

KAPITEL 3

EINFACHE DATENTYPEN

3.1. Konstanten und Variablen

Auch wenn ein Programm üblicherweise nichts mit Mathematik zu tun hat, geht es in Component Pascal (zum Trost: auch in jeder anderen Programmiersprache) nicht ganz ohne sie, denn die internen Maschinenoperationen beruhen auf mathematischer Logik; aber es handelt sich bei dem Folgenden, soviel sei versprochen, um einfache Grundrechnungsarten. Wichtig ist allerdings, daß Sie die Einzelheiten dieses Kapitels wirklich durcharbeiten und verstehen; alles Spätere wird sich auf die eine oder andere der hier erarbeiteten Grundlagen stützen und davon Gebrauch machen.

Laden Sie das Programm `ConstVar.odc`. Es ist das erste Programm, das Aussagen zwischen der Kopfzeile der Prozedur *Start* und dem Schlüsselwort *BEGIN* enthält. Dieser Teil der Prozedur wird Deklarationsteil genannt, der dem *BEGIN* folgende Teil heißt Anweisungsteil, in ihm stehen alle Handlungsanweisungen für den Compiler, während im Deklarationsteil all das steht, was späterhin im Anweisungsteil als "Zutat" des "Kuchens" erscheinen soll, denn eine Grundregel in Component Pascal besagt, daß alles vorab deklariert, also dem Compiler bekannt gemacht werden muß, was später verwendet werden soll.

Als erstes sehen Sie im Deklarationsteil ein neues Component Pascal Schlüsselwort: *CONST*. Damit wird dem Compiler erklärt, daß die nachfolgenden Bezeichner Konstanten sind, Ausdrücke also, deren Werte für den Gültigkeitsbereich der Prozedur dauerhaft festgelegt sind. Die Zeile: *Titel = "Konstanten und Variablen"* identifiziert für den Compiler den Bezeichner *Titel* mit dem Inhalt *Konstanten und Variablen*. Ebenso identifiziert die folgende Zeile: *z = -2* den Bezeichner *z* mit der Zahl *-2*.

Im Gegensatz zum Schlüsselwort *CONST* steht das anschließende Schlüsselwort *VAR*. Damit wird dem Compiler erklärt, daß die nachfolgenden Bezeichner *x*, *y* und *Summe* Variablen sind, Größen also, deren Werte im Laufe des Programms geändert werden können. Während jedoch die Werte von Konstanten durch die oben beschriebene Identifikation unmittelbar bei der Deklaration festgelegt werden, sind die Anfangswerte von Variablen nicht bereits bei der Deklaration bestimmt, sondern sie werden den Variablen erst im Anweisungsteil der Prozedur zugewiesen (und eventuell ein- oder mehrmals verändert).

Darüber hinaus ist ein Weiteres bei der Deklaration von Variablen wichtig. Der Computer hat für Daten verschiedenen Typs sehr unterschiedliche Erkennungs- und Verarbeitungsmechanismen. Für Sie wird es wahrscheinlich keinen großen Unterschied zwischen der ganzen Zahl 2 und der Dezimalzahl 2.0 geben, aber für den Computer ist er erheblich. Ihm muß also bei der Variablendeklaration mitgeteilt werden, welches der sogenannte Datentyp jeder Variablen ist. Diese Mitteilung geschieht durch den Doppelpunkt

hinter dem (oder den, durch Komma getrennten) Variablennamen; auf den Doppelpunkt folgt dann die Typangabe. In diesem Modul finden Sie den Datentyp *INTEGER*, andere werden bald folgen². Unter dem Datentyp *INTEGER* sind alle ganzen Zahlen zu verstehen, also die Zahlen: ... -2; -1; 0; 1; 2; ... u.s.w. Sie sehen an dem Programm, daß Variablen sowohl einzeln als auch in Gruppen deklariert werden können, sofern sie vom selben Typ sind, es ist ein Teil der Stilfrage, wie man diese Aufteilung vornimmt.

Eine Anmerkung zur Konstantendeklaration. Im Gegensatz zu dem, was bei Variablen gesagt war, muß für eine Konstante keine Typangabe erfolgen, denn der Compiler erkennt aus der Notationsform der Größe, die dem Gleichheitszeichen folgt, um welchen Datentyp es sich handelt. So hat in der oben angeführten Deklaration *CONST z = -2* der Bezeichner *z* den Datentyp *INTEGER*, die Deklaration *CONST z = -2.0* dagegen würde den Typ von *z* als *REAL*, also als Dezimalzahl festlegen (zum Datentyp *REAL* s. Programm 3 dieses Kapitels: *RealMath.odc*). Entsprechend erhält der Bezeichner *Titel* durch die bei der Deklaration erfolgte Identifikation mit einem Textstück (einer Folge von Schriftzeichen) den Datentyp "String" (Zeichenkette, exakt heißt der Datentyp nicht "String", sondern *ARRAY OF CHAR*; zur Definition s. das Programm 7.5, *String.odc*).

Sehen Sie sich nun den Anweisungsteil der Prozedur *Start* an. Beim Durchlesen wird er Ihnen, bis auf wenige Einzelheiten, unmittelbar klar sein, zumindest werden Sie verstehen, was die Prozedur tut. Sie finden dort einige *Out.Ln* und eine ganze Reihe von *Out.String* Anweisungen, die Sie inzwischen kennen. Dagegen sind die Ausgabeanweisungen *Out.Int(x, 4)*, *Out.Int(y, 4)*, *Out.Int(z, 4)* und *Out.Int(Summe, 4)* neu für Sie.

Außerdem finden Sie gleich zu Beginn drei sogenannte Zuweisungen. Die erste lautet *x := 12*. Das (zusammengesetzte) Zeichen *:=* ist in Component Pascal ein reservierter (Schlüssel-) Bezeichner. Es heißt Zuweisung, und tut genau das, was der Name sagt. In diesem Falle weist es der Variablen *x* den Wert 12 zu (gesprochen wird dies am besten in folgender Weise: "x werde 12"). In einer Zuweisung steht der zugewiesene Wert immer rechts vom Zuweisungszeichen *:=* und die Variable, der dieser Wert zugewiesen wird, stets auf der linken Seite. *x* ist eine Variable, weil sie als solche deklariert wurde, und sie mußte so deklariert werden, weil links vom Zuweisungszeichen *:=* nur eine Variable stehen darf, da die Zuweisung einen Wert ändert, was nur bei einer Variablen möglich ist.

In gleicher Weise werden in den folgenden Schritten den beiden Variablen *y* und *Summe* Werte zugewiesen. Bei der Zuweisung an die Variable *Summe* sehen Sie, daß einer Zahlvariablen nicht nur feste Werte, sondern auch arithmetische Ausdrücke aus Variablen und Konstanten zugewiesen werden können. Solche Ausdrücke werden vor der Zuweisung berechnet. Wichtig ist dabei, daß Wertzuweisungen an Variablen nicht gleich zu Beginn des Anweisungsteils erfolgen müssen, sie können an jeder gewünschten

² Eine Zusammenstellung der in Component Pascal vordefinierten Datentypen finden Sie im Anhang B.

Stelle geschehen. Bei einer Wertzuweisung, die ihrerseits auf der rechten Seite des Zuweisungszeichens Variablen enthält, müssen Sie darauf achten, daß jeder einzelnen der rechts stehenden Variablen vorher ein Wert zugewiesen wurde, anderenfalls sind die Ergebnisse unvorhersehbar, die Folgen eventuell fatal.

Falls Sie jetzt Schwierigkeiten haben mit dem Unterschied, der zwischen einer Variablen und dem Wert besteht, den die Variable enthält, grämen Sie sich nicht, das geht vielen Menschen an dieser Stelle so. Die Schwierigkeit wird sich im Lauf Ihrer weiteren Arbeit ganz sicher auflösen, vielleicht hilft Ihnen auch die folgende Erklärung ein wenig weiter.

Stellen Sie sich den ganzen Computer als einen Schrank mit einem Mechanismus vor, der im Innenraum Dinge hin- und herschiebt und miteinander verknüpft. Der Mechanismus heißt amerikanisch Central Processing Unit (abgekürzt CPU) und der Raum heißt Arbeitsspeicher. Sie denken sich diesen am besten als eine große Menge von nummerierten Schubladen, in die man etwas hineinlegen kann. Die Deklaration einer Konstanten, beispielsweise *z*, erfolgt nun in der Weise, daß der "kleine Mann" im Computer eine dem Datentyp der Konstanten entsprechende Schublade aussucht, die ein Etikett mit der Aufschrift: "Unveränderbar" erhält. In die Schublade legt er den Wert von *z*, die Zahl -2. Bei einer Variablendeklaration schreibt er auf die entsprechende Schublade die Bemerkung: "Inhalt veränderbar", außerdem den zugehörigen Datentyp. Zu diesem Zeitpunkt muß die richtige Schubladenart gewählt werden, daher muß bei der Anbringung des Aufklebers der Datentyp des künftigen Inhalts bereits festgelegt worden sein. In die Schublade hinein kommt im Moment noch nichts, das geschieht erst im Anweisungsteil durch die entsprechende Zuweisung.

In einer Art Liste vermerkt der "kleine Mann" die Nummern der Schubladen und die zugehörigen Namen. Da das Ganze sehr einem Eintrag in ein Adressenverzeichnis ähnelt, heißt die Speicherstelle im Computer auch ADRESSE; der Eintrag in die Adressenliste verbindet also einen Namen und die zu diesem Namen gehörende Adresse - die "Schublade" - in der sich der zum Namen gehörende Wert befindet. Wie für die Schublade das Wort ADRESSE, so gibt es auch für den Aufkleber der Schublade einen besonderen Namen, er wird *TYPDESCRIPTOR* genannt.

Zurück zu den oben kurz erwähnten Ausgabeanweisungen der Form *Out.Int(<GanzeZahl>, <Feldlänge>)*. Die vorstehende Ausgabeanweisung besagt, daß das Modul *Out* mit der Anweisung *Out.Int* eine ganze Zahl, also eine Zahl des Typs *INTEGER* ausgibt. Beachten Sie bitte, daß das *x* bei der Anweisung *Out.Int(x, 4)* innerhalb der Klammern nicht in Anführungszeichen oder Apostrophe eingeschlossen ist. Folglich interpretiert der Compiler dies *x* nicht als Text, sondern erkennt es als die zuvor deklarierte Variable *x*. Die Anweisung *Out.Int(x, 4)* bedeutet deshalb, daß der aktuelle Wert der Variablen *x* ausgegeben werden soll, das also, was in der "Schublade" des Arbeitsspeichers liegt. In diesem Fall ist das die zuvor zugewiesene Zahl 12. Entsprechendes gilt für *y* und *z* sowie für die Variable *Summe*, der zuvor das Ergebnis der Berechnung von *x+y+z-3* zugewiesen wurde, also die Zahl 20. Der Bezeichner der jeweiligen ganzen Zahl - im Modul *TutConstVar* sind dies die Konstante *z* und die Variablen *x*, *y*, *Summe* - steht in den

DIV liefert demgegenüber wieder eine ganze Zahl, nämlich den Ganzzahlteil der "normalen" Division. Somit gilt: $3 \text{ DIV } 4 = 0$, ebenso gilt: $13 \text{ DIV } 2 = 6$ und $13 \text{ DIV } 4 = 3$. Das Ergebnis der Operation *DIV* ist also stets eine ganze Zahl, diejenige Zahl, die bei der üblichen Division vor dem Dezimalpunkt des Ergebnisses steht.

Genau genommen handelt es sich bei der Ganzzahldivision um eine Rechenweise, die Sie in sehr frühen Schuljahren kennengelernt haben, deren Existenz Sie aber höchstwahrscheinlich kaum noch erinnern. Die Lehrer in der Grundschule nannten das die Division mit Rest, in mathematischem Fachjargon auch "Division modulo" genannt. Die Berechnung des Restes ist ebenso möglich wie die Berechnung des Ganzzahlteils. Der Operator, der die Berechnung durchführt, heißt *MOD* und es gilt: $13 \text{ MOD } 4 = 1$, denn nach der Grundschulrechenweise gilt $13 \div 4 = 3$ Rest 1. Ebenso finden Sie $17 \text{ MOD } 3 = 2$, denn es gilt: $17 \div 3 = 5$ Rest 2.

Möglicherweise sind Sie jetzt der Meinung, daß die Operation *DIV* vielleicht noch nützlich oder zumindest unumgänglich sei, aber den Sinn von *MOD* können Sie nicht erkennen. Ich bitte um Geduld. Sie werden diesen Sinn am einfachsten begreifen, wenn Sie an Programmen, die Sie später kennenlernen werden, die Verwendung dieses Operators studieren können. (Bevor Sie das Programm ausführen lassen, sollten Sie übrigens die einzelnen Berechnungen "zu Fuß" durchführen, Sie erhalten dadurch eine Kontrolle, ob Sie das Bisherige verstanden haben.)

Kommen wir zu den Ausgabeanweisungen. In einer Folge von *Out.String*, *Out.Int* und *Out.Ln* Anweisungen werden die Ergebnisse der vorangegangenen Berechnungen ausgegeben, wobei durch die Ihnen aus dem vorigen Programm bekannten Feldlängenangaben der *Out.Int* Anweisungen eine bestimmte Formattierung der Ausgabe im Log-Fenster erreicht wird⁴.

Sehen Sie sich die folgenden Programmzeilen an. Zunächst finden Sie die Anweisung $I := I + 1$. Bevor Sie protestieren, erinnern Sie sich bitte, daß dies als: "I werde I plus Eins" gelesen werden muß, nicht etwa als: "I gleich I plus Eins". Die zweite dieser Aussagen ist sicherlich falsch, es ist aber ebenso sicher möglich, den aktuellen Wert von *I* (aus der Schublade) zu nehmen, Eins zu addieren und das Ergebnis der Variablen *I* zuzuweisen (wieder in die Schublade zu legen), denn *I* ist als variabel deklariert. War in *I* bisher der Wert 9 gespeichert, so ab jetzt der Wert 10. In ähnlicher Weise wird in der nächsten Zeile *I* das Ergebnis einer etwas größeren Rechnung zugewiesen, der bisher dort gespeicherte Wert also erneut verändert. Diese Veränderung durch eine Zuweisung ist an jeder Stelle des Programms möglich, unabhängig davon, ob der aktuelle Wert von *I* bei der Bestimmung des neuen Wertes benutzt wurde oder nicht.

⁴ Diese Fomatierung erfordert einen sogenannten Festschriftfont, das ist eine Schriftart, bei der alle Zeichen einheitliche Breiten haben, das Gegenteil heißt Proportionalfont. Markieren Sie zur "korrekten" Darstellung der Ausgabe den entsprechenden Text und wählen Sie im Menü *Attributes* → *Font...* den Festschriftfont *Courier*. Dauerhaft wählen Sie einen Festschriftfont, indem Sie den Menüpunkt *Edit* → *Preferences* → *Default Font* wählen und dann die *Schriftart Courier*.

Der nächste Anweisungsblock enthält zwei neue Anweisungen, *INC* und *DEC*. Die Bezeichnungen stammen von den Wörtern increase und decrease, also erhöhen und erniedrigen, und das ist es, was sie tun, sie erhöhen bzw. erniedrigen die Werte der Variablen in den Klammern. *INC(I)* erhöht den Wert von *I* um Eins; dies entspricht der Zuweisung $I := I + 1$, analog erniedrigt *DEC(I)* den Wert von *I* um Eins, entspricht also der Anweisung $I := I - 1$. Jetzt werden Sie sich fragen, warum diese beiden gleichwertigen Möglichkeiten existieren, warum man sich die Mühe gemacht hat, besondere Befehle für etwas zu erdenken, das so einfach mit den vorhandenen Mitteln ausgedrückt werden kann. Manche Programmierer denken ebenso und verwenden so gut wie nie die Anweisungen *INC* und *DEC*. Es gibt aber einen Unterschied, der sich bei großen Programmen durchaus bemerkbar machen kann. *INC* und *DEC* sind direkte Befehle und dadurch sehr viel schneller als z. B. die Anweisung $I := I + 1$, bei der zuerst eine Berechnung und dann eine Zuweisung ausgeführt werden müssen. Man kann aber auch um andere Beträge als 1 erhöhen oder erniedrigen, entsprechend sind *INC(I, 3)* gleichbedeutend mit $I := I + 3$, *DEC(I, 7)* gleichbedeutend mit $I := I - 7$ und *INC(I, J * 2 + 4)* entspricht der Zuweisung $I := I + J * 2 + 4$.

Einen weiteren Unterschied zwischen den beiden Verfahren zur Wertveränderung einer Ganzzahl-Variablen finden Sie in den folgenden Programmzeilen. Es beginnt mit der Wertzuweisung an eine *BYTE*-Variable, die genauso geschieht, wie bei den bisher verwendeten *INTEGER*-Variablen. Die anschließende, in Kommentarklammern stehende Zeile weist Sie auf eine Component Pascal Regel hin, die im Zusammenhang mit dem jeweiligen Typ einer Ganzzahl-Variablen zu sehen ist. Während Zuweisungen wie $I := I + 3$ möglich sind, weil *I* eine *INTEGER*-Variable ist, erzeugt der Compiler bei dem Versuch, die Zuweisung $B := B + 3$ durchzuführen, eine Fehlermeldung, da die Variable *B* vom Typ *BYTE* ist, der der berechnete Wert nicht direkt zugewiesen werden kann. Arithmetische Berechnungen mit ganzen Zahlen, deren Typ *INTEGER* oder kleiner ist, liefern in Component Pascal stets Ergebnisse des Typs *INTEGER* (sonst sind die Ergebnisse vom Typ *LONGINT*). Dagegen lassen sich die Wertänderungen mittels *INC* und *DEC* auf Variablen aller Ganzzahltypen anwenden, ohne den Typ der jeweiligen Variablen im Ergebnis zu ändern. Die beiden Programmzeilen *INC(B, 3)* und *DEC(S, 3254)*, die vom Compiler nicht beanstandet werden, zeigen Ihnen dies.

Die Zuweisung $I := S$ ist wiederum ohne weiteres möglich; man sagt, die Variablen *S* und *I* seien aufwärts zuweisungskompatibel. Damit ist die Tatsache gemeint, daß im Sinne der oben beschriebenen mathematischen Inklusion alle Zahlen eines "kleineren" Ganzzahltyps in allen "größeren" Ganzzahltypen enthalten sind. Der Compiler führt bei der Zuweisung des Wertes der *SHORTINT*-Variablen *S* an die *INTEGER*-Variable *I* eine automatische Typumwandlung durch. Die umgekehrte "Zuweisungsinkompatibilität" eines größeren auf einen kleineren Zahltyp, wie der Sachverhalt offiziell genannt wird, finden Sie in der in Kommentarklammern stehenden Zeile $T := S$, in der (vergeblich) der Versuch gemacht wird, einer *SHORTINT*-Variablen einen *INTEGER*-Wert zuzuweisen.

Um trotzdem in der umgekehrten Richtung eine Typumwandlung durchführen zu können, gibt es in Component Pascal den besonderen Operator *SHORT*, dessen Verwendung Sie in den folgenden Programmzeilen sehen. Einmalige Anwendung von *SHORT* reduziert den jeweiligen Variablenwert auf den nächstkleineren Zahltyp, mehrmalige, geschachtelte Anwendung ist ebenfalls möglich. Allerdings müssen Sie unbedingt darauf achten, daß der Zahlwert, den Sie umwandeln wollen, im Bereich des neuen Zahltyps liegt, da anderenfalls merkwürdige Effekte auftreten, wie Sie an den nächsten Programmschritten sehen.

In der Zeile $B := c$ wird der *BYTE*-Variablen B der Wert der Konstanten c , die Zahl -128, zugewiesen, die entsprechend dem oben über den Gültigkeitsbereich des Typs *BYTE* Gesagten die kleinste in B speicherbare Zahl ist. Dieser Wert wird anschließend mit dem zu *SHORT* entgegengesetzten Operator *LONG*, der eine explizite Typumwandlung in den jeweils nächstgrößeren Zahltyp ermöglicht, in die *SHORTINT*-Variable S kopiert und nach der folgenden Anweisung *DEC(S)* enthält die Variable S den Wert -129. Dagegen erhält die Variable B durch die Anweisung *DEC(B)* nicht den vom menschlichen Leser erwarteten Wert -129 zugewiesen, sondern den zu -128 im Datentyp *BYTE* unmittelbar benachbarten Wert +127. Diese im ersten Moment verwirrende Tatsache können Sie vielleicht besser verstehen, wenn Sie sich die Zahlen des Typs *BYTE* linear auf einem Band angeordnet denken, dessen beide Enden wie ein Kreis miteinander verbunden sind. Auf diesem Kreisband sind an der Verbindungsstelle die Zahlen -128 und +127 tatsächlich Nachbarn. Sie sehen, daß Sie bei Typumwandlungen eine gewisse Vorsicht walten lassen müssen.

Die anschließenden Ausgabeanweisungen *Out.Int(B, 10)*; *Out.String(" # ");* *Out.Int(S, 20)* dürften ohne große Erklärungen verständlich sein, sie dienen dazu, die eben beschriebene Begrenztheit des Zahltyps *BYTE* darzustellen. Auf eine Neuigkeit möchte ich Sie jedoch hinweisen. In der zweiten Ausgabeanweisung sehen Sie das Zeichen #. Dieses Zeichen ist in Component Pascal ein reservierter Bezeichner, das einem mathematischen Ungleichheitszeichen (\neq), das es auf der Tastatur nicht gibt, ähnelt und deshalb als Ersatz gewählt wurde.

Die nächsten Ausgabeblöcke enthalten ebenfalls einige Neuigkeiten. In der Zeile $S := 02EH$, in der eine Wertzuweisung an die *SHORTINT*-Variable S erfolgt, sehen Sie eine etwas ungewöhnliche Darstellung der ganzen Zahl 46. Diese Schreibweise entstammt nicht dem gewohnten Dezimalsystem mit den 10 Ziffern 0 bis 9, sondern dem Hexadezimalsystem, in dem zur Darstellung von Zahlen 16 Ziffern verwendet werden. Die ersten 10 Ziffern sind dieselben wie im Dezimalsystem, also die Ziffern 0 bis 9, für die restlichen 6 Ziffern verwendet man im Hexadezimalsystem die Großbuchstaben A bis F. Die Hexadezimalzahl F entspricht also der Dezimalzahl 15. Um keine Unklarheiten über das verwendete Zahlssystem aufkommen zu lassen, werden in Component Pascal Hexadezimalzahlen stets mit der Ziffer Null eingeleitet und mit dem Großbuchstaben H (bzw. L, wie Sie etwas weiter unten sehen werden) beendet. Der Bezeichner *02EH* enthält also zwischen der einleitenden 0 und dem beendenden H als eigentlichen Inhalt

die Hexadezimalzahl 2E, die der Dezimalzahl 46 entspricht, wie Sie mit einiger Überlegung herausfinden können und Ihnen die Bildschirmausgabe des Programms zeigen wird.

Eine weitere Neuigkeit ist die Ausgabeanweisung *Out.Int(S + 1, 20)*. Alle Ausgabeanweisungen in den bisherigen Programmen hatten als Parameter, wie man die in den Klammern stehenden Ausdrücke allgemein nennt, einfache Variablen. Die vorstehende Ausgabeanweisung zeigt, daß als Parameter auch zusammengesetzte Ausdrücke möglich sind. Derartige Ausdrücke werden vor der Ausgabe berechnet, bis sich ein einfacher Wert ergibt, wie er auch in einer gesonderten Zuweisung entstanden wäre.

In gleicher Weise erhalten in den nächsten Zeilen die *INTEGER*-Variable I den Wert 0FFFFFFFH (dezimal -1) und die *LONGINT*-Variable L den Wert 0FFFFFFFL (dezimal +4294967295) zugewiesen, wobei zu beachten ist, daß der zweite Wert nicht mit einem H, sondern mit einem L endet. Dies L teilt dem Leser und dem Compiler mit, daß 0FFFFFFFL nicht als *INTEGER*-, sondern als *LONGINT*-Wert interpretiert werden soll und deshalb eine andere Zahl darstellt als 0FFFFFFFH. Der Unterschied beruht auf der oben für *BYTE*-Zahlen dargestellten, für Zahlen der anderen Typen analog geltenden Endlichkeit der Zahlbereiche, die Sie sich ebenfalls jeweils als Kreisband aufgerollt denken können. Das Ergebnis dieses Unterschieds finden Sie in den folgenden Bildschirmausgaben wieder (Ausgabeanweisungen sind Ihnen inzwischen vertraut, ich verzichte daher künftig bis auf Ausnahmefälle auf eine explizite Erwähnung).

Die weiteren Teile des Moduls *TutIntMath* sind nahezu selbsterklärend, Sie sehen einige explizite und implizite Typumwandlungen und als vorletzten Block finden Sie die in Component Pascal enthaltenen Möglichkeiten, die Unter- und Obergrenzen der verschiedenen Ganzzahltypen zu ermitteln. Dazu dienen die Operatoren *MIN* und *MAX*, die als einzigen Parameter den Zahltyp benötigen, für den die entsprechenden Werte ermittelt werden sollen.

Der letzte Programmblock zeigt Ihnen, daß *MIN* und *MAX* jeweils eine zweite Fähigkeit haben. Wenn Sie den Operatoren nicht einen, sondern zwei Parameter übergeben, die wahlweise Zahlkonstanten oder -variablen sein können, ermitteln *MIN* und *MAX* die kleinere bzw. größere der beiden Zahlen, die jeweils anschließend weiterverwendet, beispielsweise auf dem Bildschirm angezeigt werden kann.

Machen Sie sich bitte, bevor Sie das Programm ausführen lassen, die Mühe, alle Ausgabeanweisungen auszuwerten und die vollständige Bildschirmausgabe so aufzuschreiben, wie sie Ihrer Meinung nach vom Compiler erzeugt wird, denn nur durch aktives Nachvollziehen können Sie sich davon überzeugen, ob Sie wirklich alles verstanden haben.

3.3. Die reellen Zahlen

Laden Sie die Datei *RealMath.odc* und sehen Sie sich den Deklarationsteil des Programms an. Dieser ähnelt in mancher Hinsicht dem, was Sie bereits aus dem vorigen Programm kennen.

Zunächst finden Sie die beiden Ihnen inzwischen bekannten Schlüsselwörter *CONST* und *VAR*. Die Konstantendeklaration enthält einen etwas längeren Näherungswert der Zahl π , wobei Sie und der Compiler an der Schreibweise erkennen können, daß es sich um eine reelle Zahl handelt. In Component Pascal enthalten Dezimalzahlen, die im Deutschen bisher vorwiegend mit einem Komma als Trennzeichen geschrieben werden, allerdings wie im Englisch-Amerikanischen statt des Kommas einen Punkt, aus 3,75 wird somit 3.75; wenn Sie also in einem Component Pascal Programm eine Dezimalzahl verwenden, denken Sie bitte daran, diese mit einem Punkt als Trenner zu schreiben.

Bei den Variablendeklarationen sehen Sie in den ersten beiden Zeilen die Namen *Summ*, *Diff*, *Prod*, *Divi* sowie *A*, *B*, *C*, *D* und als Typ dieser Variablen *REAL*. Dieser Typ ermöglicht es, in den Variablen reelle Zahlen zu speichern, die im Rechner eine wesentlich andere Darstellung haben als ganze Zahlen. In der dritten Zeile werden die Variablen *Y* und *Z* mit dem Typ *SHORTREAL* deklariert. Ähnlich wie es bei den ganzen Zahlen vier Typen gibt, existieren bei den reellen Zahlen zwei Typ-Varianten, die sich wie die verschiedenen Ganzzahltypen durch unterschiedlichen Speicherbedarf auszeichnen. Beide Varianten werden zusammen als Fließkommatypen oder Fließpunkttypen (floating point types) bezeichnet. Schließlich werden noch zwei Ganzzahlvariablen deklariert, mit denen die im Anweisungsteil zu findenden gemischten Rechnungen mit reellen und ganzen Zahlen durchgeführt werden sollen.

Die ersten Zeilen des Anweisungsteils sehen den entsprechenden Zeilen des Moduls *TutIntMath* ähnlich, es handelt sich um Zuweisungen sowie Addition, Subtraktion und Multiplikation reeller Zahlen. In der Zeile *Divi := A / B* finden Sie ein bisher nicht aufgetauchtes Zeichen, den Schrägstrich *'/'* (amerikanisch: forward slash oder slash genannt) als Divisionsoperator für *REAL*-Zahlen, der Ihnen vielleicht als Teilungsstrich bei Bruchzahlen bekannt ist. In Component Pascal ist dies Zeichen nur für reelle Zahlen reserviert, d.h. alle Zahlen (auch *INTEGER*-Zahlen, wie *I* in der Zeile *C := I / A*), die mit dem *'/'* geteilt werden, interpretiert der Compiler als reelle Zahlen, das Ergebnis ist stets eine reelle Zahl, Sie sehen dies in der weiter unten durchgeführten Bildschirmausgabe des Moduls.

Die Zeile *B := Math.Sin(Z/2) * Math.Cos(Math.Pi())* weist der Variablen *B* die Zahl -1.0 zu, wie Sie selbst berechnen oder der Bildschirmausgabe des Moduls entnehmen können. Bei dieser Berechnung wird zum einen die im Modul *TutRealMath* selbst deklarierte und zuvor der Variablen *Z* zugewiesene Konstante *Pi* verwendet, zum anderen der Wert *Math.Pi()*. Die Schreibweise dieses Bezeichners mit dem separierenden Punkt kennen Sie bereits vom Modul *Out*, sie besagt, daß der Bezeichner *Pi()* aus dem Modul *Math* stammt, das deshalb in der Importliste von *TutRealMath* auftaucht (das leere Klammerpaar bei *Math.Pi()* bitte ich, im Moment zu ignorieren, dessen Bedeutung werden Sie im 5. Kapitel kennenlernen).

Modul *Math* ist ein BlackBox Standardmodul, das wichtige und oft gebrauchte mathematische Operationen zur Verfügung stellt, deren schwierige Programmierung Ihnen also erspart bleibt.

Sofern Sie jetzt neugierig genug sind, alle weiteren Operationen kennenzulernen, die das Modul *Math* enthält, oder aber alle Operationen des Moduls *Out*, sollten Sie den entsprechenden Modulnamen an irgendeiner geeigneten Stelle innerhalb des BlackBox Systems schreiben, markieren und sich entweder die Schnittstellenbeschreibung (Menü *Info* → *Client Interface*) oder die kommentierte Schnittstellenbeschreibung (Menü *Info* → *Documentation*) anzeigen lassen. Bei der Schnittstellenbeschreibung (interface definition) handelt es sich um eine im Stil eines Component Pascal Moduls gehaltene, vom System aktuell erstellte Zusammenfassung der vom jeweiligen Modul exportierten Bezeichner. Dies ist eine sehr bequeme Möglichkeit, sich rasch und ohne langwieriges Blättern in (entweder nicht griffbereiten oder nicht existierenden) Druckmaterialien einen Überblick über die Schnittstellendefinition zu verschaffen. Der Unterschied zu einem Modul liegt darin, daß dies interface, wie die Schnittstellenbeschreibung im Informatikerjargon auch genannt wird, lediglich eine Information für den menschlichen Leser darstellt und nicht kompilierbar ist. Die Schnittstellenbeschreibung ist in der beim Kompilieren eines Moduls generierten Symboldatei gespeichert, sie wird nicht mit dem Schlüsselwort *MODULE*, sondern mit dem Wort *DEFINITION* eingeleitet und stellt nur die Exporte, also die außerhalb des Moduls verfügbaren Bezeichner dar, nicht die sogenannte Implementation, den kompilierbaren Programmcode also, den nur das Modul selbst kennt. Die dadurch mögliche Trennung zwischen öffentlichen (interface) und privaten (implementation) Teilen eines Moduls verwirklicht ein sehr wichtiges Prinzip der Informatik, das Verstecken von Details (information hiding), die ein Benutzer der öffentlichen Dienste eines Moduls nicht erfahren muß und soll.

Bei der zweiten Informationsmöglichkeit, der Modul-Dokumentation, handelt es sich um eine von den Programmierern des BlackBox Systems erstellte Textdatei, die ebenfalls das interface, die *DEFINITION*, enthält, und im Anschluss daran eine Erläuterung der einzelnen Bestandteile dieser Schnittstelle. Verständlicherweise ist eine derartige Dokumentation nur für ausgewählte Module vorhanden, dann nämlich, wenn der Programmierer sich die Mühe der Kommentierung gemacht hat; dagegen läßt sich, sofern die entsprechende Symboldatei existiert, das reine interface immer erzeugen, weil es sich um eine im System vorhandene Dienstleistung handelt. Bei Ihrer künftigen Programmierarbeit sollten Sie so bald wie möglich anfangen, für die von Ihnen erstellten Module entsprechende Dokumentationen zu erstellen, Sie wollen schließlich, daß Andere diese Module nutzen können, ohne den gesamten Quelltext lesen (und verstehen) zu müssen, möglicherweise wollen Sie auch die Quelltexte, in denen erhebliche Arbeit steckt, nicht ohne weiteres veröffentlichen. Um so wichtiger ist eine klare und unmißverständliche Dokumentation der Möglichkeiten, die Ihre Module bieten.

Der zweite Block des Anweisungsteils enthält die Ausgabeanweisungen für die vorstehenden Berechnungen und Zuweisungen. Diese Ausgabeanweisungen unterscheiden sich fast nicht von den Ihnen vom Modul *TutIntMath* bekannten Ausgabeanweisungen, die einzige Änderung besteht darin, daß für die Bild-

schirmdarstellung der *REAL*-Zahlen der dafür im Modul *Out* existierende Operator *Out.Real* verwendet wird.

Wesentliche Neuigkeiten finden Sie erst wieder in dem mit *Typkonversion* überschriebenen Programmteil. Die erste, wieder einmal in Kommentarklammern stehende Zuweisung ist für den Compiler unzulässig, da versucht wird, der Ganzzahlvariablen *LSumm* den in *Summ*, einer *REAL*-Variablen, enthaltenen Wert zuzuweisen. Alle Ganzzahltypen sind jedoch "kleiner" als der Datentyp *REAL*, die Zuweisung ist deshalb nicht typkompatibel, es ist nicht klar, wie z. B. die Zahl 3.14 als "ganze Zahl" aussehen sollte. Trotzdem gibt es die Möglichkeit, ebenso wie in der Mathematik eine derartige Zuweisung durchzuführen. Die folgende Zeile *Lsum := ENTIER(Summ)* macht Sie mit dem dafür zuständigen Component Pascal Operator *ENTIER* bekannt. Das Wort *ENTIER* ist ein französisch-englisches Mischwort und bedeutet soviel wie "Ganz", in Component Pascal wird mit diesem Operator eine reelle Zahl in eine ganze Zahl umgewandelt. Die Umwandlung geschieht in der Form, daß der jeweilige Parameter von *ENTIER*, die in den Klammern stehende Zahl, auf die nächste ganze Zahl abgerundet wird. Es gilt also *ENTIER(7.4803) = 7* und *ENTIER(-7.4803) = -8*, ebenso gilt *ENTIER(-7.8) = -8*, denn in den beiden letzten Fällen ist -8 die nächstkleinere ganze Zahl. Wichtig ist die Tatsache, daß das Ergebnis einer Typumwandlung mit *ENTIER* immer vom Typ *LONGINT* ist, Sie müssen also gegebenenfalls mit dem Operator *SHORT* weitere Typumwandlungen durchführen, wenn die Variable, die das Ergebnis aufnehmen soll, von einem kleineren Typ ist, Beispiele dafür finden Sie in den anschließenden Programmzeilen.

Eine weitere Neuigkeit finden Sie in der Zeile *C := C * 1.0E+03*. Da *REAL*-Zahlen sehr groß und deshalb sehr lang sein können, gibt es die Möglichkeit, solche Zahlen in sogenannter wissenschaftlicher Schreibweise einzugeben. Damit ist die mathematische Exponentialdarstellung der Zahl als Potenz mit Basis 10 gemeint. Für die Zahl -230.45 ergibt sich in dieser Darstellung

$$- 230.45 = - 2.3045 \cdot 10^2 = - 2.3045E+02$$

wobei die letzte Form die für den Compiler nötige Ersatzschreibweise darstellt, da mit den normalen Tastaturen keine Exponenteneingabe möglich ist; "E" steht für Exponent, womit der Exponent zur Basis 10 gemeint ist. Diese Schreibweise ist Ihnen vielleicht bekannt, alle wissenschaftlichen Taschenrechner geben Zahlen, die in der Notation mit Dezimalpunkt nicht in die Anzeige passen, in ähnlicher Form aus.

Die anschließenden Programmzeilen enthalten ein Reihe von Berechnungen und Zuweisungen, an denen Sie die Wirkungsweise der Rechenoperatoren im Zusammenhang mit verschiedenen Typumwandlungen studieren können. Als erstes sehen Sie an den Zeilen *Y := Y - 1.94 + 2 - Y* und *Y := SHORT(Y - 1.94 + 2 - A)*, daß der Compiler die Ergebnisse von Berechnungen dem Typ *SHORTREAL* zuordnet, wenn die Typen der in der Berechnung vorkommenden Zahlkonstanten und/oder Variablen von Ganzzahl- bzw. Fließkommatypen nicht größer als *SHORTREAL* sind, im anderen Fall werden sie dem Typ *REAL* zugeord-

net. Folglich ist bei Zuweisung des Ergebnisses an eine Zahlvariable einer der Typkonversionsoperatoren nötig, wenn die Typen der Variablen und des Ergebnisses nicht zuweisungskompatibel sind.

Während die Typumwandlung einer *REAL*- in eine *INTEGER*-Zahl nur explizit mit Hilfe des Operators *ENTIER* möglich ist, ersehen Sie aus den Berechnungen und der etwas weiter unten im Programm zu findenden Zeile *A := S*, daß die Zuweisung eines *INTEGER*-Wertes an eine *REAL*-Variable problemlos durchgeführt werden kann. Beide *REAL*-Typen sind "größer" als alle Ganzzahl-Typen, deshalb kann der Compiler eine automatische Typumwandlung vornehmen, eine ganze Zahl wie -5 wird bei der Zuweisung an eine *REAL*-Variable ohne Wertänderung in die reelle Zahl -5.0 umgewandelt.

In diesem Zusammenhang lernen Sie einen weiteren Component Pascal Operator kennen, *ABS*. Dieser Operator dient wie sein namensgebender Vetter in der Mathematik dazu, den Absolutbetrag einer Zahl zu ermitteln, eine (ganze oder reelle) Zahl also vorzeichenfrei zu machen.

Die restlichen Programmteile geben zum einen mit den bekannten Operatoren *MIN* und *MAX* die Unter- und Obergrenzen der beiden *REAL*-Typen aus, zum anderen lernen Sie eine neue Component Pascal "Zahl" kennen, die "Zahl" *INF*. *INF* steht für "infinity" und symbolisiert die aus der Mathematik bekannte "Zahl" ∞ . Die Zeile *D := MAX(REAL) / INF* weist der *REAL*-Variablen *D* den Wert Null zu, denn die Division jeder noch so großen endlichen Zahl durch "Unendlich" ergibt den Wert Null.

3.4. Logische Entscheidungen

Der folgende Abschnitt mag Ihnen zunächst ein wenig schwierig vorkommen, denn er enthält Material, das Sie in dieser Weise wahrscheinlich nicht kennen. Es geht aber um Dinge, die für die Konstruktion von Programmen von großer Wichtigkeit sind, es geht um logische Vergleiche und Entscheidungen. Ich erwähnte bereits, daß rechnerintern Alles in Form von Zahlen gespeichert und verarbeitet wird, unabhängig davon, ob es sich für uns um Zahlen oder Wörter handelt. Nun müssen bei sehr vielen Arbeitsschritten eines Programms vom Rechner Entscheidungen getroffen werden, die auf einen Vergleich solcher Zahlen hinauslaufen in dem Sinne, daß geprüft wird, welche von zwei Zahlen die kleinere bzw. größere ist, ob sie gleich oder ob sie ungleich sind.

Formalisiert sieht dies so aus, daß für die Werte zweier Zahlen *a* und *b* genau eine der folgenden Möglichkeiten gilt; *a = b* (gesprochen: *a* ist gleich *b*); *a < b* (gesprochen: *a* ist kleiner als *b*); *a > b* (gesprochen: *a* ist größer als *b*) und *a ≠ b*. Außerdem läßt man noch zwei schwächere Formen des Vergleichs zu: *a ≤ b* (gesprochen: *a* ist kleiner als oder gleich *b*) und *a ≥ b* (gesprochen: *a* ist größer als oder gleich *b*)⁵.

⁵ Eine Zusammenstellung der Vergleichsoperationen für die vordefinierten Component Pascal Datentypen mit ihren jeweiligen Gültigkeitsbereichen finden Sie im Anhang B.

Auf der Grundlage dieser Vergleiche existiert nun das, was auf den Engländer George Boole zurückgeht und deshalb Boole'sche Mathematik genannt wird. Dabei handelt es sich um etwas Ungewohntes aber letztlich doch sehr Simples, nämlich die Auswertung der eben dargestellten Vergleiche. Nehmen Sie an, Sie haben zwei Zahlen, $A = 22$ und $B = 12$, und Sie (lassen den Computer) fragen, ob $A = B$ gilt. Die Antwort darauf lautet: falsch, bzw. in Component Pascal *FALSE*. Fragen Sie dagegen, ob $A \geq B$ gilt, so ist die Antwort: richtig, bzw. in Component Pascal *TRUE*. Ein derartiger Vergleich hat also stets genau eins von zwei möglichen Ergebnissen: *FALSE* oder *TRUE*. Außerdem lernen Sie damit einen weiteren Component Pascal Datentyp kennen, den Typ *BOOLEAN*. Variablen des Typs *BOOLEAN* können nur diese beiden Werte annehmen, gerade deshalb sind sie für Programmkonstruktionen äußerst nützlich.

Laden Sie die Datei BoolMath.odc und sehen Sie sich das Programm an. Die *IMPORT*-Anweisung enthält diesmal nicht das gewohnte Modul *Out*, sondern das Modul *StdLog*. Dies wird hier benutzt, um die Ausgabe BOOLE'scher Werte durchführen zu können, für die Modul *Out* keine Möglichkeit enthält. Im Deklarationsteil von *Start* finden Sie drei Variablen mit den hübschen und bedeutungslosen Namen *Alle*, *Wenige*, *Keine* und als Typ dieser Variablen *BOOLEAN*. Außerdem finden Sie drei weitere Variablen vom Typ *INTEGER*, Zahlvariablen also, denn es soll in dem Programm um die erwähnten Vergleiche von Zahlen gehen. Im Anweisungsteil werden diesen drei Variablen als erstes Werte zugewiesen. Die erste Anweisung, die sich mit einer Variablen vom Typ *BOOLEAN* beschäftigt, ist: *Alle := (A = 22)*. Die rechte Seite der Zuweisung steht hier zur Verdeutlichung in Klammern, die aber nicht notwendig sind, da Klammern, wie in der Algebra, nur gesetzt werden müssen, wenn von den allgemein gültigen Regeln abgewichen werden soll, die Zuweisung des Wertes an die Variable *Alle* jedoch erst erfolgt, wenn der rechts stehende Ausdruck vollständig ausgewertet worden ist.

Die Frage, ob $A = 22$ eine richtige oder eine falsche Aussage ist, wird der Compiler in diesem Fall mit "Wahr" beantworten, die Boole'sche Variable *Alle* erhält somit den Wert *TRUE* zugewiesen; analog werden die weiteren Ausdrücke behandelt. Natürlich können die in einer Boole'schen Variablen gespeicherten Werte, wie bei anderen Variablen auch, im Programmablauf geändert werden, z. B. kann durch die Zuweisung *Keine := Wenige* der Wert der Variablen *Wenige* in die Variable *Keine* kopiert werden. Unter Verwendung der möglichen Boole'schen Werte *FALSE* und *TRUE* sind auch direkte Wertzuweisungen möglich, wie Sie weiter unten im Programm sehen.

In den Zuweisungen: *Wenige := Alle & Keine*, *Wenige := Alle & ~Keine*, *Wenige := Alle OR Keine* und *Wenige := Alle OR ~Keine* lernen Sie die Component Pascal Schlüsselbezeichner *&* (und), *OR* (oder) und *~* (nicht, logische Negation; das Zeichen *~* steht als Ersatz für das auf der Tastatur nicht vorhandene mathematische Negationszeichen \neg) kennen, die es erlauben, die Wahrheitswerte komplexer Ausdrücke zu ermitteln. Die Auswertung der ersten dieser Zuweisungen ergibt Folgendes: *Alle & Keine* hat den Wert *FALSE*, da *Alle* an dieser Stelle des Programms den Wert *TRUE*, *Keine* den Wert *FALSE* hat. Die Werte der übrigen Zuweisungen ergeben sich entsprechend.

Am einfachsten lassen sich die Ergebnisse Boole'scher Ausdrücke in einem Diagramm darstellen.

A	B	A & B	A OR B	~ A
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	FALSE

Logische Verknüpfungen - Wahrheitstafel für 2 Variablen

Wie Sie sehen, ist der Wert einer Und-Verknüpfung genau dann *TRUE*, wenn beide Einzelwerte *TRUE* sind; bei einer Oder-Verknüpfung genügt es, daß wenigstens einer der Einzelwerte *TRUE* ist um den Gesamtwert *TRUE* zu erzeugen, und die Negation dreht einen Wert einfach um. Aus diesen drei einfachen Grundtatsachen lassen sich beliebig komplizierte Ausdrücke entwickeln, einige sind zur Illustration im Rest des Programms zu finden.

Eine in diesem Zusammenhang wichtige Tatsache muß noch erläutert werden, die abgekürzte Auswertung Boole'scher Ausdrücke. In einer Und-Verknüpfung wie *Alle & Keine* wird eine vollständige Prüfung des zweiten Einzelwertes nur durchgeführt, sofern der Wert des ersten Operanden *TRUE* ist, da andernfalls das Gesamtergebnis unabhängig vom Wert des zweiten Operanden *FALSE* sein wird. Diese Auswertungsart ist für die Programmierarbeit wichtig, da oft der zweite Ausdruck nur existiert, sofern der erste *TRUE* ist.

Werten Sie bitte alle Ausdrücke des Programms selbst aus, bevor Sie das Programm ausführen lassen. Schreiben Sie Ihre Ergebnisse auf und achten Sie dabei besonders auf die Unterschiede in den Zuweisungen *Wenige := ~Alle & Keine* und *Wenige := ~(Alle & Keine)*. Während in der ersten dieser Zuweisungen durch den Umkehroperator *~* nur der Wert der Variablen *Alle* umgekehrt wird, geschieht dies in der zweiten mit dem Wert der gesamten Klammer, nachdem dieser ermittelt wurde. Sie sollten diese Boole'schen Ausdrücke richtig verstanden haben, Sie werden sie in Ihrer zukünftigen Programmierarbeit immer wieder brauchen.

Am Ende des Programms finden Sie in der Anweisung *StdLog.Bool(ODD(A))* noch eine weitere Neuigkeit, den Component Pascal Operator *ODD*. Dieser prüft, ob der Wert des Parameters in der Klammer (in diesem Fall also der Wert von *A*) eine ungerade Zahl ist oder nicht, was natürlich nur bei *INTEGER*-Zahlen möglich ist, und er liefert als Ergebnis einen Wert vom Typ *BOOLEAN*, nämlich *FALSE*, wenn die Zahl gerade, und *TRUE*, wenn die Zahl ungerade ist. Also gilt $ODD(A) = FALSE$, wenn *A* den Wert 22, und 'logischerweise' gilt $\sim(ODD(B)) = TRUE$, wenn *B* den Wert 12 enthält.

3.5. Zeichen

In diesem Abschnitt lernen Sie den letzten der skalaren (einfachen) Component Pascal Datentypen kennen. Sehen Sie sich dazu die Datei `Char.odc` an. In der Variablendeklaration finden Sie die drei Variablenbezeichner `Char1`, `Char2`, `Katz` mit der Typbezeichnung `CHAR` (gesprochen: kar). Diese leitet sich vom englisch-amerikanischen Wort `character` ab, womit nicht etwa eine mögliche psychologische Bedeutung der Variablen gemeint ist, das Wort heißt ins Deutsche übertragen schlicht Zeichen. Gemeint ist ein einzelnes der dem Compiler bekannten Zeichen, das ein Buchstabe, eine Ziffer, ein Satzzeichen oder auch eins der Rechenzeichen sein kann. Vereinfacht gesagt handelt es sich um jedes Zeichen, das Sie mit der Tastatur Ihres Computers erzeugen können. Nicht jedes der Zeichen läßt sich auf dem Bildschirm oder durch den Drucker darstellen, da es sich bei einigen um interne Arbeitsanweisungen der Systemumgebung handelt, aber alle Zeichen können durch Tasten(-kombinationen) erzeugt werden und sie sind vom Typ `CHAR`. Diese Zeichen sind in der international standardisierten Unicode-Tabelle zusammengefaßt, in der ihre Reihenfolge festgelegt ist. Die Unicode-Tabelle enthält insgesamt 65536 Zeichen, die in einzelne Seiten (sogenannte code pages) aufgeteilt sind. Die erste dieser Seiten heißt Latin1-Zeichensatz und besteht aus 256 in (West-) Europa und den USA gebräuchlichen Zeichen. Wegen des erwähnten Umfangs des Unicode-Zeichensatzes gibt es in Component Pascal einen zweiten Zeichen-Datentyp, `SHORTCHAR`, der nur die Zeichen des Latin1-Zeichensatzes enthält⁶. Die nächsten drei im Programm deklarierten Variablen `SChar1`, `SChar2`, `SKatz` sind von diesem Typ.

In der ersten Zeile des Anweisungsteils von `Char.odc`, `Char1 := "A"`, sehen Sie, daß eine `CHAR`-Variable in üblicher Form durch Zuweisung einen Wert erhält. Wichtig dabei ist, daß alle direkt eingegebenen `CHAR`- und `SHORTCHAR`-Werte wie Text zu behandeln, also in Anführungszeichen zu setzen sind, das gilt auch für Ziffern, wenn diese vom Compiler als Zeichen gewertet werden sollen. Beachten Sie dabei bitte, daß zwischen den Anführungszeichen wirklich nur ein einzelnes Zeichen stehen darf, ein Zuweisungsversuch wie `Char1 := "z"` führt zu einer Beschwerde des Compilers, da auf der rechten Seite des Zuweisungszeichens zwischen den Anführungszeichen zwei Zeichen stehen, vor dem Buchstaben `z` befindet sich ein Leerzeichen, das anders als bei Verwendung von Leerzeichen im Programmcode als "Zeichen" ein Zeichen wie alle anderen ist. Hier wird versucht, der `CHAR`-Variablen `Char1` nicht ein einzelnes Zeichen, sondern eine Zeichenkette zuzuweisen, was der Compiler mit Grund beanstandet.

Die zweite und dritte Zeile machen klar, daß die Zuweisung von `CHAR`-Werten an `SHORTCHAR`-Variablen nur unter Verwendung des Operators `SHORT` möglich ist. Demgegenüber zeigt die Zeile `SChar1 := "T"`, daß die direkte Zuweisung von Zeichenkonstanten wie `"T"` an eine Variable des Typs

⁶ Eine Zusammenstellung aller druckbaren Latin1 Zeichen finden Sie im Anhang A und in der BlackBox Hilfe im Dokument `Character.Set`. Mit dem Programm `Latin1.odc` im Anhang können Sie diese Zeichen auf den Bildschirm holen.

`SHORTCHAR` möglich ist, sofern es sich um ein Zeichen des Latin1-Zeichensatzes handelt, und die Zeile `Char2 := SChar1` zeigt, daß die Zuweisung von `SHORTCHAR`-Werten an `CHAR`-Variablen wegen der Aufwärtskompatibilität der Typen bei Zeichen ohne Verwendung des Operators `LONG` durchgeführt werden kann.

In der nächsten Zeile `Index := ORD(Char1) + 10` lernen Sie mit `ORD` einen weiteren Component Pascal Operator kennen. `ORD` wandelt Argumente der Typen `CHAR` oder `SHORTCHAR` in der Weise in Werte des Typs `INTEGER` um, daß sich als Ergebnis der Umwandlung der Unicode- bzw. Latin1-Zahlenwert des Parameters ergibt. Der Latin1-Zahlencode des Zeichens `A` ist `65`, zu dieser Zahl läßt sich die Zahl `10` addieren, das Ergebnis, die Zahl `75` kann in der `INTEGER`-Variablen `Index` gespeichert werden. Die Umkehrung des Vorgangs sehen Sie in der Zeile `SKatz := SHORT(CHR(Index))`. Der Operator `CHR` verwandelt einen `INTEGER`-Wert in den entsprechenden `CHAR`-Wert, der anschließend - gegebenenfalls unter Verwendung des `SHORT`-Operators - einer Zeichenvariablen zugewiesen werden kann.

Die erste der Zuweisungen des letzten Programmblocks (die Ausgabeanweisungen übergehe ich wieder) zeigt Ihnen, daß ebenso wie `INTEGER`-Werte auch Zeichenkonstanten im Hexadezimalcode eingegeben werden können. Diese müssen zur Unterscheidbarkeit für den Compiler anders geschrieben werden als `INTEGER`-Hexadezimalwerte, ein Zeichen-Hexadezimalwert beginnt mit der Ziffer Null und endet mit dem Großbuchstaben `X`.

Als letzte Neuigkeit dieses Programms finden Sie in der Zeile `Char2 := CAP(Char1)` den Operator `CAP`, dessen Schreibweise vom englischen Wort "capitalize" abgeleitet ist. Er dient dazu, Kleinbuchstaben in ihr großgeschriebenes Äquivalent umzuwandeln⁷.

Schließlich finden sie eine absurd aussehende Mischung von Mathematik zusammen mit `CHR` und `ORD`-Operatoren, die Ihnen verdeutlichen soll, auf welche Weisen Zeichen in Component Pascal verändert werden können. Sie sollten diese Zeile selbst auswerten, wenn Sie die bisherigen Abschnitte richtig verstanden haben, muß das Ergebnis Ihrer Bemühungen der Buchstabe "A" sein.

⁷ Bei der Verwendung von `CAP` ist Vorsicht geboten. Zwar verwandelt `CAP` tatsächlich jeden Buchstaben in einen Großbuchstaben. `CAP` wirkt allerdings auch auf eine Reihe weiterer Zeichen des Latin1 Zeichensatzes, wobei unvorhergesehene Effekte entstehen können. Beispielsweise ergibt `CAP(+)` das Zeichen `x`, aus dem Divisionsoperator wird der "Großbuchstabe" Multiplikationsoperator. Eine bequeme Alternative zu `CAP` finden Sie im Anhang im Modul `TuStd`. Dieses stellt die Prozedur `TuStd.Cap` zur Verfügung, die tatsächlich nur Kleinbuchstaben in Großbuchstaben verwandelt, alle anderen Zeichen bleiben unverändert.