohne_Mehrwertsteuer`
4) `x, x1, x2, i, iMin, iMax, Phi`

Die Bezeichner 1) und 2) sind gleichwertig.

Hinweis: In der Programmiersprache C und in PostScript ist die Groß-/Kleinschreibung der Bezeichner im Gegensatz zu Pascal und den meisten Basic-Dialekten verbindlich.

Pascal und insbesondere Turbo-Pascal kennt ca. 1000 Standardbezeichner (Namen von vordefinierten Datentypen, Prozeduren (procedure), Funktionen (function) und Units. Diese Standardbezeichner sind *keine* reservierten Wörter und könnten somit zu einer anderen Bedeutung umdefiniert werden. Man hüte sich davor!

Beispiele für Standardbezeichner:

1) `Integer` für Datentyp Ganzzahl
2) `Real` für Datentyp Kommazahl
3) `True` für Standardkonstante Wahrheitswert »wahr«
4) `False` für Standardkonstante Wahrheitswert »falsch«
5) `Write` für Standardprozedur Schreiben

6) `Read` für Standardprozedur Lesen
7) `Sin` für Standardfunktion Sinus

Die Schreibweise der Standardbezeichner ist bezüglich Groß-/Kleinschreibung ebenfalls beliebig. Es ist aber üblich, zumindest die Standardprozeduren und Standardfunktionen, sowie die Datentypen mit einem großen Anfangsbuchstaben zu beginnen. Beachten: **`string`** und **`array`** sind reservierte Wörter.

4.6 Die Begriffe Konstante, Variable und Ausdruck

Eine **Konstante** behält ihren Wert im gesamten Programm bei, d.h. sie kann auch nicht versehentlich durch eine falsche Anweisung geändert werden, der Compiler würde diese Anweisung nicht akzeptieren. Es wird empfohlen, wichtige Konstanten in die const-Deklaration aufzunehmen und im Programm nur noch mit dem vereinbarten Namen anzusprechen.

Eine **Variable** kann, muß aber nicht, im Laufe des Programms verschiedene Werte annehmen. Alle Variablen müssen in der var-Deklaration genannt werden. Die erstmalige Zuweisung eines Wertes an eine Variable nennt man Initialisierung. **Nichtinitialisierte Variablen** werden zwar in Pascal ohne Fehlermeldung verarbeitet, die Werte sind aber **undefiniert**, das Ergebnis ist nicht vorhersehbar, mal so und mal so, auf jeden Fall aber völlig unbrauchbar. Diese Fehler sind schwer zu finden!

Ein **Ausdruck** (engl. expression) kann Konstanten, Variablen, Funktionen und Kombinationen dieser Elemente enthalten. Zu numerischen Ausdrücken könnte man auch Formelausdruck oder Term sagen.

Beispiel für numerische Konstante:
```
4711   { Datentyp: Integer }
```
Beispiel für String-Konstante:
```
'Anton Huber'
```
Beispiel für Variable:
```
x  { Datentyp hier nicht erkennbar }
```
Beispiel für numerischen Ausdruck:
```
y + Sin(z)/2 - 7   { Datentyp Real }
```
Beispiel für String-Ausdruck:
```
'Huber' + Copy(s, 3, 5)
{s muß Stringvariable sein }
```

Wenn eine Formatbeschreibung einen **Ausdruck** verlangt, dann kann an der betreffenden Stelle auch eine Konstante oder eine Variable stehen.

Wenn eine Formatbeschreibung eine **Konstante** verlangt, dann muß an der betreffenden Stelle eine Konstante stehen, nicht aber eine Variable oder ein Ausdruck.

Sinngemäßes gilt auch, wenn die Formatbeschreibung eine **Variable** verlangt. Es muß dann eine Variable eingegeben werden.

Pascal kennt darüber hinaus auch noch **typisierte Konstanten**. Das sind im Grunde Variablen, die in der const-Deklaration mit Datentyp deklariert und dann mit einem Wert initialisiert werden. Diese "Konstanten" können im Programm wie Variablen verändert werden.

Beispiel für (echte) Konstanten und typisierte Konstanten:

```
program Konstanten;

const
  x = 13;          { "echte" Konstante }
  z: Integer = 13; { typisierte Konstante = initialisierte Variable }

begin
  z := z + 1;    { Zulässig, da typisierte Konstante, init. Variable }

  x := x + 1;   { Unzulässig, da "echte" Konstante. Fehlermeldung   }

end.
```

4.7 Zur Formatbeschreibung

Für die Formatbeschreibung (Syntaxbeschreibung) der Programmteile (reservierte Wörter, Standardprozeduren, Standardfunktionen usw.) sind verschiedene Symbole gebräuchlich, die aber an dieser Stelle selbst nicht Bestandteil von Pascal sind. Dazu zählen eckige Klammern [ ] für optionale Angaben. In Pascal selbst werden eckige Klammern für Indizies von Arrays benutzt.

Den Aufbau eines Turbo-Pascal-Programms kann man mit den Optionsklammern vereinfacht wie folgt darstellen:

```
[ program   ]
[ uses      ]
[ label     ]
[ const     ]  Reihenfolge von const und type ggf. auch anders
[ type      ]
[ var       ]
[ procedure ]  Reihenfolge von procedure und function beliebig
[ function  ]
begin
end.
```

Für Wiederholungen von möglichen Eingaben sind in der Formatbeschreibung drei Wiederholungspunkte **...** gebräuchlich. Zwei Wiederholungpunkte werden in Pascal für Bereichsgrenzen benutzt, dazu später.

Ein Beispiel für die Formatbeschreibung der Standardprozedur Write:

Format: `Write(`*a1* `[, `*a2*`, ..., `*an*`])`
a1, *a2*, ... *an*: Ausdrücke

Man sieht, daß in Pascal das Komma als Trennzeichen benutzt werden muß, wenn die (Ausgabe-) Liste mehr als einen Ausdruck umfaßt.

Konkrete Beispiele (die Variablen seien als deklariert und belegt angenommen):

```
Write(4711);            { nur numerische Konstante }

Write('Der Funktionswert von x =  ', x, 'ist: ', x + Sin(x)/2 - 7);
      └──────────────────────────┘   │  └──────┘  └──────────────┘
                   a1               a2    a3          a4
```

a1 ist eine String-Konstante
a2 ist eine numerische Variable
a3 ist eine String-Konstante
a4 ist ein numerischer Ausdruck

4.8 Programmbeispiel Zahlenumwandlung

Dieses Programmbeispiel ist nur zum Studieren des Programmaufbaus, zum Üben der Zahlensysteme und zum Üben mit dem Turbo-Pascal-System (Kap. 5) gedacht. Das Programm wandelt Dezimalzahlen aus dem Bereich 0 bis 255 in binäre, oktale und hexadezimale Darstellung um.

```
program Pas04081;       { Umwandlung dezimal in binär, oktal und hex }
                        { Hier nur für Dezimalbereich von 0 bis 255  }
                        { K. Haller, Turbo-Pascal,          77170390 }
  { Der Quelltext des Programms kann vom Anfänger noch nicht voll-  }
  { ständig verstanden werden. Er dient lediglich zur Demonstration }
  { des Aufbaus eines einfachen Pascal-Programms mit einer Prozedur }
  { und einer Funktion (alles etwas gekünstelt). Ansonsten dient    }
  { das Programm zum Veranschaulichen der Zahlensysteme.            }

uses                    { Deklaration der verwendeten Unit(s) }
  CRT;

type                    { Deklaration von eigenen Datentypen }
  Str1 = string[1];

const                   { Deklaration von Konstanten }
  Esc = #27;            { Für Taste "Esc" }

var                     { Deklaration von Variablen }
  Dezimalzahl,
  DezimalTemp:  Integer;
```

```
  Basis,
  Laenge,
  Spalte,
  Zeile:        Byte;
  Beenden:      Boolean;
  Zahlenstring,
  Kommentar:    string;
  Symbol:       Str1;
                { Es folgt Deklaration einer Prozedur: }
procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
  Write(Meldung);
end;

function Eingabe: Char;   { Deklaration einer Funktion }
var
  Ch: Char;  { lokale Variable }
begin
  ClrScr;
  WriteXY(25,  2, 'Fachhochschule München, Stg DR     ');
  WriteXY(25,  3, 'Demo Umwandlung Zahlensysteme      ');
  WriteXY(25,  4, '-----------------------------------');
  WriteXY(25,  5, '1    Umwandlung dezimal-binär      ');
  WriteXY(25,  6, '2    Umwandlung dezimal-oktal      ');
  WriteXY(25,  7, '3    Umwandlung dezimal-hexadezimal');
  WriteXY(25,  8, 'Esc  Ende                          ');
  WriteXY(25,  9, '-----------------------------------');
  GotoXY( 25, 10);
  repeat
    Ch := ReadKey;
  until Ch in ['1'..'3', Esc];
  WriteLn(Ch);
  WriteLn;
  Eingabe := Ch;
end;

begin  { -------- Hauptprogramm, main ------------------------------ }
  TextBackground(Blue);
  TextColor(Yellow);

  repeat
    Beenden := False;

    case Eingabe of
    Esc: Beenden := True;
    '1': begin
           Kommentar := 'Umwandlung in binär: ';
           Basis     :=  2;
           Laenge    :=  8;
           Symbol    := '';  { Kein sichtbares Symbol. Mitunter 'b' }
         end;
    '2': begin
           Kommentar := 'Umwandlung in oktal: ';
           Basis     :=  8;
           Laenge    :=  3;
           Symbol    := 'o'; { Auch üblich: 'q' }
         end;
    '3': begin
```

```
          Kommentar := 'Umwandlung in hexadezimal: ';
          Basis     := 16;
          Laenge    :=  2;
          Symbol    := '$'; { In Pascal '$', sonst meistens 'h' }
        end;
    end;

    if not Beenden then
       repeat
         Spalte := WhereX;
         Zeile  := WhereY;
         repeat
           GotoXY(Spalte, Zeile);
           ClrEoL;
           WriteXY(Spalte, Zeile,
                   ' Eingabe Dezimalzahl 0..255, Ende mit 0: ');
           {$I-} { "$I-": Compilerbefehl. Vorgriff. Kein Kommentar }
           ReadLn(Dezimalzahl);
           {$I+}
         until (IOResult = 0) and (Dezimalzahl >= 0) and
               (Dezimalzahl <= 255);

         Zahlenstring := '';
         DezimalTemp  := Dezimalzahl;
         repeat
           Zahlenstring := Copy('0123456789ABCDEF',
                                (DezimalTemp mod Basis) + 1, 1) +
                           Zahlenstring;
           DezimalTemp  := DezimalTemp div Basis;
         until DezimalTemp = 0;

         while Length(ZahlenString) < Laenge do
           Zahlenstring := '0' + Zahlenstring;

         Zahlenstring := Symbol + Zahlenstring;

         WriteXY(50, WhereY - 1, Kommentar + Zahlenstring + #13#10);
       until Dezimalzahl = 0;
  until Beenden;

end.
```

4.9 Zur Gestaltung von Pascal-Quelltexten

Die Gestaltung der Quelltexte kann wesentlich zum Verständnis beitragen. Pascal ist zwar völlig formatfrei was die Quelltexte anbelangt. Bei Mißbrauch der Formatfreiheit kann vielleicht das Programm fehlerfrei sein, aus dem Quelltext wird man aber wahrscheinlich nicht sehr schlau.

Ein Beispiel für einen gut gestalteten Quelltext:

```
program Test;
```

```
uses
  CRT, PRINTER;
var
  x:    Real;
  i:    Integer;
  Name: string;

begin
  ClrScr;
  x := Sin(47.11);
  i := 4711;
  Name := 'Huber Toni';
  WriteLn(Lst, 'x = ', x:6:2,',i = ', i, ', Name: ', Name);
end.
```

... und das gleiche Programm in „formatfreier“ Darstellung:

```
PRogRAM teSt;useS cRt,priNTER;vAr x:
ReAl;i:intEger;naME:striNg;BegIn ClRSCr;
x:=Sin  (47.11);i:=4711;Name:='Huber Toni';wrITeln(lSt,
'x = ', x:6:2,         ', i = ',i,', Name: ',NamE)  ;eND.
```

Allgemeine Hinweise für die Gestaltung der Quelltexte:

Die nachstehenden Hinweise für die Gestaltung nehmen notgedrungen einige Pascal-Begriffe vorweg und sind für späteres Nachschlagen bestimmt.

1. Reservierte Wörter (z.B. **begin**, **end**, **while**, ...) in Kleinschreibung. Bezeichner (eigene und Pascal-Namen von Konstanten, Variablen, Prozeduren und Funktionen) mit großen Anfangsbuchstaben. Ausgenommen eigene Kurzbezeichner mit vorrangig mathematischer Bedeutung und max. zwei Zeichen Länge. Bei eigenen Bezeichnern selbsterklärende Namen verwenden, z.B. „MwSt“ und nicht „x“.
2. Folgende Operatoren sind mit einem vorstehenden und einem nachstehenden Leerzeichen freizustellen: Zuweisungsoperator, Vergleichsoperator, Additionsoperator, Subtraktionsoperator. Bei besonders komplizierten oder wichtigen numerischen Ausdrücken sind eventuell auch die Multiplikations- und Divisionsoperatoren freizustellen; nicht aber bei einfachen Ausdrücken. Überflüssige Klammern tragen bei numerischen Ausdrücken nicht zur besseren Lesbarkeit bei!

   Beispiele:

   ```
   x := 47.11;
   ```

```
y := x + Sin(x)/Sqrt(2*x) - 7;
if x >= y
   then ....;
```

3. **Nach** allen Trennzeichen (Komma, Strichpunkt, Punkt, Doppelpunkt usw.) ist ein Leerzeichen einzufügen.

4. Zwischen dem Bezeichner einer Prozedur oder einer Funktion und der öffnenden Klammer der Parameterliste ist kein Leerzeichen einzufügen. Nach der öffnenden Klammer können aber bei Bedarf Leerzeichen folgen.

   Also: `Sin(47.11)` und nicht: `Sin (47.11)`

5. Man spare nicht mit Leerzeilen zwischen logisch verschiedenen Blöcken. So ist z.B. eine Leerzeile zwischen Deklarationsteil und dem Hauptprogramm sinnvoll.

6. Einrückungen in Form von zwei Leerzeichen) zwischen **begin** und **end**. Ansonsten sind alle Anweisungen bis auf die noch genannten Maßnahmen gleichberechtigt und werden nicht eingerückt. In den folgenden schematischen Beispielen ist mit *a* eine beliebige Anweisung und mit *b* eine beliebige Bedingung gemeint. Mit *i* wird eine beliebige Laufvariable einer for-Schleife, mit *aw* und *ew* der Anfangswert bzw. der Endwert dieser Schleife bezeichnet.

   In den folgenden Beispielen soll der Senkrechtstrich | den linken Rand des Bildschirms darstellen.

```
|....
|begin
|  a1;
|  a2;
|  ....
|end.
```

   Tritt innerhalb des **begin/end**-Blockes ein weiterer auf, so wird in gleicher Weise nach **begin** nochmals eingerückt und mit dem **end** auf die vorher aktuelle Spalte wieder ausgerückt.

7. Zusätzliche Einrückungen bei Schleifen:

   7a) **repeat/until**-Schleife

```
|  ....
|  repeat
|    a1;
|    a2;
|  until b;
```

```
|  ....
```

7b) **while**-Schleife mit mehr als einer Anweisung:

```
|  ....
|  while b do
|    begin
|      a1;
|      a2;
|    end;
|  ....
```

7c) **for**-Schleife mit mehr als einer Anweisung:

```
|  ....
|  for i : aw to ew do
|    begin
|      a1;
|      a2;
|    end;
|  ....
```

Die **while**- oder **for**-Schleifen mit nur *einer* Anweisung werden wie folgt geschrieben:

```
|  ....
|  while b do a;
|  ....
|  { oder schöner: }
|  while b do
|    a;
|  ....

|  ....
|  for i := aw to ew do a;
|  ....
|  { oder schöner: }
|  for i := aw to ew do
|    a;
|  ....
```

8. Einrückungen bei Deklarationen:

```
|uses
|  CRT, DOS, PRINTER; { Unit-Namen groß schreiben }
|
|var
```

```
|  x, y: Real;
|  s:    string;
```

9. Einrückungen bei **if/then** mit nur *einer* Anweisung:

```
|  ....
|  if b then a;....
|  ....
|  { oder schöner }
|  if b....
|     then a;
|  ....
```

10. Einrückungen bei **if/then/else** mit nur je *einer* Anweisung:

```
|  ....
|  if b
|     then a1
|     else a2;
|  ....
```

11. Einrückungen bei **if/then** mit mehr als *einer* Anweisung:

```
|  ....
|  if b then
|     begin
|       a1;
|       a2;
|     end;
|  ....
```

12. Einrückungen bei **if/then/else** mit mehr als *einer* Anweisung in beiden Zweigen:

```
|  ....
|  if b
|     then begin
|            a1;
|            a2;
|          end
|     else begin
|            a3;
|            a4;
|          end ;
|  ....
```

13. Einrückungen sind auch zu machen, wenn eine Zeile nicht in das Edit-Fenster (77 Zeichen) paßt. Der Zeilenrest ist um zwei Leerzeichen gegenüber der aktuellen Spalte einzurücken. Die Zeilentrennung kann nach jedem Trennzeichen erfol-gen. Lange Strinkonstanten können in Teile aufgespalten und mit dem Verknüpfungs-operator + verbunden werden. Vor oder nach diesem Operator kann eine Zeilen-trennung erfolgen. Mit den genannten Methoden können beliebig lange Anweisun-gen erstellt werden, ohne daß der Bildschirm seitlich gescrollt werden muß oder beim Ausdruck verstümmelte Listings erscheinen.

4.10 Register der Turbo-Pascal-Begriffe

Wichtig: Die Vollständigkeit und Richtigkeit des Registers ist nicht gewährleistet. Die Studierenden werden um Hinweise zur Ergänzung gebeten.

Art:

R = reserviertes Wort
P = Prozedur
F = Funktion
U = Unit
T = Datentyp
V = vordefinierte Variable
K = vordefinierte Konstante
C = Compilerbefehl (-Schalter, -Parameter, -Bedingung)

Art	Begriff	Kap.	Erläuterung
F	#	08.4	Siehe Funktion "Chr(xx)"; Beispiel: #27 ist identisch mit Chr(27)
C	{$A+}, {$A-}	05.9	Compilerbefehl
C	{$B+}, {$B-}	05.9	Compilerbefehl
C	{$D+}, {$B-}	05.9	Compilerbefehl
C	{$DEFINE ....}	05.9	Compilerbefehl
C	{$E+}, {$E-}	05.9	Compilerbefehl
C	{$ELSE}	05.9	Compilerbefehl
C	{$ENDIF}	05.9	Compilerbefehl
C	{$I .... }	23.1	Compilerbefehl: Include-Datei
C	{$I datei }	05.9	Compilerbefehl
C	{$I-} {$I+}	14.9	Compilerbefehl: Input-Kontrolle Aus/Ein
C	{$I+}, {$I-}	05.9	Compilerbefehl
C	{$IF ....}	05.9	Compilerbefehl
C	{$IFDEF ...}	05.9	Compilerbefehl
C	{$IFNDEF ...}	05.9	Compilerbefehl
C	{$IFOPT ....}	05.9	Compilerbefehl
C	{$L+}, {$L-}	05.9	Compilerbefehl
C	{$M ...}	05.9	Compilerbefehl
C	{$N+}, {$N-}	05.9	Compilerbefehl

C	{$O ....}	05.9	Compilerbefehl
C	{$O+}, {$O-}	05.9	Compilerbefehl
C	{$R+}, {$R-}	05.9	Compilerbefehl
C	{$S+}, {$S-}	05.9	Compilerbefehl
C	{$UNDEFINE ...}	05.9	Compilerbefehl
C	{$V+}, {$V-}	05.9	Compilerbefehl
C	{$X+}, {$X-}	11.16	Comp.-befehl erweit. Syntax: Funktionen wie Prozeduren.
F	Abs	08.2	
F	Abs	08.3	
R	and	08.2	bitweises AND
R	and	08.5	logisches AND
P	Arc	21.3	
F	ArcTan	08.2	
R	array ... of	12.1	
R	asm	27.5	
P	Assign	18.1	
P	AssignCRT	18.3	
P	Bar	21.4	
R	begin	04.2	
K	BkShlashFill	21.4	
K	Black	21.2	Farbkonstante = 0
P	BlockRead	18.3	
P	BlockWrite	18.3	
K	Blue	21.2	Farbkonstante = 1
T	Boolean	08.5	
K	Brown	21.2	Farbkonstante = 6
T	Byte	08.2	
R	case ... of	09.4	
T	Char	08.4	
P	ChDir	18.3	
F	Chr	08.2	
F	Chr	08.4	
P	Circle	21.3	
P	ClearViewPort	21.4	
K	ClipOff	21.4	
K	ClipOn	21.4	
P	Close	18.1	
K	CloseDotFill	21.4	
P	CloseGraph	21.1	
P	ClrEoL	07.12	
P	ClrScr	07.6	
T	Comp	08.3	
F	Concat	14.2	Besser "+" statt "Concat"
R	const	04.3	
F	Copy	14.4	
F	Cos	08.2	
K	CPU97	05.9	Zu Compilerbefehl
U	CRT	04.3	
U	CRT	07.3	
K	Cyan	21.2	Farbkonstante = 3
K	DarkGray	21.2	Farbkonstante = 8
P	Dec	08.2	
P	Delay	07.13	
P	Delete	14.6	
P	DelLine	07.12	
K	Detect	21.1	
F	DiskFree	18.3	

F	DiskSize	18.3	
P	Dispose	19.2	
R	div	08.2	
R	do	10.2	Bei while-Scheifen. Auch bei for-Schleifen
R	do	16.2	Bei: with ... do
P	do	10.3	Bei for-Schleifen. Auch bei while-Scheifen
U	DOS	04.3	
F	DOSError	24.4	
F	DOSExitCode	24.4	
K	DottedLn	21.4	
T	Double	08.3	
R	downto	10.3	
R	else	09.2	Bei: if ... then ... else ...
R	else	09.4	Bei: case ... of ..... else ...
R	end	04.2	
F	EnvCount	24.3	
F	EoF	18.1	
P	EoLn	18.3	
P	Erase	18.3	
P	Exclude	15.7	
P	Exec	24.4	
P	Exit	11.11	
F	Exp	08.2	
T	Extended	08.3	
R	external	11.13	
K	False	08.5	
P	FExpand	18.3	
R	file of ...	18.1	
V	FileMode	18.3	
F	FilePos	18.3	
F	FileSize	18.1	
F	FileSize	18.3	
P	FillChar	22.3	
P	FillChar	14.11	
P	FindFirst	18.3	
P	FindNext	18.3	
P	Flush	18.3	
R	for ... downto	10.3	
R	for ... to ... do	10.3	
R	forward	11.8	
F	Frac	08.3	
P	FreeMem	19.2	
F	FSearch	18.3	
P	FSplit	18.3	
R	function	04.3	
R	function	11.3	
P	GetDate	07.15	
P	GetDir	18.3	
P	GetFAttr	18.3	
F	GetMaxX	21.3	
F	GetMaxY	21.3	
P	GetMem	19.2	
F	GetPixel	21.3	
P	GetTime	07.15	
K	GothicFont	21.4	
R	goto	09.5	
P	GotoXY	07.7	

U	GRAPH	04.3	
K	Green	21.2	Farbkonstante = 2
K	GrOK	21.4	
P	Halt	07.16	
F	Hi	08.2	
P	HighVideo	07.10	
K	HorizDir	21.4	
R	if ... then ...	09.1	
R	if ... then ... else	09.2	
R	in	15.2	
P	Inc	08.2	
P	Include	15.7	
P	InitGraph	21.1	
R	inline	27.5	
R	inline	11.12	
P	Insert	14.7	
P	InsLine	07.12	
T	Integer	08.2	
P	Intr	27.6	
F	KeyPressed	07.4	
F	KeyPressed	08.5	
R	label	04.3	
F	Length	14.3	
K	LightBlue	21.2	Farbkonstante = 9
K	LightCyan	21.2	Farbkonstante = 11
K	LightGray	21.2	Farbkonstante = 7
K	LightGreen	21.2	Farbkonstante = 10
K	LightMagenta	21.2	Farbkonstante = 13
K	LightRed	21.2	Farbkonstante = 12
P	Line	21.3	
P	LineTo	21.3	
F	Ln	08.2	
F	Lo	08.2	
T	LongInt	08.2	
P	LowVideo	07.10	
K	Lst	25.2	
V	Lst	07.5	
K	Magenta	21.2	Farbkonstante = 5
P	Mark	19.2	
F	MaxAvail	19.2	
K	MaxInt	08.2	
K	MaxLongInt	08.2	
V	Mem	12.10	
F	MemAvail	19.2	
V	MemL	12.10	
V	MemW	12.10	
P	MkDir	18.3	
R	mod	08.2	
P	Moveto	21.3	
P	MsDOS	27.6	
P	New	19.2	
R	nil	19.2	
P	NormVideo	07.10	
K	NormWidth	21.4	
P	NoSound	07.14	
R	not	08.2	bitweise Negation
R	not	08.5	logische Negation

F	Odd	08.2	
F	Odd	08.5	
R	of	12.1	Bei: array ... of
R	of	15.1	Bei: set of ...
R	of	18.1	Bei: file of ...
R	of	09.4	Bei: case ... of
T	OpenString	14.15	Übergabe von offenen Strings als Routinen-Parameter
R	or	08.2	bitweises OR
R	or	08.5	logisches OR
F	Ord	08.2	
P	OutText	21.3	
P	OutTextXY	21.3	
F	ParamCount	24.2	
F	ParamStr	24.2	
F	Pi	08.3	
T	Pointer	19.2	
F	Pos	14.5	
F	Pred	08.2	
U	PRINTER	04.3	
U	PRINTER	07.5	
R	procedure	04.3	
R	procedure	11.3	
R	program	04.2	
P	PutPixel	21.2	
F	Random	08.3	Real-Zufallszahl aus Bereich [0.0 ... [1.0
F	Random(a)	08.2	Word-Zufallszahl aus Bereich [0 ... (a - 1)]
P	Randomize	08.2	
P	Read, ReadLn	07.2	Von Tastatur lesen
P	Read, ReadLn	18.1	Von Datei lesen
F	ReadKey	07.3	
T	Real	08.3	
R	record	16.2	
P	RectAngle	21.3	
K	Red	21.2	Farbkonstante = 4
T	Registers	27.6	
P	Release	19.2	
P	Rename	18.3	
R	repeat / until	10.1	
P	Reset	18.1	
P	Rewrite	18.1	
P	RmDir	18.3	
F	Round	08.2	
K	SansSerifFont	21.4	
P	Seek	18.1	
F	SeekEoF	18.3	
F	SeekEoL	18.3	
R	set of ...	15.1	
P	SetDate	07.15	
P	SetFAttr	18.3	
P	SetLineStyle	21.4	
P	SetTextBuf	18.3	
P	SetTextStyle	21.4	
P	SetTime	07.15	
P	SetViewPort	21.4	
R	shl	08.2	
T	ShortInt	08.2	
R	shr	08.2	

F	Sin	08.2	
T	Single	08.3	
F	SizeOf	19.2	
K	SlashFill	21.4	
K	SmallFont	21.4	
K	SolidFill	21.4	
P	Sound	07.14	
F	Sqr	08.2	
F	Sqrt	08.2	
P	Str	14.8	
R	string	14.1	
F	Succ	08.2	
P	SwapVectors	24.4	
V	Text	18.1	Textdatei
P	TextBackGround	07.11	
P	TextColor	07.11	
P	TextMode	07.11	
R	then	09.1	Bei: if ... then ...
R	then	09.2	Bei: if ... then ... else ...
K	ThickWith	21.4	
R	to	10.3	
K	True	08.5	
F	Trunc	08.2	
P	Truncate	18.3	
R	type	04.3	
R	type	08.7	
R	unit	23.3	
F	UpCase	08.4	
R	uses	04.3	
P	Val	14.9	
R	var	04.2	
R	var	04.3	
R	var	11.7	Parameterübergabe an Routinen mit Adresse
K	VertDir	21.4	
F	WhereX, WhereY	07.8	
R	while ... do	10.2	
K	White	21.2	Farbkonstante = 15
P	Window	07.9	
R	with ... do	16.2	
T	Word	08.2	
P	Write	18.1	In Datei schreiben
P	Write, WriteLn	07.1	
P	WriteLn	18.1	In Datei schreiben
K	xHatchFill	21.4	
R	xor	08.2	bitweises XOR
R	xor	08.5	logisches XOR
K	Yellow	21.2	Farbkonstante = 14