LOGIN
INFORMATIK UND COMPUTER IN DER SCHULE

VON BASIC ZU PASCAL

von Hannes Gutzer

Sonderdruck für
Nutzer von Kleincomputern aus DDR-Produktion
in Zusammenarbeit mit Deutschlandsender Kultur

1991

Vorwort

Wer Pascal lernen will, wählt normalerweise Turbo-PASCAL auf einem PC. Was tun aber diejenigen, die vorerst noch mit einem Kleincomputer aus DDR-Produktion vorlieb nehmen müssen?

In Zusammenarbeit mit der Zeitschrift LOG IN–Zeitschrift für Informatik und Computer in der Schule, möchten wir den Kleincomputernutzern in den neuen Bundesländern helfen. Wir bieten Ihnen mit diesem Sonderdruck einen PASCAL-Kurs für die Kleincomputer KC 85/x bis KC 87 an, der auf dem Rundfunkkurs von Deutschlandsender Kultur basiert.

Der PASCAL-Kurs von Dr. Hannes Gutzer ist so aufgebaut, daß von den bisherigen BASIC-Kenntnissen ausgegangen und der Übergang zu Turbo-Pascal auf PC-Technik vorbereitet wird.

Besonderer Dank gilt dem Deutschlandsender Kultur für die Verbreitung des Rundfunkkurses und den Herren Dr. Burmeister, Dipl.-Math. Lehmann und Dr. Vetters für die Herstellung des KC-PASCAL-Compilers. Die Redaktion freut sich über Hinweise und Kritiken der Leser zu diesem PASCAL-Kurs.

LOG IN ist eine Zeitschrift, die in erster Linie die informatische Bildung unterstützen will. Dabei geht es nicht nur um den Informatikunterricht, LOG IN ist auch ein Forum für den Einsatz des Computers in anderen Unterrichtsfächern. Die Kollegen in den neuen Bundesländern möchten wir mit unseren Erfahrungen besonders unterstützen. Das Heft 6 '90 stand unter dem Titel „Informatische Bildung – eine gesamtdeutsche Bestandsaufnahme". Hier finden sich auf fast 100 Seiten zahlreiche Hinweise, Unterrichtsvorschläge und Kontaktadressen zur informatischen Bildung. Das Heft kann von der Redaktion angefordert werden (Preis: 16,80 DM). Ebenso offen sind wir aber auch für Ihre Ideen und Erfahrungen. Möchten Sie Ihre Vorstellungen einer breiten Öffentlichkeit zugänglich machen, dann senden Sie bitte Artikel bzw. Artikelangebote an:

Redaktion LOG IN
Freie Universität Berlin
Zentralinstitut für Fachdidaktiken – SE 3
Habelschwerdter Allee 45
W-1000 Berlin 33

Von BASIC zu PASCAL

Eine Einführung in KC-PASCAL für alle Kleincomputerfreunde

von Hannes Gutzer

BEGIN

Mit dem englischen Wort BEGIN in der Überschrift wollen wir in die Programmiersprache PASCAL einsteigen. Wir haben dabei vor allem an diejenigen gedacht, die bisher mit BASIC gearbeitet haben und nun nach einer weiteren Programmiersprache dürsten, um deren Vor- aber auch Nachteile kennenzulernen.

Der Interessent dieses PASCAL-Kurses für Einsteiger sollte mit einem der Computertypen KC 87, KC 85/2/3/4 oder ZX Spectrum sicher umgehen können und Kenntnisse in der Programmiersprache BASIC besitzen. Von Vorteil ist, wenn Sie selbst schon eigene BASIC-Programme geschrieben und in der Phase der Problemanalyse einen Algorithmus oder ein Struktogramm zu Papier gebracht haben. An dieser Stelle können Sie in unserem PASCAL-Kurs ansetzen, indem Sie die Algorithmen oder Struktogramme nicht in BASIC, sondern in KC-PASCAL codieren. Doch bis es soweit ist, müssen wir uns erst mit dieser Programmiersprache vertraut machen.

Niklaus Wirth, Professor an der Eidgenössischen Technischen Hochschule in Zürich, hat 1971 die Sprache PASCAL auf Grundlage seiner Erfahrungen mit ALGOL-68 als Lehrsprache erarbeitet. Den Namen PASCAL wählte er zu Ehren des französischen Mathematikers und Philosophen Blaise Pascal, der mit 19 Jahren im Jahre 1642 für seinen Vater eine mechanische Rechenmaschine baute, wovon ein Exemplar heute noch im Mathematisch-Physikalischen Salon in Dresden bewundert werden kann. Wirth beschreibt in seinem Buch „Systematisches Programmieren" noch einmal das, was wir uns nicht oft genug bewußt machen können: „Der Computer ist ein Automat, der Prozesse nach genau vorgeschriebenen Verhaltensmaßregeln ausführt. Er besitzt nur einen bescheidenen Vorrat an Befehlen, führt diese aber schnell und zuverlässig aus." Aber das wußten Sie sicherlich schon. Wirth schreibt weiter, daß der Computer sehr lange Befehlsfolgen ausführen kann, in denen ungeheuer viele Kombinationen von Elementarbefehlen stecken. Das macht letztlich die Leistung eines Computers aus.

Damit wir dem Computer auf menschenfreundliche Art und Weise mitteilen können, was er tun soll, nutzen wir eine Programmiersprache. Jede dieser Sprachen, egal ob BASIC, PASCAL, FORTH, C, LISP, PROLOG, MODULA oder andere, haben ihre Vor- und Nachteile. Eben deshalb gibt es ja so viele. Aber allen ist gemeinsam, schreibt Wirth hier weiter, daß die gewählte Programmiersprache stets Werkzeug, aber niemals Endzweck ist. Eine scherzhaft gemeinte Ausnahme bilden jene Computerfreaks, für die es nichts Langweiligeres geben soll als ein Programm, das fehlerfrei läuft.

Somit ist nach Wirth das Programmieren eine Tätigkeit, die mit Hilfe des Werkzeuges Programmiersprache Befehlsfolgen konzipiert, die den im Algorithmus oder Struktogramm geforderten Handlungsablauf vollziehen. Demnach ist ein Programm eine Reihe von Anweisungen in einer gewählten Programmiersprache, wobei die Reihenfolge der Anweisungen keineswegs der Reihenfolge der Handlungen entsprechen muß. Gerade das wird bei der Arbeit mit PASCAL, bei der viele kleine oder große Programmbausteine das Salz in der PASCAL-Suppe sind, deutlich werden. Bevor wir in unseren Kurs einsteigen, sind einige Verständigungen unerläßlich.

Interpreter oder Compiler

Die BASIC-Freunde sind eine interpretative Arbeitsweise gewöhnt, obwohl mit der Verbreitung von 16-Bit-Rechnern auch hervorragende BASIC-Compiler verfügbar sind. PASCAL existiert aber ausschließlich als Compilersprache. Hier wird für viele ein erstes Umdenken erforderlich sein.

Die Arbeitsweise eines Compilers kann mit der eines Übersetzers in einem Fremdsprachenbüro verglichen werden. In seinem Büro wird ihm ein fremdsprachiger Text vorgelegt, den er in einem Stück – die Pausen wollen wir ihm natürlich gönnen – übersetzt und dann diese Übersetzung an den Auftraggeber ausliefert. Damit ist für ihn die Sache erledigt, und er kann Urlaub machen. Zur Nutzung dieses übersetzten Textes, in unserem Fall das Abarbeiten des Programms, ist seine Anwesenheit nicht mehr erforderlich.

Der Interpreter hingegen ist mit einem Dolmetscher vergleichbar. Er muß während des gesamten Gespräches der beiden Kommunikationspartner anwesend sein und Satz für Satz übersetzen. Dieses „scheibchenweise" Übertragen in eine andere Sprache dauert natürlich länger als die

Übersetzung, dafür können Mißverständnisse sofort aus dem Weg geräumt werden, denn der Dolmetscher ist ja immer anwesend. Die sofortigen Korrekturmöglichkeiten sind ein Vorteil dieser interpretativen Arbeitsweise, z. B. in BASIC. Ein Nachteil ist aber die geringere Rechengeschwindigkeit. So wird unser KC-PASCAL immerhin rund achtmal schneller als der BASIC-Interpreter sein – gute Aussichten also für Spieleprogrammierer.

Im professionellen Bereich, also bei den Berufsprogrammierern, wird vorwiegend mit Compilersprachen gearbeitet. Die Gründe liegen u. a. in der höheren Abarbeitungsgeschwindigkeit und dank modularer Konzepte solcher Sprachen in der Möglichkeit, die Teilarbeiten von vielen Programmierern zu einer fehlerfreien Gesamtlösung „zusammenzubauen". Das ist in BASIC, wo z. B. alle Variablennamen im gesamten Programm gleichermaßen gelten, schier unmöglich.

Ein Compiler besteht aus mehreren Bausteinen. Diese Bausteine sind

◇ der Editor,
◇ der eigentliche Compiler,
◇ der Linker und
◇ das sogenannte Laufzeitsystem (engl. runtime system).

Der Linker (engl. link = verbinden) dient zum Verbinden von Programmteilen und ist in unserem KC-PASCAL und z. B. auch in Turbo-PASCAL gleich im Compiler mit enthalten.

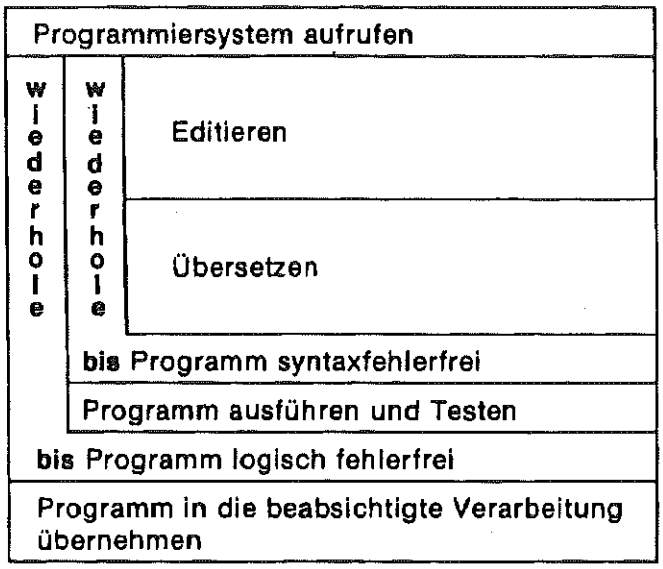
Für die Phase der Programmherstellung benötigen wir den Editor und den Compiler. Mit dem Editor (engl. edit = redigieren, herausgeben) geben wir das auf dem Papier vorliegenden PASCAL-Quellprogramm in den Arbeitsspeicher ein. Komfortable PASCAL-Systeme, z. B. Turbo-PASCAL, arbeiten mit sogenannten full-screen-Editoren. Das ist für Korrekturen im Quellprogramm von Vorteil. Da der gesamte Bildschirm im Editiermodus aktiviert ist, kann mit den Kursortasten, den Einfügetasten wie auch mit den Löschtasten sofort auf dem Bildschirm geändert werden.

Solch einen Komfort können wir von unserem KC-PASCAL nicht verlangen. Unser Editor ist zeilenorientiert. Jede Zeile auf dem Bildschirm erhält hier eine Zeilennummer. Über diese Zeilennummern wird das Quellprogramm eingegeben und auch korrigiert. Dies widerspricht leider dem PASCAL-Konzept, nach dem ein PASCAL-Programm eigentlich aus einer einzigen, mehr oder weniger langen Zeile besteht.

Das PASCAL-Programm wird mit dem englischen Wort PROGRAM eingeleitet und wie ein vollständiger Satz mit einem Punkt beendet.

Beachten Sie also bitte, daß die Zeilennummern des Editors nur eine Hilfsfunktion bei der Erstellung von Quellprogrammen haben. Sie dürfen nicht mit den „lebensnotwendigen" Zeilennummern in den uns bisher vertrauten BASIC-Dialekten verwechselt werden.

Liegt das Quellprogramm im Speicher vor, dann kann es mit dem Compiler und Linker in das Objektcodeprogramm, das letztlich ein Maschinencodeprogramm ist, übersetzt werden. Das Wechselspiel zwischen Editor und Compiler erfolgt solange, bis eine fehlerfreie Übersetzung



Quelle: LOG IN Archiv

Prinzipielle Vorgehensweise beim Erstellen eines PASCAL-Programms.

vorliegt. Damit ist die Phase der Programmherstellung abgeschlossen.

Zum Abarbeiten eines PASCAL-Programms werden nur das Objektcodeprogramm und das Laufzeitsystem benötigt. Letzteres enthält alle Routinen zum Rechnen, logischen Vergleichen u. a.m. Editor und Compiler sind also bei der Programmnutzung nicht mehr erforderlich. Wir haben deshalb auch die Möglichkeit, das Objektcodeprogramm gemeinsam mit dem Laufzeitsystem auf Kassette oder Diskette auszulagern und das Ganze dann wie ein Maschinencodeprogramm zu nutzen. Natürlich heben wir uns das Quellprogramm gut auf. Nur über diese Quelle können Veränderungen und Ergänzungen im Programm vorgenommen werden.

Für unser KC-PASCAL liegen Editor, Compiler mit Linker und das Laufzeitsystem gemeinsam als ein Kassettenfile vor. Das ist sehr bequem, allerdings umfassen Editor und Compiler rund 12 Kbyte und das Laufzeitsystem etwa 4 Kbyte, zusammen also rund 16 Kbyte. Damit könnte der Kleincomputernutzer ohne Speichererweiterungsmodule aber nichts anfangen, denn für die Quell- und Objektcodeprogramme wäre kein Platz mehr vorhanden.

Aus diesem Grund haben die Mathematiker Dr. Burmeister, Dipl.-Math. Lehmann und Dr. Vetters von der TU Dresden für diesen Kurs einen Sondercompiler geschrieben, der zusammen mit der Laufzeitbibliothek nur etwa 12 Kbyte groß ist. Damit können alle Programme, die hier vorgestellt werden, auch mit einem 16-Kbyte-RAM abgearbeitet werden. Dieser Compiler mit der Versionsbezeichnung V2.1 befindet auf der zum Kurs gelieferten Kassette. Er ist auf den Computern KC 87 und KC 85/2/3/4 sofort lauffähig. Die ZX Spectrum-Freunde werden gewiß Zugriffe zu dem Spectrum-PASCAL-Compiler zu finden wissen. Dieser Compiler der Firma HISOFT ist übrigens der Urvater des hier vorgestellten Compilers.

Ich möchte noch nachtragen, daß an der TU Dresden, aber z. B. auch an der TU Chemnitz, weitere PASCAL-Compiler für die Kleincomputer KC 87 und KC 85 / 2 / 3 / 4 entstanden sind. So werden für den 16-Kbyte großen PASCAL-Compiler für die KC 85 / 2 / 3 der TU Dresden leistungsfähige Prozeduren mitgeliefert, mit denen Fenster, Kursorpositionierung, Farbe und Vollgrafik realisiert werden können. Das dürfte vor allem für spieleprogrammierende PASCAL-Kenner von Interesse sein. Sollten Sie Interesse an diesem Compiler haben, wenden sich bitte an: Dr. Klaus Vetters, Gartenheimallee 2, O-8021 Dresden.

Es heißt zwar nachdenken, man sollte es aber vorher tun

Beginnen wir mit einer provokativen Bemerkung aus einem PASCAL-Lehrbuch. Sie lautet: Das Erlernen von PASCAL beginnt mit dem Ausschalten des Computers. Das ist natürlich übertrieben, aber mit dem wilden Tastenhacken einiger Freizeitprogrammierer ist es wahrlich vorbei. Sowohl die Programmiersprache PASCAL, als auch die compilierende Arbeitsweise machen das fast unmöglich. Aber ich halte noch eine zweite Provokation bereit:

Es gibt keinen Weg, ein PASCAL-Programm zu schreiben, wenn man weiter in BASIC denkt.

Das ist ebenso übertrieben, aber PASCAL fordert z. B. ein vorausschauendes Denken, das BASIC nicht zwingend von uns verlangte. Dies erscheint auch im ersten Moment unbequem, wird sich aber vor allem bei umfangreichen Programmen bezüglich Übersichtlichkeit und Fehlerfreiheit als Vorteil erweisen. Aber lassen wir doch einmal auf satirische Weise einen BASIC-Fan zu Wort kommen:

„PASCAL ist für mich die Sprache akademisch Neunmalkluger, die nie in ihrem Leben richtig programmiert haben. Selbst ein einfaches PASCAL-Programm ist doppelt so lang wie das entsprechende BASIC-Programm. Es ist schon verrückt, was man da so alles eintippen muß. Schon ganz am Anfang muß man das englische Wort PROGRAM eingeben, wohl um zu verhüten, daß der Computer meint, es kommt ein Einkaufszettel. Meist folgen dann noch weitere Zeilen mit Stuß. Wenn man dann endlich soweit ist und wirkliche Programmanweisungen eintippen will, muß man erst noch BEGIN eingeben, ganz so, als ob der Computer gewarnt werden soll, daß man immer noch programmieren will und seine Meinung noch nicht geändert hat. Dann ist der Rest eigentlich wie in BASIC. Aber auch hier sind Kleinigkeiten zu beachten, wie Semikolons, Doppelpunkte, Leerzeichen, eckige und geschweifte Klammern und was weiß ich noch, die einem diese BASIC-Ähnlichkeit auch noch vermiesen können".

Der BASIC-Fan übertreibt gewiß, ein Urteil sollten Sie erst am Schluß unseres Kurses fällen. Schlagwörter bei der Lobpreisung von PASCAL sind das Typ- und das Prozedur- bzw. Modulkonzept. Hier zunächst soviel dazu:

◇ Alle Daten, mit denen wir in einem PASCAL-Programm arbeiten, müssen einem Datentyp zugeordnet werden. Das kann ein in PASCAL vorgegebener Standardtyp, beispielsweise eine reelle Zahl mit der Typbezeichnung REAL, oder eine eigene Typbezeichnung, z. B. die Arbeitstage einer Woche von Montag bis Freitag, sein. Das bringt z. B. bei ökonomischen Berechnungen viele Vorteile.

◇ Des weiteren müssen in PASCAL am Programmanfang alle Bezeichner, also Namen von Variablen, vereinbart, d. h. genannt und einem Typ zugeordnet werden. Das schließt solche boshaften Fehler einfacher BASIC-Interpreter aus, bei denen einer Zahlenvariablen mit falsch geschriebenem Bezeichner „heimlich" eine Null zugewiesen wird. In PASCAL ist dieser falsch geschriebene Bezeichner nicht vereinbart und eine Fehlermeldung die Folge.

◇ Von BASIC kennen wir Unterprogramme und die Definition eigener Funktionen. PASCAL bietet hier ein Vielfaches an Leistung, Fehlersicherheit und Eleganz. Dazu gehört z. B., daß es neben globalen auch lokale Variablen gibt, die mit ihrem Bezeichner nur in einem bestimmten Programmbaustein Gültigkeit haben. Auf diese Weise lassen sich viele Module ohne Kollisionen zu einem Gesamtprogramm zusammenstellen. In PASCAL heißen die Unterprogramme übrigens Prozeduren.

Es erscheint beispielsweise zunächst als Nachteil, daß es in PASCAL keine direkte Möglichkeit zum Potenzieren mit reellem Exponenten, also y hoch x, wie in BASIC oder auf dem Taschenrechner gibt. Das ist aber kein Problem, denn eine Funktion mit dem Namen YHOCHX ist schnell geschrieben und in das Quellprogramm eingebunden.

Dieses Modulkozept verlangt nach einer strukturierten Programmierung. Das bedeutet für uns, daß zunächst nicht Computer und Tastatur, sondern Papier, Bleistift, Radiergummi und ein kühler Kopf vorhanden sein müssen. Natürlich hat PASCAL keineswegs das strukturierte Programmieren für sich allein gepachtet. Auch in BASIC lassen sich, und das haben viele Veröffentlichungen gezeigt, sauber strukturierte Programme herstellen. BASIC macht aber das Abweichen vom Pfad der Tugend leichter, wozu sich denn auch viele Hobbyprogrammierer verleiten lassen.

So tugendhaft ist aber PASCAL nun auch wieder nicht. Seine strengen Forderungen zur Verwendung von Leerzeichen und Semikolons können einem schon zu schaffen machen. Auch die Ein- und Ausgabearbeit ist nicht so schön einfach wie in BASIC. Windows, Kursorpositionierung, Farben und Vollgrafik sind leider Stiefkinder von PASCAL. Das betrifft unser KC-PASCAL, aber auch Turbo-PASCAL für die 8-Bit-Rechner. Gelöst sind diese Probleme erst in Turbo-PASCAL ab Version 4 für 16-Bit-Rechner. Bei der Arbeit mit unserem 12-Kbyte-Compiler werden wir demnach auch auf all diese schönen Dinge zunächst verzichten müssen. Die PASCAL-Freunde sollten aber nicht aufgeben und stets TURBO-PASCAL im Blickfeld haben. In unserem Einsteigerkurs wollen wir aber erst einmal die tiefer hängenden Trauben ernten.

Der Jungfernlauf

Schreiten wir also zur Tat. Wir benötigen dazu unseren KC-PASCAL-Compiler und ein Quellprogramm. Das Quellprogramm mit dem Namen KREIS soll den grundsätzlichen Programmaufbau und den Umgang mit Quellprogrammen deutlich machen. Es ist ein simples EVA-Programm (E wie Eingabe, V wie Verarbeitung und A wie Ausgabe).

Listing KREIS

```
 10 PROGRAM KREISBERECHNUNG; (*PRG.KOPF*)
 20 CONST PI=3.1459; (*BEGINN VEREINB.
    TEIL*)
 30 VAR RADIUS,UMFANG,FLAECHE:REAL;
 40 BEGIN (* BEGINN ANWEISUNGSTEIL*)
 50  WRITE('Radius in mm=');
 60  READ(RADIUS);
 70  UMFANG:=2*PI*RADIUS;
 80  FLAECHE:=PI*SQR(RADIUS);
 90  WRITELN('Kreisumfang=',UMFANG:9:3,
     ' mm');
100  WRITELN('Kreisflaeche=',
     FLAECHE:8:3,' mm^2');
110  WRITELN
120 END.
```

Wir wollen nun versuchen, das Programm zum Laufen zu bringen. Dazu sind der Compiler und anschließend das Quellprogramm KREIS zu laden. Nach dem Compilieren kann das entstandene Objektcodeprogramm abgearbeitet werden. Wir gehen am besten nach folgendem Handlungsablauf vor:

◊ KC-PASCAL-Compiler von der Kassette mit entsprechendem Kommando laden. Beim KC 85 erfolgt das über LOAD, beim KC 87 über Eingabe des Programmnamens PASCAL.

◊ Der Compiler ist selbststartend und meldet sich mit der Ausschrift „Top of RAM?". Diese Frage wird als Standard mit der ENTER-Taste beantwortet. Auch die folgenden Fragen „Top of RAM for 'T'?" und „Table Size?" quittieren wir mit der ENTER-Taste. Wir wollen uns damit jetzt nicht aufhalten.

◊ Nun meldet sich der Editor des Compilers mit einem Größerzeichen, das generell als Promptzeichen dient. Jetzt ist das Quellprogramm mit dem Namen KREIS einzulesen. Dazu geben wir folgendes ein: Ein großes G, dann zwei Kommas und das Wort KREIS in Großbuchstaben. Das G erinnert an das englische Wort get = erhalten, bekommen. Beim Ertönen des Vortones vom Kassettenrecorder wird dann wie gewohnt die ENTER-Taste betätigt.

◊ Wenn das Programm eingelesen ist, dann bewirken der Großbuchstabe L und die ENTER-Taste ein Listen des Quellprogramms auf dem Bildschirm. Führen wir nun einen Startversuch mit dem Großbuchstaben R, das erinnert an RUN, durch, dann passiert gar nichts, da das Quellprogramm ja noch nicht compiliert wurde.

◊ Das Compilieren erfolgt nach Drücken der Taste C, gefolgt von ENTER. Wenn der Übersetzungslauf fehlerfrei vollzogen wurde, erscheint die Frage auf dem Bildschirm, ob das Programm gestartet werden soll. Die Antwort mit der Taste Y steht für YES und der Programmstart wird vollzogen.

◊ Nun kann das Programm beliebig oft mit der Taste R gestartet und abgearbeitet werden.

◊ Eine Rückkehr in das Betriebssystem des Computers ist mit der Taste B möglich. Ein gewünschter Warmstart des Compilers mit Erhalt von Quell- und Objektcodeprogramm erfolgt mit dem Aufruf PASOLD. Ein Kaltstart mit Löschung aller Programme wird durch den Aufruf PASNEW bewirkt.

Das PASCAL-Programmgerüst

Das Programmgerüst verdeutlichen wir uns am Listing des Programms KREIS. Dies geschieht durch Eingabe des Kommandos L. Übrigens lassen sich große Listings, die über den Bildschirm rollen, beim KC 87 mit der PAUSE-Taste und beim KC 85 mit der STOP-Taste anhalten und mit beliebiger Taste fortsetzen. Zum Listing sei nochmals erwähnt, daß die Editorzeilennummern nur Korrekturen erleichtern sollen und keine Bedeutung für das eigentliche PASCAL-Programm haben.

Kommentare werden in BASIC mit der Anweisung REM eingeleitet. In PASCAL werden Kommentare in geschweiften Klammern geschrieben, die Bestandteil des amerikanischen Zeichensatzes sind. So steht z. B. unter der Codezahl 123 im deutschen Zeichensatz das kleine ä, im amerikanischen Zeichensatz aber geschweifte Klammer auf. Wir haben weder das eine noch das andere auf unserer Tastatur und deshalb eine Ersatzlösung parat. Sie lautet: Runde Klammer auf mit Multiplikationssternchen. Diese Ersatzlösung funktioniert übrigens auch in Turbo-PASCAL Version 3 für 8-Bit-Rechner, und ich habe sie im Listing verwendet.

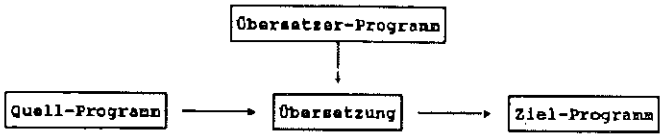
Hier sei auch gleich angemerkt, daß in PASCAL für Indizes eckige Klammern verwendet werden. So steht unter der Codezahl 91 im deutschen Zeichensatz das große Ä und im amerikanischen die öffnende eckige Klammer. Als Ersatzkonstruktion werden wir hier eine runde Klammer, gefolgt von einem Schrägstrich verwenden. In TURBO-PASCAL Version 3 gilt hingegen die Ersatzkonstruktion mit runder Klammer, gefolgt von einem Punkt.

An dem Miniprogramm KREIS ist zu erkennen, daß ein PASCAL-Programm aus dem Progammkopf, dem Vereinbarungsteil und dem Anweisungsteil besteht. Der Programmkopf umfaßt das englische Wort PROGRAM, gefolgt von einem Programmnamen und einem abschließenden Semikolon. Bitte lassen Sie beim Setzen von Leerzeichen und Semikolons besondere Sorgfalt walten, denn sie

sind bei PASCAL im Gegensatz zur Arbeit mit BASIC sehr wichtig.

Ein PASCAL- Programm beginnt also stets mit dem Wort PROGRAM und endet mit einem Punkt nach dem letzten END.

Vereinbarungs- und Anweisungsteil bilden gemeinsam einen sogenannten Block. Ein Programm kann aus vielen solcher Blöcke bestehen. Damit wird das modulare oder, wenn Sie so wollen, das Blockprinzip deutlich. Das Wort Vereinbarungsteil (oft auch Deklarationsteil genannt) weist darauf hin, daß hier all die Dinge, auf die im Anweisungsteil zugegriffen wird, vereinbart oder deklariert werden müssen. In unserem Miniprogramm sind das nur eine Konstante und drei Variablen. Da im KC-PASCAL im Gegensatz zu BASIC und zu Turbo-PASCAL die irrationale Zahl PI nicht enthalten ist, müssen wir sie als Konstante selbst definieren.



Quelle: LOG IN Archiv

Struktur von PASCAL-Programmen.

Das Wort CONST leitet die Konstantenvereinbarung ein. Nach einem Leerzeichen folgt der gewählte Bezeichner PI mit Gleichheitszeichen und dem Wert. Auch hier bitte das Semikolon nicht vergessen.

Das Wort VAR steht vor der Vereinbarung von Variablen. Als Bezeichner wählte ich RADIUS, UMFANG und FLAECHE. Jetzt ist noch der Typ dieser drei Variablen festzulegen. Sinnvoll ist die Arbeit mit reellen Zahlen, deren Typbezeichnung in PASCAL mit dem Wort REAL gekennzeichnet ist. Beachten Sie vor REAL den Doppelpunkt und nach REAL das Semikolon. Hier kurz noch einiges zu den Bezeichnern. Bei den einfachen BASIC-Dialekten war es ja oft unangenehm, daß nur die ersten zwei Zeichen des Bezeichners signifikant unterschieden wurden. Die Bezeichner ABO und ABBA bedeuteten für den BASIC-Interpreter das Gleiche. Unser KC-PASCAL überprüft die Unterschiede in den ersten zehn Zeichen. Damit ist ein schönes Arbeiten mit sinnvollen Bezeichnern möglich. Sie müssen übrigens wie in BASIC mit einem Buchstaben beginnen. Unser KC-PASCAL unterscheidet allerdings im Gegensatz zu Turbo-PASCAL streng zwischen Groß- und Kleinschreibung der Bezeichner. Deshalb schreiben wir in unseren Programmen grundsätzlich alles, mit Ausnahme von Text, groß. In Turbo-PASCAL wird wegen einfacherer Tastatureingabe meist die Kleinschreibung oder eine gemischte Schreibweise benutzt.

Nun ein paar Worte zum Anweisungsteil. Er ist durch BEGIN und END „eingeklammert". Es gibt in PASCAL-Programmen meist viele BEGINs und ENDs. All das, was zwischen einem BEGIN und dem zugehörigen END steht, heißt Verbundanweisung.

Die erste Anweisung in unserem Miniprogramm lautet WRITE, das englische Wort für Schreiben. Das auf den Bildschirm zu Schreibende wird in runde Klammern eingeschlossen und zu schreibender Text in Hochkommas gesetzt. Bei der PRINT-Anweisung in BASIC waren dazu Anführungszeichen zu setzen. Die READ-Anweisung (read = lesen) in PASCAL ist mit der INPUT-Anweisung in BASIC vergleichbar. Der Computer erwartet eine Zahleneingabe, deren Wert der Variablen mit dem Bezeichner RADIUS zugewiesen wird. Jetzt wird in den Editorzeilen 70 und 80 gerechnet. Diese Zuweisung ist uns auch von BASIC bekannt. In PASCAL herrscht aber endlich Ordnung in bezug auf das Zuweisungszeichen. Es besteht aus Doppelpunkt und Gleichheitszeichen. Dadurch sind Verwechslungen mit Überprüfungen auf Gleichheit endgültig beseitigt.

Mit der Buchstabenfolge SQR in der Editorzeile 80 greifen wir etwas vor. Es handelt sich hier um die PASCAL-Funktion SQR. SQR steht für square, dem englischen Wort für Quadrat. Es wird also das Quadrat des Wertes, der unter dem Bezeichner RADIUS abgespeichert ist, gebildet.

Und schon sind wir bei der Ausgabe der Ergebnisse angelangt. In der Editorzeile 90 steht aber nicht das erwartete WRITE, sondern WRITELN. Das LN steht für line, also Zeile, und bedeutet folgendes:

Wenn die Ausgabe der durch runde Klammern eingeschlossenen Informationen abgeschlossen ist, dann wird bei WRITELN ein Wagenrücklauf mit Zeilenschaltung ausgeführt, also das, was wir in BASIC mit PRINT ohne Semikolon gewohnt sind.

Ausgegeben wird zunächst der Text „Kreisumfang=", dann der Inhalt der Variablen mit dem Bezeichner UMFANG und schließlich der Text „mm". Die drei Ausgaben werden durch Kommas getrennt. Die Angabe UMFANG:9:3 dient einer formatierten Ausgabe, ähnlich der PRINT USING-Anweisung bei komfortablen BASIC-Dialekten. In unserem Beispiel wird eine Gesamtstellenzahl von 9 bei 3 Kommastellen angewiesen. Diese Formatierung setzt voraus, daß die Variable UMFANG vom Typ REAL ist. Zu den Formatierungen, die sich ein wenig von Turbo-PASCAL unterscheiden, folgt noch ein gesondertes Beispiel.

Wenn Sie auch die Berechnung von Kreisumfang und Kreisfläche nicht sonderlich begeistern wird, so hoffe ich doch, daß damit der Blick für Zukünftiges ein wenig geschärft worden ist. Unser KC-PASCAL legt uns dabei einige Beschränkungen auf, die aber für den Einsteiger nicht gravierend sind:

- ◊ So können wir nicht mit Diskettenfiles arbeiten. Diese Filearbeit ist eine Stärke von PASCAL, wenn auch nicht ganz leicht zu verstehen.

- ◊ Leider fehlen im KC-PASCAL auch solche Stringmanipulationen wie die Bestimmung der Stringlänge oder das Suchen von Teilstrings in einem String. Diese wertvollen Funktionen beim Umgang mit Text kennen wir vom BASIC her, und sie sind auch in Turbo-PASCAL vorhanden.

Aber wie gesagt, dies ist nicht so tragisch, und wir wollen uns im folgenden noch ein wenig mit unserem Editor beschäftigen.

Der Editor und EVA

Das Auslagern eines PASCAL-Quellprogramms auf Kassette erfolgt mit dem Kommando P wie put, dem englischen Wort für weglegen oder ablegen.. Wie das gemacht wird, entnehmen Sie bitte dem Kasten.

Speichern eines Programms

Folgende Eingabe ist nötig:
P [Anfangszeilennummer] , [Endzeilennummer] , [Programmname mit maximal acht Zeichen]
Beispielsweise geben wir zum Auslagern des Programmes KREIS ein: P 10, 120, KREIS.

Ein VERIFY gibt es leider nicht. Es läßt sich aber auch das Objektcodeprogramm gemeinsam mit dem Laufzeitsystem auslagern, so daß dann auf der Kassette ein Maschinencodeprogramm vorliegt, das ohne Anwesenheit des Compilers abgearbeitet werden kann. Wir kommen später darauf zurück. Mit dem Kommando E, gefolgt von einer Zeilennummer, wird diese Zeile auf den Bildschirm gebracht. Durch Drücken der Leertaste kopieren wir den Zeileninhalt bis zur gewünschten Korrekturstelle. Das Drücken der Taste K wie kill löscht das Zeichen an der Kursorposition, leider ohne Anzeige auf dem Bildschirm. Mit der Taste I lassen sich Zeichen einfügen und mit der Taste C überschreiben. Nach zweimaligem Drücken der ENTER-Taste kann man sich dann das korrigierte Ergebnis anschauen.

In PASCAL gibt es sogenannte Compilersteuerzeichen, die, eingeschlossen in geschweifte Klammern, in das PASCAL-Programm einzufügen sind. Fügen Sie bitte in unser Programm KREIS die Editorzeile 5 mit folgendem Inhalt ein: (* $L- *). Damit wird beim Compilieren das zeitraubende Listen auf dem Bildschirm unterdrückt. In diesem Fall werden nur fehlerhafte Editorzeilen angezeigt. Weitere Steuerzeichen sind in der beiliegenden Kurzdokumentation enthalten.

Wenden wir uns nun einem weiteren EVA-Programm zu. In ihm wird die Körperoberfläche berechnet, nicht die von irgendeiner Eva, sondern Ihre eigene.

Listing EVA

```
 10 PROGRAM KOERPEROBERFLAECHE;
 20 CONST KON1=3207E-7;KON2=0.7285;
    KON3=0.0188;
 30 VAR KOF,MASSE,LAENGE,FAK:REAL;
 40 FUNCTION YHOCHX(Y,X:REAL):REAL;
 50 BEGIN
 60  YHOCHX:=EXP(X*LN(Y));
 70 END;
 80 BEGIN(*HAUPTPROGRAMM*)
 90  WRITE('Masse in kg=');READ(MASSE);
100  WRITE('Laenge in m=');READ(LAENGE);
110  FAK:=MASSE*1000;
120  KOF:=KON1*YHOCHX(FAK,KON2-KON3*
     (LN(FAK)/LN(10)))*
     YHOCHX(LAENGE*100,0.3);
130  WRITE('Koerperob.fl.=',KOF:5:2,
     ' m^2');
140  WRITELN
150 END
```

Das Programm ist wie gewohnt mit G„EVA zu laden. Sollten Sie noch das Quellprogramm KREIS im Speicher haben, so löschen Sie es vorher mit D 10,120. Wenn Sie das Programm EVA geladen haben, dann schauen Sie es sich mit dem Editorkommando L an, compilieren es mit C und starten es beliebig oft mit R. FKK-Freunde können aber auch vorher schätzen, wieviel Quadratmeter sie der Sonne entgegenzusetzen haben.

Der Programmkopf, bestehend aus dem Wort PROGRAM und dem Programmnamen mit abschließendem Semikolon, ist uns schon geläufig. Das trifft prinzipiell auch auf die Vereinbarung von Konstanten und Variablen zu. Allerdings haben wir hier drei Konstanten vereinbart und bei KON1 die Exponentialschreibweise benutzt. Alle vier Variablen sind wieder vom Typ REAL, sollen also als reelle Zahlen behandelt werden.

Neu ist die Vereinbarung einer Funktion in den Editorzeilen 40 bis 70. Eingeleitet wird das Ganze mit dem englischen Wort FUNCTION. Als Funktionsnamen wählen wir YHOCHX, denn diese Funktion soll das Potenzieren mit reellem Exponenten ermöglichen. In den BASIC-Dialekten ist dies schon fest eingebaut, in PASCAL hingegen nicht. In den runden Klammern nach dem Funktionsnamen werden die zu übergebenden Parameter festgelegt. In unserem Fall sind das für die Variablenbezeichner X und Y reelle Zahlen.

Die Bezeichner X und Y gelten nur lokal in dieser Funktion und führen deshalb nicht zu Kollisionen mit eventuell vorhandenen Bezeichnern X und Y im Hauptprogramm. Dies ist ein wesentlicher Vorzug für die Fehlerfreiheit großer PASCAL-Programme.

Nach dem Klammerausdruck folgen noch „Doppelpunkt REAL Semikolon". Das bedeutet, daß auch das Ergebnis aus der Funktionsberechnung, also der Funktionswert, eine reelle Zahl sein soll. Die Verbundanweisung zwischen den Editorzeilen 50 und 70 besteht nur aus einer Zuweisung, die in BEGIN und END eingeschlossen ist. Vor END braucht übrigens kein Semikolon zu stehen. Steht es dennoch wie in unserem Fall, so existiert eine Leeranweisung, die den Compiler aber nicht stört.

Wichtig für die Arbeit mit Funktionen ist, daß das Ergebnis korrekt dem Funktionsnamen, in unserem Fall YHOCHX zugewiesen wird. In der Zuweisung in Zeile 60 werden die Funktionen EXP, also die Eulersche Zahl e hoch einem Wert und der natürliche Logarithmus LN benutzt. Dies ist für den BASIC-Freund schon bekannt.

Das Hauptprogramm bietet kaum Neues. Lediglich in der Editorzeile 120 ist zu sehen, daß die von uns vereinbarte Funktion YHOCHX zweimal aufgerufen wird. Dabei werden jeweils die zwei Parameter, getrennt durch ein Komma, übergeben.

Vereinbarungsteil und Datentypen

Bevor wir uns weitere Programbeispiele ansehen, möchte ich einige weiterführende Bemerkungen zum Vereinbarungsteil machen. Bisher haben wir nur die Vereinbarungen von Konstanten mit CONST, von Variablen mit VAR und von Funktionen mit FUNCTION kennengelernt. Im Vereinbarungsteil können und müssen, sofern benötigt, folgende Dinge vereinbart werden:

◊ Marken mit der Bezeichung LABEL,
◊ Konstanten mit CONST,
◊ Typen mit TYPE,
◊ Variablen mit VAR,
◊ Unterprogramme mit PROCEDURE und
◊ Funktionen mit FUNCTION.

In unserem KC-PASCAL müssen die Vereinbarungen in dieser Reihenfolge festgelegt werden. In Turbo-PASCAL ist diese strenge Reihenfolge nicht erforderlich, aber dennoch empfehlenswert.

Als Datentyp haben wir bisher nur den Typ REAL für reelle Zahlen kennengelernt. Er gehört zu den einfachen Typen. Zu den einfachen Typen gehört auch die Gruppe der Ordinaltypen. Ordinal bedeutet geordnet, und zwar so, daß jedes Element einen eindeutigen Nachfolger und Vorgänger hat.

Ein solcher Ordinaltyp ist der Aufzählungstyp. Als Beispiel sei die Aufzählung der Monate Januar bis Dezember genannt.

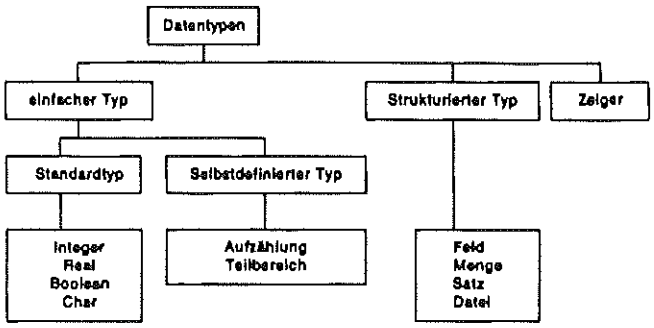
Auch der Unter- oder Teilbereichstyp ist ein Ordinaltyp. So kann man überall dort, wo es um die Verarbeitung von Monatsangaben geht, nur den Zahlenbereich von 1 bis 12 vereinbaren. So etwas kennen wir von BASIC nicht, diese Vereinbarungen und die Kontrolle ihrer Einhaltung haben aber viele Vorteile.

Ein weiterer Ordinaltyp ist der Typ INTEGER. Hier wird nur mit ganzen Zahlen gearbeitet, die ja stets einen Vorgänger und einen Nachfolger haben.

Auch der Typ CHAR ist ein Ordinaltyp. CHAR steht für das englische Wort character, das u. a. Schriftzeichen bedeutet. Über den ASCII-Zeichensatz hat jedes Schriftzeichen Vorgänger und Nachfolger. Wir kennen diese Verschlüsselung im ASCII schon von unserem BASIC-Interpreter.

Ein weiterer Ordinaltyp ist der Typ BOOLEAN. Hier geht es um die Wahrheitswerte wahr oder falsch. Dazu sind in PASCAL direkt die Worte TRUE für wahr und FALSE für falsch vereinbart. In BASIC mußten wir uns mit Zahlen behelfen. Die Null bedeutete falsch und eine wahre Aussage wurde durch –1 oder 1 gekennzeichnet. In PASCAL sind solche Unzulänglickeiten durch ein sauberes Datentypkonzept beseitigt. Den in Turbo-PASCAL möglichen Typ BYTE gibt es in unserem KC-PASCAL nicht. Soweit ein Überblick zu den einfachen Datentypen.

Darüber hinaus kennt PASCAL noch strukturierte und Zeigertypen. Zeiger behandeln wir in diesem Einführungslehrgang nicht, da das schon etwas für Kenner ist. Die strukturierten Typen FILE und STRING gibt es in unserem



Quelle: LOG IN Archiv

Typenkonzept von PASCAL.

KC-PASCAL nicht, wohl aber die Typen ARRAY, RECORD und SET. Das englische Wort array bedeutet Feld und kennzeichnet ein Datenfeld von gleichartigen Komponenten. Ein Record hingegen ist eine Zusammenstellung verschiedenartiger Komponenten. So etwas gibt es aber in BASIC überhaupt nicht. Records sind sehr leistungsfähige Typen, vor allem wenn es um Dateiarbeit geht.

Im folgenden wollen wir uns das Programm FORMAT ansehen.

Listing FORMAT

```
 10 PROGRAM ZAHLFORMAT;
 20 VAR AUS:REAL;
 30 BEGIN
 40  AUS:=-EXP(1);
 50  WRITELN('ohne Format. *',AUS,'*');
 60  WRITELN('Format.:5   *',AUS:5,'*');
 70  WRITELN('Format.:10  *',AUS:10,'*');
 80  WRITELN('Format.:14  *',AUS:14,'*');
 90  WRITELN('Format.:5:0 *',AUS:5:0,
     '*');
100  WRITELN('Format. :7:3 *',AUS:7:3,
     '*')
110 END.
```

Das Listing macht deutlich, wie die Formatierung von reellen Zahlen auf dem Bildschirm zu handhaben ist. Die Zahl, die sich aus –EXP(1) ergibt, soll hier nur als Beispiel dienen. Wenn Sie das Programm nach dem Compilieren starten, dann erscheinen die formatierten Ausgaben zwischen zwei Sternchen und machen den Stellenbedarf einschließlich Minuszeichen und Komma deutlich. Versuchen Sie doch einmal, das Programm so umzuschreiben, daß Sie über eine READ-Anweisung jede beliebige reelle Zahl eingeben können, die dann formatiert wieder ausgegeben wird.

Das nächste Programmbeispiel mit dem Namen GROSS überprüft die Eingabe eines Zeichens daraufhin, ob es sich um einen Großbuchstaben handelt. Die hier vorgestellte Überprüfungsart gibt es in BASIC nicht.

Listing GROSS

```
10 PROGRAM GROSSBUCHSTABE;
20 VAR ALPHA:SET OF 'A'..'z';
30     EIN:CHAR;
40 BEGIN
50  ALPHA:=(/'A'..'Z'/);
60  WRITE('EINGABE= ');READ(EIN);
70  IF EIN IN ALPHA THEN WRITELN(EIN,
    'ist ein Grossbuchstabe')
80 END.
```

Wir arbeiten hier mit dem Datentyp SET. Set ist das englische Wort für Menge (Satz). In unserem Fall handelt es sich um eine Menge von Zeichen, die gemäß ASCII zwischen dem großen A und dem kleinen z liegen. Die SET-Variable ALPHA kann also nur eines dieser Zeichen enthalten. Im Anweisungsteil in der Editorzeile 50 legen wir dann fest, daß ALPHA nur Großbuchstaben enthalten darf. Beachten Sie bitte, daß die Zeichen A und Z in Hochkommas eingeschlossen und durch zwei Punkte getrennt sind. Das Ganze steht in eckigen Klammern, wobei hier die Ersatzkonstruktion „(/" benutzt wird. In der Editorzeile 60 geben wir ein Zeichen ein, daß der Variablen EIN vom Typ CHAR zugewiesen wird. Hier fällt auf, daß vor die Anweisung READ(EIN) noch die Anweisung READLN gesetzt wurde. Das ist in unserem KC-PASCAL bei der Eingabe von Zeichen zum Löschen des Eingabepuffers erforderlich. Turbo-PASCAL verhält sich da etwas anders. Mit der Editorzeile 70 greifen wir wieder etwas vor, da hier eine IF-THEN-Anweisung abgearbeitet wird. Aber die Unterschiede zu BASIC sind gering. Den Wahrheitswert TRUE oder FALSE liefert die Wortfolge EIN IN ALPHA. Wenn man diese Zeile 70 wörtlich ins Deutsche übersetzt, dann wird ihre Bedeutung sofort klar.

Eine weitere Möglichkeit einer wirkungsvollen Überprüfung von Tastatureingaben möchte ich mit dem folgendem Programm EINTEST vorstellen.

Listing EINTEST

```
10 PROGRAM EINTEST;
20 VAR EIN:CHAR;
30 BEGIN
40  REPEAT
50   WRITE('EINGABE (J/N)? ');
60   READLN;READ(EIN)
70  UNTIL EIN IN (/'J','j','N','n'/);
80  WRITELN('Eingabe in Ordnung')
90 END.
```

In diesem Miniprogramm verharrt der Computer solange im Eingabemodus, bis der Klein- oder Großbuchstabe J oder N eingegeben wird. Wir benutzen dazu eine REPEAT-UNTIL-, also Wiederhole-bis-Konstruktion. In Editorzeile 70 ist zu erkennen, daß die Schleife zwischen REPEAT und UNTIL solange wiederholt wird, bis der Inhalt der Variablen mit dem Bezeichner EIN Bestandteil der Menge J, j, N und n ist. Achten Sie auch hier wieder auf die eckigen Klammern, in die diese Menge eingeschlossen ist.

Die Modulo-Rechnung

Die ganzzahlige Division mit DIV und die Modulorechnung mit MOD kennen einfache BASIC-Interpreter nicht. Die Modulo-Rechnung liefert den Rest einer Division als Ergebnis. So ist z. B. 17 MOD 5 gleich 2. Sie werden jetzt vielleicht fragen, wozu so etwas gebraucht wird. Das folgende Programmbeispiel zeigt ein Beispiel. Wir wollen die Osterformel von Gauß vorstellen, mit der für jedes beliebige Jahr das Datum des Ostersonntages ermittelt werden kann. Das gilt übrigens seit dem Konzil von Nizäa im Jahre 325. Dort wurde festgelegt, daß der Ostersonntag der erste Sonntag nach Vollmond und Frühlingsanfang ist. Gauß hat daraus einen Algorithmus entwickelt, der im Programm OSTERN enthalten ist.

Listing OSTERN

```
 10 PROGRAM OSTERFORMEL;
 20 VAR JAHR,A,B,C,D,E:INTEGER;
 30 BEGIN
 40  WRITE('Jahr=');READ(JAHR);
 50  A:=JAHR MOD 4;
 60  B:=JAHR MOD 7;
 70  C:=JAHR MOD 19;
 80  D:=(19*C+24) MOD 30;
 90  E:=(2*A+4*B+6*D+5) MOD 7;
100  IF ((D=29) AND (E=6) OR ((D=28) AND
     (E=6) AND (C>0))
105  THEN D:=D-7;
110  WRITE('Ostersonntag ist der ');
120  IF (22+D+E<=31)
130   THEN WRITELN(22+D+E:2,'. Maerz')
140   ELSE WRITELN(D+E-9:2,'. April')
150 END.
```

Wir arbeiten hier erstmalig mit dem Datentyp INTEGER, also mit ganzen Zahlen. Ob die Modulo-Rechnung auch mit reellen Zahlen funktionieren kann? Vielleicht probieren Sie es einmal aus. Wir gehen folgendermaßen vor. Nachdem das Quellprogramm OSTERN geladen wurde, lassen wir uns mit der Tastenfolge „E 20 ENTER" die Editorzeile 20 auf dem Bildschirm anzeigen. Dabei wird die Zeilennummer noch einmal darunter geschrieben, und mit der Leertaste kopieren wir den Zeileninhalt bis zum Doppelpunkt vor dem Typ INTEGER. Jetzt drücken wir die Taste C. Anschließend überschreiben wir das Wort INTEGER mit dem Wort REAL. Wir drücken einmal die ENTER-Taste und verlassen damit den Überschreibmodus. Durch dreimaliges Drücken der Kill-Taste K werden die restlichen Buchstaben GER vom Wort INTEGER gelöscht.

Mit Betätigung der ENTER-Taste wird die korrigierte Zeile in das Quellprogramm übernommen. Jetzt compilie-

ren wir das korrigierte Quellprogramm durch Drücken der Taste C. Aber es folgt die Fehlermeldung 10, die einen falschen Typ signalisiert. Der Compiler hat also einen Typkonflikt erkannt, die Modulorechnung ist mit reellen Zahlen nicht möglich. Nach Anzeige der Fehlermeldung drücken Sie wieder die Taste E, worauf vom Editor die fehlerhafte Zeile zur Korrektur angeboten wird.

Funktionsangebote

Zur Sichtung der vom KC-PASCAL angebotenen Funktionen betrachten wir das Programm ARITH.

Listing ARITH

```
 10 PROGRAM ARITHFUNKTIONEN;
 20 VAR EIN:REAL;
 30 BEGIN
 40  WRITE('Zahl=');READ(EIN);
 50  WRITELN('ABS    =',ABS(EIN));
 60  WRITELN('TRUNC  =',TRUNC(EIN));
 70  WRITELN('ROUND  =',ROUND(EIN));
 80  WRITELN('ENTIER =',ENTIER(EIN));
 90  WRITELN('FRAC   =',FRAC(EIN))
100 END.
```

Wir vereinbaren in der Editorzeile 20 eine Variable mit dem Bezeichner EIN für eine reelle Zahl. Im Anweisungsteil kann mit READ(EIN) eine beliebige reelle Zahl eingegeben werden. Die folgenden Ausgaben zeigen die Wirkungen von insgesamt fünf Konvertierungsfunktionen im KC-PASCAL.

◊ ABS liefert den Absolutwert und ist uns von BASIC bekannt.

◊ TRUNC ist die Abkürzung des englischen Wortes truncated und bedeutet abgeschnitten. Hier wird wirklich abgeschnitten und nicht abgerundet wie bei INT in BASIC. Sie können das sehr leicht durch die Eingabe einer negativen Zahl überprüfen.

◊ Die Funktion ROUND rundet nach den üblichen Rundungsregeln auf eine ganze Zahl.

◊ Das französische Wort ENTIER entspricht dem englischen Wort entire und bedeutet soviel wie ganz. Hier wird stets auf eine ganze Zahl abgerundet. Diese Funktion entspricht damit dem uns bekannten INT in in der Programmiersprache BASIC.

◊ Die Funktion FRAC kennen Sie vielleicht von einem guten Taschenrechner. Damit wird der Nachkommateil einer Zahl ermittelt, übrigens mit einem kleinen Fehler in der letzten Stelle.

An der Vielzahl von Konvertierungsfunktionen sehen Sie schon, daß eine ganze Menge in unserem PASCAL-Compiler steckt. Schade ist nur, daß er lediglich eine Genauigkeit von sechs Stellen bietet. Turbo-PASCAL in der Version 3 für 8-Bit-Rechner arbeitet hingegen mit 11 Stellen. Und mit den Versionen 4 und 5 für 16-Bit-Rechner kann man sogar den Datentyp EXTENDED mit zwanzigstelliger Genauigkeit vereinbaren. Damit sind astronomische und ökonomische Berechnungen kein Problem mehr.

Wir wollen uns nun weiteren Funktionen zuwenden und erstmals einen Typ vereinbaren. Dazu betrachten wir das Programm ORD.

Listing ORD

```
 10 PROGRAM ORDFUNKTION;
 20 TYPE TAG=(MON,DIE,MIT,DON,FRE,
    SAM,SON);
 30 VAR GESTERN,HEUTE,MORGEN:TAG;
 40 BEGIN
 50  HEUTE:=FRE;
 60  GESTERN:=PRED(HEUTE);
 70  MORGEN:=SUCC(HEUTE);
 80  WRITELN(ORD(HEUTE),ORD(GESTERN),
     ORD(MORGEN));
 90  WRITELN;
100  WRITELN(ORD('A'),'***',CHR(65))
110 END.
```

In der Editorzeile 20 vereinbaren wir einen Typ mit dem Bezeichner TAG, gefolgt von einem Gleichheitszeichen. Wir vereinbaren hier einen sogenannten Aufzählungstyp. Die einzelnen Elemente des Typs, in unserem Fall die Tage MON bis SON, werden durch Kommas getrennt. Das Ganze wird in runde Klammern eingeschlossen. Damit haben wir eine geordnete Aufzählung bewirkt. Für den Rechner ist MON das nullte Element, DIE das erste, MIT das zweite usw. Das eröffnet viele Programmiervorteile, die wir vom BASIC her nicht kennen.

In der Editorzeile 30 vereinbaren wir drei Variablen, die vom Typ TAG sind, also nur den Inhalt MON oder DIE usw. haben können. Im Anweisungsteil weisen wir der Variablen mit dem Bezeichner HEUTE das Element FRE zu. Wir nutzen jetzt die PASCAL-Funktionen PRED und SUCC. PRED ist die Abkürzung des englischen Wortes predecessor und bedeutet Vorgänger. Der Vorgänger von FRE ist DON. Dieses Element DON wird der Variablen mit dem Bezeichner GESTERN zugewiesen. SUCC ist die Abkürzung des englischen Wortes successor und bedeutet Nachfolger. Damit wird das Element SAM als Nachfolger von FRE der Variablen mit dem Bezeichner MORGEN zugewiesen.

Leider, und das ist wirklich außerordentlich bedauerlich, können die Elemente MON bis SON des Aufzählungstypes nicht mit WRITE auf Bildschirm oder Drucker ausgegeben und ebensowenig über READ eingegeben werden, da sie nur intern im Rechner existieren. Wir können uns aber mit Hilfe der Funktion ORD von der korrekten Arbeitsweise überzeugen. ORD liefert uns nämlich die Ordnungszahl

des jeweiligen Elementes. Wir erwähnten schon, daß die Zählung bei Null beginnt. Die Editorzeile 80 liefert also die Ordnungszahlen 4 für Freitag, 3 für Donnerstag und 5 für Samstag.

Bitte experimentieren Sie mit der Editorzeile 50, indem Sie statt FRE auch die anderen Elemente MON, DIE usw. zuweisen. Sie merken dann, daß die Sache bei MON und SON nicht mehr exakt arbeitet. Das liegt daran, daß MON keinen definierten Vorgänger und SON keinen definierten Nachfolger hat. Vielleicht fragen Sie jetzt nach dem Sinn der ganzen Angelegenheit. Bitte gedulden Sie sich bis zum zu unserem nächsten Programmbeispiel.

Vielleicht ist Ihnen im Zusammenhang mit der ORD-Funktion auch die Funktion ASC aus BASIC ins Gedächtnis gerückt. Das ist genau richtig. In der Editorzeile 80 liefert nämlich ORD von dem Großbuchstaben A den bekannten ASCII 65. ORD liefert demnach nicht nur die Ordnungszahlen für die von uns selbst vereinbarten Ordinaltypen, sondern auch für den ASCII. Und damit kann man schon eine ganze Menge anfangen.

Die Funktion CHR übrigens ist vergleichbar mit der Funktion CHR$ in BASIC.

Das erste richtige Programm

Schauen wir uns jetzt das etwas umfangreichere Programm TAGE an.

Listing TAGE

```
 10 PROGRAM WOCHENTAGE;
 20 TYPE TAG=(MO,DI,MI,DG,FR,SA,SO);
 30 VAR GESTERN,HEUTE,MORGEN:TAG;
 40     EINGABE:ARRAY(/1..6/) OF CHAR;
 50     I:INTEGER;
 60 PROCEDURE AUSGABE(AUSGABE:TAG);
 70 BEGIN
 80  CASE AUSGABE OF
 90   MO:WRITE('Montag');
100   DI:WRITE('Dienstag');
110   MI:WRITE('Mittwoch');
120   DG:WRITE('Donnerstag');
130   FR:WRITE('Freitag');
140   SA:WRITE('Sonnabend');
150   SO:WRITE('Sonntag');
160  END; (*END von CASE*)
170 END;
180 BEGIN (*HAUPTPROGRAMM*)
190  PAGE;
200  WRITE('Eingabe des Tages: ');
210  READLN;READ(EINGABE);
220  WRITELN;WRITELN;
230  IF EINGABE='MONTAG' THEN HEUTE:=MO;
240  IF EINGABE='DIENST' THEN HEUTE:=DI;
250  IF EINGABE='MITTWO' THEN HEUTE:=MI;
260  IF EINGABE='DONNER' THEN HEUTE:=DG;
270  IF EINGABE='FREITA' THEN HEUTE:=FR;
280  IF EINGABE='SONNAB' THEN HEUTE:=SA;
290  IF EINGABE='SONNTA' THEN HEUTE:=SO;
300  IF ORD(HEUTE)>6 THEN BEGIN
     WRITE('FEHLEINGABE!');HALT END;
310  IF HEUTE=SO THEN MORGEN:=MO ELSE
     MORGEN:= SUCC(HEUTE);
320  IF HEUTE=MO THEN GESTERN:=SO ELSE
     GESTERN:=PRED(HEUTE);
330  WRITELN('Wenn der gewuenschte Tag
     ein ');
340  AUSGABE(HEUTE);
350  WRITELN(' ist, dann war der Vortag
     ein');
360  AUSGABE(GESTERN);
370  WRITELN(' und der nachfolgende Tag
     ein');
380  AUSGABE(MORGEN);
390  WRITELN('.')
400 END.
```

Das Programm soll folgendes leisten: Nach korrekter Eingabe eines Wochentages sollen der eingegebene Tag, der Vortag und der nachfolgende Tag ausgegeben werden. Mit dem vorhin besprochenen Programm ORD haben wir schon einige Vorarbeit geleistet, aber das reicht noch nicht. In BASIC müßte man die Sache ganz ohne die Vereinbarung eines Aufzählungstyps und ohne die Funktionen PRED und SUCC realisieren. Trotz Nutzung dieser Möglichkeiten in PASCAL ist das Programm doch recht lang. Immerhin macht der Vereinbarungsteil bis zur Editorzeile 170 fast die Hälfte des gesamten Quellprogramms aus. Das ist keineswegs ungewöhnlich. Dem modularen Konzept entsprechend werden im allgemeinen viele Module als Funktionen oder Prozeduren, also Unterprogramme, im Vereinbarungsteil definiert. Der Anweisungsteil besteht dann im wesentlichen nur aus den Aufrufen der Module und ist damit relativ kurz.

Nun aber zu unserem Programm TAGE. Wir vereinbaren wiederum einen Aufzählungstyp mit dem Bezeichner TAG. Vielleicht fällt Ihnen auf, daß der Donnerstag nicht mit DO sondern mit DG vereinbart wird. Wie in BASIC, so gibt es auch in PASCAL vordefinierte Wörter, die man nicht anderweitig verwenden darf. Dazu gehört auch das englische Wort do, also tue, mache.

In der Editorzeile 40 vereinbaren wir unter dem Bezeichner EINGABE ein Feld, ARRAY genannt, mit insgesamt 6 Zeichen. Beachten Sie, daß die Kombinationen „(/“ und „/)“ als Ersatz für die eckigen Klammern dienen. Hier kann also ein String mit maximal 6 Zeichen aufbewahrt werden.

Werden z. B. bei einer Eingabe mehr als 6 Zeichen eingegeben, so werden sie einfach ignoriert.

Diese statische Stringbehandlung ist auch als Ausnahme beim BASIC-Interpreter des ZX Spectrum anzutreffen. Die meisten BASIC-Interpreter reagieren aber dynamisch und nehmen den gesamten String mit variabler Länge auf. Eine solche dynamische Behandlung gibt es auch in PASCAL, sie ist aber für den Anfänger etwas kompliziert. Die statische und die dynamische Stringverarbeitung haben

Vor- und Nachteile. Übrigens hätte man in Turbo-PASCAL statt ARRAY(/1..6/) OF CHAR einfach STRING(/6/) geschrieben, aber den Typ STRING kennt unser KC-PASCAL nicht.

In den Editorzeilen 60 bis 170 begegnen wir das erste Mal einer Prozedur, also einem Unterprogramm. Beachten Sie die englische Schreibweise von PROCEDURE. Der Name der Prozedur lautet AUSGABE und in den folgenden runden Klammern wird die Parameterübergabe geklärt. Der lokal, also nur in der Prozedur gültige Bezeichner heißt AUSGABE und ist vom Typ TAG. Diese Ausgabeprozedur ist leider nötig, da, wie schon erwähnt, die als Aufzählungstyp vereinbarten Elemente weder über READ noch über WRITE zugänglich sind.

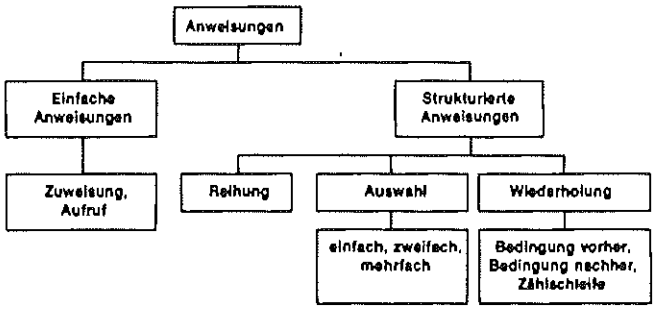
Die Prozedur besteht im wesentlichen aus einer CASE-Anweisung. Case ist das englische Wort für Fall im Sinne eines Umstands oder Vorfalls. Es handelt sich also um eine Fallauswahl, die wir in BASIC mit ON GOTO oder ON GOSUB erledigen. Wenn z. B. der Inhalt der lokalen Variablen AUSGABE FR lautet, dann erscheint der Text Freitag durch die WRITE-Anweisung auf dem Bildschirm. Übrigens wird im Hauptprogramm dafür gesorgt, daß der Prozedur AUSGABE keine unerlaubten Elemente übergeben werden.

Jede CASE-Anweisung wird mit einem END abgeschlossen. Das ist einer der wenigen Fälle, wo es zu einem END kein BEGIN, sondern hier eben das Wort CASE gibt.

Das Hauptprogramm enthält als erste Anweisung PAGE, das englische Wort für Seite. Damit wird der Bildschirm gelöscht. In Turbo-PASCAL würden wir dafür CLRSCR, die Abkürzung für clear screen schreiben.

Die Editorzeilen 200 bis 290 bedürfen wohl keiner Erklärung mehr. In Zeile 300 wird eine eventuelle Fehleingabe abgeblockt und der Programmablauf mit der HALT-Anweisung gestoppt. Das ist nicht sonderlich elegant, soll uns im Moment aber erst einmal genügen. In der Editorzeile 310 wird der Nachfolger ermittelt. Da der Nachfolger von SO nicht definiert ist, wird MO direkt zugewiesen. In allen anderen Fällen kommt die Funktion SUCC zur Anwendung. Dies gilt sinngemäß für die Festlegung des Vorgängers in Zeile 320. In den Zeilen 340, 360 und 380 wird dann jeweils die Prozedur AUSGABE aufgerufen und der entsprechende Parameter übergeben.

Vielleicht experimentieren Sie noch ein wenig mit dem Programm. Was geschieht z. B., wenn Sie Samstag statt Sonnabend eingeben? Verändern Sie das Programm so, daß auch Samstag akzeptiert wird. Prinzipiell würde auch die Eingabe der ersten drei Buchstaben der einzelnen Wochentage reichen. Vielleicht probieren Sie auch das einmal aus.



Quelle: LOG IN Archiv

PASCAL-Anweisungen.

Grundstrukturen der Programmierung

Vielleicht hat Sie an unserem KC-PASCAL-Kurs bisher gestört, daß ich etwas unsystematisch vorgegangen bin. Denn normalerweise führt man die Grundstrukturen einer Programmiersprache in der Reihenfolge

◊ Sequenz,
◊ Auswahl,
◊ Wiederholung und
◊ Unterprogramm

ein. Hier haben wir schon in einigen Beispielprogrammen vorgegriffen, was aber für den BASIC-Kenner nicht problematisch ist. So ist bei PASCL eine ein-, zwei- oder mehrseitige Auswahl möglich. In BASIC existieren dazu die Anweisungen IF THEN, IF THEN ELSE und ON GOTO oder ON GOSUB. In PASCAL lauten diese Anweisungen IF THEN, IF THEN ELSE und CASE OF, die wir alle schon in Beispielprogrammen benutzt haben.

Bei den Wiederholungen bieten Programmiersprachen im allgemeinen die Zählschleife, die Solangeschleife (auch vorprüfende Schleife genannt) und die Wiederholschleife (auch nachprüfende Schleife genannt). Die einfachen BASIC-Interpreter kennen nur die Zählschleife mit der Anweisungsfolge FOR TO NEXT. Komfortable BASIC-Interpreter, PASCAL und andere Sprachen arbeiten mit WHILE DO für die vorprüfende Schleife und REPEAT UNTIL für die nachprüfende Schleife. Die REPEAT-UNTIL-Konstruktion wurde schon im Programmbeispiel EINTEST benutzt.

Die Arbeit mit Unterprogrammen ist in BASIC nicht sehr glücklich organisiert. Ein Unterprogramm wird mit GOSUB und Zeilennummer aufgerufen, das Unterprogramm selbst mit RETURN beendet. In PASCAL ist dies sehr elegant über Prozeduren organisiert. Im Programm TAGE haben wir als Beispiel eine Prozedur mit dem Namen AUSGABE benutzt. In den folgenden Beispielen werden wir den Umgang mit diesen Grundstrukturen üben.

Ein Schmunzelprogramm

Beginnen wir mit einem Programmbeispiel zum Schmunzeln. Stellen Sie sich bitte vor, Sie finden als Gast vor der Gaststättentür nicht das früher so beliebte Schild „Sie werden plaziert", sondern Bildschirm und Tastatur vor. Sie werden aufgefordert, folgende vier Fragen zu beantworten. Sind Sie hungrig? Sind Sie durstig? Sind Sie Trinkgeldzahler? Sind Sie betrunken? Je nachdem wie die Antworten auf diese vier Fragen ausfallen, wird Ihnen der Eintritt ge-

währt oder nicht. Der Wirt in unserem Beispiel ist offenbar nicht nur ein guter Rechner, sondern er kennt sich auch in Aussagenlogik und der Programmiersprache PASCAL aus. Als Antworten sollen nur die Großbuchstaben J und N zugelassen werden. Hier werden wir unser Beispielprogramm EINTEST zu Rate ziehen. Diese Eingabekontrolle ist sehr wichtig, damit nicht die ganze Entscheidungslogik durcheinander gerät. Antwortvarianten wie „Na klar", das englische „Yes" oder das russische „Da" dürfen nicht akzeptiert werden.

Listing GAST

```
 10 PROGRAM GASTSTAETTE;
 20 VAR H,D,T,B:CHAR;
 30     ALPHA:SET OF CHAR;
 40 BEGIN (*HAUPTPROGRAMM*)
 50  ALPHA:=('J','N');(*SETKONSTANTE*)
 60  PAGE;
 70  WRITELN('Beantworten Sie bitte die
     Fragen');
 80  WRITELN('mit J oder N !');
 90  WRITELN;
100  REPEAT
110   WRITE(' Hungrig ? ');READ(H)
120  UNTIL H IN ALPHA;
130  REPEAT
140   WRITE(' Durstig ? ');READ(D)
150  UNTIL D IN ALPHA;
160  REPEAT
170   WRITE(' Trinkgeld ? ');READ(T)
180  UNTIL T IN ALPHA;
190  REPEAT
200   WRITE(' Betrunken ? ');READ(B)
210  UNTIL B IN ALPHA;
220  WRITELN;
230  IF ((H='J') OR (D='J')) AND (T='J')
     AND (B='N')
240   THEN WRITELN('Kommen Sie bitte
      herein')
250   ELSE WRITELN('Sie duerfen nicht
      eintreten')
260 END.
```

Schauen wir uns nun das Listing des Quellprogramms GAST an. Die Variablen mit den Bezeichnern H, D, T und B sind vom Typ CHAR, können also nur ein Zeichen enthalten. H steht übrigens für hungrig, D für durstig, T für Trinkgeldzahler und B für betrunken. Die Variable mit dem Bezeichner ALPHA ist vom Typ SET OF CHAR, kann also zunächst alle Zeichen des ASCII-Zeichensatzes enthalten. Das schränken wir im Anweisungsteil in der Editorzeile 50 sofort ein. Hier wird nun festgelegt, daß die Setkonstante ALPHA nur die Großbuchstaben J oder N enthalten darf. So ähnlich sind wir im Programm GROSS vorgegangen. Die Eingaben werden wie im Programm EINTEST mit einer REPEAT-UNTIL-Konstruktion realisiert. Hier verharrt der Computer solange bei der jeweiligen Eingabeaufforderung, bis entweder ein großes J oder ein großes N eingegeben wird. Die gesamte Entscheidungslogik steckt in der Editorzeile 230. Die Kombination der ORs und ANDs machen die Intentionen des Wirtes deutlich. Es darf nur derjenige eintreten, der hungrig oder durstig oder beides ist, wegen des Umsatzes. Außerdem muß er Trinkgeldzahler sein, darf sich aber nicht als betrunken einschätzen, was ja nicht immer leicht zu beantworten ist. Beachten Sie bitte die runden Klammern, die bei solchen logischen Ausdrükken in PASCAL zu setzen sind. Wird die erste Klammer, die den OR-Vergleich zwischen hungrig und durstig einschließt vergessen, dann stimmt aus Prioritätsgründen die Logik nicht mehr, denn AND hat eine höhere Priorität als OR. Aber auch das ist von BASIC her schon bekannt. Vielleicht würden Sie als Wirt anders entscheiden wollen. Bitte sehr, experimentieren Sie mit diesem Programm.

Kalendarisches

Im folgenden nun ein Kurzprogramm mit dem Namen QUARTAL, das noch einmal die Anwendung der CASE-Anweisung verdeutlichen soll.

Listing QUARTAL

```
 10 PROGRAM QUARTAL;
 20 VAR MONAT:INTEGER;
 30 BEGIN;
 40  WRITE('MONAT ALS ZAHL=');
     READ(MONAT);
 50  CASE MONAT OF
 60   1,2,3:WRITELN('I. QUARTAL');
 70   4,5,6:WRITELN('II. QUARTAL');
 80   7,8,9:WRITELN('III. QUARTAL');
 90   10,11,12:WRITELN('IV. QUARTAL');
100   ELSE WRITELN('Falsche Eingabe')
110 END.
```

Wird in diesem Programm der Monat als Zahl eingegeben, dann erfolgt dessen Zuordnung zu einem Quartal, das auf dem Bildschirm angezeigt wird. Die Variable mit dem Bezeichner MONAT wird deshalb als ganze Zahl, also als INTEGER-Typ, vereinbart. In der Editorzeile 40 wird der Monat als Zahl eingegeben. Die CASE-Anweisung berücksichtigt zunächst die Zahlen 1 bis 12. Ist der Inhalt von MONAT z. B. 4, 5 oder 6, so wird der Text „II. Quartal" ausgegeben. Sie erkennen im Listing, daß die Bedingungen 4, 5 und 6 jeweils durch Kommas getrennt sind. Nach dem Doppelpunkt folgt dann die auszuführende Anweisung. Meist sind das aber mehrere Anweisungen, die nach dem Doppelpunkt abzuarbeiten sind. Das ist in PASCAL kein Problem, denn mit BEGIN und END würden Sie daraus eine Verbundanweisung machen.

Ein Problem ist noch die Sicherung gegen Fehleingaben. Hier benutzen wir den ELSE-Zweig der CASE-Anweisung, so wie es die Editorzeile 100 zeigt. Aufmerksame Leser werden jetzt vielleicht das END der CASE-Anweisung vermissen. Das ist allerdings eine kleine Ungereimtheit des KC-PASCAL-Compilers. Tritt ein ELSE-Zweig bei CASE auf, dann darf hier kein END gesetzt werden. In Turbo-

PASCAL hingegen muß immer das END als Abschluß einer CASE-Anweisung stehen.

Das folgende Programm WOCHTAGE wurde dem Buch von Goldammer: „PASCAL für die Anwendung in der Wirtschaft" entnommen. Dieses Buch kann ich übrigens wärmstens empfehlen, da es aus meiner Sicht für den Anfänger methodisch hervorragend aufgebaut ist.

Listing WOCHTAGE (DATUM)

```
 10 PROGRAM WOCHENTAGE;
 20 (*NACH GOLDAMMMER: PASCAL...*)
 30 VAR TAG  :1..31;
 40     MONAT:1..12;
 50     JAHR :1583..MAXINT;
 60     ZEIT :INTEGER;
 70 (*TEILBEREICHE IN KC-PASCAL LEIDER
    NICHT UEBERPRUEFBAR*)
 80 BEGIN
 90  PAGE;
100  WRITELN('Ermittlung des Wochentages
     ab 1583');
110  WRITELN('ACHTUNG! Keine Kontrolle
     auf falsche')
120  WRITELN('Tagesangaben in den
     Monaten Februar,')
130  WRITELN('April, Juni, September und
     November')
140  WRITE('  Tag als Zahl=');READ(TAG);
150  WRITE('Monat als  Zahl=');
     READ(MONAT);
160  WRITE(' Jahr als Zahl=');
     READ(JAHR);
170  ZEIT:=JAHR MOD 100;
180  ZEIT:=ZEIT*(365 MOD 7)+ZEIT DIV 4;
190  ZEIT:=ZEIT+(MONAT-1)*30+TAG;
200  IF (JAHR MOD 4=0) AND (MONAT<=2)
     THEN ZEIT:=ZEIT-1;
210  CASE MONAT OF
220   2,6,7:ZEIT:=ZEIT+1;
230   3    :ZEIT:=ZEIT-1;
240   8    :ZEIT:=ZEIT+2;
250   9,10 :ZEIT:=ZEIT+3;
260   11,12:ZEIT:=ZEIT+4;
270  END; (*END OF CASE*)
280  WRITE('Dieser Tag ist oder war
     ein ');
290  CASE (ZEIT MOD 7)+1 OF
300   1:WRITELN('Sonntag');
310   2:WRITELN('Montag');
320   3:WRITELN('Dienstag');
330   4:WRITELN('Mittwoch');
340   5:WRITELN('Donnerstag');
350   6:WRITELN('Freitag');
360   7:WRITELN('Sonnabend');
370  END; (*END OF CASE*)
380 END.
```

Werfen wir einen Blick auf das schon umfangreiche Listing. In den Editorzeilen 30 bis 50 vereinbaren wir erstmalig Teilbereichstypen. Die Vereinbarung umfaßt den Bezeichner, gefolgt von einem Doppelpunkt. Der Anfangs- und der Endwert des festzulegenden Teilbereiches wird durch zwei Punkte getrennt. Der Teilbereich des Bezeichners JAHR beginnt mit dem Jahr 1583 und endet mit dem Wort MAXINT. Das Programm gilt also vom Jahre 1583 an, da Papst Gregor im Oktober des Jahres 1582 den Gregorianischen Kalender eingeführt hat, der bis zum heutigen Tage gilt. MAXINT ist ein in PASCAL vordefinierter Bezeichner, der die größte mögliche Integerzahl liefert. In einem Miniprogramm können Sie sich einmal über eine WRITE-Anweisung MAXINT ausgeben lassen. PASCAL-Bücher klären darüber auf, warum MAXINT den Wert 32767 hat.

Leider ist im KC-PASCAL die Festlegung der Teilbereiche ohne praktischen Wert, da der bescheidene Compiler keine Überpüfung der Einhaltung dieser Teilbereiche vornehmen kann. In Turbo-PASCAL ist dies mit einem entsprechenden Compilersteuerzeichen möglich. Die Editorzeilen 170 bis 370 ermitteln den Wochentag zu einem vorgegebenen Datum. Sie können also mit dem Programm überprüfen, ob Sie ein Sonntagskind sind. Allerdings ist dieses Programm noch unvollständig, da keine Sicherheit gegen Fehleingaben eingebaut ist. Darauf habe ich in den Editorzeilen 110 bis 130 hingewiesen. Es erfolgt also leider keine Fehlermeldung, wenn der 31.4. eingegeben wird, der ja gar nicht existiert. Auch die Schaltjahrregel mit dem 29.2. fehlt hier noch. Solche Kontrollen lassen das Programm natürlich umfangreicher werden. Für den Könner wäre dies aber eine lohnende Programmierübung.

Wiederholungen oder Schleifen

Zur Grundstruktur der Schleife werden wir jeweils ein Beispiel für die Zähl-, Solange- und Wiederholschleife betrachten. In BASIC haben Sie gewiß schon mit der Zählschleife gearbeitet. Die Einleitungszeile kann in BASIC beispielsweise FOR I=1 TO 10 STEP 0.5 lauten. Dann folgt der Schleifenkörper, also das, was entsprechend oft wiederholt werden soll. Abgeschlossen wird der Schleifenkörper in BASIC mit NEXT I.

In PASCAL läuft die Sache ein wenig anders. Hier gibt es die frei wählbare Schrittweite mit STEP leider nicht, sondern die Schrittweite ist stets Eins. Des weiteren darf in PASCAL die Laufvariable, z. B. I, nicht vom Typ REAL sein, denn für reelle Zahlen gibt es keinen eindeutigen Vorgänger und Nachfolger. Das schränkt die Arbeit mit der Zählschleife etwas ein. Braucht man als Schrittweite reelle Zahen, so greift man auf eine Solange- oder Wiederholschleife zurück. Auch das NEXT zum Abschluß des Schleifenkörpers kennt PASCAL nicht. Der Schleifenkörper wird hier ganz einfach mit BEGIN und END als Verbundanweisung dargestellt. In PASCAL würde man z. B. schreiben: FOR I:=1 TO 10 DO (do = tuen, ausführen). Dann folgen das Wort BEGIN, der Schleifenkörper und das Wort END. Die Laufvariable mit dem Bezeichner I muß im Vereinbarungsteil als INTEGER-Typ vereinbart werden. Ein Rückwärtszählen ist auch möglich. Dazu ist anstelle von TO das Wort DOWNTO zu setzen. Schauen wir uns das am besten im Beispielprogramm KALENDER an.

Listing KALENDER

```
 10 PROGRAM KALENDERFORMAT;
 20 VAR BEGINN,ANZAHL,I,J:INTEGER;
 30     TAG:ARRAY(/1..7,1..2/) OF CHAR;
 40 BEGIN
 50  TAG(/1/):='Mo';TAG(/2/):='Di';
 60  TAG(/3/):='Mi';TAG(/4/):='Do';
 70  TAG(/5/):='Fr';TAG(/6/):='Sa';
 80  TAG(/7/):='So';
 90  PAGE;
100  WRITELN('Zahl des Wochentages
     (MONTAG=1,');
110  WRITE('an dem der Monat beginnt: ');
120  READ(BEGINN);
130  WRITE('Anzahl der Monatstage: ');
140  READ(ANZAHL);
150  WRITELN;WRITELN;
160  FOR I=1 TO 7 DO
170   BEGIN
180    WRITE('   ',TAG(/I/));
190    J:=I-BEGINN+1;
200    IF J>0
210     THEN WRITE(J:4)
220     ELSE WRITE(' ':4);
230    J:=J+7;
240    WHILE J<= ANZAHL DO
250     BEGIN
260      WRITE(J:4);
270      J:=J+7;
280     END;
290    WRITELN;
300   END
310 END.
```

Das Programm soll uns ein ansprechendes Kalenderformat als Taschen- oder Wandkalender für jeweils einen Monat liefern. Dazu erscheinen in der ersten Spalte die Abkürzungen der Wochentage Montag bis Sonntag, in unserem Beispiel Mo, Di, Mi, usw. Daran werden die einzelnen Tageszahlen exakt angefügt. Zu diesem Zweck sind im Programm der Wochentag, mit dem der Monat beginnt und die Gesamtzahl der Tage des Monats einzugeben. In der Editorzeile 30 vereinbaren wir unter dem Bezeichner TAG erstmals ein zweidimensionales ARRAY, also ein zweidimensionales Feld. Dies haben Sie mit der Anweisung DIM in BASIC sicherlich auch schon getan. Das zweidimensionale Feld ist vom Typ CHAR, besteht also nur aus Zeichen. Die erste und die zweite Dimension werden durch ein Komma getrennt, und das Ganze steht in eckigen Klammern, die hier durch die Hilfskonstruktion „(/" ersetzt werden. Die erste Dimension geht von 1 bis 7. Das ist die Numerierung der Wochentage. Die zweite Dimension schafft für zwei Zeichen Platz, also Mo, Di, Mi, usw.

Der Anweisungsteil beginnt in der Editorzeile 40. In den Zeilen 50 bis 80 wird der Feldvariablen mit dem Bezeichner TAG der zum jeweiligen Index 1 bis 7 zugehörige Text zugewiesen. Die Zählung beginnt also mit dem Index 1 für Montag, abgekürzt mit Mo. Die Zählschleife beginnt in der Editorzeile 160. Der Schleifenkörper ist in ein BEGIN und ein END eingefaßt und umfaßt die Editorzeilen 170 bis 300. Leider geht es im Schleifenkörper etwas unübersichtlich zu. Die ordentliche Formatierung ist also gar nicht so leicht zu erreichen. Zu allem Überdruß steckt hier auch noch eine vorprüfende Schleife mit drin, eingeleitet durch das Wort WHILE. Ich bitte Sie, diesen Programmteil zunächst einmal so hinzunehmen. Die PASCAL-Freaks sollten versuchen, eine übersichtlichere Lösung zu finden.

Nun besteht natürlich der Wunsch, diesen Kalender auch auszudrucken. Leider ist unser KC-PASCALmit Druckerausgaben nicht so komfortabel wie z. B. TURBO-PASCAL. Der einfachste Weg ist hier die Benutzung des Protokolldrucks. Die entsprechende Vorgehensweise entnehmen Sie bitte dem Kasten.

Protokolldruck

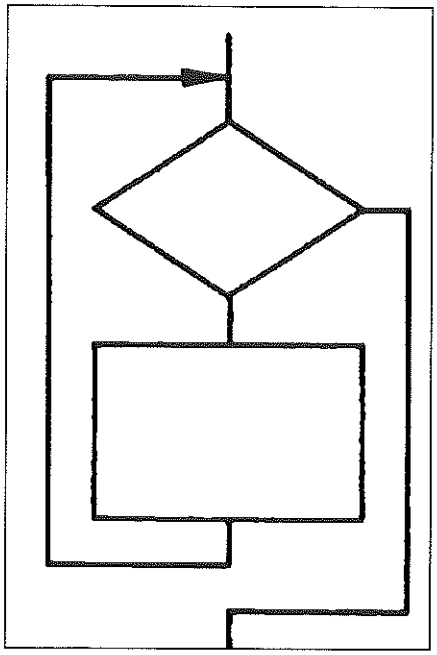
Zunächst müssen Sie Ihre Druckroutine auf Protokolldruck einstellen. Die Aktivierung kann über die Tastatur oder vom PASCAL-Programm aus erfolgen. Der Protokolldruck wird beim KC 85 mit dem ASCII-Steuerzeichen 15 ein- und auch wieder ausgeschaltet. Dazu müßte in unserem Quellprogramm zunächst die Editorzeile 150 erweitert werden. Dort steht zweimal die Anweisung WRITELN. An das zweite WRITELN fügen wir (CHR(15)) an und schalten damit den Protokolldruck ein. Zum Ausschalten des Protokolldrucks fügen wir eine Editorzeile 305 ein, die die Anweisung WRITE(CHR(15)) enthält.

Übrigens sind die hierbei genutzten WRITE, WRITELN und viele andere Anweisungen im exakten Sprachgebrauch Prozeduren. Beim Compilieren dieses erweiterten Programms wird Sie vielleicht die Fehlermeldung 2 überraschen, die auf ein fehlendes Semikolon aufmerksam macht. Dieses Semikolon ist nach dem END in der Editorzeile 300 zu setzen. Drücken Sie nach dem Erscheinen der Fehlermeldung 2 in Zeile 305 nicht die Taste E, sondern P. So gibt Ihnen der Editor gleich die vorhergehende, also die Editorzeile 300, zur Korrektur aus.

Wir wollen uns jetzt etwas genauer mit den beiden anderen Möglichkeiten von Schleifen beschäftigen. Die Solange- und die Wiederholschleife bietet unser einfacher BASIC-Interpreter nicht. Es gibt sie aber mittlerweile auch bei komfortablen BASIC-Interpretern und BASIC-Compilern, insbesondere für 16-Bit-Rechner. In unserem einfachen BASIC-Interpreter kann man diese Schleifenarten in Form von Ersatzkonstruktionen realisieren. Aber das wollen wir in unserem PASCAL-Kurs nicht tun. Wir werden im folgenden zwei kurze Programmbeispiele kennenlernen, die genau die gleiche Aufgabe, im ersten Beispiel mit einer Solangeschleife und im zweiten mit einer Wiederholschleife, lösen. Zunächst zum Programm ABWEIS.

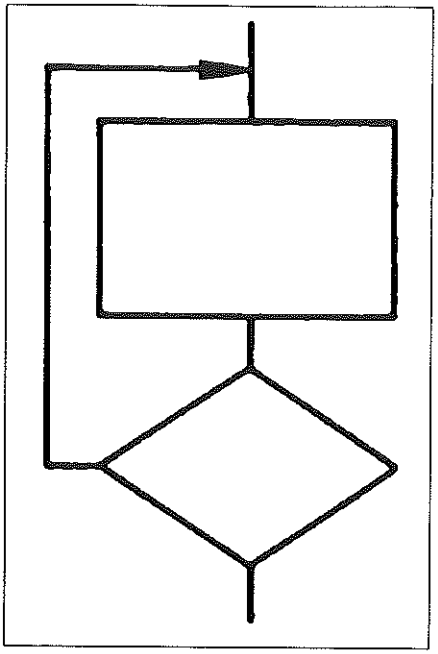
Listing ABWEIS

```
 10 PROGRAM SOLANGE;
 20 CONST EPSILON=1.0E-3;
 30 VAR SUM:REAL;
 40     N:INTEGER;
```



Vorprüfende Wiederholung (nach DIN 66 262).

Quelle: LOG IN Archiv



Nachprüfende Wiederholung (nach DIN 66 262).

Quelle: LOG IN Archiv

```
 50 BEGIN
 60  SUM:=0;N:=1;
 70  WHILE ABS(2-SUM)>=EPSILON DO
 80   BEGIN
 90    SUM:=SUM+1/N;
100    N:=N+N;
110   END;
120  WRITELN('N=',N,' SUMME=',SUM)
130 END.
```

Die Begriffe „Abweisschleife", „vorprüfende Schleife" und „Solangeschleife" werden gleichermaßen verwandt und deuten auf die Verwendung einer WHILE DO-Konstruktion in PASCAL hin. Das Listing des Quellprogramms ABWEIS zeigt in der Editorzeile 20 die Vereinbarung einer Konstanten mit dem Bezeichner EPSILON. Der Wert von EPSILON ist 1/1000. So etwas verwendet der Mathematiker als Abbruchbedingung bei der Anwendung eines Näherungsverfahrens. Wenn die Differenz zwischen zwei aufeinanderfolgenden Berechnungsdurchläufen kleiner als EPSILON, in unserem Beispiel also 1/1000, ist, dann gilt das Ergebnis als hinreichend genau und die Berechnung wird abgebrochen. Das Rechenergebnis soll in unserem Beispiel in der Variablen mit dem Bezeichner SUM abgespeichert werden. Der Bezeichner I fungiert als Zähler für die Anzahl der Durchläufe. Beide Bezeichner werden in der Editorzeile 60 auf die entsprechenden Anfangswerte gesetzt. Bis hierhin gleicht dieses Programm dem nachher folgenden.

Die Solangeschleife wird in der Editorzeile 70 eingeleitet. Das englische Wort WHILE bedeutet während oder solange. Damit können wir die Zeile 70 etwa so ins Deutsche übersetzen: Solange der Absolutwertwert von 2-SUM größer oder gleich EPSILON ist, tue das folgende. Dieses Folgende ist wieder mit BEGIN und END in eine Verbundanweisung eingeschlossen.

Ist diese Bedingung nicht mehr erfüllt, also die Differenz zwischen zwei aufeinanderfolgenden Berechnungen kleiner als EPSILON, dann wird die Berechnung abgebrochen und in der Editorzeile 120 das Ergebnis auf dem Bildschirm angezeigt.

Das Programmbeispiel zeigt, daß bei der Solangeschleife die Durchführbedingung vor dem Schleifenkörper steht. Wenn diese Durchführbedingung schon beim Programmstart nicht erfüllt ist, dann wird der Schleifenkörper nie durchlaufen, sondern die Berechnung sofort abgewiesen, daher der Name Abweisschleife. Bei der Nichtabweis- oder Wiederholschleife liegen die Dinge anders. Schauen wir uns dazu das Programm NICHTAB an.

Listing NICHTAB

```
 10 PROGRAM WIEDERHOLE;
 20 CONST EPSILON=1.0E-3;
 30 VAR SUM:REAL;
 40     N:INTEGER;
 50 BEGIN
 60  SUM:=0;N:=1;
 70  REPEAT
 80    SUM:=SUM+1/N;
 90    N:=N+N;
100  UNTIL ABS(2-SUM)<EPSILON
110  WRITELN('N=',N,' SUMME=',SUM)
120 END.
```

Bis zur Editorzeile 70 sind die Listings der Quellprogramme ABWEIS und NICHTAB identisch. Die nachprüfende Scleife oder Nichtabweisschleife wird mit dem Wort REPEAT, also „Wiederhole!", eingeleitet. Dann folgt der Schleifenkörper. Er endet bei dem Wort UNTIL, also „bis". Wir können auch das wieder in einem deutschen Satz formulieren: Wiederhole den Schleifenkörper bis der Absolutwert von 2-SUM kleiner als EPSILON ist.

Bei der Solangeschleife mit der WHILE-DO-Anweisung lautete die Bedingung größer gleich EPSILON. Bei der Wiederholschleife mit der REPEAT-UNTIL-Anweisung

lautet die Bedingung kleiner EPSILON. Damit liefern beide Programme genau das gleiche Ergebnis. Bei der Festlegung der Durchführungsbedingung mit WHILE DO und der Abbruchbedingung mit REPEAT UNTIL muß der Programmierer also etwas aufpassen. Die Wiederholschleife wird auch Nichtabweisschleife genannt, da der Schleifenkörper in jedem Fall mindestens einmal durchlaufen wird. Genau das ist der Unterschied zur Abweisschleife, und der Programmierer wird sich jeweils die günstigere Variante für seinen Anwendungsfall aussuchen.

Das folgende Beispiel arbeitet mit ineinander geschachtelten Wiederholschleifen.

Listing DEZBIN

```
 10 PROGRAM DEZBIN;
 20 CONST BASIS=2;
 30 VAR ZAHL,HILF,ABBRUCH:INTEGER;
 40 BEGIN
 50  PAGE;
 60  REPEAT
 70   WRITE('Dezimalzahl (Abbruch mit
      0)= ');READ(ZAHL);
 80   ABBRUCH:=ZAHL;
 90   HILF:=1;
100   REPEAT
110    HILF:=HILF*BASIS
120   UNTIL HILF>ZAHL;
130   REPEAT
140    HILF:=HILF DIV BASIS;
150    WRITE(ZAHL DIV HILF:1);
160    ZAHL:=ZAHL MOD HILF
170   UNTIL HILF=1;
180   WRITELN;WRITELN;
190  UNTIL ABBRUCH=0
200  WRITELN
210 END.
```

Das Programm rechnet solange Dezimalzahlen in Binärzahlen um, bis als Abbruchbedingung eine Null eingegeben wird. Die größte Dezimalzahl, die eingegeben werden kann, wird durch den Zahlentyp INTEGER begrenzt. Er ermöglicht auch die ganzzahlige Division mit DIV und die Modulorechnung mit MOD. Beides kam schon bei der Osterformel im Programm OSTERN zum Tragen. Vielleicht experimentieren Sie einmal mit diesem Programm, in dem Sie die Konstante mit dem Bezeichner BASIS ändern. Wählen Sie z. B. BASIS = 8, dann erhalten Sie Oktalzahlen. Hexadezimalzahlen können wir mit diesem Programm leider nicht berechnen. Vielleicht versuchen Sie einmal die Programmierung.

Der Record – ein starker Typ

Das folgende Programmbeispiel UHR soll mit einem leistungsfähigen Datentyp bekanntmachen.

Listing UHR

```
 10 PROGRAM UHR;
 20 VAR UHR:RECORD
 30          STD:0..23;
 40          MIN,SEK:0..59;
 50      END; (*ENDE VON RECORD*)
 60 PROCEDURE PAUSE(P:INTEGER);
 70 VAR VERZOEGERUNG:INTEGER;
 80 BEGIN
 90  FOR VERZOEGERUNG:=1 TO P DO
100 END
110 BEGIN (*HAUPTPROGRAMM*)
120  WRITELN('ZEITEINGABE');
130  WRITE('STUNDE=');READ(UHR.STD);
140  WRITE('MINUTE=');READ(UHR.MIN);
150  WRITE('SEKUNDE=');READ(UHR.SEK);
160  WITH UHR DO
170   REPEAT
180    PAUSE(1300);(*ZAHLENWERT FUER KC
       85/2/3*)
190    SEK:=SUCC(SEK) MOD 60;
200    IF SEK=0
210     THEN
220      BEGIN
230       MIN:=SUCC(MIN) MOD 60;
240       IF MIN=0
250        THEN STD:=SUCC(STD) MOD 24
260      END;
270    WRITELN(STD:2,':',MIN:2,':',SEK:2)
280   UNTIL INCH=CHR(32)
290 END.
```

Das Programm UHR gibt uns nach Eingabe der Startzeit im Sekundenrhythmus die Uhrzeit an. Durch die Wahl einer Verzögerungszeit wird die Uhr dem jeweiligen Rechnertyp angepaßt. Interessant ist die einfache Zeitrechnung zu 60 Sekunden und Minuten sowie zu 24 Stunden mit Hilfe der Modulorechnung. Diese drei Zeitangaben werden mit Hilfe eines besonderen Datentyps zusammengefaßt.

Schauen wir uns nun das Listing des Progamms UHR an. In der Editorzeile 20 wird eine Variable mit dem Bezeichner UHR vereinbart, die vom Datentyp RECORD ist. Die Übersetzung des englischen Wortes record lautet Aufzeichnung, daher auch der Name Kassettenrecorder. Aber diese Übersetzung hilft uns nicht weiter. Wir hatten die Datentypen ARRAY und RECORD schon zu Beginn unseres Kurses kurz erwähnt. Mit dem ARRAY als einem Datenfeld von gleichartigen Komponenten haben wir schon gearbeitet.

Ein RECORD hingegen kann verschiedenartige Komponenten zusammenfassen. Diese Komponenten sind die sogenannten Datenfelder, also z. B. Name, Anschrift, Steuernummer und Monatsverdienst.

Diese vier Datenfelder ergeben zusammen für eine Person einen Datensatz, und dieser Datensatz heißt RECORD. Viele solcher Datesätze, z. B. aller Beschäftigten eines Unternehmens, ergeben dann eine Datei, die mit Hil-

fe des RECORD-Typs in PASCAL sehr elegant bearbeitet werden kann. So kann z. B. auf einen Ritt ein ganzer Datensatz einer Prozedur, die irgendeine Verarbeitung realisieren soll, übergeben werden. In BASIC werden solche Datensätze mit der DIM-Anweisung aufgebaut, wobei dann die erforderlichen Operationen mühsam für jedes Datenfeld einzeln organisiert werden müssen. In unserem Beispiel besteht der RECORD mit dem Bezeichner UHR aus den Recordkomponenten STD, MIN und SEK. Der Typ der Komponente STD ist ein Teilbereichstyp, der von 0 bis 23 geht. Die Komponenten MIN und SEK sind auch vom Teilbereichstyp, allerdings für den Bereich von 0 bis 59. Mit diesen Zahlenangaben ist für den Compiler automatisch klar, daß es sich um INTEGER-Zahlen handelt. Damit hat jede Zahl, außer Anfangs- und Endwert automatisch einen definierten Vorgänger und Nachfolger und auch der Modulorechnung mit MOD steht nichts im Wege. Das END in der Editorzeile 50 markiert das Ende des Records. Ebenso wie bei der CASE-Anweisung existiert zu diesem END ausnahmsweise kein BEGIN.

Die Prozedur mit dem Namen PAUSE umfaßt die Editorzeilen 60 bis 100. In Turbo-PASCAL benötigen wir aber diese Prozedur nicht, denn hier gibt es die Anweisung „DELAY [Zeit in ms]" (delay = aufschieben, verzögern). In der Prozedur PAUSE existieren die beiden lokalen, also nur in dieser Prozedur gültigen Variablen, mit den Bezeichnern P und VERZOEGERUNG. Der Prozedur wird beim Aufruf vom Hauptprogramm aus ein Zahlenwert übergeben, den die lokale Variable mit dem Bezeichner P aufnimmt. Die Laufanweisung in der Editorzeile 90 fungiert dann als der gewünschte „Zeitfresser". Das ist übrigens nicht die feine englische Art einer Pause-Prozedur, da die Normung in Sekunden oder Millisekunden fehlt. Für uns erfüllt diese Form zunächst ihren Zweck.

Wenden wir uns nun dem Anweisungsteil des Hauptprogramms zu. In den Editorzeilen 120 bis 150 werden Stunde, Minute und Sekunde eingegeben, bei denen die Uhr gestartet werden soll. Die drei READ-Anweisungen zeigen, wie die einzelnen Recordkomponenten exakt zugewiesen werden. Dazu wird zunächst der Bezeichner des Records, in unserem Fall UHR, angegeben. Dann folgen ein Punkt und der Bezeichner der entsprechenden Recordkomponente. Hier sei nochmals erwähnt, daß unser KC-PASCAL leider keine Möglichkeit bietet, die Einhaltung der Grenzen der festgelegten Teilbereichstypen zu überwachen. In Turbo-PASCAL ist dies mit dem Compilersteuerzeichen {$R+} und dem Preis eines langsameren Programmablaufes möglich.

Wird nach Eingabe der Sekunden die ENTER-Taste betätigt, dann beginnt die Uhr zu laufen. Eingeleitet wird das Ganze mit der Wortfolge WITH UHR DO in der Editorzeile 160. Ins Deutsche übersetzt heißt dies: Mit dem Bezeichner UHR tue folgendes. Diese Anweisung erspart uns im folgenden Programmablauf das ständige Schreiben von UHR.STD, UHR.MIN und UHR.SEK. Der Bezeichner UHR mit Punkt kann weggelassen, denn mit der Angabe WITH UHR DO ist für den Compiler klar, daß es sich um die entsprechenden Komponenten des Records mit dem Bezeichner UHR handelt.

Unsere Uhr läuft in einer REPEAT UNTIL-Schleife, deren Abbruchbedingung in der Editorzeile 280 ausgewiesen ist. Die Anweisung INCH entspricht dem bekannten INKEY$ aus BASIC. Die Zahl 32 ist der ASCII-Code für die Leertaste. Der Programmablauf wird demnach durch Drücken der Leertaste abgebrochen. In TURBO-PASCAL hätten wir REPEAT UNTIL KEYPRESSED geschrieben. Die Funktion KEYPRESSED liefert nämlich dann einen wahren Wert, also TRUE, wenn eine beliebige Taste gedrückt wird.

Aber nun noch ein paar Bemerkungen zum Schleifenkörper, der die Editorzeilen 180 bis 270 umfaßt. Hier steckt einiges von dem drin, was wir schon in früheren Beispielprogrammen behandelt haben. In Zeile 180 wird die Prozedur PAUSE mit einem geeigneten Verzögerungszahlenwert aufgerufen. In Zeile 190 wird mit SUCC der Nachfolger des aktuellen Wertes der Recordkomponente SEK ermittelt und als aktueller Wert SEK zugewiesen. Wenn SEK=0 ist, dann erfolgt der Minutensprung. Das Gleiche gilt für den Stundensprung, wenn MIN=0 ist. Die Ausgabe erfolgt in der Editorzeile 270 mit 2 Stellen für Stunden, Minuten und Sekunden, jeweils durch einen Doppelpunkt getrennt.

Starten Sie Ihren Testlauf am besten mit 23 Uhr, 58 Minuten und 50 Sekunden. Sie können dann das Umschalten sehr schön beobachten. Das Bildschirmrollen wird Sie bei diesem Programm sicherlich stören. Leider fehlt in unserem KC-PASCAL eine Anweisung zur Kursorpositionierung auf dem Bildschirm. In Turbo-PASCAL gibt es dazu die Anweisung GOTOXY(Spalte,Zeile), wobei die Zählung bei 1 und nicht bei 0 beginnt. In BASIC würden wir PRINT AT oder LOCATE verwenden. Für den 16-Kbyte großen KC-PASCAL-Compiler bieten die Hersteller von der TU Dresden in ihrem Service-Prozedurpaket auch eine LOCATE-Prozedur an, die aber, da es sich um Maschinencodeprogramme handelt, nur auf dem KC 85/2/3 lauffähig ist. Die Arbeit mit Maschinencode können wir in unserem Anfängerkurs leider nicht behandeln. Wer sich das für seinen Computer zutraut, dem seien hier nur die Anweisungen INLINE und USER als Schlüssel zum Erfolg genannt. Er findet dazu Hinweise in einschlägiger PASCAL-Literatur.

Vom Quellcode zum Maschinencode

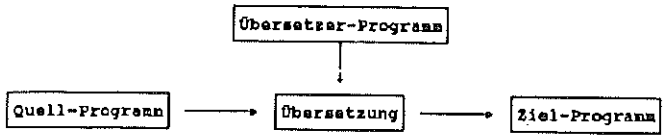
Wir wollen jetzt aus dem Quellprogramm UHR ein Maschinencodeprogramm herstellen, das direkt vom Betriebssystem des Kleincomputers, also ohne Anwesenheit unseres PASCAL-Compilers, eingelesen und abgearbeitet werden kann. Dazu gehen wir folgendermaßen vor:

◊ Zunächst wird das Programm UHR nicht mit dem Kommando C, sondern mit T, wie translate, also übertragen oder auch übersetzen, bearbeitet. Dazu wird das Kommando T 10,,UHR eingegeben. Es erfolgt jetzt ein Übersetzungsvorgang, der aber nicht mit der Frage RUN?, sondern mit der Ausschrift OK? abgeschlossen wird.

◊ Wir schalten jetzt den Kassettenrecorder auf Aufnahme und geben Y für yes über die Tastatur ein. Daraufhin wird das Maschinencodeprogramm zusammen mit dem rund 4 Kbyte großen Laufzeitsystem auf die Kassette ausgelagert. Es entsteht ein sogenanntes Commonfile.

Das englische Wort common heißt allgemein, gemeinsam oder öffentlich. Arbeitet man mit Turbo-PASCAL und Diskette, dann wird dieser Filetyp mit der Bezeichnung COM oder EXE kenntlich gemacht. Exe ist die Abkürzung für execute, was soviel wie ausführen oder durchführen bedeutet.

Ein Hinweis für die Nutzer des ZX Spectrum. Bei ihm wird bei Herstellung des Maschinencodeprogramms mit dem Kommando T der Compiler zerstört und muß bei Bedarf neu geladen werden. Bei unseren Kleincomputern KC 87 und KC 85 geschieht dies nicht.



Quelle: Koerber, 1990

Vom Text des PASCAL-Programms zum Maschinencode.

Probieren Sie nun die erfolgreiche Erzeugung des Maschinencodeprogramms aus. Schalten Sie dazu am besten den Computer aus und wieder ein. Laden Sie nun das Maschinencodeprogramm UHR genauso wie andere Maschinencodeprogramme in Ihren Computer. Nach Beenden des Einlesvorganges startet es automatisch. Ein Wiederstart nach Programmabbruch ist vom Betriebssystem aus mit dem Kommando G möglich.

Prozeduren und ihre Parameter

Im folgenden soll es um die Parameterübergabe bei Prozeduren gehen. Dazu betrachten wir das nächste Programm ROEMDEZ.

Listing ROEMDEZ

```
 10 PROGRAM ROEMxDEZ;
 20 VAR EIN,ZAHL:INTEGER;
 30 PROCEDURE AUS(ZWERT:INTEGER;BUCH:
    CHAR);
 40 BEGIN
 50  WHILE ZAHL>=ZWERT DO
 60    BEGIN
 70     WRITE(BUCH);ZAHL:=ZAHL-ZWERT
 80    END
 90 END;
100 BEGIN (*HAUPTPROGRAMM*)
110  PAGE;
120  REPEAT
130    WRITE('Zahl (Abbruch mit 0)=');
       READ(EIN);
140    ZAHL:=EIN;
150    AUS(1000,'M');
160    AUS(500,'D');
170    AUS(100,'C');
180    AUS(50,'L');
190    AUS(10,'X');
200    AUS(5,'V');
210    AUS(1,'I');
220    WRITELN;WRITELN;
230  UNTIL EIN=0
240 END.
```

Dieses Programm wandelt Dezimalzahlen bis zum maximalen Integerwert von 32767, hier also der Konstanten MAXINT, in römische Zahlen um. Der umgekehrte Fall, also die Umwandlung von römischen in Dezimalzahlen wäre für Touristen, die gern alte Schlösser und Kirchen besichtigen, zweifellos von größerem Wert. Er ist aber auch etwas schwieriger zu programmieren. Vielleicht versuchen Sie es einmal.

Unser Programm ROEMDEZ ist hingegen relativ einfach aufgebaut. Zunächst vereinbaren wir zwei Variablen vom Typ INTEGER mit den Bezeichnern EIN und ZAHL. Das sind globale Variable, die sowohl im Hauptprogramm, als auch in Prozeduren oder Funktionen Gültigkeit haben. Die Editorzeilen 30 bis 90 beinhalten die Prozedur mit dem Namen AUS. Schauen wir uns zunächst den Prozedurkopf in der Editorzeile 30 genauer an. Nach dem Wort PROCEDURE und dem Prozedurnamen AUS folgen in runden Klammern die Parameterübergaben. Es gibt auch Prozeduren, die keinerlei Parameter benötigen (z. B. feststehende Formatierungsangaben für Bildschirm oder Drucker). Hier wird der Klammerausdruck einfach weggelassen. Unserer Prozedur AUS werden zwei Parameter übergeben. Der erste ist vom Typ INTEGER und trägt die Bezeichnung ZWERT, der zweite ist vom Typ CHAR und hat die Bezeichnung BUCH, was an Buchstabe erinnern soll. Beide Parameter sind durch ein Semikolon voneinander getrennt. Die Bezeichner ZWERT und BUCH werden als formale Parameter bezeichnet.

Den formalen Parametern werden beim Prozeduraufruf die aktuellen Parameter von der Aufrufstelle im Programm übergeben, und sie verrichten ihre Tätigkeit ausschließlich in der Prozedur AUS.

Damit sind Kollisionen mit Bezeichnern außerhalb der Prozedur AUS grundsätzlich ausgeschlossen.

Die Prozedur AUS besteht aus einer WHILE-DO-Schleife, also einer vorprüfenden Schleife. Die Editorzeile 70 macht deutlich, daß in 1000 er, 500 er, 100 er usw. Schritten der eingegebene Zahlenwert vermindert wird. Bei jeder Verminderung wird der entsprechende Buchstabe, z. B. das M für 1000, auf den Bildschirm ausgegeben. Beachten Sie dabei bitte, daß in der Prozedur AUS die Bezeichner BUCH und ZWERT nur lokal in der Prozedur gelten, während der Bezeichner ZAHL im Gesamtprogramm, also global, wirkt.

Das Hauptprogramm bewegt sich in einer REPEAT-UNTIL-Schleife, weil damit auf einfache Weise der Programmabbruch realisiert werden kann. In der Editorzeile 130 wird der Dezimalwert der Variablen mit dem Bezeichner EIN zugewiesen. In Zeile 140 folgt eine Zuweisung von

EIN auf ZAHL, damit EIN als Abbruchbedingung in Zeile 230 erhalten bleibt. Der wesentliche Teil des Hauptprogramms sind die insgesamt sieben Aufrufe der Prozedur AUS. So bedeutet der Aufruf AUS(1000,'M'), daß dem formalen Parameter ZWERT in der Prozedur AUS der Zahlenwert 1000 und dem formalen Parameter BUCH der Buchstabe M übergeben wird. Falls hier die Typverträglichkeit nicht gesichert ist, kommt es zu einer Fehlermeldung. Die Aufrufe werden solange fortgesetzt, bis die Variable mit dem Bezeichner ZAHL den Wert Null erreicht hat. Auch mit diesem Programm sollten Sie ein wenig experimentieren. Geben Sie z. B. die Zahl 40 000 ein, dann ist eine Fehlermeldung aus dem Laufzeitsystem die Folge, da der zulässige Bereich für den Typ INTEGER überschritten wurde. Vielleicht versuchen Sie auch einmal, die WHILE-DO-Schleife in der Prozedur AUS durch eine REPEAT-UNTIL-Schleife zu ersetzen, natürlich so, daß die Ergebnisse noch korrekt sind.

Das folgende Programm SORT ist schon etwas umfangreicher. Es dient dem Sortieren von Wörtern, z. B. Namen für das Klassenbuch oder von Verfassern für ein Literaturverzeichnis.

Listing SORT

```
10 PROGRAM ALPHASORT;
 20 CONST UMFANG=51;
 30       LAENGE=15;
 40 TYPE WORT=ARRAY(/1..LAENGE/) OF CHAR;
 50      WOERTER=ARRAY(/1..UMFANG/) OF
         WORT;
 60 VAR EIN:WOERTER;
 70     ENDE:BOOLEAN;
 80     NUMMER,I:INTEGER;
 90 PROCEDURE SORTIEREN(VAR SORTWORT:
    WOERTER;ANZAHL:INTEGER);
100 VAR TAUSCH:BOOLEAN;
110     SICHERN:WORT;
120     I:INTEGER;
130 BEGIN
140  REPEAT
150   TAUSCH:=FALSE;
160   FOR I:=1 TO ANZAHL-1 DO
170    IF SORTWORT(/I/)>SORTWORT(/I+1/)
180     THEN
190      BEGIN
200       TAUSCH:=TRUE;
210       SICHERN:=SORTWORT(/I/);
220       SORTWORT(/I/):=SORTWORT(/I+1/);
230       SORTWORT(/I+1/):=SICHERN
240      END;
250  UNTIL NOT TAUSCH;
260 END;(*END VON PROZEDUR*)
270
280 BEGIN (*HAUPTPROGRAMM*)
290  PAGE;
300  WRITELN('Eingabe der Woerter
     (max.',UMFANG-1,')');
310  WRITELN('Abbruch mit #');
320  NUMMER:=1;
330  REPEAT
340   WRITE(NUMMER:3,'. Wort=');
350   READLN;READ(EIN(/NUMMER/));
360   ENDE:=EIN(/NUMMER,1/)='#';
370   IF NOT ENDE
380    THEN NUMMER:=SUCC(NUMMER);
390    ELSE NUMMER:=PRED(NUMMER);
400  UNTIL ENDE;
410 IF NUMMER >0
420  THEN
430   BEGIN
440    SORTIEREN(EIN,NUMMER);(*PROZEDUR
       AUFRUF*)
450    WRITELN;
460    WRITELN('Alphabetisches Verz.');
470    FOR I:=1 TO NUMMER DO
480     WRITELN(I:3,'. Wort=',EIN(/I/));
490   END;
500 WRITELN;
510 WRITELN('E N D E')
520 END.
```

Auf die vielfältigen, in der Literatur beschriebenen Sortieralgorithmen wollen wir hier nicht eingehen. Sie sollten das Programm gleich einmal ausprobieren. Compilieren Sie es also mit dem Editorkommando C und geben Sie dann etwa 10 oder 20 Namen ein. Sicherlich werden auch Sie als Kenner des BASIC-Interpreters über die Geschwindigkeit, mit der unser KC-PASCAL sortiert, erfreut sein.

Schauen wir uns nun das zunächst etwas verwirrend erscheinende Listing an. Die Konstante mit dem Bezeichner UMFANG legt die Maximalzahl der zu sortierenden Wörter fest. Mit dem Bezeichner LAENGE wird die Wortlänge auf 15 Buchstaben begrenzt. Was bei der Eingabe darüber hinaus geht, wird einfach abgeschnitten und nicht berücksichtigt. Größere Zahlenwerte für UMFANG und LAENGE erfordern natürlich mehr Speicherplatz. Sie können damit an Ihrem Rechner je nach RAM-Speicherkapaziät experimentieren.

Eingeleitet durch das Wort TYPE vereinbaren wir jetzt zwei Datentypen, die im gesamten Programm, also im Hauptprogramm und in Prozeduren und Funktionen, Gültigkeit haben. Der Typ mit dem Bezeichner WORT charakterisiert ein einzelnes Wort, dessen Buchstabenlänge von 1 bis LAENGE, in unserem Fall 15, gehen kann. In Turbo-PASCAL hätte man dafür WORT = STRING[15] geschrieben, aber den Typ STRING kennt unser KC-PASCAL leider nicht. Der Typ WOERTER kennzeichnet ein Feld, das maximal die Größe des Inhaltes der Konstanten UMFANG haben kann. Beachten Sie dabei, daß dieses Feld vom Typ WORT ist, also auf den vorher vereinbarten Typ zurückgreift. Beachten Sie bitte weiter, daß wir bis jetzt nur Typen und noch keine Variablen vereinbart haben. Dies erfolgt in den Editorzeilen 60 bis 80. Die Variable mit dem Bezeichner EIN ist vom Typ WOERTER. Das bedeutet, daß z. B. EIN mit dem Index 1, also „EIN(/1/)", ein Wort mit maximal 15 Buchstaben aufnehmen kann. Das bedeutet aber auch, daß z. B. „EIN(/10,3/)" den dritten Buchstaben des zehnten Wortes liefert.

Die Variable mit dem Bezeichner ENDE ist vom Typ BOOLEAN, kann also nur die logischen Zustände TRUE für wahr und FALSE für falsch annehmen. Der Prozedur mit dem Namen SORTIEREN wollen wir besondere Auf-

merksamkeit schenken. Sie umfaßt die Editorzeilen 90 bis 260. Die Experten unter Ihnen könnten diese Prozedur auch durch solche mit anderen und schnelleren Sortieralgorithmen ersetzen. Dazu muß allerdings eine korrekte Parameterübergabe gewährleistet sein. Unserer Prozedur SORTIEREN werden zwei Parameter übergeben. Den Prozeduraufruf finden Sie in der Editorzeile 440 im Hauptprogramm. Der Inhalt der globalen Variablen EIN wird dabei dem formalen Parameter SORTWORT übergeben, wobei natürlich beide vom Typ WOERTER sein müssen. Als zweiter Parameter wird der Inhalt der globalen Variablen NUMMER dem formalen Parameter ANZAHL übergeben.

Von besonderer Wichtigkeit ist noch das Wörtchen VAR im Prozedurkopf in der Editorzeile 90. Hier handelt es sich um einen sogenannten Variablenparameter oder Referenzparameter (Referenz bedeutet Beziehung).

Durch das Wörtchen VAR wird eine Beziehung zwischen der globalen Variablen EIN und dem formalen Parameter SORTWORT hergestellt. Diese Beziehung ist rückwirkender Natur. Das bedeutet, daß eine Veränderung des Inhaltes von SORTWORT auch in die Variable mit dem Bezeichner EIN eingeschrieben wird.

Wir benötigen dies hier, damit am Schluß auch in der Variablen mit dem Bezeichner EIN die alphabetisch geordnete Folge vorliegt, die dann in den Edtorzeilen 470 und 480 ausgegeben wird. Da das Wörtchen VAR zurück wirkt, also auch den Inhalt der globalen Variablen ändert, sind hier Vorsicht und ein kühler Kopf vonnöten.

Wer sich nach bisherigen Erklärungen überfordert fühlt, sollte nicht resignieren, sondern in einem PASCAL-Lehrbuch unter dem Stichwort Prozeduren noch einmal in Ruhe nachschlagen. In der Prozedur SORTIEREN werden in den Editorzeilen 100 bis 120 noch die nur lokal gültigen Variablen mit den Bezeichnern TAUSCH, SICHERN und I vereinbart. Sie sehen, daß dieses I nichts mit dem I, das in der Editorzeile 80 vereinbart wurde, zu tun hat. Die Sortierung selbst läuft in einer REPEAT-UNTIL-Schleife ab. In ihrem Inneren befindet sich eine Zählschleife, in der jedes Wort mit den nachfolgenden verglichen wird. Liegt schon eine alphabetische Ordnung vor, dann wird nichts verändert. Im anderen Fall werden die beiden Wörter unter Nutzung der Hilfvariablen SICHERN vertauscht. Das Ganze muß wiederholt werden, solange der Inhalt des Bezeichners TAUSCH falsch, also FALSE, bleibt. Der Fachmann erkennt, daß dies ein uneffektiver Sortieralgorithmus ist. Bei umfangreichen Sortierproblemen steigt die Rechenzeit sehr schnell an. Dafür ist dieser Algorithmus aber relativ übersichtlich.

Zum Hauptprogramm ist nur wenig zu sagen. Die Editorzeilen 300 bis 400 realisieren die Eingabe. Der Bezeichner NUMMER fungiert dabei als Index für die eingegebenen Wörter. Die Editorzeilen 410 bis 510 umfassen den Aufruf der Sortierprozedur und die Ausgabe der sortierten Folge, sofern überhaupt Wörter eingegeben worden sind.

Das Programm hat noch einen Schönheitsfehler. Wird bei der Aufforderung, das 51. Wort einzutasten, kein Doppelkreuz als Abbruchbedingung eingegeben, dann erscheint die Aufforderung zur Eingabe eines 52. Wortes. Die Folge ist eine Fehlermeldung des Laufzeitsystems. Vielleicht beseitigen Sie selbst diese kleine Ungereimtheit.

Rekursionen

Zum Abschluß unseres PASCAL-Kurses für Einsteiger soll es anhand von drei Programmbeispielen um rekursive Prozeduren gehen. Das Wort rekursiv bedeutet zurückverfolgen. Eine Rekursion darf nicht mit einer Iteration verwechselt werden. Iteration bedeutet Wiederholung. Man spricht z. B. dann von einer Iteration, wenn ein Rechenverfahren wiederholt angewendet wird, um zu einer Näherungslösung zu gelangen. Je mehr Wiederholungen man durchführt, desto genauer ist die Lösung. Eine solche Iteration habe ich z. B. zur Berechnung von Fraktalgrafiken in meinem Buch „Kreativ mit dem Computer" angewendet. Für den Grafikfreund, der mit Hilfe der Mathematik interessante Gebilde auf dem Bildschirm erzeugen will, sind Iterationen unerläßlich. Schließlich sind wir auch bei unseren PASCAL-Programmen ABWEIS und NICHTAB iterativ vorgegangen.

Die Rekursion ist leider nicht so einfach wie die Iteration zu begreifen. Das liegt unter anderem an der zu starken Ausprägung des linearen Schritt-für-Schritt-Denkens des naiven Programmierers. Leider gibt es dazu auch wenig Literatur. Ich habe aber zwei Tips für Sie. Das ist zum ersten der Beitrag „Rekursion – was, wie, wozu?" von Dr. Wagenknecht in den Kleinstrechner-Tips, Heft 7, das 1987 im Fachbuchverlag Leipzig erschienen ist. Im zweiten Fall handelt es sich um das Buch „In BASIC effektiv programmieren" von Prof. Völz. Es erschien 1989 im Verlag Die Wirtschaft Berlin.

Eine anschauliche Erklärung für eine rekursive Struktur liefert Dr. Wagenknecht mit dem Märchen von der Fee, die Ihnen drei Wünsche freistellt. Nachdem Sie den ersten und zweiten Wunsch geäußert haben, formulieren Sie den dritten Wunsch. Er lautet: Ich hätte gern drei Wünsche frei. Innerhalb der von der Fee angebotenen Prozedur mit dem Namen „Drei Wünsche frei" rufen Sie also wiederum die Prozedur „Drei Wünsche frei" auf. Wenn die Fee hier nicht für einen Abbruch dieser Prozedur sorgt, dann haben Sie letztlich unendlich viele Wünsche frei, sicherlich eine moralisch nicht ganz saubere Angelegenheit. Solche rekursiven Prozeduren ohne Abbruchbedingung nennt man regressiv. Sie sind für unseren Computer nicht geeignet, denn er würde sich gewissermaßen totlaufen.

Rekursive Prozeduren, die man mittels Computerprogramm abarbeiten will, haben folgende Merkmale:

◇ Die Prozedur muß sich selbst aufrufen. Professor Völz nennt das ein Selbstzitat.

◇ Für die Abarbeitung der Prozedur muß eine Abbruchbedingung existieren.

Bei solchen Selbstaufrufen werden die Zwischenergebnisse in einem Stapel- oder Kellerspeicher aufbewahrt und dann bei der Zurückverfolgung wieder bereitgestellt. Das erfolgt nach dem LIFO-Prinzip – last in, first out. Was zuletzt zur Aufbewahrung gegeben wurde, wird als erstes wieder ausgegeben. Das Rückwärtslesen eines eingegebenen Textes dürfte demnach mit einer rekursiven Prozedur kein Problem sein. Sehen wir uns dazu das Programm TEXTDREH an.

Listing TEXTDREH

```
 10 PROGRAM TEXTDREH;
 20 PROCEDURE UMDREH;
 30 VAR EIN:CHAR;
 40 BEGIN
 50  READ(EIN);
 60  IF EIN<>'.' THEN UMDREH;
     (*REKURSIVER PROZEDURAUFRUF*)
 70  WRITE(EIN);
 80 END;
 90 BEGIN(*HAUPTPROGRAMM*)
 95  PAGE;
100  WRITELN('Texteingabe (Abschluss
     mit .)=');
110  UMDREH;(*PROZEDURAUFRUF*)
120  WRITELN
130 END.
```

In Turbo-PASCAL läuft dieses Programm nicht in dieser Form, da sich der Eingabepuffer bei READ etwas anders als in unserem KC-PASCAL verhält. Außerdem muß in TURBO-PASCAL mit dem Compilersteuerzeichen {$A–} erst die rekursive Arbeit des Compilers zugelassen werden.

In unserem Programmbeispiel hat die rekursive Prozedur den Namen UMDREH. Sie arbeitet übrigens parameterlos. In dieser Prozedur wird die lokale Variable mit dem Bezeichner EIN vom Typ CHAR vereinbart. Der Anweisungsteil beginnt in der Editorzeile 50 mit der Texteingabe. Sie kennen sicherlich den Text „Ein Neger mit Gazelle zagt im Regen nie". Aber vielleicht reicht Ihnen auch das Wort „Lagerregal". Solche Wörter oder Wortreihen, die von vorn und von hinten gleichermaßen lesbar sind, nennt man Palindrome.

Aber zurück zu unserem Programm und der ganz wichtigen Editorzeile 60. Sie enthält zunächst die Abbruchbedingung. Bei allen eingegebenen Zeichen außer dem Punkt ruft sich die Prozedur UMDREH selbst auf. Wird hingegen ein Punkt eingegeben, dann erfolgt der Abbruch und in Zeile 70 wird der Text nach dem LIFO-Prinzip, also mit dem letzten eingegebenen Zeichen zuerst, auf dem Bildschirm ausgegeben. Das Hauptprogramm liefert in diesem Beispiel wenig. Es realisiert nur eine Textausgabe und ruft dann die Prozedur UMDREH auf.

Unser zweites Programmbeispiel heißt GAUSSUM. Es greift die Anekdote um den Schüler Gauß auf, der zusammen mit seinen Mitschülern die Zahlen von 1 bis 100 zusammenrechnen sollte. Der kleine Gauß ging anders als seine Mitschüler vor und fand durch Überlegung eine allgemeingültige Gleichung für dieses Problem. In dieser Hinsicht ist unser Programm GAUSSUM also überflüssig Es zeigt aber die Berechnungsmöglichkeit mit Hilfe einer rekursiven Funktion.

Listing GAUSSUM

```
 10 PROGRAM GAUSSUMME;
 20 VAR EIN:INTEGER;
 30 FUNCTION SUM(N:INTEGER):INTEGER;
 40 BEGIN
 50  IF N=1
 60   THEN SUM:=1;
 70   ELSE SUM:=SUM(N-1)+N (*REK. FUNKT.
      AUFRUF*)
 80 END;
 90 BEGIN(*HAUPTPROGRAMM*)
100  WRITE('Die Summe von 1 bis ');
110  READ(EIN);
120  WRITE('betraegt ',SUM(EIN));
130  WRITELN
140 END.
```

Die Funktion mit dem Namen SUM umfaßt die Editorzeilen 30 bis 80. Dieser Funktion wird vom Hauptprogramm in Zeile 120 der Integerwert der globalen Variablen EIN übergeben. Diesen Wert erhält die lokale Variable N in der Funktion SUM. Die Abbruchbedingung wird bei N=1 erreicht. Falls im Programm eine 1 eingegeben wird, dann kommt gemäß Zeile 60 keine Rekursion zustande, denn der rekursive Funktionsaufruf erfolgt erst in der Editorzeile 70. Hier wird N solange vermindert, bis N=1 erreicht ist. Und dann wird zurückverfolgend die Summation ausgeführt und das Ergebnis als Funktionswert in Zeile 120 ausgegeben.

Zum „rekursiven" Abschluß nun noch das Programm HANOI mit dem Problem der Türme von Hanoi, das Dr. Wagenknecht in dem oben erwähnten Artikel auch in der Programmiersprache LOGO und mit Hilfe von Ersatzkonstruktionen in BASIC behandelt. In BASIC ist der rekursive Aufruf von Prozeduren und Funktionen leider nicht direkt möglich. In dem erwähnten Buch von Professor Völz sind dazu interessante Beispiele mit BASIC-Ersatzkonstruktionen enthalten.

Listing HANOI

```
 10 PROGRAM TURMVONHANOI;
 20 TYPE SCHEIBE=INTEGER;
 30      STAB=CHAR;
 40 VAR ANZAHL,I:SCHEIBE;
 50 PROCEDURE ZUG(ANZ:SCHEIBE;
    A,B,C:STAB);
 60 BEGIN
 70  IF ANZ>0
 80   THEN
 90    BEGIN
100     ZUG(ANZ-1,A,C,B);
110     I:=SUCC(I);WRITE(I:3,'. Zug:');
120     WRITELN('  VON  ',A,'  NACH   ',
        C);
130     ZUG(ANZ-1,B,A,C)
140    END;
150 END;
160 BEGIN(*HAUPTPROGRAMM*)
170  PAGE;
180  WRITE('Anzahl der Scheiben=');
     READ(ANZAHL);
190  WRITELN;
```

```
200   WRITELN('Staebe: S=Start, H=Hilf,
      Z=Ziel');
210   WRITELN;
220   I:=0;
230   ZUG(ANZAHL,'S','H','Z')
240 END.
```

Vielleicht versuchen Sie selbst einmal, das Listing des Programms HANOI zu interpretieren. Hier ruft sich die Prozedur ZUG gleich zweimal, nämlich in den Editorzeilen 100 und 130, selbst auf. Wer Verständnisschwierigkeiten hat, sollte für ein einfaches Beispiel mit drei Scheiben den Stapelspeicher und den gesamten Rechengang mit Bleistift und Papier nachvollziehen. Um das Programm komfortabler zu machen, kann der Experte versuchen, eine Bildschirmrollsperre in das Programm einzubauen.

Das letzte Beispiel zeigt, daß man nicht um jeden Preis rekursive Strukturen anwenden soll, ganz abgesehen davon, daß die Rechenzeiten höher als bei iterativen Lösungen sind. Dafür steckt in wenigen Programmierzeilen eine gaze Menge drin, was für den Einsteiger meist schwer durchschaubar ist. Rekursive Strukturen sind auch im Zusammenhang mit der Verarbeitung von Listen interessant. Favorisiert hierzu ist die Programmiersprache LISP unter dem Stichwort „Künstliche Intelligenz", was nicht unbedingt als Gegenteil für natürliche Dummheit aufgefaßt werden sollte. Die dynamische Verwaltung solcher Listen ist auch in PASCAL mit Hilfe sogenannter Zeiger oder Pointer möglich. Aber das ist schon etwas für Fortgeschrittene. Wir sind deshalb in unserem Anfängerkurs nicht darauf eingegangen.

END.

Zum Schluß möchte ich weitere Dinge nennen, die ich Ihnen leider vorenthalten mußte. Dazu gehört die in PASCAL so wichtige Arbeit mit Diskettenfiles. Aber auch mit Labels, das englische Wort für Aufkleber oder Etikett, wir sagen auch Marken dazu, haben wir nicht gearbeitet. Sie sollten ohnehin nur sparsam und nur in größeren Programmen verwendet werden, um eine saubere Programmstruktur nicht wieder zu verschmutzen. Ich habe auch nichts über direkte Speicherzugriffe mit den Anweisungen SIZE, ADDR, POKE und PEEK gesagt, desgleichen nichts über die Einbindung von Maschinencode mit den Anweisungen INLINE und USER. Außerdem steckt noch mehr in unserem KC-PASCAL-Editor drin. Dazu gehört z. B. das Neunumerieren von Quelltexten mit dem Kommando N Anfangszeile, Schrittweite. Auch die zusätzlichen Compilersteuerzeichen sind bei uns zu kurz gekommen, ebenso die Betrachtung der Fehlermeldungen des Compilers und des Laufzeitsystems. Auch für unsere Einsteigerkurs gilt das chinesische Sprichwort:

Was wir hören, vergessen wir, was wir sehen, behalten wir, was wir tun, verstehen wir.

Wir alle würden uns sehr freuen, wenn Sie durch diesen Kurs zu mehr „Tun" angeregt worden sind.

Literaturhinweise

Für unser KC-PASCAL gibt es leider nur wenig Literatur. Wir möchten hier auf den Beitrag von Lennartz mit dem Titel Pascal für Kleincomputer verweisen, der in der Mikroprozessortechnik Heft 4/1990 erschienen ist. Da das KC-PASCAL aus dem PASCAL für den ZX Spectrum abgeleitet wurde, wollen wir hier noch das folgende Buch nennen:

Dupont, R.; u. a.: Pascal auf dem ZX Spectrum. Düsseldorf: Sybex-Verlag 1985.

Es wird aber schwer sein, dieses Buch auszuleihen, denn die Bibliotheken der ostdeutschen Länder haben es gewiß nicht. Wer nun vom KC-PASCAL auf Turbo-PASCAL für 8- oder 16-Bit-Rechner umsteigen möchte, sieht sich einer Fülle guter, aber auch teurer Literatur gegenüber. Wir empfehlen deshalb zunächst Literatur, die auch in den Bibliotheken der ostdeutschen Länder vorhanden ist:

PASCAL-Serie von C. Kofer in den Heften 9/1987 bis 11/1988 der Zeitschrift Mikroprozessortechnik.
Serie Turbo-PASCAL-PRAXIS von M. Zander in den Heften 6,8,10,12/1989 und 3,5/1990 der Zeitschrift Mikroprozessortechnik
Fahr, K.: Die Leistungsfähigkeit von TPASCAL. In: EDV-Aspekte, Heft 4/1988.
Goldammer, G.: Pascal für die Anwendung in der Wirtschaft. Berlin: Verlag Die Wirtschaft 1987.
Paulin, G.: TURBO-PASCAL. Berlin: Verlag Technik 1988.
Paulin, G.; Schiemangk, H.: Programmieren mit PASCAL. Berlin: Akademie-Verlag 1989.

Wer sich ein neues Buch über Turbo-PASCAL Version 5.0 oder 5.5 für 16-Bit-Rechner zulegen möchte, wird natürlich nach etwas Preiswertem fragen. Hier unsere Empfehlungen:

Kaier, E.: TURBO PASCAL griffbereit. Vieweg-Verlag 1989 (Preis: 16,80 DM).
Beisecker, M.-A.; Brickwede, P.: TURBO PASCAL 5.5. QuickStart-Reihe des SYBEX-Verlages 1989 (Preis: 9,80 DM).
Füssinger, M.: Programmierkurs TURBO PASCAL 5.0. Sybex-Verlag 1989 (Preis: 39,00 DM).
Rollke, K.-H.: Grundkurs in TURBO PASCAL – Band 1. Sybex-Verlag 1987 (Preis: 29,80 DM).
Geise, T.: Programmieren mit TURBO PASCAL. Reihe Chip-Wissen. Vogel-Verlag 1989 (Preis: 35,00 DM).
Steiner, J.: Schnellübersicht TURBO PASCAL 5.5. Verlag Markt&Technik 1990 (Preis: 39,00 DM).

Das ist natürlich nur eine kleine Auswahl preisgünstiger Bücher. Sollten Sie zu dieser Literatur Fragen oder Bestellwünsche haben, so schreiben Sie an:

bookware individual
Dr. Hannes Gutzer
Telemannstraße 581/6
O-4090 Halle (Saale)

KC-Bibliographie

Diese Bibliographie beinhaltet eine Auswahl von Veröffentlichungen zum Einsatz von Kleincomputern in der ehemaligen DDR. Schwerpunkte sind die Computer- und Elektronik-Zeitschriften Radio Fernsehen Elektronik (RFE), Mikroprozessortechnik (MP), Funkamateur (FA), Rechentechnik/Datenverabeitung (R/D) und die einst so erfolgreiche Reihe Kleinstrechner-TIPS des Fachbuchverlages Leipzig.

Für die teilweise unvollständigen bibliographischen Angaben (z. B. fehlende Seitenzahlen) bitten wir um Nachsicht.

Hardware

Bedrich, M.: Kleinplotter XY 4131. In: RFE, Berlin 38 (1989) H. 10.

Domschke, W.: Kleinplotter XY 4131. In: RFE, Berlin 38 (1989) H.4.

Gutzer, H.: Kleincomputer in der DDR - Eine Übersicht. In: FA, Berlin (1987) Hefte 5, 6, 7.

Jagdmann, E.; Domschke, W.: 64-KByte-RAM-Module als Datenspeicher für BASIC-Programme im KC 85/3. In: MP, Berlin 3 (1989) H. 4.

Keller, G.: A4-Plotter für Robotron-KC. In: MP, Berlin 1 (1987) H. 12.

Kirves, K.-D.; Schiwon, K.: Computerkopplung KC 85/3- PC 1715 über V.24-Interface. In: MP, Berlin 1 (1987) H. 5.

Kirves, K.-D. u.a.: Digital-Ein-/Ausgabemodul für KC 85/2 und /3. In:MP, Berlin 1 (1987) H.10.

Kirves, K.-D.: V.24-Modul M003. In: MP, Berlin 1 (1987) H. 4.

Poppe, D.: Bustreiberaufsatz D002. In: MP, Berlin 2 (1988) H. 5.

Schiwon, K.; Krives, K.-D.: Floppydisk-Erweiterung für KC 85. In: R/D, Berlin 26 (1989) H. 7.

Sieder, R. u.a.: Analogeingabemodul M010 ADU 1 für KC 85/ und KC 85/3. In: RFE, Berlin 36 (1987) H. 4.

Völz, H.: Kleincomputer KC Compact. In: RFE, Berlin 39 (1990) H. 7, S. 457-459.

Grafe, H.: Kleincomputer als Speicheroszilloskop. In: FA, Berlin 39 (1990) H. 3, S. 134.

Hähle, B.: Anschluß einer Hexadezimaltastatur an den KC 85/3. In: RFE, Berlin 37 (1988) H. 1.

Knur, R.: Midi-Module für Kleincomputer. In: RFE, Berlin 37 (1988) H. 11.

Krüger, K.: Analogwerteingabe in Kleincomputer. In: RFE, Berlin 35 (1986) H. 9.

van der Meer, M.: Parallelschnittstelle für KC 85/2. In: RFE, Berlin 35 (1986) H. 10.

Möckel, F.: Eprom-Programmierer für KC 85/2 und KC 85/3. In: RFE, Berlin 36 (1987) H. 7.

Seveke, L.: ROM-Disk für KC 85/1. In: RFE, Berlin 37 (1988) H. 1.

Süss, F.: Sprachausgabe für KC 85/2 und KC 85/3. Jugend und Technik, Berlin (1987) H. 8.

Völz, H.: Serielle Schnittstelle für KC 85-2. In: RFE, Berlin 35 (1986) H.2.

Werner, H.-G.: Joystickmodul für KC 85/2(/3). In: MP, Berlin 2 (1988) H. 4.

Software

Domschke, W.: Das Softwarekonzept des KC85/3. In: MP, Berlin 1 (1987) H. 3.

Domschke, W.; Oppitz, V.: Malen mit Leonardo. In: R/D, Berlin 24 (1987) H. 8.

Girlich, U.: Bewegte Bilder aus dem KC. In: Kleinstrechner-Tips, Leipzig 1989 H. 11.

Gutzer, H.; Wendt, S.: Zeichen beliebiger Größe auf dem Bildschirm des KC 85/3. In: FA, Berlin 37 (1988) H. 2.

Gutzer, H.; Wendt, S.: Grafikdruck mit Grauwerten vom KC 85/3. In: FA, Berlin 37 (1988) H. 7.

Gutzer, H.; Wendt, S.: Sprite-Grafik mit dem Kleincomputern KC 85/3 UND 85/4. In: FA, Berlin 39 (1990) H. 3, S. 117-118.

Junek, H.: Magnetbandkatalog für KC 85/2,3 mit Zählwerk. In: MP, Berlin 2 (1988) H. 6.

Kirves, K.-D.: Arbeit mit BASIC-Datenfeldern beim KC 85/3. In: MP, Berlin 1 (1987) H. 3.

Kirves, K.-D.: KC 85/3-BASIC-Tip Maschinenprogramme (3). In: MP, Berlin 2 (1988) H. 4.

Kühnel, C.; Schekat, D.: Ein- und Ausgabe über die Nutzerschnittstelle des KC 85/1. In: RFE, Berlin 36 (1987) H. 4.

Lippold, R.: REDABAS für KC 87 (Kurzmitteilung mit Adresse). In: MP, Berlin 4 (1990) H. 6, S. 189.

Lorenz, P. u.a.: Computeranimation. In: R/D, Berlin 25 (1988) H. 1.

Oppitz, I.; Oppitz, V.: Kleincomputergrafik Teil I: Designansätze. In: R/D, Berlin 24 (1987) H. 8.

Oppitz, I.: Anlegen und Führen von Dateien mit dem KC 85/3. In; R/D, Berlin 25 (1988) H. 6.

Schiewe, C.: KC 85/1 generiert 24x80 Zeichen. In: MP, Berlin 4 (1990) H. 6, S. 186-187.

Schiwon, K.: Ausgabe von Pseudografikzeichen auf Matrixdrucker. In: MP, Berlin 1 (1987) H. 6.

Schlenzig, S.: Bildschirm-Fensterkopien mit dem KC85/2(/3). In: MP, Berlin 1 (1987) H. 1, Korrektur in MP 1 (1987) H. 3, S. 66.

Stammler, L.: Geometrie mit dem KC 85/1 „getupft". In: Kleinstrechner-Tips, Leipzig 1988, H. 9.

Völz, H.: Grafik auf dem KC 85-2. In: RFE, Berlin 35 (1986) 1, Korrektur in RFE 35 (1986) H.7, S.414.

Völz, H.: Fabas lange frei definierbare Festkomma-Arithmetik in BASIC. In: MP, Berlin 1 (1987) H. 12.

Völz, H.: Effektives Programmieren in BASIC. In: MP, Berlin 1 (1987) H. 6.

Weller, T.; Kotkowskij, S.: Simulation eines PASCAL-Compilers auf KC 85/2. In: RFE, Berlin 36 (1987) H. 3.

Wendt, S.: Zeichnen mit schnellen Grafikbefehlen. In: Kleinstrechner-Tips, Leipzig 1988 H. 8.

Werner J.: Interne Uhr für den KC 85/3. In: FA, Berlin 37 (1988) H. 1.