Pascal-M
KIM-1 version
1979

Userguide (dutch only)

Collection Hans Otten – 1979

KIM - PASCAL

Korte gebruiksaanwijzing :

1 Laad Micro-Ade (= editor)

2 Schrijf Pascal programma en save (bv met S1) de Pascal source op cassette (zie Micro-Ade Manual)
Elke source regel wordt als aparte file geduumpt (automatisch)
KIM Pascal vereist twee cassette recorders met software besturing zoals in M.A. manual beschreven

3 Compilatie fase :
- Laad zero page Pascal
- Laad procedure buffer
- Laad interpreter
- Laad compiler

Start compilatie met 2003 G .
Compiler gaat nu source van cassette inlezen en verwerken. Object code wordt naar cassette gestuurd.
Fouten in source worden geprint. Foutloze source levert geen output.
Als compilatie succesvol was, kan de objectcode worden uitgevoerd. Verwissel cassette uit schrijf recorder naar lees recorder.
Start 2000G. Code wordt nu geladen.
Geprint wordt : (programma naam) LOADED.
Uitvoeren nu met 2003 G.
Herhaald uitvoeren : nogmaals 2003 G.

Alleen code uitvoeren : laad zero page + interpreter en dan
Compileren vereist zeropage, proc. buffer, interprete + compiler!

PASCAL MANUAL

VOOR DE 6502 KIM VERSIE

DOOR:F.J.S.JASPERS FOCKS
WESTVRIES COMPUTER CONSULTING
LIMMEN

PAR. INHOUD BLZ.

1 ..... INLEIDING 1
2 ..... ALGEMEEN 2
3 ..... STRUCTUUR VAN HET PROGRAMMA 3
4 ..... DATA 4
4.1 ..... CONSTANT DECLARATION SECTION 4
4.2 ..... TYPE DECLARATION SECTION 4
4.2.1 ... SIMPLE TYPE 4
4.2.2 ... STRUCTURED TYPES 5
4.3 ..... VARIABLE DECLARATION SECTION 8
4.4 ..... PROC/FUNC DECLARATION SECTION 8
5 ..... STATEMENTS 12
5.1 ..... ASSIGNEMENT STATEMENT 12
5.2 ..... COMPOUND STATEMENT 12
5.3 ..... IF STATEMENT 12
5.4 ..... FOR STATEMENT 13
5.5 ..... WHILE STATEMENT 13
5.6 ..... REPEAT STATEMENT 13
5.7 ..... CASE STATEMENT 14
6 ..... INTERPRETER EN COMPILER 15
6.1 ..... COMPILER 15
6.2 ..... INTERPRETER 15

1. INLEIDING

In dit manual wordt heel in het kort weergegeven hoe de programmeer taal PASCAL is opgebouwd en wat voor mogelijkheden hij biedt. Bij de behandeling van de diverse onderdelen is er van uitgegaan dat naast dit manual ook andere lektuur over PASCAL geraadpleegd zal worden. Er zijn dan ook slechts enkele voorbeelden van programma's gegeven en de nadere bijzonderheden van de diverse statements zijn niet verder uitgewerkt.

De interpreter die bij dit manual hoort is geschreven voor de 6502 versie op de KIM. Dit houdt in dat de I/O-routines en de jump-to-monitor aangepast zijn aan de KIM-monitor. Voor gebruik op andere, op de 6502 gebaseerde computers moeten deze dan ook gewijzigd worden. De I/O interface routines beginnen op adres $0206. De jump-to-monitor staat op adres $0237.

In de hier beschreven versie van PASCAL worden enkele data-structuren en statements, die de oorspronkelijke versie van PASCAL wel heeft en die ook in PASCAL USER MANUAL AND REPORT beschreven worden, niet genoemd omdat deze compiler die niet aankan. De voornaamste zijn: REAL'S, FILE-TYPE en WITH STATEMENT. Hoewel het ontbreken van REAL'S soms wel lastig kan zijn heeft dit als voordeel dat zowel de compiler als de interpreter betrekkelijk klein gebleven zijn. Daarnaast zou de executie-tijd ook aanzienlijk langer zijn geworden.

Voor hen die geinteresseerd zijn in en willen werken met PASCAL is het dan ook raadzaam de volgende boeken te lezen:

PASCAL USER MANUAL AND REPORT
Jensen en Wirth
Springer Verlag 1978

INLEIDING PROGRAMMEREN IN PASCAL
vd Wijgaart
Academic Service Den Haag 1977

2. ALGEMEEN

PASCAL is een computertaal die het moselijk maakt dat mensen hun ideeen aan een computer kunnen doorseven in een taal die ze zelf besrijven en niet alleen maar in een taal die ze seleerd hebben. PASCAL is mischien meer dan welke andere computer taal ook het eenvoudigst te besrijven , wat voor een deel komt omdat PASCAL sebruik maakt van een straisht forward behandelins van de meeste alsorithmen.

De ontwikkelins van PASCAL is sebaseerd op twee principiele doeleinden: Ten eerste het ontwerpen van een taal die seschikt is voor een systematisch onderricht in het programmeren. Studenten en programmeurs die als eerste taal PASCAL seleerd hebben, leren daarna dan ook heel vlot FORTRAN, COBOL of ALGOL en leveren veel beter, dat wil zessen betrouwbaarder en overzichtelijker werk dan diesenen die deze ondersrond missen.

Daarnaast heeft PASCAL tot doel om implementaties van deze taal te ontwikkelen die zowel betrouwbaar als efficient zijn op de huidise computers.

Als basis voor PASCAL heeft ALGOL 60 sediend, omdat het de vereisten t.a.v. onderricht beter benadert dan welke andere standaard taal. Hierdoor heeft PASCAL dezelfde structuur, in feite dezelfde vorm van de expressies, als ALGOL 60. De srootste verschillen tussen deze twee talen lissen op het sebied van de data structurerins: PASCAL maakt het de sebruiker moselijk zelf variabelen te definieren en kent het sebruik van records, sets en pointers.

3. STRUCTUUR VAN HET PROGRAMMA

Ten gevolge van de gestructureerde vorm van zowel de statements als de data is de vorm van een PASCAL programma aan bepaalde regels gebonden.

Een programma is gestructureerd in blokken. Elk blok bestaat uit een <HEADING> die de naam van het blok en z'n parameters aangeeft.De rest van het blok is onder te verdelen in een <DEFINITION PART>, waarin de constanten, types, variabelen, procedures en functions gedefinieerd worden, en een <ACTION PART>, die de algorithmen van het blok bevat.

```
bv.PROGRAM TEST(INPUT,OUTPUT);                    (* HEADING *)
   CONST MAXVALUE=32767; (*CONSTANT DECLARATION SECTION*)
   TYPE KLEUR=(ROOD,ORANJE,GEEL); (*TYPE DECL.  SECTION*)
   VAR BLOEM:KLEUR;    (* VARIABLE DECLARATION SECTION *)
       A,B,C:INTEGER;

   BEGIN          (* ACTION PART *)
     A:=MAXVALUE;
     BLOEM:=ROOD;
   END.       (* END ACTION PART *)
```

Op het eerste gezicht kan dit vastzitten aan bepaalde regels als een gemis aan vrijheid van de programmeur opgevat worden. Maar in plaats van het belemmeren in het uitvoeren en uitleven van z'n talenten, dwingt het hem ertoe om eerst een logisch programma uit te schrijven voordat hij achter het toetsenbord kruipt. De beperkingen hebben dan ook niet betrekking op de vorm van het programma, zoals bij veel andere talen het geval is (bv. reekseorienteerde talen als ALGOL). PASCAL staat extra spaties en CR. toe op elke plaats in het programma.

4. DATA

Elk BLOK begint met het declareren van de data in verschillende secties. Dit zijn, in de volgorde waarin ze gedeclareerd moeten worden:

4.1 CONSTANT DECLARATION SECTION

Het is moelijk om aan constanten een zinvolle naam te geven, waardoor het programma beter leesbaar wordt.

```
vb.  CONST A=35;
         MAXVALUE = 32767;
```

Over het algemeen behoren in PASCAL programma's geen onbenoemde constanten voor te komen.

4.2 TYPE DECLARATION SECTION

Het is moelijk om naast de <STANDARD TYPES>, dat zijn INTEGER, BOOLEAN en CHARACTER, zelf data types te creeren, die door de programmeur gedefinieerd worden.
Deze types zijn onder te verdelen in 2 hoofdsoorten:

4.2.1 SIMPLE TYPE

Dit zijn de fundamentele data types in PASCAL, waaruit elk <STRUCTURED TYPE> uit bestaat. Behalve de <STANDARD TYPE>, die dus niet meer gedeclareerd hoeven te worden, zijn er nog een tweetal <SIMPLE TYPES>. De eerste is het <SCALAR TYPE> en wordt gedefinieerd door eenvoudigweg de waarden op te schrijven die het nieuwe type aan kan nemen.

```
vb.  TYPE KLEUR = (ROOD, WIT, BLAUW);
         DAG = (MA, DI, WO, DO, VR, ZA, ZO);
```

Het is nu moelijk een function te declareren die als waarde aflevert: MA,DI of WO.

```
FUNCTION WERKDAG(A,B:INTEGER):DAG;  (*HEADER*)
  BEGIN
    IF A<B THEN WERKDAG:=MA
           ELSE IF A=B
             THEN WERKDAG:=DI
             ELSE WERKDAG:=WO
  END;
```

Daarnaast is er het <SUBRANGE TYPE>. Dit geeft van een nieuw type de onder en bovengrens aan van een al bestaand type. Het subrange type wordt m.n. gebruikt bij array's en heeft de volgende vorm:

```
TYPE DAG =(MA,DI,WO,DO,VR,ZA,ZO);
     WERKDAG=MA..VR;    (*SUBRANGE VAN DAG*)
     LETTER='A'..'Z';   (*SUBRANGE VAN CHAR*)
```

- 4.2.2 STRUCTURED TYPES -

Een structured type wordt gekarakteriseerd door de types van zijn samenstellende delen en door de manier waarop deze onderling samenhang vertonen. Een structured type kan al dan niet "gepacked" zijn. Bij een "packed" type wordt voor de opslag van gegevens een zo klein mogelijk deel van het geheuhen in beslag genomen. De programmeur moet dus zelf beralen of hij van deze mogelijkheid gebruik wil maken. Hieronder volgt een opsommins van de structured types.

<FILES>

Omdat deze compiler geen file-handling aan kan, wordt aan dit type verder geen aandacht geschonken. Voor geinteresseerden wordt verwezen naar het PASCAL USER'S MANUAL AND REPORT (Jensen en Wirth) blz.55 v.v.

<ARRAY'S>

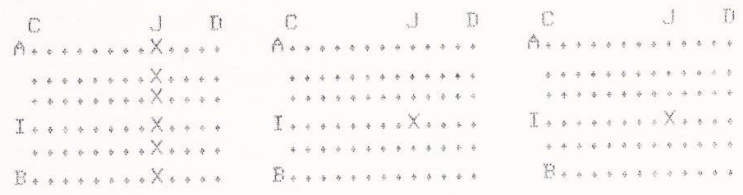
Een array is een begrensde verzameling van elementen, die allen van hetzelfde type zijn, het zgn. component of base type. Ieder element is direkt bereikbaar door de naam van de array aan te roepen, gevolgd door de zgn. index tussen vierkante haken. De onder en bovengrens liggen voor iedere index tijdens het compileren al vast en kunnen dus tijdens het programma niet berekend worden(i.t.t. ALGOL).
Het index type moet van een al bestaand scalar of subrange type zijn. Het standaard type INTEGER mag niet als index type dienst doen.

```
vb.TYPE GETAL = ARRAY[1..100] OF INTEGER;
          DAG = (MA,DI,WO,DO,VR,ZA,ZO);
         ZIEK = ARRAY[DAG] OF BOOLEAN;
```

De elementen van een array mogen zelf ook weer een array zijn, zodat meer-dimensionale matrices opgebouwd kunnen worden.

```
vb.TYPE TWEEDIM=ARRAY[A..B] OF ARRAY[C..D] OF
          OF INTEGER;
        MATRIX=ARRAY[A..B,C..D] OF INTEGER;
  VAR     TD:TWEEDIM;M:MATRIX;
```

Met TD[I] wordt een hele rij uit de matrix geselecteerd;
Met TD[I][J] 1 element uit die rij;
Met M[I,J] kunnen alleen enkelvoudige elementen geselecteerd worden;



Tussen array's met hetzelfde basis type is een assignment (:=) mogelijk. Het "packen" maakt een array van een ander type dan het niet gepackte.

```
vb.TYPE KLEUR=ARRAY[1..10] OF CHAR;
        TEKEN =ARRAY[1..10] OF CHAR;
        ASCII =PACKED ARRAY[1..10] OF CHAR;
   VAR K:KLEUR;T:TEKEN;A:ASCII;
```

Toegestaan is nu K:=T; fout is K:=A;

<RECORD TYPE>

In een record type kan data van verschillende types gegroepeerd worden tot een logisch geheel. I.t.t. array's is het niet nodig dat de data fields van hetzelfde type zijn.

```
vb.TYPE DATUM=RECORD MAAND:(JAN,FEB,MRT,APR,MEI,
                            JUN,JUL,AUG,SEP,OKT,NOV,DEC);
                     DAG:1..31;
                     JAAR:INTEGER;
                END;
```

Om nu gegevens in de record in te vullen kan worden volstaan met:

```
DATUM.MAAND:=NOV;
DATUM.DAG:=26;
DATUM.JAAR:=1953;
```

<SET TYPE>

De set type definieert een aantal waarden die behoren bij een al gedefinieerd type. Dit laatste type moet of een scalar type of een subrange type zijn.

```
vb.TYPE DAG=(MA,DI,WO,DO,VR,ZA,ZO);
        WERKDAG=SET OF DAG;
```

WERKDAG kan nu een of meerdere elementen van "DAG" bevatten.

In een programma kunnen sets als volgt opgebouwd worden:

```
PROGRAM DAGEN(INPUT,OUTPUT);
 TYPE DAG=(MA,DI,WO,DO,VR,ZA,ZO);
      WEEK=SET OF DAG;
 VAR VRIJ,WERK:WEEK;
     D:DAG;
 BEGIN
   WERK:=[MA..VR];VRIJ:=[ZA..ZO];
   IF WERK=VRIJ THEN WERK:=[];
               ELSE VRIJ:=[ZO];
 END.
```

<POINTER TYPE>

Het pointer type is een dynamische variabele. Tot nu toe zijn alleen statische variabelen behandeld. Deze worden aan het begin van een blok door hun declaratie van een naam voorzien. Het gebied waarbinnen ermee gewerkt kan worden is dan ook beperkt tot het blok waarin ze gedeclareerd zijn, en is als zodanig statisch uit de programmatekst af te leiden. De geheugenruimte, die per blok voor de daarin gedeclareerde statische variabelen en (in het geval van een proc/func) in de parameterlijst als valuegespecificeerde parameters nodig is, is al tijdens het compileren te bepalen. Dank zij de blokstructuur van PASCAL kan dan ook voor wat de statische variabelen betreft bij het werkgeheugen van de computer van een Last-In-First-Out stapel (STACK) gebruik worden gemaakt: bij het binnengaan van een blok wordt op de stack de benodigde ruimte gereserveerd, bij het verlaten van een blok wordt deze weer vrijgegeven.

Dynamische variabelen worden niet gedeclareerd, en hebben dus ook geen naam, maar worden tijdens de uitvoering van het programma gegenereerd m.b.v. de standaardprocedure NEW(P). Aan de pointer variabele P wordt een adres toegekend dat toegang geeft tot een variabele van het type waaraan P gebonden is. Men kan de pointervariabele d.m.v. een assignment een nieuwe (adres-)waarde toekennen. De pointer wijst dus naar deze variabele, die in deze versie van PASCAL alleen maar van het record type kan zijn.

```
vb.PROGRAM STARTOP(INPUT,OUTPUT);
   TYPE LINK=^PERSON;
      PERSON= RECORD SS:INTEGER;
                     NEXT:LINK;
                END;
   VAR FIRST,P:LINK; N,S,I:INTEGER;
   BEGIN
     FIRST:=NIL; N:=10;
     FOR I:=1 TO N DO
     BEGIN
       READ(S);NEW(P);
       P^.NEXT:=FIRST;
       P^.SS:=S;
       FIRST:=P;
     END;
   END.
```

Dit programma bouwt een "databank" op van 10 personen, waarvan de gegevens in de records staan. Iedere record bevat behalve het data gedeelte een pointer-veld. Dit bevat een variabele van het type LINK dat wijst naar de vorige record. De eerst ingelezen record wijst nergens naar (NEXT bevat NIL) en de variabele FIRST wijst naar de laatst ingelezen record.

4.3 VARIABLE DECLARATION SECTION

In de variable declaration section moeten alle variabelen die binnen een blok gebruikt worden gedeclareerd worden. PASCAL dwingt de programmeur dus om te overdenken wat hij wil zeggen, voordat hij het zegt.

```
vb.VAR X,Y:INTEGER;
       FLAG:BOOLEAN;
       KOSTEN:INTEGER;
```

4.4 PROC/FUNC DECLARATION SECTION

M.b.v. het procedure statement wordt de genoemde procedure uitgevoerd. Samen met het assignment statement vormt dit de onderdelen van structured statements.

Wanneer aan een statement een naam wordt gegeven, waardoor dat statement aangeroepen kan worden, heet dat statement een procedure of functie en de declaratie daarvan een procedure of function declaratie. Een procedure mag behalve de variabelen die in de declaratie voorkomen, ook nog lokale types en variabelen hebben. Deze mogen dan alleen binnen de procedure gebruikt worden.

vb.
```
PROGRAM SCHRIJF(INPUT,OUTPUT);
  VAR A,B:INTEGER;

  PROCEDURE TELOP(X,Y:INTEGER);
  VAR K:INTEGER;
  BEGIN
    X:=X+1; Y:=Y+1; K:=10;
    WRITELN (X,Y,K)
  END; (* TELOP *)

  BEGIN (* SCHRIJF *)
    A:=0; B:=0;
    TELOP (A,B);
    WRITELN (A,B)
  END.
```

```
   1    1
   0    0
```

In dit voorbeeld wordt PROC. TELOP uitgevoerd met als waarden voor X en Y resp. A en B. Merk op dat de waarden van A en B in het hoofdprogramma niet wijzigen. De variabele K is alleen in de procedure gebruikt en komt niet in het hoofdprogramma voor.

De parameters die bij het aanroepen van een proc/func. meegegeven worden bestaan uit twee soorten:

<VALUE PARAMETERS>
De waarde van de actuele expressie (het argument) wordt overgedragen aan de proc/func. Dit soort parameters is m.n. geschikt voor invoerparameters. Zie het programma hierboven.

<VARIABLE PARAMETER>
Bij het begin van de proc/func wordt eenmalig het adres van de betreffende parameter, die een variabele moet zijn, uitgerekend en elke keer dat de corresponderende parameter voorkomt, wordt er aan dit adres gerefereerd. Dit soort parameters is geschikt voor uitvoer en doorvoer (de parameter is dan zowel invoer als uitvoerparameter).

```
vb.PROGRAM SCHRIJF(INPUT,OUTPUT);
   VAR A,B:INTEGER;

   PROCEDURE TELOP(X:INTEGER; VAR Y:INTEGER);
   BEGIN
     X:=X+1; Y:=Y+1;
     WRITELN(X,Y)
   END; (* TELOP *)

   BEGIN (* SCRIJF *)
     A:=0; B:=0;
     TELOP(A,B);
     WRITELN (A,B)
   END.

      1        1
      0        1
```

Nu heeft de variabele B uit het hoofdprogramma de waarde van Y uit de procedure.

Omdat over het algemeen in de literatuur over PASCAL weinig aandacht geschonken wordt aan externe procedures, zal dit hier wat uitgebreider worden behandeld.

Externe proc/func. zijn proc/func. die zich buiten het programma bevinden en afzonderlijk gecompileerd zijn. Dit stelt de PASCAL programmeur in staat om gebruik te maken van een programma bibliotheek. Een andere toepassing is dat een deel van het programma in assembler geschreven kan worden. Met behulp van een externe procedure wordt dan naar dit stuk programma gesprongen. In het assembler gedeelte moet er wel voor gezorgd worden dat de variabelen van de stack gehaald worden en dat weer terug gesprongen wordt naar de PASCAL interpreter.

```
vb.PROGRAM ASSEMB(INPUT,OUTPUT);
   VAR A,B:INTEGER;

   PROCEDURE INIT(X:INTEGER);
   EXTERN=$2000;

   PROCEDURE RESULT(VAR Y:INTEGER);
   EXTERN=$2100;

   BEGIN (* HOOFDPROGRAMMA *)
     A:=10;
     INIT (A);
     RESULT(B);
     WRITELN(A,B)
   END.
```

Procedure INIT begint nu op $2000. En het volgende moet van de stack gehaald worden:
2 bytes die het aantal bytes dat van de stack gehaald moet worden aangeeft. Over het algemeen wordt hier weinig mee gedaan omdat dit aantal toch al wel bekend is.
2 bytes die de waarde van de parameter aangeven, in dit geval dus $000A.

Procedure RESULT begint op $2100 en hier moet het volgende van de stack gehaald worden:
2 bytes die het aantal bytes dat van de stack gehaald moet worden aangeeft.
2 bytes die het adres aangeven dat parameter B heeft. Omdat dit adres bekend is, kan aan B in het assembler programma één waarde toegekend worden.

5. STATEMENTS

5.1 ASSIGNMENT STATEMENT

Dit meest voorkomende statement wordt o.a. gebruikt om aan een variabele of functie een waarde toe te kennen. De vorm van het ASSIGNMENT statement is:

```
variabele:=expressie
```

```
vb.  A:=5;
     A:=A+2*B;
```

5.2 COMPOUND STATEMENT

Een stuk programma waarin meerdere statements tussen de statementhaken BEGIN en END staan heet een COMPOUND statement. COMPOUND statements kunnen genest voorkomen:

```
BEGIN
  ******
  BEGIN
    ******
    BEGIN
      ******
    END;
    ******
  END;
END.
```

5.3 IF STATEMENT

Met het IF statement bepaald men of een bepaald statement wel of niet uitgevoerd wordt. Dit gebeurd m.b.v. een Boolean vergelijking.

```
vb. IF CHR='P' THEN A:=10;
```

Wanneer ingeval CHR niet gelijk is aan 'P' wel een ander statement uitgevoerd moet worden, dan heeft het IF statement de vorm:

```
IF CHR='P' THEN A:=10 else A:=20;
```

Ook bij dit statement is nesting mogelijk:

```
vb.IF CHR='P' THEN
     IF CHR='M' THEN A:=10 ELSE A:=20
     ELSE A:=30;
```

Merk op dat voor ELSE geen ; mag staan.

5.4 FOR STATEMENT

Het FOR statement wordt gebruikt wanneer bij het insaan van een lus bekend is hoeveel maal de lus doorlopen dient te worden. Het FOR statement heeft de vorm:

```
FOR CV :=A TO B DO S
```
of
```
FOR CV :=A DOWNTO B DO S
```

Hierin is CV de control variabele met als beginwaarde Á. Na elke keer dat de lus doorlopen is wordt CV automatisch met 1 verhoogd of verlaagd. De laatste maal dat de lus doorlopen wordt is als de CV gelijk is aan B.
De control variabele en A en B moeten van hetzelfde scalaire type zijn en mogen door het statement S niet veranderd worden.

```
vb. SOM:=0;
    FOR N:=1 TO 100 DO
    BEGIN
      READ(X); SOM:=SOM+X;
    END;
    WRITELN(SOM);
```

5.5 WHILE STATEMENT

Het WHILE statement heeft de vorm:

```
WHILE BE DO S
```

De Boolean expressie BE bepaalt of het statement S wel of niet uitgevoerd wordt. Om uit de lus te komen is het dus zaak dat in het statement S BE gewijzigd kan worden. Is BE de eerste keer al false dan wordt S nooit uitgevoerd.
Een voorbeeld van het gebruik van het WHILE statement is het inlezen van een regel totdat de Boolean EOLN true is.

```
LINELENGTH:=0;
WHILE NOT EOLN AND (LINELENGTH<MAXCHAR) DO
BEGIN
  LINELENGTH:=LINELENGTH+1;
  READ(LINE[LINELENGTH])
END;
READLN;
```

5.6 REPEAT STATEMENT

Het REPEAT statement heeft de vorm:

```
REPEAT S1;S2;...;Sn UNTIL BE;
```

Het verschil met het WHILE statement is dat de statement S1..Sn uitgevoerd worden totdat BE true is. Ze worden dus altijd een keer uitgevoerd.

5.7 CASE STATEMENT

Wanneer aan variabele van het scalaire type verschillende waarden aan kan nemen en afhankelijk van deze waarde een bepaald statement uitgevoerd moet worden kan het CASE statement gebruikt worden i.p.v. het IF statement, omdat dit laatste statement in dat geval een ers lange en onoverzichtelijke notatie vereist.

Een voorbeeld van een CASE statement:

```
CASE CHR OF
  'P' : WRITE('PROGRAM');
  'B' : WRITE('BEGIN');
  'E' : WRITE('END');
  'U' : WRITE('UNTIL');
  'F' : WRITE('FOR');
  'D' : WRITE('DO');
END;
```

6. INTERPRETER EN COMPILER

Om PASCAL op een KIM te kunnen draaien zijn twee programma's nodig: een compiler en een interpreter.

6.1 COMPILER

Een compiler is een programma dat de statements van een hogere programmeertaal (zoals PASCAL en FORTRAN) vertaalt in een gelijkwaardig programma in de een of andere voor de machine afhankelijke vorm (zoals machine code). In ons geval wordt er door de compiler geen machine code gegenereerd, maar een code voor een hypothetische computer, de zgn. P-Computer. Het voordeel hiervan is dat de compiler los van een bepaalde machine geschreven kan worden.

6.2 INTERPRETER

De interpreter maakt van de KIM een fictieve P-machine, die werkt met de door de compiler gegenereerde P-code. I.t.t. de compiler is de interpreter dus wel machine afhankelijk. Omdat voor een PASCAL statement meerdere P-codes gegenereerd worden, die elk weer uit een aantal machine codes bestaat, is het zaak om de interpreter zo efficient moglijk te maken.

Omdat de compiler nogal veel geheugen inneemt(20K), blijkt dat een KIM met 24K geheugen minimaal vereist is. Zo gauw echter de PASCAL programma's groter worden dan 25 regels is een uitbreiding tot minstens 32K noodzakelijk, omdat anders al gauw een "STACK OVERFLOW" optreedt.

Nadat de compiler de P-code gegenereerd heeft, wordt de compiler niet meer gebruikt en kan de geheugenruimte voor het PASCAL programma gebruikt worden. Fig.1 laat zien hoe een PASCAL programma ontwikkeld dient te worden.

ELEMENTS OF A PASCAL PROGRAM

- o CONST Definition
- o TYPE Definition
- o VAR Declaration
- o Procedure & Function Declarations
- o Statement Part

CONST Definition

```
CONST
  K1  = 1;
  K10 = 10;
  MAXCNT = 1000B;
```

TYPE Definition

```
TYPE
  T1 = PACKED ARRAY[1..10] OF CHAR;
  T2 = 0..15;
```

VAR Declaration

```
VAR
  A : INTEGER;
  B : T1;
  C : RECORD
        F1 : T2;
        F2 : CHAR;
        F3 : ARRAY[0..5] OF INTEGER;
      END;
```

Procedure & Function Declaration

```
PROCEDURE TEST(X,Y : INTEGER;
               VAR Z : INTEGER);
  BEGIN
    Z := Z + Y;
  END;
```

Statement Part

```
BEGIN
  Statement list
END.
```

ARRAYS

```
VAR
  AREA : ARRAY[1..5,1..10] OF INTEGER;
  I    : INTEGER;

BEGIN
  I := AREA[2,7] + 10;
  AREA[2*I-9,3] := 32000;

END;
```

RECORDS

```
VAR
  INREC : RECORD
            FLD1 : INTEGER;
            FLD2 : 0..20;
            FLD3 : RECORD
                     XRAY : -10..10;
                     POSTVE: BOOLEAN;
                     IND : CHAR;
                   END;
            FLD4 : CHAR;
          END;

BEGIN
  INREC.FLD2 := 19;
  INREC.FLD3.IND := "W";

END;
```

RECORD VARIANTS

```
TYPE
   T1 = RECORD
          F1 : CHAR;
          F2 : INTEGER;
            CASE F3 : INTEGER OF
             0 : (CNTX : INTEGER;
                  AREAX : ARRAY[1..5] OF CHAR);
             1 : (FLAG1 : BOOLEAN);
             2 : (B2 : -10..10);
             3 : (TABLE : ARRAY[1..10,1..10] OF INTEGER);

        END;

VAR

   A : T1;
   B : T1;

BEGIN
   A.CNTX := 10;
   B.FLAG1 := TRUE;

END;
```

SETS

```
TYPE
  T1 = [E1,E2,E3,E4, E5,E6,E7,E8,E9,
        E10,E11,E12,E13,E14,E15,E16];
  T2 = SET OF T1;
  T3 = 0..15;

VAR

  SET1 : T2;
  SET2 : T2;
  SET3 : SET OF T3;

BEGIN
  SET1 := SET1 + SET2;
  SET1 := SET1 - [E1,E5,E8];
  IF E1 IN SET1
    THEN SET2 := SET1;
  SET3 := [5,9,12,14];

END;
```

DYNAMIC VARIABLES

```
TYPE
  PTR = ^LNKLST;
  LNKLST = RECORD
             LNKPTR : PTR;
             A : INTEGER
           END;

VAR
  PTRV1 : PTR;
  PTRV2 : PTR;
  I : INTEGER;

BEGIN
  PTRV2 := NIL;
  FOR I : 1 TO 10 DO
    BEGIN
      NEW(PTRV1);
      PTRV1^ .LNKPTR := PTRV2;
      PTRV1^ .A := 0;
      PTRV2 := PTRV1;
    END;

END;
```

OPERATORS

Arithmetic

+	addition
-	subtraction
*	multiplication
div	division
mod	modulus

Relational (arithmetic)

=	equal
<>	not equal
<	less than
<=	less than or equal
>	greater than
>=	greather than or equal

Boolean

NOT	not
OR	or
AND	and

SET

+	union (or)
*	intersection (and)
-	difference (xor)

Relational (set)

=	equal
<>	not equal
<=	set inclusion
>=	set inclusion
in	membership

STATEMENTS

Assignment statement

```
I := J + 1;
```

Procedure statement

```
PROC1;
PROC2(X,Y,Z);
```

Compound statement

```
BEGIN
  I := J;
  PROC1
END;
```

IF statement

```
IF I = J
  THEN PROC1
  ELSE PROC2(A,B,C);
```

CASE statement

```
CASE NODE OF
  1 : BRNCH := 1;
  2 : BRNCH := 2;
  3 : BRNCH := 3
END;
```

REPEAT statement

```
REPEAT
  I := J + 1;
  PROC1
UNTIL I>10;
```

WHILE statement

```
WHILE I<=10 DO
  BEGIN
    I := J + 1;
    PROC1
  END;
```

FOR statement

```
FOR I := 1 to 10 DO
  BEGIN
    J := I * 100;
    PROC1;
    K := J - 3
  END;
```

FUNCTION statement

```
FUNCTION NEXT(CT:INTEGER):INTEGER;
  CONST
    INC = 1;
  BEGIN
  NEXT := CT+INC;
  END;

COUNT := Next(COUNT);
```

INPUT/OUTPUT

```
INPUT: READ(PARAM);      READ PARAMETER LIST
       READLN(PARAM);    READ PARAMETER LIST
                         AFTER PROCESSING PARAM
                         WAIT UNTIL END OF LINE
       (PARAM)       VARIABLES OF CHAR OR INTEGER

  VAR CH : CHAR; INT : INTEGER;
  BEGIN
   READLN (CH, INT);
                    INPUT  A1(CR)

OUTPUT: WRITE(PARAM)     WRITE PARAMETER LIST
        WRITELN(PARAM)   WRITE PARAMETER LIST
                         AND CLOSE LINE WITH CR,LF
        (PARAM)       VARIABLES OF
        (PARAM:LENGTH)   CHAR
                         STRING
                         STRING CONSTANT
                         INTEGER
  WRITE(CH:10,"STRING")
                       GIVES
                 9 BLANK-CH-STRING
```

STANDARD PROCEDURES/FUNCTIONS

EOLN
EOF
NEW( )
RELEASE( )
RESET
EXIT
HALT
READ( )
READLN( )
WRITE( )
WRITELN( )
SUCC( )
PRED( )
ODD( )
ORD( )
CHR( )