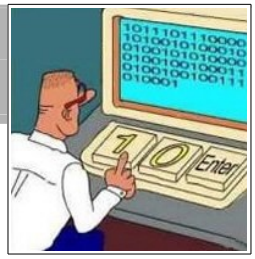
Wichtige Delphi-Befehle und -Routinen – eine Kurzreferenz



Die folgende Kurzreferenz soll ihrer Bezeichnung nach lediglich eine grobe Übersicht über die wichtigsten Befehle, Funktionen und Routinen in Delphi geben. Für weitergehende Informationen sei auf die Delphi-Hilfe, entsprechende Fachliteratur und einschlägige Internet-Portale/-Foren verwiesen.

1) Grundlegendes / Allgemeines

Folgende grundlegende Regelungen und Prämissen passen in keine der anderen Kategorien und finden hier Erwähnung.

Struktur	Bedeutung	Beispiel(e)

:=	Zuweisung (Doppelpunkt – gleich):
	
	Mit Hilfe dieser Struktur erfolgt die Wertzuweisung an z. B. eine Variable. Form: <Variablenname> := <Zuzuweisender Wert>;
		`x := 2;`
		
		`[x hat nun den Wert 2, also x = 2.]`

		`x := x + 1;`
		`EdtAusgabe.Text := 'Info ist geil!';`
;	Befehlsabschluss (Semikolon): Jede Anweisung in Delphi muss mit einem Semikolon abgeschlossen werden.	`x := 2;`
		`ShowMessage('Info ist geil!');`
		`if x = 8 then x := x + 3` `  else x := x - 1;`

begin ... end;	Mit begin und end setzt man in Delphi eine Art „Klammer“ um mehrere Befehle, die folglich als ein Block zusammen betrachtet und ausgeführt bzw. ignoriert werden.
	Diese Klammerung ist vergleichbar mit der Rechenklammer in der Mathematik. Die Klammerung mit begin und end kann sehr vielfältig eingesetzt werden.
		`procedure TForm1.BtnCloseClick[...];`
		`begin`
		`  ...`
		`end;`
		
		`[Alle Befehle zwischen begin und end gehören zur Prozedur und werden ausgeführt]`

		`if Fehlercode = 3 then begin`
		`  Windows.Beep;`
		`  Windows.Beep;`
		`  Windows.Beep;`
		`end`
		`else if Fehlercode = 2 then begin`
		`  Windows.Beep;`
		`  Windows.Beep;`
		`end;`
		
		`[Wenn Fehlercode = 3, dann 3x Beep-Ton; sonst, wenn Fehlercode = 2, nur 2x Beep.]`

<Komponentenname>.<Eigenschaft>	Allgemeine Form zum Zugriff auf eine Eigenschaft einer Komponente.
	
	(Weitere komponentenspezifische Anweisungen s. u.)
		`EdtAusgabe.Text := 'Hallo Welt!';`
		
		`[„EdtAusgabe“ ist der Name der Komponente Edit-Feld, „.“ ist das Trennzeichen und „Text“ ist die Eigenschaft des Edit-Feldes (hier: Eingabe eines Textes).]`

2) Kommentare

Kommentare dienen beim Programmieren zum strukturieren und kommentieren des Quellcodes, z. B. Formulierung von Überschriften zu Befehlsblöcken oder knappes Erklären des Codes, was man sich dabei gedacht hat. Die Kommentare werden von Delphi selbst ignoriert und dienen lediglich als Hilfe für den Menschen zum Lesen des Codes. Auch kann man mit Kommentaren Codeblöcke zu Testzwecken „einkommentieren“, damit diese bei der Programmausführung ignoriert werden. Der Vorteil: Man muss den Code nicht löschen und später ggf. wieder hinschreiben. Delphi unterscheidet drei Arten von Kommentaren. Man sollte sich für ein oder zwei Arten entscheiden und diese gezielt (d. h. nicht zu viel, nicht zu wenig) im Quellcode einbauen.

Typ	Bedeutung	Beispiel(e)
//	Einzeiliger Kommentar, gilt also nur bis zum Ende der Zeile im Quelltexteditor.	`x := 0;  // Initialisierung`
{ und *}*	Mehrzeilige Kommentar (alles zwischen der öffnenden und schließenden Klammer).	`x := 0;  { Die Variable x wird mit dem Wert Null initialisiert }`
*(** und **)*	Ebenfalls mehrzeilige Kommentar. Er ist der „stärkste“ Kommentar und schließt wiederum jeden anderen Kommentar ein.	`(* VORUEBERGEHENDES EINKOMMENTIEREN ZU TESTZWECKEN` `x := 0;  { Die Variable x wird mit dem Wert Null initialisiert }  *)`

3) Variablen(-typen) und Konstanten

Variablen sind in ihrem Wert („Inhalt“) veränderbare Platzhalter für programmrelevante Daten. Dabei ist jede Variable von einem bestimmten Variablentyp, die in Delphi klar unterschieden werden. In erster Linie sind dies Typen zur Speicherung von Ganzzahlen (in Delphi genannt Integer-Typen), Gleit- bzw. Fließkommazahlen (Real-Typen), Zeichen und Zeichenketten (String-Typen) sowie Wahrheitswerten (Boolesche Typen). Eine Auswahl der wichtigsten Typen:

Typ	Bedeutung	Beispiel(e)	
Integer-Typen (Ganzzahlen)			
integer	Speicherung von negativen und positiven ganzen Zahlen (inkl. Null). Wertebereich: -2147483648 bis 2147483647. Speicherbedarf: 4 Byte, mit Vorzeichen.	`var` `  x,y: Integer;`	Variablendeklaration.
		`x := -100001;` `y := 0;`	Wertzuweisungen im Programm.
byte	Speicherung von positiven ganzen Zahlen (inkl. Null). Wertebereich: 0 bis 255. Speicherbedarf: 1 Byte, ohne Vorzeichen.	`var` `  x,y: Byte;`	Variablendeklaration.
		`x := 42;` `y := 0;`	Wertzuweisungen im Programm.
Real-Typen (Gleit- bzw. Fließkommazahlen)			
real	Speicherung von negativen und positiven gebrochenen Zahlen (inkl. Null). Dabei ist der Wert einer ganzen Zahl wie z. B. Vier nicht 4, sondern 4,0. Wertebereich: ± 2,9 * 10^{-39} bis 1,7 * 10^{38}. Speicherbedarf: 6 Byte, mit Vorzeichen.	`var` `  x,y,z: Real;`	Variablendeklaration.

		`x := -3.88;`
		`y := 0.0;`
		`z := Pi;`
			Wertzuweisungen im Programm.

extended	Speicherung von negativen und positiven gebrochenen Zahlen (inkl. Null). Dabei ist der Wert einer ganzen Zahl wie z. B. Vier nicht 4, sondern 4,0. Wertebereich: ± 3,6 * 10^{-4951} bis 1,1 * 10^{4932}. Speicherbedarf: 10 Byte, mit Vorzeichen.	`var` `  x,y: Extended;`	Variablendeklaration.
		`x := -42.0;` `y := 1.000000000001;`	Wertzuweisungen im Programm.
String-Typen (Zeichen und Zeichenketten)			
char	Speicherung eines einzelnen Zeichens. Handelt es sich um eine Ziffer (z. B. Vier), wird sie als Zeichen und nicht als Zahl betrachtet (d. h. '4' und nicht 4). Wertebereich: Alle Zeichen des am PC eingestellten Gebietsschemas (z. B. ANSI-Zeichensatz), d. h. druckbare und Steuerzeichen.	`var` `  x,y,z: Char;`	Variablendeklaration.

		`x := 'm';`
		`y := '!';`
		`z := '3';`
		
		`[Hochkommas müssen hier gesetzt werden]`
			Wertzuweisungen im Programm.

string	Erweiterung zu char: Speicherung von einem oder mehr Zeichen (sog. Zeichenketten). Zugriff auf einzelne Zeichen mit eckigen Klammern („[“ und „]“), worin die Position des zu betrachtenden Zeichens angegeben wird. Wertebereich: Alle Zeichen des am PC eingestellten Gebietsschemas (z. B. ANSI-Zeichensatz), d. h. druckbare und Steuerzeichen.	`var` `  Str: string;`	Variablendeklaration.

		`Str := '';`
		
		`[„Leerstring“, d. h. „Nichts“.]`
			Wertzuweisung im Programm.

		`Str := 'Lehrer';`
		`Str[3] := 'e';`
		
		`[Str-Wert: 'Leerer']`	

Boolesche Typen (Wahrheitswerte)			
boolean	Dieser spezielle Typ dient zur Speicherung des Wahrheitsgehalts einer Aussage. Es können nur 2 Werte gespeichert werden: „False“ (Aussage ist falsch) und „True“ (wahr). False ist kleiner als True, d. h. man kann auch sagen: False entspricht dem Wert 0, True 1. Speicherbedarf: 1 Byte.	`var` `  x,y: Boolean;`	Variablendeklaration.

		`x := True;`
		`if x then`
		`  Windows.Beep;`
		
		`[Kurzform für`
		`if x = True then`
		`  Windows.Beep;`
		`Beep-Ton wird ausgegeben.]`
			Wertzuweisung und Abfrage im Programm.

Neben Variablen existieren in Delphi *Konstanten*. Diese kann man sich als unveränderbare Variablen vorstellen. Ihr Einsatz bietet sich dann an, wenn im Programm ein und derselbe konstante Wert mehrfach vorkommt (z. B. „5“, für 5 zu erzeugende Zufallszahlen) und man an den jeweiligen Stellen im Quellcode anstelle des konkreten Wertes („5“) eine zuvor deklarierte Konstante schreibt („const Anzahl = 5“ in der Deklaration und „Anzahl“ anstelle „5“ an den Codestellen). Möchte man zu einem späteren Zeitpunkt der Programmentwicklung diesen konstanten Wert aus irgendeinem Grund ändern (bspw. von „5“ auf „10“ Zufallszahlen), so ist nur *eine* Änderung an nur *einer* Stelle – nämlich bei der Deklaration der Konstanten – nötig. Dadurch entfällt ein unnötiges Suchen und Anpassen der einzelnen konstanten Werte im Quellcode, was zeiteffizienter, fehlerunanfälliger und eleganter ist.

Als weiterer Unterschied zu Variablen muss bei der Deklaration kein Variablen- (besser: Konstanten-)Typ angegeben werden; Delphi erkennt diesen anhand des Konstantenwerts automatisch. Außerdem wird in der Deklaration ein „=“ anstelle eines „:“ verwendet.

Beispiel	**Bedeutung**
`const` `  Max = 100;`	Eine Konstante „Max“ mit dem Wert 100 wird deklariert. Aufgrund des Wertes erkennt Delphi automatisch, dass es sich bei „Max“ um einen Integertyp handelt, so dass dies nicht mehr explizit angegeben werden muss. Der Wert von „Max“ ist im weiteren Programmcodeverlauf *nicht* änderbar.
`const` `  Message = 'Out of memory';`	Eine (Fehler-)Meldung in der Konstanten „Message“ wird deklariert. Als Konstantentyp wird von Delphi automatisch String angenommen.

`const`
`  Str = 'Test';`
`  Pos = 5 - Length(Str) div 2;`
	Auch Terme zur Berechnung von Konstanten sind erlaubt. Hier ist eine Konstante „Str“ (eine Variable „Str“ wäre hier nicht möglich!) vor „Pos“ deklariert, da ihr Wert zur Berechnung von „Pos“ bekannt sein muss (zur Erinnerung: der Compiler arbeitet den Quellcode von oben nach unten ab).

4) Operatoren

Mit Hilfe von Operatoren kann man Vergleiche anstellen oder Mengen von Zeichen und Ziffern beschreiben. Delphi kennt verschiedene Arten von Operatoren, darunter arithmetische, logische/boolesche und Vergleichs-/Mengenoperatoren.
Sämtliche in Delphi auftretenden Operatoren werden bei der Auswertung eines Ausdrucks in der folgenden Reihenfolge bearbeitet:

1. *(...)* (Klammern)
2. *not* , + , - (+ und - hier als Vorzeichen!)
3. * , / , *div* , *mod* , *and*
4. + , - , *or* , *xor*
5. < , <= , = ,<> , >= , > (Vergleichsoperatoren)

Beispiele:

(R < S) or P
A and B or (C and not D)
not (A and ((R < S) or (I = 10))
((R + 0.7) > S * T) and P or (I = J + 10)

Operator	Operation	Typ	Beispiel(e)	
(und *)*	Klammer	-	`x := (5 + 3) * 2;` `[x = 16]`	
Arithmetische Operatoren				

+	Addition	Real, Integer, String	`x := 4;`
			`y := x + 2;`
			`[y = 6]`
				`Str1 := 'Star';`
				`Str2 := Str1 + ' Trek';`
				`[Str2 = 'Star Trek']`

-	Subtraktion	Real, Integer	`x := 4;`
			`y := x - 2;`
			`[y = 2]`
				`z := 4.2 - 5;`
				`[z = -0.8]`

*	Multiplikation	Real, Integer	`x := 4;`
			`y := x * 2;`
			`[y = 8]`
				`z := 4.2 * 5;`
				`[z = 21.0]`

/	Gleitkommadivision	Real	`x := 4.2;`
			`y := x / 2;`
			`[y = 2.1]`	

div	Ganzzahlige Division (ein evtl. vorhandener Rest wird verworfen)	Integer	`x := 4;`
			`y := x div 2;`
			`[y = 2]`
				`x := 5;`
				`y := x div 2;`
				`[y = 2]`

mod	Rest der ganzzahligen Division	Integer	`x := 4;`
			`y := x mod 2;`
			`[y = 0]`
			`Man kann so testen, ob eine Zahl gerade (Rest ist 0) oder ungerade (Rest 1) ist!`
				`x := 5;`
				`y := x mod 2;`
				`[y = 1]`

Logische/boolesche Operatoren				

not	Negation (die Aussage wird umgekehrt)	Boolean	`x := True;` `[x = True]`
			`x := not(x);` `[x = False]`
			`y := not(True);` `[y = False]`	

and	Konjunktion (wahr, wenn alle Teile wahr sind)	Boolean	`x := True; y := True;`
			`if x and y then Windows.Beep;`
			`[x und y sind beide wahr, also Beep-Ton.]`	

or	Inklusive Disjunktion	Boolean	`x := True; y := True;`	

	(wahr, wenn *min.* ein Teil wahr ist)		`if x or y then Windows.Beep;`
			
			`[x ist wahr (y obwohl nicht mehr nötig auch), also Beep-Ton.]`

xor	Exklusive Disjunktion (wahr, wenn *genau* ein Teil wahr ist)	Boolean	`x := True;  y := True;`
			`if x xor y then Windows.Beep;`
			
			`[x ist zwar wahr, aber y auch (bei xor darf aber nur ein Teil wahr sein!), also` *kein* `Beep-Ton.]`

Vergleichs-/Mengenoperatoren			

=	Gleich
	(nicht verwechseln mit der Zuweisung „:=“)
		Boolean	`x := 42;`
			`if x = 42 then Windows.Beep;`
			
			`[Ausdruck stimmt (da x = 42), also Beep-Ton.]`

<>	Ungleich	Boolean	`x := 42;`
			`if x <> 42 then Windows.Beep;`
			
			`[Ausdruck stimmt nicht (da x = 42), also` *kein* `Beep-Ton.]`

<	Kleiner als	Boolean	`x := 42;`
			`if x < 42 then Windows.Beep;`
			
			`[Ausdruck stimmt nicht (da x = 42), also` *kein* `Beep-Ton.]`

>	Größer als	Boolean	`x := 42;`
			`if x > 42 then Windows.Beep;`
			
			`[Ausdruck stimmt nicht (da x = 42), also` *kein* `Beep-Ton.]`

<=	Kleiner oder gleich	Boolean	`x := 42;`
			`if x <= 42 then Windows.Beep;`
			
			`[Ausdruck stimmt (da x = 42), also Beep-Ton.]`

>=	Größer oder gleich	Boolean	`x := 42;`
			`if x >= 42 then Windows.Beep;`
			
			`[Ausdruck stimmt (da x = 42), also Beep-Ton.]`

in	Element von / „in“ der folgenden (z. B. Zahlen-) Menge	Boolean	`x := 42;`
			`if x in [1,3,5,10..13] then Windows.Beep;`
			
			`[Ausdruck stimmt nicht (da x = 42), also` *kein* `Beep-Ton.]`
			`[Hinweis: 10..13 bedeutet: 10,11,12,13.]`

5) Arithmetische Routinen

Mit den folgenden Funktionen lassen sich verschiedenste Integer- und Fließkommaberechnungen sowie andere mathematische Operationen durchführen.

Funktion	Bedeutung	Beispiel(e)
Abs	Absolutwert des Integer- oder Real- Arguments (Betragsfunktion).	`x := Abs(-8.2);` `[x = 8.2]`
Cos	Kosinus des Real-Arguments (Winkel); dieser ist im Bogenmaß (Rad) anzugeben.	`x := Cos(0.0*Pi);` `[x = 1.0]`
Exp	Exponentialfunktion e hoch x, wobei e die Basis des natürlichen Logarithmus ist.	`x := Exp(1);` `[x = 2,71828182845905]`
Frac	Nachkommaanteil des Real-Arguments.	`x := Frac(5.8);` `[x = 0.8]`
Int	Ganzzahliger Anteil des Real-Arguments, Ergebnis wiederum als Real-Wert.	`x := Int(5.8);` `[x = 5.0]`

IntPower	Gibt das Ergebnis einer Potenz (Basis hoch Exponent) als Kommazahl zurück. [Hinweis: Routine befindet sich in der Unit Math, d. h. „Math“ unter „uses“ angeben.]	`x := IntPower(2,4);`
		`[x = 16.0]`
		`[Entspricht der Potenz 2 hoch 4.]`

Ln	Natürlicher Logarithmus des Real-Arguments, Ergebnis wiederum als Real-Wert.	`x := Ln(2.71828182845905);` `[x = 1.0]`
Pi	Wert der Konstanten Pi.	`x := Pi + 2;`

Random (Bereich) *	Liefert eine (Pseudo-)Zufallszahl innerhalb 0 <= x < *Bereich* zurück. Wird der Parameter *Bereich* nicht angegeben, liefert Random einen Real-Wert innerhalb 0 <= x < 1 zurück.
	Beachte: Random-Zufallszahlen sind reproduzierbar und nicht wirklich zufällig (siehe dazu „Randomize“).
		`Zufallszahl := Random(6);`
		`[Zufallszahl wird einen Integer-Wert zw. 0 und 5 (insges. 6 mögliche Zahlen) erhalten.]`
		`Zufallszahl := Random;`
		`[Zufallszahl wird einen Real-Wert zw. 0 und 1 (ohne 1) erhalten, i. d. R. mit einer Genauigkeit von 15 Dezimalen.]`

Randomize *	Startet den Zufallszahlengenerator. Damit werden Pseudo-Zufallszahlen erzeugt, die sich *nicht* bei jedem Prorammaufruf (gleich) wiederholen. „Randomize“ muss nur einmal gestartet werden und gilt dann während des gesamten Programmablaufs.	`Randomize;`
		`Zufallszahl := Random(6) + 1;`
		`[Erzeugt eine „echtere“ (Pseudo-) Zufallszahl zw. 1 und 6 (Würfelwurf), die nicht bei jedem Programmaufruf die gleiche ist.]`

Round	Das Real-Argument wird auf einen ganzzahligen Wert gerundet.	`x := Round(5.78);` `[x = 6.0]`
Sin	Sinus des Real-Arguments (Winkel); dieser ist im Bogenmaß (Rad) anzugeben.	`x := Sin(0.5*Pi);` `[x = 1.0]`
Sqr	Quadrat des Arguments.	`x := 5 * 5;  ist das gleiche wie` `y := Sqr(5);`
Sqrt	Quadratwurzel des Real-Arguments.	`x := Sqrt(Sqr(5));` `[x = 5, da sich Sqr und Sqrt aufheben.]`

Tan	Tangens des Real-Arguments (Winkel); dieser ist im Bogenmaß (Rad) anzugeben.	`x := Tan(1);`
		
		`[x = 1.5574077246549]`

Trunc	Wandelt einen Real-Wert in einen Integer-Wert um, indem die Nachkommastellen verworfen werden.	`x := Trunc(5.8);`
		
		`[x = 5]`

* Das Erzeugen von „echtem“ Zufall, dessen Zustandekommen und Ergebnis nicht gezielt wiederholbar ist (also keine „Pseudozufallszahlen“), gestaltet sich nicht zuletzt programmiertechnisch als kompliziert. Für Näheres fragen Sie das Internet, ein Informatik-Lehrbuch oder Ihren Lehrer.

6) Behandlung von Ordinalwerten [= aufzählbare Variablentypen]

Eine Reihe von Routinen hilft uns beim Umgang mit Ordinalwerten – also mit aufzählbaren Typen wie Integer, Boolean, Char oder anderen Aufzählungstypen (Real also nicht!).

Funktion	Bedeutung	Beispiel(e)	

Dec	Vermindert (dekrementiert) eine Variable um einen bestimmten Wert. Ist nur ein Argument in Klammern angegeben, wird um 1 vermindert.	`x := 42;`
		`Dec(x);`
		
		`[x = 41]`
			`x := 42;`
			`Dec(x,4);`
			
			`[x = 38]`

Inc	Erhöht (inkrementiert) eine Variable um einen bestimmten Wert. Ist nur ein Argument in Klammern angegeben, wird um 1 erhöht.	`x := 13;`
		`Inc(x);`
		
		`[x = 14]`
			`x := 13;`
			`Inc(x,3);`
			
			`[x = 16]`

Odd	Prüft, ob das Argument eine ungerade Zahl ist.	`if Odd(5) then` `  ShowMessage('Zahl ungerade!');`	

Pred	Liefert den Vorgänger des Arguments.	`x := Pred(42);`
		
		`[x = 41]`
			`x := Pred('B');`
			
			`[x = 'A']`

Succ	Liefert den Nachfolger des Arguments.	`x := Succ(42);`
		
		`[x = 43]`
			`x := Succ('B');`
			
			`[x = 'C']`

Chr	Wandelt eine Ordinalzahl gemäß ASCII-Tabelle in das dazugehörige Zeichen um (Gegenteil zu Ord).	`Str := Chr(65);`
		
		`[Str = 'A']`	

High	Ergibt den höchsten Wert im Bereich des Arguments (siehe Delphi-Hilfe).	–	
Low	Ergibt den niedrigsten Wert im Bereich des Arguments (siehe Delphi-Hilfe).	–	

Ord	Wandelt einen ordinalen Typ gemäß ASCII-Tabelle in die dazugehörige Ordinalzahl um (Gegenteil zu Chr).	`x := Ord('A');`
		
		`[x = 65]`	

7) Routinen für die String-Bearbeitung

Delphi bietet eine umfangreiche Auswahl von Routinen zur Stringverwaltung, zumeist um eine Zeichenkette zu formatieren. Die dabei gängigsten Routinen sind:

Funktion	Bedeutung	Beispiel(e)

Concat	Verbindet mehrere String durch Hintereinanderreihung.	`Str := 'Star Trek';`
		`EdtAusgabe.Text := Concat(Str,' TNG');`
		
		`[EdtAusgabe.Text = 'Star Trek TNG']`

Copy	Kopiert einen Teil aus einem String (ab Position x werden y Zeichen kopiert).	`Str := 'Star Trek';`
		`EdtAusgabe.Text := Copy(Str,6,4);`
		
		`[EdtAusgabe.Text = 'Trek']`

Delete	Löscht einen Teil innerhalb eines Strings (ab Position x werden y Zeichen gelöscht).	`Str := 'Informatik';`
		`Delete(Str,5,6);`
		
		`[Str = 'Info']`

Insert	Fügt einen String in einen anderen ein.	`Str := 'Star Trek';`
		`Insert('let',Str,5);`
		
		`[Str = 'Starlet Trek']`

Length	Ermittelt die Anzahl der Zeichen in einem String.	`Str := 'Star Trek';`
		`x := Length(Str);`
		
		`[x = 9]`

Pos	Sucht einen String in einem anderen und gibt die Position des *ersten* Zeichens der Fundstelle zurück.	`Str := 'Star Trek';`
		`x := Pos('r',Str);`
		
		`[x = 4]`

LowerCase	Konvertiert alle Zeichen in einem String in Kleinbuchstaben (Ziffern und Sonderzeichen bleiben unverändert).	`Str := 'Star Trek 10!';`
		`EdtAusgabe.Text := LowerCase(Str);`
		
		`[EdtAusgabe.Text = 'star trek 10!']`

UpperCase	Konvertiert alle Zeichen in einem String in Großbuchstaben (Ziffern und Sonderzeichen bleiben unverändert).	`Str := 'Star Trek 10!';`
		`EdtAusgabe.Text := UpperCase(Str);`
		
		`[EdtAusgabe.Text = 'STAR TREK 10!']`

Trim	Entfernt Leerzeichen in einem String, die am Anfang oder am Ende stehen.	`Str := '  Star Trek ';`
		`EdtAusgabe.Text := Trim(Str);`
		
		`[EdtAusgabe.Text = 'Star Trek']`

TrimLeft	Entfernt Leerzeichen am Anfang des String.	`Str := '  Star Trek ';`
		`EdtAusgabe.Text := TrimLeft(Str);`
		
		`[EdtAusgabe.Text = 'Star Trek ']`

TrimRight	Entfernt Leerzeichen am Ende des String.	`Str := '  Star Trek ';`
		`EdtAusgabe.Text := TrimRight(Str);`
		
		`[EdtAusgabe.Text = '  Star Trek']`

8) Typumwandlung (Zahl-Zeichen- und Zeichen-Zahl-Konvertierungen)

Nicht zuletzt zur Kommunikation mit den Delphi-Komponenten existieren eine Reihe von Umwandlungsfunktionen für Zeichen- und Zahlentypen.

Funktion	Bedeutung	Beispiel(e)
FloatToStr	Wandelt einen Real-Wert in eine Zeichenfolge (String) um.	`Zahl := Pi;` `EdtAusgabe.Text := FloatToStr(Zahl);`

FloatToStrF	Wandelt einen Real-Wert in eine vordefinierte Zeichenfolge (String) um. Das abschließende „F“ steht für „Format“/„formatiren“. Der Aufbau ist folgender:
	
	`FloatToStrF(UmzuwandelnderRealWert,`
	`ffFixed, Rundungsgenauigkeit,`
	`AnzahlNachkommastellen)`
	
	(Für eine genaue Aufschlüsselung der notwendigen Parameter (insbes. „ffFixed“, wozu es noch Alternativen gibt) siehe Delphi-Hilfe.)
		`EdtAusgabe.Text :=`
		`FloatToStrF(Pi,ffFixed,8,2);`
		
		`[EdtAusgabe.Text = '3,14'.]`
		
		`[Der umzuwandelnde Real-Wert „Pi“ (erster Parameter) ist in Delphi vordefiniert. Der letzte Parameter „2“ gibt die Anzahl der Nachkommastellen an, entsprechend wird „Pi“ mit dem Nachkommaanteil „14“ ausgegeben.]`

FormatFloat	Formatiert den übergebenen Real-Wert analog zum ebenfalls übergebenen Formatierungsstring (siehe Delphi-Hilfe).	`...`

IntToHex	Wandelt einen Integer-Wert in einen String um, wobei das Zahlenformat im Hexadezimalsystem benutzt wird.	`Zahl := 42;`
		`EdtAusgabe.Text := IntToHex(Zahl,4);`
		
		`[EdtAusgabe.Text = '002A'.]`
		
		`[Der zweite Parameter (oben: 4) gibt an, aus wie vielen Stellen der HexString max. bestehen soll.]`

IntToStr	Wandelt einen Integer-Wert in eine Zeichenfolge (String) um.	`Zahl := 5;` `EdtAusgabe.Text := IntToStr(Zahl);`
StrToFloat	Wandelt eine Zeichenfolge in einen Real-Wert um.	`EdtEingabe.Text := '3.141';` `Zahl := StrToFloat(EdtEingabe.Text);`
StrToInt	Wandelt eine Zeichenfolge in einen Integer-Wert um.	`EdtEingabe.Text := '42';` `Zahl := StrToInt(EdtEingabe.Text);`

9) Kontrollstrukturen

Kontrollstrukturen sind in der Delphi-Programmierung fundamentale Konstrukte von Algorithmen, d. h. grundlegende Bausteine zur Beschreibung von Problemlösungen. Von besonderer Bedeutung sind dabei Verzweigungen und Schleifen.

Struktur	Bedeutung	Beispiel(e)
Verzweigungen (dienen je nach Situation zur alternativen Auswahl verschiedener Anweisungen)		

if...then...	Einseitige Verzweigung:
	
	if <Bedingung(en) wahr> then
	<Anweisung(en) ausführen>;
	
	Wenn <Bedingung(en) erfüllt> dann
	<Anweisung(en) ausführen>;
		`x := 2;`
		`if x = 2 then Windows.Beep;`
		
		`[Ausdruck stimmt (da x = 2), also Beep-Ton.]`
		
		`x := 2; y := 4;`
		`if (x = 2) and (y < 4) then Windows.Beep;`
		
		`[Ausdruck ist nicht wahr (zwar ist x = 2, aber y nicht kleiner 4), also kein Beep-Ton.]`

if...then... else...	Zweiseitige Verzweigung:
	
	if <Bedingung(en) wahr> then
	<Anweisung(en) ausführen>
	else <AlternativAnweisung(en) ausführen>;
	
	Wenn <Bedingung(en) erfüllt> dann
	<Anweisung(en) ausführen>
	sonst <AlternativAnweisung(en) ausführen>;
		`x := 2;`
		`if x = 1 then Windows.Beep`
		`  else ShowMessage('x ist nicht 1!');`
		
		`[Ausdruck stimmt nicht (da x = 2 und nicht 1), also kein Beep-Ton, sondern alternativ Ausgabe obiger Nachricht.]`
		
		`[Hinweis: Vor einem else darf kein Semikolon stehen! Daher folgt auch auf den Befehl „Windows.Beep" ausnahmsweise keines.]`

case...of... [*else...*]	Mehrseitige Verzweigung (Fallunterscheidung):
	
	case <Variable> of
	<Variablenwert>: <Anweisung(en) ausf.>;
	[else <AlternativAnweisung(en) ausführen>];
	end;
	
	Im Fall, dass <Variable> hat den Wert...
	<Variablenwert>: <Anweisung(en) ausf.>;
	[sonst <AlternativAnweisung(en) ausführen>];
	end;
		`x := 2;`
		`case x of`
		`  1: ShowMessage('x ist 1!');`
		`  2: ShowMessage('x ist 2!');`
		`  3..5: ShowMessage('x ist 3, 4 od. 5!');`
		`  [else Windows.Beep;]`
		`end;`
		
		`[Fall 2 tritt ein (da x = 2), d. h. Ausgabe von 'x ist 2!'.`
		`Man kann wie bei 3..5 auch einen Wertebereich angeben.`
		`Der else-Teil ist optional, d. h. er muss nicht vorhanden sein.]`

Schleifen (dienen zum automatischen Wiederholen von Anweisungen)		

for...to...do...	For-Schleife:
	
	for <Variable> := <Startwert> to/downto <Endwert> do <Anweisung(en) ausführen>
	
	Zähle <Variable> von <Startwert> bis <Endwert> und wiederhole <Anweisung(en) ausführen>;
		`for i := 1 to 5 do`
		`  Windows.Beep;`
		
		`[Beep-Ton wird 5x (von 1 bis 5) ausgegeben.]`
		
		`x := 5;`
		`for i := 10 downto x do`
		`  Windows.Beep;`
		
		`[Beep-Ton wird 6x (von 10 bis 5) ausgegeben.]`

while...do...	While-Schleife:
	
	while <Bedingung(en) wahr>, wiederhole
		`x := 1;`
		`while x < 5 do`
		`  x := x + 1;`

	<Anweisungen(en) ausführen>; Solange <Bedingung(en) wahr>, wiederhole <Anweisungen(en) ausführen>;	`[Die Schleife tut nichts anderes, als x von 1 bis 5 hochzuzählen.]`

repeat...until..	Repeat-Schleife:
	
	repeat <Anweisungen(en) ausführen> until <Bedingung(en) wahr>;
	
	Wiederhole <Anweisungen(en) ausführen> bis <Bedingung(en) wahr>;
		`Abbruch := False;`
		`repeat`
		`  Zufallszahl := Random(6) + 1;`
		`  `**`if`**` Zufallszahl=1 `**`then`**` Abbruch := True;`
		`until`` Abbruch;`
		
		`[So lange Würfeln (Zufallszahlen zw. 1 und 6), bis eine 1 erscheint.]`
		
		`[Hinweis: Man könnte auch unschöner (!) „`**`until`**` Abbruch = True;“ schreiben.]`

10) Ablaufsteuerung

Mit den Methoden zur Ablaufsteuerung lassen sich u. a. Schleifendurchläufe manipulieren und die Programmbeendigung kontrollieren.

Funktion	Bedeutung	Beispiel(e)

Close	Schließt ein Formular.
	Wenn das Hauptformular der Anwendung geschlossen wird, wird das komplette Programm beendet.
	(Versuche, ein Formular zu schließen, können mit der Ereignisbehandlungsroutine für OnCloseQuery abgebrochen werden.)
		`procedure TForm1.BtnCloseClick(Sender: TObject);`
		`begin`
		`  Close;`
		`end;`

Break	Bewirkt, dass eine For-, While- oder Repeat-Schleife verlassen und die Ausführung mit der nächsten Anweisung fortgesetzt wird.
	Die Verwendung von Break sollte durch eine geschicktere Schleifen-Abbruchbedingung vermieden werden.
	Wird Break außerhalb eines For-, While- oder Repeat-Konstrukts aufgerufen, gibt der Compiler eine Fehlermeldung aus.
		`for i := 1 to 10 do begin`
		`  ShowMessage('Hallo!');`
		`  if i = 5 then Break;`
		`end;`

Continue	Bewirkt, dass der nächste Durchlauf der For-, While- oder Repeat-Schleife ausgeführt wird.
	Wird Continue außerhalb eines For-, While- oder Repeat-Konstrukts aufgerufen, gibt der Compiler eine Fehlermeldung aus.
		`i := 0;`
		
		`repeat`
		`  Inc(i);`
		`  if i mod 2 = 0 then Continue`
		`  else ShowMessage(IntToStr(i));`
		`until i = 100;`

Exit	Entzieht der aktuellen Prozedur sofort die Programmsteuerung, sprich: beendet radikal die Prozedur.
	Ist die aktuelle Prozedur das Hauptprogramm, wird die Anwendung beendet.
	Nach dem Aufruf von Exit wird die aufrufende Routine mit der nach dem Prozeduraufruf folgenden Anweisung fortgesetzt.
	Nicht zu verwenden, um eine Prozedur regulär zu beenden.
		`procedure Test(i: Integer);`
		`begin`
		`  if i = 1 then Exit`
		`  else begin`
		`  ...`
		`  end;`
		`end;`

Halt	Löst einen Programmabbruch aus und gibt die Steuerung an Windows zurück. Nicht zu verwenden, um ein Programm regulär zu beenden.	`Halt;`
Terminate	Beendet die Anwendung programmgesteuert (regulär/normal). Durch einen Aufruf der Methode Terminate wird das Anwendungsobjekt nicht einfach gelöscht, die Anwendung kann vielmehr ordnungsgemäß heruntergefahren werden. Terminate muss nicht direkt aufgerufen werden; es wird automatisch beim Schließen des Hauptformulars aufgerufen.	`Application.Terminate;`

11) Dialoge

Delphi stellt einige vordefinierte Dialoge zur Verfügung, um eine Meldung anzuzeigen oder vom Anwender einen Wert zu erfragen.

Funktion	Bedeutung	Beispiel(e)
InputBox	Stellt eine Dialogbox dar, in der der Benutzer etwas eingeben kann.	`InputBox('Überschrift','Bitte Text eingeben:','Default-Eingabe');` `Str := InputBox('Überschrift','Bitte Text eingeben:','Default-Eingabe');`

InputQuery	Wie InputBox, aber die Funktion liefert True oder False zurück, je nachdem, ob der Benutzer den Schalter „OK“ oder „Abbruch“ angeklickt hat.	`var`
		`  Button: Boolean;`
		`  Eingabe: string;`
		`[...]`
		`Button := InputQuery('Überschrift', 'Bitte Text eingeben:',Eingabe);`
		`if Button then ShowMessage('OK-Klick') else ShowMessage('Abbruch-Klick');`

MessageDlg	Zeigt ein Meldungsfenster an, wobei der Text, ein Hinweissymbol und die Buttons innerhalb des Fensters festgelegt werden können (siehe Delphi-Hilfe).	`MessageDlg('Caesar-Verschlüsselung (V1.0)' + #13 + 'M. Zirbes, 2008', mtInformation,[mbOK],0);`
MessageDlgPos	Wie MessageDlg, aber zusätzlich kann noch die Position des Meldungsfensters angegeben werden (siehe Delphi-Hilfe).	–
ShowMessage	Zeigt ein einfaches Meldungsfenster mit einer Meldung (Text) an.	`ShowMessage('Hallo Welt!');`
ShowMessagePos	Wie ShowMessage, aber zusätzlich kann die Position des Meldungsfensters angegeben werden (x- und y-Koordinate).	`ShowMessage('Hallo Welt!',50,50);`

12) Routinen für Datum und Uhrzeit

Delphi bietet etliche Routinen, mit denen man bequem mit Datum und Uhrzeit arbeiten kann. Eine Auswahl im einzelnen:

Funktion	Bedeutung	Beispiel(e)
Date	Liefert das aktuelle Datum im TDateTime-Format zurück.	-

DateTimeToStr	Wandelt das TDateTime-Format in einen String um.	`Str := DateTimeToStr(Now);`
		`EdtAusgabe.Text := Str;`
		
		`[Str = <AktuellesDatum> <AktuelleZeit>,`
		`z. B. Str = 17.02.2008 20:02:59]`

DateToStr	Wandelt einen Datumsformat-Wert in einen String um.	`Str := DateToStr(Date);`
		`EdtAusgabe.Text := Str;`
		
		`[Str = <AktuellesDatum>, z. B.`
		`Str = 17.02.2008]`

DayOfWeek	Gibt den Wochentag des angegebenen Datums als Integer zwischen 1 und 7 zurück. Dabei ist Sonntag der erste Tag der Woche und Samstag der siebte.	`TagWert := DayOfWeek(Date);`
		`EdtAusgabe.Text := IntToStr(TagWert);`
		
		`[Für einen Sonntag:`
		`Str = 1]`

DecodeDate	Dekodiert das angegebene Datum in die Werte Jahr, Monat und Tag.	`var`
		`  Jahr,Monat,Tag: Word;  // Ein Int-Typ`
		
		`[...]`
		
		`DecodeDate(Date,Jahr,Monat,Tag);`
		`EdtAusgabe.Text := IntToStr(Jahr);`
		
		`[EdtAusgabe = <AktuellesJahr>, z. B.`
		`EdtAusgabe.Text = 2008]`

DecodeTime	Dekodiert die angegebene Uhrzeit in Stunden, Minuten, Sekunden und Millisekunden.	`var`
		`  Std,Min,Sek,MSek: Word;  // Int-Typ`
		
		`[...]`
		
		`DecodeTime(Time,Std,Min,Sek,MSek);`
		`EdtAusgabe.Text := IntToStr(Std);`
		
		`[EdtAusgabe = <AktuelleStunde>, z. B.`
		`EdtAusgabe.Text = 14]`

GetTickCount	Liefert die Anzahl der vergangen Millisekunden seit dem Windows-Start.	`Zeitstempel1 := GetTickCount;`
		`Zeitstempel2 := GetTickCount;`
		`Differenz := Zeitstempel1-Zeitstempel1;`

Now	Liefert das aktuelle Datum und Uhrzeit (Kombination aus Date und Time).	-
Time	Liefert die aktuelle Uhrzeit zurück.	-

TimeToStr	Wandelt das Zeitformat in eine Zeichenkette (String) um.	`Str := TimeToStr(Time);`
		`EdtAusgabe.Text := Str;`
		
		`[Str = <AktuelleZeit>, z. B.`
		`Str = 20:02:59]`

13) Sonstige nützliche Routinen

Die Systembibliotheken bieten noch einige nützliche Routinen, die sich nicht ohne weiteres in die anderen Kategorien einordnen lassen.

Funktion	Bedeutung	Beispiel(e)
Exclude	Ausschließen eines Elements aus einer Menge.	-
ParamCount	Liefert die Anzahl der Kommandozeilen-Parameter.	-
ParamStr	Liefert einen Kommandozeilen-Parameter.	-

Windows.Beep	Gibt einen Warnton über den PC-Speaker (in jedem PC fest eingebauter rudimentärer Lautsprecher) aus. Der erste Parameter gibt die Tonfrequenz an, der zweite die Dauer in Millisekunden.	`Windows.Beep(100,1000);`
		
		`[Näheres siehe Delphi-Hilfe.]`

14) Komponentenspezifische Anweisungen

Im Folgenden findet sich eine Reihe exemplarischer/typischer Anweisungen zu bestimmten Komponenten.

Komponente/Anweisung	Bedeutung
Hauptformular	
Form1.ActiveControl := EdtEingabe;	Mit der Eigenschaft ActiveControl eines (Haupt-)Formulars kann man festlegen, welches Steuerelement im Formular den (Eingabe-)Fokus hat. In einer Anwendung kann immer nur ein Steuerelement aktiv sein. Im Beispiel wird der Fokus auf das Edit „EdtEingabe“ gesetzt.
procedure *TForm1.FormCreate (Sender: TObject);*	Diese Ereignismethode wird durch einen Doppelklick auf eine freie Stelle im Formular erzeugt (d. h. wo sich keine Komponente befindet). Delphi führt sie automatisch zur Laufzeit aus, wenn das Formular erzeugt wird („FormCreate“), also gleich zu Beginn des Programms. So ist es möglich, beim Programmstart Anweisungen ausführen zu lassen, *ohne* dass der Benutzer etwas tun (z. B. einen Button anklicken) muss.
„Edit“-Komponente	
procedure *TForm1.EdtEingabeKeyPress (Sender: TObject;* ***var*** *Key: Char);* ***begin*** ***if*** *Key = #8* ***then*** *EdtEingabe.SelectAll;* ***end****;*	Diese Prozedur wird aufgerufen, wenn man im Edit „EdtEingabe“ eine beliebige Taste drückt. Sollte es sich dabei um Backspace handeln (entspricht #8), kann es wünschenswert sein, dass nicht wie gewohnt ein einzelnes Zeichen gelöscht, sondern zunächst der gesamte Edit-Inhalt markiert wird. Dies erreicht man mit Hilfe der Edit-Eigenschaft „SelectAll“.
procedure *TForm1.EdtEingabeKeyPress (Sender: TObject;* ***var*** *Key: Char);* ***begin*** ***if*** *Key = #13* ***then*** *BtnOKClick(Sender);* ***end****;*	Diese Prozedur wird aufgerufen, wenn man im Edit „EdtEingabe“ eine beliebige Taste drückt. Dabei kann es wünschenswert sein, dass wenn der User die Enter-Taste (entspricht #13) drückt, dies wie ein Maus-Klick auf einen bestimmten (Bestätigungs-/Berechnungs-/OK-)Button interpretiert wird. Dies erreicht man durch den Aufruf der entsprechenden Button-Komponente im Quellcode. Im Beispiel wird mit „BtnOKClick(Sender)“ ein Button „BtnClick“ ausgelöst.
„Image“-Komponente	
ImgWuerfel.Picture.LoadFromFile ('Bilder\Bild1.bmp');	Lädt die Bilddatei „Bild1.bmp“ in die Imagekomponente „ImgWuerfel“. Die Datei befindet sich dabei in einem – vom aktuellen Projektordner des Programms aus betrachtet – Unterverzeichnis „Bilder“ (Angabe hier also über einen relativen Dateipfad; ein absoluter Dateipfad wäre natürlich auch möglich).
„Memo“-Komponente (analog dazu: **„RichEdit“-Komponente**)	
MemAusgabe.Lines.Add('Eine neue Zeile einfügen');	Fügt in das Memo „MemAusgabe“ eine neue Zeile mit dem String „Eine neue Zeile“ hinzu.
„StringGrid“-Komponente	
StrGrdTabelle.Cells[Spalte,Zeile] := 'Zelleninhalt';	Schreibt in ein StringGrid „StrGrdTabelle“ in die Zelle mit den (als Variab-len angegebenen) Koordinaten „[Spalte,Zeile]“ (also bspw. „[2,4]“) den String-Wert „Zelleninhalt“ rein.