de Gruyter Lehrbuch
Niemeyer · Programmieren in PASCAL

Gerhard Niemeyer

Einführung in das Programmieren in PASCAL

mit Sonderteil TURBO-PASCAL-System

Dritte, bearbeitete und erweiterte Auflage

Walter de Gruyter · Berlin · New York 1990

Dr. rer. pol. *Gerhard Niemeyer,*
o. Professor für Wirtschaftsinformatik
an der Universität Regensburg

∞ Gedruckt auf säurefreiem Papier, das die US-ANSI-Norm über Haltbarkeit erfüllt.

CIP-Titelaufnahme der Deutschen Bibliothek

Niemeyer, Gerhard:
Einführung in das Programmieren in PASCAL : mit Sonderteil TURBO-PASCAL-System / Gerhard Niemeyer. - 3., bearb. u. erw. Aufl. - Berlin ; New York : de Gruyter, 1990
(De-Gruyter-Lehrbuch)
ISBN 3-11-012726-1

Printed in Germany
Druck: Kupijai & Prochnow, Berlin.
Buchbinderische Verarbeitung: Dieter Mikolai, Berlin.

Vorbemerkungen

Seit einigen Jahren macht das Schlagwort von der Strukturierten Programmierung die Runde. Mit dieser insbesondere von E.W. Dijkstra konzipierten Methode wird gegenüber dem herkömmlichen Vorgehen ein grundsätzlich neuer Ansatz zur Problemlösung im allgemeinen und zur Programmentwicklung im besonderen aufgezeigt.

Das Neuartige besteht in einer hierarchisch-rekursiven Zerlegung eines Problems in Teilprobleme, und zwar solange, bis leicht überschaubare und lösbare Problemgrößen erreicht werden. Der bei der Zerlegung anzuwendende strenge Hierarchismus bietet die Gewähr, daß sich alle Teillösungen einer bestimmten Unterordnung zur Lösung des übergeordneten Problems zusammenfügen.

Auf die Programmentwicklung übertragen ermöglicht dieses Konzept die Untergliederung einer Datenverarbeitungsaufgabe von der allgemeinen Problemdefinition über Prozedurhierarchien bis hin zu den ausführenden Anweisungen einer Programmiersprache. Dabei sind die Prozeduren einer bestimmten Hierarchiestufe innerhalb einer Substruktur "arbeitsteilig" voneinander abhängig, da zwischen ihnen ein Datenaustausch stattfindet. Hingegen herrscht zwischen den Prozeduren verschiedener Substrukturen Unabhängigkeit, da ein Datenaustausch nur über die nächsthöhere Hierarchiestufe möglich ist.

Daraus ergeben sich insbesondere bei hochkomplexen Datenverarbeitungsaufgaben folgende Vorteile gegenüber den herkömmlichen "flow-chart"-Lösungen:

(1) Die Programmentwicklung wird wegen der größeren Übersichtlichkeit und Unabhängigkeit der Teilprobleme erheblich beschleunigt.
(2) Die Programmwartung wird wegen der besseren Lesbarkeit und Auffindbarkeit der zu ändernden Prozeduren wesentlich vereinfacht.
(3) Die Übertragung von Teilaufgaben auf verschiedene Personen wird wegen der guten Integrierbarkeit erleichtert.

Mit dem Aufkommen dieses Konzepts wurde allerdings deutlich, daß die meisten der gängigen Programmiersprachen keine unmittelbare Unterstützung für den streng

hierarchischen Programmaufbau bieten. Zwar können auch mit FORTRAN oder COBOL strukturierte Programme geschrieben werden; jedoch kann dies nur durch eine intensive Anwendung der intermodularen Unterprogrammtechnik sowie über selbst auferlegte Beschränkungen bei der Anwendung von Sprunganweisungen erreicht werden.

Neuere Sprachen erzwingen hingegen förmlich die hierarchische Strukturierung durch das Fehlen oder die Einschränkung von Sprunganweisungen in der Sprachdefinition, durch die Begrenzung der Zeilenzahlen in einer Prozedur sowie durch die Möglichkeit der geschachtelten Prozedurdefinition.

Zu diesen neueren Entwicklungen gehört neben ELAN und ALGOL-68 vor allem die von Jensen und Wirth definierte Programmiersprache PASCAL.

PASCAL hat dank der Aktivitäten der University of California / San Diego (UCSD) insofern eine gewisse Verbreitung gefunden, als dort ein komplettes Entwicklungssystem für Mikrocomputer geschaffen wurde. Mikrocomputer werden in großen Stückzahlen verkauft und das UCSD-PASCAL-System ist mit Abstand das sicherste und komfortabelste Programmierinstrument in dieser Computerkategorie.

Allmählich dringt PASCAL aber auch in die Domäne der größeren Computer ein, so daß zumindest an den meisten Hochschulrechenzentren PASCAL-Compiler verfügbar sind.

Beim gegenwärtigen Definitions- und Implementierungsstand von PASCAL erscheint jedoch der in den USA verbreitete Optimismus ein wenig verfrüht, daß ALGOL, COBOL, PL/I und FORTRAN völlig ersetzt werden könnten. Hierzu bedarf es sicherlich noch eines weiteren Ausbaus u.a. bei den mathematischen Standardfunktionen, bei den Ein- und Ausgabeprozeduren und bei den Dateizugriffsmethoden. Andererseits ist zumindest das UCSD-PASCAL durch Schnittstellen zu ASSEMBLER so flexibel, daß fehlende Teile vom Benutzer selbst erzeugt werden können.

Ziel des vorliegenden Lehrbuchs ist es, eine einfache Einführung in die Sprache PASCAL zu geben; denn die Durchsicht der einschlägigen PASCAL-Publikationen hat gezeigt, daß PASCAL zumeist als "esoterische" Domäne der Informatiker begriffen wird. Der gewöhnliche Anwender wird i.d.R. durch den abstrakten Formalismus bei den

Definitionen der Sprachelemente sowie durch die Kompliziertheit der Programmbeispiele von der Benutzung abgeschreckt oder gar ausgeschlossen.

Beim eigenen Umgang mit dieser Sprache hat sich indessen gezeigt, daß PASCAL zu den am leichtesten erlernbaren Programmiersprachen gehört, sofern man sich mit dem für normale betriebswirtschaftliche und wissenschaftliche Anwendungen erforderlichen Niveau begnügt. Eben dieses Niveau wird im folgenden angestrebt und zwar in der Absicht, PASCAL und das Konzept der strukturierten Programmierung einem möglichst großen Benutzerkreis zu erschließen.

Zugleich besteht die Absicht, eine solide Basis für ein vertieftes Studium dieser Sprache zu schaffen. Der Leser soll durch Programmbeispiele zu allen wichtigen Sprachelementen zum Selbststudium angeregt werden. Die Beispiele sind bewußt einfach gehalten, um die Aufmerksamkeit direkt auf die Sprache lenken zu können. Ferner wird auf die nur Informatikern geläufige Backus-Notation verzichtet, denn es erscheint didaktisch nicht sinnvoll, dem Anfänger das Erlernen einer Sprache nur über den Umweg einer formalen Metasprache zu gestatten.

Schließlich werden die anspruchsvollen Datentypen SET und POINTER nicht behandelt, weil deren Anwendungen das Niveau einer Einführung überschreiten.

Besonderes Augenmerk wird indessen auf die hierarchische Strukturierung der Programme gelegt. Dazu wird für den Programmentwurf eine an meinem Lehrstuhl entwickelte Variante der Nassi-Shneiderman-Diagramme verwendet, die den Vorzug einer flexibleren graphischen Gestaltung bietet. Sicherlich kann das Anliegen der Strukturierung wegen der Einfachheit der Beispiele nicht voll überzeugend dargestellt werden. Es muß also an die Phantasie und die Kreativität des Lesers appelliert werden, sich die abgedruckten Programme als Teillösungen komplexerer Datenverarbeitungsaufgaben vorzustellen und die Inhalte einzelner Strukturblöcke durch Abwandlung und Erweiterung zu komplizieren.

Im letzten Teil des Buches findet sich eine Darstellung der wichtigsten Optionen des UCSD-PASCAL-Systems[0)] in Form einer Benutzeranleitung; insbesondere werden der

---

[0)] In der vorliegenden dritten Auflage wurde dieses System durch TURBO-PASCAL ersetzt.

Editor und das Dateisystem behandelt. Mit dieser Benutzeranleitung werden die Voraussetzungen für einen unmittelbaren Einstieg in das Selbststudium geschaffen.

An dieser Stelle möchte ich meinen Mitarbeitern Fräulein Gutschmidt, Fräulein Riegger und Herrn Kammermeier für ihre Unterstützung bei der Entwicklung und Erprobung der Programmbeispiele danken. Anerkennung gebührt insbesondere auch meinem Assistenten, Herrn Dr. Ferstl, für die Entwicklung des o.g. Struktogrammkonzepts. Schließlich möchte ich meiner Sekretärin Fräulein Thumser für ihre Geduld bei der "iterativen" Herstellung des druckreifen Manuskripts danken.

Regensburg, im Februar 1980 *Gerhard Niemeyer*

Vorwort zur zweiten Auflage

Eine Analyse der Sprachsituation im Jahre 1982 zeigt, daß sich PASCAL im Bereich der Mikrocomputer einen sicheren Platz erobert hat und hier auch gegenüber BASIC an Bedeutung gewinnt. Neuere Sprachversionen bieten gegenüber der ursprünglichen Wirthschen Definition erhebliche Komfortsteigerungen insbesondere bei der Dateiorganisation und bei den Input/Output-Funktionen. Einige Versionen bieten integrierte ASSEMBLER-Schnittstellen sowie Möglichkeiten der Bit-Manipulation, der Port-Adressierung und der physikalischen Dateimanipulation, wodurch PASCAL zur idealen Implementierungssprache wird.

Nach wie vor ist das UCSD-PASCAL-System das sicherste und komfortabelste Entwicklungssystem, dessen Standard selbst von Großrechnersystemen nicht erreicht wird.

Bei den Herstellern mittlerer und größerer Anlagen wie z.B. Kienzle oder Siemens gehört PASCAL mittlerweile zum Standardangebot. Mit Interesse wird vermerkt, daß PASCAL-Compiler auch für IBM-Anlagen verfügbar sind und daß das UCSD-PASCAL-System auf dem neuen Personal-Computer von IBM läuft.

Die vorliegende zweite Auflage des Buches "Programmieren in PASCAL" wurde in didaktischen Einzelheiten überarbeitet und von den bisher entdeckten Druckfehlern befreit.

Regensburg, im Juli 1982 *Gerhard Niemeyer*

Vorwort zur dritten Auflage

Die Software-Situation ist seit Mitte der 80er Jahre in einem tiefgreifenden Wandel begriffen. Die Anwendung der sogenannten Integrierten Pakete auf PC's sowie von Standardpaketen auf Großrechnern hat stark zugenommen und damit die klassische Programmentwicklung auf den zweiten Platz verdrängt. Als Reaktion darauf sind bereits einige Hochschullehrer für Wirtschaftsinformatik dazu übergegangen, die Programmierausbildung weitgehend durch die Unterweisung im Handhaben von Integrierten Paketen und Standardprogrammen zu ersetzen.

Absolut gesehen haben jedoch wegen der starken Zunahme der Rechnerinstallationen die Programmieraktivitäten zugenommen. Folglich wurden auch die Programmentwicklungssysteme ständig verbessert und haben heute in puncto Sicherheit, Schnelligkeit und Komfort einen bisher nicht für möglich gehaltenen Standard erreicht. Auf der Großrechnerseite sind in diesem Zusammenhang die sogenannten Vierte-Generations-Systeme INFORMIX, ADABAS, DELTA, u.a. zu nennen, während auf der PC-Seite die integrierten Entwicklungssysteme der Firmen Borland und Microsoft für die Sprachen PASCAL, "C" und PROLOG hervorzuheben sind.

In Bezug auf die Programmiersprache PASCAL ist festzustellen, daß sie sowohl in der Hochschullandschaft als auch in der Industrie einen beachtlichen Stellenwert erreicht hat, jedoch von der Sprache "C" hart bedrängt wird. Dies liegt einmal an der zunehmenden Verbreitung des Betriebssystems UNIX, dessen Basis- und Implementierungssprache "C" ist. Zum anderen ist festzustellen, daß "C" in den USA als amerikanisches Konzept deutlich bevorzugt wird.

Aus der Sicht des Software Engineering ist diese Entwicklung zu bedauern, weil "C" aus der Denkwelt der veralteten Sprachkonzepte von FORTRAN und BASIC stammt und die modernen Konzepte des hierarchischen Programmaufbaus nach dem Bausteinprinzip nur unvollkommen mit Hilfe von Baum- und Netzkonstruktionen realisieren kann. Hinzu kommt eine stark gewöhnungsbedürftige Syntax, die die Lesbarkeit der Programme im Vergleich zu PASCAL deutlich erschwert.

Die Strukturierungsideen von Nassi und Shneiderman sowie von Jensen und Wirth sind im Angesicht der weiterhin galoppierenden Software-Krise wichtiger denn je. Aus

diesem Grunde ist eine Programmiersprache wie PASCAL, die diese Konzepte 1:1 unterstützt, nach wie vor ein optimaler "stilbildender" Einstieg in die Welt des Programmierens.

Die vorliegende dritte Auflage nimmt auf eine Reihe neuerer Entwicklungen Rücksicht:

Im allgemeinen Sprachteil wurden
(1) alle noch vorhandenen Reste der "GOTO-Programmierung" eliminiert,
(2) das Konzept der Rekursion eingeführt und
(3) die Datentypen SET und POINTER aufgenommen.

Das Kapitel 4 wurde komplett neu gefaßt, wobei anstelle des vom Markt verschwundenen UCSD-PASCAL-Systems nunmehr das allgemein verbreitete und sensationell effiziente TURBO-PASCAL-System der Firma Borland dargestellt wird.

Eine weitere wichtige Neuerung ist die Aufnahme einer Aufgabensammlung mit Musterlösungen; diese Sammlung bildet die Basis für den am Lehrstuhl für Wirtschaftsinformatik an der Universität Regensburg regelmäßig veranstalteten Programmierkurs in PASCAL.

An dieser Stelle möchte ich allen Mitarbeitern für ihre Hilfe bei der Neugestaltung des Buches und beim Testen der Beispielprogramme danken.

Regensburg, im Mai 1990 *Gerhard Niemeyer*

Inhalt

1 Der Aufbau von PASCAL-Programmen

Der Aufbau eines PASCAL-Programms läßt sich mit folgendem Schema darstellen:

PROGRAM *name*;
  Definitionsteil
BEGIN
  Ablaufteil
END.

Für die häufig parallel zum Code angegebene Struktogrammdarstellung soll jede zwischen BEGIN und END eingeschlossene Anweisungsfolge als Block bezeichnet und graphisch als Rechteck dargestellt werden.

Gelegentlich werden auch einzelne Anweisungen als Blöcke gezeichnet. Es soll ferner vereinbart werden, sämtliche Programmwörter, die den Übersetzungsvorgang steuern (reserved words), in Großbuchstaben und sämtliche Benutzerwörter in Kleinbuchstaben zu notieren. Metasprachliche Benutzerwörter (Erklärungen für Benutzerwörter in Schemata oder Formalbeschreibungen) werden kursiv geschrieben.

Zu den nachfolgenden zahlreichen Programmbeispielen ist anzumerken, daß diese wegen der stets angestrebten Lauffähigkeit i.d.R. dem im Text erreichten Definitionsstand vorauseilen. Es wird jedoch immer auf die Punkte des Programms hingewiesen, auf die es in der betreffenden Situation ankommt.

Programmbeispiel:

```
PROGRAM beispiel_1;
(* Bottom up - Entwurf (1-stufig) *)
VAR operand,
    summe,
    produkt : REAL;
BEGIN
  WRITELN ('Bitte Operand eingeben :');
  READLN (operand);
  summe   := operand + 10;
  produkt := operand * 10;
  WRITELN ('Summe   = ',summe);
  WRITELN ('Produkt = ',produkt);
END.
```

Die Zeichenfolge "(* Bottom up - Entwurf (1-stufig) *)" ist ein Kommentar, der für den Übersetzungsvorgang ohne Wirkung ist und lediglich der Programmdokumentation dient. Kommentare können in beliebiger Zahl auftreten und an beliebigen Stellen eines Programms stehen.

Struktogramm: Bottom up-Entwurf (ein-stufig)

Das Programm beispiel_1 zeigt die drei Grundphasen jeder vollständigen Datenverarbeitungsaufgabe, nämlich die **Eingabe**, die **Verarbeitung** und die **Ausgabe**. Der Definitionsteil besteht in diesem Beispiel nur aus der Variablenerklärung. Allgemein kann der Definitionsteil jedoch folgende Komponenten enthalten:

(1) Konstantenerklärungen
(2) Typenerklärungen
(3) Variablenerklärungen
(4) Prozeduren, Funktionen

Eine rekursiv-hierarchische Zerlegung einer Datenverarbeitungsaufgabe wird dadurch möglich, daß für die unter (4) genannten Prozeduren und Funktionen wiederum das gleiche Aufbauschema gilt wie für das Programm.

PROCEDURE *name* [(*parameter*)];
  Definitionsteil
BEGIN
  Ablaufteil
END;

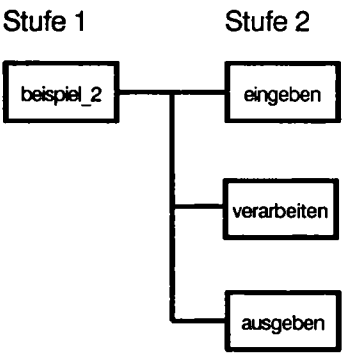
FUNCTION *name* [(*parameter*)]: *typ*;
  Definitionsteil
BEGIN
  Ablaufteil
END;

Ein geringfügiger Unterschied besteht lediglich darin, daß eine Prozedur bzw. Funktion mit einem ";" endet, während ein Programm mit einem "." abschließt. Die Schreibweise "[(*paramter*)]" bedeutet, daß "(*parameter*)" eine Option ist, die auch entfallen kann.

Programmbeispiel:

```
PROGRAM beispiel_2;
(* Bottom up - Entwurf (2-stufig) *)
VAR operand,
    summe,
    produkt  : REAL;

  PROCEDURE eingeben;
  (* Einlesen einer Zahl von der Tastatur *)
  BEGIN
    WRITELN ('Bitte Operand eingeben :');
    READLN (operand);
  END;

  PROCEDURE verarbeiten;
  (* Durchfuehrung der Berechnungen *)
  BEGIN
    summe   := operand + 10;
    produkt := operand * 10;
  END;

  PROCEDURE ausgeben;
  (* Ausgeben der Ergebnisse am Bildschirm *)
  BEGIN
    WRITELN ('Summe   = ',summe);
    WRITELN ('Produkt = ',produkt);
  END;

BEGIN
  eingeben;
  verarbeiten;
  ausgeben;
END.
```

Struktogramm: Bottom up-Entwurf (zwei-stufig)



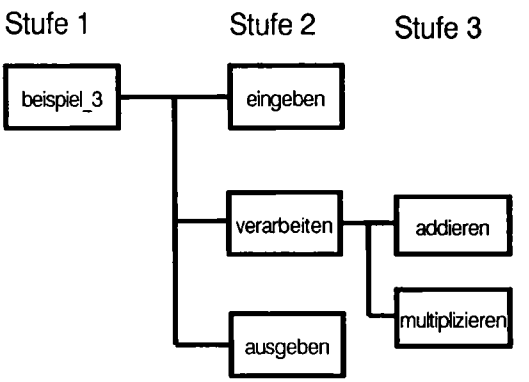
Das Programm beispiel_2 zeigt, wie die drei Grundphasen einer Datenverarbeitungsaufgabe **Eingabe, Verarbeitung** und **Ausgabe** als Prozeduren vereinbart und im Ablaufteil des Programms aufgerufen werden.

Struktogrammtechnisch ergeben sich drei Blöcke, die sich in einer sequentiellen Anordnung befinden; und es gilt die Konvention, daß diese in der angegebenen Reihenfolge von oben nach unten durchlaufen werden.

Programmbeispiel:

```
PROGRAM beispiel_3;
(* Bottom up - Entwurf (3-stufig) *)
VAR operand,
    summe,
    produkt    : REAL;


  PROCEDURE eingeben;
  (* Einlesen einer Zahl von der Tastatur *)
  BEGIN
    WRITELN ('Bitte Operand eingeben :');
    READLN (operand);
  END;


  PROCEDURE verarbeiten;
  (* Durchfuehrung der Berechnungen  *)


    PROCEDURE addieren;
    (* Berechnung der Summe *)
    BEGIN
      summe := operand + 10;
    END;


    PROCEDURE multiplizieren;
    (* Berechnung des Produkts *)
    BEGIN
      produkt := operand * 10;
    END;


  BEGIN
    addieren;
    multiplizieren;
  END;


  PROCEDURE ausgeben;
  (* Ausgabe der Ergebnisse am Bildschirm *)
  BEGIN
    WRITELN ('Summe   = ',summe);
    WRITELN ('Produkt = ',produkt);
  END;

BEGIN

  eingeben;

  verarbeiten;

  ausgeben;

END.
```

Struktogramm: Bottom up-Entwurf (drei-stufig)



Das Programm beispiel_3 zeigt dasselbe Problem in einer dreistufigen Hierarchie.

Eine Vorgehensweise, die zunächst die prozedurale Zerlegung eines Problems und ganz zum Schluß die unterste Anweisungsebene realisiert, nennt man **top down-Methode.** Umgekehrt nennt man ein Vorgehen, das von der untersten Anweisungsebene ausgehend allmählich zu einem hierarchischen Programmaufbau gelangt, **bottom up-Methode.**

In der Praxis werden beide Methoden der Strukturierung in beliebiger Folge und auf beliebigen Zerlegungsstufen eine Problems gemischt. Die bevorzugte Richtung sollte jedoch stets "top down" sein, weil man hiermit am schnellsten zu einer sauberen Programmstruktur gelangt. Da jedoch die Abschätzung der Komplexität der einzelnen Prozeduren i.d.R. schwerfällt, wird man häufig erst bei der Realisierung auf Anweisungsebene die Notwendigkeit einer weiteren Problemzerlegung erkennen und dann als Korrekturphase die bottom up-Richtung einschlagen.

Die bisherigen Programmbeispiele zeigen in der Reihenfolge beispiel_1, beispiel_2, beispiel_3 ein reines bottom up-Verfahren. Die nachfolgenden Codebeispiele zeigen dagegen die zwei Vorphasen des Programmbeispiels beispiel_3 bei reiner top down-Konstruktion:

1. Phase (nicht lauffähig)

```
PROGRAM beispiel_3;{top down - Entwurf(1.Phase)}

BEGIN
  eingeben;
  verarbeiten;
  ausgeben;
END.
```

2. Phase (nicht lauffähig)

```
PROGRAM beispiel_3;{top down - Entwurf(2.Phase)}
VAR operand,summe,produkt : REAL;

  PROCEDURE eingeben;
  BEGIN
    WRITELN ('bitte operand eingeben :');
    READLN (operand);
  END;

  PROCEDURE verarbeiten;
  BEGIN
    addieren;
    multiplizieren;
  END;

  PROCEDURE ausgeben;
  BEGIN
    WRITELN ('SUMME    = ',summe);
    WRITELN ('PRODUKT = ',produkt);
  END;

BEGIN
  eingeben;
  verarbeiten;
  ausgeben;
END.
```

Die dritte Phase ist identisch mit der dritten Stufe des zuvor beschriebenen bottom up-Entwurfs und wird daher nicht mehr gezeigt.

2 Der Definitionsteil von PASCAL-Programmen

Definitionen sollten aus Gründen der Verfügbarkeit und Übersichtlichkeit in der nachfolgend angegebenen Reihenfolge vorgenommen werden:

(1) Konstantendefinitionen
(2) Typendefinitionen
(3) Variablendefinitionen
(4) Prozeduren, Funktionen

Einzelne Definitionsarten können fallweise fehlen.

Für die Namensbildung bei Konstanten, Typen, Variablen, Prozeduren und Funktionen gelten folgende Regeln: Ein Name kann beliebig viele Zeichen (Buchstaben, Ziffern) haben, von denen allerdings nur die ersten 63 signifikant sind. Das erste Zeichen muß ein Buchstabe oder ein Underscore "_" sein; Sonderzeichen dürfen nicht verwendet werden.

2.1 Konstantendefinition

Konstanten sind Werte, die im Ablaufteil oder bei den Definitionen eines Programms verwendet werden und die ihre Werte stets unverändert beibehalten. Konstanten können entweder durch Wertangabe unmittelbar oder durch Zuordnung zu frei wählbaren Namen im Definitionsteil vereinbart werden. Im ersten Fall spricht man von **direkten** Konstanten und im zweiten von **indirekten** Konstanten. Zur Namensbildung siehe oben.

Indirekte Konstanten können unter ihrem Namen aufgerufen werden; dabei kann sich ein Codiervorteil ergeben, wenn eine Konstante von erheblicher Länge ist, der Name hingegen kurz und die Verwendung häufig ist. Andernfalls empfiehlt sich aus Gründen der Übersichtlichkeit die Anwendung von direkten Konstanten.

PASCAL kennt folgende Konstantentypen:
Ganzzahlige, reelle, boolesche Konstanten, Zeichen- und Zeichenketten-Konstanten.

Die Erzeugung indirekter Konstanten geschieht durch eine CONST-Anweisung. Das allgemeine Format lautet:

CONST $kname_1 = wert_1;...;kname_n = wert_n;$

Codierbeispiel:

```
CONST x     = 31745; negx = -x;
      e     = 2.7183;
      on    = TRUE; off = FALSE;
      key   = 'A';
      meldung = 'unzulaessiger Operand; Eingabe wiederholen';
```

Im obigen Beispiel sind "x" und "negx" ganzzahlige, "e" eine reelle, "on" und "off" boolesche Konstanten, "key" eine Zeichenkonstante und "meldung" eine Zeichenketten-Konstante. Der Typ wird durch die Wertzuweisung festgelegt; besonderer Typenerklärungen bedarf es nicht. Beachtenswert ist, daß "negx" die unmittelbar vorausgehende Definition benutzen kann.

Manche PASCAL-Implementierungen liefern häufig vorkommende Konstanten wie e, π, etc., gleich als Systemkonstanten, d.h. eine Vereinbarung ist nicht erforderlich. Eine häufig implementierte Systemkonstante ist ferner MAXINT, die die größtmögliche ganze Zahl des betreffenden Rechners angibt.

Programmbeispiel:

```
PROGRAM beispiel_4;
(* Definition und Verwendung von Konstanten *)
CONST const_integ = 10000;
      const_reell = 0.001;
      txt_integ   = 'INTEGER';
      txt_reell   = 'REAL';
      txt_const   = ' - KONSTANTE';
VAR   var_integ : INTEGER;
      var_reell : REAL;
      var_char  : CHAR;
BEGIN
  var_char := ':';
  WRITELN (txt_integ, txt_const, var_char,' ',const_integ);
  WRITELN (txt_reell, txt_const, var_char, const_reell);
  var_integ := const_integ  + 10000;
  var_reell := const_reell + 0.009;
  WRITELN (txt_integ, txt_const, var_char, ' ', var_integ);
  WRITELN (txt_reell, txt_const, var_char, var_reell);
END.
```

Das Beispiel beispiel_4 zeigt die arithmetische Verknüpfung von indirekten und direkten Konstanten sowie die Verwendung von Zeichen- und Zeichenketten-Konstanten.

2.2 Typendefinition

Typen sind die Codeformen, mit denen die Datenelemente oder Datenstrukturen eines PASCAL-Programms verarbeitet werden können. Typen können im Definitionsteil durch Zuordnung frei vorgebbarer Benutzernamen als **indirekte Typen** (Benutzertypen) vereinbart werden (zur Namensbildung vgl. S. 7). Die Namen sind bei nachfolgenden Typendefinitionen und vor allem bei Variablendefinitionen verwendbar, wodurch komplexe Definitionen strukturiert und leichter lesbar werden.

Daneben ist bei der Variablendefinition auch eine **direkte Typenvereinbarung** möglich, bei der der Typ unmittelbar bei den Variablen angegeben wird (vgl. S. 21f.).

In diesem Zusammenhang muß betont werden, daß mit der indirekten Typendefinition keine Speicherplatzreservierung und keine Wertzuweisung an die einzelnen Datenelemente verbunden ist. Es werden lediglich die Codierung und der Wertebereich festgelegt. Die Speicherplatzreservierung erfolgt erst durch die Variablendefinition und die Wertzuweisung erst durch Lese- oder Wertzuweisungsbefehle.

Die Erzeugung indirekter Typen geschieht durch eine TYPE-Anweisung.

Das allgemeine Format lautet:

TYPE $tname_1$ = typ_1;...;$tname_n$ = typ_n;

Für typ_1 bis typ_n können System- oder Benutzertypen eingesetzt werden; letztere müssen jedoch zuvor durch Systemtypen definiert worden sein. Die Systemtypen werden in den nachfolgenden Abschnitten beschrieben.

2.2.1 Einfache Typen

2.2.1.1 Der Typ BOOLEAN

Der Typ BOOLEAN hat den Wertebereich {TRUE, FALSE}; d.h. ein Datenelement dieses Typs kann nur diese beiden Werte annehmen.

Das allgemeine Format für die Definitionsanweisung lautet:

TYPE *tname* = BOOLEAN;

Codierbeispiel:

```
TYPE janein = BOOLEAN;
```

2.2.1.2 Der Typ INTEGER

Der Typ INTEGER hat den Wertebereich {-32768, 32767} und umfaßt damit den gesamten Vorrat der bei einem Mikrocomputer in 16 Bits darstellbaren ganzen Zahlen.[1)] Das allgemeine Format für die Definitionsanweisung lautet:

TYPE *tname* = INTEGER;

Codierbeispiel:

```
TYPE ganz_typ = INTEGER;
```

2.2.1.3 Der Typ REAL

Der Typ REAL hat den Wertebereich aller darstellbaren reellen Zahlen.[1)] Eine REAL-Zahl setzt sich aus drei Komponenten zusammen: dem Vorzeichen, einer Mantisse und einem Exponenten, wobei der Exponent wiederum ein Vorzeichen hat.

Die in der Mantisse gespeicherten Dezimalstellen bestimmen die Genauigkeit des Wertes und der Wert des Exponenten legt den Platz auf der Zahlengerade fest. TURBO-PASCAL speichert REAL-Zahlen in 6 Bytes und erreicht damit einen Wertebereich von etwa $\pm\ 10^{-38}$ bis $\pm\ 10^{38}$ bei einer Genauigkeit von ca. 11 Dezimalstellen.[2)]

Neben der Darstellung mit Mantisse und Exponent ist auch die normalisierte Darstellung mit Dezimalpunkt möglich.

[1)] Ausführliche Angaben: siehe Anhang D

[2)] Diese Angaben gelten für Compiler-Einstellungen ohne Arithmetikprozessor bzw. ohne Emulation.

Das allgemeine Format für die Definitionsanweisung lautet:

TYPE *tname* = REAL;

Codierbeispiel:

```
TYPE ergebnis_typ = REAL;
```

2.2.1.4 Der Typ CHAR

Der Typ CHAR umfaßt den Wertebereich aller darstellbaren Zeichen des ASCII-Code, nämlich die Großbuchstaben A bis Z, die Kleinbuchstaben a bis z und die Ziffern 0 bis 9, sowie eine Reihe von Sonderzeichen. Eine Tabelle des ASCII-Code befindet sich im Anhang C.

Das allgemeine Format für die Definitionsanweisung lautet:

TYPE *tname* = CHAR;

Codierbeispiel:

```
TYPE zeichen_typ = CHAR;
```

Programmbeispiel:

```
PROGRAM beispiel_5;
(* Benutzertypen- und Variablendefinition *)
CONST  const_integ = 2;
       const_reell = 0.999999;
       kommentar   = 'Bitte Faktor eingeben für ';
       txt_integ   = 'INTEGER - Zahl';
       txt_reell   = 'REAL - Zahl';
TYPE   ganz_typ  = INTEGER;
       reell_typ = REAL;
VAR    faktor_integ,
       integ_zahl   : ganz_typ;
       faktor_reell,
       reell_zahl   : reell_typ;
BEGIN
  WRITELN (kommentar,txt_integ);
  READLN (faktor_integ);
  WRITELN (kommentar,txt_reell);
  READLN (faktor_reell);
  integ_zahl := const_integ * faktor_integ;
  reell_zahl := const_reell * faktor_reell;
  WRITELN ('Integerzahl = ',integ_zahl);
  WRITELN ('Realzahl    = ',reell_zahl);
END.
```

Das Programm beispiel_5 zeigt die Definition der Benutzertypen "ganztyp" und "reelltyp" sowie deren Verwendung bei der Variablendefinition.

2.2.2 Strukturierte Typen

Strukturierte Datentypen dienen der Definition von **Datenstrukturen,** die stets aus mehreren Datenelementen zusammengesetzt sind.

2.2.2.1 Der Aufzählungstyp

Der Aufzählungstyp gestattet die Angabe einer Liste von Wertnamen und deren Zuordnung zu einem Benutzernamen. Das allgemeine Format für die Definitionsanweisung lautet:

TYPE *tname* = (*wertname*$_1$,...,*wertname*$_n$);

Intern werden den Wertnamen 1 bis n die Ordinalwerte 0 bis n-1 zugeordnet, die über die PASCAL-Funktion ORD(wertname$_k$) erreicht werden können; dabei darf n den Wert 256 nicht überschreiten. Der Zweck dieser Typenvereinbarung ist die Einschränkung des Wertebereichs einer Variable auf die Elemente der Aufzählung, bzw. genauer auf die hinter den Wertnamen stehenden Ordinalzahlen. Allerdings können nur die vereinbarten Wertnamen (und nicht etwa die Ordinalzahlen) im Zusammenhang mit dieser Variable verwendet werden. Andere Wertzuweisungen oder Vergleiche führen zu Übersetzungsfehlern.

Da die Ordinalzahlen andererseits i.d.R. nicht alleiniger Gegenstand der Verarbeitung sind, werden Aufzählungen vielfach in Verbindung mit Feldtypen (vgl. S. 15ff.) als Indexmengen verwendet; d.h. sie dienen der qualifizierten Adressierung von weitergehenden Informationen in den betreffenden Datenfeldern.

Codierbeispiel:

```
TYPE werktage = (montag,dienstag,mittwoch,donnerstag,freitag);
     bits     = (b7,b6,b5,b4,b3,b2,b1,b0);
```