

## KAPITEL 7

## FELDER - Der Datentyp ARRAY

## 7.1. Eindimensionale Felder

Mit den Mengen, die Sie im letzten Kapitel kennengelernt haben, sind Sie dem ersten strukturierten Component Pascal Datentyp begegnet. Strukturierte Datentypen werden oft auch zusammengesetzte Datentypen genannt. Zusammengesetzt bedeutet dabei, daß in Variablen derartiger Typen mehr als ein einzelnes Element desselben Typs (oder auch verschiedener Typen, wie Sie später sehen werden) zusammengefaßt sein können.

Ein weiterer strukturierter Datentyp ist das "Feld", dessen Component Pascal Bezeichnung *ARRAY* lautet. Damit ist im Englischen eine Anordnung gemeint, solche *ARRAY*s werden deshalb im Deutschen manchmal auch "Reihungen" genannt. Die Bedeutung von Feldtypen liegt darin, daß es in den unterschiedlichsten Situationen möglich ist, in *ARRAY*-Variablen verschiedene Größen mit gleichartigen Eigenschaften, zum Beispiel dem einheitlichen Datentyp *CHAR*, zusammenzufassen.

Sie könnten natürlich wie bisher einzeln Variablen des Typs *CHAR* für solche Gegenstände einführen und in jeder Variablen ein Zeichen speichern; eine einfachere und für die Erstellung zusammenhängender Texte sinnvollere Form stellt in diesem Fall jedoch eine Feldvariable dar, in der die Daten unter einem Oberbegriff, dem gemeinsamen Variablennamen, zusammengefaßt werden.

Allgemein lautet die Deklaration eines Feldtyps *TYPE ARRAY <Indexbereich> OF <Basistyp>*. *<Indexbereich>* ist eine Konstante des Typs *INTEGER* und gibt die Länge des Feldes, das heißt die Anzahl der Basisvariablen, an. Mit *<Basistyp>* ist der einheitliche Typ aller Basisvariablen gemeint, dieser Typ kann jeder der in Component Pascal möglichen einfachen oder zusammengesetzten Typen sein. Zwischen der Anzahl der Basisvariablen und dem Indexbereich eines Feldes besteht ein enger Zusammenhang, der Indexbereich eines Feldes fängt stets bei Null an und muß folglich bei Anzahl - 1 enden, er muß also eine positive ganze Zahl sein.

Die Deklaration benutzerdefinierter Typen kann in zwei unterschiedlichen Formen geschehen, entweder als explizit benannter oder als anonymer strukturierter Datentyp. Sie finden beide Varianten in dem Programm *Feld.odc*. Im Deklarationsteil des Moduls *TutFeld* stehen noch vor der Deklaration der Prozedur *Start* die Zeilen *TYPE Feld1Typ = ARRAY 13 OF INTEGER* in denen die Deklaration des expliziten Datentyps *Feld1Typ* erfolgt. Bei vom Benutzer deklarierten expliziten Typen ist zu berücksichtigen, daß sie nur in ihrem Sichtbarkeitsbereich verwendet werden können, außerhalb dieses Bereiches lassen sich ver-

ständlicherweise nur die vordefinierten, dem Compiler generell bekannten Typen verwenden, *Feld1Typ* ist also nur im Modul *TutFeld* verwendbar.

Die Prozedur *Start* des Moduls *TutFeld* enthält in den Variablendeklarationen nach der Ihnen bekannten Deklaration der Variablen *Index: BYTE* die Variable *Feld1* mit dem expliziten Typ *Feld1Typ* sowie mehrere Feldvariablen, deren (Feld-)Typen anonym sind. Die Typdeklaration ist bei einem anonym deklarierten Feld strukturell identisch mit dem rechts vom Gleichheitszeichen stehenden Teil des explizit deklarierten Typs *Feld1Typ*, Sie sehen dies an den Deklarationen der Variablen *Feld2*, *Feld3* und *Feld4*, die sich von der Deklaration des Typs *Feld1Typ* nur in den Indexbereichen und den Basisvariablentypen unterscheiden.

Wie Sie an den Variablendeklarationen sehen, kann der Indexbereich eines Feldes erheblich variieren. Er kann sehr klein sein, im Extremfall wie in der Deklaration *Feld4: ARRAY 1 OF REAL* sogar nur eine einzelne Basisvariable enthalten, was allerdings nicht besonders sinnvoll ist, er kann andererseits eine beachtliche Größe haben. Bei Deklarationen der Art *Feld3: ARRAY 1000000 OF CHAR* müssen Sie den Speicherbedarf einer Variablen dieses Typs berücksichtigen, *Feld3* belegt, wie Sie leicht errechnen können, immerhin 2 Megabyte im Arbeitsspeicher.

Die einzelnen Basisvariablen eines Feldes können individuell adressiert werden, was dadurch geschieht, daß dem Namen der Feldvariablen der Index der entsprechenden Basisvariablen in eckigen Klammern angefügt wird. Sie sehen dies beispielsweise in den Zeilen *Feld1[Index] := Index \* 3* und *Feld1[5] := -55*. Jede Basisvariable eines Feldes hat alle Eigenschaften einer einzeln deklarierten Variablen des Basistyps, ihr können Werte zugewiesen und Werte können verändert oder auf dem Bildschirm angezeigt werden, wobei selbstverständlich die dem jeweiligen Variablentyp entsprechenden Formatierungsverfahren oder Ein- und Ausgabeanweisungen benutzt werden müssen. Außerdem sehen Sie an der Zeile *Feld1[4 \* 3] := -1212*, daß Feldindizes arithmetische Ausdrücke sein können; diese Ausdrücke werden vom Compiler ebenso wie z.B. in Zuweisungen zuerst berechnet, das Ergebnis wird anschließend als Feldindex ausgewertet.

## 7.2. Mehrdimensionale Felder

Während die eindimensionalen Felder, die Sie im letzten Abschnitt kennengelernt haben, durch den Ausdruck *Reihung* vielleicht richtig beschrieben sind, lernen Sie mit den mehrdimensionalen Feldern in diesem Abschnitt "richtige" Felder kennen. Ein Beispiel eines zweidimensionalen Feldes, das Ihnen bekannt sein wird, ist ein Schachbrett mit seinen 8 x 8 Grundfeldern. Jedes einzelne dieser Felder ist eindeutig bestimmbar durch die Angabe seiner Position in einer Reihe und einer Spalte. Benennt man, wie bei einem Schachbrett üblich, die Reihen mit den Zahlen "1" bis "8", die Spalten mit den Buchstaben "A" bis

"H", so hat z.B. der weiße König die Anfangsposition "1E". In gleicher Weise trägt jedes der Einzelfelder die Bezeichnung, die sich aus der Zusammensetzung der Reihen- und Spaltenposition ergibt. Nennt man Spalten und Reihen einheitlich "Reihungen", so kann man ein solches zweidimensionales Feld, wie das Schachbrett, als "Reihung von Reihung" bezeichnen. Entsprechend lautet in Component Pascal die Typbezeichnung für ein zweidimensionales Feld *ARRAY <Indexbereich 1> OF ARRAY <Indexbereich 2> OF <Basistyp>*.

Sie finden in dem Programm *Felder.odc* die Deklaration einer nach diesem Muster aufgebauten Variablen *Schachbrett: ARRAY Zeilen OF ARRAY Spalten OF INTEGER*, wobei die Konstanten *Zeilen* und *Spalten* jeweils mit dem Wert "8" deklariert wurden. In *Schachbrett* sind also insgesamt 64 Einzelvariablen des Typs *INTEGER* zusammengefaßt. Ginge es nur darum, eine Feldvariable mit 64 Elementen zu schaffen, wäre natürlich eine Deklaration wie *Schachbrett: ARRAY 64 OF INTEGER* einfacher gewesen. Eine zweidimensionale Variable bietet jedoch bei der Handhabung der Einzelwerte einen Vorteil, der Zugriff auf ein bestimmtes Einzelfeld ist verständlicher, ich werde in der Erläuterung des Anweisungsteils von *Start* genauer darauf eingehen.

Von ähnlichem Typ wie die Variable *Schachbrett* ist die Variable *FeldA*. Die Typdeklaration unterscheidet sich allerdings etwas von der vorangegangenen, es gibt bei mehrdimensionalen Feldvariablen eine abkürzende Deklarationsform, die für die Variablen *FeldA* und *FeldC* verwendet worden ist. Diese erspart Ihnen und mir einige Schreiarbeit und lautet allgemein *ARRAY <Indexbereich 1>, <Indexbereich 2>, ... OF <Basistyp>*; die Wiederholung der Wörter *ARRAY OF* kann entfallen, es genügt statt dessen, die durch Komma(ta) getrennten Indexbereiche hintereinandergereiht aufzuführen. Abhängig von der Anzahl der Teilindexbereiche spricht man von ein-, zwei-, drei-, ... dimensional Feldern.

Eine etwas andere Art der Deklaration eines mehrdimensionalen Feldes stellt die Deklaration *FeldB: ARRAY 8 OF Vertikale* vor, bei der ein - scheinbar - eindimensionales Feld deklariert wird, das tatsächlich jedoch zweidimensional ist und in seiner Größe und Struktur den Variablen *FeldA* und *Schachbrett* entspricht, da *Vertikale* nicht eine Variable oder eine Konstante darstellt, sondern einen benannten Typ, der im Deklarationsteil der Prozedur *Start* eingeführt wird. Üblicherweise würde man eine Typdeklaration wie im vorigen Programm in den Modulrumpf plazieren, damit der Typ bei mehreren im Modul existierenden Prozeduren global ist; hier soll gezeigt werden, daß Typdeklarationen wie andere Deklarationen lokal zu einer Prozedur sein können.

Mit der letzten deklarierten, in Kommentarklammern stehenden Variablen *FeldC* finden Sie ein Feld, das Sie zur Belastungsprobe Ihres Computers verwenden können. Entfernen Sie die Kommentarklammern und lassen Sie anschließend das Modul kompilieren. Sollte Ihr Rechner keine Fehlermeldung ausgeben, kann ich Sie zu dem Gerät beglückwünschen, es hat offensichtlich die Fähigkeit, einen Speicherraum von 64 Gigabyte zu adressieren.

Will man eine zweidimensionale Feldvariable initialisieren, d.h. allen ihrer Elemente Werte zuweisen, bietet sich bei mehrdimensionalen Feldern wie im eindimensionalen Fall die *FOR*-Schleife an, allerdings wird dafür eine der Dimension des Feldes entsprechende Zahl von Schleifen gebraucht, wobei es nötig ist, die für jeden Indexbereich gesonderten Schleifen zu schachteln, Sie sehen das Verfahren im Anweisungsteil der Prozedur *Start* einmal für die Initialisierung der Variablen *Schachbrett* und ein weiteres Mal bei den Ausgabeanweisungen für *Schachbrett* und *FeldA* durchgeführt.

Sie sollten sich das Zusammenwirken der Schleifen in einer solchen Schachtelung genau klarmachen, da dieses ein häufig verwendbares Verfahren ist. Bei der Initialisierung von *Schachbrett* wird zu Beginn der äußeren, ersten *FOR*-Schleife die Kontrollvariable *Zeile* auf den Wert "0" gesetzt. Mit diesem Wert wird die zweite, innere *FOR*-Schleife aufgerufen, deren Kontrollvariable *Spalte* ebenfalls auf "0", den Wert der Konstanten *A*, gesetzt wird, damit ist der erste Feld(doppel)index vollständig bestimmt, dem Feldelement *Schachbrett*[0, *A*] kann jetzt der Wert "11" zugewiesen werden. Die Zuweisung erfolgt wie bei eindimensionalen Feldern durch Nennung des Variablenamens zusammen mit dem in eckige Klammern gesetzten Bezeichner des gewünschten Feldelementes, wobei der Bezeichner in diesem Fall entsprechend der Felddimension aus zwei durch Komma getrennten Einzelindizes besteht. Beachten Sie bitte, daß es bei der Indizierung auf die Reihenfolge der Einzelindizes ankommt; dies scheint vielleicht bei der Variablen *Schachbrett* selbstverständlich, aber es ist wichtig, sich klar zu machen, daß auch bei einer Variablen wie *FeldA* mit *FeldA*[3, 4] ein anderes Feldelement adressiert wird als mit dem Bezeichner *FeldA*[4, 3], es kommt hierbei wesentlich auf die Reihenfolge der Indizes an.

Wichtig bei der weiteren Abarbeitung der *FOR*-Schleifen ist, daß nach der Wertzuweisung an das Feldelement *Schachbrett*[0, *A*] bei unverändertem Wert von *Zeile* ("0") die innere *FOR*-Schleife für alle weiteren Werte von *Spalte* ("B" bis "H") abgearbeitet wird, erst danach wird *Zeile* in der äußeren *FOR*-Schleife auf "1" gesetzt. Mit diesem Wert wird die innere Schleife erneut vollständig abgearbeitet und dies setzt sich fort, bis die äußere und die innere *FOR*-Schleife mit dem letzten Wert von *Zeile* ("7") abgearbeitet sind.

Bildlich gesprochen wird also in der ersten Zeile des Schachbretts von links nach rechts jeder Platz gefüllt, anschließend geschieht dies in der zweiten Zeile, dann in der dritten, bis allen Feldelementen ein Wert zugewiesen ist. Man nennt bei zweidimensionalen Feldern deshalb auch den ersten Index den Zeilenindex, der zweite Index wird entsprechend Spaltenindex genannt.

Auf zwei Einzelheiten möchte ich Sie noch aufmerksam machen. Wie bereits erwähnt, können Feldindizes implizit angegeben werden, d.h. ihr Wert kann durch einen vom Compiler auswertbaren Ausdruck bestimmt sein. In der Zeile *Schachbrett*[*Schachbrett*[0, *A*] - 10, *C*] := - 23 wird der erste Index der Feldvariablen *Schachbrett* nicht durch eine Zahl, sondern durch den Ausdruck *Schachbrett*[0, *A*] - 10 bestimmt, der den Wert "1" liefert. Ein solcher Indexwert kann offensichtlich ebensogut als Wert eines Ausdrucks ermittelt werden, der von unterschiedlichen Programmbedingungen abhängt; dementsprechend sind unter-

schiedliche Indexwerte und damit unterschiedliche Einzelvariablen des Feldes programmgesteuert adressierbar.

Als zweites möchte ich Sie auf die Ausgabeanweisungen für die Variable *FeldA* hinweisen. Diese zeigen Ihnen, daß bei entsprechender Korrektur die Werte der Kontrollvariablen der *FOR*-Schleifen aus anderen Zahlintervallen sein können als die Feldindizes. Außerdem sehen Sie eine alternative Schreibmöglichkeit für mehrfache Feldindizes, der Ausdruck *FeldA*[*Zeile* - 1][*Spalte* + 3] mit zwei Einzelindizes entspricht der bisherigen Schreibweise *FeldA*[*Zeile* - 1, *Spalte* + 3], bei der ein durch Komma separierter Doppelindex verwendet wird.

### 7.3. Typbindung bei Feldern

Für den Umgang mit Feldvariablen und die ihnen zugeordneten Typen gibt es einige wichtige Regeln, die Sie im Programm *FeldTyp.odc* finden. Ignorieren Sie für den Augenblick die Prozedur *Ausgeben*. Die im Modulrumpf und in der Prozedur *Start* stehenden Typ- und Variablendeklarationen enthalten keine Neuigkeiten, ebensowenig die erste *FOR*-Schleife zur Initialisierung der Variablen *Pferd1* und *Hund1* im Anweisungsteil von *Start*. Den anschließenden beiden Wertzuweisungen *Pferd2* := *Pferd1* und *Hund2* := *Hund1* können Sie entnehmen, daß Sie ebenso wie bei Variablen einfacher, vordefinierter Typen durch Zuweisungen die Inhalte einer Feldvariablen in eine Variable desselben Typs kopieren können.

Die den beiden Ausgabeanweisungen für *Pferd2* und *Hund2* in Kommentarklammern folgenden Zuweisungen *Pferd3* := *Pferd1* und *KatzA* := *Katz1* sind jedoch nicht möglich, denn die jeweiligen Typen der rechts und links des Zuweisungszeichens stehenden Variablen sind für den Compiler verschieden. Im Fall von *Pferd1* und *Pferd3* liegt dies daran, daß es sich um separate Variablendeklarationen mit anonymen Typen handelt, die vom Compiler, auch wenn sie für den Leser inhaltlich übereinstimmen, als unabhängig aufgefaßt werden und deshalb nicht zuweisungskompatibel sind. Typidentität existiert für den Compiler bei anonymen Typen lediglich für Variablen, die wie *Pferd1* und *Pferd2* in derselben Deklaration vereinbart worden sind.

Anders liegt die Sache bei expliziten Typdeklarationen. *Hund1* und *Hund2* sind zwar separat deklariert worden, wegen des expliziten Typs *GanzzahlFeld*, dem beide angehören, ist ihre Kompatibilität für den Compiler jedoch geklärt. Umgekehrt gilt für die Variablen *KatzA* und *Katz1* mit den zwar strukturell identischen, aber formal unterschiedlichen Typen *Acker* und *SpielFeld*, daß sie wegen ihrer verschiedenen Typen nicht kompatibel sind. Wiederum anders liegt der Fall bei den in der folgenden Zeile zu findenden Variablen *Katz1* und *KatzB*. Zwar haben beide mit *SpielFeld* und *Beet* ebenso wie *KatzA* und *Katz1* nominell unterschiedliche Typen, auf Grund der Typdeklaration *TYPE Beet = SpielFeld* sind Variablen dieser beiden Typen für den Compiler jedoch zuweisungskompatibel.

Die folgende, in Kommentarklammern stehende Abfrage *IF Hund1 = Hund2 THEN END* weist Sie daraufhin, daß für Felder außer der Zuweisung keinerlei vordefinierte Operationen im Compiler existieren; wenn Sie die Kommentarklammern entfernen, werden Sie beim Kompilieren des Programms eine Fehlermeldung erhalten, denn trotz identischem Typ ist ein Vergleich von Feldvariablen (sowie aller zusammengesetzten Datentypen mit Ausnahme eines noch zu besprechenden Falls, des Typs *ARRAY OF CHAR*) mit Hilfe der bekannten Vergleichsoperatoren anders als bei den einfachen, vordefinierten Typen nur dann möglich, wenn Sie diese Operationen selbst programmieren. Dagegen können Sie zwischen einzelnen Elementen von Feldern Vergleiche durchführen, sofern es sich bei dem Basistyp des Feldes um einen einfachen - vordeklarierten - Typ handelt. In diesem Fall sind alle Vergleichsoperatoren anwendbar, die für den jeweiligen Basistyp definiert sind.

Wie schon im vorigen Programm sind auch bei den hier vorliegenden Feldern doppelte *FOR*-Schleifen für die Wertzuweisungen an die Variablen *KatzI* und *KatzG* nötig, da es sich wieder um zweidimensionale Felder handelt. Die erste und dritte dieser am Ende von *Start* stehenden Schleifen enthalten einen neuen Operator, *LEN*. Mit dieser Component Pascal Standardprozedur läßt sich bequem die Länge eines Feldes ermitteln. *LEN* erwartet zwei Parameter, als ersten eine Feldvariable wie *KatzI*, als zweiten die Angabe der Dimension des (Sub-) *ARRAYS*, für den die Länge, das heißt die Anzahl der Basisvariablen, ermittelt werden soll. Der von *LEN* zurückgegebene Wert ist vom Typ *INTEGER* und gleich dem (Sub-) Indexbereich des Feldes. Da die Werte von Indexbereichen immer von Null bis Indexbereich minus Eins laufen, müssen Sie darauf achten, daß Sie bei der Verwendung von *LEN* (beispielsweise wie hier in den *FOR*-Schleifen) immer eine Einheit abziehen, anderenfalls erhalten Sie zwar keinen Kompilationsfehler, dafür aber bei der Ausführung des Programms einen der besonders ärgerlichen Laufzeitfehler (Sie können dies durch Änderung der Zeile *FOR Index := 0 TO LEN(KatzI, 0) - 1 DO* in *FOR Index := 0 TO LEN(KatzI, 0) DO* ausprobieren).

In der letzten doppelten *FOR*-Schleife finden Sie eine vereinfachte Version von *LEN*, in der der zweite Parameterwert, die Dimension, fehlt. Bei dieser Version setzt der Compiler von sich aus den Wert Null als zweiten Parameterwert ein, die vereinfachte Version kann also immer dann verwendet werden, wenn ein Feld entweder eindimensional ist, oder die Länge der nullten (äußersten) Dimension eines mehrdimensionalen Feldes bestimmt werden soll.

Zwei weitere für die Programmierarbeit wesentliche Tatsachen finden Sie in der bisher ignorierten Prozedur *Ausgeben*, die in der letzten Zeile von *Start* aufgerufen wird. *Ausgeben* erhält als Parameter die Variable *KatzG* übergeben, deren Typ *Garten* ein Feldtyp, also ein strukturierter Datentyp ist. Der bei der Deklaration von *Ausgeben* in Kommentarklammern stehende Prozedurkopf *PROCEDURE Ausgeben (IN GartenFeld: ARRAY 11, 8 OF BOOLEAN)* enthält den formalen Parameter *GartenFeld* mit dem anonymen Typ *ARRAY 11, 8 OF BOOLEAN*, der strukturell mit dem Typ *Garten* übereinstimmt. Zwar erhielten Sie keine Fehlermeldung vom Compiler, falls Sie die derart deklarierte Prozedur kompilieren ließen, aber for-

male Parameter strukturierter Datentypen können nicht anonym sein, da keine Parameterübergabe möglich ist. Der Compiler führt eine Parameterübergabe nur aus, wenn die Typen von formalem und aktuellem Parameter explizit gleich sind, ebenso wie die Zuweisung von Variablenwerten in der Prozedur *Start* nur bei explizit gleichen Typen möglich war. Die Prozedur *Ausgeben* kann also nur aufgerufen werden, wenn ihr formaler Parameter einen explizit deklarierten Typ hat, wie Sie es in dem zweiten Prozedurkopf von *Ausgeben* sehen.

Eine andere Neuigkeit stellt das Schlüsselwort *IN* dar, das Sie in der Signatur von *Ausgeben* finden. Wie schon früher erwähnt, sollten Sie Prozedurparameter aus Sicherheitsgründen soweit möglich als *VAL*-Parameter deklarieren. Nun können, wie Ihnen in den beiden vorigen Programmen klar geworden ist, Variablen strukturierter Datentypen einen erheblichen Umfang haben. Da ein *VAL*-Parameter eine Kopie des übergebenen aktuellen Parameters erzeugt, kann es bei Parametern strukturierter Datentypen schnell zu einem Arbeitsspeicherproblem kommen, außerdem ist die Erstellung der Kopie zeitlich aufwendiger, als die Übergabe der Adresse des Originals. In vielen Programmiersprachen besteht daher bei strukturierten Parametern der "Ausweg" darin, einen *VAR*-Parameter zu verwenden und das damit verbundene erhöhte Risiko in Kauf zu nehmen. Component Pascal bietet statt dessen eine sehr viel elegantere Möglichkeit, ein strukturierter formaler Prozedurparameter kann als *IN*-Parameter eingeführt werden. *IN*-Parameter sind spezielle *VAR*-Parameter, sie werden als Adressen übergeben, können aber innerhalb der Prozedur nur gelesen, nicht jedoch verändert werden (read only parameter). Damit offerieren *IN*-Parameter die gleiche Sicherheit wie *VAL*-Parameter, ohne jedoch die mit ihnen verbundenen Probleme aufzuweisen<sup>9</sup>.

In der zweiten, inneren *FOR*-Schleife der Prozedur *Ausgeben* sehen Sie eine letzte Neuigkeit, die Standardprozedur *LEN* akzeptiert als ersten Parameter außer einer Feldvariablen auch einen Feldtyp; diese Eigenschaft ist in Fällen nützlich, in denen die Länge des Feldtyps benötigt wird, aber keine Variable des Typs deklariert ist.

#### 7.4. Offene Feldparameter

Das Programm *FeldProc.odc* macht Sie mit einer sehr nützlichen Möglichkeit beim Umgang mit Feldern bekannt, den offenen Feldparametern. Felder sind bekanntlich Datenstrukturen, die feste Längen besitzen. Als Programmierer gerät man daher immer wieder in das Dilemma, bei der Deklaration von Feldvariablen zwischen zwei Übeln entscheiden zu müssen; auf der einen Seite soll - beispielsweise zur Aufnahme einer Zeichenkette - ein Feld eine ausreichende Größe haben, man möchte sich für diesen Zweck

<sup>9</sup> Zu Prozedurparametern s. Anhang B.3.2.3.

am liebsten eine Variable des Typs *ARRAY 1000000 OF CHAR* einrichten. Dabei stößt man jedoch möglicherweise an die realen Kapazitätsgrenzen des Computerspeichers, der vielleicht eine Variable dieser Größe nicht verarbeiten mag, außerdem existieren neben dieser einen Variablen auch noch andere, konkurrierende, die ebenfalls Speicher beanspruchen. Macht man aus diesen Gründen den Indexbereich des Feldes jedoch klein, gerät man schnell in die Gefahr, eine der Praxis wiederum nicht gerecht werdende Größe gewählt zu haben. Die notwendigen Kompromisse sind eine Frage der Erfahrung und der Einschätzung des Anwendungsbereichs eines Programms, dennoch bleiben sie stets Kompromisse. Vollends unmöglich wird eine realistische Abwägung jedoch dann, wenn es sich um das Erstellen universeller Prozeduren handelt, die in nicht vorhersehbaren Zusammenhängen verwendet werden sollen. Zuviel Speicherverbrauch eines Programms ärgert den Anwender ebenso wie eine zu starke Einschränkung seiner Möglichkeiten.

Um das Problem zu entschärfen, sind die offenen Feldparameter für Prozeduren eingeführt worden. Sehen Sie sich zum Verständnis im Programm *FeldProc.odc* zuerst die Deklaration der Prozedur *Feldprozedur* an. Diese hat einen formalen Parameter des global deklarierten Typs *Feld1*, das heißt des Typs *ARRAY 6 OF INTEGER*. Hier handelt es sich um einen Typ, wie Sie ihn kennen, der aktuelle Parameter kann maximal sechs Ganzzahlwerte enthalten. Dagegen finden Sie im Kopf der Prozedur *OffenFeldProzedur* (*IN Tee: ARRAY OF INTEGER*) eine neue, im Zusammenhang mit Prozeduren verwendbare Form einer Felddeklaration, ein Feld variabler Länge. Dies ist deshalb möglich, weil für formale Parameter von Prozeduren erst zur Laufzeit des Programms - bei ihrem Aufruf - die in ihnen deklarierten Datenstrukturen eingerichtet werden. Die tatsächliche Länge des dem formalen Parameter *Tee* übergebenen Feldes muß also erst im Moment der Übergabe, zur Laufzeit also, bekannt sein, das System reserviert daraufhin den aktuell benötigten Speicher. Die Prozedur ist auf diese Weise flexibel, sie kann stets genau so viel Speicher zur Verfügung gestellt bekommen, wie zur Bearbeitung des Problems erforderlich ist.

Die Prozedur *OffenFelderProzedur* (*IN Soda: ARRAY OF ARRAY OF INTEGER*) zeigt Ihnen, daß nicht nur eindimensionale offene Parameter möglich sind, sondern auch mehrdimensionale Felder als offene Prozedurparameter existieren, wobei die verschiedenen Dimensionen eines mehrdimensionalen Feldes ebenso wie bei fest deklarierten mehrdimensionalen Feldern einzeln verarbeitet werden können.

Auf zwei Besonderheiten möchte ich Sie aufmerksam machen. Im Gegensatz zu dem, was bei dem Programm *FeldTyp.odc* über Typen strukturierter Parameter einer Prozedur gesagt wurde, sind bei offenen Feldparametern anonyme Typdeklarationen nicht nur möglich, sondern sogar unbedingt nötig, da ein explizit deklarierter Typ, wie beispielsweise *Feld1*, notwendigerweise eine feste Länge hat. Eine weitere Besonderheit sehen Sie in den *FOR*-Schleifen der Prozeduren *OffenFeldProzedur* und *OffenFelderProzedur*. Innerhalb einer Prozedur mit offenen Feldparametern ist die tatsächliche Größe eines aktuellen Parameters wegen dessen nicht vorab bestimmbarer Länge selbstverständlich unbekannt. Wollen Sie wie in den *FOR*-Schleifen die Längen der einzelnen Dimensionen der Prozedurparameter verwenden, können diese nur mit der Standardfunktion *LEN* ermittelt werden. Beachten Sie bitte, daß *LEN* einen Wert des Typs *INTEGER*

zurückgibt, weshalb in der inneren *FOR*-Schleife der *OffenFelderProzedur* dieser Wert mittels *SHORT* dem Typ der Kontrollvariablen *Zähler2* angepaßt werden muß.

Die Kommandoprozedur *Start* enthält keine großen Überraschungen, das interessanteste Detail findet sich in der auskommentierten Zeile *FeldProzedur(FeldEins)*, in der der Versuch gemacht wird, der *FeldProzedur* mit dem Parameter *FeldEins* eine unzulässige Variable zu übergeben, der Compiler quittiert diesen Versuch mit einer entsprechenden Fehlermeldung, da die Typen von formalem und aktuellem Parameter trotz der faktischen Gleichheit nicht übereinstimmen. Dagegen kann die *OffenFeldProzedur* wegen des offenen Parameters *Tee* sowohl die Variable *FeldNull* als auch die Variable *FeldEins* problemlos verarbeiten.

### 7.5. Zeichenketten

Ein bereits mehrfach erwähnter Datentyp soll mit dem Programm *String.odc* jetzt seine Erklärung finden, der Feldtyp *ARRAY OF CHAR*. Übergehen Sie für den Moment die beiden Prozeduren *Schreibe* und *Schreib*, die zur Ausgabe der Zeichenketten des Programms dienen und wenden Sie sich der Prozedur *Start* zu. Diese enthält mit *String1*, *String2* und *String3* die Deklarationen dreier Feldtypen des Basistyps *CHAR*. Außerdem werden sechs diesen Typen zugeordnete Variablen deklariert sowie eine siebte, *Titel*, mit dem anonymen Typ *ARRAY 55 OF CHAR*.

Im Anweisungsteil der Prozedur wird der Variablen *Titel* als erstes ein Wert zugewiesen, der wie bei Zeichen und Zeichenketten nötig, zwischen Anführungszeichen steht. Der Ausgabeanweisung für den "Titel" folgt ein Kommentar, der eine wesentliche Tatsache enthält. In Component Pascal werden alle Zeichenketten mit dem Zeichen *NUL* (Zeichen *OX*) abgeschlossen, die Zeichenkettenvariable *Titel: ARRAY 55 OF CHAR* kann also maximal 54 Nutzzeichen aufnehmen, das letzte Zeichen ist der "string terminator" *OX*.

Als nächstes werden der Variablen *Kette1* die leere Zeichenkette "", der Variablen *Kette2* die Zeichenkette "ABCDE" sowie der Variablen *Schnur* die Zeichenkette "12" zugewiesen und jeweils durch die Prozedur *Schreibe* ausgegeben. Der Kopf dieser Prozedur enthält mit *Sache* einen offenen formalen *IN*-Parameter des Typs *ARRAY OF CHAR*, dem entsprechend den Erläuterungen des vorigen Abschnitts Zeichenkettenvariablen unterschiedlicher Längen übergeben werden können.

Die Prozedur gibt nun nicht nur die Werte des aktuellen Parameters aus, sondern stellt sowohl die jeweiligen Zeichenketten als auch die Unicodenummern der einzelnen Zeichen dar. Die Analyse der Prozedur sollte Ihnen inzwischen keine Schwierigkeiten mehr machen, ich beschränke mich deshalb auf die Erläuterung einer für Sie neuen Einzelheit. Sie sehen innerhalb der *FOR*-Schleifen in den Ausdrücken *LEN(Sache\$)* ein *\$*-Zeichen am Ende des Parameters *Sache*. Dieses Zeichen ist ein reserviertes Component

Pascal Zeichen, dessen Wirkung Sie am einfachsten an der zweiten *FOR*-Schleife der Prozedur *Schreibe* und den anschließenden Programmzeilen verstehen können.

Sie wissen, daß *LEN* die Länge des jeweiligen Parametertyps an die aufrufende Stelle zurückgibt, der Ausdruck *LEN(Kette2)* liefert folglich den Wert "10", die Länge des Typs von *Kette2*, unabhängig von der tatsächlichen Länge der in *Kette2* stehenden Zeichenkette. Dagegen liefert *LEN(Kette2\$)* den Wert "5", die tatsächliche Zeichenzahl des aktuell in der Variablen *Kette2* stehenden Wertes, wenn dieser Wert die Zeichenkette "ABCDE" ist. Sie können diese Tatsachen den Bildschirmausgaben der einzelnen Zeichenketten entnehmen, darüber hinaus sehen Sie an der Wiedergabe der Unicodenummern, daß tatsächlich in den Variablen als letztes Zeichen der string-terminator *OX* gespeichert ist.

Die in der Prozedur *Start* in Kommentarklammern stehende Zeile *Kette1 := Kette2* und die folgende Zeile *Kette1 := Kette2\$* weisen Sie auf eine weitere Verwendungsmöglichkeit des *\$*-Zeichens hin. Die erste Zeile wird vom Compiler wegen der unterschiedlichen Typen der beiden Variablen als unzulässige Zuweisung beanstandet. Die zweite Zeile stellt dagegen eigentlich keine Zuweisung, sondern einen Kopiervorgang dar, das Programm liest Zeichen für Zeichen den aktuellen Wert von *Kette2* und schreibt diesen anschließend in die Variable *Kette1*, die "Zuweisung" ist also trotz unterschiedlichen Typen der Variablen möglich.

Eine weitere Neuigkeit finden Sie in der Zeile *IF Perlen1 <= Perlen2 THEN*. Im Abschnitt 7.3 haben Sie erfahren, daß für strukturierte Variablen im Compiler keine Vergleichsoperationen existieren. Für Zeichenketten gibt eine sinnvolle Ausnahme von dieser Regel, Sie können auf Zeichenketten alle in Component Pascal definierten Vergleichsoperatoren (<, >, <=, >=, = und #) anwenden. Allerdings müssen Sie berücksichtigen, daß die Vergleiche anhand der Unicodenummern der Zeichen durchgeführt werden, was bedeutet, daß alle Großbuchstaben "kleiner" sind als die entsprechenden Kleinbuchstaben, außerdem werden die deutschen Sonderbuchstaben Ä, Ö, Ü, ä, ö, ü, ß entsprechend ihren Unicodenummern als "sehr groß" eingeordnet.

Eine andere Neuigkeit, die Sie in der Zeile *Kette1 := Perlen1 + Perlen2 + "e"* illustriert sehen, liegt in der Verwendung des "+" Zeichens im Zusammenhang mit Zeichenketten. Sie können mit diesem Operator Zeichenketten "addieren", die einzelnen "Summanden" werden in der Reihenfolge der "Addition" zusammengefügt (Concatenation). Das Ergebnis kann einer Variablen wie *Kette1* oder einer Prozedur wie *Schreibe* als Parameter zugewiesen werden.

Eine letzte Neuigkeit gibt es im Zusammenhang mit den beiden Anweisungen *Schreib(Kette1)* und *Schreib(Kette1\$)*. Die beiden Prozeduren *Schreibe* und *Schreib* sind inhaltlich identisch, sie unterscheiden sich lediglich in dem jeweiligen formalen Parameter. Der erste Prozeduraufruf *Schreib(Kette1)* steht in Kommentarklammern, der Compiler beanstandet den Versuch, der Prozedur *Schreib* die Variable *Kette1* als aktuellen Parameter zu übergeben, da die Typen von aktuellem und formalem Parameter inkompatibel sind. Der Aufruf *Schreib(Kette1\$)* ist dagegen zulässig, da wegen des *\$*-Zeichens wiederum der aktuelle

Wert von *Kette1* in den formalen Parameter von *Schreib* kopiert wird, wobei die Typinkompatibilität der Parameter keine Rolle spielt. Die Anmerkungen in Kommentarklammern zeigen Ihnen, daß den beiden Prozeduren *Schreib* und *Schreibe* auch zusammengesetzte Zeichenketten als aktuelle Parameter übergeben werden können, beide Prozeduren akzeptieren die Zeichenkette *Perlen1 + Perlen2 + "e"* als Parameter. Ähnlich wie bei dem Parameter *Kette1\$* werden mit dem "+" Zeichen zusammengesetzte Zeichenketten in die formalen Parameter der Prozeduren kopiert, in diesem Fall allerdings ohne explizite Verwendung des *\$*-Zeichens.

Warum überhaupt die Prozedur *Schreib*, wenn es mit *Schreibe* doch keinerlei Probleme gibt? Ein wesentlicher Aspekt von Component Pascal ist die Fähigkeit, in Zusammenhang mit dem BlackBox System mögliche Fehler eines Programms auch zur Laufzeit, bei der Ausführung, zu erkennen. Offene Feldparameter bieten auf der einen Seite die Bequemlichkeit, jede beliebig lange Variable zu akzeptieren, auf der anderen Seite hat der Compiler bei offenen Parametern keine Möglichkeit, die tatsächliche Länge eines aktuellen Parameters zu überprüfen. Sollte ein derartiger Parameter eine Länge haben, die die Größe des verfügbaren Arbeitsspeichers überfordert, gäbe es den Ihnen wahrscheinlich nicht unbekanntem Effekt eines Systemzusammenbruchs. Mit Parametern ausreichend großer, aber fester Länge sind solche Fehler vermeidbar, denn das System kann zur Kompilationszeit prüfen, ob die tatsächliche Länge des aktuellen Parameters mit dem formalen Parameter vereinbar ist. Entfernen Sie in der Prozedur *Start* die Kommentarklammern um die Zeilen *Schreib("Eine Zeichenkettenkonstante, die länger als ... abgewiesen.")*, werden Sie beim Kompilieren die dahinter stehende Fehlermeldung erhalten, der Compiler erkennt die Zeichenkette als nicht zuweisungskompatibel.

Bei der Kompatibilitätsprüfung gibt es jedoch eine wesentliche Ausnahme. Die Anweisung *Kette3 := "Eine Zeichenkettenvariable wird, ... Zeichenkettenlänge übergeben!"* und die auskommentierte Zeile *Schreib(Kette3\$)* verdeutlichen, daß der Compiler bei Verwendung des *\$*-Operators - das gleiche gilt für die Verwendung des "+" Zeichens zur Verbindung von Zeichenketten - keine Typprüfung durchführt, bei Verwendung dieser Operatoren liegt die Einhaltung der maximal möglichen Zeichenkettenlängen aktueller Parameter ausschließlich in Ihrer Verantwortung. Wenn Sie nicht sicher sein können, daß die jeweiligen aktuellen Parameter die durch einen formalen Parameter fester Länge gesetzten Grenzen einhalten werden, müssen Sie sich bei Zeichenketten trotz den damit verbundenen Risiken für offene formale Prozedurparameter entscheiden.