Teil 2

PASCAL leistungsfähiger und eleganter als Basic

Funktionen, Prozeduren, dynamische Variablen sowie Mengenoperationen sind wichtige Elemente der Programmiersprache UCSD Pascal. Sie werden in diesem Bericht beschrieben und entsprechenden Basic-Lösungen gegenübergestellt. Abschließend werden sechs Pascal-Versionen verglichen.

Listing 10 zeigt die Verwendung von Funktionen »Forward«-Deklaration, neue Statements und den Funktionsaufruf. Zuerst nun die Statements:

»DIV« und »MOD« dürften zum Teil sicher schon bekannt sein, dennoch zur Erinnerung: »MOD x« ergibt den Rest der Division einer Zahl durch »x«, »DIV x« hingegen den ganzzahligen Anteil des Quotienten.

Kurz: 345 MOD 7 = 2 und 345 DIV 7 = 49;

Mit der »Forward-Deklaration« existiert nun die notwendige Möglichkeit, eine Prozedur oder eine Funktion zu deklarieren ohne sie zu definieren. Stellen Sie sich vor, Sie hätten zwei Prozeduren P1 und P2. Sie beginnen mit der Deklaration der Prozedur P1 und bemerken, daß P1 intern P2 aufruft. Nun darf man bekanntlich nichts in einer Deklaration verwenden, was nicht schon vorher deklariert worden ist. Nun, werden Sie denken, das ist kein Problem, ich deklariere eben zuerst P2. Jetzt stellen Sie aber fest, daß P2 ihrerseits P1 aufruft. Was machen Sie nun? Für diesen Fall gibt es die »Forward«-Deklaration. Sie erlaubt es, wenigstens einmal den Namen der Prozedur oder Funktion sowie deren formale Parameter zu deklarieren, natürlich muß dann später die Definition nachgeholt werden, aber dann ohne die formalen Parameter. Wie das aussehen kann, steht im obigen Beispielprogramm.

Zur Funktion: Da die Funktion einen Wert ausgibt, kann man ihr auch einen Wert zuweisen. Das sieht man am Beginn des Hauptprogrammes. In der Funktion selbst muß man dem »Namen« der Funktion den Wert zuweisen, der in ihr bereitgestellt wurde. Damit hat nun die Funktion einen berechneten Übergabewert, mit welchem sie zurückkommt. Im Beispiel sind das die Zuweisungen: »QUERSUMME := x« und »QUERSUMME := temp«.

```
PROGRAM QUERSUMMEN (INPUT,OUTPUT);
VAR       check,resultat : INTEGER;

 FUNCTION QUERSUMME (x :INTEGER) :INTEGER; FORWARD;

 PROCEDURE EINGABE;
 BEGIN
   PAGE (OUTPUT);
   WRITELN; WRITELN; WRITELN;
   WRITELN ('BITTE GEBEN SIE EINE GANZE ZAHL EIN,');
   WRITELN ('VON DER SIE DIE QUERSUMME WISSEN');
   WRITELN ('WOLLEN. MIT '0' KOENNEN SIE AUFHOEREN');
   WRITELN;
   READLN (check);
   resultat := QUERSUMME (check);
 END;        (* VON EINGABE *)

 FUNCTION QUERSUMME;
 VAR          temp : INTEGER;
 BEGIN
   x := ABS (x);  temp := 0;
   IF x < 10
      THEN QUERSUMME := x;
      ELSE REPEAT
             temp := x MOD 10 + temp;
             x := x DIV 10;
           UNTIL x <= 0;
   QUERSUMME := temp;
 END;        (* DER QUERSUMME *)

BEGIN        (* DES HAUPTPROGRAMMES *)
  QUERSUMME := 1;
  REPEAT
    EINGABE;
    WRITELN ('QUERSUMME: ',resultat);
  UNTIL QUERSUMME (check) = 0
END.         (* DES HAUPTPROGRAMMES *)
```

Listing 10. Dank der Forward-Deklaration kann die Funktion Quersumme aufgerufen werden, ohne bereits deklariert zu sein.

Mit der Funktion kann man sehr einfach rekursive Strukturen erschaffen. Eine rekursive Funktion ist einfach gesagt eine Funktion, die sich selbst aufruft, die sich selbst enthält. Man kann durch derartige Programmierung sehr viel an Speicherplatz sparen oder auch unvernünftig verschleudern, sie kann sehr schnell und effizient sein oder auch langsam und ineffizient. Rekursionen sind aber im allgemeinen »stackbelastend«. Wenn Sie sich das nächste Beispiel genau anschauen, werden Sie herausfinden weshalb. Die Funktion soll die sogenannten Fibboconacci-Zahlen ausgeben. Sie werden nach folgender Rekursionsformel berechnet:

$x(n+1) := x(n) + x(n-1)$ wobei $x(0) = x(1) = 1$ und $n => 1$

Im voraus sei gesagt, daß diese Funktion sehr ineffizient ist. Vielleicht merken Sie schon beim ersten Durchlesen (Listing 11) weshalb.

```
FUNCTION FIBBOCO (n :INTEGER) :INTEGER;
BEGIN
  IF n <= 1 THEN FIBBOCO := 1
            ELSE FIBBOCO := FIBBOCO (n-1)  +  FIBBOCO (n-2)
END;
```

Listing 11. Rekursive Funktion

Diese kleine rekursive Funktion berechnet also nach der Vorschrift ein beliebiges »x (n (>1))« bei Übergabe von »n«. Versuchen Sie doch einmal, den Verlauf für die Zahl »x (5)« nachzuvollziehen. Sie merken, daß sehr viele Aufrufe nötig sind, bis sich die Funktion nicht mehr selbst aufruft. Für den Fall, wo »n = 5« ist, sind 15 Aufrufe nötig. Das braucht Zeit und belastet eben den Stapel. Hier beträgt die maximale Rekursionstiefe 4. Ein weiteres schönes Beispiel für die Rekursion wäre die Lösung des Problems von Hanoi. Ich möchte sie hier nicht zeigen, der interessierte Leser findet sie in fast allen Büchern über strukturiertes Programmieren. Sie ist zwar sehr elegant, verbraucht aber sehr viel Speicherplatz und belastet den Stapel recht stark.

Basic ohne Mengen

Eine weitere sehr mächtige Gruppe von Statements sind diejenigen, welche sich mit Mengen beschäftigen. Solche Befehle sind dem Basic gänzlich unbekannt.

Angenommen, Sie wollen in einem Programm nur eine bestimmte Eingabe erlauben, sagen wir alle Zahlen sowie die vier Operatoren »+,-,/,*«, so müssen Sie in Basic etwa das Programm in Listing 12 programmieren.

In der Schleife das Pascal-Programm wird getestet, ob die Eingabe der Bedingung genügt. Ist doch viel eleganter, nicht wahr? Nun bietet Pascal nicht nur solche einfachen Zugehörigkeitstests, sondern umfassende Mengenoperationen. Einige Beispiele dazu:

```
TYPE        farbe      = (ROT,BLAU,
                         GRÜN,GELB,
                         BRAUN,WEIß,
                         SCHWARZ);
            tag        = (MON,DIE,
                         MIT,DON,FRE-
                         ,SAM,SON);
            farbton    = SET OF
                         farbe;
            besetzt    = SET OF tag;
            zahlen     = SET OF
                         1..500;
            zeichen    = SET OF
                         char;
            buchstaben = SET OF
                         »A«..»Z«;
```

Ein solches Set besteht jeweils aus der Potenzmenge aller zugehörigen Objekte. Sein Speicherbedarf kann unter Umständen immens sein. Man kann nun auf diesen Mengen alle Mengenoperationen durchführen: Vereinigungs-(+), Schnitt-(*) und Differenzmenge (-), Mengengleichheit (=) und -ungleichheit (< >, Teil-(<=) und Obermengen (=>) und Zugehörigkeiten (IN).

Sie sehen, man kann sehr universell mit Mengen hantieren (Listing 13). Es gibt dafür unzählige Anwendungsgebiete. Ihnen fallen sicher mehr ein. Ein kleiner Seitenblick auf Basic zeigt, daß solche Mengenoperationen dort überhaupt nicht unterstützt werden.

Vielseitige Records

Ich komme nun zu einem Merkmal, das Wirth von Cobol übernommen hat, welcher aber typisch für Pascal ist. Es sind dies die Records. Schön, aber was ist denn ein Record?

Am besten ist es, wenn man einen Record mit einem Array vergleicht. Ein Array ist ja bekanntlich eine Anzahl Variablen, die denselben Namen (identifier) besitzen. Man unterscheidet die einzelnen Elemente dadurch, daß man sie indiziert. Aber einen großen Nachteil haben Arrays: Alle Elemente müssen von demselben Typ sein, das heißt man kann nicht in ein und demselben Array Strings, Reals und Booleans zugleich haben. Nun, das ist ja nicht weiter schlimm, man nimmt eben mehrere Arrays. Stellen Sie sich aber vor, Sie möchten eine einfache Adreßverwaltung herstellen und die einzelnen Elemente so einfach ansprechen wie die eines Arrays. Für jede Adresse brauchen Sie aber mehrere Felder, zum Beispiel für Namen, Wohnort, Postleitzahl etc. Wollen Sie noch irgendein Feld in der Adresse haben, welches sich für Berechnungen verwenden läßt, so haben Sie schon zwei Typen in der Adresse, nämlich Felder der Typen »ARRAY OF CHAR« oder, wenn vorhanden, »STRING« und, für Rechnungen, »REAL« oder »INTEGER«. Aufgrund dieser Betrachtung fällt

Listing 12. In Basic fehlen Mengen

```
10 GET a$: IF a$ = "" GOTO 10
20 IF a$ = "+" THEN  REM ADDITION
30 IF a$ = "-" THEN  REM SUBTRAKTION
40 IF a$ = "/" THEN  REM DIVISION
50 IF a$ = "*" THEN  REM MULTIPLIKATION
60 IF ASC(a$) < 48 OR ASC(a$) > 57 GOTO 10
70 REM  HIER FOLGT DAS WEITERE PROGRAMM

In Pascal ist das alles viel einfacher:

READ (c);  (* 'c' SEI VOM TYP 'CHAR' *)
WHILE c IN ('0'..'9','+','-','/','*') DO
  (*                                   *)
  (* HIER FOLGT DAS WEITERE PROGRAMM *)
  (*                                   *)
```

```
PROGRAM MENGEN (INPUT,OUTPUT);
TYPE     farbe = (ROT,BLAU,GRUEN,GELB,BRAUN,WEISS,SCHWARZ);
           tag = (MON,DIE,MIT,DON,FRE,SAM,SON);
      farbton = SET OF farbe;
      besetzt = SET OF tag;
       zahlen = SET OF 1..500;
      zeichen = SET OF char;
   buchstaben = SET OF 'A'..'Z';
VAR    menge1,menge2 : farbton;
       woche1,woche2 : besetzt;
              antwort : besetzt;

BEGIN
  menge1 := (SCHWARZ,ROT,GRUEN,BLAU);
  menge2 := (SCHWARZ.GELB,BRAUN,BLAU,WEISS,ROT);
  woche1 := (MIT,FRE,SAM);
  woche2 := (MON,MIT,DON,FRE,SAM);
  WRITELN ('ZUERST DIE VEREINIGUNGSMENGE:');
  WRITELN (menge1 + menge2);
  WRITELN;
  WRITELN ('JETZT DIE SCHNITTMENGE:');
  WRITELN (menge1 * menge2);
  WRITELN;
  WRITELN ('NUN DIE DIFFERENZMENGE:');
  WRITELN (menge1 - menge2);
  WRITELN; WRITELN;
  WRITELN ('WANN KOENNEN SIE KOMMEN?');
  READLN (antwort);
  WRITELN;
  IF antwort <= woche2
     THEN WRITELN ('WIR HABEN IMMER ZEIT FUER SIE.');
  IF (antwort => woche2) AND ('DIE' IN antwort)
     THEN BEGIN
            WRITELN ('DIENSTAGS IST IMMER GESCHLOSSEN, ABER');
            WRITELN ('SONST KOENNEN SIE IMMER KOMMEN.');
          END;
  IF antwort => woche1
     THEN WRITELN ('SIE KOENNEN AUF JEDEN FALL IN DER 1.WOCHE KOMMEN.');
END.
```

Listing 13. Umgang mit Mengen, Aufzählungs- und Ausschnittyp

ein Array schon aus. Als »Ersatz« dafür bietet Pascal nun den Record. In ihm kann man nun beliebige Felder der verschiedensten Typen unterbringen. Die Records können nun so leicht angesprochen werden, wie die Elemente eines Arrays.

Ein Record wird im Typendeklarationsteil deklariert. Eine Vorbemerkung sei erlaubt: Alphanumerische Eingaben kann man im UCSD-Pascal als zum Typen »STRING« gehörig deklarieren, in Standardpascal muß man sie als »PACKED ARRAY OF CHAR« deklarieren. Beim Array kann man natürlich die Länge bestimmen, aber das ist bei Strings auch möglich und zwar folgendermaßen:

STRING (x) erzeugt Platz für einen String der Länge »x«. Ich habe im nächsten Beispiel (Listing 14) die Deklaration für eine Adreßdatei von 100 Adressen erstellt. Mit einem Index »i« habe ich nun Zugriff auf alle 100 Adressen.

Man könnte dies mit dem Statement »WITH« viel eleganter lösen, aber dieses Statement ist bis jetzt in keiner einzigen Pascal-Version für den Commodore 64 implementiert.

Im Record sind deshalb nur Felder vom Typ »STRING« benützt worden, weil man mit ihnen leicht eine Eingaberoutine programmieren kann, wie ich es oben getan habe. Für verschiedene Typen, die natürlich ihrerseits auch wieder Records sein können, muß man mehrere »READLN«-Routinen für verschiedene Typen einführen oder gegebenenfalls eine »STRING to INTEGER«-Konversion durchführen.

```
PROGRAM DATEI (INPUT,OUTPUT);
TYPE    maske = RECORD
                  VORNAME   : STRING[25];
                  NACHNAME  : STRING[25];
                  STRASSE   : STRING[25];
                  NUMMER    : STRING[4];
                  PLZ       : STRING[6];
                  WOHNORT   : STRING[25];
                  TELEFON   : STRING[14];
                  SATZNR    : 0..100
                END;
      felder = (VORNAME,NACHNAME,STRASSE,NUMMER,PLZ,WOHNORT,TELEFON);
VAR   adresse : ARRAY [0..100] OF maske;
            i : felder;
      eingabe : STRING;
            c : CHAR;
            j : INTEGER;

 PROCEDURE BILDSCHIRM;
 BEGIN
   WRITELN ('BITTE');
   CASE i OF
     VORNAME : WRITE ('IHREN VORNAMEN :');
     NACHNAME: WRITE ('IHREN NACHNAMEN:');
     STRASSE : WRITE ('DIE STRASSE    :');
     NUMMER  : WRITE ('DIE HAUSNUMMER :');
     PLZ     : WRITE ('DIE POSTLEIZAHL:');
     WOHNORT : WRITE ('IHREN WOHNORT  :');
     TELEFON : WRITE ('IHRE TELEFONNUMMER:')
   END
 END;

 PROCEDURE ZUWEISUNG;
 BEGIN
   CASE i OF
     VORNAME : adresse[j].VORNAME := eingabe;
     NACHNAME: adresse[j].NACHNAME:= eingabe;
     STRASSE : adresse[j].STRASSE := eingabe;
     NUMMER  : adresse[j].NUMMER  := eingabe;
     PLZ     : adresse[j].PLZ     := eingabe;
     WOHNORT : adresse[j].WOHNORT := eingabe;
     TELEFON : adresse[j].TELEFON := eingabe
   END
 END;

BEGIN
  j := 0;
  REPEAT
    PAGE (OUTPUT);
    FOR i := VORNAME TO TELEFON DO
      BEGIN
        REPEAT
          BILDSCHIRM;
          READLN (eingabe);
          ZUWEISUNG;
          adresse[j].SATZNR := j;
          WRITELN ('SIND ALLE ANGABEN KORREKT?  J/N');
          READ (c);
        UNTIL c = 'J'
      END;
    WRITELN ('NOCH EINE EINGABE?');
    READ (c);
    j := j + 1;
  UNTIL (c <> 'J') OR (j > 100)
END.
```

Listing 14.
Ein Adreßsatz wird definiert und Adressen eingegeben

```
PROGRAM DATEI (INPUT,OUTPUT,ADRESSE);
TYPE     maske = RECORD
                   (* SIEHE OBEN *)
                 END;
         liste = FILE OF maske;
VAR     adresse : liste;
BEGIN
  REWRITE (adresse);
  (*                                                    *)
  (* DIESE STANDARDPROZEDUR öFFNET EIN FILE ZUM SCHREIBEN *)
  (* FALLS DIESES FILE SCHON EXISTIERT, WIRD ES GELöSCHT  *)
  (*                                                    *)
END.
```

Listing 15.
Eine Datei wird angelegt

Mit dieser Eingaberoutine kann man nun alle 100 Adressen eingeben, wenn man will. Nun haben wir also schon eine einfache Adreßdatei aufgebaut. Das sollte man einmal in Basic versuchen, wobei das Programm aber vergleichbar viele Zeilen haben sollte.

Eine Routine zur Ausgabe der Adressen ist nach dem Studium des obigen Beispiels leicht erstellbar. Mit Angabe des entsprechenden Indexes hat man nun Zugriff auf alle Datensätze. Nun, was natürlich noch fehlt, ist die Abspeicherung der Datensätze auf irgendeinem externen Massenspeicher. Das ist in Pascal sehr einfach, ist aber mit zwei Einschränkungen behaftet. Die erste ist, daß man (in einfacher Weise) nur sequentiell abspeichern und lesen kann, die zweite, daß man ein File ausdrücklich als extern deklarieren muß, wenn man nicht will, daß es mit Beendigung des Programmes verschwindet (diese Tatsache kann man benützen, wenn man nur Zwischenwerte für zeitlich beschränkten Gebrauch extern abspeichern will). Die Deklaration und Eröffnung eines externen Files geschieht wie im Listing 15.

Hier gleich noch ein Wort zu »INPUT« und »OUTPUT«. Hier wird klar, was sie darstellen: Es sind Dateien. »INPUT« ist die Standard-Eingabedatei (meistens die Tastatur) und »OUTPUT« die Standard-Ausgabedatei (meistens der Bildschirm).

Anders als in Basic ist aber das Beschreiben des Files nicht so ohne

weiteres möglich. Jedes zu schreibende Element muß vorher in ein sogenanntes Dateifenster gebracht werden, denn nur dieses Fenster kann auf das File geschrieben werden. Das Dateifenster hat denselben Namen wie die dazugehörige Datei, man hängt ihm nur noch ein »^« hinten an. Die Standardprozedur, die nun das Dateifenster im File ablegt, heißt »PUT (dateifenstername)«. Das Fenster wird geschrieben und ist danach unbestimmt. Nachdem man also das Fenster belegt hat (hier also das Fenster »adresse^«), kann man es mit »PUT (adresse)« ins File schreiben. Aber Vorsicht: Man muß noch wissen, ob man an dieser Stelle überhaupt schreiben darf. Deshlab sollte man mit zum Beispiel »WHILE EOF (adresse) DO PUT (adresse)« jeweils testen, ob hier Platz frei ist oder nicht (EOF = End Of File). In unserem Beispiel (Listing 16) sähe dies so aus: Zuerst die Zuweisung (hier einmal mit »WITH«, vergleiche oben), dann das Abspeichern:

Damit hat man nun also einen Record abgespeichert. Das Zurückholen funktioniert ähnlich. Nehmen wir an, wir hätten mehrere Records abgespeichert, so befinden wir uns längst nicht mehr am Anfang des Files. Dorthin müssen wir aber, um erfolgreich einlesen zu können. Die Standardprozedur, die uns an den Anfang bringt und das File zum Lesen eröffnet, heißt »RESET (dateifenstername)«. Sie hat auch dafür gesorgt, daß bereits das erste Element des Files im Dateifenster »adresse^« bereitsteht. Analog zum Beispiel oben verhält sich die Belegung eines Records mit den Daten des Fensters. Dazu gibt es das Gegenstück zu »PUT«, nämlich die Standardprozedur »GET (dateifenstername)«. Damit man auch hier merkt, wann man aufhören soll zu lesen, sollte man mit dem Statement »WHILE NOT EOF (adresse) DO GET (adresse)« prüfen, ob das Ende des Files schon erreicht ist oder nicht. Sie sehen, mit dem Record läßt sich vieles sehr vereinfachen, wo es um große, verschiedenartige Datenmengen geht.

Pascal: Auch dynamisch

Ziemlich am Anfang wurde festgestellt, daß Pascal eine statische Speicherverwaltung besitzt. Das stimmt nur bedingt. Es gibt in Pascal dynamische Speicherverwaltung, die sogar dynamischer ist, als diejenige in Basic. In Basic ist es bekanntermaßen nicht erlaubt, ein Feld mit »DIM« ein zweites Mal zu dimensionieren.

```
BEGIN
  WITH adresse^ DO
    BEGIN
      VORNAME   := adresse[j].VORNAME;
      NACHNAME  := adresse[j].NACHNAME;
      STRASSE   := adresse[j].STRASSE;
      NUMMER    := adresse[j].NUMMER;
      PLZ       := adresse[j].PLZ;
      WOHNORT   := adresse[j].WOHNORT;
      TELEFON   := adresse[j].TELEFON
    END;
  IF EOF THEN PUT (adresse)
         ELSE WRITELN ('BESETZT !');
END;
```

Listing 16. Mit der With-Anweisung läßt sich die Dateneingabe in Listing 14 viel eleganter lösen

In Pascal kann man dies erreichen, man muß nur anstelle der Arrays »Dynamische Variablen« verwenden. Man überläßt damit dem Computer die Verwaltung des Speicherplatzes jeder einzelnen Variablen, die dafür überall im Speicher verteilt sein können, zum Beispiel Variable »a« bei 3508, »b« bei 26876 etc. Anstelle von Indizes benützt man dann eben sogenannte »Zeiger« (pointers). Der Computer verwaltet alle diese Zeiger selbst, man kann die Werte dieser Zeiger somit niemals auslesen. Mit dieser Methode machen wir unsere Adreßverwaltung dynamisch. Das heißt man sollte problemlos löschen, sortieren, ändern und vor allem erweitern können. Es ist leicht ersichtlich, daß dies natürlich mit solchen Zeigern sehr einfach ist, weil man nicht die Datensätze verschieben, sondern nur die Zeiger ändern muß. Eine Variable wird zum Zeiger, indem man dem Typennamen, auf den er zeigen soll, ein »^« voranstellt. In unserem Beispiel:

```
TYPE    maske          = RECORD
                           (* SIEHE
                         OBEN *)
                         END;
        maskzeiger     =^maske;
VAR     adr1,adr2      : maskzeiger;
        j              :^INTEGER;
```

Eine Deklaration schafft aber noch keinen Platz, da die Zeiger zuerst noch nicht definiert sind. Um nun wirklich im Speicher Platz für die dynamische Variable zu bekommen, gibt es die Standardprozedur »NEW (adr1) sucht Platz für den dynamischen Record »maske«, NEW (j) für die dynamische Variable »j«. Durch

adr1^.VORNAME := »PETER«;
j^:= 9;

werden dem ersten Feld im dynamsichen Record »adr1« der Name »PETER« zugewiesen und die dynamische Variable »j« bekommt via den Zeiger den Wert »9«. Beachten Sie den Unterschied:

»j« bezeichnet eine Zeigervariable auf ein Objekt vom Typ »INTEGER«.
»j^« bezeichnet eine dynamische Variable vom Typ »INTEGER«.
»adr1« bezeichnet eine Zeigervariable auf ein Objekt vom Typ »maske«.
»adr1^« bezeichnet eine ganze dynamische Variable vom Typ »maske«, das heißt einen ganzen dynamischen Record.
»adr1^.feld« bezeichnet ein Feld des dynamischen Records, auf den »adr1« zeigt.

Wenn man im Record ein Feld einführt, das selbst ein Zeiger ist, kann man sehr leicht verkettete Listen aufbauen etc. Speziell für diese Anwendung gibt es das Statement »NIL«. Wenn man nämlich einen Zeiger hat, der auf das Ende einer Liste zeigt, so darf er ja nichts mehr enthalten. Um diesen »Nichts«-Zustand genau zu definieren, kann man einem Zeiger den Wert »NIL« zuweisen. Die Verkettung beginnt mit einem Zeiger und endet mit »NIL«. Wenn man nun eine Sortierung der Datensätze durchführen will, so hat man im Wesentlichen »nur« die Zeiger neu zu berechnen, damit die Verkettung gemäß den Kriterien richtig verläuft. Es lassen sich so alle Arten von Datenstrukturen verwirklichen zum Beispiel Baum- oder Sternstrukturen.

Ich hoffe, daß man sich mit Hilfe meiner Ausführung einen kleinen Überblick über Pascal verschaffen konnte. Einen Schnellkurs über die Programmierung in Pascal ersetzt dieser Artikel sicher nicht. Der interessierte Leser wird ohnehin Fachliteratur lesen müssen, um alle Finessen von Pascal kennenzulernen.

Sechs Pascal-Versionen im Vergleich

Mir sind bis jetzt sechs komplette Pascalsysteme bekannt: Pascal 64, Pascal 64 V3.0, beide von Data Becker, das Pascal von Abacus Software, G-Pascal, das KMMM-Pascal und das Oxford Pascal. Darüber hinaus gibt es noch einen 4-pass UCSD-Pascal-Compiler, über den ich aber noch nichts sagen kann. Die 35 reservierten Pascalstatements (AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DONWTO, ELSE, END, FILE, FOR ,FUNKTION, GOTO, IF, IN , LABEL, MOD, NIL, NOT, OF, OR, PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT, SET, THEN, TO, TYPE, UNTIL, VAR, WHILE, WITH) werden nur von zwei Versionen vollständig unterstützt.

Im Pascal 64 fehlen: FILE nur bedingt, GOTO, IN, LABEL, NIL, PACKED, PROGRAM, RECORD, SET, TYPE und WITH.

Pascal 64 V3.0 enthält alle Standard-Pascalstatements!

Im Pascal von Abacus fehlen: FILE, GOTO, IN, LABEL, NIL, PACKED, PROGRAM, RECORD, SET und WITH.

Im G-Pascal fehlen: FILE, GOTO, IN, LABEL, NIL, PACKED, PROGRAM, RECORD, SET, TYPE und WITH.

Im KMMM-Pascal fehlen: GOTO, LABEL, SET und WITH.

Im Oxford-Pascal fehlt: kein einziges Standard-Pascalstatement!

Meiner Meinung nach ist ein Programm kein vernünftiges Pascal, wenn TYPE, SET und besonders RECORD fehlen, denn WHILE..DO, REPEAT..UNTIL, lokale Variabeln etc. bieten bekanntlich schon Simons Basic, welches dann auch noch Grafik und Sound recht gut unterstützt.

Es muß auch gesagt werden, daß lange nicht alle Standardprozeduren implementiert wurden. Ich möchte nun alle Versionen kurz besprechen.

Pascal 64

Dieses Pascal wird in Basic emuliert! Es ist kein UCSD-Pascal, das heißt, komplette Stringbehandlung fehlt, wichtige Standardprozeduren wie NEW, EOF, PAGE, PUT, GET, PACK, UNPACK fehlen. Es besitzt dagegen eine kleine Grafikunterstützung, Sprites, zwei Rechengeschwindigkeiten (INTEGER/REAL), PEEK, POKE und SYS. Die Compilierzeit ist unerträglich lang. Nach Aussage eines Benutzers brauchte der Compiler für 234 Zeilen Sourcetext zirka 45 Minuten. Der Editor ist, weil es der eingebaute Full-Screen-Editor ist, natürlich zeilenorientiert und verlangt zum Teil unsichtbare GeSHIFTete SPACEs als Begrenzer von Zahlen, was man natürlich oft vergißt. Es werden darüber hinaus keine Editierhilfen geboten. Der Compiler ist nach dem Laden resident ab der Basiczeile 10000. Man kann daher den Pascalcompiler selbst nicht mit dem Austrocompiler oder Petspeed verschnellern, weil man die Zeile 1 bis 9999 für das eigene Programm benutzen muß. Fehlermeldungen nur wie in Basic (zum Beispiel ERROR in 20). Der P-Code ist nur zusammen mit einem Lader ausführbar. Das Handbuch hat einige Druckfehler und Ungenauigkeiten.

Alle reservierten Pascalstatements sowie alle Standardprozeduren sind implementiert! Weiterhin Mathematikfunktionen und Unterstützungen für die Programmierung von Hires- und Multicolor-Grafik und Sprites. Die Soundprogrammierung und der UCSD-Typ 'STRING', der aber im Typendeklarationsteil

Pascal 64 V3.0

mit 'PACKED ARRAY (x..y) OF CHAR' definiert werden kann, werden nicht unterstützt. Bis zu vierdimensionale Arrays und komplexe Parameterübergabe an Prozeduren und Funktionen sind möglich. Als Anpassung an den Commodore 64 werden außerdem noch die relativen Files und ein bequemeres Diskettenhandling geboten. PEEK, POKE sowie Aufruf von Maschinenroutinen sind möglich. Eine Datenschnittstelle zum Profimat ist eingebaut. Als besonderes Feature gibt es die Interruptprocedure, die während eines jeden Interrupts ausgeführt wird, unabhängig davon, ob man sich noch im Pascalprogramm oder wieder im Basic befindet. Der Editor ist derselbe wie beim Pascal 64. Da aber der Compiler nicht schon wie beim Pascal 64 zur Erzeugung der Sourcefiles vorhanden sein muß, kann man Basiceditierhilfen wie zum Beispiel Exbasic Level II verwenden. Der langsame Compiler liest und compiliert von beziehungsweise auf Diskette. Nach erfolgreicher Compilierung muß ein spezielles Ladeprogramm eingeladen werden, das aus dem P-Code Maschinensprache und somit ein, ohne zusätzliche Hilfsprogramme, ablauffähiges Pascalprogramm erstellt. Erscheint ein Syntaxfehler, so steigt der Compiler aus ohne eine konkrete Angabe über Fehlerart und das Auftreten des Fehlers. Auch jetzt noch muß der Benutzer dem Compiler unverständlicherweise einige Parameter »von Hand« übergeben wie schon beim Pascal 64. Ein interessantes Detail für alle Beisitzer des alten Pascal 64: Nach Auskunft von Data Becker können alle diejenigen, die das alte Pascal 64 besitzen, dieses gegen zirka 50 Mark auf den Stand der neuen Pascalversion Pascal 64 V3. 0 aktualisieren lassen. Das neue Pascal 64 V3. 0 soll übrigens auch nur 99 Mark kosten.

Pascal von Abacus

Dieses Pascal wird in Basic emuliert! Abcus-Pascal ist kein UCSD-Pascal: die komplette Stringbehandlung fehlt, wichtige Standardprozeduren wie NEW, EOF, PAGE, PUT, GET, PACK, UNPACK sowie Mathematikfunktionen fehlen ebenfalls. Es besteht aus drei Teilen: Editor (5,5 KByte), Compiler (10 KByte) und Interpreter (7,25 KByte). Der Editor arbeitet zeilenorientiert. Ein Pascalsourcefile wird als sequentielles File abgelegt, das heißt es ist möglich, einen anderen Editor zu benützen. Der Compiler bearbeitet zehn kurze Sourcezeilen in 25 Sekunden. Beim Auftreten eines Fehlers wird nur ein Fehlercode ausgegeben und der Compiler steigt aus. Man muß dann zum Beispiel den Editor selbst laden. Im Falle eines Fehlers wird man also in keiner Form unterstützt. Der P-Code läuft nur mit Interpreter.

G-Pascal

G-Pascal ist kein UCSD-Pascal. Die komplette Stringbehandlung, wichtige Starndardprozeduren wie NEW, EOF, PAGE, PUT, GET, PACK, UNPACK, ORD sowie sämtliche Mathematikfunktionen, Type Real (kann nicht deklariert werden) und Boolean fehlen. Dafür bietet dieses Pascal eine extrem umfangreiche Sprite-, Grafik- und Soundunterstützung. Einiges mehr als zwei Dutzend Statements bietet G-Pascal allein für Grafik, Sound, Sprites (vor allem Kollisionen) und Timersteuerung. Es besitzt auch als einziges ein »GOTOXY«. Die Sprites scheinen IRQgesteuert. Maschinensprache-Routinen können aufgerufen werden. Abfragen für Joystick, Paddles, Lightpen sind vorhanden. Bitmanipulationen sind möglich. 16 KByte Maschinensprache werden geladen, und dann ist das Pascal inklusive Compiler und Editor resident. Der Editor ist ein zeilenorientierter Line-By-Line Editor recht beachtlichen Komforts, zum Beispiel mit Help-Befehl und Syntax-Checking Part vor der Compilierung. Fehlermeldungen werden im englischen Klartext ausgegeben. Der Compiler ist sehr schnell. Zudem bietet es einen Tracer, Debugger und Filer. Mit G-Pascal läßt sich bequem und übersichtlich arbeiten.

KMMM-Pascal

KMMM-Pascal bietet komplette Stringbehandlungen sowie alle Mathematikfunktionen, Type Text und einen Zufallsgenerator. Von allen Standardprozeduren fehlen nur PACK, UNPACK, PAGE. Automati-

sche Typenkonversion (String ↔ CHAR) und Bitmanipulationen sind möglich, wie auch Bitoperationen auf Integervariablen. Maschinensprache-Routinen können aufgerufen werden. PEEK und POKE sind auch möglich. Das Programm besteht aus drei Teilen: Editor (7,75 KByte), Editor/Compiler (23 KByte), Compiler (18,25 KByte) und Translator (19,5 KByte).

KMMM-Pascal erzeugt echten Maschinencode. Nach der Übersetzung kann man das Pascal-Programm abspeichern und nachher wie ein normales Maschinenprogramm laden. Es blendet das Basic-ROM aus, das heißt man hat neben dem Compiler noch runde 25 KBytes, für Pascalprogramme! Der Editor ist ein formatfreier Full Screen Editor überdurchschnittlichen Komforts (komfortabler als der eingebaute Editor), mit eingebautem Syntax Checking Part und belegten Funktionstasten. Eine häufig benötigte Steuercodesequenz kann als Macrobefehl definiert werden. Der Syntaxchecker spürt alle Syntaxfehler auf, wobei der Cursor die verursachende Stelle markiert und eine englische Klartextmeldung erscheint. Der Editor generiert ein sequentielles File, so daß man einen anderen Editor oder auch eine Textverarbeitung wie Easyscript benützen kann. Nach erfolgreichem Erstellen des Sourcefiles lädt man den

Schnelle Compiler

Compiler, der bei Bedarf ein ASCII — oder »PETSCII«-Listing erstellt, während er (übrigens sehr schnell) compiliert. Der P-Code ist abspeicherbar. Nach der Compilierung wird der Translator nachgeladen, der nun den P-Code ziemlich schnell in Maschinencode übersetzt. Nach dessen Fertigstellung wird die Kontrolle wieder dem Basic übergeben. Die Zeiger sind automatisch so gesetzt worden, daß man das Programm sofort abspeichern kann. KMMM-Pascal unterstützt den IEEE-Bus und erlaubt die Verwendung von DOS 5.1. Im Lieferumfang gibt es auch noch mehr als zehn Demoprogramme sowie ein 70-seitiges Manual. Dieses Pascal kann gegen eine sehr geringe Gebühr dem jeweiligen aktuellen Stand angepaßt werden, was einem die Herstellerfirma garaniert. Der Hersteller gab auf eine entsprechende Anfrage bekannt, daß bis Ende 1984 das KMMM-Pascal den Sprachumfang eines Jensen/Wirth-Pascals besitzen soll, wodurch es dann ein sowohl im Handling als auch in bezug auf die Compilerleistung sehr überzeugendes Pascal sein wird. Von der Firma kann man auch gegen etwa 13 SFr. eine Programmbibliothek der schon bestehenden Programme erstehen. Diese Pascalversion wird von der Firma seit 1983 für den Commodore 64 angeboten und betreut. Alle Programme, außer denjenigen, die sich mit 'SET' oder 'WITH' befassen, wurden unter KMMM-Pascal erstellt.

Oxford Pascal

Über dieses Pascal waren mir zu diesem Zeitpunkt erst diejenigen Informationen bekannt, die man einem englischen, nicht allzu umfangreichen Handbuch entnehmen kann. Es unterstützt alle 35 reservierten Pascalstatements und alle 40 Pascal-Standardprozeduren, alle Standardtypen (außer 'STRING', der aber leicht simuliert werden kann, siehe oben) sowie Mathematikfunktionen und Bitmanipulationen, auch auf Integern. Mengenoperationen sowie sogar variante Records(!) werden unterstützt. Darüber hinaus ist es auch an die Fähigkeiten des Commodore 64 angepaßt, was bedeutet, daß es Grafik und Sound unterstützt. Dazu besitzt es sehr komfortable Befehle, welche zum Beispiel in der Grafik Linien ziehen beziehungsweise löschen, Flächen ausfüllen beziehungsweise leeren und Punkte setzen beziehungsweise löschen oder prüfen können. Für den normalen Bildschirm gibt es die Prozedur »VDU«, die ein Zeichen an jeder beliebigen Stelle des Schirms ausgibt. Für den Sound gibt es die Prozedur »ENVEL«, die die Hüllkurve bestimmt, und für Frequenz, Wellenform und Dauer die Prozedur »VOICE« sowie eine eigene Prozedur »VOLUME«. Es unterstützt PEEK und POKE und erlaubt, in Pascal Maschinenprogramme zu editieren und als Pascalprozedur abzuspeicher. Weitere Features: Restore-Taste blockieren, Zugriff auf die Time Of Day-Clock, sedezimale Zahlenein- und -ausgabe, externe, linkbare Programm-Module, Forwarddeklarationen, Übergabeparameter dürfen auch Prozeduren und Funktionen sein, Prozeduren »PACK« und »UNPACK«, n-dimensionale Arrays. Es erzeugt Maschinencode. Es besitzt einen eigenen Editor, der im wesentlichen dem eingebauten Full-Screen-Editor entspricht, der aber um so nützliche Hilfen wie AUTO, DISK, NUMBER, FIND CHANGE, DELETE, PUT, GET, HEX, DECIMAL, DUMP, EX, LINK, COMPILE und einige Basicbefehle erweitert ist. Es gibt zwei Compiler. Der eine ist resident und compiliert und führt ein Programm sofort aus, was das Oxford Pascal vor allem für den Einsteiger, neben dem G-Pascal, sehr interessant macht. Der zweite Compiler ist ein Diskettencompiler, das heißt er liest und compiliert von beziehungsweise auf Diskette mit der Möglichkeit, externe Programm-Module einzubinden. Der Befehl »LOCATE« erzeugt danach ein ohne Hilfsmittel ablauffähiges Programm. Fehlermeldungen werden im englischen Klartext ausgegeben mit Angabe über die ungefähre Stelle im Listing.

Nur drei Versionen brauchbar

Meine persönlichen Erfahrungen haben gezeigt, daß bis auf das KMMM-Pascal, das Oxford Pascal (soweit man ein Programm theoretisch beurteilen kann) und dann noch das Pascal 64 V.3 0 keine einzige der anderen Versionen befriedigen kann. So ist beispielsweise der erste Pascal 64 sein Geld nicht wert, das neue Pascal 64 V3.0 desselben hingegen ist in bezug auf die Leistung des Compilers das krasse Gegenteil vom Pascal 64. Einen komfortablen Editor besitzen beide Versionen nicht, was besonders beim neuen Pascal 64 V3.0 ein echtes Ärgernis darstellt. Die Fähigkeiten des neuen Compilers wurden gegenüber demjenigen des Pascal 64 drastisch verbesert. Wer sich ein zwar auch nicht überzeugendes aber dennoch in puncto Grafik und Geschwindigkeit dem Pascal 64 überlegenes Pascal zulegen möchte, der sollte sich das G-Pascal besorgen. Das Pascal von Abacus ist nur unwesentlich besser als das Pascal 64, denn es bietet eigentlich nur ein spezielles Pascalstatement 'TYPE'. Von allen Versionen ist wohl das G-Pascal das bequemste, weil es menügeesteuert ist. Das Pascal 64 ist bei Laufzeitfehlern bis auf das G-Pascal am leichtesten editierbar. Weil das G-Pascal und das KMMM-Pascal einen eingebauten Syntax Checking Part besitzen, sind sie bei Syntaxfehlern am leichtesten editierbar. Bei allen Versionen außer dem KMMM-Pascal, dem neuen Pascal 64 V3.0 und dem Oxford Pascal ist das Programm nur lauffähig,

Fortsetzung auf Seite 163

Fortsetzung von Seite 54

wenn ein entsprechendes Ladeprogramm oder das Pascal selbst (G-Pascal) vorhanden ist. Betrachtet man die reine Compilierzeit, so schneidet das G-Pascal bei weitem am besten ab.

KMMM-Pascal schnell und mit gutem Editor

Bezüglich der Ablaufgeschwindigkeit ist die Sache klar: Das KMMM-Pascal und das neue Pascal 64 V3.0 sind klar die schnellsten (über das Oxford Pascal ist hierzu noch nichts bekannt). Ich benutze das KMMM-Pascal, denn für den engagierten Pascal-Anwender gibt es außerdem immer noch unbequemen Pascal 64 V3.0 auf dem Commdore 64 bis heute noch nichts besseres, obwohl wahrscheinlich das Oxford Pascal in näherer Zukunft eine Leaderstellung erreichen wird, außer man will sich die CP/M-Karte kaufen. Hat man einmal das Problem mit dem Diskettenformat gelöst, dann ist allerdings die Frage nach einem geeigneten Pascal akademisch. Das KMMM-Pascal kostet in der Schweiz zirka 280 SFr., das Oxford Pascal zirka 190 SFr., was im Vergleich zu den anderen Versionen, bis auf das neue Pascal 64 V3.0, dessen Preis auch 99 Mark sein wird, etwas hoch erscheinen mag. Falls diese Preise dem Einsteiger zu hoch sind, dann kann ich ihm höchstens noch das G-Pascal empfehlen, weil es dem Anfänger wohl eher auf leichte Editierbarkeit ankommt, als auf ein möglichst komplettes Pascal (Pascal 64 V3.0), das ihm die übrigen Versionen sowieso auch nicht bieten können. Ein Pascalsystem sollte eben nicht nur aus einem Compiler (Data Becker Pascals) bestehen, sondern auch einen guten Editor (KMMM-Pascal, G-Pascal, Oxford-Pascal) besitzen, womit man frustrationslos und einfach Sourcefiles erstellen kann. Pascal ist eben (leider!!) eine Compilersprache.

(Martin Baur)

Inserentenverzeichnis

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael M. Pauly (py)
Stellv. Chefredakteur: Michael Scharfenberger (sc)
Redakteure: aa = Albert Absmeier, leitender Redakteur (130), ev = Volker Everts (278), kg = Karin Gößlinghoff (269), gk = Georg Klinge (130), rg = Christian Rogge (278)
Redaktionsassistenz: Dagmar Zednik (237)
Fotografie: Janos Feister, Titelfoto: Alex Kempkens

Layout: Leo Eder (Ltg.), Willi Gründl, Walter Höß, Cornelia Weber

Auslandsrepräsentation:
Schweiz: Markt&Technik Vertriebs AG, Alpenstrasse 14, CH-6300 Zug, Tel. 042-22 31 55/56, Telex: 862329 mut ch
USA: M&T Publishing, 2464 Embarcadero Way, Palo Alto, CA 94303; Tel. 001-424 0600; Telex 752351

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlags AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Herstellung: Klaus Buck (180), Leo Eder (181)

Anzeigenleitung: Peter Schrödel (156)
Anzeigenverkauf: Alfred Reeb (211)

Anzeigenverwaltung und Disposition: Michaela Hörl (171)

Anzeigenformate: 1/1-Seite ist 266 Millimeter hoch und 185 Millimeter breit (3 Spalten à 58 mm oder 4 Spalten à 43 Millimeter). Vollformat 297 x 210 Millimeter. Beilagen und Beihefter siehe Anzeigenpreisliste.

Anzeigenpreise: Es gilt die Anzeigenpreisliste Nr. 1 vom 1. März 1984.
Anzeigengrundpreise: 1/1 Seite sw: DM 7400,-. Farbzuschlag: erste und zweite Zusatzfarbe aus Europaskala je DM 1000,-. Vierfarbzuschlag DM 3000,-. Plazierung innerhalb der redaktionellen Beiträge: Mindestgröße 1/1-Seite
Anzeigen im Einkaufs-Magazin: Die ermäßigten Preise im Einkaufs-Magazin gelten nur innerhalb des geschlossenen Anzeigenteils, der ohne redaktionelle Beiträge ist. 1/1-Seite sw: DM 5400,-. Farbzuschlag: erste und zweite Zusatzfarbe aus Europaskala je DM 1000,-. Vierfarbzuschlag DM 3000,- **Anzeigen in der Fundgrube: Private Kleinanzeigen** mit maximal 5 Zeilen Text DM 5,- je Anzeige. **Gewerbliche Kleinanzeigen:** DM 10,- je Zeile Text.
Auf alle Anzeigenpreise wird die gesetzliche MwSt. jeweils zugerechnet.

Vertriebsleitung, Werbung: Hans Hörl (114)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Plieninger Straße 100, 7000 Stuttgart 80 (Möhringen), Telefon (07 11) 72 04-0

Erscheinungsweise: 64'er, Magazin für Computerfans, erscheint monatlich, Mitte des Vormonats.

Bezugsmöglichkeiten: Leser-Service: Telefon 089/46 13-1 19. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen. Das Abonnement verlängert sich zu den dann jeweils gültigen Bedingungen um ein Jahr, wenn es nicht zwei Monate vor Ablauf schriftlich gekündigt wird.

Bezugspreise: Das Einzelheft kostet DM 6,-. Der Abonnementspreis beträgt im Inland DM 72,- pro Jahr für 12 Ausgaben. Darin enthalten sind die gesetzliche Mehrwertsteuer und die Zustellgebühren. Der Abonnementspreis erhöht sich um DM 18,- für die Zustellung im Ausland, für die Luftpostzustellung in Ländergruppe 1 (z.B. USA) um DM 38,-, in Ländergruppe 2 (z.B. Hongkong) um DM 58,-, in Ländergruppe 3 (z.B. Australien) um DM 68,-.

Druck: Druckerei E. Schwend GmbH, Schmollerstr. 31, 7170 Schwäbisch Hall

Verantwortlich: Für redaktionellen Teil: Michael M. Pauly.
Für Anzeigen: Peter Schrödel.

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/46 13-0, Telex 522052

Mitteilung gem. Bayerischem Pressegesetz: Aktionäre, die mehr als 25% des Kapitals halten: Otmar Weber, Ingenieur, München; Carl-Franz von Quadt, Betriebswirt, München. Aufsichtsrat: Dr. Robert Dissmann (Vorsitzender), Karl-Heinz Fanselow, Hans-Jochen Wolf, Eduard Heilmayr.

Telefon-Durchwahl im Verlag:

Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089-46 13 und dann die Nummer, die in Klammern hinter dem jeweiligen Namen angegeben ist.