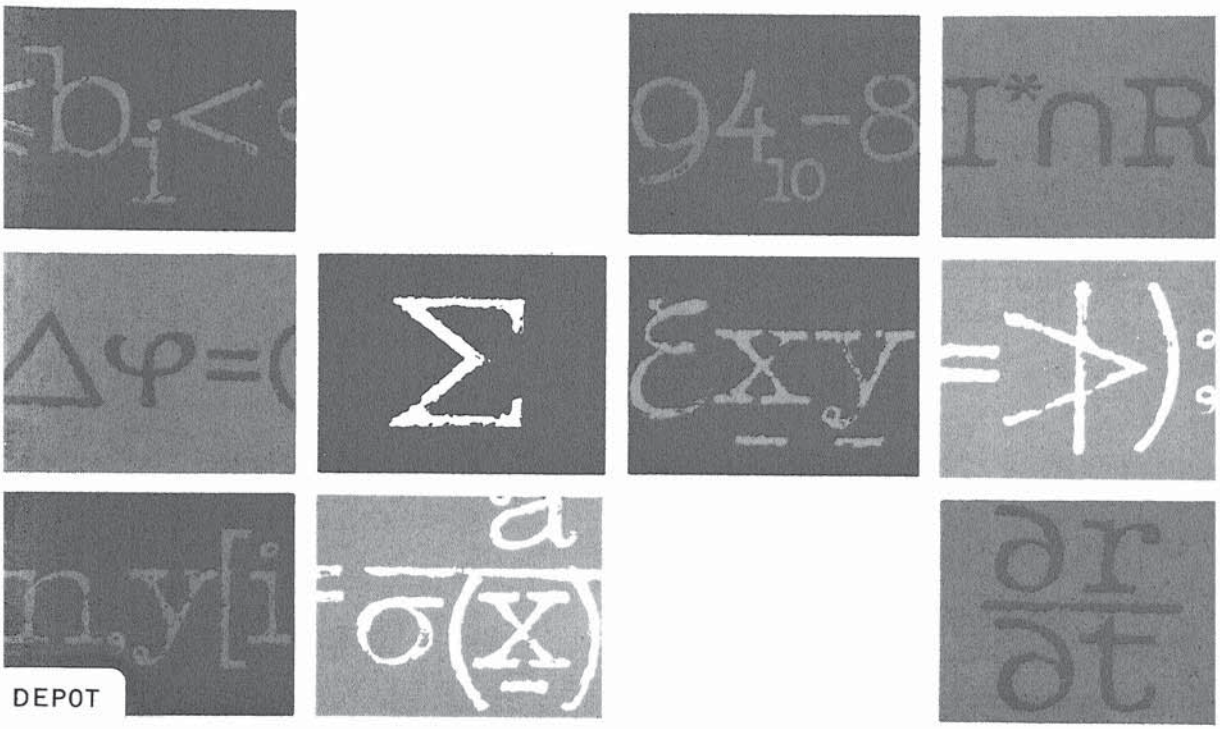
CURSUS ALGOL 60

K.K.KOKSMA



DEPOT
04989

M C SYLLABUS 6

MC SYLLABUS
6

CURSUS ALGOL 60

DOOR

K.K.KOKSMA

RA

Tweede Druk

MATHEMATISCH CENTRUM AMSTERDAM
1970

```
AAAAAAAA    A           AAAAAAAA    AAAAAAAA    A
AAAAAAAA    A           AAAAAAAA    AAAAAAAA    A
AAAAAAAA    A           AAAAAAAA    AAAAAAAA    A
AAAAAAAA    A           AAAAAAAA    AAAAAAAA    A
L      L    L           L      L    L      L    L
L      L    L           L      L    L      L    L
L      L    L           L      L    L      L    L
L      L    L           L      L    L      L    L
G      G    G           G           G      G    G
G      G    G           G           G      G    G
G      G    G           G           G      G    G
G      G    G           G           G      G    G
OOOOOOOO    O           O   OOOO    O      O    O
OOOOOOOO    O           O   OOOO    O      O    O
OOOOOOOO    O           O   OOOO    O      O    O
OOOOOOOO    O           O   OOOO    O      O    O
L      L    LLLLLLLL    LLLLLLLL    LLLLLLLL    LLLLLLLL
L      L    LLLLLLLL    LLLLLLLL    LLLLLLLL    LLLLLLLL
L      L    LLLLLLLL    LLLLLLLL    LLLLLLLL    LLLLLLLL
L      L    LLLLLLLL    LLLLLLLL    LLLLLLLL    LLLLLLLL
```

L.S.

Deze syllabus is in eerste instantie bedoeld om te gebruiken bij de cursussen ALGOL 60, die het Mathematisch Centrum organiseert. In deze cursussen worden de karaktertrekken van ALGOL 60 aan de hand van volledige programma's besproken.
De lezer zij gewaarschuwd dat hierdoor veel begrippen in hoofdstuk I en II wat losjes worden ingevoerd en dat zij pas later grondig worden besproken.
Alle programma's uit de syllabus zijn gedraaid op de X8 onder het MC-ALGOL 60 systeem.
Het vermenigvuldigprocédé van deze syllabus (foto-offset) laat meerdere representaties van een programma toe. Een programma getypt met een I.B.M. schrijfmachine vindt men b.v. op blz. 9. Een afdruk van een programma met behulp van een flexowriter vindt men b.v. op blz. 17, 18, 64. Een afdruk door het MC-ALGOL 60 systeem, zoals die gegeven wordt als het programma ter executie wordt aangeboden vindt men b.v. op blz. 73 en 74.
Voor een instructeur die niet over het MC-ALGOL 60 systeem beschikt is een map van alle programma's voorzien van output verkrijgbaar.

De schrijver wil zijn erkentelijkheid uitdrukken voor de vele op- en aanmerkingen van andere medewerkers van het Mathematisch Centrum.

K.K.K.

Afgezien van enige kleine correcties is de tweede druk nagenoeg gelijk aan de eerste.

K.K.K.

Inhoudsopgave

I. Inleiding 1

1. algorithme, programma, ALGOL 60, MR 81 1
2. ponsband, getallenband, executie, analyse 2
3. assignment statement, input, output 3
4. for statement, compound statement, if statement 5
5. conditional statement 8

II. Declaraties 10

1. quantity 10
2. label 11
3. array 12
4. switch 15
5. block, scope, procedure 19

III. Syntax en Semantiek 24

1. algemeen 24
2. metalinguïstische variabele, metalinguïstische formule, terminale productie, identifier 25
3. Syntax als middel om de Semantiek te verkorten 26
4. naam, basic symbol, ALGOL 60 programma 27
5. spreken over ALGOL 60 30

IV. Expressies 34

1. algemeen 34
2. arithmetische expressie 34
3. Boolean expressie 38
4. designational expressie 40

V. Statements 41

1. overzicht, basic statement 41
2. conditional statement 43
3. for statement 46

VI. Procedures 51

1. eenvoudige procedures, overzichtelijkheid en inzicht in de werking van het programma, call by name 51
2. call by value of call by name?, Jensen's device, function designator 57
3. overzicht actuele parameters 60
4. bij het formeel-actueel vervangingsmechanisme gaat het natuurlijke boven het letterlijke, omsluiting met haakjes, conflicten tussen identifiers 61
5. principe van de volledige inductie, recursieve procedures 65

Index 75

I. Inleiding

1. algorithme, programma, ALGOL 60, MR 81

Vele problemen zijn te herleiden tot de vraag hoe te manipuleren met zekere getallen. Het antwoord op deze vraag wordt gegeven in de vorm van een recept - een zgn. algorithme.
Een taal, waarin een algorithme zo duidelijk kan worden beschreven dat de uitvoering van de algorithme - het eigenlijke manipuleren - aan een rekenautomaat kan worden overgelaten heet een programmeertaal. Een beschrijving van een algorithme in een programmeertaal noemt men een programma.
Wil men een programma, dat in een bepaalde programmeertaal geschreven is, door een rekenautomaat laten uitvoeren, dan moet men over een "implementatie" van die bepaalde programmeertaal op die rekenautomaat beschikken. Een probleem oplossen met behulp van een rekenautomaat houdt dus zowel de keuze van een programmeertaal als van een implementatie in.
In deze syllabus is als programmeertaal ALGOL 60 gekozen en als implementatie het MC-ALGOL 60-systeem voor de X8 *).
ALGOL 60 is gedefiniëerd in het Revised Report on the Algorithmic Language ALGOL 60, kortweg Revised Report genaamd.
Een programma ter verwerking aangeboden aan het MC-ALGOL 60-systeem voor de X8, moet voldoen aan de regels beschreven in:
"Het MC-ALGOL 60-systeem voor de X8, voorlopige programmeurshandleiding" in het vervolg aangeduid met MR 81.
Wil men voor een ander systeem (implementatie) in ALGOL 60 programmeren, dan heeft men een andere handleiding nodig, die - zoals MR 81 voor het MC-ALGOL 60-systeem voor de X8 doet - de specifieke eigenschappen en functies van dat systeem beschrijft.
De bedoeling van deze syllabus is, dat de lezer door het bespreken van eenvoudige programma's leert programmeren in ALGOL 60. Deze opzet leidt onvermijdelijk tot confrontatie met de regels uit MR 81. Waar dit ge-

---

*) De grens tussen specifieke implementatie en rekenautomaat wordt in deze syllabus niet aangegeven. Deze grens valt samen met die tussen hardware (rekenautomaat) en software. Software omvat meer dan de specifieke implementatie, is nl. alles wat geprogrammeerd wordt.

beurt zal dit zoveel mogelijk vermeld worden, teneinde verwarring van ALGOL 60 regels met specifieke computer-restricties te vermijden.

2. wijze van behandeling van programma's, ponsband, getallenband, executie, analyse

In deze syllabus worden verschillende programma's behandeld. Bij de behandeling van een programma hebben we enerzijds te maken met de vorm - de tekst, zoals afgedrukt - anderzijds met de betekenis. Om onderdelen van het programma eenvoudig aan te kunnen geven zal soms een regelnummer, dat met het programma zelf niets te maken heeft, vooraan de regel afgedrukt worden. De betekenis zal of uitgelegd worden met zinnen als "de rekenautomaat doet dit of dat" of met zinnen als "er wordt dit of dat gedaan". Dit laatste om te benadrukken dat de betekenis van een ALGOL 60 programma niet afhangt van een rekenautomaat en daarom ook geschikt is als communicatiemiddel tussen mensen onderling.
Opgemerkt zij dat waar er in deze syllabus sprake is van "de rekenautomaat doet dit of dat" er - tenzij anders vermeld - van uitgegaan wordt, dat de rekenautomaat al onder controle van het programma werkt. De fase, waarin de verwerking van het programma door een rekenautomaat dan verkeert, heet executie. Hieraan vooraf gaat een analyse van de aangeboden tekst. Alleen als de tekst opgevat kan worden als een ALGOL 60 programma, zal de rekenautomaat na analyse overgaan tot executie van het programma (het programma uitvoeren).
Bij de oplossing van een probleem met behulp van een rekenautomaat wordt onderscheid gemaakt tussen het programma en indien aanwezig de data (gegevens of getallen), die door het programma verwerkt worden. Beide moeten zij via een ponsband *) (of een ander medium) in de rekenautomaat worden ingebracht. Pas als het programma in executie is zal de rekenautomaat op de data reageren.

---

*) Het ponsen van een programmaband of getallenband is mogelijk met behulp van een flexowriter. Men kan zich een flexowriter het beste voorstellen als een typemachine, waaruit behalve een afdruk ook een ponsband komt, waarvan de gaatjespatronen (bv. volgens de MC-flexowritercode) informatie representeren.

In de praktijk noemt men de ponsband, die de data bevat, de getallenband. Voor de te bespreken programma's is het nuttig een getallenband te zien als een ding, dat een rij getallen bevat; de getallen van deze getallenband kunnen door de rekenautomaat onder controle van het programma één voor één verwerkt worden.
Bovengenoemde rij heeft de volgende eigenschappen:

1. Er is een eerste getal.
2. Op het laatste na heeft ieder getal een opvolger.
3. Zolang de rekenautomaat de getallenband verwerkt is er steeds één getal aan de beurt.

Als de getallenband in de rekenautomaat gelegd wordt, houdt dat in, dat het eerste getal aan de beurt komt.

3. assignment statement, Input, Output

We bekijken een programma dat een getal van een getallenband leest en daarna afdrukt.

```
begin comment 1739, N. van de Auteur (MC), druk getal af;
      real a;
      a:= read;
      print(a)
end
```

Elk ALGOL 60 programma heeft een begin en een end symbool, die aangeven waar het programma begint en waar het ophoudt.
Na begin mag willekeurig commentaar toegevoegd worden tussen comment en puntkomma. Dit commentaar heeft geen invloed op de werking van het programma en is bestemd voor de menselijke lezer. Het is een goed gebruik in het commentaar tenminste te vermelden:

a opdrachtnummer
b naam van de auteur
c onderwerp.

We gaan nu over tot de bespreking van de betekenis van het programma.
We zien in het programma "read" staan. Dit is een aanroep van een functie

die tijdens executie bewerkstelligt dat

1. de waarde van het getal, dat aan de beurt is op de getallenband, wordt afgeleverd;

en 2. als er een volgende getal is, dit aan de beurt komt.

Is er geen volgende getal dan is de getallenband geheel verwerkt; wat er als de band geheel verwerkt is gebeurt als het programma toch weer read aanroept, hangt af van de implementatie. (In het MC-ALGOL 60-systeem wordt een foutmelding gegeven en de verwerking van het programma afgebroken.)

Onder punt 1 is sprake van het afleveren van een waarde, aan wie dan wel? De waarde wordt afgeleverd aan de drager a. In het programma wordt dit aangegeven door a := read

(spreek uit a wordt read).

"a:= read" heet in ALGOL 60 een assignment statement. De werking van een assignment statement is: de waarde afgeleverd door de expressie rechts van het := teken toekennen aan de drager links van het wordt-teken.

Het is in ALGOL 60 verplicht een drager, voordat hij gebruikt wordt, door middel van een declaratie (zoals <u>real</u> a) te introduceren d.w.z. zijn naam en functie te noemen. De naam van de drager is a, zijn functie is dat hij waarden van het type <u>real</u> (dat zijn gehele of decimale getallen als 2, 2.5 en -3.14) kan dragen. Behalve in de assignment statement "a:= read" wordt a ook gebruikt in de procedure statement "print(a)".

In de ene statement wordt het getal ingelezen, in de andere wordt het geprint.

ALGOL 60 eist dat statements onderling gescheiden worden door een puntkomma. Om het programma goed te kunnen overzien staat elke te behandelen eenheid op een aparte regel. Voor de betekenis van een ALGOL 60 programma tellen overgang op een nieuwe regel en spaties (tenminste buiten strings, waarover later meer) niet mee. Het programma kan dus ook als volgt worden geschreven:

"<u>begin</u> <u>real</u> a; a:= read; print(a) <u>end</u>".

MR 81 eist dat 2 word-delimiters (scheiders die een onderlijnd woord zijn, zoals real, begin) onderling gescheiden worden door een spatie, "overgang op een nieuwe regel" of een "tabulatie" (dat is één teken in de ponsband, dat bij afdrukken van de band de positie op de regel tot de eerstvolgende tabulatorstop opschuift) en voorts dat na het laatste end nog een flexowritersymbool volgt, bij voorkeur overgang op een nieuwe regel.
In het algemeen zal een programma uit statements en declaraties bestaan. In de statements wordt het eigenlijke proces beschreven, terwijl in de declaraties verteld wordt welke functies de grootheden vervullen, die door de in de statements voorkomende namen worden aangegeven.

Men zal zich wellicht afvragen of read en print dan geen namen zijn en zo ja waarom ze dan niet gedeclareerd worden. Het antwoord hierop is dat read en print inderdaad namen zijn, maar dat deze namen zonder meer aan het MC-ALGOL 60-systeem voor de X8 bekend zijn.
Van dit soort namen zijn er nog meer en de meeste houden verband met de Input (zoals read) en Output (zoals print); dit komt omdat een input of outputproces nooit geheel in ALGOL 60 beschreven kan worden. De werking van read zoals boven uitgelegd staat dan ook in MR 81, waar al deze procedures vermeld staan.
Men dient zich goed te realiseren, dat in een ander systeem in plaats van read iets anders gebruikt moet worden; wat dat is hangt af van de specifieke implementatie, waarmee men werkt.

4. for statement, compound statement, if statement

Een iets algemener programma dan het vorige is:

```
begin comment 1739, N. van de Auteur (MC), druk 100 getallen af;
      real a; integer i;
      for i:= 1 step 1 until 100 do
          begin a:= read; print(a) end
end
```

We herkennen in het programma de twee statements uit I-3

nl. de assignment statement: "a:= read"

en de procedure statement : "print(a)"

Deze twee statements zijn, door het begin symbool ervoor en het end symbool erachter, aanééngesmeed tot één statement: een zogenaamde compound statement. (Een compound statement is een rij statements onderling gescheiden door puntkomma's en omsloten door begin en end.)

De teksten "real a" en "integer i" zijn declaraties. Zij geven aan dat er dragers zijn met de naam "a" resp. "i", die waarden van het type real resp. integer (dat zijn gehele getallen als 0, 1, 2 en -3) kunnen dragen. De for clause:

```
for i:= 1 step 1 until 100 do
```

geeft aan, dat bij executie van het programma, de statement na do steeds weer moet worden uitgevoerd, zolang de waarde van i de 100 niet overschrijdt, beginnende met aan i de waarde 1 te assigneren en vervolgens elke keer, nadat de statement is uitgevoerd, de waarde van i met 1 te verhogen. Daar in de compound statement de waarde van i niet wordt aangetast (vaak zal dat juist wel het geval zijn) zal de compound statement 100 keer uitgevoerd worden.

De tekst for i:= 1 step 1 until 100 do begin a:= read; print(a) end vormt weer één statement - een zogenaamde for statement.

Na een for clause mag een willekeurige statement staan. Dit statement kan zeer gecompliceerd zijn, zoals in onderstaand programma het geval is.

```
begin comment 1739, N. van de Auteur (MC), dit programma drukt de kwadra-
      ten van 1 tot 1000 af in een tabel van 20 pagina's met elk 10 blok-
      ken van 5 kwadraten;
      integer pagina, blok, regel, argument;
      argument := 1;
      for pagina := 1 step 1 until 20 do
      begin if pagina ≠ 1 then NEWPAGE;
            for blok := 1 step 1 until 10 do
            begin if blok ≠ 1 then begin NLCR; NLCR end;
```

```
                    for regel := 1 step 1 until 5 do
                    begin if regel ≠ 1 then NLCR;
                          print(argument);
                          print(argument ↑ 2);
                          argument := argument + 1
                    end
              end
        end
end
```

Voor een gedetailleerde beschrijving van de betekenis van NLCR en NEWPAGE, die overgang op een nieuwe regel resp. nieuwe bladzijde regelen, zij verwezen naar MR 81. Bij bestudering van het programma zal het duidelijk zijn dat de statements

"print(argument); print(argument ↑ 2); argument := argument + 1"

1000 maal worden uitgevoerd. Hierdoor worden de volgende twee acties bewerkstelligd:

1. het argument gevolgd door zijn kwadraat wordt afgedrukt
2. het argument wordt met 1 verhoogd.

Daar het argument op 1 geïnitialiseerd wordt, worden de eerste 1000 getallen en hun kwadraten afgedrukt.
Dat deze tabel ook een nette lay-out heeft, wordt verzorgd door de if statements als

```
if pagina ≠ 1 then NEWPAGE
```

Bij de executie heeft dit statement tot gevolg dat de procedure NEWPAGE wordt uitgevoerd als pagina ≠ 1 is, terwijl als pagina de waarde 1 draagt er niets gebeurt.
Hoe moeten we ons de werking van de assignment statement

```
argument := argument + 1
```

voorstellen?

De expressie rechts van het wordt-teken is nu iets ingewikkelder geworden. Er spelen zich een aantal processen in onderstaande volgorde af:

1. Links van het wordt-teken wordt uitgemaakt wie de drager is.
2. Rechts van het wordt-teken wordt gevraagd naar de waarde van de expressie. De expressie bestaat uit een + teken met 2 operanden. De waarde van de expressie is de som van de operand links en de operand rechts. De linker operand is een drager; staat een drager als operand dan is de waarde van de operand de waarde die de drager op dat moment draagt. Rechts van het plusteken staat een getal; staat een getal als operand dan is de waarde van de operand juist de waarde, die dat getal in de "gewone wiskunde" representeert.
3. De waarde van de expressie afgeleverd als antwoord op de vraag in punt 2 wordt geassigneerd (toegekend) aan de drager, die daardoor die waarde gaat dragen.

Resultaat:
Door de assignment statement "argument := argument + 1" wordt er bij de waarde, die door de drager "argument" gedragen wordt, 1 opgeteld.

5. <u>conditional statement</u>

Naast een taalstructuur als de for statement, die ervoor zorgt dat een statement meerdere malen wordt uitgevoerd, beschikt ALGOL 60 over een taalstructuur, die - afhankelijk van een conditie - uit twee statements één selecteert en uitvoert.

We bekijken een programma dat de 10 getallen a_1, a_2, ..., a_{10} van een getallenband leest en de getallen en hun differenties volgens onderstaand patroon afdrukt:

a_1
$\quad a_2 - a_1$
a_2
.
.
.
a_9
$\quad a_{10} - a_9$
a_{10}

```
 1. begin comment 1739, N. van de Auteur (MC), differentietabel;
 2.        integer i;
 3.        real a, vorige;
 4.
 5.        for i:= 1 step 1 until 10 do
 6.        begin a:= read;
 7.              if i = 1 then begin print(a); vorige := a end
 8.                       else begin NLCR; SPACE(10); print(a - vorige);
 9.                                  NLCR; print(a); vorige := a
10.                            end
11.        end
12. end
13.
```

Voor de betekenis van de procedure statement SPACE(10), die de positie op de regel met 10 verhoogt, zij verwezen naar MR 81.

We herkennen de structuur op regel 5 t/m 11 als een for statement.
Ten gevolge van de for clause op regel 5 wordt tijdens executie de compound statement op regel 6 t/m 11 tien maal uitgevoerd.
De uitvoering van de compound statement houdt in:

1. Er wordt een getal gelezen en afgeleverd aan de drager "a".

2. De conditional statement op regel 7 t/m 10 wordt uitgevoerd.
   Dit houdt, wanneer i de waarde 1 draagt (i = 1), in dat de (compound) statement tussen then en else wordt uitgevoerd en anders dat de (compound) statement na else wordt uitgevoerd. In het ene geval wordt alleen het onder punt 1 ingelezen getal afgedrukt en afgeleverd aan de drager "vorige", in het andere geval wordt de differentie van het ingelezen getal met het vorige afgedrukt en vervolgens het ingelezen getal zelf. Het patroon wordt verzorgd door de procedure statements "NLCR" en "SPACE(10)".

Opmerking: Dit probleem kan ook opgelost worden met een if statement.

II. Declaraties

1. quantity

ALGOL 60 kent tweeërlei symbolen:

1. symbolen, die voor zichzelf staan, zoals +, - en end en
2. symbolen of rijtjes symbolen - namen -, die iets anders aanduiden, naar een ander object - een quantity - refereren;

Welke quantity en van wat voor soort die quantity is, blijkt pas uit de context: de structuur van het hele programma.
Iets analoogs vinden we in de "gewone wiskunde" bij

$$ax^2 + bx + c = 0,$$

waar uit de context moet blijken of a, b, c staan voor zekere getallen en x voor een te bepalen getal of dat de wortels x_1 en x_2 gegeven zijn en gevraagd wordt a, b en c te bepalen; onafhankelijk hiervan hebben +, = en 0 hun zelfde vaste betekenis.

Namen in ALGOL 60 kunnen nu de volgende soorten objecten - quantities - aanduiden:

1. een drager - een integer, real of Boolean - die een waarde - value - kan dragen.
2. een aanwijzer - label - die een statement in het programma aanwijst.
3. een legertje dragers - array.
4. een legertje aanwijzers - switch.
5. een proces - procedure.

Namen dienen ervoor de quantities, die in het rekenproces een rol spelen, van elkaar te onderscheiden.
De relatie tussen een quantity en zijn naam wordt voor simple variables,*) arrays, switches en procedures gevestigd door een declaratie in een block head **).

---

*) integers, reals of Booleans

**) Net zoals geneste for statements kent ALGOL 60 ook geneste blocks. De voorzichtige formulering maakt dat we er ook na behandeling van gecompliceerde taalregels nog naar kunnen verwijzen.

Een declaratie zoals real a geeft bij executie aanleiding tot de volgende acties:

1. Er wordt een nieuwe quantity - van de soort aangegeven door de declarator (real) - gecreëerd, die weer ophoudt te bestaan als het block in welks head de declaratie staat verlaten wordt.
2. Er wordt een naam (a) aan gehecht.
3. Bestaat de declaratie uit een declarator gevolgd door een rijtje namen (bv. integer pagina, blok, regel, argument zoals in het programma op blz. 6) dan worden 1. en 2. voor elke naam afzonderlijk gedaan.

De relatie tussen een label en zijn naam bestaat krachtens het voorkomen van zijn naam, gevolgd door een dubbele punt, vóór een statement.
Een dergelijk voorkomen van een naam geeft aan:

1. In het kleinste block, dat deze statement omvat, bestaat een quantity van de soort label (daarbuiten uiteraard niet).
2. Binnen het onder 1 vermelde block geeft bovengenoemde naam de onder 1 vermelde label aan.

Quantities, die binnen een block bestaan en erbuiten niet, heten locaal ten opzichte van dat block. ALGOL 60 eist dat de namen gehecht aan de localen van één block alle verschillend zijn.

2. label

Labels scheppen de mogelijkheid statements, die in de tekst van het programma niet op elkaar volgen, tijdens executie wel achter elkaar te laten uitvoeren. Men zij gewaarschuwd, dat een overdadig gebruik van labels de overzichtelijkheid niet ten goede komt.

We bekijken het volgende programma:

```
begin comment 1739, N. van de Auteur (MC), druk getallen af;
      real a;
opnieuw: a:= read;
         print(a);
         goto opnieuw
end
```

Dit programma is een block. Tijdens de executie begint de rekenautomaat na kennisname van de declaratie real a met de uitvoering van:

opnieuw: a:= read.

"opnieuw" is hier (een naam die gehecht is aan) een label en is voor de werking van deze statement niet van belang. Daarna wordt de waarde, die door de drager "a" gedragen wordt, afgedrukt in de procedure statement "print(a)". Vervolgens wordt de goto-statement

goto opnieuw

uitgevoerd.
De uitvoering van een goto-statement omvat de volgende acties:

1. De waarde van de designational expressie na goto wordt bepaald.
   (De waarde van een designational expressie is een label, hier "opnieuw".)
2. De executie wordt voortgezet bij de statement, die door die waarde aangegeven wordt.

In dit programma wordt steeds weer een getal gelezen en afgedrukt. In de praktijk zal dit altijd spaak lopen, omdat de getallenband niet oneindig lang is. Het MC-ALGOL 60-systeem voor de X8 geeft dan ook vroeg of laat foutmelding 998 met de betekenis: het programma is afgebroken wegens het gebrek aan verdere Input.

3. array

Een probleem, dat als vanzelf tot het gebruik van een array noodt, is het opmaken van scores behaald bij een multiple choice test.
Het programma op blz. 17 beschrijft een algorithme voor het nakijken van een test bestaande uit 60 vragen met elk 4 alternatieven, waarvan slechts één als goed gehonoreerd wordt. Naast de score van elke student levert uitvoering van het programma ook informatie over de vragen zelf. Er wordt nl. voor elk alternatief bepaald hoe groot het percentage studenten is, die het gekozen hebben.

De getallenband bevat achtereenvolgens:

1. de sleutel d.w.z. 60 getallen tussen 1 en 4.

2. voor iedere student: zijn volgnummer gevolgd door zijn antwoorden (bij geen antwoord een 0).

3. een 0 ter afsluiting.

Tijdens de executie zal de kennis van de steutel steeds weer vereist zijn. Er moeten dus 60 dragers geïntroduceerd worden om de sleutel te onthouden.
In de voorafgaande programma's had elke drager zijn eigen naam. Deze naam werd expliciet in het programma genoemd. Een array declaratie zoals <u>integer</u> <u>array</u> sleutel[1 : 60] op regel 3 van het hier behandelde programma, introduceert meerdere dragers (hier 60) onder één naam; deze dragers kunnen in het programma worden aangeduid door:

sleutel[1], sleutel[2], ..., sleutel[60]

d.w.z. door middel van een naam en een gehele waarde tussen 1 en 60, of in het algemeen door:

sleutel[subscript expression]

waar de subscript expression een arithmetische expressie is, die een gehele waarde eventueel na afronding aflevert tussen de lower bound (hier 1) en de upper bound (hier 60).

In de assignment statement

sleutel[v]:= read

spelen weer 3 processen, die in onderstaande volgorde worden uitgevoerd:

1. het klaarzetten van een drager; welke dat is hangt af van de waarde die de subscript expression aflevert;
2. het bepalen van de waarde van de expressie rechts;
3. het toekennen van de onder 2 bepaalde waarde aan de onder 1 klaargezette drager.

<u>Opmerking</u>. Dat het precies vastleggen van de volgorde zin heeft moge blijken uit een assignment statement als sleutel[read]:= read.

De declaratie

<u>integer</u> <u>array</u> frequentie[1: 60, 0 : 4]

declareert een twee-dimensionaal array - in ALGOL 60 mag een array net zoveel dimensies hebben als het probleem eist. Hier worden door middel

van één naam 60 × 5 = 300 dragers geïntroduceerd. In de statements van het programma wordt een individuele drager aangegeven door:

frequentie[v,a]

waarbij de waarden van v en a tussen de corresponderende lower en upper bounds moeten liggen. De functie van dit array in het programma is uiteraard te tellen hoe vaak een bepaald alternatief bij een bepaalde vraag gekozen is.
Als we het programma lezen, zien we dat de namen iets suggereren over de rol van hun variabele in het proces.
Voor de betekenis van NLCR, PRINTTEXT en ABSFIXT zij verwezen naar MR 81. Kort gezegd zorgt NLCR voor overgang op een nieuwe regel, PRINTTEXT voor afdrukken van het stukje tekst tussen "⊦" en "⊣" en ABSFIXT voor het afdrukken van de absolute waarde van de 3e expressie tussen de haakjes volgens een lay-out bepaalde door de eerste 2 expressies aldaar.

Het is aanbevelenswaard om MR 81 op deze procedures na te lezen, enerzijds om de te verkrijgen informatie zelf, anderzijds om nog eens goed te beseffen dat, als men niet voor het MC-ALGOL 60-systeem voor de X8 programmeert, men een dergelijke handleiding moet hebben die informatie over het systeem, waar men voor programmeert, verschaft.

Op regel 13 t/m 28 zien we een voorbeeld van een for statement nu eens niet met een <u>step</u> <u>until</u> element maar met een <u>while</u> element.
Tijdens de executie omvat de uitvoering van deze for statement de volgende acties:

1. De assignment achter <u>for</u> wordt uitgevoerd.
2. De waarde van de Boolean expressie achter <u>while</u> wordt bepaald.
3. Is deze waarde <u>false</u> dan wordt de uitvoering van de for statement beëindigd, anders wordt de statement achter <u>do</u> uitgevoerd en daarna verder gegaan bij 1.

Het programma heeft ten aanzien van de executie de volgende delen:
1. Een gedeelte dat de sleutel inleest en de telvariabelen op nul zet (regel 6 t/m 11).
2. Een gedeelte dat de score van iedere student telt en afdrukt (regel

13 t/m 28), bijhoudt welk alternatief hij kiest (regel 18) en "aantal studenten" steeds met één verhoogt (regel 14).

3. Een gedeelte dat voor elke vraag afdrukt:
   a. het nummer van de vraag (regel 32)
   b. het juiste alternatief (regel 32)
   c. voor elk alternatief bij de vraag: hoe groot het percentage studenten, die het gekozen hebben, is (regel 34).
4. switch

Door het array begrip kan men met behulp van slechts één naam met meerdere dragers, die in het programma eenzelfde soort rol vervullen, handig manipuleren.
Iets dergelijks kan men ook met labels doen met behulp van een switch.

Men zal in het algemeen dan een switch in zijn programma inlassen als het probleem een strooisprong over een veld van mogelijkheden eist.
Stel dat de opgave bij het nakijken van de multiple choice test niet is het geven van de score, maar het geven van een beoordeling in de vorm van uitmuntend, goed, voldoende, etc., dan kan dit opgelost worden door het programma (zie blz.18) als volgt te wijzigen:

a. Inlassing op regel 5 van de declaratie:

```
switch beoordeling:= 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
```

b. Vervanging op regel 21 van ABSFIXT(6,0,score) door:

```
goto beoordeling[(score÷6)+(if score-score÷6×6>3 then 2 else 1) ];
```

```
1:PRINTTEXT(⊰  zeer slecht ⊱);goto e;2:PRINTTEXT(⊰            slecht ⊱);goto e;
3:PRINTTEXT(⊰        matig ⊱);goto e;4:PRINTTEXT(⊰beslist onvoldoende⊱);goto e;
5:PRINTTEXT(⊰  onvoldoende ⊱);goto e;6:PRINTTEXT(⊰          voldoende⊱);goto e;
7:PRINTTEXT(⊰ruim voldoende⊱);goto e; 8:PRINTTEXT(⊰            goed ⊱);goto e;
9:PRINTTEXT(⊰     zeer goed ⊱);goto e;10:PRINTTEXT(⊰        uitmuntend ⊱);goto e;
0:PRINTTEXT(⊰          nul ⊱);e:
```

Zoals we onder a. zien bestaat een switchdeclaratie uit:
een switchdeclarator <u>switch</u> gevolgd door zijn naam (hier beoordeling) gevolgd door een wordt-teken, gevolgd door een rijtje designational expressies onderling gescheiden door een komma. (Als de termen dimensie, lower en upper bound bij een switch gebruikt zouden kunnen worden, dan konden we zeggen dimensie en lower bound altijd 1, upper bound is aantal designational expressies.)

We kunnen nu spreken, tellend van links naar rechts, van de eerste, tweede, etc. designational expression. (Hier is de eerste expressie de integer label 0, de tweede de integer label 1 etc.).

Bij uitvoering van de goto-statement

<u>goto</u> beoordeling[(score÷6)+(<u>if</u> score-score÷6×6>3 <u>then</u> 2 <u>else</u> 1)]

gebeurt het volgende:

1. De waarde van de indexexpressie tussen [en] wordt uitgerekend.
2. Zij p de waarde gevonden in 1. dan wordt de waarde bepaald van de p^e designational expressie in de switchdeclaratie; deze laatste waarde is een label.
3. De executie wordt voortgezet bij de statement die gelabeld wordt door de in 2 bepaalde label.

De operatie aangegeven door het teken ÷ heet helen. Deze operatie is ingevoerd omdat bij deling van de ene integer door de andere niet altijd een resultaat van het type integer behoeft te ontstaan. Bij heling - de integerdeling - is dat wel zo. Bovendien is het resultaat gelijk aan het resultaat bij gewone deling wanneer de laatste een integer oplevert. De heling $p \div q$ levert als resultaat op het getal a zodanig dat $p = qa + b$ met a, b geheel $|b| < q$ en het teken van b gelijk aan het teken van p.

```
1  begin   comment   1739, N. van de Auteur(MC), nakijken multiple choice test
2              van 60 vragen met elk 4 alternatieven;
3              integer array sleutel[1:60], frequentie[1:60, 0:4];
               integer v,a, score, antwoord, volgnummer, aantal studenten;

6              aantal studenten := 0;
7              for v:= 1 step 1 until 60 do sleutel[v] := read;

9              for v:= 1 step 1 until 60 do
               for a:= 0 step 1 until  4 do
11             frequentie[v,a]:= 0;

13             for volgnummer:= read while volgnummer ≠ 0 do
14             begin   NLCR; score:= 0; aantal studenten:= aantal studenten + 1;
15                     for v:= 1 step 1 until 60 do
16                     begin   antwoord:= read;
17                             if antwoord = sleutel[v] then score:= score + 1 ;
18                             frequentie[v,antwoord]:= frequentie[v,antwoord] + 1
19                     end;
20                     ABSFIXT(6,0,volgnummer);
21                     ABSFIXT(6,0,score);
22
23
24
25
26
27
28             end;
29             NLCR; PRINTTEXT(⊂ informatie over de vragen zelf⊃);
30             for v:= 1 step 1 until 60 do
31             begin   NLCR;
32                     ABSFIXT(3,0,v); ABSFIXT(6,0,sleutel[v]);
33                     for a:= 0 step 1 until 4 do
34                     ABSFIXT(4,2,frequentie[v,a]×100/aantal studenten)
35             end
36 end
37
```

```
begin
comment   1739, N. van de Auteur(MC), nakijken multiple choice test
van 60 vragen met elk 4 alternatieven;
integer array sleutel[1:60], frequentie[1:60, 0:4];
integer v,a, score, antwoord, volgnummer, aantal studenten;
switch beoordeling := 0,1,2,3,4,5,6,7,8,9,10;
aantal studenten := 0;
for v:= 1 step 1 until 60 do sleutel[v] := read;

for v:= 1 step 1 until 60 do
for a:= 0 step 1 until  4 do
frequentie[v,a]:= 0;

for volgnummer:= read while volgnummer ≠ 0 do
begin   NLCR; score:= 0; aantal studenten:= aantal studenten + 1;
        for v:= 1 step 1 until 60 do
        begin   antwoord:= read;
                if antwoord = sleutel[v] then score:= score + 1 ;
                frequentie[v,antwoord]:= frequentie[v,antwoord] + 1
        end;
        ABSFIXT(6,0,volgnummer);
        goto beoordeling[(score:6) +( if score - ((score:6)×6) > 3 then 2 else1)];
1: PRINTTEXT(⊂   zeer slecht⊃); goto e; 2: PRINTTEXT(⊂              slecht⊃); goto e;
3: PRINTTEXT(⊂          matig⊃); goto e; 4: PRINTTEXT(⊂beslist onvoldoende⊃); goto e;
5: PRINTTEXT(⊂   onvoldoende⊃); goto e; 6: PRINTTEXT(⊂           voldoende⊃); goto e;
7: PRINTTEXT(⊂ruim voldoende⊃); goto e; 8: PRINTTEXT(⊂                goed⊃); goto e;
9: PRINTTEXT(⊂      zeer goed⊃); goto e;10: PRINTTEXT(⊂          uitmuntend⊃); goto e;
0: PRINTTEXT(⊂            nul⊃); e:
end;
NLCR; PRINTTEXT(⊂ informatie over de vragen zelf⊃);
for v:= 1 step 1 until 60 do
begin   NLCR;
        ABSFIXT(3,0,v); ABSFIXT(6,0,sleutel[v]);
        for a:= 0 step 1 until 4 do
        ABSFIXT(4,2,frequentie[v,a]×100/aantal studenten)
end
end
```

5. block, scope, procedure

De in het voorafgaande behandelde programma's waren alle een block en dat block was ook het enige block van het programma. Dit geldt niet voor elk ALGOL-60 programma.
Een ALGOL 60 statement kan nl. zelf weer een block zijn zodat programma's een geneste blockstructuur kunnen hebben analoog aan geneste for statements.

Moet men bv. een programma schrijven voor het nakijken van een multiple choice test met een willekeurig aantal vragen, dan kan men op voordelige wijze gebruik maken van de blockstructuur.
We behandelen een eenvoudiger probleem, dat even instructief is:
Gegeven: een band met daarop 2 getallenrijen, voorafgegaan door een getal dat aangeeft hoe lang deze rijen zijn.
Gevraagd: schrijf een programma, dat de overeenkomstige elementen optelt en de sommen afdrukt.
Oplossing:

```
 1 begin comment 1739, N. van de Auteur (MC), sommeren getallenrijen;

 2       integer n;
 3       n:= read;
 4       begin real array a[1 : n];
 5             integer i;
 6             for i:= 1 step 1 until n do a[i]:= read;
 7             for i:= 1 step 1 until n do print(a[i] + read)
 8       end
 9 end
10
```

Als gevolg van de declaratie real array a[1 : n] wordt tijdens de executie een aantal dragers gecreëerd. Hoe groot dit aantal is hangt af van de waarde, die de arithmetische expressie, die de upper bound vormt, aflevert; deze waarde is de waarde die de drager n draagt en dat is (regel 3) de waarde van het eerste getal op de band.

Voor een array declaratie geldt:

1. De expressies in de bounds worden één keer bij het binnenkomen van het block uitgerekend; deze grenzen liggen dan vast tot het verlaten van het block, waarop alle gecreëerde dragers ophouden te bestaan.
2. Is een bovengrens kleiner dan zijn corresponderende ondergrens dan worden er geen dragers gecreëerd *).
3. Namen, die in de expressies van de bounds voorkomen, moeten in een omvattend block gedeclareerd zijn.

De blockstructuur biedt de mogelijkheid om op verschillende niveau's quantities te introduceren.
Zoals reeds in paragraaf 1 vermeld is,eist ALGOL 60 dat de namen gehecht aan de localen van eenzelfde block verschillend zijn.
De programmeur heeft echter de vrijheid dezelfde naam in meer dan één block te declareren.
Tijdens de executie heeft een declaratie van - een al in een omvattend block gebruikte - naam het volgende effect:

1. De naam wordt losgemaakt van de oude quantity en gehecht aan de nieuw in te voeren quantity.
2. De quantity, waaraan deze naam buiten dit block gehecht was, kan binnen dit block niet meer worden aangeduid. (Noch de waarde kan worden gevraagd noch kan er aan geassigneerd worden.)
3. Bij het verlaten van het block wordt de naam weer aan de oude quantity gehecht en is deze weer bereikbaar.

---

*) Bij het MC-ALGOL 60-systeem voor de X8 wordt de executie beëindigd onder opgave van foutnummer 501 (zie MR 81).

De output van het programma op blz. 17 begint voor elke student op een nieuwe regel. Doen er veel studenten mee, dan zal de output meerdere pagina's beslaan. Weliswaar zorgt een implementatie, als het MC-ALGOL 60-systeem voor de X8, voor het voordraaien van een nieuwe pagina, maar het zou kunnen zijn dat men boven elke pagina een kopje: "Uitslag tentamen 16 juli 1968" wenst. Een programma dat dit doet staat op blz. 23; het is letterlijk het oude programma met regel 1 t/m 11 ervoor en een end erachter. Daar het oude programma een block was en een statement in een programma zelf weer een block mag zijn, is dit zonder meer geoorloofd. We merken op dat de betekenis van regel 12 t/m 43 in het nieuwe programma niet meer dezelfde is als van de tekst in het oude programma.
Dit komt door de declaratie:

```
procedure NLCR;
     begin a:= a + 1;
           if a = 51 then begin a:= 1; NEW PAGE;
                                PRINTTEXT({uitslag tentamen 16 juli 1968})
                           end;
           PRSYM(nlcr)
     end;
```

Deze procedure declaratie bestaat uit een declarator (procedure) gevolgd door een procedure identifier (NLCR) gevolgd door een puntkomma (;), die samen de procedure heading (procedure NLCR;) vormen; de statement hierna is de zogenaamde procedure body (in dit geval een compound statement). Door de procedure declaratie van "NLCR" wordt overal in het block, dat begint op regel 1 (i.e. het hele programma), de betekenis van de procedure statement "NLCR": het uitvoeren van de procedure body op regel 5 t/m 10.

Door deze procedure declaratie van "NLCR" is de aan het systeem bekende procedure "NLCR" onbereikbaar geworden. Het effect van de aan het systeem bekende procedure "NLCR" kan echter nog wel door PRSYM(nlcr) bereikt worden, mits "nlcr" een drager is, die de waarde 119 draagt (zie regel 11). Krijgt PRSYM nl. als waarde 119 aangeboden, dan is het effect overgang op een nieuwe regel (zie MR 81).

Door inbedding van het oude programma in het nieuwe veranderde de betekenis van "NLCR". De naam "a", die op regel 3 gedeclareerd wordt, komt ook in het oude programma voor. Dit is precies het geval behandeld op blz. 20. De quantity door de declaratie integer a op regel 3 gecreëerd is derhalve binnen het block van regel 12 t/m 43 onbereikbaar. Dat deel van het programma, waar de naam van een quantity ook inderdaad aan die quantity gehecht is (waar de quantity bereikbaar is), heet de "scope van die quantity".
Zo is de scope van de quantity "a" gecreëerd op regel 3: regel 1 t/m 11 en niet verder.
Executie van het programma houdt in dat na de 4 statements op regel 11 gewoon het oude programma wordt uitgevoerd met dien verstande dat bij aanroep van "NLCR":

1. de waarde van de drager "a", geintroduceerd op regel 3 met één verhoogd wordt.
2. als "a" (van regel 3) hierna de waarde 51 draagt, overgegaan wordt op een nieuwe pagina, de waarde 1 aan "a" (van regel 3) geassigneerd wordt en vervolgens het kopje afgedrukt wordt.
3. een regelopvoer gegeven wordt. (Het kopje staat dus apart op de eerste regel van elke bladzijde.)

```
begin   comment R 1739, N. van de Auteur(MC), nakijken multiple choice test
            met elke pagina output voorzien van een kopje;
            integer a, nlcr;
            procedure NLCR;
            begin   a:= a + 1;
                    if a = 51 then begin      a:= 1; NEWPAGE;
                                              PRINTTEXT(⊂uitslag tentamen 16 juli 1968⊃)
                                         end;
                    PRSYM(nlcr)
            end;
            nlcr:= 119; a:= 1; PRINTTEXT(⊂uitslag tentamen 16 juli 1968⊃); NLCR;
begin   comment R 1739, N. van de Auteur(MC), nakijken multiple choice test
            van 60 vragen met elk 4 alternatieven;
            integer array sleutel[1:60], frequentie[1:60, 0:4];
            integer v,a, score, antwoord, volgnummer, aantal studenten;

            aantal studenten := 0;
            for v:= 1 step 1 until 60 do sleutel[v] := read;

            for v:= 1 step 1 until 60 do
            for a:= 0 step 1 until  4 do
            frequentie[v,a]:= 0;

            for volgnummer:= read while volgnummer ≠ 0 do
            begin   NLCR; score:= 0; aantal studenten:= aantal studenten + 1;
                    for v:= 1 step 1 until 60 do
                    begin   antwoord:= read;
                            if antwoord = sleutel[v] then score:= score + 1 ;
                            frequentie[v,antwoord]:= frequentie[v,antwoord] + 1
                    end;
                    ABSFIXT(6,0,volgnummer);
                    ABSFIXT(6,0,score);


            end;
            NLCR; PRINTTEXT(⊂ informatie over de vragen zelf⊃);
            for v:= 1 step 1 until 60 do
            begin   NLCR;
                    ABSFIXT(3,0,v); ABSFIXT(6,0,sleutel[v]);
                    for a:= 0 step 1 until 4 do
                    ABSFIXT(4,2,frequentie[v,a]×100/aantal studenten)
            end
end
end
```

III. Syntax en Semantiek

1. algemeen

Een programma roept de volgende vragen op:

1. Is het inderdaad een ALGOL 60 programma?
2. Zo ja, wat doet het, wat betekent het?

De eerste vraag kan beantwoord worden met behulp van de Syntax, de tweede met behulp van de Semantiek.
De Syntax bestaat uit een geformaliseerd gedeelte en een niet-geformaliseerd gedeelte. Het geformaliseerde gedeelte is opgebouwd uit een aantal regels, zgn. metalinguïstische formules (zie III-2); deze formules zijn geschreven in de zgn. BACKUS' NORMAL FORM. Het niet-geformaliseerde gedeelte bestaat uit een aantal opmerkingen her en der verspreid in het Revised Report, die iets over de vorm van een programma zeggen (zie III-3).
De Semantiek bestaat uit zinnen - soms gegroepeerd onder "Semantics" - in het Revised Report, die de betekenis van een ALGOL 60 programma vastleggen. De grens tussen Syntax en Semantiek is daardoor moeilijk aan te geven.
Toch zijn het werkzame begrippen, temeer als men zich realiseert hoe een programma met behulp van een rekenautomaat verwerkt wordt. Daar de Syntax de vorm van het programma vastlegt, wordt tijdens de analysefase eigenlijk gecheckt of het programma aan de Syntax voldoet. Men spreekt daarom van een syntactische analyse en van een syntactische foutmelding.
Tijdens de analysefase werkt de rekenautomaat onder controle van de "specifieke implementatie" en wordt tevens geprobeerd een vertaling te maken van ALGOL 60 naar instructies, die voor de machine begrijpelijk zijn, in de zgn. machinecode; men noemt het onderdeel van het systeem dat in deze fase werkzaam is dan ook wel: de compiler of vertaler.
Tijdens de executiefase worden indien nodig ook foutmeldingen gegeven bijv. bij uitvoering van a[i]:= 10 als de waarde van i buiten de bounds van het array blijkt te liggen. Deze fout heeft niets te maken met de vorm van het programma. De syntactische foutmeldingen, die het MC-ALGOL 60-systeem kan geven, staan op blz. 17, 18, 19 en 20 van MR 81. Op blz. 20

staan ook de fouten genoemd, die op kunnen treden tijdens de executie-fase. Wordt het programma in executie genomen, dan weet men dus zeker dat het een ALGOL 60 programma is; of het ook dat programma is, dat ons probleem oplost, blijft de vraag.

2. metalinguïstische variabele, metalinguïstische formule, terminale productie, identifier

Het geformaliseerde gedeelte van de Syntax bestaat uit metalinguïstische formules. In een metalinguïstische formule onderscheiden we:

1. woorden omsloten door een haakjespaar < >
   de zgn. metalinguïstische variabelen;
2. metalinguïstische verbindingstekens te weten
   ::= (spreek uit: wordt gedefiniëerd als)
   en | (spreek uit: of);
3. karakters - zgn. basic symbols - uit het ALGOL 60 alfabet.

Voorbeeld:

<identifier>::= <letter>|<identifier><letter>|<identifier><digit>
<letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<digit>::= 0|1|2|3|4|5|6|7|8|9

In deze 3 formules komen voor de metalinguïstische variabelen: <identifier>, <letter> en <digit> en de basic symbols a, b, c, ..., Z en 0, 1, ..., 9.

De bedoeling van deze formules is, dat hiermee de verzameling van identifiers geheel beschreven wordt, doordat iedere identifier voort te brengen is uit <identifier> door bij herhaalde toepassing van één van deze formules steeds een specifieke keuze te doen.

Voordat we dit verder duidelijk kunnen maken, merken we het volgende op:

1. In een metalinguïstische formule komt juist één ::= teken voor, dat de formule splitst in een linker- en rechterlid.
2. In het linkerlid staat slechts één metalinguïstische variabele.

3. Het rechterlid wordt door | tekens - indien aanwezig - gesplitst in een aantal delen, die zelf geen | tekens meer bevatten.

4. Eén zo'n deel, dat zowel metalinguïstische variabelen als basic symbols mag bevatten, heet een directe productie van de metalinguïstische variabele in het linkerlid.

Het voortbrengen van een identifier uit de metalinguïstische variabele <identifier> gaat op de volgende wijze:

a. Vervang de metalinguïstische variabele (hier dus <identifier>) door één van zijn directe producties.

b. Vervang in het resultaat elke metalinguïstische variabele, indien aanwezig, door één van zijn directe producties.

c. Herhaal b tot alle metalinguïstische variabelen verdreven zijn.

Een volgens bovenstaand algorithme uit een metalinguïstische variabele verkregen resultaat heet een terminale productie van de metalinguïstische variabele. Zo kan men X8 uit identifier voortbrengen via de stappen:

<identifier>
<identifier><digit>
<letter> 8
X 8

3. Syntax als middel om de Semantiek te verkorten

Ter verdediging van de gecompliceerde structuur van de Syntax kan men aanvoeren:

1. dat er zeer compact van een groot aantal rijen basic symbols wordt uitgemaakt dat het géén programma's zijn;

2. dat, hoewel er oneindig veel ALGOL 60 programma's zijn door betrekkelijk weinig regels aan alle een betekenis wordt toegekend.

Ter toelichting van punt 2 diene de volgende vergelijking:

Stel, men wil een telefoonboek maken voor een stad met minder dan 999 straten elk met minder dan 999 huisnummers. Structureert men de "telefoonnummers" dan kan het telefoonboek aanzienlijk dunner worden dan anders.
Dit zou bijv. met behulp van de volgende Syntax kunnen:

```
<telefoonnummer>::= <straat><huisnummer>
<huisnummer>    ::= <drie cijfers>
<straat>        ::= <drie cijfers>
<drie cijfers>  ::= <digit><digit><digit>
<digit>         ::= 0|1|2|3|4|5|6|7|8|9
```

In het telefoonboek hoeft men nu voor één abonnée slechts zijn naam en telefoonnummer op te geven. Hoe uit het "telefoonnummer" kan worden afgeleid, wat "straat" en "huisnummer" van de telefoonbezitter zijn moet in de inleiding vermeld worden. Dit kan bijv. als volgt:
De eerste "drie cijfers" van het "telefoonnummer" bepalen een ingang in een tabel waar de naam van de "straat" is af te lezen.
De tweede "drie cijfers" bepalen het "huisnummer".
Doordat de metalinguïstische variabelen een suggestieve naam hebben, is eenvoudig uit te leggen hoe een adres uit een telefoonnummer bepaald wordt. We zien dus dat het telefoonboek - de Semantiek - aanzienlijk bekort wordt door invoering van een structurering - een gecompliceerde Syntax.

4. naam, basic symbol, ALGOL 60 programma

De metalinguïstische variabelen zijn zo gekozen dat het woord binnen de haakjes iets suggereert van de semantische of syntactische eigenschappen van de terminale producties van de metalinguïstische variabelen.
Willen we over een terminale productie van <identifier> spreken, dan mogen we die aanduiden met: een identifier. Zo kunnen we zeggen:
"Een identifier in ALGOL 60 is een letter gevolgd door een willekeurig lange rij letters of digits."

Voor de beschrijving van identifiers maken we alleen gebruik van de geformaliseerde Syntax. Anders wordt het b.v. bij de metalinguïstische variabele <block>. We zullen uiteraard onder "een block" alleen een terminale productie verstaan, die ook aan de niet-geformaliseerde Syntax voldoet, m.a.w.
"begin integer a; real a; end" is geen block, maar
"begin integer a; real b; end" wel.

In het voorafgaande hebben we het woord "naam" zonder meer gebruikt.
We preciseren nu:

1. Namen van simple variables, arrays, switches en procedures moeten identifiers zijn.
2. Namen van labels moeten identifiers of unsigned integers zijn.

Hoofdstuk 2 van het Revised Report geeft de metalinguïstische formule voor de metalinguïstische variabele <basic symbol>.
Hierdoor is vastgelegd wat basic symbols zijn.

In het algemeen *) kan men basic symbols ook als volgt herkennen:

Staat iets in een metalinguïstische formule uit het Revised Report en
1. is het geen metalinguïstische variabele
en 2. is het geen ::= teken of | teken
dan is het een basic symbol.

Een rij basic symbols is een ALGOL 60 programma als zij:

1. een terminale productie is van <program>;
en 2. voldoet aan de 3 zogenaamde context-condities:
   a. Iedere identifier moet worden gedeclareerd;
   b. De localen van één block moeten alle een verschillende naam hebben;
   c. Staat een identifier ergens niet in een declaratie, dan moet de functie van de quantity die hij representeert, overeenstem-

---

*) met uitzondering van de combinatie ":(" in de regel:
<parameter delimiter>::=, |)<letter string>:(

men met de functie vermeld in de declaratie, waar die quantity geïntroduceerd wordt.

en 3. niet in strijd is met verbodsregels zoals:
een label in een goto-statement mag niet leiden binnen een block, waar de goto-statement buiten staat.

Opmerking ad 2c: <u>begin</u> <u>Boolean</u> b; b:= 3 <u>end</u> is geen programma.

In het Nederlands zou men terminale producties van <program>, die ook aan de niet-geformaliseerde Syntax voldoen - evenals bij <identifier> - aan moeten geven met programs. Uiteraard spreekt men van programma's in plaats van programs. Gelukkig geldt de algemene definitie van "programma" in I-1 als beschrijving van een algorithme ook voor een ALGOL 60 programma.
Welk probleem de algorithme oplost is helaas niet altijd duidelijk. Dit blijkt in de praktijk dikwijls wanneer men een programma van een ander zonder begeleidend commentaar moet lezen.

<u>Opgaven</u>

In de volgende opgaven zijn de volgende productieregels gegeven:

<program>::= <block>|<compound statement>
<block>::= <unlabelled block>|<label>:<block>
<compound statement>::= <unlabelled compound>|<label>:<compound statement>
<unlabelled block>::= <block head>;<compound tail>
<unlabelled compound>::= <u>begin</u> <compound tail>
<block head>::= <u>begin</u> <declaration>|<block head>;<declaration>
<compound tail>::= <statement> <u>end</u> |<statement>;<compound tail>

en is gegeven dat

1. <u>real</u> a en <u>integer</u> i terminale producties zijn van <declaration>
2. a:= read en en print(a) terminale producties zijn van <statement>
   (tussen de woordjes "en" en "en" uit bovenstaande zin staat de zgn. empty string een rij van 0 symbolen)

Verifiëer nu dat de volgende rijtjes symbolen te genereren zijn uit <program>:

1. begin end
2. begin real a; end
3. begin real a; a:= read; print(a) end
4. begin real a; ; end
5. begin real a; print(a); a:= read end

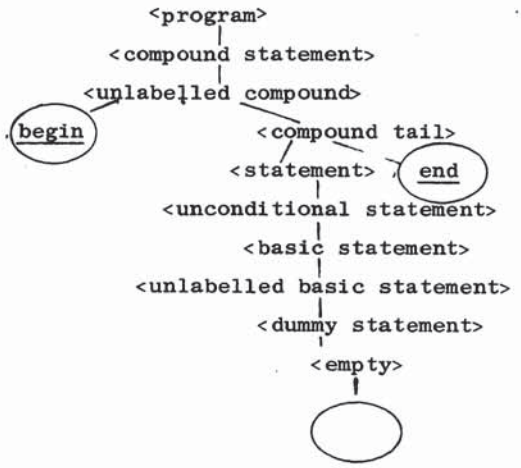
Opmerking:
Bij een implementatie als voor de X8 kan men het verifiëren of het programma syntactisch correct is aan de machine overlaten. Zijn er syntactische fouten dan geeft de machine foutmeldingen, die meestal voldoende informatie verschaffen om de fouten te verbeteren. In ieder geval weet men wanneer het programma in executie genomen wordt vrijwel zeker dat het syntactisch correct is.

5. spreken over ALGOL 60

Hoe kan men in het gewoon Nederlands spreken over ALGOL 60?
Zodra men over een bepaald ALGOL 60 programma spreekt, heeft men het over een rij basic symbols bv.:

begin end

Deze rij is met behulp van de metalinguïstische formules gegeven in het Revised Report uit de metalinguïstische variabele <program> te genereren op de volgende wijze:



Gezien de afspraken in III-4 is het duidelijk wat we bedoelen met: "Het programma begin end heeft geen declaraties en bestaat alleen uit een dummy statement omsloten door de 2 basic symbols begin en end." De opmerking: "begin bestaat uit de letter b gevolgd door de letters e, g, i, n onderlijnd door een zwarte streep" is in gewoon Nederlands juist. Spreekt men echter over ALGOL 60 en bedoelt men met "begin" en "letter" de abstracties die overeenkomen met het basic symbol begin resp. de metalinguïstische variabele <letter> en met "bestaat uit" een syntactisch verband, dan is deze opmerking niet juist (begin is nl. een basic symbol dus net zo'n onverbreekbare eenheid als "," of ";" of ":=").

We concluderen dat als men over ALGOL 60 spreekt er een onderling begrip tussen de sprekers moet zijn of hun woorden hun betekenis ontlenen aan ALGOL 60 of niet.

Behalve dat de woorden binnen de haakjes van een metalinguïstische variabele gebruikt worden om de terminale producties zelf als onderdeel van de tekst van het programma aan te duiden, worden ze ook gebruikt om die dingen, die door de terminale producties worden aangeduid, aan te duiden.
Zeggen we bijv.:
"In het programma begin integer a; a:= 3 end staat een identifier a", dan bedoelen we de letter a.
Zeggen we:
"In het programma begin integer a; a:= 3 end wordt aan de simple variable a de waarde 3 geassigneerd, dan bedoelen we uiteraard niet dat aan de letter a de waarde 3 geassigneerd wordt, maar bedoelen we dat aan de drager (quantity), geïntroduceerd door de declaratie integer a, de waarde 3 geassigneerd wordt.
Iets dergelijks heeft men in het Nederlands ook bij zinnen als: "Parijs is een mooie stad" en "Parijs heeft 5 letters".

Dat de keus van de woorden in de metalinguïstische variabelen mede door een dergelijk doel bepaald is, moge blijken uit de bespreking van het volgende programma:

begin real a; a:= read; print(a) end

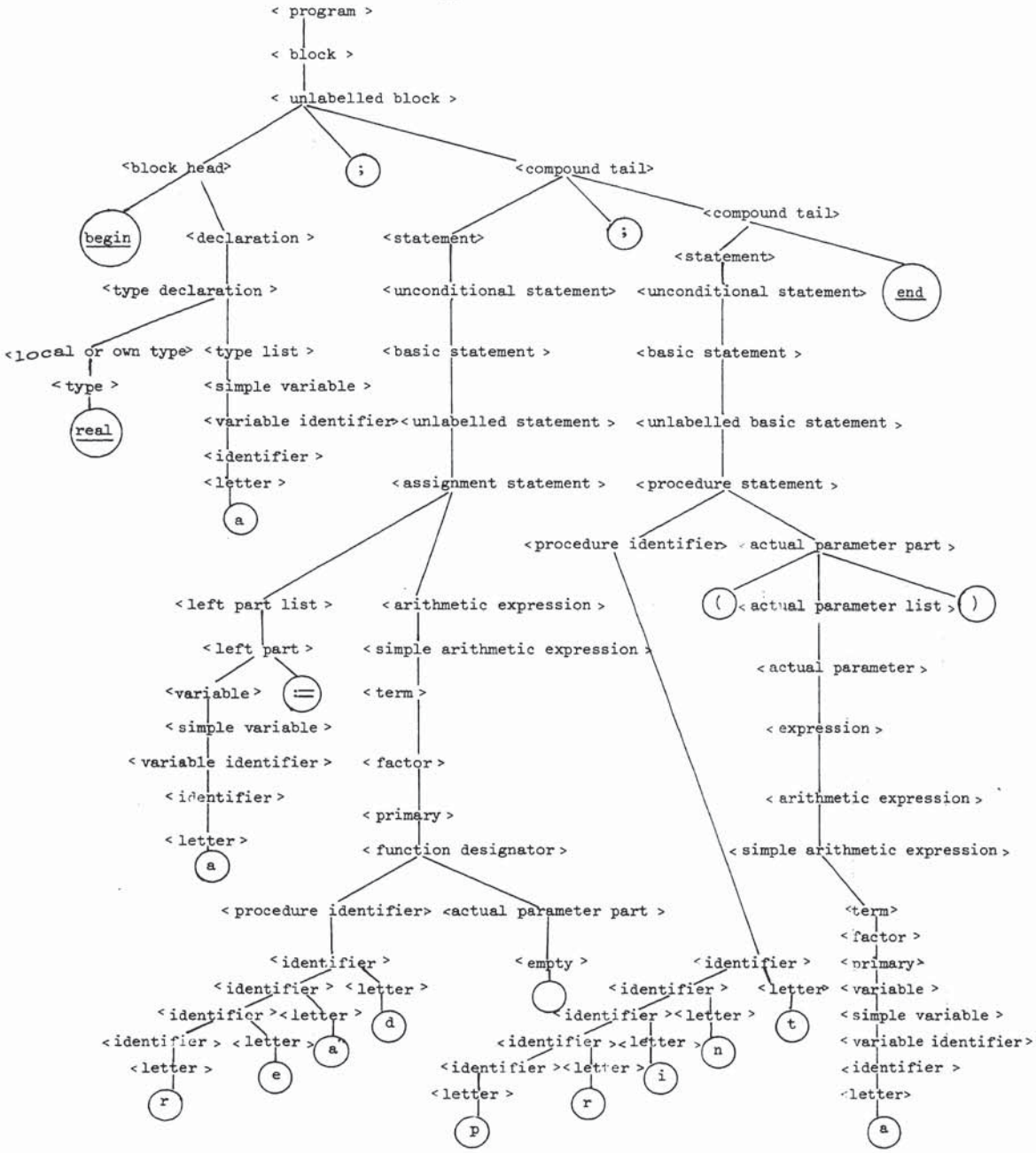
Hoe dit programma uit <program> voort te brengen is, wordt getoond op blz. 33 met behulp van een zgn. derivatieboom.
We merken op dat het programma een block is, dat bestaat uit een declaratie gevolgd door twee statements onderling van elkaar gescheiden door een puntkomma. De declaratie "real a" introduceert (II-1) een variabele *), die waarden van het type real kan dragen. De uitvoering van de assignment statement "a:= read" houdt in dat de waarde afgeleverd door de arithmetische expressie geassigneerd wordt aan de drager (hier een simple variable) "a".
Voortdurend gebruiken we bij de behandeling van een programma dus begrippen, die hun definitie vinden in de Syntax van ALGOL 60.
Verder willen wij erop wijzen, dat de Semantiek de dingen ook efficiënt uitlegt. Begrijpt men b.v. wat uitvoering van a:= read inhoudt, dan zal men ook van a:= 3.14 of a:= 3.14 × 2 de betekenis inzien.
We merken op dat we een derivatieboom getekend hebben, die erop gericht is precies onze oorspronkelijke tekst te genereren.
Met behulp van het schema kan men evengoed

begin real a; b:= read; print(c) end

genereren, maar dat is dan geen ALGOL 60 programma overeenkomstig onze definitie in III-4.

---

*) Om didactische redenen spreken we in de syllabus vaak van drager. Het is wellicht goed op te merken dat hiermee hetzelfde bedoeld wordt als met het begrip variabele.


< program >
< block >
< unlabelled block >
<block head>
;
<compound tail>
begin
<declaration >
<statement>
;
<compound tail>
<statement>
end
<type declaration >
<unconditional statement>
<unconditional statement>
<local or own type>
<type list >
<basic statement >
<basic statement >
< type >
<simple variable >
real
<variable identifier>
<unlabelled statement >
<unlabelled basic statement >
<identifier >
<letter >
<assignment statement >
<procedure statement >
a
<procedure identifier>
<actual parameter part >
<left part list >
<arithmetic expression >
(
<actual parameter list >
)
< left part >
<simple arithmetic expression >
<actual parameter >
<variable >
:=
< term >
< simple variable >
< expression >
< variable identifier >
< factor >
< identifier >
< arithmetic expression >
< primary >
< letter >
< function designator >
< simple arithmetic expression >
a
< procedure identifier>
<actual parameter part >
<term>
< factor >
< identifier >
< empty >
< identifier >
< primary>
< identifier >
< letter >
< identifier >
< letter>
< variable >
< identifier >
< letter >
d
< identifier >
< letter >
t
< simple variable >
< identifier >
< letter >
a
< letter >
n
< variable identifier>
< identifier >
< letter >
e
< identifier >
< letter >
i
< identifier >
< letter >
r
< letter >
r
< letter>
p
a

IV. Expressies

1. algemeen

Een expressie is een regel voor het bepalen van een waarde.

Men onderscheidt:

1. Arithmetische expressies, die een getalwaarde afleveren.
2. Boolean expressies, die een logische waarde (true of false) afleveren.
3. Designational expressies, die een label afleveren.

Toegestane bestanddelen van expressies zijn:

1. variables, function designators, switch designators en labels
2. getalwaarden of logische waarden (bijv. 1, 2, 1.5, true of false)
3. arithmetische operatoren: +, −, ×, /, ÷ of ↑
4. relationele operatoren: <, ≤, =, ≥, > of ≠
5. logische operatoren: ≡, ⊃, ∨, ∧ of ¬
6. if then else constructies
7. haakjes: (en)

De waarde, die door een expressie bepaald wordt, is afhankelijk van:

1. de waarden door de erin voorkomende variables en function designators gedragen resp. afgeleverd.
2. het effect en de prioriteit van de afzonderlijke operatoren.

2. arithmetische expressie

Een arithmetische expressie dient om een nieuwe getalwaarde te berekenen uit een aantal bekende getalwaarden. Dit is iets dat in de "gewone wiskunde" ook vaak gedaan wordt, alleen zijn de conventies die daar gelden niet zonder meer overgenomen in ALGOL 60; de oorzaak hiervan ligt in het feit dat een variabele in ALGOL 60 een naam van meerdere letters mag hebben (x:= A2 betekent niet x gaat een waarde, die gelijk is aan tweemaal de waarde van A, dragen). Een expressie zal in het algemeen bestaan uit een rij getallen of namen (eventueel geïndiceerd of "geparametriseerd") onderling gescheiden door operatoren.

De vorm, waaraan een expressie in ALGOL 60 moet voldoen, wordt gegeven door de volgende syntactische regels:

<adding operator>::= + | -

<multiplying operator>::= × | / | ÷

<primary>::= <unsigned number>|<variable>|<function designator>|
(<arithmetic expression>)

<factor>::= <primary>|<factor>↑<primary>

<term>::= <factor>|<term><multiplying operator><factor>

<simple arithmetic expression>::= <term>|<adding operator><term>|
<simple arithmetic expression><adding operator><term>

<if clause>::= <u>if</u> <Boolean expression> <u>then</u>

<arithmetic expression>::= <simple arithmetic expression>|
<if clause><simple arithmetic expression> <u>else</u> <arithmetic expression>

Hieruit kan men afleiden:

a. Twee operatoren kunnen nooit naast elkaar voorkomen (a:= 2↑ - 3 is geen ALGOL 60).

b. Een primary kan weer een arithmetische expressie omsloten door haakjes zijn; dit houdt b.v. in dat a:= 2↑(-3) wel correct is en dat, zoals bij de statements het <u>then</u> - <u>if</u> verbod opgevangen kan worden door gebruik van <u>begin</u> en <u>end</u>, het hier kan door het zetten van haakjes.

c. Als operatoren zijn alleen toegelaten:

+, -, ×, /, ↑ en ÷

<u>Opmerking</u>: Optellen (+), aftrekken (-), vermenigvuldigen (×), delen (/) en machtsverheffen (↑) hebben de gebruikelijke betekenis. Voor de betekenis van de operator helen (÷)*) zij verwezen naar blz. 16.
De bewerkingen in een expressie worden, indien mogelijk in onderstaande volgorde naar prioriteit uitgevoerd:

---

*) De operator "helen" wordt in het MC-ALGOL 60-systeem gepresenteerd als <u>:</u> d.w.z. als een onderstreepte dubbele punt.

1. dat wat binnen de haakjes staat moet worden opgevat als één geheel.
2. ↑
3. ×, /, ÷
4. +, -

Bij operatoren van gelijke prioriteit geeft bovenstaand schema geen uitsluitsel. In dat geval geldt dat de meest linkse operator het eerst wordt toegepast.

d. Een expressie kan nooit beginnen met ×, ↑, / of ÷; dit zijn zgn. dyadische operatoren.

e. De operatoren + en - kunnen zowel monadisch als dyadisch zijn.

De waarde van de arithmetische expressie wordt nu als volgt bepaald:

Als de arithmetische expressie de vorm:
"<u>if</u> <Boolean expression> <u>then</u> <simple arithmetic expression> <u>else</u> <arithmetic expression>"
heeft, wordt de "simple arithmetic expression" of de "arithmetic expression" op grond van de uitkomst van de "Boolean expression" geselecteerd en uitgevoerd.
Anders heeft de arithmetische expressie de vorm:
een aantal termen minstens één, gescheiden door + of - tekens eventueel voorafgegaan door een + of - teken (gedurige som). Nu worden eerst alle termen geëvalueerd en vervolgens de inversie, optellingen of aftrekkingen van links naar rechts uitgevoerd. Elke term bestaat uit één of meer factoren, gescheiden door ×, / of ÷ tekens. Alle factoren worden geëvalueerd en vervolgens de inversie, vermenigvuldigingen, delingen of helingen van links naar rechts uitgevoerd.
N.B. a/b × c is (a/b) × c en niet a/(b × c).
Elke factor bestaat uit één of meer primaries gescheiden door ↑ tekens. Alle primaries worden geëvalueerd en de machtsverheffingen van links naar rechts uitgevoerd. Dus 2 ↑ 3 ↑ 4 = (2 ↑ 3) ↑ 4 en niet 2 ↑ (3 ↑ 4).
De waarde van een primary is:

a. voor een "unsigned number" - de waarde, die het representeert
b. voor een "variable" of "drager" - de waarde, die hij op dat moment draagt

c. voor een "function designator" - de waarde, die bepaald wordt uit de procedure-body van zijn declaratie (zie hoofdstuk VI)

d. voor een "arithmetische expressie omsloten door haakjes" - de waarde van de arithmetische expressie.

Behalve in de waarde van het resultaat van de uitwerking van een expressie, zijn we tevens geïnteresseerd in het type van het resultaat.
Wat de monadische operatoren betreft, liggen hier geen moeilijkheden, want het type van een monadische operator gevolgd door een operand is hetzelfde als het type van de operand.
Bij de dyadische operatoren +, -, ×, / en ÷ kan men het type van het resultaat van

A operator B

aflezen in de volgende tabel:

A	B	A+B	A-B	A×B	A/B	A÷B
real	real	real	real	real	real	ongedefiniëerd
real	integer	real	real	real	real	ongedefiniëerd
integer	real	real	real	real	real	ongedefiniëerd
integer	integer	integer	integer	integer	real	integer

Is A ↑ B gedefiniëerd dan is het type van het resultaat integer als

1. A en B van het type integer zijn en
2. B ≥ 0 is;

In alle andere gevallen, waar A ↑ B gedefiniëerd is, is het type van het resultaat real.

Waartoe dient deze kennis? Wel denk eens aan een assignment (bv. a:= read") van de algemene gedaante:

<variable>:= <expression>

daar speelden zoals in het voorafgaande vermeld 3 processen:

1. het klaarzetten van de drager(s)

2. het berekenen van de waarde van de expressie

3. het assigneren van de waarde onder 2 aan de drager(s) onder 1; Is het resultaat onder 2. van een ander type dan de drager onder 1. kan dragen dan wordt dit type automatisch aangepast aan de drager d.w.z. een real wordt afgerond tot een integer of een integer wordt tot real gemaakt; zijn er meerdere dragers dan moeten die alle waarden van hetzelfde type kunnen dragen (Revised Report 4.2.4).

In het programma:

```
begin integer a; a:= 3.14; print(a) end
```

zal zoals uit het bovenstaande valt af te leiden de waarde 3 geprint worden. Behalve in een assignment statement kan een expressie ook voorkomen als subscript, maar ook daar hebben we het begrip assignment nodig.
Zo zal in het programma:

```
"begin real array a[1:10]; real b; b:= 3.14; a[b]:= b end"
```

a[3] de waarde 3.14 krijgen, daar volgens het Revised Report 3.1.4.2 iedere subscript positie zich gedraagt als een variabele van het type integer en de berekening van deze expressie aequivalent is met een assignment aan deze fictieve variabele. Hetzelfde geldt voor expressies in de "bounds" van een array, bij activering van een block.

3. Boolean expressie

Boolean *) expressies vertonen een grote analogie met arithmetische expressies. In het algemeen wordt ook bij uitvoering van Boolean expressies eerst een simple Boolean expression geselecteerd met behulp van een if then else geraamte.
Afgaande op de prioriteit van de logische operatoren worden nu de constituerende delen van deze simple Boolean expression geëvalueerd.

---

*) George Boole (1815-1864) is één van de grondleggers van de formele logica. Hij schreef o.a. "An investigation of the Laws of Thought".

Dit proces loopt altijd uit op de vraag naar de waarde van zekere Boolean primaries. Een Boolean primary kan nu behalve een logische waarde, een variabele, een function designator of een Boolean expressie omsloten door haakjes ook nog een relatie zijn.
Een relatie bestaat uit twee simpele arithmetische expressies als operanden en een dyadische relationele operator ertussen.
Wordt nu tijdens de executie de logische waarde van een relatie gevraagd, dan levert hij true af als de overeenkomstige relatie tussen de getalwaarden, die de arithmetische expressies afleveren, - zoals we in de "gewone wiskunde" zeggen - geldt, anders false.
De waarde bepaald door een logische waarde, een variabele of een function designator is - zoals bij arithmetische expressies - de waarde, die hij representeert, draagt resp. aflevert. Een Boolean variabele noemt men een Boolean.
Daar er maar 2 verschillende logische waarden zijn, kan men het effect van een operator vastleggen door alle mogelijkheden op te sommen. Dit kan met behulp van een zgn. waarheidstafel:

a	b	a∧b	a∨b	¬a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Daar de operatoren ∨ en ∧ symmetrisch zijn kan men hun werking ook als volgt samenvatten:
De operator ∧ levert true af als beide operanden true afleveren anders false.
De operator ∨ levert false af als beide operanden false afleveren anders true.
De monadische operator ¬ levert true af als de operand false is anders false.

Voor de meeste programma's zijn ∧, ∨ en ¬ toereikend; ALGOL 60 kent nog twee operatoren te weten "≡" en "⊃", die echter zelden gebruikt worden. Het effect van deze operatoren is echter eenvoudig te vinden uit de waarheidstafel in paragraaf 3.4.5 van het Revised Report.
De prioriteitsregels zijn:

1. Bepaal de waarde van de arithmetische expressies.
2. Bepaal de waarde van de relaties.
3. ¬
4. ∧
5. ∨

Ook hier geldt dat bij gelijke prioriteit volgens bovenstaande regels, de meest linkse operator het eerst wordt toegepast.

4. <u>designational expressie</u>

Designational expressies vertonen weinig overeenkomst met de andere expressies door het ontbreken van operatoren op labels.
De enige componenten van designational expressies zijn: labels, switch designators, haakjes en <u>if</u> <u>then</u> <u>else</u> constructies. We laten hierbij dus de Boolean expressies tussen <u>if</u> en <u>then</u> en arithmetische expressies als index buiten beschouwing.

De waarde van een designational expressie wordt nu als volgt bepaald:
Als de designational expressie de vorm:

"<u>if</u> <Boolean expression> <u>then</u> <simple designational expression>
<u>else</u> <designational expression>

heeft, wordt op grond van de uitkomst van de "Boolean expression", de "simple designational expression" of de "designational expression" geselecteerd en uitgevoerd. Anders is het een simple designational expression. De waarde van een simple designational expression is:
voor een label, die label
voor een designational expression omsloten door haakjes, de waarde van die designational expression
voor een switch designator de waarde van de designational expression, die in de met de switch designator corresponderende switch declaratie door de subscript expression wordt aangewezen.

V. Statements

1. overzicht, basic statement

De statements zijn als het ware de "werkwoorden" in ALGOL 60. Dit hoofdstuk is gewijd aan een overzicht van de verschillende statements en hun betekenis. Theoretisch is dit overzicht in de vorm van metalinguïstische formules in het Revised Report gegeven, practisch is onderstaand overzicht handiger. De reden hiervan is dat elke statement gelabeld mag worden, maar niet overal waar een statement staat een willekeurige andere statement mag staan.
Laten we begrippen als "unlabelled block", "unlabelled compound" en "unlabelled basic statement" buiten beschouwing, dan kunnen we de statements als volgt onderverdelen:

- statement
  - unconditional statement
    - compound statement
    - block
    - basic statement
      - assignment statement
      - goto statement
      - dummy statement
      - procedure statement
  - for statement
    - <for clause> statement
  - conditional statement
    - <if clause> unconditional statement
    - <if clause> for statement
    - <if clause> unconditional statement <u>else</u> statement

Op pagina 29 gaven we de volgende Syntaxregels uit het Revised Report (4.1).

<program>::= <block>|<compound statement>
<block>::= <unlabelled block>|<label>:<block>
<unlabelled block>::= <block head>;<compound tail>
<compound tail>::= <statement> <u>end</u> |<statement>;<compound tail>
<compound statement>::= <unlabelled compound>|<label>:<compound statement>
<unlabelled compound>::= <u>begin</u> <compound tail>

Uit het schema en bovengenoemde syntaxregels kan men de volgende conclusies trekken:

1. In elk programma staat minstens één basic statement.
2. De basic statement's doen "het eigenlijke werk".
   Zelfs in het programma begin end staat één basic statement nl. de dummy statement; de uitvoering hiervan houdt in dat er niets gebeurt.

Als we sprongen in het programma buiten beschouwing laten, gebeurt er dus alleen iets als er een assignment statement of een procedure statement uitgevoerd wordt. Zoals we in hoofdstuk VI zullen zien is het uitvoeren van een procedure statement, wanneer de procedure body in ALGOL 60 geschreven is, te herleiden tot de uitvoering van andere basic statements.
Generaliserend kunnen we dus zeggen dat het er bij de executie van een programma, afgezien van niet in ALGOL 60 geschreven procedure bodies, om gaat in welke volgorde de assignment statements van het programma worden uitgevoerd.

Deze volgorde wordt bepaald:

1. expliciet door een goto statement
2. impliciet door een for statement of conditional statement
   en anders door de volgorde in de tekst over het programma.

Een label afgeleverd door de designational expressie van een goto statement bepaalt de statement die na de goto statement zal worden uitgevoerd. Kan dat elke statement uit het programma zijn?

Hiervoor geldt:

Daar elke statement van het programma tijdens de executie een - weliswaar beperkte - kennis van de voorgeschiedenis van het rekenproces veronderstelt, mag de label alleen die statements aanwijzen, waarvan de voorgeschiedenis bekend is of niet terzake doet.

Dit principe is uitgewerkt in:

1. Revised Report 4.3.4.
   Staat een goto statement buiten een block dan mag hij geen labels bevatten, die locaal zijn ten opzichte van dat block (relevante voorgeschiedenis is kennisname declaraties van dat block).

2. Revised Report 4.6.6.
   Staat een goto statement buiten een for statement en bevat hij een label, die een statement binnen die for statement aanwijst, dan is de betekenis van die goto statement ongedefiniëerd (relevante voorgeschiedenis is de assignment direct na for).
3. Revised Report 4.3.4.
   Staat een goto statement buiten een compound statement en bevat hij een label, die een statement binnen die compound statement aanwijst, dan wordt de aangewezen statement tijdens de executie direct na de goto statement uitgevoerd (relevante voorgeschiedenis is passeren {begin}).
4. Revised Report 4.5.4.
   Voor de betekenis van een goto statement die een label bevat die leidt in een conditional statement zie V-2 (relevante voorgeschiedenis is de waarde van de Boolean expressie).

2. conditional statement

Een voorbeeld van een conditional statement vinden we in het volgende programma dat een tekst afdrukt, waarin alle hoofdletters vervangen worden door de corresponderende kleine letters. De getallenband, die de tekst bevat, moet worden afgesloten met een ⌉ (interne representatie 76).

```
1 begin comment 1739, N. van de Auteur (MC), afdrukken van een tekst,
2       waarin alle hoofdletters vervangen worden door de corresponderen-
3       de kleine letters;
4       integer a, bandafsluiter;
5       bandafsluiter:= 76;
6       for a:= RESYM while a ≠ 76 do
7       if a > 35 ∧ a < 63 then PRSYM(a - 27) else PRSYM(a)
8 end
9
```

We zien op regel 6 en 7 een for statement, die bestaat uit een for clause op regel 6 en een conditional statement op regel 7.
De werking of - zo men wil - de betekenis van statements wordt, wanneer zij meer gecompliceerde structuren voorstellen, b.v. de for statement en de conditional statement, uitgelegd in termen van constituerende delen. Wil men dit begrijpen, dan moet men de structuur, hier de conditional statement, syntactisch herkennen.
Hiervoor is het nodig dat men de volgende syntaxregels (Revised Report 4.5.1) kent:

```
<conditional statement>::= <label>:<conditional statement>|
                           <if statement>|
                           <if clause><for statement>|
                           <if statement> else <statement>

<if statement>::= <if clause><unconditional statement>

<if clause>::=  if <Boolean expression> then
```

Als we in het programma een conditional statement herkennen, moeten we een structuur ontwaren van de gedaante:

```
     if Boolean expression then unconditional statement
                        of
     if Boolean expression then for statement
                        of
     if Boolean expression then unconditional statement else statement
```

Krachtens deze herkenning spreken we bij de uitleg over zijn Boolean expressie, zijn if, then en else (indien aanwezig). Er kunnen nog wel meer basic symbols if in voorkomen, maar die bedoelen we niet als we het over zijn if hebben.

De werking van een conditional statement is:

1. Vraag de waarde van zijn Boolean expression (reken hem uit).
2. Is deze true, dan is de werking: het uitvoeren van de unconditional statement of for statement na zijn then.

Is deze false dan is de werking: indien zijn else aanwezig is het uitvoeren van de statement na else anders beperkt de uitvoering zich tot punt 1.

Soms wordt niet de hele conditional statement, maar slechts een gedeelte uitgevoerd. Een voorbeeld hiervan vinden we in het volgende programma.

```
1  begin comment 1739, N. van de Auteur (MC), afdrukken van een tekst,
2        waarin alle hoofdletters vervangen worden door de corresponderen-
3        de kleine letters (hoe het niet moet);
4        integer symbol, bandafsluiter;
5
6        bandafsluiter:= 76;
7
8        symbol:= RESYM;
9  loop: if symbol > 35 ∧ symbol < 63
10       then begin symbol:= symbol - 27;
11                1: PRSYM(symbol);
12                   symbol:= RESYM
13            end
14       else if symbol ≠ bandafsluiter then goto 1 else goto klaar;
15 111:  goto loop;
16 klaar:
17 end
18
```

In dit programma is steeds, wanneer de door "loop" gelabelde statement moet worden uitgevoerd, begrip van de werking van de conditional statement vereist.
Het is duidelijk wat het effect van de statement

"goto 1"

moet zijn: achtereenvolgens worden de statements PRSYM(symbol) en symbol:= RESYM uitgevoerd. Maar wat gebeurt er daarna?
Een goto, die leidt binnen een compound statement, is volgens V-1 geoorloofd. Het end levert dus geen moeilijkheden op.

Het effect van de delimiter else is dat de uitvoering van de conditional statement, die dat else als zijn else heeft, als beëindigd wordt beschouwd.
In dit geval zal de executie voortgezet worden bij de label 111, die alleen om instructieve redenen in de tekst van het programma is opgenomen (in wezen dus niet fout maar wel overbodig).

Tot slot merken we op dat we uit het schema op blz. 41 (en eveneens uit de syntax) kunnen afleiden:

1. then mag niet direct gevolgd worden door if (dit is het zgn. then if verbod dat ook in expressies geldt).
2. na de combinatie then for mag geen else volgen, dat met then correspondeert.

3. for statement

We gaan het volgende probleem oplossen:
In een tekst die bestaat uit identifiers, moeten de frequenties geteld worden, waarmee die identifiers in de tekst voorkomen. Het komt er dus op neer dat er een getallenband - de tekst - gelezen moet worden en dat de erin voorkomende identifiers alsmede hun frequenties moeten worden afgedrukt.

De getallenband moet aan de volgende voorwaarden voldoen:

1. Hij moet beginnen met één of meer "nieuwe regels" of spaties.
2. Identifiers moeten onderling worden gescheiden door één of meer spaties of één of meer nieuwe regels (binnen identifiers zijn er dus geen spaties of overgangen op een nieuwe regel).
3. De tekst moet worden afgesloten door één of meer spaties of één of meer nieuwe regels gevolgd door het ⌉ symbool.

Het programma op blz. 50 dat dit probleem oplost, zit in grote lijnen als volgt in elkaar:

De statement op regel 10 t/m 37 wordt onder controle van de for clause op regel 11 steeds weer opnieuw uitgevoerd, zolang er nog een identifier op de band staat; deze statement valt uiteen in 4 logische stukken:

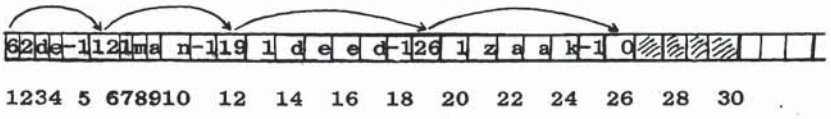
1. Regel 12 - 13; de scheiders tussen de identifiers worden gelezen.
2. Regel 16 - 19; er wordt een identifier ingelezen en achter na de vorige identifier in "lijst" gezet.
3. Regel 23 - 26; de identifier wordt in "lijst" opgezocht, beginnende bij het begin van de lijst.
4. Regel 29 - 36; wordt de identifier gevonden op de plaats, waar hij onder punt 2 is neergezet (m.a.w. kwam hij toen nog niet in de lijst voor) dan wordt de frequentie op 1 gezet; wordt hij eerder aangetroffen dan wordt daar ter plaatse de frequentie met 1 verhoogd en de plaats waar de identifier zojuist is neergezet blijft beschikbaar voor de volgende identifier.

In de statement op regel 39 t/m 46 wordt de inhoud van "lijst" met de gevonden frequenties afgedrukt.

Voor een goed begrip is het misschien goed te weten dat na inlezen van de tekst:

"de man deed de zaak"

het eerste stuk van het array "lijst" als volgt gevuld moet zijn:



Hierbij tekenen we het volgende aan:

1. Een identifier wordt in "lijst" afgesloten met -1 (zie lijst [5], lijst [11], lijst [18] en lijst [25]).
2. De index die aangeeft waar de plaats van een identifier in "lijst" begint, wordt in het programma steeds aangeduid met "pointer"; de eerste letter staat in lijst[pointer + 2].
3. Is lijst[pointer] $\neq$ 0 dan houdt dat in dat er minstens nog één identifier in lijst staat. Tevens geeft de waarde van lijst[pointer] aan, waar de plaats van die identifier begint; in bovenstaande tekening wordt dit benadrukt door de "pijlen".

4. In lijst[pointer + 1] staat de frequentie van de identifier behorende bij pointer.
5. De waarde van endpointer geeft steeds het einde van de laatste van de tot nu toe verwerkte en verschillend bevonden identifiers aan. Na het verwerken van een identifier geldt: lijst[endpointer] = 0.

Bevat de tekst zeer veel verschillende woorden dan zal op regel 17 tijdens inlezen een foutmelding gegeven worden en is al het werk voor niets geweest. Er is kennelijk van uitgegaan dat een array met 10.000 elementen ruimschoots voldoende is. In het voorafgaande hadden alle for statements slechts één for list element. Volgens het Revised Report (4.6.1) mag een "for list" ook uit meerdere "for list elements" bestaan.

Regel 16 van ons programma bevat een dergelijke "for-list":

<u>for</u> a:= endpointer + 2, a + 1 <u>while</u> symbol < 38 <u>do</u>

De betekenis van een for statement met een dergelijke for list is:

Eerst wordt de assignment a:= endpointer + 2 uitgevoerd en daarna de statement achter <u>do</u>. Vervolgens wordt a steeds met 1 verhoogd en de statement achter <u>do</u> uitgevoerd zolang symbool < 38 is.

Voor een zeer precieze beschrijving van de for statement verwijzen we naar het Revised Report (4.6.4), waar de werking van de for statement in primitieve - voor ons nu begrijpelijke - statements wordt uitgelegd. Dit betekent onder meer dat in plaats van for statements aequivalente constructies met behulp van goto statements gemaakt kunnen worden (aan de raden is dit uiteraard niet).

We merken verder op:

1. De expressies behoeven niet af te hangen van de variabele die direct na <u>for</u> genoemd wordt - de zgn. controlled variable (b.v. <u>while</u> <u>true</u> op regel 23).
2. Is de for list "exhausted" dan is de waarde van de controlled variable ongedefiniëerd; de waarde van "a" na de for statement op regel 16 t/m 19 is dus ongedefiniëerd.
3. Wordt een for statement met een goto verlaten dan is de waarde van de controlled variable wel gedefiniëerd. Tijdens executie kan dit b.v.

voorkomen op regel 25 - er worden dan zelfs twee for statements tegelijk verlaten. De controlled variable "pointer" wijst aan, waar de identifier zich in "lijst" bevindt, de controlled variable "a" geeft de lengte van de identifier plus 3.

Over het programma zij tot slot opgemerkt, dat het niet nagaat of de getallenband inderdaad aan de gestelde voorwaarden voldoet.

```
1   begin   comment   1739, N. van de Auteur(MC) frequenties van identifiers;
2               integer array lijst[1:10000];
3               integer a, symbool, pointer, endpointer, space, nlcr,
4                       bandafsluiter;
7               space:= 93; nlcr:= 119; bandafsluiter:= 76;
8               endpointer:= 1;
9
10  eerste letter:
11              for symbool:= RESYM while symbool ≠ bandafsluiter do
12              begin   if symbool = nlcr ∨ symbool = space
13                      then goto eerste letter;

16                      for a:= endpointer + 2, a+ 1 while symbool < 38 do
17                      begin   lijst[a] := symbool; lijst[a+1] := -1
18                              symbool:= RESYM
19                      end identifier gelezen;


23                      for pointer:= 1, lijst[pointer] while true do

25                      for a:=2,a+1 while lijst[pointer+a]≠lijst[endpointer+a] do
26                      if lijst[pointer + a]= -1 then goto honoreer identifier;


29              honoreer identifier:
30                      if pointer = endpointer
31                      then    begin   endpointer := endpointer + a + 1;
32                                      lijst[pointer] := endpointer;
33                                      lijst[pointer + 1]:= 1;
34                                      lijst[endpointer]:= 0
35                              end
36                      else    lijst[pointer + 1]:= lijst[pointer + 1] + 1
37              end tekst is nu geheel gelezen;
38
39              for pointer:= 1, lijst[pointer] while lijst[pointer] ≠ 0 do
40              begin   NLCR;
41                      for a:= pointer + 2, a+ 1 while lijst[a]≠ -1 do
42                      PRSYM(lijst[a]);
43
44                      TAB; PRINTTEXT(‡komt voor‡); ABSFIXT(6,0,lijst[pointer + 1 ]);
45                      PRINTTEXT(‡maal‡)
46              end alle woorden met hun frequenties zijn nu afgedrukt
47  end
48
```

VI. Procedures

1. eenvoudige procedures, overzichtelijkheid en inzicht in de werking van het programma, call by name *)

We hebben één soort quantity - een grootheid, waar een identifier aan gehecht kan worden - nog niet uitvoerig behandeld nl. de procedure.
De funktie van een procedure in een programma is het beschrijven van een bepaald deelproces. Dit deelproces wordt beschreven in één statement - de procedure body -, welke een onderdeel vormt van de procedure-declaratie. De werking van dit deelproces kan beïnvloed worden door eventueel aanwezige parameters.
Enkele motieven voor het gebruik van procedures zijn:

1. bekorting van het programma door afkorting van een stuk tekst, dat anders op meerdere plaatsen in het programma voluit herhaald had moeten worden;
2. vergroting van het inzicht in de werking en de structuur van het programma;
3. verbetering van de communicatie tussen programmeurs, die aan hetzelfde programma werken;
4. oplossing van een op natuurlijke wijze recursief te formuleren probleem.

Een eenvoudige toepassing van procedures vindt men in het programma op blz. 56. De executie van dit programma met de getallenband van blz. 54 levert o.a. het titelblad van deze syllabus.
Een procedure-declaratie heeft de vorm:

---

*) Een formele parameter is een value-parameter of een name-parameter, afhankelijk van het feit of hij in de value-lijst is opgenomen of niet. De formele parameters in VI-1 zijn alle name-parameters. Value-parameters worden behandeld in VI-2.

<u>procedure</u> <procedure heading><procedure body>

In het bedoelde programma is overal de formule toegepast:

<procedure body> ::= <statement>

Dit statement kan een block zijn zoals op regel 26 t/m 29 bij "horizontaal" of een - meestal gecompliceerde - statement.
De procedure A gedeclareerd in regel 6, wordt alleen op regel 49 één keer gebruikt. De procedures A, L, G en O zijn geïntroduceerd om het inzicht in de werking van het programma te bevorderen. Waren we hier niet in geïnteresseerd dan hadden we regel 6 t/m 8 kunnen schrappen en

A(regel)

kunnen vervangen door

(*) <u>if</u> regel = 1 ∨ regel = 4 <u>then</u> horizontaal(regel,letterbreedte)
<u>else</u> verticaal(regel,<u>true</u>)

De output van het programma zou dan exact hetzelfde geweest zijn. (We zouden een volledig aequivalent programma hebben gekregen.)
We hebben echter aan overzichtelijkheid verloren, terwijl we aan tekst nauwelijks hebben gewonnen (3 symbolen - maar in basic symbols geteld, zelfs verloren!).
De procedure L wordt op regel 49 twee keer gebruikt en bekort dus inderdaad het programma.
Komt de rekenautomaat aan de uitvoering van A(regel) toe, dan moet men zich voorstellen dat hij eerst een proces uitvoert dat aequivalent is met (maar niet gelijk is aan) het vervangen van A(regel) door de statement (*) en vervolgens deze statement uitvoert.
Wanneer de statement (*) wordt uitgevoerd moet weer een andere procedure nl. of horizontaal of verticaal worden uitgevoerd.
We zien dat er kennelijk meerdere actuele parameters mogen zijn, mits ze onderling gescheiden worden door een komma. Hierdoor kunnen we ze nummeren van links naar rechts. In de procedure statement moet nu het aantal actuele parameters even groot zijn als het aantal formele in de procedure-declaratie. Een formele parameter, called by name, is een identifier die

niet gehecht is aan een quantity, maar in de tekst slechts een plaats openhoudt voor een actuele parameter. Hij wordt dus ook nergens gedeclareerd.
Wordt de procedure uitgevoerd en zijn het zoals in dit programma allemaal "name"-parameters dan wordt de i^e formele overal in de body vervangen door de i^e actuele.
Soms wil men wel eens iets vertellen over de actuele parameter die men van plan is aan te bieden. Dit is mogelijk door middel van een zgn. specificatie. De syntax geeft gelegenheid voor de specificatie in de procedure heading, Een voorbeeld van een specificatie geeft

<u>Boolean</u> twee (zie regel 30 van het programma).

De informatie hier verstrekt is:
Wordt tijdens de executie "verticaal" aangeroepen dan zal de plaats door "twee" opengehouden ingenomen moeten worden door ofwel een identifier, die gehecht is aan een Boolean variabele ofwel een expressie, die een waarde van het type Boolean aflevert (zie ook VI-3).
In het programma zien we eerst 8 declaraties en dan de statements, die beginnen op regel 37 met een for statement.

Wat doet nu het programma?
Het drukt "ALGOL" af als volgt:

Alle letters zijn even breed en de afstand tussen twee letters is gelijk aan de letterbreedte. Er worden 5 "regels" opgebouwd, de eerste uit A's, de tweede uit L's, de derde uit G's, de vierde uit O's en de vijfde weer uit L's.

De dikte van een "regel" kan van titelblad tot titelblad met behulp van lijndikte gevariëerd worden. De statements op regel 41 t/m 44 zorgen ervoor dat de te variëren gegevens ingelezen worden. Een afdruk van de getallenband met behulp van de flexowriter *) geeft:

```
algol
5        20        6        1        1
5        20        6        2        1
5        20        8        3        1
5        20       12        3        1
5        20       10       10        1
5        20       12       10        1
5        20       16                10        0.
```

Het array "letter" wordt gevuld door de for statement op regel 37 met letters, die op de getallenband staan, nl. "a", "l", "g", "o" en "l".
Bij het titelblad was de "beginregel" 5, de "beginpositie" 20, de "letterbreedte" 8 en de "lijndikte" 4.
Op regel 46 zien we dat voor 5 waarden van de controlled variable "regel" de for statement op regel 47 t/m 50 tijdens de executie uitgevoerd zal worden.
Zoals in het voorafgaande uitéengezet is de werking van de procedure statement A(regel) de uitvoering van de statement die de body vormt van de procedure A na vervanging overal van de formele parameter "r" door de actuele "regel". Hierbij moeten we ons dan weer afvragen wat de werking is van de statement

horizontaal(regel,letterbreedte)

als "regel" de waarde 1 draagt. Men ziet gemakkelijk in dat dan over een letterbreedte a's gedrukt worden.
De procedure "horizontaal" is geïntroduceerd om het programma te bekorten. Hij wordt dan ook 5 maal gebruikt.

---

*) Zie voetnoot I-1.

Bij het afdrukken van een gedeelte van een letter is het belangrijk dat de positie op de regel bij het begin van de volgende letter goed uitkomt. Dit wordt gegarandeerd door 3 feiten:

a. In de procedure bodies van A, L, G en O wordt éénmaal of "horizontaal" of "verticaal" aangeroepen;

b. De procedure bodies van "horizontaal" en "verticaal" eindigen beide met SPACE(letterbreedte);

c. Wordt als gevolg van een call van de procedures A, L, G of O "horizontaal" of "verticaal" aangeroepen, dan zal er voor de uitvoering van hun laatste statement SPACE(letterbreedte) precies één letterbreedte van de letter a, l, g of o geprint zijn.

Opmerking: De eerste twee kunnen direct aan het programma afgelezen worden; c is te controleren door alle gevallen in de programmatekst na te gaan.

De letters zullen dus goed afgedrukt worden, wanneer er met de A goed begonnen wordt (vandaar: NLCR; SPACE(beginpositie)) en als het aantal aanslagen, dat gelijk is aan 10 × letterbreedte + beginpositie, op één regel kan (kleiner 145 is).

```
BEGIN   COMMENT R1739 M.K.KOKSMA PROGRAMMA VOOR AFDRUKKEN
        TITELBLAD ALGOL-60 SYLLABUS;
        INTEGER BEGINREGEL, BEGINPOSITIE, LETTERBREEDTE,
                LIJNDIKTE,REGEL, DIKTE, I;
        INTEGER ARRAY LETTER[1:5];
        PROCEDURE A(R);
                IF R=1 ∨ R=4 THEN HORIZONTAAL(R,LETTERBREEDTE)
                             ELSE VERTICAAL(R,TRUE);
        PROCEDURE B(R);
                IF R=5       THEN HORIZONTAAL(R,LETTERBREEDTE)
                             ELSE VERTICAAL(R,FALSE);
        PROCEDURE C(R);
                IF R=1 ∨ R=5 THEN HORIZONTAAL(R,LETTERBREEDTE)
                ELSE
                IF R=2       THEN VERTICAAL(R,TRUE)
                ELSE
                IF R=3       THEN VERTICAAL(R,FALSE)
                ELSE
                BEGIN   PRSYM(LETTER[R]); SPACE(LETTERBREEDTE÷2 -1);
                        HORIZONTAAL(R,LETTERBREEDTE - LETTERBREEDTE÷ 2)
                END;
        PROCEDURE D(R);
                IF R=1 ∨ R=5  THEN HORIZONTAAL(R,LETTERBREEDTE)
                              ELSE VERTICAAL(R,TRUE);
        PROCEDURE HORIZONTAAL(R,L);
                BEGIN   INTEGER P;
                        FOR P:=1 STEP 1 UNTIL L DO PRSYM(LETTER[R]);
                        SPACE(LETTERBREEDTE)
                END;
        PROCEDURE VERTICAAL(R,TWEE); BOOLEAN TWEE;
                BEGIN   INTEGER P;
                        PRSYM(LETTER[R]); SPACE(LETTERBREEDTE -2);
                        IF TWEE THEN BSSYM(LETTER[R]) ELSE BSSYM(93);
                        SPACE(LETTERBREEDTE)
                END;

        FOR I:= 1 STEP 1 UNTIL 5 DO LETTER[I] := RESYM;


VOLGENDE TITELBLAD:
        BEGINREGEL:=READ;
        BEGINPOSITIE:=READ;
        LETTERBREEDTE:=READ;
        LIJNDIKTE:=READ;
        FOR I:= 1 STEP 1 UNTIL BEGINREGEL DO NEGR;
        FOR REGEL:= 1 STEP 1 UNTIL 5 DO
        FOR DIKTE := 1 STEP 1 UNTIL LIJNDIKTE DO
        BEGIN   NEGR; SPACE(BEGINPOSITIE);
                A(REGEL); B(REGEL); C(REGEL); D(REGEL); B(REGEL)
        END;
        IF READ ≠  0 THEN BEGIN NEW PAGE; GOTO VOLGENDE TITELBLAD END
END
```

2. call by value of call by name?, Jensen's device, function designator

In het voorafgaande hebben we alleen "name"-parameters gebruikt. ALGOL 60 kent echter ook de zgn. call by value. Is een parameter in de value-lijst opgenomen dan is het een value-parameter, anders is het een name-parameter. Zoals in VI-1 uiteengezet wordt een formele parameter, called by name, bij aanroep van de procedure overal in de body door de corresponderende actuele vervangen. Formele parameters, called by value, zijn identifiers die evenmin als "name"-parameters gedeclareerd worden. Men moet zich voorstellen dat ze bij elke aanroep van de procedure in een fictief block, dat de procedure body omvat, worden gedeclareerd met het type als gegeven in de specificatie en vervolgens geïnitialiseerd op de waarden, die de actuele parameters afleveren.
Strings, non-type procedures en procedures met parameters kunnen uiteraard slechts als name-parameter fungeren, omdat bovenstaand proces er niet op toegepast kan worden.
Anders verdient in het algemeen een value-parameter de voorkeur boven een name-parameter. Een reden om een parameter niet in de value-lijst op te nemen kan zijn:

a. tijdverlies (bij arrays);
b. de parameter moet een resultaat naar buiten afleveren (de zgn. output-parameters);
c. men wil Jensen's device toepassen (zie eind van deze sectie).

Voordelen van het opnemen in de value-lijst zijn:

a. Een call by value kan dikwijls efficiënter afgehandeld worden dan een name-replacement; voorbeelden hiervan zijn:
   1. een formele wordt niet onnodig vaak doorgegeven;
   2. een ingewikkelde expressie wordt slechts één keer uitgerekend;
   3. een expressie wordt niet steeds opnieuw uitgerekend.
b. Men benadrukt in de tekst van het programma dat de uitvoering van de procedure (bibliotheekprocedures) de waarde van de actuele parameter niet aantast.
c. Een variabele die alleen binnen de procedure een rol speelt, kan dan op verschillende waarden geïnitialiseerd worden.

```
begin comment 1739, N. van de Auteur (MC), programma voor het zoeken
      van een nulpunt van z↑3+3×z↑2+z+5 en van -z↑3+3×z↑2+z+5;
      real z;
      real procedure nulpunt (x, fx, xvanfpos, xvanfneg);
      value xvanfpos, xvanfneg; real x, fx, xvanfpos, xvanfneg;
      begin   for x:= (xvanfpos + xvanfneg)/2
              while abs(xvanfpos - xvanfneg) ≥ 10-8 do
              if fx > 0 then xvanfpos:= x
              else
              if fx < 0 then xvanfneg:= x
              else goto klaar;
      klaar:  nulpunt:= (xvanfpos + xvanfneg)/2
      end;
      NLCR;
      PRINTTEXT(‹een oplossing van z↑3+3×z↑2+z+5=0 is z =›);
      print(nulpunt(z, ((z+3)×z+1)×z+5,0,-5));
      NLCR;
      PRINTTEXT(‹een oplossing van -z↑3+3×z↑2+z+5 = 0 is z =›);
      print(nulpunt(z, ((-z+3)×z+1)×z+5, 0, 5))
end
```

We gaan nu over tot de behandeling van bovenstaand programma, dat een nulpunt zoekt van $f(z) = z^3 + 3z^2 + z + 5$ en van $f(z) = - z^3 + 3z^2 + z + 5$.

De algorithme *) - beschreven in "nulpunt" -, volgens hetwelk het nulpunt bepaald wordt, is:

1. Ga uit van 2 punten: xvanfpos dat is een argumentwaarde, waarvoor de funktie positief is en xvanfneg dat is een argumentwaarde, waarvoor de funktie negatief is.

2. Bepaal het midden x van xvanfpos en xvanfneg.

---

*) Opgemerkt zij dat deze procedure verre van efficiënt is en ook niet betrouwbaar. Aan deze eisen voldoet b.v. wel de procedure ZEROIN, uit MCA 2310 ALGOL 60 PROCEDURES IN NUMERICAL ALGEBRA, T.J. Dekker en W. Hoffmann, Mathematical Centre Tracts nr. 23, Mathematisch Centrum Amsterdam 1968.

3. Is f(x) = 0 dan is het proces afgelopen en is x een nulpunt, is f(x) > 0 dan xvanfpos:= x en anders xvanfneg:= x.
4. Herhaal 2 en 3 zo vaak tot xvanfpos en xvanfneg minder dan 10^{-8} verschillen of tot de funktie in het midden nul is.

Men heeft dan met een fout van 10^{-8} een nulpunt gevonden van f(z). Deze algorithme werkt alleen voor funkties f(z), die de eigenschap hebben dat tussen een z-waarde, waar f(z) positief is en een z-waarde, waar f(z) negatief is, een nulpunt van f(z) ligt. Dit geldt met name voor continue funkties als $z^3 + 3z^2 + z + 5$.

De procedure nulpunt staat op regel 4 t/m 13 en heeft 2 value- en 2 name-parameters. Tijdens de executie wordt "nulpunt" voor het eerst aangeroepen op regel 16 in de procedure statement:

```
print(nulpunt(z,((z + 3) × z + 1) × z + 5 ,0,-5))
```

De actuele parameter van print wordt gevormd door een expressie, die bestaat uit één function designator. De declaratie van deze function designator geeft ons het recept hoe de waarde die afgeleverd wordt, bepaald wordt. Moest men zich bij een procedure statement (denk aan A(regel) uit VI-1) voorstellen dat eerst het programma gewijzigd werd en vervolgens doorgegaan werd in het gewijzigde programma, bij een function designator moet men zich voorstellen dat de rekenautomaat even "op een kladje" deze waarde berekent met de kennis van het programma die hij op dat moment heeft. De waarde, die "nulpunt" aflevert, is de waarde als laatste geassigneerd aan "nulpunt" bij de uitvoering van onderstaand statement.

```
 1  begin real xvanfpos,xvanfneg;
 2
 3        xvanfpos:= 0;
 4        xvanfneg:= -5;
 5        begin  for   z:= (xvanfpos + xvanfneg)/2
 6               while abs(xvanfpos - xvanfneg) ≥ 10-8 do
 7               if (((z + 3) × z + 1) × z + 5) > 0 then xvanfpos:= (z)
 8               else
 9               if (((z + 3) × z + 1) × z + 5) < 0 then xvanfneg:= (z)
10               else goto klaar;
11        klaar: nulpunt:= (xvanfpos + xvanfneg)/2
12        end
13  end
14
```

We zien dat regel 5 t/m 12 van dit statement wel gelijkenis vertoont met regel 6 t/m 13 uit het oorspronkelijke programma. Bij de "name"-parameters is alleen de formele parameter (na omsluiting met haakjes zie VI-3) vervangen door de corresponderende actuele parameter. Is de ene actuele parameter een funktie van de andere dan spreken we van Jensen's device. We merken nog op dat de drager z alleen in "nulpunt" gebruikt wordt. (Jensen's device eist een buiten de procedure gedeclareerde variabele.)

3. overzicht actuele parameters, vervanging formele parameter door actuele parameter

Een actuele parameter (Revised Report 4.7.1) kan zijn:

1. een string
2. een expressie, d.w.z. een Boolean of een arithmetische of designational expressie (zie hoofdstuk IV)
3. een array identifier
4. een switch identifier
5. een procedure identifier

Wat betreft value parameters merken we op:

a. Strings en switch identifiers kunnen niet optreden als value parameters.
b. Een procedure identifier kan alleen value zijn, als hij op zichzelf een complete expressie vormt (b.v. read).
c. Wanneer een expressie optreedt als value parameter moet het type van de waarde, die hij aflevert in overeenstemming te brengen zijn met zijn specificatie (zonodig wordt een real afgerond tot een integer).
d. Bij een array identifier aangeboden als value-parameter moeten we ons voorstellen dat:
   1. in het fictieve block om de procedure body een array gedeclareerd wordt met dezelfde dimensie en grenzen als het aangeboden array maar van het type als gegeven in de specificatie;
   2. iedere subscripted variable van dit array geïnitialiseerd wordt op de waarde (zonodig afgerond) van de subscripted variable met dezelfde indices van het aangeboden array.

Hebben we te maken met een "name"-parameter, dan moet de vervanging correct ALGOL 60 opleveren. Consequenties hiervan zijn o.a.:
a. Wordt er aan de parameter geassigneerd dan moet de expressie uit één variabele bestaan.
b. Een string wordt doorgegeven aan een volgende procedure of aan een procedure, waarvan de body niet in ALGOL 60 geschreven is (b.v. PRINTTEXT).

4. <u>bij het formeel-actueel vervangingsmechanisme gaat het natuurlijke boven het letterlijke, omsluiting met haakjes, conflicten tussen identifiers</u>

Over het formeel-actueel vervangingsmechanisme merken we op:

a. Bij het proces van de vervanging moeten we ons voorstellen (Revised Report 4.7.3.2) dat de rekenautomaat de formele identifier, waar mogelijk eerst tussen haakjes zet. Is f een formele identifier in $3 \times f + 2$ en moet f vervangen worden door $x + 4$ dan wordt het resultaat $3 \times (x + 4) + 2$ {en niet $3 \times x + 4 + 2$}. De vervanging is hierdoor meer natuurlijk dan letterlijk.

b. In het programma op blz. 64 zien we dat de identifier a zowel op regel 4 als op regel 28 gedeclareerd wordt; drukletter(b) moet nu tijdens de executie vervangen worden door regel 7 t/m 13. Is de a, die dan gebruikt wordt, gehecht aan de quantity geïntroduceerd op regel 4 of aan de quantity geïntroduceerd op regel 28?

We kunnen de situatie als volgt weergeven:

```
 1  begin integer a; andere declaraties;
 2  procedure identifier (formelen);
 3        begin localen
 4              en niet localen bv. gebruikt in een statement
 .              als; a:= 3;
 .         .
 .         .
          end;
                .
                .
10            begin <declaraties>; integer a;
11                        .
                          .
12                          begin integer a;
 .                                  call (actuelen)
 .                          end;
 .
 .            end;
 .  end
```

De verzameling van de identifiers van de actuelen duiden we aan met I.
We moeten ons nu voorstellen dat:

1. de rekenautomaat tijdens de executie de identifiers van alle formele parameters en locale quantities indien nodig zo verandert dat er géén identifier van een formele of locale in I voorkomt (Revised Report 4.7.3.2);
2. als de call "outside the scope" van één der niet localen staat (b.v. a regel 1) de rekenautomaat aan de andere quantity(ties) (a regel 10 en a regel 12) die met dezelfde identifier worden aangeduid, één nieuwe identifier geeft die nog niet in het programma voorkomt (Revised Report 4.7.3.3).

Dit houdt voor het programma op blz.64 enerzijds ten gevolge van punt 1 in:
Is de procedure statement "drukletter(b)" één maal uitgevoerd voor b = 1 dan is na de executie de actuele parameter "b" nog steeds 1 en anderzijds tengevolge van punt 2:
De "a", die op regel 8 gebruikt wordt, is gehecht aan de quantity geïntroduceerd op regel 4.

Na het programma is direct de getallenband afgedrukt. We zien dat de getallenband ingewikkelder is geworden en het programma korter. Een bewering uitspreken over de output van het programma zoals in VI-1, die alleen aan de hand van de tekst te verifiëren is, kan nu niet meer. Een kort programma is dus geen doel op zichzelf, want ook het programma

```
begin L: PRSYM(RESYM); goto L end
```

levert bij passende getallenband het gevraagde titelblad. Deze getallenband is nu waarschijnlijk zo ingewikkeld geworden, dat er een programma voor nodig is om hem te ponsen.

```
begin   comment   1739 K.K. Koksma, ander programma
        voor afdrukken titelblad;

        integer beginregel, beginpositie, lijndikte, a,b, c;
        integer array letter[1:5], layout[1:5, 1:5, 1:3];
        procedure drukletter(r);
        begin   integer b;
                PRSYM(letter[a]);
                SPACE(layout[a, r, 1]);
                for b:= 1 step 1 until layout[a, r, 2] do
                PRSYM(letter[a]);
                SPACE(layout[a, r, 3])
        end;

        for a:= 1 step 1 until 5 do letter[a]:= RESYM;

volgende titelblad:
        beginregel:= read;
        beginpositie:= read;
        lijndikte:= read;
        for a:= 1 step 1 until 5 do
        for b:= 1 step 1 until 5 do
        for c:= 1 step 1 until 3 do layout[a, b, c]:= read;

        for a:= 1 step 1 until beginregel do NLCR;

        for a:= 1 step 1 until 5 do
        begin   integer a;
                for a:= 1 step 1 until lijndikte do
                begin   NLCR; SPACE(beginpositie);
                        for b:= 1 step 1 until 5 do
                        drukletter(b)
                end
        end;
        if read ≠ 0 then
        begin   NEWPAGE; goto volgende titelblad end
end
algol
5,20,1
0,5,6   5,0,6   0,5,6   0,5,6   0,0,0
4,1,6   5,0,6   4,1,6   4,1,6   0,0,0
4,1,6   5,0,6   5,0,6   4,1,6   0,0,0
0,5,6   5,0,6   2,3,6   4,1,6   0,0,0
4,1,6   0,5,6   0,5,6   0,5,6   0,5,6

1

5,20,3

0,7,8   7,0,8   0,7,8   0,7,8   0,0,0
6,1,8   7,0,8   6,1,8   6,1,8   0,0,0
6,1,8   7,0,8   7,0,8   6,1,8   0,0,0
0,7,8   7,0,8   4,6,5   6,1,8   0,0,0
6,1,8   0,7,8   0,7,8   0,7,8   0,7,0

0
```

5. principe van de volledige inductie, recursieve procedures

Voor een goed begrip van de werking van recursieve procedures is de kennis van het principe van de volledige inductie vereist.

Het principe van de volledige inductie zegt:

Een uitspraak E(n) afhangend van een natuurlijk getal n is geldig voor iedere n als:

1) hij geldig is voor n = 1

2) uit de geldigheid van E(n-1) de geldigheid van E(n) volgt.

Een voorbeeld hiervan is de stelling:

De uitspraak E(n): $\sum_{k=1}^{n} k = \frac{1}{2} n(n + 1)$ geldt voor iedere $n \geq 1$.

bewijs:

1) E(1) is waar want

$$\sum_{k=1}^{1} k = 1 \quad \text{en} \quad \frac{1}{2} \cdot (1 + 1) = 1.$$

2) Stel E(n - 1) is waar $\therefore \sum_{k=1}^{n-1} k = \frac{1}{2} (n - 1)n.$

Tel nu links en rechts n erbij: $\quad n = \quad n$

En dan krijgen we E(n) $\Leftarrow \quad \sum_{k=1}^{n} k = n\{\frac{1}{2} (n-1)+1\} = \frac{1}{2} n(n+1)$

Het volgende programma geeft een eenvoudig voorbeeld van een recursieve procedure:

```
begin comment 1739, N. van de Auteur (MC), berekening van nfaculteit;
      integer a, resultaat;
      integer procedure nfaculteit(r); value r; integer r;
              nfaculteit:= if r = 1 then 1 else r × nfaculteit(r-1);

      a:= read;
      resultaat:= nfaculteit(a);
      FIXT(6,0,resultaat)
end
```

Hier wordt een procedure nfaculteit gedeclareerd waarvan de naam erop wijst dat hier n! (spreek uit: enfaculteit, dit is het gedurig product van alle gehele getallen $\geq$ 1 en $\leq$ n, waarbij 1! = 1) wordt uitgerekend.

Hoe kunnen we nu aan de tekst van het programma zien dat nfaculteit dit ook inderdaad doet? Dit kunnen we inzien met behulp van volledige inductie, we kunnen nl. de volgende uitspraak E(n) doen:

Levert de actuele parameter van nfaculteit de waarde n af, dan levert nfaculteit de waarde n! af.

<u>bewijs</u>: 1. E(1) geldt, want

wordt een actuele parameter met de waarde 1 aangeboden dan wordt de waarde, die nfaculteit aflevert, bepaald in onderstaand statement:

```
begin integer r;
      r:= 1;
      nfaculteit:= if r = 1 then 1 else r × nfaculteit(r-1)
end
```

De waarde die nfaculteit aflevert, wordt dus 1 = 1!

2. Stel E(n-1) geldt, nu te bewijzen E(n) geldt.

Wanneer "a" de waarde n draagt (n geheel en groter 1) dan wordt nfaculteit(a) berekend in de statement:

```
begin integer r;
      r:= a;
      nfaculteit:= if r = 1 then 1 else r × nfaculteit(r-1)
end
```

De relatie r = 1 levert de waarde <u>false</u> af.

∴ De waarde die nfaculteit(a) aflevert, is: r × nfaculteit(r-1).

Nu draagt r de waarde n.

∴ De expressie r-1 levert de waarde n-1 af.

∴ nfaculteit(r-1) levert de waarde (n-1)! af (gegeven).

∴ nfaculteit(a) levert de waarde n! af.

Opgemerkt zij dat dit programma alleen diende als eenvoudig voorbeeld voor recursief gebruik van een procedure. Het programma is nl. zeer tijdrovend en behalve dat zal de integercapaciteit vrij gauw overschreden zijn. Heeft men in een programma op meerdere plaatsen n! nodig dan doet men er goed aan één keer deze waarden uit te rekenen en op te bergen in een array, dat men later kan raadplegen.

We gaan nu een ingewikkelder probleem recursief programmeren en wel het bepalen van een zet in het spel Boter, kaas en eieren. Het spel wordt gespeeld op een bord met 9 vakken door 2 spelers. In het begin zijn alle vakken leeg. De ene speler mag alleen kruisen zetten de andere alleen cirkels.
Wanneer een speler aan zet is, dan mag hij één teken in een leeg vak zetten. Komen daardoor 3 van zijn tekens op één rij, op één kolom of op één diagonaal te staan, dan heeft hij gewonnen en is het spel uit. Is dit niet het geval, maar is het bord vol dan is het spel ook uit en heeft niemand gewonnen. In alle andere gevallen gaat het spel voort en komt de andere speler aan zet.
De vakken van het bord zullen we nummeren als in onderstaande tekening.

1	2	3
4	5	6
7	8	9

Een vak bezet door een kruis geven we aan met +1, door een cirkel met -1 en een leeg vak met een 0 (zie blz. 68).
Bij de behandeling van het programma gebruiken we de volgende begrippen:

1. Een <u>zet</u> in een spel is het vullen van een leeg hokje met een cirkel of kruis, zolang er nog geen partij gewonnen heeft.
2. Een zet heet <u>winnend</u> als of door de zet een rijtje van drie gecompleteerd wordt, of er minstens één tegenzet is en alle tegenzetten tot verlies leiden.

3. Een zet heet <u>verliezend</u> als er minstens één tegenzet is, die tot winst leidt.

Daar het aantal lege vakken in het spel begrensd is (nl. $\leq$ 9), is het duidelijk dat we dit zowel in de definities als in het programma impliciet zullen terugvinden.

In grote lijnen werkt het programma als volgt:

1. Het leest 9 getallen in, die de stand van het spel bepalen (-1 = cirkel, 1 = kruis, 0 = vrij).
2. Het controleert of dit een stand is, die in het spel kan optreden.
3. Het bepaalt wie aan zet is.
4. Het programma analyseert de stand en wanneer er een zet mogelijk is die winnend of anders niet-verliezend is, dan drukt het die zet af, anders geeft het de partij gewonnen.
5. Staat er nog een stand op de getallenband dan gaat het programma voort met 1 anders is het klaar.

De kern van het programma wordt gevormd door de procedures Winst en Verlies. Waarom werken deze correct?

We gaan de geldigheid van de uitspraak E(n) (met $0 \leq n \leq 9$, n geheel) bewijzen.

E(n) luidt: Wanneer er nog n lege hokjes zijn in "Spel" dan levert enerzijds

a. Winst(n,teken,Spel,p) de waarde <u>true</u> af als de zet "teken" op het (lege) hokje met hokjenummer p wint en anders <u>false</u>

en anderzijds

b. Verlies(n,teken,Spel,p) de waarde <u>true</u> af als de zet "teken" op het (lege) hokje met nummer p verliest en anders <u>false</u>.

<u>Bewijs</u>: We nemen aan dat de function designator "drie(teken,Spel,p)" de waarde <u>true</u> aflevert wanneer we een"drietje" kunnen maken door op hokjenummer p in Spel teken te zetten.

Het eigenlijke bewijs berust op volledige inductie en verloopt als volgt:

1. E(1) geldt want:

   Bij een aanroep van "Winst" met 1 als actuele waarde van "j" zien we op regel 32 dat als drie(teken,Spel,p) true aflevert - d.w.z. dat we direct kunnen winnen door op hokjenummer p "teken" te plaatsen - Winst true wordt en de procedure verlaten wordt. Is dit niet het geval (daar er nog maar één leeg hokje was, kunnen we dus niet winnen) dan wordt de statement op regel 34 t/m 43 uitgevoerd; daar de eerste actuele parameter (aantal lege vakjes) = 1 wordt "Winst" false en wordt de procedure verlaten. Hiermee is E(1)a bewezen.
   Ingevolge de definitie van een verliezende zet kunnen we als er nog maar één hokje is niet verliezen, daar er na onze zet geen tegenzet meer is. Verlies(1,teken,Spel,p) moet dus false afleveren. We zien inderdaad op regel 48, dat als de eerste actuele parameter 1 is "Verlies" false wordt en we de procedure verlaten. Hiermee is E(1)b dus E(1) bewezen.

2. Wanneer E(n-1) geldt, dan geldt E(n) want:

   Als we door op hokjenummer p "teken" te zetten direct een gewonnen spel verkrijgen dan garandeert regel 32 dat "Winst" true wordt en de procedure verlaten wordt. Is dit niet zo, dan wordt daar de eerste actuele parameter $\neq$ 1, het block op regel 35 t/m 43 uitgevoerd. In dit block wordt eerst de zet "teken" op hokjenummer p in "Spel" gedaan. Het array "Spel" is dus nu anders dan het array "Spel" bij binnenkomst.

   Vervolgens wordt uitgevoerd de for statement:

```
for i:= 1 step 1 until 9 do
if zet(i,Spel) then
begin if ¬ Verlies(j-1,-teken,Spel,i)
      then begin Winst:= false; goto end end;
end
```

Hoe moeten we ons dit nu voorstellen? "Spel" is een value array. ∴Na eventuele aanroep van "Verlies" met "Verlies(j-1,-teken,Spel,i)" is "Spel" nog steeds hetzelfde array en alle elementen hebben nog steeds dezelfde waarde. Hetzelfde geldt voor j, i en teken omdat aan geen van drieën noch in de body van "Verlies" noch in de body van "Winst" noch in de body van "drie" geassigneerd wordt. Volgens de inductieaanname geldt E(n-1). ∴ Verlies (j-1,-teken,Spel,i) levert true af als de zet "-teken" op het lege hokje met nummer i verliest en anders false. Dit houdt in dat als er maar één mogelijke tegenzet niet verliest "Winst" false wordt (regel 40) en de procedure verlaten wordt. Indien alle mogelijke tegenzetten verlies opleveren dan wordt de volgende statement (regel 42) uitgevoerd. Dit houdt in dat "Winst" true wordt en de procedure verlaten wordt. Hiermee is E(n)a bewezen. Het bewijs voor E(n)b gaat met een analoge redenering.

De statement op regel 68 t/m 109 kunnen we in grote lijnen als volgt verdelen:

71- 79 leest een stand in en checkt of er wel alleen +1, -1 of 0 als teken wordt aangeboden.
82- 85 checkt of de stand in het spel kan optreden.
92 bepaalt wie aan zet is.
93- 97 gaat na of er een winnende zet is, zo ja ga naar 106.
99-104 gaat na of er een niet-verliezende zet is, zo ja ga naar 106.
105 geeft het spel gewonnen.
106-107 drukt de gevonden zet af.

We merken op dat een winnende zet niet ook zo snel mogelijk winnend behoeft te zijn. Het programma controleert uitgebreid of de databand correct is. Het is nl. zo dat "Winst" en "Verlies" alleen goed werken als ze met de goede actuele parameters worden aangeroepen. Door dit vooraf te checken worden de resultaten van het programma betrouwbaarder. Het zou wellicht aanbeveling verdienen na regel 92 eerst nog het array "Spel" af te laten drukken. Men kan dan controleren of programmeur en programma de getallenband op dezelfde wijze interpreteren. Het programma gedraagt zich op deze manier net zo ten opzichte van de getallenband als het MC-ALGOL 60-

systeem zich gedraagt ten opzichte van het programma. Ieder programma is dus in dit licht bezien als het ware een "compiler" voor zijn getallenband.

Bieden we aan de getallenband:

1	1,	-1,	1,	-1,	1,	-1,	1,	0,	0
2	1,	-1,	-1,	-1,	1,	1,	1,	-1,	0
3	1,	-1,	0,	0,	1,	-1,	-1,	1,	0
4	-1,	1,	1,	1,	-1,	-1,	-1,	1,	0, 0

dan geeft de X8 als output:

1	spel is uit niet gespeeld
2	win door +1 op hokjenummer 9
3	win door +1 op hokjenummer 9
4	verlies niet door +1 op hokjenummer 9

In plaatjes:

X	O	X
O	X	O
X		

NIET GESPEELD

X	O	O
O	X	X
X	O	

WIN DOOR X OP HOKJE 9

X	O	
	X	O
O	X	

WIN DOOR X OP HOKJE 9

O	X	X
X	O	O
O	X	

VERLIES NIET DOOR X OP HOKJE 9

Bij de getallenband:

1	1,	0,	0,	0,	-1,	0,	1,	0,	-1
2	1,	0,	0,	0,	-1,	-1,	1,	0,	-1
3	1,	0,	-1,	0,	-1,	0,	0,	0,	1
4	0,	0,	0,	0,	0,	0,	0,	0,	0,

geeft het programma bij executie op de X8:

1 win door +1 op hokjenummer 2

2 win door +1 op hokjenummer 4

3 win door +1 op hokjenummer 7

4 verlies niet door +1 op hokjenummer 1

X		
	O	
X		O

WIN DOOR
X OP HOKJE 2
Hij wint dus
niet zo snel
mogelijk

X		
	O	O
X		O

WIN DOOR
X OP HOKJE 4

X		O
	O	
		X

WIN DOOR
X OP HOKJE 7

VERLIES NIET
DOOR X OP HOKJE 1

```
BEGIN
COMMENT R 1710. K.K.KOKSMA , VOORBEELD VAN RECURSIEVE
         PROCEDURE BOTER,KAAS EN EIEREN;
INTEGER ARRAY SPEL[1:9];
INTEGER TEKEN, AANTAL LEEG, HOKJENUMMER, CONTROLE,
        TELLER, CHECKER;
BOOLEAN ACCEPTED;
BOOLEAN PROCEDURE DRIE(TEKEN, SPEL, P); INTEGER ARRAY SPEL;
                                INTEGER TEKEN,P;
        BEGIN SWITCH INSPECTEER := 1,2,3,4,5,6,7,8,9;
                BOOLEAN D;
                PROCEDURE F; GOTO END;
                BOOLEAN PROCEDURE G(A,B);
                        G:= SPEL[A] = TEKEN ∧ SPEL[B] = TEKEN;

                GOTO INSPECTEER[P];
            1: D:= G(5,9) ∨ G(2,3) ∨ G(4,7); F;
            2: D:= G(1,3) ∨ G(5,8);F;
            3: D:= G(1,2) ∨ G(7,5) ∨ G(6,9); F;
            4: D:= G(1,7) ∨ G(5,6); F;
            5: D:= G(1,9) ∨ G(3,7) ∨ G(4,6) ∨ G(2,8); F;
            6: D:= G(3,9) ∨ G(4,5); F;
            7: D:= G(5,3) ∨ G(1,4) ∨ G(8,9); F;
            8: D:= G(2,5) ∨ G(7,9); F;
            9: D:= G(1,5) ∨ G(7,8) ∨ G(3,6); F;

          END:  DRIE:=D
        END;

BOOLEAN PROCEDURE WINST(J,TEKEN,SPEL,P); VALUE SPEL;
                        INTEGER ARRAY SPEL;
        BEGIN   IF DRIE(TEKEN, SPEL, P) THEN WINST:= TRUE
                ELSE
                IF J = 1 THEN WINST:= FALSE
                ELSE BEGIN INTEGER I;
                           SPEL[P] := TEKEN;
                           FOR I:= 1 STEP 1 UNTIL 9 DO
                           IF ZET(I,SPEL) THEN
                           BEGIN IF ¬ VERLIES(J-1, -TEKEN, SPEL, I)
                                 THEN BEGIN WINST:= FALSE; GOTO END END;
                           END;
                           WINST:=TRUE
                     END;
           END:
        END;
BOOLEAN PROCEDURE VERLIES(J,TEKEN,SPEL,P); VALUE SPEL;
                        INTEGER ARRAY SPEL;
        BEGIN   IF DRIE(TEKEN, SPEL,P) ∨ J = 1 THEN VERLIES:=FALSE
                ELSE
                BEGIN   INTEGER I;
                        SPEL[P] := TEKEN;
                        FOR I:= 1 STEP 1 UNTIL 9 DO
                        IF ZET(I,SPEL) THEN
                        BEGIN   IF WINST(J-1, -TEKEN,SPEL,I)
                                THEN BEGIN VERLIES:=TRUE; GOTO END END
                        END;
                        VERLIES:=FALSE;
                  END;
                END
```

```
            END;


    BOOLEAN PROCEDURE ZET(I,SPEL);
            ZET:= SPEL[I] = 0;


    FOR TELLER:= READ WHILE TELLER ≠ 0 DO
    BEGIN   NWGR;
            ACCEPTED:= TRUE; CONTROLE:= 0; AANTAL LEEG:= 9;
            ABSFXF(3, 0, TELLER);
            FOR HOKJENUMMER:= 1 STEP 1 UNTIL 9 DO
            BEGIN   TEKEN:= READ;
                    CONTROLE:= CONTROLE + TEKEN;
                    IF TEKEN ≠ 0 THEN AANTAL LEEG:= AANTAL LEEG - 1;
                    IF ABS(TEKEN) = 1 ∨ TEKEN = 0
                    THEN SPEL[HOKJENUMMER]:= TEKEN
                    ELSE
                    BEGIN ACCEPTED:=FALSE; PRINTEXT(‡TEKEN FOUT‡) END
            END;
            IF ABS(CONTROLE) > 1 THEN
            BEGIN ACCEPTED:= FALSE; PRINTEXT(‡TEVEEL TEKENS‡) END;
            FOR CHECKER:= 1, 5, 9 DO
            BEGIN   TEKEN:= SPEL[CHECKER];
                    IF ¬ZET(CHECKER, SPEL) ∧DRIE(TEKEN,SPEL, CHECKER)
                    THEN
                    BEGIN ACCEPTED:= FALSE; PRINTEXT(‡SPEL IS UIT‡) END
            END;
            IF AANTAL LEEG= 0 THEN
            BEGIN ACCEPTED:=FALSE; PRINTEXT(‡ALLE ZETTEN GEDAAN‡) END;
            IF ¬ ACCEPTED THEN
            BEGIN TAB; PRINTEXT(‡NIET GESPEELD‡); GOTO END END;
            TEKEN:= IF CONTROLE ≠ 0 THEN -CONTROLE ELSE 1;
            FOR HOKJENUMMER:= 1 STEP 1 UNTIL 9 DO
            IF ZET(HOKJENUMMER, SPEL)
            THEN BEGIN IF WINST(AANTAL LEEG, TEKEN, SPEL, HOKJENUMMER)
                    THEN BEGIN PRINTEXT(‡WIN DOOR‡); GOTO FINISH END
                END;

            FOR HOKJENUMMER:= 1 STEP 1 UNTIL 9 DO
            IF ZET(HOKJENUMMER, SPEL) THEN
            BEGIN   IF ¬ VERLIES(AANTAL LEEG,TEKEN,SPEL,HOKJENUMMER)
                    THEN
                    BEGIN PRINTEXT(‡VERLIES NIET DOOR‡); GOTO FINISH END
            END;
            PRINTEXT(‡VERLOREN EN GEEF OP‡); GOTO END;
    FINISH: FXF(3, 0, TEKEN); PRINTEXT(‡OP HOKJENUMMER‡);
            ABSFXF(3, 0, HOKJENUMMER);
    END:
    END
    END
```

INDEX

+ zie plus
- zie minus
* zie vermenigvuldiging
/ zie deling
÷ zie heling
<,≤,=,≥,≠ zie relationele operator
≡,¬,∧,∨,⊃ zie logische operator
aanwijzer 10
ABSFIXT 14
actuele parameter 60
ALGOL 60 1
algorithme 1
analyse 2,24
arithmetische expressie 34
array 10,12,20,61
assignment statement 3,7,8,13,37,38
BACKUS' NORMAL FORM 24
basic statement 41,42
basic symbol 28
block 11,19
Boolean expressie 38
Boolean primary 39
call by name 57
call by value 57
commentaar 3
compiler 24,71
compound statement 6
conditional statement 9,43
context condities 28
declaratie 11
declarator 11
deling 35,36,37
derivatieboom 32
designational expressie 40
dimensie 13
directe productie 26
drager 10,32
dummy statement 31
executie 2,24
expressie 34
false 39
flexowriter 2
for list 48
for list element 48
formele parameter 60,62
for statement 6,14,46
function designator 57
getallenband 2
goto statement 12,16
heling 16,35,36,37
identifier 25,27
if statement 7
implementatie 1,4,21
inductie,principe van volledige 65 ev
Input 3,5
Jensens device 57
label 11,15
locaal 11,62
logische operator 39,40
lower bound 13
metalinguïstische formule 25
metalinguïstische variabele 25
minus 35,36,37
MR 81 1
naam 27,28
name parameter 61
NLCR 14
output 3,5
plus 35,36,37
PRINTTEXT 14

INDEX

primary 36
ponsband 2
procedure 10,21
programma 1,2,8
quantity 10,62
read 3,4
recursie 65
relationele operator 39
Revised Report 1,24
scope 19
semantiek 24,32
specificatie 53
statement 41
string 61
subscript expression 13
switch 10,15
syntax 24,26
systeem 1
terminale productie 25,26
true 39
unconditional statement 41,44
upper bound 13
variabele 32
vermenigvuldiging 35,36,37
waarheidstafel 39
while element 14
woord delimiter 5