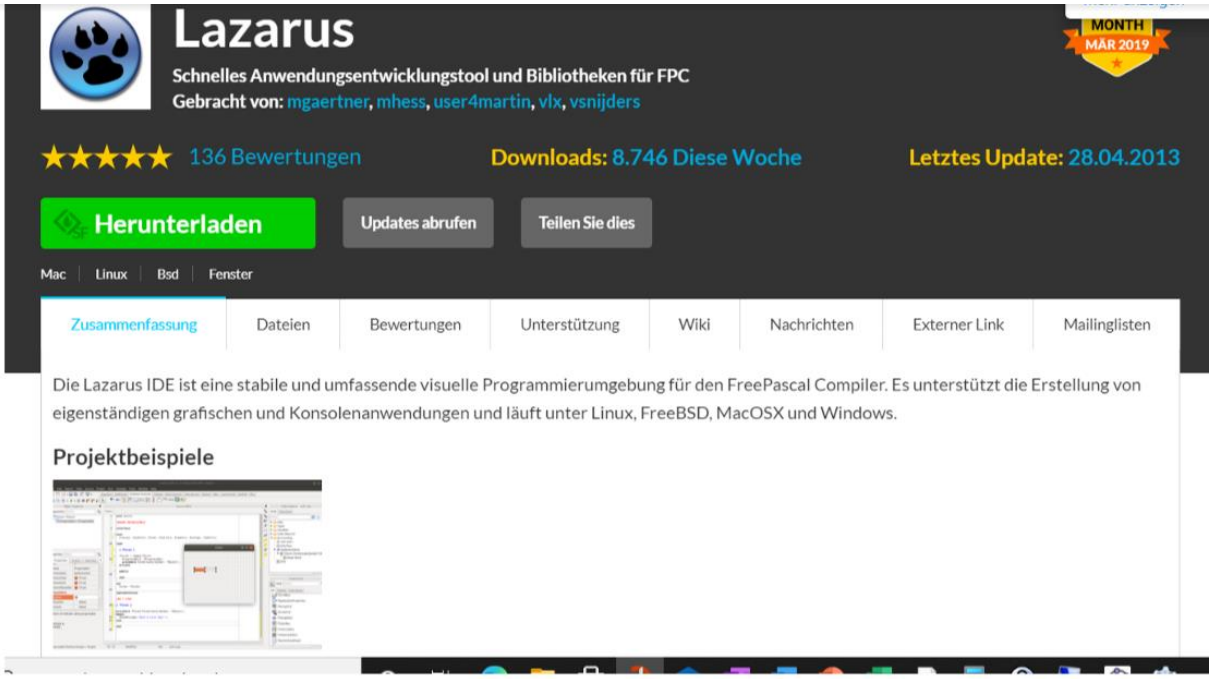
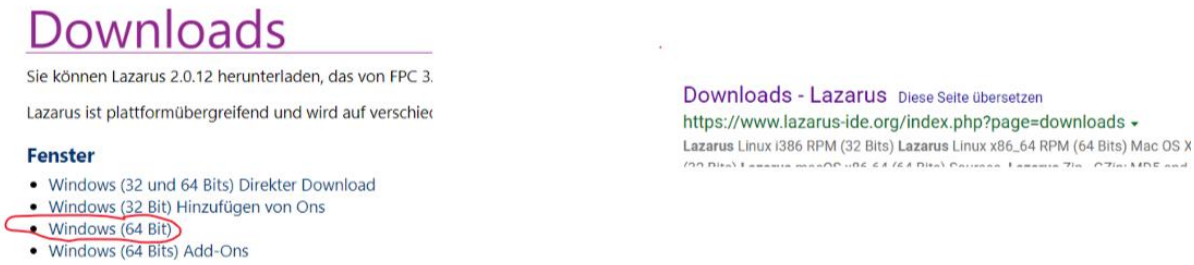
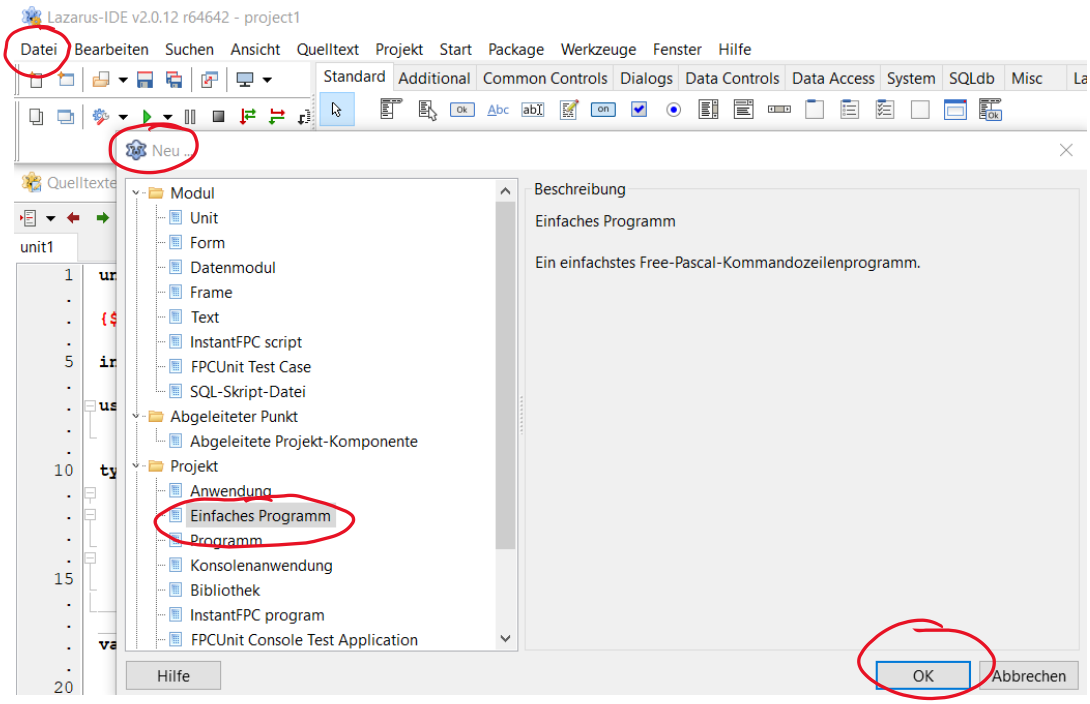
1.0 Pascal mit Lazarus







Die Programme in diesem Skriptum sind mit Delphi 6 geschrieben worden und gelten als Vorlage für Lazarus.
Damit die Programme unter Lazarus in einem Windowsfenster ablaufen, wird die Unit crt (uses crt;) benötigt.

VORSICHT im Skript steht vorwiegend Win32Crt (Ist für Delphi 6). Bei Verwendung des Programmes Lazarus darf nur Crt eigegeben werden,

2.0 Lineare Programme

2.1 Zinsenberechnung

Nach dem Programmstart öffnet sich automatisch ein neues Projekt. Es kann über das Menü *Datei* geschlossen werden.

Über *Datei – Neu – Einfaches Programm* kann ein neues Projekt geöffnet werden.

Das folgende Projekt soll mit dem Namen Zinsrechnung abgespeichert werden. Dann kann das folgende Programm eingegeben werden. (Das Programm wurde um einige erläuternde Texte erweitert.)

```
PROGRAM Zinsrechnung;
USES Crt;
CONST ZINSSATZ = 2;
VAR Kapital, Zinsen : REAL; { 2 Variable vereinbaren }
BEGIN ClrScr;
writeln ('Berechnung der Verzinsung eines Kapitals');
writeln;
write ('Bitte gib das Kapital ein: ');
readln (Kapital);
Zinsen := Kapital*ZINSSATZ/100;
Kapital := Kapital + Zinsen;
writeln ('Das Kapital ist: ', Kapital:10:2, 'Euro. ');
readln;  {Haltepunkt}
END.
```

Compiler starten durch Drücken der <Strg F9> Taste oder  . Findet der Compiler dabei einen Fehler, erscheint dieser in der Statuszeile am Fuß des Fensters. Der Cursor springt an die Stelle, wo der Compiler den Fehler vermutet.
Bei der Ausführung des Programms öffnet sich ein neues Fenster, welches nach Ablauf des Programms wieder automatisch geschlossen wird. Um das Fenster betrachten zu können, muss noch ein Haltepunkt mit *readln* gesetzt werden.

2.2 Aufbau eines Pascalprogramms

Bezeichnung	Wortsymbol	Zweck
Programmkopf	PROGRAM	Programm erhält seinen Namen
Vereinbarungsteil (Definitionsteil)	CONST, TYPE, VAR, FUNCTION, PROCEDURE	Im Programm verwendete Konstanten, Typen, Variablen und Unterprogramme werden definiert.
Anweisungsteil (Hauptprogramm)	BEGIN ... END.	Formulieren der einzelnen Programmschritte in der Reihenfolge ihrer Durchführung.

Im obigen Programmbeispiel sind die Wortsymbole (reservierten Wörter) groß geschrieben. In Pascal wird bei den Wortsymbolen, Bezeichnern etc. nicht zwischen Groß- und Kleinbuchstaben unterschieden.

Die Version 7.0 bringt hier eine Verbesserung, da automatisch alle erkannten Wortsymbole bereits im Quelltext farblich hervorgehoben werden und man Schreibfehler bereits vor dem Compilieren erkennen kann.

2.3 Erläuterung des Programms

```
PROGRAM Zins1j;     (* Programmkopf *)
USES Win32Crt;
      ^
      └──                           Unit Win32Crt einfügen
                                          (dort ist "ClrScr" vereinbart)
CONST ZINSSATZ = 2;
       ^       ^
       │       └──                  Legt den Wert der Konstanten fest.
       └──  Hier wird die Bezeichnung einer Konstanten vereinbart.
VAR Kapital, Zinsen : REAL; { 2 Variable vereinbaren }
       ^               ^
       │               └──          Legt den Typ der Variablen fest.
       │                            Hier handelt es sich um eine reelle Zahl.
       └──                          Bezeichnung der Variablen.
BEGIN                               Beginn des Anweisungsteils.
  ClrScr;                           Löscht den Bildschirm.
  writeln ('Berechnung ...');       Schreibt einen Text. Cursor springt dann in die
                                    nächste Bildschirmzeile.
  writeln;                          Schreibt eine Leerzeile.
  write ('Bitte gib ... ');         Schreibt ohne Zeilenvorschub.
  readln (Kapital);                 Erwartet Eingabe von Tastatur, die mit ↵
                                    abzuschließen ist. Dann Sprung in nächste
                                    Bildschirmzeile.
  Zinsen :=  Kapital*ZINSSATZ/100;  Berechnung der Zinsen
  Kapital := Kapital + Zinsen;      Berechnung des neuen Kapitals
  writeln ('Das Kapital ist: ',     Ausgabe eines Textes
           Kapital);                Ausgabe des Kapitalwertes, dann Sprung in nächste
                                    Bildschirmzeile.
END.                                Programm ist zu Ende. Auf den Punkt nicht
                                    vergessen!
```

2.4 Syntax eines Pascalprogramms

Ein Pascalprogramm kann aus folgenden Zeichen aufgebaut sein:

Buchstaben:	A bis Z, a bis z und _	
Ziffern:	0 1 2 3 4 5 6 7 8 9	
Sonderzeichen:	+ - * / = ^ < > ( ) [ ] { } . , :	; ' # $ und dem Leerzeichen

Deutsche Umlaute oder "ß" dürfen nur in Kommentaren oder in Zeichenketten vorkommen.

Bezeichner (Namen, identifier)

können aus Buchstaben, Ziffern oder dem Unterstreichungszeichen zusammengesetzt sein und müssen mit einem Buchstaben oder dem Unterstreichungszeichen beginnen. Im obigen Beispiel ist `Gruss` der Bezeichner für den Programmnamen und `Vorname` der Bezeichner für eine Variable.

In Borland-Pascal wird zwischen Groß- und Kleinbuchstaben nicht unterschieden, die maximale Länge eines Bezeichners kann 127 Zeichen betragen, die alle signifikant sind. Es dürfen keine Leerzeichen enthalten sein und keine reservierten Wörter als Namen verwendet werden.

Reservierte Wörter (Wortsymbole)

sind z. B.: `AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNTO, ELSE, END`usw.

Variable

Sind Größen, deren Wert sich ändern kann. Jede Variable ist eine Einheit aus

Bezeichner (Variablenname) - Speicherplatz im Computer (Adresse) - Typ der Variablen - Wert der Variablen

Im Definitionsteil werden Bezeichner und Typ der Variablen vereinbart, der Compiler reserviert für die Variable bei der Übersetzung einen Speicherplatz, im Anweisungsteil muss ihr ein Wert zugewiesen werden. In unserem Beispiel geschieht dies über eine `read`-Anweisung. Eine weitere Möglichkeit besteht in einer sogannten Wertzuweisung.

Konstante

Werden auf Seite 8 genauer behandelt. Vorläufig verwenden wir nur ganzzahlige, reelle und Zeichenkettenkonstanten.

Ganzzahlige Konstante sind z. B.:	-4	351
Reelle Konstante sind z. B.:	-3.14	3E-16
Zeichenkettenkonstante sind z. B.:	'Servus!'	'Sehr gut'

Letzere müssen immer von einfachen Anführungszeichen eingeschlossen sein. Ein Anführungszeichen kann in einem String stehen, wenn es doppelt geschrieben wird:

'That"s it!'

Zu unterscheiden sind Variablenbezeichner und Stringkonstante. Mit der Anweisung

```
write (Kapital);
```

wird der Inhalt der Variablen `Kapital` auf dem Bildschirm ausgegeben. Die Anweisung

```
write ('Kapital');
```

gibt eine Zeichenkettenkonstante auf dem Bildschirm aus.

Mit der Vereinbarung

```
CONST ZINSSATZ = 2;
```

können den Konstanten auch Namen zugewiesen werden. Diese Namen können im Programm stellvertretend für die Konstanten verwendet werden.

Trennzeichen

Leerzeichen, ein Zeilenwechsel oder Kommentare müssen zur Trennung zwischen Wortsymbolen und Bezeichnern verwendet werden. Ansonsten sind Leerzeichen und Zeilenwechsel meist bedeutungslos, dürfen aber nicht innerhalb eines Wortsymbols, Bezeichners oder einer Konstanten vorkommen.

Kommentare

müssen zwischen geschlungene Klammern

{ }

gesetzt werden. Statt dieser Klammern kann man auch die Kombination

(* *)

verwenden.

Kommentare werden in Delphi oder Pascal Version 7.0 farblich im Editor gekennzeichnet.

2.5 Variablenvereinbarung und Wertzuweisung

In Pascal muss jede Variable, die man im Programm verwenden möchte, zuerst vereinbart werden, wobei auch der sogenannte Variabentyp anzugeben ist. Im nächsten Programm soll den Kreisumfang und die Kreisfläche bei gegebem Radius berechnen. Die benötigen Variablen sollen `Radius, Umfang` und `Flaeche` heißen, die vom Typ REAL sein müssen. Variable vom gleichen Typ können in einer einzigen Anweisung vereinbart werden.

Allgemein hat diese Vereinbarung folgende Form:

VAR Bezeichner {,Bezeichner} : REAL;

wobei die in geschlungenen Klammern andeuten sollen, dass die eingeschlossenen Begriffe beliebig oft wiederholt werden können.

Wertzuweisungen erfolgen durch den Zuweisungsoperator

:=

("ergibt sich zu", "wird zugewiesen", "wird belegt mit"). Auch hier darf zwischen dem Doppelpunkt und dem Gleichheitszeichen kein Leerzeichen stehen. Auf der rechten Seite des Zuweisungsoperators kann eine konstante Zahl oder ein Ausdruck stehen.

Für das Rechnen mit Zahlen vom Typ REAL stehen in Pascal die binären Rechenoperatoren

+, -, * und /

zur Verfügung, wobei * und / höhere Priorität gegenüber den Operatoren + und - haben. Durch Setzen von Klammern kann man jedoch diese Prioritäten umgehen.

Beispiel:

Die Formel $A = \frac{a+c}{2} h$

wird in Pascal als

A := (a + c)*h/2;

geschrieben.

Oft muss in einem Programm der Wert einer Variablen geändert werden. Wird einer Variablen eine neuer Wert zugewiesen, dann wird der alte Wert gelöscht. Benötigt man den alten Wert, muss man ihn in einer Hilfsvariablen sichern:

```
AltKapital := Kapital;
Kapital := 100.50;
```

Soll der Variablenwert um einen konstanten Wert erhöht werden, dann geschieht dies z. B. durch folgende Anweisung:

```
Kapital := Kapital + 10.25;
```

Will man die Werte zweier Variablen vertauschen, muss man ebenfalls eine Hilfsvariable einführen:

```
Hilfe := AnfKapital;
AnfKapital := EndKapital; { EndKapital und AnfKapital haben denselben Wert  }
EndKapital := Hilfe; { EndKapital erhält den alten Wert von AnfKapital }
```

2.6 Formatieren von Ausgaben

In einer einzigen write- oder writeln-Anweisung können auch mehrere unterschiedlichen Typs ausgegeben werden:

```
writeln ('Das Kapital ist: ', Kapital);
```

Durch diese Anweisung wird sowohl die Zeichenkettenkonstante als auch der Wert der Variablen `Kapital` am Bildschirm ausgegeben. Die Elemente der Ausgabeliste müssen in Pascal durch einen Beistrich getrennt werden.

Neben `read` bzw. `write` gibt es noch die Prozeduren `readln` und `writeln`. Dabei hat `writeln` dieselbe Funktion wie `write`, anschließend springt der Cursor in eine neue Zeile ("ln" = line (egl.) Zeile). Bei `read` und `readln` besteht bei der Eingabe von der Tastatur kein Unterschied. Der Cursor springt nach der Eingabe immer in eine neue Zeile, da das die Taste ↵ als Zeilenvorschub interpretiert wird. Bei Eingabe von der Tastatur sollte daher immer `readln` verwendet werden. `read` und `readln` lesen stets Zeichenketten ein, die anschließend in den entsprechenden Variablentyp umgewandelt werden. Ist dies nicht möglich, kommt es zu einem Laufzeitfehler.

Das Programm ZINS1J.DPR gibt die Werte von Umfang und Fläche im Exponentialformat aus. Z. B.:

1.002000000E+00.

Will man eine geringere Genauigkeit bei der Ausgabe, kann man dies durch

```
writeln ('Das Kapital ist: ', Kapital:10);
```

erreichen, wobei die Zahl nach dem Doppelpunkt die Größe des Feldes angibt, in die der Computer das Ergebnis rechtsbündig schreiben soll.

Man kann die Zahl aber auch in Festkommaformat darstellen, indem man

```
writeln ('Das Kapital ist: ', Kapital:10:2);
```

schreibt. Dabei gibt die 1. Zahl die Feldgröße, die 2. Zahl aber die Anzahl der Nachkommastellen an. Auch hier wird die Zahl rechtbündig in dieses gedachte Feld geschrieben, vorne werden (wenn nötig) Leerzeichen eingefügt, die letzte Dezimalstelle wird gerundet.

2.7 Das Hilfesystem

Fast alle Informationen über Borland Pascal kann man mit dem Hilfesystem erhalten, das in Borland Pascal integriert ist. Durch Drücken von <F1> kann man die Hilfe aufrufen. Sie ist "kontextsensitiv": man erhält Hilfe zu dem Wort, über dem sich der Cursor gerade befindet.

3.0 Einfache Typen

Dies sind Typen, die geordnete Mengen gleichartiger Werte definieren. Sie werden auch skalare Typ genannt. Diese können wieder unterteilt werden in

a) ordinale Typen

b) REAL-Typen

Zu den ordinalen Typen gehören die Teilbereichstypen, die Aufzähltypen und die vordefinierten ordinalen Typen SHORTINT, INTEGER, LONGINT, BYTE, WORD, CHAR und BOOLEAN. In Delphi 6 zusätzlich noch CARDINAL, SMALLINT, INT64 und LONGWORD. Zu den REAL-Typen gehören die Typen REAL, SINGLE, DOUBLE, COMP und EXTENDED. In Delphi 6 zusätzlich noch REAL48 und CURRENCY.

Einzelheiten zu Teilbereichstypen und aufzählbaren Typen sind im Handbuch oder in der Hilfe zu finden.

3.1 Überblick über die REAL-Typen

Typ	Bereich	Speicher	Genauigkeit
SINGLE	$2.3*10^{-38}$ bis $3.4*10^{38}$	4 Bytes	7 - 8 Stellen
REAL48	$2.9*10^{-39}$ bis $1.7*10^{38}$	6 Bytes	11 - 12 Stellen
DOUBLE	$1.1*10^{-308}$ bis $1.7*10^{308}$	8 Bytes	15 - 16 Stellen
EXTENDED	$3.3*10^{-4932}$ bis $1.1*10^{4932}$	10 Bytes	19 - 20 Stellen
COMP	$-2*10^{63} - 1$ bis $-2*10^{63} - 1$	8 Bytes	19 - 20 Stellen
CURRENCY	-922337203685477.5808 bis 922337203685477.5807	8 Bytes	19 - 20 Stellen

In früheren Versionen wurde mit REAL der 6 Byte-Typ bezeichnet. In Delphi 6 ist REAL identisch mit DOUBLE!

3.2 Variablentyp INTEGER

Um Teilmengen ganzer Zahlen darzustellen, sind in Pascal folgende INTEGER-Typen vorhanden:

Type	Bereich			Format
SHORTINT	-128	..	127	8 Bit mit Vorzeichen
BYTE	0	..	255	8 Bit ohne Vorzeichen
SMALLINT	-32768	..	32767	16 Bit mit Vorzeichen
WORD	0	..	65535	16 Bit ohne Vorzeichen
LONGINT	-2147483648	..	2147483647	32 Bit mit Vorzeichen
CARDINAL	0	..	4294967295	32 Bit ohne Vorzeichen
LONGWORD	0	..	4294967295	32 Bit ohne Vorzeichen
INT64	$-2.10^{63} - 1$	..	$2.10^{63} - 1$	64 Bit mit Vorzeichen

Der INTEGER-Typ ist in Delphi 6 äquivalent mit LONGINT, in früheren Versionen war er gleichbedeutend mit SMALLINT.

Allgemein hat die Vereinbarung für den Typ INTEGER folgende Form:

VAR Bezeichner {,Bezeichner} : INTEGER;

Beispiel:

```
VAR i, j, k, Zahl, Index : INTEGER;
```

Für die Addition, Subtraktion und Multiplikation verwendet man wie bei reellen Zahlen die binären Operatoren +, - und *. Die Division erfolgt jedoch durch den Operator DIV, wobei "ganzzahlig" dividiert wird, d. h. vom Quotienten wird nur der ganzzahlige Teil berücksichtigt:

```
i := 6 DIV 3;        (* i hat den Wert 2  *)
j := -9 DIV 2;       (* j hat den Wert -4 *)
```

Den Divisionsrest erhält man durch den Operator MOD.

```
i := 6 mod 3;        (* i hat den Wert 0 *)
j := 9 mod 2;        (* j hat den Wert 1 *)
```

Arithmetische Operationen können mit beliebigen Integertypen durchgeführt werden. Das Ergebnis ist eine Zahl mit 8, 16 oder 32 nach folgenden Regeln:

1.) Konstante werden im kleinsten Typ gespeichert, dessen Wertebereich den angegebenen Wert umfasst

2.) Verknüpfungen mit binären Operatoren erzeugen eine automatische Konvertierung auf jenen Integertyp, dessen Wertebereich beide Operanden umfasst. Das Ergebnis hat ebenfalls dieses Format. Z. B.:

BYTE o INTEGER	ergibt	INTEGER
INTEGER o LONGINT	ergibt	LONGINT

3.) Ausdrücke auf der rechten Seite einer Zuweisung werden unabhängig vom Typ der Variablen auf der linken Seite berechnet.

4.) Variable, die nur ein Byte Speicherplatz belegen (CHAR, BYTE, SHORTINT und Aufzähltypen) werden unter Berücksichtigung des Vorzeichens auf 16 Bit erweitert.

Vorsicht

Beim Rechnen mit ganzen Zahlen ist zu beachten, dass es zu keinem Laufzeitfehler kommt, wenn es dabei z.·B. zu einer Überschreitung der größten oder Unterschreitung der kleinsten Wertes des betreffenden Typs kommt. Der Computer rechnet vielmehr mit den falschen Werten weiter.

3.2.1 Umwandlung reeller Zahlen in ganze Zahlen

Die Umwandlung ganzer Zahlen in reelle Zahlen erfolgt bei jeder Zuweisung automatisch:

```
VAR   r REAL
      i INTEGER;
```

Dann wird durch die Zuweisung

```
      r := i;
```

automatisch der ganzzahlige Wert von i in den entsprechenden REAL-Wert umgewandelt und das Ergebnis *r* zugewiesen. Umgekehrt muss dem Computer aber mitgeteilt werden, wie er den Wert von r in eine INTEGER-Größe umwandeln soll. Pascal bietet dafür die beiden Standardfunktionen `trunc (r)` und `round (r)` an. `trunc (r)` schneidet den Wert von `r` beim Dezimalkomma ab, während `round (r)` den Wert `r` vor einer Umwandlung in eine ganze Zahl rundet. Es gibt daher

`round (-3.5)` den Wert -4,
`trunc (-3.5)` den Wert -3.

Wenn der übergebene REAL-Wert bei `round` oder `trunc` nicht innerhalb von -2147483648 bis 2147483648 liegt, erhält man einen Laufzeitfehler, da die Bereichsgrenze von LONGINT überschritten wurde.

3.2.2 Weitere Rechenoperatoren für ganze Zahlen

Auf ganze Zahlen können in Borland-Pascal noch weitere Operatoren außer

+, -, *, DIV und MOD

angewendet werden. Es sind dies

NOT	invertiert die Bits, durch die die INTEGER-Zahl dargestellt wird
AND, OR, XOR:	verknüpfen die Bits zweier Zahlen durch ein logisches "und", "oder" bzw. "exklusives oder"
SHL, SHR	verschieben die Bits einer ganzen Zahl nach links bzw. nach rechts

Gleiche Priorität haben

AND, SHL, SHR	wie	*, DIV, MOD
OR, XOR	wie	+, -.

Der XOR-Operator wird u. a. in der Kryptographie verwendet. Eine zweimalige XOR-Verknüpfung mit derselben Zahl führt nämlich zur ursprünglichen Zahl zurück. D. h.

(a XOR k) XOR k = a

Das folgende Programm zeigt, wie man eine Zahl von 0 bis 255 damit verschlüsseln und wieder entschlüsseln kann:

```
PROGRAM xor_byte;
uses Win32Crt;
VAR key, cquelle, ckrypt, cretour : BYTE;

BEGIN
   ClrScr;
   writeln ('Gib eine Zahl von 0 bis 255 ein, die als Schlüssel verwendet ',
            'werden soll!');
   write   ('Schlüssel: ');
   readln (key);
   writeln ('Gib eine Zahl von 0 bis 255 ein, die verschlüsselt werden soll!');
   write   ('Zahl: ');
   readln (cquelle);
   ckrypt := cquelle XOR key;
   writeln ('Die verschlüsselte Zahl: ',ckrypt);
   cretour := ckrypt XOR key;
```

```
   writeln ('Der entschlüsselte Zahl: ',cretour);
END.
```

(Programm BEISPIEL1\XOR_BYTE.DPR)

3.3 Variablentyp CHAR

Dieser Typ dient zur Abspeicherung eines einzigen Zeichens aus der ASCII-Zeichenmenge. Die Vereinbarung hat folgende Form:

VAR Bezeichner {, Bezeichner } : CHAR;

Konstante vom Typ CHAR werden wie Zeichenketten unter einfache Anführungszeichen gesetzt:

'A', 'd', '≤' usw.

Sie können in Borland-Pascal aber auch durch das Zeichen "#" gefolgt von einer ganzen Zahl von 0 bis 255 (Wert des ASCII-Codes) festgelegt werden:

#48 = '0', #66 = 'B'.

Für Kontrollzeichen gibt es auch die Darstellung durch das Symbol "^" gefolgt von dem betreffenden Zeichen:

^@ = #0, ^L = #12.

Die Ein- und Ausgabe von Zeichen kann durch `read/readln` bzw. `write/writeln` erfolgen.

3.3.1 Umwandlung ganzer Zahlen in Zeichen

Für die Umwandlung ganzer Zahlen in Zeichen dient die Funktion `Chr`, das Argument muss eine Zahl von 0 bis 255 sein.

Beispiel:

```
write (chr(66));  (* schreibt 'B' *)
```

3.3.2 Umwandlung von Zeichen in ganze Zahlen

Für die umgekehrte Umwandlung ist die Funktion `Ord` zuständig, die den ASCII-Wert des Zeichens liefert.

Beispiel:

```
write (ord('0'));  (* schreibt 48 *)
```

3.4 Variablentyp BOOLEAN

In einer Variablen dieses Typs kann der logische Wert TRUE oder FALSE abgespeichert werden. TRUE und FALSE sind vordefinierte logische Konstante. Die Vereinbarung hat folgende Form:

VAR Bezeichner {, Bezeichner } : BOOLEAN;

Zur Bildung zusammengesetzter logischer Ausdrücke können Vergleichsoperatoren und logische Operatoren verwendet werden. Die Vergleichsoperatoren sind:

=	gleich
>	größer
<	kleiner
>=	größer oder gleich
<=	kleiner oder gleich
<>	ungleich

Diese Operatoren können zwischen skalaren Operanden stehen (Typ INTEGER, REAL, BYTE, CHAR oder BOOLEAN). Es können Operanden vom Typ INTEGER, REAL und BYTE miteinander gemischt werden. Das Ergebnis ist vom Typ BOOLEAN, d. h. entweder wahr (TRUE) oder falsch (FALSE). (TRUE und FALSE sind vordefinierte logische Konstante.)

Beispiel:

```
Zahl  > 3.52238;
Zahl <> Wert;
```

Es können in Pascal auch eigene Variablen vom Typ BOOLEAN definiert werden, die dann nur die Werte TRUE oder FALSE annehmen können (sie belegen in Borland-Pascal ein Byte Speicherplatz):

```
VAR Ungleich, Fertig, IstNull : BOOLEAN;
```

Diesen Variablen können Werte zugewiesen werden:

```
Ungleich := Zahl <> Wert;
Fertig   := FALSE;
IstNull  := Zahl = 0;
```

(Beachte den Unterschied von ":=" und "=" beim letzten Beispiel!) Der Wert eines logischen Ausdrucks oder einer logischen Variablen kann mit *write* ausgegeben werden:

```
write (1=0);
```

schreibt "FALSE" auf den Bildschirm, während eine Eingabe mit *read* nicht möglich ist.

Zum Verknüpfen von zwei logische Ausdrücke dienen die logische Operatoren OR, XOR und AND. Der Verneinungsoperator NOT negiert den Wert eines logischen Ausdrucks. Zu beachten ist, dass folgende Prioritätsreihe in Pascal gilt:

1.) Minusoperator
2.) NOT-Operator
3.) *, /, DIV, MOD, AND, SHR, SHL
4.) +, -, OR, XOR
5.) =, <, >, <=, >=, <>, IN

Bei zusammengesetzten logischen Ausdrücken empfiehlt es sich, Klammern zu setzen:

```
IF (Zahl > 0) AND (Zahl < 10) THEN ...;
IF NOT (Zahl/Wert < 0.342) THEN ...;
```

4.0 Konstantenvereinbarung

In Borland-Pascal sind bereits Konstanten vordefiniert. Z. B.:

Pi	(Typ REAL, Werte 3.14159...)
TRUE	(Typ BOOLEAN, Wert wahr)
FALSE	(Typ BOOLEAN, Wert falsch)
Maxint	(Typ INTEGER, Wert 32767 bzw. bei Delphi 6: 2147483647)

Pascal kennt jedoch auch die Möglichkeit, im Vereinbarungsteil eigene Bezeichner für INTEGER-Zahlen, Zeichenketten oder REAL-Zahlen einzuführen. Die allgemeine Form ist:

CONST Bezeichner = Ausdruck;

"Ausdruck" kann eine Konstante sein. Ab der Version 6.0 ist es aber auch möglich, den Wert der Konstanten berechnen zu lassen:

```
CONST Zwei = 2;
      Drei = Zwei + 1;
      Faktor = Pi/180;
```

Bei großen Programmen erspart man sich oft viel Arbeit, wenn man für logisch zusammenhängende Konstanten eigene Bezeichner vereinbart. Will man später den Wert der Konstanten ändern, braucht man dies nur an einer einzigen Stelle im Vereinbarungsteil zu tun und erspart sich das Suchen im Programm.

5.0 Programmschleifen

Während die bisherigen Programme so aufgebaut waren, dass die Anweisungen in der Reihenfolge durchgeführt werden, in der sie im Anweisungsteil stehen, sollen nun Möglichkeiten besprochen werden, den Programmablauf durch bestimmte Bedingungen zu steuern. In Pascal gibt es dafür zwei Möglichkeiten: Schleifen und Verzweigungen.

5.1 REPEAT-Schleifen

REPEAT-Schleifen sind sogenannte "fußgesteuerte Schleifen". Eine bestimmte Folge von Anweisungen wird so lange wiederholt, bis eine Bedingung erfüllt ist, die am Ende der Anweisungsfolge (am Fuß der Schleife) abgefragt wird. Die erste Anwendung einer REPEAT-Schleife soll an einer Erweiterung der Programms ZINS1J.DPR gezeigt werden.

Das Programm ZINS1J.DPR lässt falsche Eingabewerte (z. B. negative Radien) zu. Wir wollen dies jetzt korrigieren, indem wir den Benutzer zwingen, die Eingabe so lange zu wiederholen, bis er einen positiven Kapitalwert eigegeben hat: Im Pseudocode sieht dies folgendermaßen aus:

Konstante:
ZINSSATZ = 2
Variable:
kapital reell
zinsen reell

Pseudocode: Zinsberechnung

WIEDERHOLE
kapital eingeben
SOLANGE BIS *kapital* > 0
zinsen ←*kapital* * ZINSSATZ / 100
kapital ←*kapital* + *zinsen*
kapital ausgeben

Mit einem Struktogramm wird das Problem so beschrieben:

Konstante:
ZINSSATZ = 2
Variable:
kapital reell
zinsen reell

Struktogramm: Zinsberechnung

kapital eingeben
SOLANGE BIS *kapital* > 0
zinsen ←*kapital* * ZINSSATZ / 100
kapital ←*kapital* + *zinsen*
kapital ausgeben

Eine derartige Schleife wird in Pascal durch die reservierten Wörter REPEAT und UNTIL programmiert. Unser Problem lässt sich durch die Anweisungsfolge

```
REPEAT                    { wiederhole }
   write ('Bitte gib das Kapital ein: ');
   readln (Kapital);
UNTIL Kapital > 0;        { bis Kapital größer als Null }
```

lösen. Alle Anweisungen, die zwischen REPEAT und UNTIL stehen, werden so lange wiederholt, bis die Bedingung, die nach UNTIL steht, erfüllt ist (bis der boolsche Ausdruck den Wert wahr annimmt). Jede REPEAT-Schleife wird also mindestens einmal durchlaufen. (Programm BEISPIEL1\ZINSREP.DPR)

Vorsicht

Durch einen Programmierfehler kann es vorkommen, dass die Abbruchbedingung nie erfüllt ist und der Computer in einer Endlosschleife landet. (Wenn die Pascalprogramme unter DOS mit dem Editor compiliert und gestartet werden, lassen sie sich durch zweimaliges Drücken von <Strg BREAK> abstoppen.)

Im obigen Beispiel werden zwar sinnlose Zahlenwerte, nicht aber die Eingabe eines undgültiger Zahlen abgefangen. Dies lässt sich durch die Compilerschalter (*$I-*) und (*$I+*) und die Funktion `IOResult` erreichen. (Siehe Handbuch.)

Im folgenden Übungsbeispiel soll eine Wertetabelle einer Funktion ausgegeben werden. Dazu muss ein Variablenwert so lange um einen bestimmeten Wert erhöht werden, bis der Endwert erreicht wurde. Die Schleifenanweisungen sind:

```
x := -3.0;          (* Anfangswert außerhalb der Schleife zuweisen *)
REPEAT              (* wiederhole *)
   (* Funktionswerte berechnen *)
   x := x + 0.1;    (* Variablenwert erhöhen *)
UNTIL x > 3.0;      (* bis Endwert erreicht wurde *)
```

5.2 FOR-Schleifen

Während man REPEAT-Schleifen vor allem dann einsetzt, wenn man die Anzahl der Wiederholungen nicht kennt, bis die Abbruchbedingung erfüllt ist, verwendet man FOR-Schleifen dann, wenn die Anzahl der Wiederholungen schon vorher bekannt sind. Die Überprüfung der Abbruchbedingung erfolgt hier vor Beginn der Schleife. Es handelt sich um eine **kopfgesteuerte** Schleife.

Der Computer soll z. B. 10-mal "Hallo" schreiben. Im Pseudocode sieht dieses Problem so aus:

Variable:
i ganzzahlig

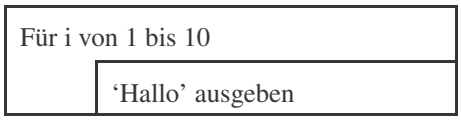
Pseudocode: Halloausgabe

FÜR i VON 1 BIS 10
‘Hallo’ ausgeben

Mit einem Struktogramm wird das Problem so beschrieben:

Variable:
i ganzzahlig

Struktogramm: Halloausgabe



Dies sieht in Pascal bei Verwendung einer einer FOR-Schleife so aus:

```
FOR i := 1 TO 10 DO writeln ('Hallo');
    ↑    ↑     ↑        ↑
    |    |     |    Anweisung, die wiederholt wird
    |    |   Endwert von i
    |    Anfangswert von i
Zählvariable
```

Zu Beginn der FOR-Schleife wird die Zählvariable gleich dem Anfangswert gesetzt und geprüft, ob ihr Wert kleiner oder gleich dem Endwert ist. Ist dies der Fall wird die Anweisung (oder der Anweisungsblock) ausgeführt. Danach wird der Wert der Zählvariablen um 1 erhöht, bis der Endwert erreicht ist.

Nach Beendigung der FOR-Schleife ist der Wert der Zählvariablen unbestimmt. Man sollte sich nicht darauf verlassen, dass die Zählvariable gleich dem Endwert ist (wie dies für die Version 6.0 von Turbo-Pascal zutrifft)!

Mit FOR-Schleifen kann man auch von oben hinunterzählen, indem man statt TO das Wort DOWNTO verwendet:

```
FOR i := 5 DOWNTO -5 DO writeln ('i hat den Wert ',i);
```

Sollen in einer FOR-Schleife mehrere Anweisungen wiederholt werden, dann sind sie durch BEGIN und END einzugrenzen. Z. B.:

```
FOR i := 1 TO 5 DO BEGIN j := i * i;
                   writeln ('i = ',i,' i² = ', j);
                   END; (* FOR *)
```

Die Wörter BEGIN und END fassen alle Anweisungen dazwischen derart zusammen, als ob sie nur eine einzige Anweisung wären. Auch der Anweisungsteil eines Pascalprogramms selber wird durch diese beiden Wörter abgegrenzt. Jedem BEGIN muss immer ein END entsprechen. Vergisst man auf das Wort END erhält man oft beim Compilieren Fehlermel-dungen bei Anweisungen, die zwar syntaktisch richtig sind, die der Computer aber nicht versteht, weil er noch auf ein END wartet.

Man sollte es sich zur Gewohnheit machen, das Wort END stets zu kommentieren und durch Einrücken der Anweisungen deutlich zu machen, welchen Block das Wort END abschließt. Vor END kann der Strichpunkt entfallen.

FOR-Schleifen können auch geschachtelt werden:

```
FOR i := 1 TO 10 DO
    BEGIN
    FOR j := 1 TO 2*i DO write ('*');
    writeln;
    END;
```

Zu beachten ist noch, dass die Zählvariable vom einfachen Typ (z. B.: INTEGER, CHAR, BOOLEAN) sein muss. Benötigt man eine Zählvariable vom Typ REAL, kann man statt einer FOR-Schleife eine WHILE-Schleife programmieren (siehe unten).

In Pascal kann die Schrittweite immer nur +1 (oder -1) sein. Andere Schrittweiten kann man durch Einführung von Hilfsvariablen oder durch Verwendung von WHILE-Schleifen erreichen.

Mit Hilfe einer FOR-Schleife soll das Problem der Mittelwertberechnung von Zahlen gelöst werden. Der Mittelwert von s Zahlen soll nach der Formel:

$$m = \frac{a1 + a2 + \ldots + as}{s}$$

berechnet werden. Zu Beginn des Programms soll die Anzahl s eingegeben werden. Danach wird der Benutzer aufgefordert, s Zahlen einzugeben, von denen der Mittelwert berechnet wird. Der Pseudocode zu diesem Programm sieht so aus:

Variable:
Summe, *Mittel*, *Zahl* reell
Anzahl, *i* ganzzahlig

Pseudocode: Mittelwertberechnung

Anzahl eingeben
Summe ← 0
FÜR i VON 1 BIS *Anzahl*
  Zahl eingeben
  Summe ← *Summe* + *Zahl*
Mittel ← *Summe* / *Anzahl*
Mittel ausgeben

Das entsprechende Pascalprogramm:

```
PROGRAM Mittelwert;
USES Win32Crt;
VAR Summe, Mittel, Zahl : REAL;
    Anzahl, i           : INTEGER; { ganze Zahlen }

BEGIN ClrScr;   { Beginn des Hauptprogramms }
   write ('Gib die Anzahl der Zahlen ein:  ');
   readln (Anzahl); writeln;
   Summe:=0;    { in Pascal werden die Variablen nicht Null gesetzt!!! }
   FOR i :=1 to Anzahl DO
       BEGIN write('Gib die ',i:2,'. Zahl ein: ');
       readln (Zahl);
       Summe := Summe+Zahl;
       END;     { FOR-Schleife ist hier zu Ende }
   writeln;
   Mittel := Summe/Anzahl;
   writeln ('Der Mittelwert lautet: ',Mittel:10:3);
END.
```

(Programm BEISPIEL1\MITTEL.DPR.)

5.3 WHILE-Schleifen

Dies ist der 3. Schleifentyp, den Pascal zur Verfügung stellt. Die WHILE-Schleifen ähneln sehr den REPEAT-Schleifen, sie werden jedoch durch eine Bedingung gesteuert, die am Beginn steht, während die Steuerbedingung bei den REPEAT-Schleifen am Ende steht. Es handelt sich hier ebenfalls um eine kopfgesteuerte Schleife. Dies hat zur Folge, dass die REPEAT-Schleifen mindestens einmal durchlaufen werden, während WHILE- und FOR-Schleifen unter Umständen auch kein einziges Mal ausgeführt werden.

Die allgemeine Form lautet:

WHILE logischer Ausdruck DO Anweisung(sblock)

Die Anweisung wird so lange durchgeführt, als der logische Ausdruck wahr ist. Ähnlich wie bei REPEAT-Schleifen muss man auch hier darauf achten, keine Endlosschleifen zu programmieren!

Als Beispiel soll eine Variante der Eingabeabsicherung dienen. Eine reelle Zahl größer als Null soll eingelesen werden. Bei einer Fehleingabe soll der Benutzer durch eine Meldung am Bildschirm zu einer neuerlichen Eingabe aufgefordert werden.

Pseudocode, Struktogrammund Pascalprogramm sehen so aus:

Variable:
Zahl reell

Pseudocode: Eingabeprüfung

Zahl eingeben
WIEDERHOLE SOLANGE *Zahl* <= 0
  'Zahl muss größer als Null sein!' ausgeben
  'Gib eine Zahl ein: ' ausgeben
  Zahl eingeben

Variable:
Zahl reell

Struktogramm: Eingabeprüfung

Zahl eingeben	
WIEDERHOLE SOLANGE *Zahl* <=0	
	‘Zahl muss größer als Null sein!’ ausgeben
	‘Gib eine Zahl ein: ’ ausgeben
	Zahl eingeben

```
Program WHILE_Test;
VAR Zahl : REAL;
BEGIN
   write ('Bitte gib eine Zahl ein: ');
   readln (Zahl);
   WHILE Zahl <= 0 DO   (* wiederhole, wenn Eingabe falsch *)
         BEGIN
         writeln ('Die Zahl muss größer als Null sein!');
         write ('Bitte gib eine Zahl ein: ');
         readln (Zahl);
         END; (* WHILE *)
END.
```

In unserem Beispiel folgt auf das Wort DO ein ganzer Anweisungsblock, der von BEGIN und END eingeschlossen ist. Ähnlich wie bei der FOR-Schleife können bei einer einzigen Anweisung die Wörter BEGIN und END wegfallen oder es kann auch eine Leeranweisung folgen. Ein Beispiel dafür ist:

```
WHILE NOT keypressed DO;
```

Die in Unit Win32Crt definierte Funktion keypressed ist wahr, wenn eine Taste gedrückt wurde (genauer: wenn sich ein Zeichen im Tastaturpuffer befindet). Die WHILE-Schleife wird also so lange durchlaufen (d. h. es wird nichts getan), bis eine Taste gedrückt wird. Man kann dadurch Haltepunkte ins Programm einbauen, die durch Drücken einer Taste vom Benutzer beendet werden können.

Abschließend soll aber noch das Programm WHILE_Test zu einem sinnvollen Programm ausgebaut werden. Das Programm DREIECK.DPR soll aus der eingebenen Seitenlänge den Umfang, die Höhe und den Flächeninhalt eines gleichseitigen Dreiecks ausrechnen. Fehlerhafte Eingaben (Seitenlängen, die kleiner oder gleich Null sind, sollen mit einer WHILE-Schleife abgefangen werden. Die entsprechenden Formeln sind für die Seitenlänge s:

$u = 3s$ $\quad h = \frac{s\sqrt{3}}{2}$ $\quad A = \frac{s^2\sqrt{3}}{4}$.

Es ist $\sqrt{3} = 1.7320508$. Diese Zahl kann auch mit Hilfe der Standardfunktion Quadratwurzel (`sqrt`) berechnet werden:

```
wurz3 := sqrt(3);
```

```
PROGRAM Dreieck;
USES Win32Crt;
VAR Seite, Umfang, Flaeche : REAL; (* 3 Variable vereinbaren *)

BEGIN ClrScr;
   writeln ('Berechnung des Umfangs und Flächeninhalts eines ',
            'gleichseitigen Dreiecks.');
   writeln;
   write ('Bitte gib die Seitenlänge eines gleichseitigen Dreiecks ein: ');
   readln (Seite);
   WHILE Seite <= 0 DO BEGIN  (* wiederhole, wenn Eingabe falsch *)
         writeln ('Die Seitenlänge muss größer als Null sein!');
         write ('Bitte gib die Seitenlänge ein: ');
         readln (Seite);
         END;
```

```
    Umfang := 3*Seite;
    Flaeche := Seite*Seite*sqrt(3)/4;
    writeln;
    writeln ('Der Umfang ist: ', Umfang:10:4);   (* 4 Dezimalstellen *)
    writeln ('Die Fläche ist: ', Flaeche:10:4);
    writeln; writeln;
    writeln ('Ende des Programms "gleichzeitiges Dreiecks"');
END.
```

(Programm BEISPIEL1\DREIECK.DPR.)

6.0 Unterprogramme

Bei umfangreichen Programmen ist eine Gliederung in einfache und kleine Programmabschnitte vorteilhaft. Diese Möglichkeit wird von Pascal sehr gut unterstützt. Vorerst werden wir Unterprogramme (oder Prozeduren) als neudefinierte Befehlswörter kennenlernen, bevor wir uns der Übergabe von Variablen zuwenden.

6.1 Prozeduren ohne Parameter

Unterprogrammen müssen folgende Form haben:

PROCEDURE Prozedurname;
Vereinbarungsteil der Prozedur
BEGIN Anweisungen der Prozedur
END;

Die Vereinbarung von Prozeduren erfolgt im Vereinbarungsteil des Pascalprogramms. Der Aufruf erfolgt im Hauptprogramm durch den Prozedurnamen.

Eine Prozedur *Titel* soll auf den Bildschirm den Namen des Programmautors schreiben. Die Prozedur *Ende* soll das Programmende am Bildschirm anzeigen und auf einen Tastendruck warten.

```
PROGRAM UnterprogrammTest;
  USES Win32Crt;

  PROCEDURE Titel;
    BEGIN ClrScr;
       writeln (' Programm von ....');
       write (' Drücke auf die Eingabetaste!');
       readln; ClrScr;
    END;

  PROCEDURE Ende;
    BEGIN write (' Ende ... Drücke auf die Eingabetaste.');
       readln;
    END;

BEGIN
   Titel;  (* Beginn des Hauptprogramms *)
   .....   (* Anweisungen des Hauptprogramms *)
   Ende;
END.
```

(Datei BEISPIEL1\KREISPRO.DPR.)

6.2 Standardprozeduren ohne Parameter

Hier soll ein kurzer Überblick der Standardprozeduren von Borland-Pascal gegeben werden. Weitere Beispiel sind über die Hilfe oder im Handbuch zu finden:

Exit	Verlassen des Unterprogramms
Halt	Beenden des Pascalprogramms; wahlweise kann als Paramter eine Zahl übergeben werden, die von DOS aus mit dem Befehl ERRORLEVEL ermittelt werden kann.
Randomize	initialisiert den Zufallsgenerator

6.3 Prozeduren mit Parameterm

Häufig benötigt man ein Unterprogramm, das bei jedem Aufruf eine bestimmte Aktion mit anderen Variablenwerten durchführen soll. Eine Prozedur *Leerzeilen* soll soll eine beliebig viele Leerzeilen am Bildschirm ausgeben. Die Anzahl der Leerzeilen wird als Parameter oder Argumente an die Prozedur übergeben. Die allgemeine Defintion sieht folgendermaßen aus:

PROCEDURE Prozedurname (Argumentname {,Argumentname} : Typ);

In unserem Beispiel sieht dies folgendermaßen aus:

```
PROCEDURE Leerzeilen (Zahl : INTEGER);
 VAR Zeile : INTEGER;
 BEGIN
    FOR Zeile := 1 TO Zahl DO writeln;
 END;
```

In diesem Beispiel ist die Variable `Zeile` eine "lokale Variable", die nur innnerhalb der Prozedur *Leerzeichen* verwendet werden kann. Im Hauptprogramm können Variable mit dem gleichen Namen definiert sein, die man "globale Variable" nennt. Wird in einem Unterprogramm der Wert einer lokalen Variablen geändert, dann bleibt der Wert der gleichnamigen globalen Variablen unverändert. Zählvariable für FOR-Schleifen müssen immer lokal definiert werden!

Auch die Parameter `Text` bzw. `Zahl` sind nur innerhalb der Prozedur gültig. Auch bei diesen Parametern führt eine Wertzuweisung im Unterprogramm zu keiner Wertänderung im Hauptprogramm. Man nennt diese Argumente auch Eingabeparameter oder Wertparameter.

Das Hauptprogramm hat folgende Form:

```
BEGIN
   ClrScr;
   Leerzeilen (10);
   writeln ('Text nach den Leerzeilen.');
   Leerzeilen (10);
   writeln ('Drücke auf die Eingabetaste!');
   readln;
END.
```

(Programm BEISPIEL1\MITTE.DPR.)

Sollen von einer Prozedur auch Werte an das Hauptprogramm übergeben werden, dann ist vor den Argumentnamen das reservierte Wort

VAR

zu setzen. Man nennt diesen Parameter dann "VAR-Parameter" oder "Ein/Ausgabeparameter". Es erfolgt dann ein Aufruf durch Referenz (call by reference). Eine Veränderung dieser Argumente bewirkt eine Änderung der aktuellen Parameter, die beim Prozeduraufruf verwendet werden.

Die folgende Prozedur *Vertausche* soll die Werte zweier Variablen vertauschen:

```
PROGRAM Tausch;

USES Win32Crt;

PROCEDURE Vertausche (VAR X, Y : REAL);
 VAR Hilfe : REAL;
 BEGIN
    Hilfe := X;
    X := Y;
    Y := Hilfe;
 END;

VAR Zahl1, Zahl2 : REAL;

BEGIN
   ClrScr;
   Zahl1 := 1.0;
   Zahl2 := 2.0;
   writeln ('Die 1. Zahl vor dem Tausch ist:  ',Zahl1:10:2);
   writeln ('Die 2. Zahl vor dem Tausch ist:  ',Zahl2:10:2);
   Vertausche (Zahl1, Zahl2);
```

```
   writeln ('Die 1. Zahl nach dem Tausch ist: ',Zahl1:10:2);
   writeln ('Die 2. Zahl nach dem Tausch ist: ',Zahl2:10:2);
END.
```

(Programm BEISPIEL1\TAUSCH.DPR.)

Da über die Ein/Ausgabeparameter Werte an die aktuellen Parameter übergeben werden, können diese beim Aufruf nur Variable sein. Für Wertparameter können beim Aufruf sowohl Variable, Ausdrücke oder Konstante verwendet werden. Der Aufruf

```
Vertausche (3,4);
```

würde zu einem Compilerfehler führen.

6.4 Standardprozeduren mit Parametern

Häufig verwendete Standardprozeduren sind:

Inc(var X [; N: Longint]);	Erhöht den Wert von X um 1. Ist N angegeben, wird X um N erhöht.
Dec(var X[; N: Longint]);	Vermindert den Wert von X um 1. Ist N angegeben, wird X um N vermindert. In beiden Fällen muss X vom ordinalen Typ sein.

7.0 Strings (Zeichenketten)

Strings gehören zu den strukturierten Datentypen und sind in vielen Punkten Arrays (Feldern) sehr ähnlich, können aber unterschiedliche Länge von 0 bis zu einem vereinbarten Maximalwert haben. Der Typ STRING hat allgemein die Form:

STRING [Maximallänge];

wobei Maximallänge eine ganzzahlige Konstante von 1 bis 255 sein kann. Beispiele:

```
TYPE str80 = STRING[80];
VAR  s1, s2 : STRING[50];
     s      : str80;
     smax   : STRING;
```

Die Angabe der Maximallänge kann ab Version 4.0 auch entfallen. Der Typ STRING ist gleichbedeutend mit STRING [255].

Dabei stellt z. B. die Variable *s* ein Feld vom Typ CHAR der Länge 80 dar, in dem die einzelnen Zeichen durch

s[1], s[2], ... s[80]

erreicht werden können. Dazu kommt noch ein Byte, in dem die aktuelle Länge des Strings gespeichert ist. Dieses Byte kann durch `s[0]` angesprochen werden, wobei zu beachten ist, dass die einzelnen Komponenten des Strings (also auch `s[0]`) vom Typ CHAR sind. Es belegen also Variablen vom Typ STRING um ein Byte mehr als die Maximallänge angibt.

In Delphi gibt es einen neuen Stringtyp, der im 1. Byte nicht die Länge gespeichert hat, sondern wie in der Programmiersprache C durch ein Zeichen mit dem ASCII-Code 0 begrenzt wird. Dadurch sind auch Zeichenketten bis zu einer Länge von 64000 möglich.

Die Länge eines Strings lässt sich durch die Standardfunktion `length (St)` leicht bestimmen. (Der Funktionswert von length ist vom Typ INTEGER.) Hat s z. B. den Wert 'ABCD', dann ergibt `length (s)` den Wert 4. Durch

```
s = '';
```

wird s ein Leerstring zugewiesen, der die Länge 0 hat.

Als kleines Beispiel soll ein Programm dienen, das eine Zeichenkette einliest und anschließend diese Zeichenkette verkehrt ausgibt:

```
PROGRAM Umkehr;
 USES Win32Crt;
 VAR ein : STRING;
     i   : INTEGER;

 BEGIN ClrScr;
    write ('Bitte gib eine Zeichenkette ein: ');
    readln (ein);
```

```
      write ('Ich gebe den Text verkehrt aus:  ');
      FOR i := length (ein) DOWNTO 1 DO write (ein[i]);
   END.
```

(Programm BEISPIEL1\UMKEHR.DPR.)

7.1 Stringoperationen

Auch wenn Strings verschiedene Maximallänge haben, können ihre Werte einander zugewiesen werden:

```
VAR s5  : STRING [5];
    s80 : STRING [80];
 BEGIN
   s80 := 'Pascal';
   s5  := s80;
 END.
```

Dabei wird aber die Zeichenkette 'Pascal' bei der Zuweisung auf mit dem 5. Zeichen abgeschnitten, da s5 die Maximallänge 5 hat. Der Wert von *s5* ist daher 'Pasca'.

Durch das Pluszeichen könne Strings miteinander verkettet werden:

```
s80 := 'Das '+'ist '+'ein '+'Test!';
```

Die oben programmierte Ausgabe einer Zeichenkette in verkehrter Reihenfolge könnte man auch so lösen, dass man zuerst den String verkehrt zusammensetzt und anschließend ausgibt:

```
PROGRAM Umkehrpro;
  USES Win32Crt;
PROCEDURE ReversString (ein : STRING; VAR aus : STRING);
 VAR i : INTEGER;
 BEGIN
    aus := '';
    FOR i := length (ein) DOWNTO 1 DO aus := aus + ein[i];
 END;

 VAR ein, aus : STRING;

 BEGIN ClrScr;
    writeln ('Dieses Programm gibt eine Zeichenkette verkehrt aus':58);
    writeln;
    write ('Bitte gib eine Zeichenkette ein: ');
    readln (ein);
    ReversString (ein, aus);
    write ('Ich gebe den Text verkehrt aus:  ');
    writeln (aus);
 END.
```

(Programm BEISPIEL1\UMKEHRPRO.DPR.)

Ebenso lassen sich die Vergleichsoperatoren =, <, > usw. auf Zeichenketten anwenden. Zwei Strings sind genau dann gleich, wenn sie in der aktuellen Länge und in allen Zeichen übereinstimmen. Bei den anderen Vergleichsoperatoren werden die Strings Zeichen für Zeichen von links nach rechts gemäß ihrem ASCII-Wert verglichen. Es ist z. B.:

'1234' < '1235'	wahr
'Test' > 'Tex'	falsch
'Pasca' < 'Pascal'	wahr.

Borland-Pascal stellt eine Reihe von nützlichen Stringprozeduren und -funktionen zur Verfügung, die im folgenden kurz zusammengestellt werden sollen. Nähere Erläuterungen dazu finden sich im Referenzhandbuch.

7.2 Stringprozeduren

Delete (St, Pos, Num)

löscht vom String St ab Position Pos Num Zeichen.

Insert (Von, Ein, Pos)

fügt den String Von in den String Ein ab Position Pos ein.

Str (Zahl, St)
wandelt den Wert von Zahl in einen String St um. Formatierungsbefehle sind erlaubt:
Str(R:10:3,St) oder Str(n:5,St)
Val (St, Zahl, Code)
wandelt String St in einen REAL- oder INTEGER-Wert um, der in Zahl gespeichert wird. Code hat Wert des 1. fehlerhaften Zeichen oder 0 bei keinem Fehler.

7.3 Stringfunktionen

Copy (St, Pos, Num)
Funktionswert ist ein Teilstring von St ab Position Pos mit Num Zeichen.
Concat (St1, St2, .. StN)
Funktionswert ist der Gesamtstring von St1 bis StN. (Entspricht St1+...+StN)
Length (St) (Typ: INTEGER)
Funktionswert ist die Länge des Strings St.
Pos (St1, St2) (Typ: INTEGER)
Untersucht, ob String St1 in String St2 vorkommt. Funktionswert ist die Position des 1. übereinstimmenden Zeichens, sonst Null.

8.0 Programmverzweigungen

8.1 IF-Anweisung

Abhängig davon, ob eine Bedingung erfüllt ist (ein logischer Ausdruck den Wert TRUE hat), wird entweder eine Anweisung1 oder eine Anweisung2 ausgeführt. Im Pseudocode bzw. Struktogramm sieht dies so aus:

Pseudocode: Verzweigung

```
WENN Bedingung erfüllt
   DANN Anweisung1
   SONST Anweisung2
```

Struktogramm: Verzweigung

Bedingung erfüllt	
Ja	Nein
Anweisung1	Anweisung2

In Pascal hat die allgemeine Form der IF-Anweisung folgende Form:

```
IF logischer Ausdruck
   THEN Anweisung1
   ELSE Anweisung2;
```

wobei der ELSE-Zweig entfallen kann. Einige Beispiele:

```
IF Zahl < 0 THEN writeln ('Zahl ist negativ!')
            ELSE writeln ('Zahl ist positiv oder Null!');
```

oder ohne ELSE-Zweig:

```
IF Zahl = 0 THEN writeln ('Zahl ist gleich Null');
```

Auswahlabfragen können auch geschachtelt werden:

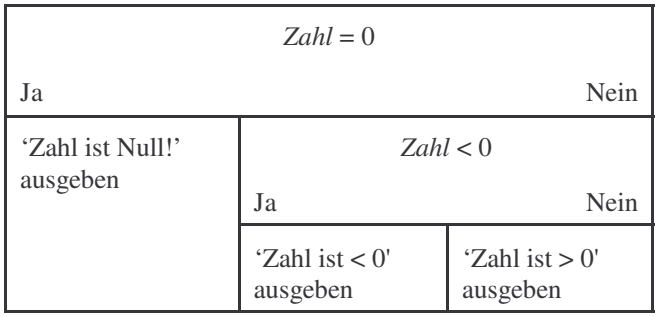
```
IF Zahl = 0
   THEN writeln ('Zahl ist Null!')
   ELSE IF Zahl < 0
           THEN writeln ('Zahl ist negativ!')
           ELSE writeln ('Zahl ist positiv!');
```

Variable
Zahl reell

Pseudocode: Mehrfachverzweigung

```
WENN Zahl = 0
   DANN 'Zahl ist Null!' ausgeben
   SONST WENN Zahl < 0
                DANN 'Zahl ist < 0' ausgeben
                SONST 'Zahl ist > 0' ausgeben
```

Struktogramm: Mehrfachverzweigung



Zu beachten ist, dass vor ELSE nie ein Strichpunkt stehen darf, da dieser die gesamte IF-Anweisung abschließt und anzeigt, dass kein ELSE-Zweig folgt.

Bei verschachtelten IF-Anweisungen ist zu beachten, dass der ELSE-Zweig immer zur letzten IF-Anweisung gehört, die noch keinen ELSE-Teil besitzt:

```
IF Zahl <> 0
   THEN IF Zahl < 0 THEN writeln ('Zahl ist negativ!')
                    ELSE writeln ('Zahl ist positiv!');
```

Notfalls kann durch Einfügen von BEGIN ... END die Mehrdeutigkeit beseitigt werden:

```
IF Zahl <> 0
   THEN BEGIN
        IF Zahl < 0 THEN writeln ('Zahl ist negativ!')
                    ELSE writeln ('Zahl ist positiv!');
         END;
```

Diese Beispiele sollen jetzt zu einem einfachen Programm erweitert werden, das nach Eingabe einer Zahl feststellt, ob diese kleiner als Null bzw. größer oder gleich Null ist. (Programm BEISPIEL1\VERGLEICH.DPR.)

Für viele Aufgabenstellungen wird eine Prozedur zur Umwandlung von Dezimal- in Binärzahlen benötigt. Die folgende Prozedur `IntToBin` löst diese Aufgabe und zeigt nochmals die Verwendung der Operatoren `MOD` und `DIV`:

```
PROGRAM Binaerform;
USES Win32Crt;
Procedure IntToBin (i : LONGWORD; VAR bin : STRING);
 VAR rest : LONGWORD;
 BEGIN
    bin := '';
    WHILE i <> 0 DO BEGIN
       rest := i MOD 2;
       if rest = 1
          THEN bin := '1'+bin
          ELSE bin := '0'+bin;
       i := i DIV 2;
    END;
    if bin = '' THEN bin := '0';
 END;

VAR zahl : LONGWORD;
    bin  : STRING;

BEGIN
   ClrScr;
   write ('Gib eine natürliche Zahl ein: ');
   readln (zahl);
   IntToBin (zahl, bin);
   writeln ('Diese Zahl in Binärform: ',bin);
END.
```

(Programm BEISPIEL1\BINAERFORM.DPR.)

8.2 CASE-Anweisung

Diese Anweisung ermöglicht die Verzweigung zu beliebig vielen Alternativen in Abhängigkeit von einem Ausdruck (dem Sortierer). Die allgemeine Form dieser Anweisung ist:

```
CASE Ausdruck  OF Konstante   :  Anweisung;
          { Konstante : Anweisung; }
          ELSE Anweisung;
END;
```

Dabei muss der Sortierausdruck vom einfachen Typ ausgenommen vom Typ REAL sein. Die Konstanten müssen vom selben Typ wie der Sortierer sein. Der ELSE-Zweig ist eine Erweiterung von Borland-Pascal und kann entfallen.

Im folgenden Beispiel wird nach Eingabe einer natürlichen Zahl der entsprechende Wochentag ausgegeben (1 für Montag, 2 für Dienstag usw.):

```
readln(Zahl);
CASE Zahl OF
     1: writeln('Das ist der Montag.');
     2: writeln('Das ist der Dienstag.');
     3: writeln('Das ist der Mittwoch.');
     4: writeln('Das ist der Donnerstag.');
     5: writeln('Das ist der Freitag.');
     6: writeln('Das ist der Samstag.');
     7: writeln('Das ist der Sonntag.');
   ELSE writeln('Falsche Eingabe!');
END; (* CASE *)
```

(Das gesamte Programm ist unter BEISPIEL1\WOCHE.DPR zu finden.)

Es können auch mehrere Konstante durch Beistriche getrennt vor dem Doppelpunkt stehen:

```
readln(Zahl);
CASE Zahl OF
     1,2,3,4,5,6 : writeln('Das ist der Wochentag.');
     7           : writeln('Das ist der Sonntag.');
END;
```

oder noch einfacher durch den Teilbereichsbegrenzer "..":

```
readln(Zahl);
CASE Zahl OF
  1..6 : writeln('Das ist der Wochentag.');
  7    : writeln('Das ist der Sonntag.');
END;
```

Statt einer einfachen Anweisung kann auf den Doppelpunkt auch ein durch BEGIN und END begrenzter Anweisungsblock folgen.

9.0 Liste der Beispielprogramme

Dateiname	Stoffgebiet	Seite
BINAERFORM.PAS	Strings	18
DREIECK.PAS	WHILE-Schleife	13
KREISPRO.DPR	Unterprogramme	13
MITTE.DPR	Prozeduren	14
MITTEL.DPR	FOR-Schleifen	11
TAUSCH.DPR	Prozeduren	15
UMKEHR.DPR	Strings	16
UMKEHRPRO.DPR	Strings	16
VERGLEICH.DPR	IF-Abfrage	18
WOCHE.DPR	CASE-Anweisung	19
XOR_BYTE.DPR	XOR-Operation	7
ZINS1J.DPR	REAL-Zahlen	1
ZINSREP.DPR	REPEAT-Schleifen	9

10.0 Stichwortverzeichnis

AND 2, 6, 8
Anweisungsteil 1, 3, 8, 10
Argument 7
ARRAY 2
Aufzähltypen 5, 6
BEGIN 1, 2, 10, 18, 19
Bezeichner 2, 3, 5, 7, 8
BOOLEAN 5, 7, 8, 10
BREAK 9
BYTE 5-7, 15, 19
CARDINAL 5
CASE 2, 19
CHAR 5-7, 10
Chr 7
Close 1
ClrScr 2
COMP 5
Compilerschalter 9
  $I 9
Compilieren 2, 10
Concat 17
CONST 1-3, 8
Copy 17
CURRENCY 5
DEC 15
Definitionsteil 1
Delete 16
DIV 2, 5, 6, 8, 18
DOUBLE 5
DOWNTO 2, 10, 16
Editor 3, 9
ELSE 2, 17-19
END 1, 10, 18, 19
Exit 13
Exponentialformat 4
EXTENDED 5
FALSE 7, 8
Feld 4, 15
Festkommaformat 4
FILE 1
FOR 9
FUNCTION 1
Halt 13
Handbuch 5, 9, 13
Hauptprogramm 1
Hilfesystem 4
Identifier 2
IF 17
IN 8
INC 15
Insert 16
INT64 5
INTEGER 5, 10
IOResult 9
Keypressed 12
Konstante 3, 7, 19
Laufzeitfehler 4, 6
Leerstring 15
Length 15-17
Line 4
Ln 4
LONGINT 5, 6, 15
LONGWORD 5, 18
MaxInt 8
Mengen 5
MOD 5, 6, 8, 18
NOT 6, 8, 12
OF 19
OR 6, 8
Ord 7
Parameter
  Ein/Ausgabeparameter 14
  Eingabeparameter 14
  VAR-Parameter 14
  Wertparameter 14
Pi 8
Pos 16, 17
PROCEDURE 1, 13, 14
PROGRAM 1, 2
Programmkopf 1, 2
Prozedur 13
Randomize 13
Read 2, 7
REAL 1-3, 5-8, 10-12, 14, 17, 19
REAL48 5
Rechenoperator
  binär 3
Rechenoperatoren 3, 6
REPEAT 8, 9, 11, 19
Reservierte Wörter 2
Round 6
SHL 6, 8
SHORTINT 5, 6
SHR 6, 8
SINGLE 5
SMALLINT 5
Sqrt 12, 13
Standardfunktionen 6
Standardprozeduren 13, 15
Statuszeile 1
Str 17
STRING 3, 15-18
Struktogramm 9, 10, 12, 17, 18
Teilbereich 19
TO 10
Trennzeichen 3
TRUE 7, 8, 17
Trunc 6
Typ
  Aufzähl- 5
  BOOLEAN 5, 7
  BYTE 5, 7
  CHAR 5, 7, 15
  DOUBLE 5
  einfach 5, 10, 19
  EXTENDED 5

INTEGER 5, 7
LONGINT 5
ordinal 5
REAL 3, 5, 7, 19
SHORTINT 5
SINGLE 5
skalar 5
Teilbereichs- 5
WORD 5
TYPE 1, 5, 15
Unit 2
Crt 1, 12
Win32Crt 1
WinCrt 2
UNTIL 9
Val 17
VAR 14
Variable
global 14
lokal 14
Vereinbarungsteil 1, 8, 13
Vergleichsoperatoren 7, 16
WHILE 10-12, 18, 19
WORD 5
Wortsymbol 1, 2
Wortsymbole 2
Write 4, 7, 8
XOR 6-8, 19
Zuweisungsoperator 3

11.0 Inhaltsverzeichnis

1.0 Pascal mit Delphi 1
2.0 Lineare Programme 1
2.1 Zinsenberechnung 1
2.2 Aufbau eines Pascalprogramms 1
2.3 Erläuterung des Programms 2
2.4 Syntax eines Pascalprogramms 2
2.5 Variablenvereinbarung und Wertzuweisung 3
2.6 Formatieren von Ausgaben 4
2.7 Das Hilfesystem 4
3.0 Einfache Typen 5
3.1 Überblick über die REAL-Typen 5
3.2 Variablentyp INTEGER 5
3.2.1 Umwandlung reeller Zahlen in ganze Zahlen 6
3.2.2 Weitere Rechenoperatoren für ganze Zahlen 6
3.3 Variablentyp CHAR 7
3.3.1 Umwandlung ganzer Zahlen in Zeichen 7
3.3.2 Umwandlung von Zeichen in ganze Zahlen 7
3.4 Variablentyp BOOLEAN 7
4.0 Konstantenvereinbarung 8
5.0 Programmschleifen 8
5.1 REPEAT-Schleifen 8
5.2 FOR-Schleifen 9
5.3 WHILE-Schleifen 11
6.0 Unterprogramme 13
6.1 Prozeduren ohne Parameter 13
6.2 Standardprozeduren ohne Parameter 13
6.3 Prozeduren mit Parameterm 14
6.4 Standardprozeduren mit Parametern 15
7.0 Strings (Zeichenketten) 15
7.1 Stringoperationen 16
7.2 Stringprozeduren 16
7.3 Stringfunktionen 17
8.0 Programmverzweigungen 17
8.1 IF-Anweisung 17
8.2 CASE-Anweisung 19
9.0 Liste der Beispielprogramme 19
10.0 Stichwortverzeichnis 20
11.0 Inhaltsverzeichnis 21