stichting
mathematisch
centrum

MC

REKENAFDELING CR 13/72 JANUARI

E.W. DIJKSTRA

CURSUS PROGRAMMEREN IN ALGOL 60

13e UITGAVE

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat 49, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

VOORWOORD

Een van de doelstellingen, die de samenstellers van ALGOL 60 voor ogen heeft gezweefd, is geweest, dat ALGOL 60 de enorme mogelijkheden van de moderne automatische rekenmachines binnen het bereik zou brengen van een grote groep potentiele gebruikers, met inbegrip van die gebruikers voor wie een automatische rekenmachine in de eerste plaats een stuk gereedschap is, dat ze zouden kunnen gebruiken om de resultaten te verkrijgen, die hen interesseren. Bij de compositie van ALGOL 60 is er naar gestreefd, "dat dit stuk gereedschap lekker in de hand ligt".

Het is verder de bedoeling, dat de taal ALGOL 60 voor de beschrijving van een berekening gebruikt kan worden, onafhankelijk ervan, welke specifieke machine voor de werkelijke uitvoering ervan gekozen wordt. Om te kunnen garanderen, dat twee verschillende rekenmachines op eenzelfde ALGOL 60 programma voldoende gelijk zouden reageren, was het nodig eerst de taalregels van ALGOL 60 precies vast te leggen. Aldus is geschied en deze taalregels zijn in een ongebruikelijke, maar voor dit doel noodzakelijke ondubbelzinnigheid vastgelegd in het officiele ALGOL 60 rapport "Report on the Algorithmic Language ALGOL 60, by J.W. Backus e.a.". Als leerboek is dit rapport nooit bedoeld en dit heeft vanzelfsprekend tot gevolg, dat voor hem, die voor zijn eerste kennismaking met ALGOL 60 op het officiele rapport is aangewezen, de weg tamelijk moeilijk is. Onnodig moeilijk zelfs.

Om de kennismaking te vergemakkelijken is deze syllabus geschreven. Ten opzichte van het officiele rapport heeft dit dupliceringen met zich meegebracht, zij het in andere bewoordingen; wij stellen er prijs op te verklaren, dat waar onverhoopt de strekking van deze syllabus afwijkt van de bedoelingen van het officiele rapport, het laatste doorslaggevend is.

INLEIDING

ALGOL (een samentrekking van ALGOrithmic Language) is de naam van een taal, die ontworpen is met het doel, rekenprocessen te beschrijven. Deze taal is zo precies gedefinieerd, dat een in ALGOL opgestelde beschrijving van een rekenproces voor een rekenmachine voldoende is, om dit proces ook werkelijk uit te voeren.

Op de Rekenafdeling van het Mathematisch Centrum is voor de X1 een methode ontwikkeld om in ALGOL opgeschreven rekenprocessen - kortweg ALGOL-programmas genoemd - uit te voeren. De verwerking van een ALGOL-programma geschiedt in de volgende fasen.

De tekst van het ALGOL-programma, opgesteld met inachtneming van de regels, die wij hieronder zullen expliceren, wordt uitgetypt op een z.g. Flexowriter. Dit is een electrische schrijfmachine, die alles wat men er op typt tevens vast kan leggen in een zevengats papieren ponsband. De band, die door een Flexowriter geproduceerd wordt, is een medium, dat grote analogie vertoont met de iets smallere vijfgats telexband, die in de automatische telegrafie gebruikt wordt. Omdat de band van de Flexowriter breder is, kunnen we er wat meer symbolen op vastleggen dan in de telexband. Zo kent de Flexowriter elke letter in duplo, nl. eenmaal als hoofdletter en eenmaal als kleine letter, zulks in tegenstelling tot de telex, die maar een enkelvoudig alfabet kent. In het volgende zullen we zien, dat we dit grotere aantal symbolen heel goed zullen kunnen gebruiken. (Bovendien kan de Flexowriter gebruikt worden om de tekst, die eenmaal in een band geponst is, autmatisch nog een keer uit te typen: behalve een ponsstation heeft de Flexowriter nl. ook een leesstation. Een hierin gelegde band kan nog eens uitgetypt worden en desgewenst en passant bovendien weer geponst worden. De Flexowriter is dus ook bruikbaar als bandreproducer.)

De band, die op de Flexowriter geproduceerd is en waarin de tekst van het ALGOL-programma nu is vastgelegd, kan door de X1 zonder meer verwerkt worden. Deze verwerking geschiedt in tweeen. De X1 beschikt over een z.g. "vertaalprogramma" (de MC-vertaler), waardoor de zevengats band met ALGOL-tekst gelezen kan worden. Terwijl deze tekst gelezen wordt, wordt meteen deze ALGOL-beschrijving vertaald in een rekenvoorschrift, dat meer is aangepast aan de eisen, die de X1 met het oog op vlotte uitvoering van het rekenproces mag stellen. Het resultaat van deze vertaalarbeid, dat het z.g. "objectprogramma" genoemd wordt, wordt onder de hand weer uitgeponst. De vertaler produceert uit een ALGOL-programma dus een aequivalent objectprogramma. Als dit gebeurd is, heeft de zevengats band met ALGOL-tekst zijn werk gedaan. Verder werken we met de band, die door de vertaler geproduceerd is.

Als de berekening werkelijk uitgevoerd gaat worden, nemen we de band met het objectprogramma en leggen deze onder de bandlezer. Door een speciaal inleesprogramma wordt deze band gelezen, de erop staande informatie wordt in het geheugen van de X1 opgenomen en de machine is gereed om de gevraagde berekening uit te voeren.

Het voordeel van dit arrangement is, dat de X1 per probleem het werk van de vertaling – en dit werk is aanzienlijk – maar eenmaal hoeft uit te voeren. Andere arrangementen hadden meer machinetijd gevergd en hogere eisen gesteld aan de capaciteit van het geheugen.

Wij zullen ons voorlopig met de verwerking van de ALGOL-band niet bezig houden. Wij zullen eerst uiteenzetten, wat de taal ALGOL inhoudt en hoe men ALGOL kan gebruiken om een proces te beschrijven.

ASSIGNMENT STATEMENT

Laten wij met een heel eenvoudig voorbeeld beginnen. In een bepaald stadium moet een grootheid, zeg f, gevormd worden als de som van twee andere grootheden, zeg a en b. In een "normaal" rekenvoorschrift zou men dan b.v. geschreven hebben:

"Bereken f = a + b ."

In ALGOL, dat zich waar mogelijk bij gangbare notaties aansluit, schrijft men

f:= a + b .

Het hier gebruikte symbool " := " (uit te spreken "wordt") heeft de functie van een z.g. gericht glijkteken. Het betekent, dat het linkerlid gedefinieerd wordt in termen van het rechter, precieser, dat aan de links staande variabele een waarde toegekend wordt, die verkregen wordt door de rechterkant uit te werken. Een formule als boven heet dan ook een "assignment statement". (De term "statement" wordt met betrekking tot ALGOL gereserveerd voor bepaalde types van betrekkelijk afgeronde, een geheeltje vormende handelingen. We zullen de verschillende types gaandeweg tegenkomen. De assignment statement heet zo, omdat hierin aan een variabele een nieuwe waarde wordt toegekend.)

Wil de boven gegeven assignment statement zin hebben, dan is het vereist, dat op het moment, dat hij aan de beurt is, om uitgevoerd te worden, de variabelen a en b inmiddels (in vorige assignment statements) een welgedefinieerde waarde hebben gekregen. Voor de variabele aan de linkerkant hoeft dit niet het geval te zijn. Is de variabele f in de berekening nog niet gebruikt, dan was zijn waarde voor de uitvoering van deze assignment statement ongedefinieerd; erna is zijn waarde gelijk aan de som van de waarden, die a en b op dat moment hebben. De variabele f blijft deze waarde behouden, in principe totdat door een later assignment statement er een nieuwe waarde aan toegekend wordt. Het effect van de assignment statement is dus, dat een eventuele waarde, die de linker variabele van tevoren had, verloren gaat: hij wordt door een nieuwe waarde vervangen.

(De reden, dat het gebruikelijke gelijkteken " = " door het wordtteken " := " is vervangen, is om de nadruk te leggen op de asymmetrie. In het boven gegeven voorbeeld is dat nu niet zo noodzakelijk, want het is duidelijk, dat de rechterkant degene is die uitgerekend moet worden, die dus definieert en dat dus de linkerkant gedefinieerd wordt. Maar in het geval van copiering

f:= a

is het wel gewenst, dat men op de asymmetrie de nadruk legt. De tweede reden is, dat het au fond ook wel iets correcter is, om tot uitdrukking te laten komen, dat het hier gaat om een handeling, die uitgevoerd moet worden en niet, zoals bij het gelijkteken, om een relatie, waaraan al of niet voldaan kan zijn. Willen wij in een zeker stadium van de berekening een of ander tussenresultaat, zeg e, van teken wisselen, dan doen we dit met de assignment statement

e:= - e ;

hadden we hier in plaats van het wordtteken het gelijkteken gebruikt, dan had er een vergelijking gestaan met als enige wortel e = 0, d.w.z. net de waarde, waarvoor tekenwisseling een zinloze operatie is.)

Een speciale vorm van assignment statement is de z.g. herhaalde assignment, waarbij de waarde van een expressie aan een aantal variabelen wordt toegekend, b.v.

" x:= y:= z:= 1 " .

Dit betekent, dat zowel x, als y als z de waarde 1 krijgen.

NAMEN EN GETALLEN

In de bovenstaande voorbeelden hebben wij variabelen aangeduid met letters, nl. a, b, e en f. Dit zijn de eenvoudigste voorbeelden van namen (in het engelse rapport "identifiers" genoemd). Wij zullen later zien, dat namen een vitale rol spelen, dat we ze niet alleen gebruiken om variabelen mee aan te duiden, maar soms ook groepen van variabelen, punten in het programma of zelfs hele processen. Het is daarom vereist, dat we eerst vertellen, wat voor structuur de namen hebben, die we in ALGOL mogen gebruiken.

Voor de opbouw van een naam hebben we de keuze uit 62 symbolen, te weten de 26 kleine letters a t/m z, de 26 hoofdletters A t/m Z en tenslotte de 10 cijfers 0 t/m 9. Een naam bestaat uit een willekeurig aantal (minstens 1) van deze symbolen, maar het eerste symbool mag geen cijfer zijn. Toelaatbare namen zijn b.v.

q
eps
Y3
Hoge druk
SPANNING
RK1ST

Verboden als naam zijn b.v.

K-punt
5de keer
notitie's
art.15.828

De eis, dat een naam niet met een cijfer begint, maakt het onmogelijk, een naam uit louter cijfers op te bouwen. Gelukkig. Want anders was het onmogelijk om uit te maken, of met de assignment statement

Rs:= phi - 15

bedoeld wordt, dat phi met het getal vijftien, dan wel met de variabele met de naam "15" verminderd moet worden, om de nieuwe waarde van Rs te krijgen. Nu namen, opgebouwd uit enkel cijfers, niet zijn toegestaan, is het duidelijk, dat hier het getal vijftien bedoeld is.

Een andere wezenlijke afspraak is, dat spaties (en sterker: overgang op een nieuwe regel) geen informatie dragen. De genoemde naam "Hoge druk" mag dus ook als "Hogedruk" of desnoods "Hog edr uk" geschreven worden, in ALGOL blijft het een opeenvolging van dezelfde karakters, en daarmee dezelfde naam.

Wij willen er tevens op wijzen, dat in weerwil van menigeens handschrift en een groot aantal schrijfmachines, de hoofdletter O en het cijfer 0 verschillende symbolen zijn. Wie hiertussen onvoldoende onderscheid kan maken, doet er goed aan, in de namen, die hij naar eigen goeddunken kiezen mag, het gebruik van de hoofdletter O te vermijden. Hetzelfde geldt voor de hoofdletter I, de kleine letter l en het cijfer 1.

In uitdrukkingen mag (we hebben dit in het laatste voorbeeld al gezien) op de plaats van (de naam van) een variabele ook een getal voorkomen. De getallen worden normaal in het tientallig stelsel genoteerd, maar de preciese vorm is ook hier gereglementeerd.

Een ongetekend getal bestaat uit een numeriek gedeelte, eventueel gevolgd door een macht van tien. Het numeriek gedeelte is of een geheel getal, bestaande uit een aantal (minstens 1) decimale cijfers, of een niet-geheel getal, bestaande uit een aantal decimale cijfers (minstens 1) waarvan een voorafgegaan wordt door de decimale punt".". (Hier wordt dus de engelse conventie gevolgd van "decimal point" en niet de "decimale komma" zoals o.a. in Nederland gebruikelijk is.) Desgewenst kan men het numeriek gedeelte laten volgen door een gehele macht van tien; deze bestaat uit een gehele exponent, voorafgegaan door het speciale symbool"$_{10}$" (lees "maal tien tot de macht"). Een extra regel is, dat in het geval van een expliciete decimale exponent een numeriek gedeelte = 1 weggelaten mag worden. Voorbeelden van getallen zijn:

0	200.084	3.1415 2653 7
177	+07.43$_{10}$8	36 61 53
.5384	9.34$_{10}$+10	$_{10}$-4
-0.7300	2$_{10}$-4	-$_{10}$5

NB. In weerwil van de gangbare uitspraak "maal tien tot de macht" late men zich niet verleiden het symbool "$_{10}$" als operatorteken op te vatten. Het is slechts een notatiewijze voor de plaats van de komma in een getal en vermeende uitdrukking als

"ALPHA$_{10}$5", "-7$_{10}$delta" of "a$_{10}$b"

zijn dus niet toegestaan.

UITDRUKKINGEN

Met variabelen en getallen kunnen we allerlei expressies aan de rechterkant van onze assignment statements opbouwen. We hebben hier in eerste instantie de beschikking over de normale vier algebraische operaties: optellen, aftrekken, vermenigvuldigen, delen. Zij worden aangeduid met de symbolen "+", "−", "×" (of een andere weergave van het maalteken, duidelijk onderscheidbaar van de letter x; b.v. het sterretje voldoet hieraan, het lijkt verstandig zich in manuscript dit symbool aan te wennen) en "/".

Het gebruik van deze operatietekens is aan enkele regels gebonden; de belangrijkste is wel, dat men het maalteken nooit mag weglaten. De reden hiervoor is duidelijk. Als men de assignment statement

x:= a × b

zou mogen weergeven met x:= ab

dan zou in de taal ALGOL een dubbelzinnigheid geintroduceerd zijn. Immers, is de nieuwe waarde van x dan gedefinieerd als het product van a en b, of als de waarde van de variabele met de naam "ab", die ook in de berekening voor mag komen?

De verplichting, de operatietekens altijd te vermelden, heeft tot gevolg, dat de symbolen, waaruit de naam of het getal zijn opgebouwd, altijd zijn ingekapseld tussen twee symbolen die niet tot de naam of het getal kunnen behoren. Dit maakt het voor de vertaler mogelijk om eenduidig vast te stellen, waar de naam begint en waar deze eindigt. Dit is dan ook de reden, waarom in plaats van het maalteken niet de punt is toegestaan. We kunnen dan nl. geen onderscheid meer maken tussen het getal 15.5 (=31/2) en het product 15×5 (=75). Wanneer wij dus beweren, dat in ALGOL arithmetische expressies volgens de normale algebraische notatie opgeschreven worden, dan bedoelen we daarmee "met uitsluiting van de normale dubbelzinnigheden, die men zich tegenover een menselijke lezer wel, maar tegenover een machine niet kan veroorloven." Immers, geen physicus zal in de relatie "PV=RT" niet de wet van Boyle-Gay Lussac herkennen; van een machine mag je niet verwachten, dat hij uit deze symbolen destilleert, dat het hier gaat om een gelijkheid van producten.

NB. Uit het bovenstaande volgt, dat het maalteken dus ook verplicht is, als de eerste factor een getal is. Het nde oneven natuurlijke getal wordt in ALGOL gegeven door "2 × n − 1" en niet door het zoveel gewonere "2n−1".

Een afspraak, waarvan de achtergrond veel minder fundamenteel is, betreft de prioriteit. Vermenigvuldiging en deling hebben prioriteit boven optelling en aftrekking; verder worden de operaties van links naar rechts uitgevoerd.

Voorbeelden:

eps:= C + BIR × BAR	betekent	eps:= C + (BIR × BAR)
Sg:= x / 1 − x	betekent	Sg:= (x / 1) − x
prod:= a / b × c	betekent	prod:= (a / b) × c

De vermenigvuldiging heeft dus geen prioriteit boven de deling.

Prioriteitsregels als boven zijn alleen maar afspraken, om het nodige aantal haakjes wat te verminderen, maar soms komen we er natuurlijk niet onder uit. (Haakjes, die dankzij de prioriteitsregels overbodig zijn, mogen weggelaten worden, maar hoeven dat niet.) Normaal is de assignment statement

$$y:=(1 + x)/(1 - x)$$

maar als je daar nou lust in hebt, mag je dus ook schrijven

$$y:=(((1)+ x))/((1 - ((x))))\quad .$$

Ook wanneer haakjesparen elkaar omvatten, blijven we gewone ronde haakjes gebruiken en gaan niet over op accolades, vierkante haken, grotere haken etc.

Algebraische haakjes worden gewoonlijk genoemd in verband met de prioriteit: men leert het uitwerkvoorschrift "binnenste haakjes eerst uitwerken". Bij het product in

$$x:=(a + b)\times(c + d)$$

moet men eerst de sommen $a + b$ en $c + d$ gevormd hebben, voordat er vermenigvuldigd kan worden.

Met het oog op latere toepassingen lijkt het niet ondienstig om het effect van haakjes nog op een beetje andere manier te belichten. Het voorschrift "binnenste haakjes eerst uitwerken" leidt natuurlijk wel tot het correcte antwoord, maar het geeft eigenlijk slechts het (arithmetisch) gevolg van de betekenis van haakjes en niet de betekenis zelf. De wezenlijke functie van algebraische haakjes is datgene, wat er door omvat wordt, hoe ingewikkeld ook, in te kapselen, van de buitenwereld af te schermen, zodat het van buitenaf beschouwd kan worden als een doodgewone variabele, waar niets bijzonders mee aan de hand is. Wanneer we zeggen, dat in de boven gegeven assignment statement de grootheid x uitgerekend moet worden als het product van twee sommen, dan is dat correct, maar dan hebben we al verder gekeken, dan onze neus lang is: in eerste instantie is x gegeven als product en hoe we aan de factoren komen, is (bij gratie van de haakjes) van later zorg. Er had bij wijze van spreken ook

$$x:= v1 \times v2$$

kunnen staan.

We komen dit facet meteen tegen bij de volgende arithmetische operatie, de machtsverheffing. Tot nog toe was het gebruikelijk, om een exponent wat hoger dan het grondtal te schrijven. Maar net zo goed als spatie en nieuwe regel in ALGOL geen informatie overdragen, evenmin is het aantrekkelijk om er betekenis aan te moeten hechten "wanneer de symbolen wat hobbelig op de regel staan". Om alle misverstand te voorkomen is een nieuwe operator ingevoerd, nl. "↑" (lees: "tot de macht"), dus "a ↑ b" betekent "a tot de macht b".

Machtsverheffing heeft grotere prioriteit dan vermenigvuldiging en deling. Voorts lette men erop, dat zowel als grondtal als als exponent gewone variabelen en ongetekende getallen mogen optreden, maar dat men haakjes gebruiken moet, zodra het ingewikkelder wordt. Dus

c:= a ↑ 2 + b ↑ 2 in plaats van $c := a^2 + b^2$.

De klassieke conventie van wat hoger schrijven, maakt duidelijk, waar de exponent eindigt; dit einde is hier bepaald door de eis, dat de exponent tussen haakjes moet staan, zodra het meer is dan een variabele of een ongetekend getal. Zo wordt

$$a^{2r+1}$$

in ALGOL

$$a \uparrow (2 \times r) + 1 \quad .$$

Evenzo wordt x^{-2} in ALGOL $x \uparrow (-2)$.

Hadden we in de laatste twee voorbeelden de haakjes om de exponent weggelaten, dan zouden we in het tweede geval een incorrecte ALGOL-formule gekregen hebben, en in het eerste geval zou de uitdrukking uitgelegd zijn als

$$(a \uparrow 2) \times r + 1 \quad \text{betekent} \quad (a^2) \times r + 1 \quad .$$

De moraal hiervan is natuurlijk, dat men er verstandig aan doet, om overal, waar men niet apert zeker ervan is, hoe de prioriteitsregels een bepaalde expressie uitleggen, deze twijfel door haakjes op te heffen. Men vermindert hiermee de kans op fouten, men verhoogt de leesbaarheid van zijn ALGOL-programma voor hen, die niet zo in de finesses doorkneed zijn en tenslotte: het kan nooit kwaad.

OPEENVOLGING VAN STATEMENTS

Tot nog toe hebben we ons beperkt tot voorbeeldjes, bestaande uit een assignment statement. Men heeft echter al heel gauw meer dan een assignment statement nodig om een deelberekeningetje te formuleren. Stel, dat we het complexe getal $z = x + yi$ vermenigvuldigen willen met $0.6 + 0.8i$ en dit product weer z willen noemen. Nu kent ALGOL niet de mogelijkheid, om in expressies complexe getallen met een identifier aan te duiden, m.a.w. we zullen deze complexe vermenigvuldiging in termen van reeel en imaginair deel moeten formuleren. Een kleine complicatie is, dat als we eerst het nieuwe reele deel uitrekenen, we het oude reele deel nog nodig hebben, om het nieuwe imaginaire deel te vormen; eerst het nieuwe imaginaire deel bepalen confronteert ons mutatis mutandis met dezelfde moeilijkheid. De oplossing wordt gegeven door de introductie van een hulpgrootheid. De vermenigvuldiging kunnen we in ALGOL programmeren met b.v. de volgende drie statements.

```
u:= 0.6 × x - 0.8 × y;
y:= 0.8 × x + 0.6 × y;
x:= u
```

De hierin beschreven arithmetiek spreekt voor zich zelf. Nieuw is het optreden van de puntkomma ";" tussen de statements. De puntkomma heet in ALGOL de "statement separator": alle op elkaar volgende statements moeten door het symbool ";" van elkaar gescheiden worden.

Het is duidelijk, dat in het algemeen een dergelijk scheidingssymbool noodzakelijk is. Immers, de ene statement kan met een naam eindigen, de volgende statement kan met een naam beginnen. Als deze statements zonder scheidingssymbool botweg op elkaar mochten volgen, zou de vertaler in het algemeen niet kunnen uitmaken, waar de laatste naam van het voorste statement eindigde en de eerste naam van de volgende statement begon. Onze boven gedane bewering "dat elke naam altijd is ingekapseld tussen twee symbolen, die niet tot de naam kunnen behoren", impliceert een dergelijke conventie. Met de introductie van de puntkomma is (op een niveau een hoger) voor de statements hetzelfde bereikt: elke statement is ingekapseld tussen symbolen, die niet tot de statement kunnen behoren.

STATEMENTHAKEN

Zoals in een statement de namen door operatoren gescheiden worden, zo worden in een stuk ALGOL-tekst de opeenvolgende statements door de separator ";" gescheiden. (Zo je wilt door de operator ";" met de betekenis "en ga over tot de nu volgende statement".) Maar de analogie gaat verder: we hebben gezien, dat in een expressie een aantal namen, door operatoren gescheiden, door algebraische haakjes omvat kunnen worden, die deze deelexpressie tot een geheel verklaren. Op precies dezelfde manier bestaan er "op statementniveau" twee haakjes, nl. het symbool "begin" als statementopeningshaak en het symbool "end" als statementsluitingshaak.

Voordat we de functie van de statementhaken begin en end nader uiteenzetten, moeten wij de rol van de onderstreping expliceren. Wij hebben inmiddels kennisgemaakt met ruim 70 z.g. basissymbolen van ALGOL, nl. de 26 kleine letters, de 26 hoofdletters, de 10 cijfers, de operatoren := + - × / en ↑, de haakjes (en) en de separatoren ; , . . ALGOL heeft behoefte aan een aanzienlijk groter aantal basissymbolen en om typografische moeilijkheden, die daardoor zouden kunnen ontstaan, eens en vooral op te lossen, is er een mechanisme geschapen, om nieuwe symbolen te creeren. Dit mechanisme bestaat uit de onderstreping. Als wij schrijven begin, dan geven wij daarmee een specifiek ALGOL-symbool aan, dat niets te maken heeft met de vijf letters b, e, g, i en n. De conventie van de onderstreping promoveert deze configuratie tot een nieuw basissymbool. Op deze manier zijn er ruim 20 nieuwe basissymbolen geintroduceerd. (Men lette erop, dat de conventie van onderstreping voor het wezen van de taal niet essentieel is: het is een afspraak, die gemaakt is om een uniforme representatie van ALGOL-teksten op papier te verkrijgen, maar elke andere conventie van dezelfde strekking kan, mits het er duidelijk bij gezegd wordt, voor dit doel dienen. Zo is het in gedrukte teksten inmiddels niet ongebruikelijk, om in plaats van te onderstrepen de gebruikelijke lettercombinaties in een dikker lettertype te zetten.)

Het eerste gebruik van de statementhaken begin en end is om aan te geven, waar ons ALGOL-programma begint en waar de tekst weer eindigt. De conventie is, dat wij ons ALGOL-programma beginnen met het symbool begin en afsluiten met het symbool end. De statementhaken worden uitsluitend keurig "genest" gebruikt, d.w.z. elke openingshaak wordt later gevolgd door een bijbehorende sluitingshaak en bij welke openingshaak een sluitingshaak hoort, volgt uit het feit, dat haakjesparen elkaar mogen omvatten. Als de vertaler de tekst van begin tot eind leest, kan hij bij elke sluitingshaak vaststellen, bij welke openingshaak deze hoort: vindt de vertaler de sluitingshaak end, die toegevoegd is aan de allereerste begin, dan is daarmee tevens bekend, dat het gehele ALGOL-programma gelezen is.

(Voor een goed verstaander is het noodzakelijk, dat hij vast kan stellen, wanneer de spreker uitgesproken is. Men denke b.v. aan het radioverkeer tussen schepen, waar het einde van een mededeling gemarkeerd wordt door een zangerig "Over", d.w.z. door een nieuw symbool.)

Ons voorbeeld van de complexe vermenigvuldiging gaan wij nu wat completeren. Behalve dat we de al omvattende statementhaken toevoegen, geven wij, voordat de complexe vermenigvuldiging uitgevoerd gaat worden, reeel en imaginair deel een of andere waarde; zonder die maatregel zijn de expressies immers niet gedefinieerd. We komen b.v. tot:

```
begin
        x:= 5 / 13; y:= 12 / 13;
        u:= 0.6 × x − 0.8 × y;
        y:= 0.8 × x + 0.6 × y;
        x:= u
end
```

TYPE DECLARATIES

Dat dit programma zinloos is, omdat het geen resultaten aan de buitenwereld aflevert, deert ons voorlopig niet; belangrijker is, dat de tekst nog niet aan alle eisen voldoet. De boven bedoelde berekening luidt als volledig ALGOL-programma nl. als volgt.

```
begin   real x, y, u;
           x:= 5/13; y:= 12/13;
           u:= 0.6 × x - 0.8 × y;
           y:= 0.8 × x + 0.6 × y;
           x:= u
end
```

De regel, die we voor de eerste assignment statement ingelast hebben, is geen statement, maar een z.g. declaratie. (Wij hebben eerder het symbool ";" de statement separator genoemd; nu is het ogenblik, om de functie van dit symbool iets ruimer te omschrijven, want de puntkomma scheidt statements en declaraties.) De functie van de boven gegeven declaratie is niets anders, dan aan te kondigen, dat de namen x, y en u in de volgende tekst gebezigd zullen worden, om normale, reele variabelen mee aan te duiden. Voorlopig lijkt het wat overdreven om dat zo expliciet van tevoren in een declaratie aan te moeten kondigen, omdat dergelijke variabelen tot nog toe de enige zaken zijn, die we met namen hebben aangeduid. Er bestaan in ALGOL echter ook andere grootheden, van andere types, die we een naam moeten geven. Zodra deze ook in een programma gaan voorkomen, is het wel heel erg prettig om te weten naar wat voor soort grootheid een naam zal refereren.

De plicht in een declaratie van tevoren aan te kondigen in wat voor soort betekenis de namen gebruikt zullen gaan worden, is niet zo verwonderlijk, als men bedenkt, dat ALGOL een taal is, waarvan uitsluitend de structuur gegeven is, maar in het gebruik waarvan men zijn eigen woordenschat grotendeels kiezen mag. Wij komen dit probleem in het dagelijkse leven nu niet zoveel tegen, omdat in het Nederlands de meeste voorkomende woorden een vrij vaste betekenis hebben. Maar b.v. het feit, dat in het noorden van het land de naam "Pietje" als meisjesnaam niet ongebruikelijk is, heeft, zoals men zich denken kan, wel de nodige verwarring gesticht. Het is misschien verhelderend, zich te realiseren, dat in ALGOL zich haast bij elke naam een dergelijke situatie voordoet.

De real-declaratie bestaat uit het symbool "real" gevolgd door een of meer namen; in het geval van meer namen worden deze gescheiden door de komma "," . Achter de laatste naam komt ter afsluiting van de lijst een puntkomma, die de declaratie als geheel scheidt van de volgende declaratie of statement. De komma, die we hier voor het eerst hebben zien optreden, fungeert in ALGOL algemeen als separator van de individuele elementen van een lijst, in dit geval van een lijst namen.

Analoog aan de real-declaratie kent ALGOL de integer-declaratie. Deze bestaat uit het symbool "integer", gevolgd door een of meer namen. Ook hier worden in het geval van meer namen deze onderling door een komma gescheiden. Afgezien van het eerste symbool, dat in het ene geval real, in het andere geval integer is, is de structuur van de beide declaraties gelijk.

De integer-declaratie behelst, dat de nu volgende namen betrekking hebben op variabelen, die slechts gehele waarden kunnen aannemen. Voor de MC-vertaler geldt de restrictie, dat integer-gedeclareerde variabelen in absolute waarde kleiner moeten zijn dan 67108864 = 2↑26. Tegenover deze beperking staat de zekerheid, dat de waarden van deze variabelen tijdens de uitvoering van het ALGOL-programma exact gerepresenteerd zullen worden. Deze exacte representatie wordt voor de real-gedeclareerde variabelen met zoveel woorden niet gegarandeerd. Wij moeten er in het gebruik van ALGOL dus vanuit gaan, dat alle real-gedeclareerde variabelen slechts in een eindige precisie worden voorgesteld en dat we dientengevolge van geen enkele arithmetische operatie op deze variabelen verlangen mogen, dat deze exact wordt uitgevoerd. Hoewel het in het officiele ALGOL-rapport met opzet nergens expliciet gezegd wordt, mogen we ons bij real-gedeclareerde variabelen getallen voorstellen, die in de berekening met een vaste relatieve precisie worden voorgesteld. Hoe groot deze relatieve precisie is, is karakteristiek voor het "systeem" (machine, vertaler, etc.), dat het ALGOL-programma zal realiseren. Een goed ALGOL-programma maakt zo min mogelijk veronderstellingen over deze specifieke eigenschappen van een verwerkend "systeem".

Dat de preciese vorm van de getalvoorstelling en de arithmetiek onzeker is, geeft in de meeste gevallen niet de minste moeilijkheden. Waar ze wel optreden zijn ze essentieel van numeriek wiskundige aard - en dat deze zaken moeilijk blijven, kan men ALGOL niet aanrekenen - of de moeilijkheden kunnen opgelost worden door de introductie van de integer-gedeclareerde variabelen. Hier koopt men exacte representatie tegen de prijs van een beperkter waardebereik; voor het specifieke toepassingsgebied van integers (tellingen en indices) is deze beperking zelden hinderlijk.

Ten aanzien van de arithmetische operaties specificeert ALGOL, dat het resultaat van de deling "a/b" altijd van type real is, ook als a en b allebei van type integer waren en de deling toevallig opging. Het resultaat van de optelling "a + b", van de aftrekking "a - b" en van de vermenigvuldiging "a × b" is slechts dan van type integer, als zowel a als ook b van dit type is. Voor de MC-vertaler komt daar de conditie bij, dat het resultaat in absolute waarde kleiner moet zijn dan de bovengrens 67108864 (zie boven); is aan deze extra voorwaarde niet voldaan, dan wordt automatisch de overgang naar real-representatie ingelast.

Beschouwen wij nu de assignment statement:

"x:= E" ,

waarin E een of andere expressie is. Als de arithmetiek bij de uitwerking van de expressie E een resultaat van type integer aflevert, terwijl x een integer-gedeclareerde variabele is, dan is de functie van deze assignment statement tamelijk ondubbelzinnig.

Als de arithmetiek bij de uitwerking van de expressie f een resultaat van type real aflevert, terwijl x een real-gedeclareerde variabele is, dan valt er over deze assignment al iets meer te vertellen. Immers, het toegestane bereik van de absolute waarde van een real-gedeclareerde variabele is wel heel groot, maar niemand heeft ooit gezegd, dat dit bereik loopt van nul tot oneindig. Voor de MC-vertaler geldt het volgende: als de modulus onder een zekere grens ligt - ongeveer 10↑(-600) - , dan doet het teken niet meer ter zake, want het getal wordt dan geacht gelijk te zijn aan nul, als de modulus boven een zekere grens ligt - ongeveer 10↑(+600) - dan geldt het getal als plus of min oneindig. Bij de uitwerking van de expressie E kunnen de resultaten nul en oneindig in een grote hoeveelheid van representaties gevormd worden: bij de werkelijke waardetoekenning aan de variabele x behouden wij ons het recht voor, om voor de waarden nul en plus of min oneindig een "passende" representatie te kiezen. Evenzo is het niet uitgesloten dat het resultaat E primair in een groter aantal cijfers gevormd is, dan voor de representatie van de variabele x ter beschikking staan: in dat geval impliceert bij de MC-vertaler de waardetoekenning een afronding.

Als de arithmetiek bij de uitwerking van de expressie E een resultaat van type integer aflevert, maar x is een real-gedeclareerde variabele, dan eist ALGOL 60, dat in deze waardetoekenning de overgang naar real-representatie automatisch wordt ingelast. Voor de MC-vertaler kunnen wij hieraan toevoegen, dat, hoewel real-gedeclareerde variabelen principieel slechts in eindige precisie voorgesteld kunnen worden, in dit geval - zoals in elk bonafide systeem - de real-gedeclareerde variabele x de gehele waarde van E exact zal representeren.

Is omgekeerd het resultaat van de uitwerking van E in eerste instantie van type real, terwijl de variabele x van type integer is, dan impliceert deze waardetoekenning een afronding op de dichtstbij gelegen gehele waarde; ligt deze waarde buiten het voor integers toegestane waardebereik, dan stopt het programma. Men lette erop, dat op grond van de eindige precisie van resultaten van type real, bij integer-gedeclareerde variabele x het effect van de assignment statement

"x:= 7/2"

ongedefinieerd is: x kan net zo goed = 3 als = 4 worden.
In een herhaalde assignment moeten alle variabelen links van een wordtteken van hetzelfde type zijn.

Om het werken met integers iets te vergemakkelijken is een speciale integer-deling ingevoerd, aangeduid door het symbool "$\underline{:}$" (De officiele representatie van dit deelteken bestaat uit de superpositie van ":" en "–". Om technische redenen zullen wij het weergeven met een onderstreping van de dubbele punt.) Het quotient "m $\underline{:}$ n" is uitsluitend gedefinieerd als zowel m als n van type integer zijn. Voor n = 0 is het quotient "m $\underline{:}$ n" ongedefinieerd, in alle andere gevallen is het resultaat van type integer en de numerieke waarde is gelijk aan het gehele getal met de maximale absolute waarde, dat als quotient bij deling van m door n een rest nul of (als de deling niet opgaat) een rest met hetzelfde teken als m achterlaat. (Voor de absolute waarden geldt:

$$|m/n|-1 < |m\underline{:}n| \leq |m/n| \quad ;$$

in deze definitie representeert "m/n" bij wijze van uitzondering de exacte waarde van het quotient en niet, zoals elders in dit rapport, deze waarde in slechts een eindige precisie.)

We gaan nu ons oude voorbeeldje uitbreiden. Het zal de oplettende lezer niet zijn ontgaan, dat we het complexe getal z = x + yi, dat oorspronkelijk een modulus 1 heeft, vermenigvuldigd hebben met een factor 0.6 + 0.8i, die eveneens een modulus = 1 heeft. De nieuwe z heeft dus weer een modulus = 1 en moet dit houden, als we z herhaald met deze factor vermenigvuldigen. D.w.z. mathematisch blijft de modulus exact = 1, numeriek blijft de modulus ongeveer = 1; de afwijking wordt veroorzaakt door de afrondingen tijdens de berekening en mogelijk, omdat we reeel en imaginair deel van 0.6 + 0.8i niet exact representeren. (Men realisere zich, dat de breuken 0.6 en 0.8 b.v. in het tweetallig stelsel repeteren.) Als we de vermenigvuldiging met 0.6 + 0.8i nu herhaald uitvoeren, kunnen we een idee van de "kwaliteit" van de arithmetiek krijgen door te kijken, hoe snel de modulus van z van 1 afzeilt.

BESTEMMING STATEMENT

Een manier om de vermenigvuldiging van z herhaald uit te laten voeren is om het drietal assignment statements, dat de complexe vermenigvuldiging beschrijft, evenveel malen achter elkaar op te schrijven. Als we de vermenigvuldiging een groot aantal malen willen laten uitvoeren, zou op deze manier het maken van een ALGOL-programma meer gaan lijken op het maken van strafregels, dan op arbeid, die de pretentie van intelligent heeft. Wat we in de taal willen uitdrukken is zoiets als "en nu net als boven". We kunnen dit uitdrukken met behulp van een speciaal soort statement, de z.g. "go to statement". (Hierbij wordt gebruik gemaakt van een nieuw symbool go to, dat weer met behulp van onderlijning aangegeven wordt. Op het M.C. is inmiddels de gewoonte ontstaan, de spatie in het midden weg te laten en het symbool als "goto" weer te geven; men vestigt er zodoende nog eens extra de nadruk op, dat het hier een enkel symbool betreft en we zullen ons in het volgende bij deze gewoonte aansluiten.) Worden normaliter de statements uitgevoerd in de volgorde, waarin ze in de tekst staan, de goto-statement stelt ons in staat, deze volgorde te doorbreken: het kan een (voorlopig) willekeurig ander statement als opvolger aanwijzen.

De statement, die door de uitvoering van een goto-statement aan de beurt komt, moet daartoe een z.g. label dragen. Als label mag elke naam gekozen worden, mits natuurlijk in dit bestek deze naam niet al een andere betekenis heeft. Men labelt een statement door er een label voor te zetten en label en statement te scheiden door een dubbele punt ":". De goto-statement, die bewerkstelligt, dat bij deze statement de berekening voortgezet zal worden, kan dan bestaan uit het symbool "goto", gevolgd door de label in kwestie. NB. Labels hoeven niet van tevoren gedeclareerd te worden: hun optreden in de tekst op plaatsen waar een statement kan beginnen en gevolgd door het symbool ":" is voldoende indicatie, dat in deze tekst deze naam als label gebruikt wordt.

Het programma, dat z herhaald met 0.6 + 0.8i vermenigvuldigt, luidt onder gebruikmaking van de goto-statement als volgt.

```
begin   real x, y, u;
            x:= 5/13; y:= 12/13;
   AA:    u:= 0.6 × x - 0.8 × y;
            y:= 0.8 × x + 0.6 × y;
            x:= u; goto AA
end
```

DE CONDITIONELE STATEMENT

Als label hebben wij volkomen willekeurig de naam "AA" gekozen. Wij merken op, dat het programma nu een statement meer omvat en dat er dus een puntkomma meer in de tekst voorkomt. Nu gaat het bovenstaande programma tot in lengte van dagen door met vermenigvuldigen en het is niet onwaarschijnlijk, dat men eigenlijk liever had, dat het een bepaald aantal keren, b.v. 1000 keer, vermenigvuldigde, opdat men dan eens kijken kon, hoeveel de modulus van z inmiddels van 1 is afgeweken. Dat betekent, dat de goto-statement, die de herhaling bewerkstelligt, de eerste 999 keer wel uitgevoerd moet worden, maar dat het dan maar eens afgelopen moet zijn, d.w.z. dat dan de goto-statement overgeslagen moet worden. M.a.w. deze statement moet niet onder alle omstandigheden uitgevoerd worden, er zijn omstandigheden denkbaar, dat deze overgeslagen moet worden: ALGOL biedt de mogelijkheid, van een statement een z.g. "conditionele statement" te maken. De voorwaarde, waaraan voldaan moet zijn, wil de statement uitgevoerd worden, wordt onmiddellijk voor de statement gezet, ingeleid door het symbool "if" en afgesloten door het symbool "then"; is aan de hiertussen geformuleerde voorwaarde niet voldaan, dan wordt de statement, die op het symbool "then" volgt, overgeslagen.

Het programma, dat de vermenigvuldiging 1000 keer uitvoert, luidt b.v. als volgt.

```
begin   real x, y, u; integer k;
          x:= 5/13; y:= 12/13; k:= 0;
   AA:   u:= 0.6 × x - 0.8 × y;
          y:= 0.8 × x + 0.6 × y;
          x:= u;
          k:= k + 1; if k < 1000 then goto AA
end
```

In het boven gegeven voorbeeld is de z.g. "if clause" (d.w.z. het stuk van if tot en met then) gevolgd door een goto-statement, maar het mag ook best een assignment statement zijn. Wordt in de loop van een berekening gevraagd om b.v. de grootheid genaamd "Tol" door zijn absolute waarde te vervangen, dan kan men schrijven de conditionele statement

"if Tol < 0 then Tol:= - Tol" .

De if clause fungeert als voorvorgsel van de assignment statement en geeft dus geen aanleiding tot extra puntkomma.

Heel vaak zal de conditioneel uit te voeren handeling niet, zoals boven, in een enkel statement te formuleren zijn, b.v. niet als de variabelen Tel en Tal van teken gewisseld moeten worden, als Tol negatief is. En het is in een dergelijke situatie, dat de statementhaken begin en end tot hun recht komen:

de statements die tezamen de conditionele handeling beschrijven leidt men in met de openingshaak begin en sluit men af met de sluitingshaak end, met deze afscherming bereikend, dat dit compositum van buitenaf beschouwdt wordt als een enkele statement, in casu de statement die, als geheel, de if clause als voorvoegsel heeft gekregen. Het zojuist genoemde conditionele wisselen van teken van Tel en Tal formuleert men in ALGOL

"if Tol < 0 then begin Tel:= - Tel; Tal:= - Tal end"

Het stuk van begin tot en met end heeft de plaats van een enkele statement ingenomen; men lette erop, dat de enige puntkomma, die hierdoor in de tekst is gekomen, degene is, die de twee deelstatements van het compositum onderling scheidt.

Er is nog een geval, waarin het gebruik van de if clause aanleiding geeft tot een extra paar statementhaken: de statement, volgend op de if clause mag niet zelf al conditioneel zijn. (De reden hiervoor kan later duidelijk worden.) Moeten we dus, als Tol negatief is, Tel door zijn absolute waarde vervangen, dan kan dit in de volgende statement:

"if Tol < 0 then begin if Tel < 0 then Tel:= - Tel end" .

De voorwaarde in de if clause is hier geformuleerd als een relatie tussen twee grootheden, waaraan al dan niet voldaan kan zijn. ALGOL kent hier zes relaties, nl.

"<" kleiner dan

"≤" hoogstens

"=" gelijk

"≥" minstens

">" groter dan

"≠" ongelijk .

Tot nog toe hebben deze relatietekens uitsluitend tussen simpele variabelen en getallen gestaan, maar ze mogen ook tussen willekeurige arithmetische uitdrukkingen staan, dus b.v.

"if a + b = c + d then"
"if Flip × Flop ≤ Flap ↑ 2 then" etc.

De preciese betekenis van deze voorwaarden is duidelijk, als links en rechts van het relatieteken grootheden van het type integer staan. Zijn ze echter van het type real, dan is niet zonder meer duidelijk, wanneer we twee grootheden gelijk noemen: we zijn ons er immers steeds van bewust, dat deze variabelen slechts in een eindige nauwkeurigheid gerepresenteerd worden.

Voor de MC-vertaler geldt, dat aan de relatie "a = b" slechts dan voldaan is, wanneer zij identiek gelijk zijn. In andere woorden: de eindige precisie van real variabelen komt niet zozeer tot uiting in hun representatie, als wel in de arithmetische operaties, waarin zij gevormd worden. Men lette er op, dat de voorwaarde "a - b = 0" vervuld zou kunnen zijn voor niet identiek gelijke waarden van a en b, dwz. waarden waarvoor aan "a = b" niet voldaan is.

Tot nu toe hebben we if clause gebruikt om een statement al of niet uit te voeren; ALGOL kent daarnaast een ander gebruik van de if clause, waarbij het vervuld zijn van de voorwaarde uitmaakt of de ene, dan wel de andere statement uitgevoerd moet worden. In dat geval wordt de if clause gevolgd door de statement, die slechts uitgevoerd wordt, als aan de voorwaarde voldaan is, maar deze statement wordt nu gevolgd door het symbool "else", gevolgd door de statement, die slechts uitgevoerd wordt, als aan de voorwaarde niet voldaan is. Voorwaarden met B1, B2, etc. en statements met S1, S2, etc. aanduidend, heeft:

"if B1 then S1; S2"

tot gevolg, dat S1 slechts uitgevoerd wordt, als aan de voorwaarde B1 voldaan is, maar dat S2 als normale statement altijd aan bod komt, terwijl

"if B1 then S1 else S2; S3"

tot gevolg heeft dat S1 slechts uitgevoerd wordt, als aan de voorwaarde B1 wel voldaan is, S2 slechts als aan de voorwaarde B1 niet voldaan is, maar dat S3 als normale statement altijd aan bod komt. Men kan het effect van else dus ook zien als overslaan van S2 in het geval, dat S1 wel werd uitgevoerd; hieruit volgt, dat als S1 een goto-statement is het symbool else zonder bezwaar door een puntkomma vervangen kan worden. (De strekking van het ALGOL-programma wordt hierdoor niet beinvloed en de MC-vertaler construeert in het geval van puntkomma een iets korter programma.)

In tegenstelling tot S1 mag S2 wel een conditionele statement zijn, en wel van beiderlei type. De statements

"if B1 then S1 else if B2 then S3; S5"

en

"if B1 then S1 else if B2 then S3 else S4; S5"

zijn eenduidig uitlegbaar. Als aan B1 voldaan is, wordt in beide gevallen S1, en daarna S5 uitgevoerd. Als aan B1 niet voldaan is, maar aan B2 wel, dan worden in beide voorbeelden S3 en S5 uitgevoerd. Is aan geen van beide voorwaarden voldaan, dan geeft het eerste voorbeeld alleen S5, het laatste S4 gevolgd door S5.

Het is nu ook duidelijk, waarom de statement S1, d.w.z. de statement volgend op then niet conditioneel mag zijn. Ware dit niet verboden, dan zou de volgende constructie toegestaan zijn:

"if B1 then if B2 then S6 else S7; S8"

wat op twee manieren uitgelegd zou kunnen worden, nl.

"if B1 then begin if B2 then S6 else S7 end; S8"

en als

"if B1 then begin if B2 then S6 end else S7; S8" .

Ter verduidelijking geven wij nu het stukje programma, waarin de variabele genaamd MAX gelijk gemaakt wordt aan de grootste van drie variabelen met namen a, b en c.

```
"if a < b then
 begin if b < c then MAX:= c else MAX:= b end
       else
 if a < c then MAX:= c else MAX:= a"
```

Het programma heeft hier macroscopisch de vorm gekregen van een enkele conditionele statement van het tweede type; wij vestigen er en passant de aandacht op, dat hoewel hier vier assignment statements in voorkomen, de statement separator ";" er niet aan te pas is gekomen.

DE for-STATEMENT

Wij geven nu een andere vorm van het ALGOL-programma, waarin het complexe getal x + yi duizend keer met 0.6 + 0.8i vermenigvuldigd wordt; hierbij introduceren wij vier nieuwe ALGOL-symbolen, nl. "for", "do", "step" en "until". Het programma luidt in deze vorm:

```
begin   real x, y, u; integer k;
          x:= 5/13; y:= 12/13;
          for k:= 1 step 1 until 1000 do
          begin   u:= 0.6 × x − 0.8 × y;
                  y:= 0.8 × x + 0.6 × y;
                  x:= u
          end
end
```

Het stuk "for k:= 1 step 1 until 1000 do" is een z.g. "for clause". Een for clause behelst, dat aan een variabele (in dit geval aan de integer k) in volgorde een serie waarden wordt toegekend (in dit geval de beginwaarde 1 en dan met 1 opklimmend tot en met 1000) en dat voor elk van deze waarden van k de statement, die volgt op het symbool "do" een keer uitgevoerd wordt. Een for clause gevolgd door een statement worden samen een for-statement genoemd; de introductie van de for-statement is niet een wezenlijke verrijking van ALGOL, want alles wat met een for-statement uitgedrukt kan worden, zou ook zonder gebruikmaking daarvan met behulp van de andere statements uitgedrukt kunnen worden. De opgave een statement een aantal malen te herhalen is echter een zoveel voorkomende opgaaf, dat we de for-statement mogen beschouwen als een buitengewoon nuttige afkorting; er eenmaal aan gewend de for-statement te lezen, zal men bovendien merken, dat het gebruik ervan aanleiding pleegt te geven tot heel overzichtelijke programmas.

De herhaling strekt zich uit over de statement, die volgt op het symbool "do"; wordt die handeling primair door een aantal statements omschreven – in ons voorbeeld dus door drie assignment statements – dan omvat men deze door de statementhaken begin n end, op precies dezelfde manier als we bij de conditionele statement gedaan hebben. De analogie met de if clause gaat verder, want de statement, volgend op de for clause mag ook niet een conditionele statement zijn. (Ook deze verbodsbepaling is opgenomen om de programmeur te dwingen in zo een geval de statementhaken te gebruiken en daarmee een mogelijke dubbelzinnigheid op te heffen.)

Het boven gegeven voorbeeld maakt van de mogelijkheden van de for-statement slechts voor een gedeelte gebruik: de for-statement heeft hier de functie van een telling ("herhaal de volgende statement 1000 maal") en nergens is er gebruik gemaakt van het feit dat tijdens de eerste uitvoering van de te herhalen statement k = 1 is, tijdens de volgende uitvoering k = 2 is, etc. Dit gebruik wordt wel geillustreerd in het volgende programma, dat e (= de basis der natuurlijke logarithmen) benadert door de opeenvolgende omgekeerde faculteiten, b.v. tot en met die van 20 te sommeren.

```
begin   real eprox, term; integer n;
          eprox:= term:= 1;
          for n:= 1 step 1 until 20 do
          begin   term:= term / n;
                  eprox:= eprox + term;
          end
end
```

Het waardebereik van de lopende variabele (hier de integer k resp. n, maar het mag ook een real-gedeclareerde variabele zijn) is hier gegeven door een z.g. "step-until-element", waarvan de structuur gegeven is door

"beginwaarde step increment until eindgrens" .

Voor de laatste grootheid bezig ik expres niet de term "eindwaarde", want de laatste waarde hoeft nooit aangenomen te worden: het step-until-element wordt beschouwd als afgewerkt, zodra de eindgrens gepasseerd is. Zo is het element

"0 step 3 until 24"

equivalent aan het element

"0 step 3 until 25.73"

want in beide gevallen is 24 de laatste waarde, omdat de volgende (27) de eindgrens overschreden heeft.

De eindgrens hebben we ook niet "bovengrens" genoemd, want het increment kan negatief zijn, b.v.

"10 step -2 until 4"

doet de lopende variabele de waarden 10, 8, 6, 4 in deze volgorde doorlopen.

De grootheden, die we hier met "beginwaarde", "increment" en "eindgrens" hebben aangeduid, waren in onze voorbeelden getallen. Er zijn op deze plaatsen echter willekeurige uitdrukkingen toegestaan; in deze uitdrukkingen mogen variabelen voorkomen, waarvan de waarde in de herhaalde statement gewijzigd kan worden. Ter verhoging van de complicatie mogen increment en eindgrens een functie van de lopende variabele zijn. Dit is niet de plaats om in detail uit de doeken te doen, wat onder die omstandigheden de implicaties van het step-until-element zijn: het staat in het officiele ALGOL-rapport precies vastgelegd, meer ten bate van hen, die een vertaler moeten construeren dan voor de gebruikers van ALGOL, die in alle normale gevallen dergelijke gekunstelde constructies niet nodig hebben. Iets willen we toch van de gang van zaken schilderen.

We beschouwen de for-statement, symbolisch weergegeven door

"for LV:= A step B until C do S1"

waar LV de lopende variabele en S1 de te herhalen statement voorstelt. Beginwaarde, increment en eindgrens zijn voorgesteld door A, resp. B, resp. C, wat in het algemeen uitdrukkingen mogen zijn. Het specifieke step-until mechanisme komt in werking, iedere keer, dat de statement S1 mogelijkerwijs weer een keer aan bod komt. Nu spelen hierbij in principe twee waarden van LV een rol, de oude en de nieuwe. Bij het begin van een step - until - element wordt alleen de nieuwe waarde van LV expliciet gedefinieerd, nl. gelijkgemaakt aan de beginwaarde A; de volgende keren wordt de nieuwe waarde van LV berekend als door de assignment statement

"LV:= LV + B" .

Als het increment B een expressie is, waarvan de waarde afhangt van LV, dan is dit noodzakelijkerwijze de oude waarde van LV: de nieuwe is immers nog niet gevormd. Als de eindgrens C afhangt van LV, dan hangt deze af van de nieuwe waarde van LV. Voordat de statement S1 met de nieuwe waarde van LV uitgevoerd gaat worden, wordt onderzocht, of het step - until - element wegens passeren van de eindgrens soms uitgeput is. Het element wordt geacht volledig afgewerkt te zijn zodra de ongelijkheid

$$0 < B \times (LV - C)$$

blijkt te gelden. Voor het aanleggen van deze test hebben we de waarde (voornamelijk het teken) van het increment B nodig. Aan deze test wordt LV := A ook onderworpen, dwz. als we aan het step - until - element beginnen (en voor de vorming van de nieuwe LV het increment B nog niet nodig hebben). In geval van integer k en n wordt de statement S1 door

"for k:= 0 step 1 until n do S ; S2"

voor nonnegatieve n dus n+1 maal uitgevoerd, voor negatieve n dus niet m.a.w. de "lege cyclus" is ook toegestaan.

Wij laten aan de lezer over om na te gaan, hoevaak en met welke successieve waarden van k de statement S1 uitgevoerd wordt in:

```
begin   integer k, m, n;
          .......;
          n:= expressie;
          m:= 11;
          for k:= 0 step m until n do begin S1; m:= m - 1 end
end
```

wanneer buiten de aangegeven plaatsen geen assignments aan n , m en k voorkomen; men onderzoeke dit voor de volgende waarden van "expressie": -10, 0, 5, 30, 53, 54, 55 en 56.

Tussen wordtteken en do stond in onze voorbeelden een step-until-element. Het mogen er echter meer zijn; ze worden dan door een komma gescheiden en van links naar rechts afgewerkt. Zo is b.v.

"for LIM:= 0 step 2 until 100 do S1; S2"

equivalent aan

"for LIM:= 0 step 2 until 30, 32 step 2 until 100 do S1; S2" .

Een zinvoller gebruik zou zijn – b.v. voor het maken van een tabel met verschillende intervallen – iets in de trant van:

"for arg:= 0 step .01 until .5, .52 step .02 until 1 do" .

In plaats van de step-until-elementen mogen we ook eenvoudigere dingen zetten, nl. een enkele waarde of een z.g. while-element.

De enkele waarde is in het algemeen een expressie; deze waarde wordt aan de lopende variabele toegekend, de statement volgend op do wordt een keer uitgevoerd en daarna wordt gekeken of er nog volgende waarden voor de lopende variabele in de for clause gedefinieerd zijn.

In plaats van

"for k:= -5 step 3 until +7 do S1; S2"

mag – in de veronderstelling, dat de statement S1 de waarde van de lopende variabele k niet wijzigt – dus ook geschreven worden als:

"for k:= -5, -2, 1, +4, 7 do S1; S2"

of, wat ingewikkelder,

"for k:= -5, k+3, +1 step +3 until 7 do S1; S2" .

Het while-element bestaat uit een arithmetische expressie, gevolgd door het nieuwe symbool "while", waarachter een voorwaarde. Het while-element behelst, dat de expressie wordt uitgewerkt, de waarde hiervan aan de lopende variabele wordt toegekend, de statement volgend op do wordt uitgevoerd, waarna opnieuw de expressie wordt uitgewerkt en de waarde daarvan aan de lopende variabele wordt toegekend, etc. Omdat dit proces op deze manier ad infinitum zou doorgaan, is echter de voorwaarde, ingeleid door while eraan toegevoegd. Vlak voordat de statement, volgend op do aan de beurt komt, wordt gecontroleerd, of aan de voorwaarde in kwestie voldaan is, zo ja, dan is de gang van zaken, als boven beschreven, zo nee, dan eindigt dit proces en wordt het while-element beschouwd als afgehandeld.

Met behulp van het while-element kunnen we ons laatste voorbeeld ook formuleren als volgt:

"for k:= -5, k + 3 while k $\leq$ 7 do S1; S2" .

In het algemeen volgt op het wordtteken in de for clause dus een lijst van "elementen", onderling gescheiden door komma en afgesloten door het symbool do; we hebben de drie types gezien - het step-until-element, de enkele waarde en het while-element - en deze drie types mogen elkaar op elke manier afwisselen. Als het laatste element van de lijst afgewerkt is, geldt de hele for-statement als voltooid.

De statement, volgend op do kan gelabeld zijn of, als deze statement "samengesteld" is - d.w.z. bestaat uit een aantal statements, onderling gescheiden door puntkomma en als geheel omvat door de statementhaken begin en end - , dan kan een van de "binnenstatements" gelabeld zijn. De statement, volgend op het symbool do (d.w.z. het stuk, waarop de herhaling zich betrekt) heet het "bereik van de for-statement: het is verboden via een goto-statement buiten het bereik van een for-statement het bereik van die for-statement binnen te gaan.

Anderzijds kan in het bereik van een for-statement een goto-statement staan, die leidt naar een gelabeld statement erbuiten. De for-statement is dan (voortijdig) voltooid en de waarde van de lopende variabele is gelijk aan de waarde, vlak voor de voltooiing van de goto-statement. Dit moet expliciet vermeld worden, omdat in het geval, dat de for-statement afgehandeld is, omdat het laatste element van de lijst volledig is afgewerkt, de waarde van de lopende variabele ongedefinieerd is.

(De afspraak is gemaakt in de veronderstelling, dat men, als het laatste element een step-until-element of een while-element is, de constructeurs van ALGOL-vertalers de vrijheid zou laten, of de waarde van de lopende variabele de laatst geaccepteerde, dan wel de eerst verworpene zou zijn. Aangezien men bij het onderzoek van de voorwaarde de nieuwe waarde van de lopende variabele nodig kan hebben, is er in feite geen keus, hoe vertalers de for-statement realiseren; de afspraak, dat de waarde van de lopende variabele ongedefinieerd is na de afwerking van het laatste lijstelement is dan ook zinloos. Bij de MC-vertaler is de waarde van de lopende variabele na de afwerking van een element in de for clause wel degelijk gedefinieerd; voor een enkele waarde is het deze waarde, voor de twee andere soorten element is het de eerst verworpen waarde (dit alles natuurlijk in de veronderstelling, dat in het bereik van de for-statement de lopende variabele niet door een assignment statement een nieuwe waarde toegekend krijgt).)

De lezer realisere zich, dat het lijstelement in

"for x:= a step b until x do ....."

nooit is afgehandeld; tenzij deze for-statement beeindigd wordt door een verlating van het bereik via een goto-statement, eindigt deze for-statement nooit.

SPECIALE FUNCTIES

Negen namen hebben in ALGOL 60 een speciale betekenis. Het zijn de namen van de volgende standaardfuncties. Het argument, dat tussen haakjes achter de naam van de functie geplaatst wordt, mag een willekeurige arithmetische uitdrukking zijn; wij duiden hieronder deze expressie aan met "E".

abs(E)	= de absolute waarde (de modulus) van de waarde van de uitdrukking E.
sign(E)	= het "teken" van de waarde van E (d.w.z. = +1, als de waarde van E positief is, = 0 als de waarde van E nul is en = −1, als de waarde van E negatief is).
sqrt(E)	= de vierkantswortel uit de waarde van E
sin(E)	= de sinus van de waarde van E
cos(E)	= de cosinus van de waarde van E
arctan(E)	= de hoofdwaarde van de arctangens van de waarde van E
ln(E)	= de natuurlijke logarithme van de waarde van E
exp(E)	= de exponentiele functie van de waarde van E
entier(E)	= het grootste gehele getal, dat niet groter is dan de waarde van E.

Het argument van deze negen functies mag zowel van type integer als van type real zijn; het antwoord is bij de functie abs van hetzelfde type als het meegegeven argument, bij de functies sign en entier van type integer en voor de overige zes altijd van type real. (Aldus geldt het voor de MC-vertaler. In de officiele ALGOL is het antwoord van de functie abs altijd real, ook als het meegegeven argument van type integer is.)

Waar een argument (bij sin en cos) of het antwoord (bij arctan) als hoekmaat geduid pleegt te worden, is dit natuurlijk een hoekmaat in radialen; als basis van de logarithme en de exponentiele functie is – even vanzelfsprekend – e gekozen.

Op nog een klein punt wijkt de MC-vertaler van het officiele ALGOL-rapport af: de functies sqrt en ln opereren bij de MC-vertaler op de absolute waarde van het meegegeven argument. (Wij hebben dit zo gekozen om eventuele moeilijkheden te ondervangen, die anders zouden kunnen optreden bij een wiskundig heel klein positief argument, dat door afrondingen abusievelijk net onder de nul gezakt is.)

Opm. Kennelijk geldt abs(E) = E × sign(E).

ALGOL 60 is, zoals gezegd, in eerste instantie een taal, die ontworpen is om numerieke processen in te beschrijven. Wil men deze taal gebruiken voor de documentatie van rekentechnieken, dan vormt ALGOL 60 een afgerond geheel, dat zich voor dit doel goed leent. Zo goed zelfs, dat wij er meer mee willen: wij willen een reken-proces in ALGOL 60 kunnen beschrijven en een rekenmachine aan de hand van deze beschrijving dit proces uit kunnen laten voeren. Maar zodra wij geinteresseerd zijn in de feitelijke uitvoering, dan zal dat haast altijd voortvloeien uit het feit, dat wij geinteresseerd zijn in de antwoorden. Wij willen in de beschrijving van het uit te voeren rekenproces aan kunnen geven, wat de resultaten zijn, die de rekenmachine bij de feitelijke uitvoering van het proces aan de buitenwereld zal moeten afleveren. Het officiele ALGOL 60 rapport zwijgt hierover in alle talen en met opzet.

Men kan de facto nl. niet volstaan met een aanduiding, welke resultaten aan de buitenwereld afgeleverd moeten worden. Hoe dit kan geschieden, hangt immers in hoge mate af van de communicatie-apparatuur, waarover de machine beschikt. Zo is de X1 in staat, tijdens zijn werkzaamheden een electrische schrijfmachine te bedienen en we zullen deze schrijfmachine door de X1 laten gebruiken om de gewenste resultaten af te leveren. Maar een ieder weet, dat de lengte van een schrijfmachinewagen eindig is, m.a.w. we zullen niet alleen aan moeten geven, welke getallen getypt moeten worden, maar we zullen ook aan moeten geven, wanneer we op een nieuwe regel moeten beginnen. Was de machine uitgerust met een drukmechanisme, dat als in een telmachine alle getallen onder elkaar op een strook afdrukt, dan was dit probleem er niet geweest. Omdat men nu niet alleen aan moet geven, wat er aan resultaten afgeleverd moet worden, maar ook hoe en omdat de specificaties van het laatste van machine tot machine sterk zullen verschillen, heeft men dit opzettelijk buiten de taal gehouden. Men heeft dit gedaan in de veronderstelling, dat elke organisatie, waarmee ALGOL-programmas door een machine uitgevoerd zouden kunnen worden, die uitbreidingen van de taal zou omvatten, die door de specifieke behoeften ter plaatse gedicteerd zijn.

Dergelijke uitbreidingen zijn ook voor de MC-vertaler ingevoerd. Zij vallen buiten de officiele taal, en zijn mede daardoor iets minder definitief. Wanneer de organisatie van de MC-vertaler in de naaste toekomst nog gewijzigd zal worden of uitgebreid, dan mogen wij die veranderingen in deze hoek verwachten.

De gekozen oplossing vertoont grote analogie met de wijze waarop de standaard functies geincorporeerd zijn. Werden daar speciale namen geintroduceerd om een functie aan te geven (met als resultaat een term in een expressie), hier worden speciale namen geintroduceerd, om een (iets minder) standaard handeling aan te geven. In de tekst, waarin deze standaard handeling is opgenomen, fungeert de laatste als zelfstandige statement. Wij noemen er nu twee.

NLCR : New Line Carriage Return
print(E) : Typ de waarde van de expressie E .

Zolang de statement NLCR niet wordt uitgevoerd, typen opeenvolgende print statements de getallen achter elkaar op dezelfde regel. Na de statement NLCR wordt het eerstvolgende te typen getal getypt aan het begin van de volgende regel. Wij zullen het gebruik van deze statements aan enkele voorbeelden illustreren.

Eerst pakken we ons voorbeeldje van de complexe vermenigvuldiging weer op. We maken een programma, dat ad infinitum doorgaat met vermenigvuldigen, maar dat na elke duizendste vermenigvuldiging uittypt, hoeveel de modulus van 1 afwijkt; deze afwijkingen worden onder elkaar getypt.

```
begin   real x, y, u; integer k;
          x:= 5/13; y:= 12/13;
   BB:    for k:= 1 step 1 until 1000 do
            begin   u:= 0.6 × x - 0.8 × y;
                    y:= 0.8 × x + 0.6 × y;
                    x:= u
          end;
          NLCR; print (x × x + y × y - 1); goto BB
end
```

Het volgende voorbeeld produceert een tabel van sinh(x) en cosh(x) voor x = 1(.01)2. Elke regel van de tabel bevat drie getallen, het argument, de bijbehorende sinh en cosh. Men lette erop, dat de eindgrens in de for-statement, nu de lopende variabele van type real is, iets te ruim is genomen. Op deze wijze garanderen we, dat 2 nog wel en 2.01 niet meer meedoet.

```
begin   real x;
          for x:= 1 step 0.01 until 2.001 do
          begin   NLCR; print (x);
                  print (0.5 × (exp(x) - exp(-x)));
                  print (0.5 × (exp(x) + exp(-x)))
          end
end
```

Helemaal netjes is bovenstaand programma niet, omdat we niet kunnen garanderen, dat het increment in de for-statement exact wordt gerepresenteerd. Aan het einde van onze tabel zal dan het argument met een honderd maal zo grote fout behept zijn. In de volgende versie is dit bezwaar ondervangen. Tevens is, in de veronderstelling, dat de berekening van de exponentiele functie een relatief bewerkelijke operatie is, de hoeveelheid rekenwerk wat verminderd. (Volledigheidshalve vermelden wij, dat voor de X1 de snelheidsverhoudingen zodanig liggen, dat deze vermindering van de hoeveelheid rekenwerk niet leidt tot versnelling van het programma: de X1 rekent zo snel, dat al in de boven gegeven versie de totale tijd bepaald wordt door de snelheid van de schrijfmachine, die continu zal staan te typen.)

```
begin     integer k; real x;
            for k:= 100 step 1 until 200 do
            begin   NLCR; print (k/100); x:= 0.5 × exp(k / 100);
                    print(x - .25/x);
                    print(x + .25/x)
            end
end
```

COMMENTAAR

ALGOL 60 kent twee methoden, om ter verhoging van de leesbaarheid het programma op saillante punten van commentaar te voorzien. Dit commentaar is uitsluitend bestemd voor de menselijke lezer, het moet door de vertaler overgeslagen worden. Opdat de vertaler eenduidig vast kan stellen, welk gedeelte van de tekst overgeslagen kan worden en waar het eigenlijke programma weer doorgaat, is dit commentaar aan enige regels gebonden.

De eerste vorm van commentaar wordt ingeleid door het speciaal daartoe ingevoerde symbool "comment". Vanaf dit symbool worden alle volgende symbolen tot en met de eerstvolgende puntkomma als niet ter zake doend commentaar geskipt. Met andere woorden, voor dit soort commentaar, dat de puntkomma niet mag bevatten, fungeert het symbool "comment" als openingshaak en de puntkomma als sluitingshaak. Deze vorm van commentaar is officieel in ALGOL 60 alleen toegestaan na een puntkomma of na het symbool "begin"; de MC-vertaler staat het overal toe.

De tweede vorm van commentaar is toegestaan na het symbool "end". Hierna worden, indien aanwezig, als commentaar alle symbolen geskipt tot aan (en niet tot en met) de eerstvolgende puntkomma, end of else. Voor de MC-vertaler gelden hierbij echter twee restricties. Ten eerste mag deze reeks symbolen, die overgeslagen moet worden, niet de symbolen "begin", "comment" en de z.g. string quotes "‹" en "›" bevatten, ten tweede is deze vorm van commentaar niet toegestaan na de end van het alomvattend hakenpaar, d.w.z. aan het einde van het programma. Deze vorm van commentaar wordt vooral gebruikt om aan te geven, bij welke begin de end in kwestie behoort, dit is vooral van belang als men deze toevoeging door de layout niet meer duidelijk kan maken.

ARRAYS

Het uitdrukkingsvermogen van ALGOL 60 is aanmerkelijk uitgebreid door de introductie van het concept "array". Het eenvoudigste voorbeeld van een array is een vector, d.w.z. een rij geindiceerde variabelen. Om het array als geheel aan te duiden, kiest men een of andere naam, om een bepaald element aan te wijzen, specificeert men achter de naam van het array de betrokken indexwaarde tussen vierkante haken. Een index mag altijd een aaneensluitende rij gehele getallen doorlopen, in de keuze van minimum en maximum waarde (onder- resp. bovengrens) is de programmeur vrij.

Voordat wij de gekozen array-namen als zodanig mogen gebruiken, moeten wij dit gebruik van tevoren aangekondigd hebben in een z.g. "array-declaratie". Deze bevat dan tevens het gegeven, van welk type (real of integer) de individuele elementen zijn, alsmede onder- en bovengrenzen voor de indices. Onder- en bovengrens worden in de array-declaratie gespecificeerd door ze, onderling gescheiden door een dubbele punt, tussen vierkante haken te plaatsen achter de namen, waarop deze grenzen betrekking hebben. Een voorbeeld moge dit toelichten.

Stel, we willen vier vectoren - of, zoals we ze liever noemen: eendimensionale arrays - introduceren. Drie daarvan, te weten de arrays met de namen Mozart, Bach en Brahms, zijn opgebouwd uit elementen van type integer en het vierde array, Beethoven genaamd, heeft elementen van type real. (Alle elementen van eenzelfde array zijn altijd van hetzelfde type.) De arrays Mozart en Bach tellen elk vijf elementen, genummerd van 1 t/m 5, de arrays Brahms en Beethoven tellen elk negen elementen, genummerd van -4 t/m +4. Wij kunnen dit declareren met behulp van twee declaraties, nl. een voor elk type. Zij luiden:

"integer array Mozart, Bach [1:5], Brahms [-4:+4];
real array Beethoven [-4:4]"

Uit dit voorbeeld zijn de belangrijkste regels te destilleren, volgens welke de array-declaratie's opgesteld dienen te worden. Alle arrays van hetzelfde type kunnen (maar hoeven niet) in eenzelfde declaratie behandeld worden; is het type integer, dan begint de declaratie met de symbolen "integer array", zijn ze van type real, dan begint de declaratie met de symbolen "real array". (In het geval van "real array" mag men het symbool "real" weglaten, m.a.w. laat men in een array-declaratie het type onvermeld, dan is het type real bedoeld.) Hierachter volgt een naam (of meer namen, onderling gescheiden door een komma), waarachter de indexgrenzen tussen vierkante haken. Deze indexgrenzen hebben betrekking op de hier vlak voor staande lijst namen. Is hiermee het laatste array van het onderhavige type geintroduceerd, dan wordt de declaratie met een puntkomma afgesloten, anders komt er een komma ter aanduiding van een of meer arrays van hetzelfde type, maar met mogelijk andere indexgrenzen.

Wij hebben hiermee 28 arrayelementen geintroduceerd, b.v. Mozart [1], Mozart [2], Mozart [3], Mozart [4], Mozart [5], Bach [3], Brahms [-2] en Beethoven [4]. Een index mag niet buiten de aangegeven grenzen liggen: het element Beethoven [9] b.v. is ongedefinieerd.

Behalve dergelijke vectoren mag men ook arrays met meer dan een index introduceren. In de array-declaratie geeft men dit aan door achter de bovengrens van de eerste index niet meteen de vierkante sluitingshaak te zetten, maar een komma en dan het grenzenpaar voor de volgende index, etc.; de vierkante sluitingshaak volgt dan op de bovengrens voor de laatste index. Om een element van een een- of meerdimensionaal array aan te wijzen, moet men, waar bij het vectorelement slechts een indexwaarde gespecificeerd behoefde te worden, nu alle indexwaarden in volgorde geven, onderling weer door komma's gescheiden.

De volgende array-declaratie introduceert drie arrays met elementen van type real, twee tweedimensionale en een driedimensionale.

"array Rembrandt, Vermeer [-10: +10, 3:7],
le Corbusier [0:10, 0:7, 0:4] "

De eerste twee arrays omvatten elk 21 × 5 = 105, het laatste omvat 11×8×5 = 440 elementen. Zo kan men praten over de elementen Rembrandt [8,4] en le Corbusier [3,3,3]; het element Rembrandt [4,8] is weer ongedefinieerd, omdat een van de indexwaarden (nl. de laatste) buiten de bijbehorende grenzen valt. Het element le Corbusier [5,2] is ongedefinieerd, omdat het aantal indices niet in overeenstemming is met de declaratie.

De index van de geindiceerde variabele is in wezen een gehele waarde. Tussen de vierkante haken mag een willekeurige expressie staan - b.v. Mozart [i × i - 1] - en de waarde van een dergelijke expressie mag zelfs van type real zijn - b.v. Bach [sin(x) + cos(pi × exp(y + yy) - 7)] - . In een dergelijk geval wordt de expressie afgerond op de dichtstbij gelegen gehele waarde; het element Beethoven [7/2] is weer onbepaald.

In ALGOL 60 moeten de declaraties van een programma vooraan staan, voor de eerste statement; hun onderlinge volgorde is in principe vrij.

Wij zullen nu, complee met in- en uitvoer van gegevens, een matrix A, bestaande uit 20 kolommen, elk van zes elementen, vermenigvuldigen met een kolomvector B van dus 20 elementen en de antwoordvector (van dus 6 elementen) uittypen. Wij nemen aan, dat de elementen van A kolomsgewijs op een band geponst staan en dat na de laatste kolom van A op diezelfde band de elementen van vector B staan. Om de informatie, die op een band geponst staat, in een werkend ALGOL-programma op te kunnen nemen, is het "vocabulaire" van de MC-vertaler uitgebreid met de speciale functie "read". De waarde hiervan is gelijk aan die van "het volgende getal op de band"; uit de aard der zaak moet, voordat een ALGOL-programma om getallen van een band vraagt, deze band met zijn begin in de bandlezer van de X1 gelegd zijn.

```
begin   integer i, j; real s; array A[1:6, 1:20], B[1:20];
        NLCR;
        for j:= 1 step 1 until 20 do
           for i:= 1 step 1 until 6 do A[i,j]:= read;
        for i:= 1 step 1 until 20 do B[i]:= read;
        for i:=  1 step 1 until 6 do
        begin   s:= 0;
                for j:= 1 step 1 until 20 do s:= A[i,j] × B[j] + s;
                print(s)
        end
end
```

Het programma bestaat primair uit vier statements, nl. NLCR, gevolgd door drie for-statements. Bij de eerste hiervan zien we, dat de statement, die volgt op een for clause, zelf weer een for-statement mag zijn; bij de laatste hiervan, waarin de eigenlijke berekening is beschreven, hebben we van de statement-haken begin en end gebruik moeten maken.

De vorm van dit programma werd gedicteerd door de wijze, waarop de getallen op de band stonden. Had op deze band eerst de vector B gestaan en dan de matrix A, maar nu rijgewijs , dan hadden we met een veel eenvoudiger programma kunnen volstaan, nl.

```
begin   integer i, j; real s; array B[1:20];
        NLCR;
        for j:= 1 step 1 until 20 do B[j]:= read;
        for i:= 1 step 1 until 6 do
        begin   s:= 0;
                for j:= 1 step 1 until 20 do s:= read × B[j] + s;
                print(s)
        end
end
```

EENDUIDIGHEID VAN NAMEN

Wij hebben nu inmiddels "namen" ontmoet voor een aantal verschillende objecten:

scalairen
labels
speciale functies en standaard handelingen
arrays .

Het is verboden eenzelfde naam in een bepaald bestek met een dubbele betekenis te gebruiken, ook al zou hierdoor geen dubbelzinnigheid optreden, omdat in de tekst duidelijk is, van welke aard het benoemde objec t is. Zo mogen we niet tegelijkertijd willen kunnen praten over:

de scalar "sin" (in expressie of links van ":=")
de label "sin" (in goto-statement of links van ":")
de functie "sin" (gevolgd door "(")
het array "sin" (gevolgd door "[") .

Dit verbod is kort en cryptisch uitgedrukt in het principe "identifier identifies", dat een van de hoekstenen van ALGOL 60 is. Dit verbod strekt zich ook uit over de namen van de nog onbesproken objecten (de Boolean, de switch en de procedure).

DE PROCEDURE

Ons voorbeeld van de vermenigvuldiging "matrix maal vector" hebben we gegeven als illustratie van het gebruik van arrays; het toonde daarnaast de for-statement in al zijn glorie. Zo liet het duidelijk zien, dat het for-mechanisme niet enkel een telling is, die zorgt, dat een statement een aantal malen herhaald wordt, maar dat in deze statement met vrucht gebruik gemaakt kan worden van de waarde van de "lopende variabele". Verder is dit voorbeeld, een programma, waardoor o .a. 120 vermenigvuldigingen uitgevoerd zullen worden, een treffend bewijs van de compactheid van beschrijvingen in ALGOL. De for-statement is niet de enige methode, waardoor we de programmatekst kunnen verkorten: een zeker zo belangrijke mogelijkheid is het bijzonder flexibele afkortingsmechanisme, dat ons ter beschikking staat in de vorm van de z.g. "procedure".

In een programma komt een variabele voor, genaamd "cosphi", die op een groot aantal verschillende plaatsen in het programma een wijziging moet ondergaan, die beschreven kan worden door de assignment statement

"cosphi:= 2 × cosphi × cosphi − 1" .

In plaats van nu deze statement op alle plaatsen, waar cosphi aldus gewijzigd moet worden, voluit neer te schrijven, kunnen wij deze handeling een naam geven; dit eenmaal gebeurd zijnde hoeven wij daar waar deze handeling plaats moet vinden, slechts deze naam te vermelden.

Wij kiezen voor deze handeling een naam, die in dit bestek geen andere betekenis heeft; dit zij de naam "DUB". Deze keuze leggen wij vast in een z.g. "procedure declaratie". In zijn eenvoudigste vorm bestaat deze uit het hierbij te introduceren nieuwe symbool "procedure", onmiddellijk gevolgd door de gekozen naam; deze wordt dan afgesloten door een puntkomma en hierna volgt de statement, die deze naam gekregen heeft. (Hiermee is de declaratie ten einde. Aangezien de declaraties in de regel gevolgd worden door andere declaraties of door een statement, volgt er na de benoemde statement dus een puntkomma.)

In ons voorbeeld zou de procedure declaratie geluid hebben:

"procedure DUB; cosphi:= 2 × cosphi × cosphi − 1" .

Hierna mogen we overal in de tekst, waar deze statement uitgevoerd moet worden, in plaats daarvan "DUB" schrijven. Dit is dan een zelfstandige statement − dit soort heet "een procedure statement" − , die normaal als alle andere statements met behulp van puntkomma's van voorganger en volgeling gescheiden moet worden.

In gebruik onderscheidt de procedure DUB zich niet van de reeds eerder genoemde standaard handeling NLCR; het enige verschil is, dat de naam NLCR behoort tot het primitieve vocabulaire van de MC-vertaler en daardoor zonder voorafgaande declaratie gebruikt mag worden. En nu is het ook duidelijk, hoe de statement NLCR, die we eerder min of meer als extra'tje invoerden, syntactisch volledig in ALGOL 60 past: het is in het gebruik een procedure statement, net als ieder ander. Voor een klein aantal procedures (waaronder NLCR) is de functie eens en voor al afgesproken; daarnaast heeft de programmeur de vrijheid om met behulp van een procedure declaratie naar behoefte nieuwe procedures te definieren.

Nu beschouwen we het geval, dat op vele plaatsen in het programma twee variabelen genaamd "a" en "b", verwisseld moeten worden. Dit vergt, zoals gemakkelijk is na te gaan, drie statements en er is dus reden genoeg, om deze handeling als procedure te behandelen. Dat de procedure declaratie de naam hecht aan "de erin voorkomende statement", deert ons niet, want we kennen inmiddels de geijkte methode om een aantal statements naar buiten als een enkel te laten fungeren: de statementhaken begin en end. De procedure declaratie voor de verwisseling van a en b luidt, als we hiervoor de naam "wsl" kiezen:

```
"procedure wsl;
 begin   real s;
         s:= a; a:= b; b:= s
 end"    .
```

(Hierin zal end, het laatste symbool van de declaratie, wel weer door een puntkomma gevolgd worden.)

Het zal de oplettende lezer niet ontgaan zijn, dat wij in deze procedure declaratie tussen de haken begin en end nog iets meer gezet hebben, dan de drie statements, nl. eerst nog een declaratie voor de hulpgrootheid, die we voor het verwisselproces nodig hebben. We bereiken hiermee, dat iedere keer, dat later de procedure statement "wsl" uitgevoerd wordt, voor de tijdsduur van deze activering van de procedure wsl even de grootheid s geintroducerd wordt, om de verwisseling uit te kunnen voeren. Deze variabele s, die een z.g. "locale" variabele van de procedure wsl genoemd wordt, omdat hij binnenin de procedure gedeclareerd is, heeft geen betekenis daarbuiten. Om te kunnen expliceren, wat met "binnenin" en "daarbuiten bedoeld is, kunnen wij niet volstaan met enkel te kijken naar de procedure declaratie, maar moeten wij de declaratie van de procedure wsl "in zijn omgeving" beschouwen, d.w.z. in het programma, waarin deze declaratie voorkomt.

LOCALE GROOTHEDEN

Een programma waarin de procedure wsl gedeclareerd en gebruikt wordt, zou er ruwweg als volgt uit kunnen zien.

```
begin   real a, b, ...;
        integer ...;
        integer array s,...[...:...]...;
        real array ...;
        procedure wsl;
        begin   real s;
                s:= a; a:= b; b:= s
        end
        ...; wsl;...; wsl;...; s[...]:=...; wsl;...
end
```

Het programma begint met vijf declaraties; de eerste declareert minstens de variabelen a en b, waarvan we immers hadden aangenomen, dat ze in het programma voorkwamen. De tweede declareert eventuele integers; met opzet hebben we hier de naam "s" aan gegeven, hier als naam van een integer array. De vijfde declaratie is de declaratie van de procedure, genaamd "wsl". De statement, die de functie van de procedure vastlegt (hier het stuk van en met de tweede begin tot en met de bijbehorende end) heet "het procedure lichaam". Begint het procedure lichaam met declaraties, dan zijn deze declaraties "locaal" ten opzichte van dit procedure lichaam. Het concept "locaal" heeft twee aspecten, een lexicographisch en een dynamisch.

Het lexigcografisch aspect betreft over welk gedeelte van de tekst een declaratie van toepassing is. Dit is in pricipe van de laatst voorafgaande begin totaan de bijbehorende end. Een dergelijk stuk heet een "blok" - waarover later meer - ; het hele programma is dus "een blok" en de declaraties aan het begin van het programma strekken zich in principe dan ook over het hele programma uit. Wij zien dit bij de real gedeclareerde variabelen a en b; overal in het programma refereren de namen a en b naar deze variabelen. Het procedurelichaam op zichzelf is weer een blok(een "binnenblok"van het programmablok). Declaraties aan het begin van dit blok ("real s") hebben alleen betekenis in dit blok; heeft een hier gedeclareerde naam in het omvattende blok al een betekenis, dan raakt deze betekenis (hier dus de naam "s" ter aanduiding van een array) tijdelijk in het vergeetboek, om aan het einde van het binnenblok weer in ere hersteld te worden. (Het gebruik van het woord "tijdelijk" in de vorige zin is erop gebaseerd, dat men het ALGOL programma van begin tot eind leest: om het lexicografisch aspect te onderstrepen zou "ruimtelijk" misschien correcter geweest zijn.) Men kan dus zeggen, dat de betekenis,

de reikwijdte van een declaratie zich uitstrekt over het gehele eigen blok, met inbegrip van zijn binnenblokken (en binnen-binnenblokken, etc.), tenzij de naam in zo'n binnenblok een nieuwe functie krijgt. Omgekeerd: hoe vindt men de betekenis (d.w.z. de bijbehorende declaratie)van een ergens gebruikte naam? Men kijkt naar het kleinste blok, dat de plaats van gebruik omvat en wordt de naam hier gedeclareerd, dan zijn we klaar; zo nee, dan kijken we naar het kleinste dit blok omvattende blok, of aan het begin hiervan de naam in een declaratie voorkomt etc. In ons voorbeeld is de consequentie van het dubbele gebruik van de naam "s", dat we in het procedure lichaam van wsl niet over het array s kunnen praten. Hadden we dit gewild, dan hadden we voor de hulpgrootheid in wsl een andere naam moeten kiezen.

Ter aanduiding ervan, dat achter de procedure declaratie de naam s weer in zijn oorspronkelijke betekenis terugkomt, hebben we in de regel, die de statements van het programma voorstelt, de naam s door [...] laten volgen.

Het dynamisch aspect van het locaal zijn van de declaratie "real s" betreft de procedure statements "...; wsl; ...", zoals ze in ons voorbeeld driemaal voorkomen. Als dit programma uitgevoerd gaat worden, moet voor elk van deze procedure statements de uitvoering van het bijbehorende procedure lichaam geinsereerd worden; dit impliceert, dat dan een nieuwe variabele s geintroduceerd wordt, die dan nog geen waarde heeft, in de loop van de uitvoering mogelijk een waarde krijgt (in ons geval de waarde van a) maar deze variabele houdt op te bestaan, d.w.z. zijn waarde gaat onherroepelijk verloren, zodra de procedure statement voltooid, althans beeindigd is. Tijdelijk gezien is een locale variabele dus een variabele met een beperkte levensduur: opeenvolgende "incarnaties" van een dergelijke variabele staan geheel los van elkaar.

Het officiele ALGOL 60 rapport zwijgt angstvallig over de mogelijkheid, een ALGOL-programma met behulp van een automatische rekenmachine uit te voeren; aardse zaken als geheugens komen a fortiori niet ter sprake. Met een enkel woord willen wij daarom aanduiden, hoe de MC-vertaler op het locaal zijn van variabelen reageert. Bij elke procedure statement, d.w.z. iedere keer, dat een procedure opnieuw geactiveerd wordt, wordt voor de locale grootheden van deze procedure de nodige geheugenruimte gereserveerd; zodra deze activering van deze procedure beeindigd is, wordt deze reservering ongedaan gemaakt en hetzelfde stuk geheugen wordt in de regel onmiddellijk voor andere doeleinden gebruikt.

Onze procedure "DUB" was louter een afkorting; bij de procedure "wsl" werd het procedure-mechanisme verrijkt met de tijdelijke introductie van hulpgrootheden. Het volgende element, de naamsubstitutie, zullen we aan een uitbreiding van dit eenvoudige voorbeeld laten zien.

FORMELE PARAMETERS

We hebben een programma, waarin een groot aantal variabelen voorkomt en waarin geregeld verwisselingen uitgevoerd moeten worden, maar nu niet altijd tussen hetzelfde tweetal waarden (a en b in ons vorige voorbeeld) maar hier tussen dit paar, ginder tussen een ander paar. ALGOL 60 kent nu de mogelijkheid om in een procedure alleen een mechanisme (hier dus het verwisselen) te definieren, maar in de declaratie van de procedure in het midden te laten, op welke variabelen dit proces zal werken. In plaats daarvan wordt dit dan bij elke procedure statement aangegeven. Dit mechanisme is gegeven in de z.g. "procedure met formele parameters". Een voorbeeld moge dit illustreren.

```
begin    real a, b, c, d, e, ...;
         integer i, ...;
         array s[2:12], ...;
         procedure w2(e,f); real e, f;
         begin real s;
               s:= e; e:= f; f:= s
         end;
         ...; w2(a,b); ...; w2(d,c); i:= 5; ...; w2(s[i], e); ...
end
```

De procedure, genaamd "w2", heeft twee z.g. "formele parameters", in zijn declaratie, "e" resp., "f" gedoopt. Deze namen zijn in de lexicographische zin locaal ten opzichte van het procedure lichaam. Zij hebben als naam van een formele parameter geen betekenis buiten het procedure lichaam; bij gratie van het feit, dat ze in het begin van de procedure declaratie tussen ronde haken als lijst op de procedure naam volgen, fungeren zij in het procedure lichaam als naam van de eerste resp. tweede (algemeen: nde) formele parameter. Heeft de naam, die voor een formele parameter gekozen is, buiten de declaratie al een betekenis, dan blijft deze "buitenbetekenis" over het gebied van de declaratie sluimeren en pas achter de procedure declaratie wordt de buitenbetekenis weer in ere hersteld.

Achter het ronde sluithaakje achter de laatste formele parameter moet een puntkomma volgen. Voordat nu het procedure lichaam begint, is er ruimte voor de z.g. "specificaties". Hoewel van vorm mogelijk niet van declaraties te onderscheiden (als in het bovenstaande voorbeeld), is de specificatie als zodanig herkenbaar door de plaats, waar hij voorkomt. De specificatie "real e, f" behelst, dat de zojuist geintroduceerde formele parameters e en f in het bijbehorende procedure lichaam slechts gebruikt zullen worden zoals men namen van real-gedeclareerde variabelen gebruiken mag.

De dynamische implicatie van het begrip der formele variabelen is geheel anders dan die van de locale variabelen (zoals "real s"). De procedure statement bestaat nu niet enkel uit de naam van de procedure, maar daarachter tussen ronde haken in volgorde de namen (of expressies, zie later) die bij de inserering de (namen van de) formele parameters dienen te vervangen. Populair gezegd fungeert

de formele parameter als open plaats in het procedure lichaam, die bij de procedure statement ingevuld wordt.

In ons voorbeeld bewerkstelligt de procedure statement "w2(a,b)", dat de variabelen genaamd a en b van waarde verwisselen, "w2(s[i], e)" heeft dit effect op de variabelen s[i] en e. In het laatste geval zal de verwisselfunctie dus mede afhangen van de momentane waarde van i. In deze procedure statements fungeren a, b, c, d, s[i] en e als z.g. "werkelijke parameters". De werkelijke parameter geeft aan wat bij de interpretatie van de procedure statement in plaats van de overeenkomstige formele parameter gelezen moet worden.

In de procedure declaratie van w2 worden e en f gespecificeerd, s wordt gedeclareerd. Juist bij variabelen is het verschil tussen declaratie en specificatie heel gemarkeerd: de declaratie bewerkstelligt, dat bij de procedure statement tijdelijk een nieuwe variabele geintroduceerd wordt, de specificatie betreft een formele parameter, die in de procedure statement gekoppeld wordt aan de overeenkomstige werkelijke parameter, d.w.z. iets, wat al bestaat.

Bij de vervanging van de formele parameters door de werkelijke kan een kleine moeilijkheid ontstaan. De procedure statement "w2(d,c)" geeft aanleiding tot de drie assignment statements

"s:= d; d:= c; c:= s"

en dit bewerkstelligt in correcte ALGOL-statements de bedoelde verwisseling. Op dezelfde manier zou "w2(s[i],e)" aanleiding geven tot

"s:= s[i]; s[i]:= e; e:= s"

maar dit zijn geen legitieme ALGOL-statements, want het is verboden in hetzelfde bestek eenzelfde naam in verschillende betekenissen te gebruiken, zoals hier met de naam "s" is gebeurd. Deze moeilijkheid is maar schijn: Hij vloeit voort uit het dubbele gebruik van eenzelfde naam, maar namen mag men vrij kiezen. Als we een ALGOL programma hebben kunnen we zonder meer een equivalent programma opschrijven, dat slechts van het oorspronkelijke daarin verschilt, dat alle benoemde objecten consequent andere namen hebben gekregen. In het bijzonder kunnen we het programma zo opschrijven, dat geen enkele naam ooit in dubbele betekenis gebruik wordt (we hadden de locale variabele, genaamd "s" b.v. kunnen herdopen tot "s uit w2").

De hier gesignaleerde moeilijkheid treedt dan niet meer op. Elke vertaler voor ALGOL 60 heeft dan ook de plicht in dergelijke gevallen te reageren, alsof door herdoping dergelijke "botsingen van namen" (of zelfs dubbelzinnigheden), zoals die zouden kunnen ontstaan bij de interpretatie van de procedure statements, uit de weg zijn geruimd.

Er is nog een andere moeilijkheid. We beschouwen de – onwaarschijnlijk korte – procedure, die gedeclareerd is door:

"procedure Q(u,v); real u, v; u:= v × v" .

Als a, b en c de namen zijn van real-gedeclareerde variabelen, dan zijn mogelijke procedure statements:

"Q(a,b)" met de betekenis "a:= b× b"
"Q(b,10)" met de betekenis "b:= 10 × 10"
"Q(c,a+1)" met de betekenis "c:=(a+1)×(a+1)" en niet met de betekenis "c:= a + 1 × a + 1", waar doorgaans 2 × a + 1 uitkomt.

In het officiele ALGOL 60 rapport is daarom vermeld, dat bij de vervanging van formele door werkelijke parameter de laatste door een rond hakenpaar: () omvat dient te worden, overal, waar dit syntactisch is toegestaan. (Dus b.v. niet links van het wordtteken, maar daar kan het nooit noodzakelijk zijn: bij de bovengenoemde procedure is een statement in de trant van "Q(a+b, 15)" niet toelaatbaar.)

Nu we de formele parameter behandeld hebben, kunnen we aan de eerder genoemde standaard handeling "print(E)" de plaats toekennen, die hem toekomt: dit is een procedure met een parameter, die we zonder voorafgaande declaratie mogen gebruiken.

DE FUNCTIE PROCEDURE

Naast de procedure, die als zelfstandige statement gebruikt moet worden, kent ALGOL 60 een ietwat andere vorm van procedure, nl. een procedure, die in een expressie voor mag komen. Wij zullen dit soort procedures ter onderscheiding "functie-procedure"noemen, en hun gebruik in een expressie een "functie-aanroep".

Voorbeelden van functie-procedures, die zonder voorafgaande declaratie gebruikt mogen worden, hebben wij al gezien: een voorbeeld van een parameterloze functie-procedure is "read", van een functie-procedure met een parameter "sin", "cos" etc. Statements, waarin deze gebruikt worden, als

"bibo:= read; Fidel:= 7 - cos(2 × bibo)"

illustreren, dat de functie-aanroep, die in een expressie voorkomt op een plaats, waar ook een getal syntactisch toelaatbaar is, een (numerieke) waarde representeert. Zolang een expressie alleen getallen en variabelen bevat, impliceert de uitwerking van een expressie, dat in volgorde van links naar rechts de waarden opgehaald worden en dat de erin voorkomende arithmetische operaties uitgevoerd worden, zodra de beide operanden opgehaald, dan wel gevormd zijn. Komt er in een expressie een functie-aanroep voor, dan vragen we "op een ingewikkelde manier" naar een waarde. Het uitwerken van de expressie blijft even rusten, schuift naar het tweede plan; nu wordt de functie-procedure uitgevoerd en als deze voltooid is, is (als nevenactiviteit) een waarde afgeleverd. Met deze waarde wordt het uitwerken van de expressie voortgezet. (Het effect van de functie-aanroep konden we niet zo eenvoudig beschrijven als dat van de procedure statement: ook bij een functie-procedure bestaat het lichaam uit een statement, maar je kunt in ALGOL 60 niet zo maar een statement insereren midden in een expressie.)

De declaratie van de functie-procedure onderscheidt zich in tweeerlei opzicht van de non-functie-procedure. Ten eerste wordt helemaal vooraan het symbool real of integer toegevoegd, ter aanduiding van het type van het afgeleverde resultaat. Ten tweede moet in het procedure lichaam duidelijk zijn, op welk moment het resultaat gevormd is, dat als "als waarde van de functie-procedure" aan de buitenwereld afgeleverd moet worden. Hiervoor is als conventie getroffen, dat de naam van de functie-procedure in zijn eigen lichaam als was het een gewone variabele links van het wordtteken voorkomt. Enige voorbeelden ter illustratie, eerst een parameterloze functie-procedure.

Stel, dat in een programma vaak in expressies de term

"sqrt(x × x + y × y)"

voorkomt, zovaak, dat het de moeite loont, hiervoor een afkorting in te voeren. Kiezen wij voor deze functie van x en y omdat het de voerstraal is de naam "r", dan zijn de relevante declaraties:

```
begin   real x, y, ...;
        real procedure r; r:= sqrt(x × x + y × y);
        ..................
end
```

Omdat de procedure parameterloos is, ontbreekt de specificatie; omdat het procedure lichaam uit een enkele assignment statement bestaat, hebben we de statementhaken begin en end daar niet van node. Komt nu in het programma, dat we hierboven door een rij puntjes hebben aangegeven, de ogenschijnlijk onschuldige assignment statement

"a:= r + 8.134"

voor, dan impliceert dit de berekening van de wortel uit de som van de kwadraten van de waarden, die de variabelen x en y op dat moment hebben.

Het volgende voorbeeld, dat wij ontlenen aan P. Naur, geven wij meer als test voor de lezer of hij de implicaties van de functie-procedure doorgrond heeft, dan als voorbeeld, dat frequente navolging zou verdienen. De functie-procedure

```
"real procedure Sneaky(z); real z;
 begin   Sneaky:= z + (z - 2)↑2;
         W:= z + 1
 end
```

zal, wanneer hij gebruikt wordt, b.v. in de statement

"P:= Sneaky(v - 1) + 2"

bij wijze van spreken "stiekem" de waarde van de variabele W wijzigen. Maar dit heeft tot gevolg, dat het effect van de statement

"Pip:= Sneaky(k) × W"

verschilt van dat van

"Pip:= W × Sneaky(k)" .

Onnodig te zeggen, dat men bij het gebruik van dergelijke constructies de grootst mogelijke voorzichtigheid in acht moet nemen.

Opm. De figuur van een functie-procedure, die bij aanroep een waarde aanneemt, maar daarnaast nog andere wijzigingen teweegbrengt, hebben we eigenlijk ook al lang eerder ontmoet. De standaard handeling "read" heeft immers als neveneffect, dat de volgende keer het volgende getal van de band gelezen zal worden.

Als volgend voorbeeld geven we de functie van twee integers, die gedefinieerd is als de kleinste non-negatieve rest bij deling van de eerste door de tweede; tevens heeft deze functie-procedure een nooduitgang voor het geval, dat de deler = 0 is.

```
"integer procedure MOD(p,q,L); integer p, q; label L;
 begin   integer s;
         if q = 0 then goto L;
         s:= p - p : q × q;
         if s < 0 then MOD:= s + abs(q) else MOD:= s
 end"
```

Een expressie, waarin deze functie-procedure gebruikt wordt, is b.v. in

"if MOD(n ×(2 × n + 1), TRIAL, ALARM) = 0 then ..." .

Hierbij moeten n en TRIAL gedeclareerd zijn als variabelen of procedures van type integer en ergens moet een punt voorkomen, dat de label "ALARM" draagt; zodra deze voorwaarde uitgewerkt wordt op een ogenblik, dat de variabele TRIAL gelijk is aan nul, dan wordt de if clause voor goed verlaten en wordt de berekening voortgezet in het punt, dat door de label ALARM wordt aangewezen. Als een formele parameter, zoals hier de laatste, door de specificatie als label gekwalificeerd is, dan betekent dit, dat de overeenkomstige werkelijke parameter zodanig moet zijn, dat hij als bestemmingsuitdrukking na goto toelaatbaar is; andere mogelijkheden zijn een switch-element en een conditionele bestemmingsuitdrukking (voor beide, zie later).

Wij kunnen de naam van een (functie-)procedure en van een array ook als parameter aan een (functie-)procedure meegeven. B.v.

```
"procedure TABEL(arg, ant, n, fun); integer n; array arg, ant;
           real procedure fun;
 begin     integer i;
           for i:= 1 step 1 until n do ant[i]:= fun(arg[i])
end"
```

Met deze procedure kan men een rij functiewaarden uitrekenen bij een rij argumenten; als x en y lineaire arrays zijn, waarvoor de onderste indexgrens hoogstens 1 en de bovenste minstens 10 is, dan is een mogelijke procedure statement

"TABEL(x, y, 10, sin)"

Hierbij is dus aangenomen, dat de elementen x[1] t/m x[10] een waarde hadden; de overeenkomstige elementen van y worden gelijk gemaakt aan de sinus hiervan.

Wij zien aan dit voorbeeld, dat de specificaties slechts onvolledige informatie over de toelaatbare werkelijke parameters geven. Zo is er niet in vastgelegd, dat de eerste twee werkelijke parameters namen van lineaire arrays moeten zijn, waarvan de grenzen aan enige voorwaarden moeten voldoen; evenmin is er in aangegeven, dat de laatste parameter de naam is van een functie-procedure met een numeriek argument. De specificaties zijn dan ook meer bestemd voor de vertaler, dan voor de gebruiker.

DE VALUE LIJST

De tot nog toe beschreven procedures zijn correct – dat hopen we tenminste – , maar ze zijn niet helemaal realistisch. Teruggrijpend op ons voorlaatste voorbeeld bekijken we eens de functie-aanroep in

"k:= MOD(n ×(2 × n + 1), m ×(m + 7) – 3, m is fout)" .

De formele parameters p en q zijn in deze functie-aanroep bepaald door tamelijk ingewikkelde uitdrukkingen; als wij de tekst van de procedure MOD bekijken, dan zien we, dat we de waarde van p normalerwijze twee keer en die van q drie of vier keer nodig hebben Maar dat impliceert, dat we bij de uitvoering van de bovengenoemde statement de eerste parameter twee en de tweede drie of vier keer uitwerken. Het is kennelijk niet de bedoeling de machine een aantal keren achter elkaar hetzelfde sommetje uit te laten rekenen en in dit opzicht is de functie-procedure MOD dus onrealistisch.

Met de invoering van twee extra locale variabelen is dit te ondervangen:

```
"integer procedure MOD(p,q,L); integer p,q; label L;
 begin   integer s, ploc, qloc;
         ploc:= p; qloc:= q;
         if qloc = 0 then goto L;
         s:= ploc – ploc : qloc × qloc;
         if s < 0 then MOD:= s + abs(qloc) else MOD:= s
end" .
```

Deze "hercodering" komt zovaak voor, dat hiervoor een afkorting is ingevoerd: voor de specificaties komt het nieuwe symbool "value", dan een lijst formele parameternamen. Wij kunnen de nieuwe versie van MOD ook schrijven door in de oude een z.g. value lijst tussen te voegen:

```
"integer procedure MOD(p,q,L); value p,q; integer p,q; label L;
 begin   integer s;
         if q = 0 then goto L;
         s:= p – p : q × q;
         if s < 0 then MOD:= s + abs(q) else MOD:= s
end" .
```

Het effect van de value lijst kan ook als volgt beschreven worden. Voor alle formele parameters, die in de value lijst voorkomen, wordt bij binnenkomst in het procedure lichaam de overeenkomstige werkelijke parameter eenmaal uitgewerkt en de zo verkregen waarde wordt toegekend aan de formele parameter, die dan in het procedure lichaam verder beschouwd wordt als een normale locale variabele. Wij vestigen er nogmaals de nadruk op, dat er door de introductie van het concept "value" niets wezenlijks aan de taal

is toegevoegd: we kunnen het effect ook zonder dat in ALGOL 60 beschrijven. Dat we bij die beschrijving een nieuwe nomenclatuur (ploc en qloc) moeten introduceren, is bij de laatste definitie wat weggesmoesd.

Behalve scalaire formele parameters mogen in de value lijst ook de namen van formele parameters voorkomen, die blijkens de volgende specificatie een array-naam representeren. Als een formele parameter als array gespecificeerd is, dan mag de overeenkomstige werkelijke parameter uitsluitend een naam van een array zijn met het juiste aantal indices. Staat deze formele parameter in de value lijst, dan wordt er een locaal array geintroduceerd met hetzelfde aantal indices en dezelfde indexgrenzen als het array, dat door de werkelijke parameter wordt aangewezen; vervolgens wordt het "buiten-array" element voor element in dit locale array overgenomen. Het effect van een formele array-naam in de value lijst kan dus heel omvangrijk zijn.

Wij laten aan de lezer de verificatie van de volgende bewering over. Indien een als scalar gespecificeerde formele parameter in het procedure lichaam links van het wordteken van een assignment statement staat, is een getal of een expressie (dus "15", "+a", "b/c" of ook iets als "(d)") als overeenkomstige werkelijke parameter niet toegestaan, maar weer wel als de formele parameter in de value lijst voorkomt.

DE GEBONDEN VARIABELE

Een bepaald spitsvondig gebruik van formele parameters die niet in de value lijst zijn opgenomen, dat bij ons weten voor het eerst ontdekt is door J. Jensen, mag niet onvermeld blijven. Ten eerste is het een goede test, of men de implicaties van het begrip "formele parameter" overziet, ten tweede maakt men kennis met een methode van ALGOL-programmering, die inmiddels buitengewoon nuttig is gebleken.

Hierbij spelen twee formele parameters een rol, zeg i en t. De procedure kent in een assignment statement waarden toe aan de formele parameter i, de hieraan beantwoordende werkelijke parameter is dus geen getal of expressie, maar een enkele variabele. De andere formele parameter komt alleen in expressies voor, de overeenkomstige werkelijke parameter mag dus een expressie zijn, mag in het bijzonder een expressie zijn, waarvan de waarde afhangt van de waarde van de variabele, die overeenkomt met de formele parameter i. Maar dit betekent, dat de procedure over de formele parameter t beschikt als een functie van de onafhankelijke variabele i. Enige voorbeelden mogen dit illustreren.

Ten eerste de functie-procedure, die een bepaald aantal termen van een reeks sommeert; deze functie-procedure vervult in wezen de rol van het sommatieteken.

```
real procedure SIGMA(i,h,k,t); value k; integer i,h,k; real t;
begin   real s;
        s:= 0;
        for i:= h step 1 until k do s:= t + s;
        SIGMA:= s
end
```

We hadden h in de value lijst op kunnen nemen, maar dat heeft uit efficiency overwegingen geen zin, omdat de beginwaarde van de lopende variabele in de for-statement maar een keer opgehaald wordt; anders ligt dat voor de eindgrens k, want bij iedere doorgang door de cyclus moet onderzocht worden, of de eindgrens nog niet overschreden is. De functie-procedure SIGMA "sommeert de term t voor i met 1 oplopend van h tot en met k".

Zo is de expressie

```
SIGMA(n,1,10,H[n])
```

gelijk aan de som van de array-elementen H[1], H[2], ... t/m H[10]. Immers de naamsubstitutie resulteert in het programma:

```
begin   real s;
        s:= 0;
        for n:= 1 step 1 until 10 do s:= H[n] + s;
        SIGMA: s
end
```

en dit programma-onderdeel geeft kennelijk SIGMA de gewenste waarde.

Bij passende declaratie is

SIGMA(k,1,20,A[k,k])

het spoor van de matrix A van 20 bij 20 en is

SIGMA(t,1,20,A[r,t] × B[t])

het scalair product van de derde rij (of kolom) van A met de vector B. De som van alle elementen van de matrix A is dan

SIGMA(k,1,20,SIGMA(r,1,20,A[r,k])) .

Wie bij dit laatste voorbeeld het resultaat van inserering en naam-substitutie opschrijft, zal merken, dat door de speciale vorm van de laatste werkelijke parameter de "vertaling" van het procedure lichaam de procedure zelf weer aanroept. De MC-vertaler accepteert dergelijke "geneste" activeringen van eenzelfde procedure zonder meer.

Wij hadden het procedure lichaam ook anders kunnen formuleren, z.g. recursief, d.w.z. dat in de declaratie de procedure zelf al gebruikt wordt. Ook dit accepteert de MC-vertaler zonder bezwaar.

```
real procedure SIGMA (i,h,k,t); value h; integer i,h,k; real t;
begin   if k < h then SIGMA:= 0
                 else
        begin   i:= h;
                SIGMA:= t + SIGMA(i,h+1,k,t)
        end
end
```

In de laatste assignment statement wordt de som geschreven als de eerste term plus de som van de resterende termen; men lette erop, dat we hier de volgorde van de twee addenda niet om mogen keren. De term "t" staat hier voor de eerste term en is dit ook, zolang i de beginwaarde heeft, maar de activering van SIGMA wijzigt in de regel de waarde van i. Hier hebben wij dus weer een voorbeeld van het bij "Sneaky" gesignaleerde verschijnsel. Eerlijkheid gebiedt ons er de aandacht op te vestigen dat bij de functie-procedure SIGMA in zijn laatste vorm de uitvoering een hoeveelheid geheugen in beslag neemt, die evenredig toeneemt met het aantal termen, dat gesommeerd moet worden.

HET BLOK

Terloops hebben wij bij de procedure declaratie het concept "blok" genoemd; dit is het laatste belangrijke begrip, dat nog behandeld moet worden. Het betreft de macroscopische structuur van een ALGOL-programma.

Wij zijn de statementhaken begin en end eigenlijk in twee functies tegengekomen. Ten eerste als hakenpaar om een aantal statements, om te zorgen, dat deze groep statements van buiten beschouwd konden worden als een enkele statement. Ter onderscheiding van de enkelvoudige statement noemt men een dergelijk rijtje statements, dat met behulp van de statementhaken "tot een geheel" verklaard wordt, wel een z.g. samengestelde statement. Wij hebben de samengestelde statement ontmoet eerst bij de if clause, daarna bij de for clause en tenslotte bij de procedure declaratie ter definitie van het procedure lichaam. In het laatste geval hebben we echter gezien, dat tussen het symbool begin en de eerste daarop volgende statement nog declaraties voor mochten komen, en dat deze declaraties slechts betrekking hadden op het stuk tekst tot aan de bijbehorende end. Een nevenfunctie van de statementhaken kan dus zijn het aangeven, over welk stuk tekst bepaalde declaraties van toepassing zijn; deze nevenfunctie vinden we in wezen ook bij het alomvattende hakenpaar, dat aangeeft, waar het programma begint en waar het weer eindigt.

Elke samengestelde statement, waarin tussen de openings-begin en de eerste daarop volgende statement een of meer declaraties voorkomen, heet een blok; omgekeerd moeten declaraties altijd aan het begin van een blok staan. Dit impliceert dat een declaratie altijd volgt op het symbool begin of op een andere declaratie (en daar dan van gescheiden door een puntkomma).

Locaal ten opzichte van een blok zijn

a) alle namen, die aan het begin ervan gedeclareerd zijn,
b) alle eventuele labels, die statements in het blok labelen,
c) alle formele parameters van de procedure (alleen van toepassing, als het blok een procedure lichaam is).

De lexicographische consequenties van de localiteit van een naam zijn:

a) dat het object, dat bij deze naam hoort, buiten dit blok ophoudt te bestaan,
b) dat naar een ander object, dat buiten dit blok dezelfde naam draagt, binnenin dit blok directe referentie niet meer mogelijk is.

Doordat ook labels per definitie locaal zijn ten opzichte van het blok, waarvan de door hen gelabelde statement deel uitmaakt, en die labels van buiten het blok dus ontoegankelijk zijn, is het onmogelijk om van buiten een blok via een goto-statement een blok binnen te komen: de enige manier om een blok binnen te komen, is via zijn openingshaak begin en dan passeert men dus alle bij het blok behorende declaraties.

(In overeenstemming hiermee beschouwt de MC-vertaler elk procedure lichaam, ongeacht aanwezigheid van locale declaraties, en bovendien elke for-statement, waar je immers ook niet via een goto-statement van buiten binnenin mag springen, als een blok.)

Nu wij het concept blok genoemd hebben los van zijn meest voorkomende vorm, het procedure lichaam, kunnen wij het effect van de procedure statement iets scherper beschrijven; we zullen dit doen aan de hand van de destijds genoemde verwisselingsprocedure "wsl", gedeclareerd door:

```
"procedure wsl;
 begin   real s;
         s:= a; a:= b; b:= s
end"
```

Komt nu in de tekst van het programma de procedure statement "wsl" voor, b.v.

"S1; wsl; S2"

dan moeten we in plaats van de procedure statement niet alleen de drie statements van het lichaam insereren, maar het hele lichaam van begin tot end. We krijgen dan

```
"S1;
 begin   real s;
         s:= a; a:= b; b:= s
 end;
 S2"
```

Het effect van de procedure statement is hiermee onder gebruikmaking van een "binnenblok" in legitieme ALGOL 60 beschreven.

Er is een geval, waarin men het concept van het blok heel beslist nodig heeft, nl. zodra men opereren moet op arrays, waarover de indexgrenzen niet van tevoren vaststaan. In onze voorbeelden van array-declaraties zijn de grenzen altijd gehele getallen (eventueel voorzien van teken) geweest; in het buitenste blok - het "hoofdprogramma" - is dit voor de MC-vertaler de enige toelaatbare vorm voor indexgrens. Bij binnenblokken mogen de indexgrenzen echter willekeurige uitdrukkingen zijn, die af mogen hangen van variabelen, mits die variabelen op moment van binnenkomst in het blok een welgedefinieerde waarde hebben. De indexgrenzen worden dan eenmaal uitgewerkt en de aldus gevonden indexgrenzen blijven van kracht voor deze activering van het blok, ook al verandert een van de bepalende variabelen in de loop van dit blok van waarde.

Veronderstel, dat we een programma moeten maken dat opereert op twee vectoren A en B, waarvan de waarden van de elementen achter elkaar op een band geponst staan, en wel eerst de elementen van A en dan de elementen van B. Omdat de lengte van de vectoren kan varieren, wordt elke vector op de band voorafgegaan door zijn eigen lengte. De structuur van dit programma kan dan als volgt zijn.

```
begin   integer lengte A, lengte B, k;
        "en alle andere declaraties, maar niet die voor de een-
         dimensionale arrays A en B"
         lengte A:= read;
         begin   array A[1:lengte A];
                 for k:= 1 step 1 until lengte A do
                                      A[k]:= read;
                 lengte B:= read;
                 begin   array B[1:lengte B];
                         for k:= 1 step 1 until lengte B do
                                          B[k]:= read;
                         "en nu in dit binnen-binnenblok het
                          eigenlijke programma"
                 end
        end
end
```

DYNAMISCHE ARRAYS

In de array-declaraties van het buitenste blok kunnen de indexgrenzen niet van variabelen afhangen; zij zijn dus constant en de MC-vertaler accepteert hier alleen gehele getallen, eventueel voorzien van teken. In de array-declaraties van binnenblokken zijn willekeurige expressies toegestaan: levert de uitwerking van een dergelijke expressie een resultaat van type real af, dan wordt dit weer op normale manier afgerond tot het dichtstbij gelegen gehele getal. Ook bij deze arrays geldt (na eventuele afrondingen), dat voor elke index de bovengrens niet kleiner mag zijn dan de bijbehorende ondergrens.

Declaraties van scalairen en arrays aan het begin van een binnenblok (aan het begin van het buitenste blok is het zinloos) kunnen voorafgegaan worden door het extra symbool "own". Dit heeft tot effect, dat bij verlating van het blok de waarden van deze variabelen niet, zoals bij gewone locale variabelen, verloren gaan, maar behouden blijven en onder hun eigen namen weer ter beschikking staan bij de eerstvolgende activering van dit blok. Dit concept is in de MC-vertaler slechts gedeeltelijk geincorporeerd. Ten eerste zijn bij de own-gedeclareerde arrays als index-grenzen slechts gehele getallen toegestaan (dus net als bij de arrays, die in het buitenste blok gedeclareerd worden). Ten tweede wordt er voor elke own-gedeclareerde variabele (scalar of array element) slechts een waarde tegelijkertijd onthouden; dit is niet in overeenstemming met de officiele regels van ALGOL 60, zodra een procedure, waarin own-gedeclareerde variabelen voorkomen, recursief gebruikt wordt.

DE CONDITIONELE EXPRESSIE

Naast de conditionele statement kent ALGOL 60 de z.g. conditionele expressie. Hier bepaalt het al of niet vervuld zijn van een voorwaarde niet de keuze tussen twee statements, maar tussen twee expressies; het geheel speelt weer de rol van een expressie.

Het belangrijkste verschil met de conditionele statement is wel, dat bij de conditionele expressie else verplicht is. (Dit vloeit voort uit het feit, dat ALGOL geen "lege expressie" kent, maar wel een "lege statement": doe niets en ga door.)

Bij de conditionele expressie geldt een verbod, analoog aan de conditionele statement. De expressie tussen then en else mag niet zelf een conditionele expressie zijn. Hieraan kan men altijd voldoen door de insertie van haken, maar dan natuurlijk de expressiehaken "(" en ")".

Ter illustratie zullen wij weer de variabelen MAX gelijk maken aan de grootste van de drie variabelen a, b en c. We zullen dit doen met het minimum aantal haakjes.

```
"MAX:= if a < b then (if b < c then c else b)
                else if a < c then c else a"  .
```

Wij adviseren echter een ieder, die conditionele expressies gebruikt, niet te karig te zijn met haakjes. Zo mag men in plaats van

```
"y:= abs(x)"
```

ook schrijven

```
"y:= if x < 0 then - x else x"  ,
```

maar voor

```
"y:= abs(x) + 1"
```

moet men op dezelfde manier schrijven....

```
"y:= (if x < 0 then - x else x) + 1"  ,
```

want had men de haakjes weggelaten, dan had men effectief

```
"y:= if x < 0 then (- x) else (x + 1)"
```

opgeschreven. Ook hier doet men er dus verstandiger aan het niet op de prioriteitsregels aan te laten komen.

DE LOGISCHE VARIABELE

Beschouwen wij een stuk programma, waarin twee variabelen, zeg a en b, niet van waarde veranderen. Als nu in dit stuk berekening een bepaalde functie van deze twee variabelen, zeg a/b, geregeld voorkomt, dan kunnen we dit programma eenvoudiger opschrijven. We declareren aan het begin van het blok in kwestie een nieuwe hulpgrootheid, zeg qab, door

"real qab"

en lassen, zodra de variabelen a en b de gewenste waarde hebben, de assignment statement

"qab:= a / b"

in. Van dit punt af kunnen we (zolang a en b niet van waarde veranderen) overal, waar het quotient "a / b" als (deel-)uitdrukking voorkomt, dit quotient vervangen door de simpele variabele qab. (Deze verkorting haalt natuurlijk des te meer uit, naarmate de uitdrukking, waarvoor we hier een quotient gekozen hebben, ingewikkelder is.) Deze verkorting is een van de wezenlijke functies van de assignment statement. (Een andere is, dat men op de tweede manier a of b van waarde kan laten veranderen, zonder de beschikking over het quotient van de "oorspronkelijke" waarden te verliezen.)

Veronderstel nu, dat in een dergelijk stuk programma waarin a en b niet van waarde veranderen, niet een arithmetische uitdrukking, maar wat we een voorwaarde genoemd hebben, herhaaldelijk voorkomt, b.v.

"a ↑ 3 < b ↑ 5" .

Iedere keer, dat deze voorwaarde een rol speelt (b.v. tussen de symbolen if en then van een conditionele statement of expressie) impliceert dit twee machtsverheffingen. Het is natuurlijk veel aanlokkelijker om, zodra de waarden van a en b bekend zijn, eens en vooral vast te stellen, of aan deze voorwaarde voldaan is. Om een dergelijk gegeven vast te kunnen leggen, is in ALGOL 60 naast de beide types arithmetische variabelen een derde type variabele ingevoerd, de z.g. "logische variabele". Om nu de analoge verkorting in te voeren, declareren we aan het begin van het blok een nieuwe logische variabele, zeg vab, met behulp van de declaratie

Boolean vab"

en lassen, zodra a en b de betrokken waarden hebben aangenomen, de assignment statement

"vab:= a ↑ 3 < b ↑ 5"

in. Van dit punt af kunnen we (natuurlijk zolang a en b niet van waarde veranderen) de voorwaarde vervangen door de simpele logische variabele vab; we schrijven dan...

"if vab then..."

Hiermee hebben wij bereikt, dat de twee machtsverheffingen niet vaker dan nodig worden uitgevoerd.

Bij de declaratie van de logische variabele hebben wij kennis gemaakt met een nieuw symbool, nl. het symbool "Boolean"; de MC-vertaler accepteert voor dit symbool nog een tweede representatie, nl. "boolean", een versie, die op de Flexowriter iets aantrekkelijker is om te ponsen. (Dit symbool begint met een hoofdletter, omdat de logische variabelen genoemd zijn naar George Boole, auteur van het in 1853 verschenen werk, getiteld "An Investigation of the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities.") Wij zullen het verder met een kleine letter schrijven.

Als links van het wordtteken de naam van een boolean-gedeclareerde variabele staat, moet rechts ervan een z.g. "logische uitdrukking" staan. Wij hebben zojuist in een assignment statement en vroeger in de if clause voorbeelden van logische uitdrukkingen gezien. Voor het vormen van ingewikkeldere logische uitdrukkingen kent ALGOL 60 een rijke schakering van mogelijkheden, waarvoor wij de lezer verwijzen naar het officiele rapport, sectie 3.4 BOOLEAN EXPRESSIONS.

Nog een enkel woord over de waarde van een logische variabele of uitdrukking. Een logische variabele heeft nl. evengoed een waardebereik als de arithmetische variabele. De arithmetische variabele heeft een waardebereik met oneindig veel elementen, die we met de decimale cijfers etc. kunnen aanwijzen, het waardebereik van de logische variabele telt slechts twee elementen, omdat aan een relatie voldaan is, of niet; ter indicatie van die waarden heeft men de beschikking over de symbolen "true" en "false".

Enkele voorbeelden willen we de lezer toch niet onthouden.

De logische operator "$\wedge$"(uit te spreken "en") verbindt twee logische uitdrukkingen; het resultaat heeft slechts de waarde true, als beide operanden deze waarde hebben. Aan de voorwaarde

$$"0 < x \wedge x \leq 1"$$

is dus slechts voldaan als x ligt in het gebied $0 < x \leq 1$.

De voorwaarde, dat x buiten dit gebied ligt, kan men aangeven met behulp van de logische operator "$\vee$" (uit te spreken "of"). Deze kan eveneens twee logische uitdrukkingen verbinden, het resultaat heeft de waarde true als minstens een van beide operanden deze waarde heeft. De voorwaarde, dat x buiten het gebied $0 < x \leq 1$ ligt, wordt dus gegeven in de logische uitdrukking

$$"x \leq 0 \vee 1 < x" \quad .$$

We hadden dit ook aan mogen geven met behulp van de negatie "$\neg$" (uit te spreken "non"), die werkt op een logische uitdrukking. De voorwaarde, dat x buiten het gebied $0 < x \leq 1$ ligt, wordt dan

$$"\neg(0 < x \wedge x \leq 1)" \quad .$$

DE SWITCH DECLARATIE

Terloops hebben wij al een keer de switch genoemd; dit is de laatste soort object, dat nog in ALGOL 60 voorkomt. Een switch is niet anders dan een eendimensionaal array, waarvan de elementen geen getallen maar "bestemmingen" zijn (in het eenvoudigste geval labels). Indexgrenzen hoeft men hierbij niet aan te geven, want de afspraak is, dat men de elementen nummert, te beginnen bij 1 en de bovengrens is impliciet gegeven, doordat de switch-declaratie eindigt met een opsomming van de elementen.

De switch-declaratie bestaat uit het symbool "switch", gevolgd door de voor de switch gekozen naam, dan (helaas) het wordtteken, en daarachter de lijst van bestemmingen, onderling gescheiden door een komma.

Als in het blok, waarin de hieronder staande declaratie voorkomt, de namen AA, BB, CC en DD de betekenis van labels hebben, dan kan de declaratie van de switch genaamd SWITCH b.v. luiden:

"switch SWITCH:= AA, BB, if i = k then CC else DD, SWITCH[abs(i-k)]"

Staat nu ergens in dit blok

"goto SWITCH[E]" ,

dan zal het effect van deze goto-statement afhangen van de waarde van de expressie E. (We nemen in het volgende aan, dat de waarde van de expressie E van type integer is; zou hij van type real zijn, dan zou zijn voorkomen als index toch afronding op de dichtstbijgelegen gehele waarde geimpliceerd hebben.) Het effect van deze goto-statement is alleen maar gedefinieerd, als de waarde van E = 1, 2, 3 of 4 is. Indien E = 1 is, dan is het effect gelijk aan dat van

"goto AA" ,

indien E = 2 is, dan is het effect gelijk aan dat van

"goto BB" ,

indien E = 3 is, dan is het effect gelijk aan dat van

"goto if i = k then CC else DD" ,

wat (zie sectie 3.5. DESIGNATIONAL EXPRESSIONS, van het officiele rapport) op zijn beurt equivalent is aan

"if i = k then goto CC else goto DD" .

Is tenslotte E = 4, dan is het effect gelijk aan

"goto SWITCH[abs(i - k)]" ;

wat gebeurt er, als dan blijkt, dat abs(i - k) = 4 is?

Voor verdere details verwijzen we naar sectie 5.3 SWITCH DECLARATIONS, uit het officiele rapport. Wij voegen hier slechts aan toe, dat de onder 5.3.5. aldaar genoemde restrictie voor de MC-vertaler niet geldt.

DE MC-VERTALER VOOR DE X1

volledigheidshalve geven wij nu een lijst van speciale eigenschappen van de MC-vertaler voor de x1,

1) Als indexgrenzen in array-declaraties in het buitenste blok of voorafgegaan door het symbool "own" zijn slechts gehele getallen, eventueel voorzien van teken, toegestaan.

2) Procedures, waarin door het symbool "own" gemarkeerde declaraties voorkomen, functioneren bij recursief gebruik niet op de officiele wijze.

3) De MC-vertaler maakt bij de declaratie van de functie-procedure geen onderscheid tussen "real" en "integer" als eerste symbool: bij elke aanroep wordt het type van het antwoord bepaald door de ditmaal in het procedure lichaam uitgevoerde arithmetiek.

4) Declaraties aan het begin van een blok en specificaties in de procedure-declaratie moeten gegeven worden in de volgorde:

   1: scalairen
   2: arrays
   3: bestemmingen
   4: procedures .

5) Van namen tellen alleen de eerste negen symbolen mee.

6) Labels, die beginnen met een cijfer, zijn niet toegestaan.

7) Voor getallen, die in de tekst van een ALGOL-programma voorkomen, gelden de volgende regels:
   Het getal nul wordt altijd als van type integer opgevat, dus ook, wanneer er een punt in voorkomt of men het numerieke gedeelte = 0 nog door een exponent laat volgen.
   Een getal, dat wegens het ontbreken van punt en exponent volgens de regels van type integer is, wordt echter als van type real behandeld, zodra de absolute waarde groter is dan 67108863.
   De decimale exponent mag in absolute waarde niet groter zijn dan 600.

8) Een array in de value lijst mag hoogstens vijf indices hebben.

9) In de procedure-declaratie eist de MC-vertaler voor elke formele parameter een specificatie.

10) Behoudens expliciet verbod voor bepaalde procedures, mag op de plaats van een formele parameter, gespecificeerd van type real, in de procedure statement een werkelijke parameter van type integer (of omgekeerd) worden meegegeven.

11) De waarde van de speciale functie abs is van hetzelfde type als het meegegeven argument.

12) Functie-procedures mogen in ALGOL 60 behalve in expressies ook als **zelfstandige** statements aangeroepen worden. Er bestaat dan geen interesse in de waarde van de functie-procedure, die dan verder buiten beschouwing blijft. Voor de eerder genoemde speciale functies is dit bij de MC-vertaler niet toegestaan. (Men mag dus niet drie gatallen op de band overslaan door:

"read; read; read" .)

13) De "primaries" van een uitdrukking worden in volgorde van links naar rechts uitgewerkt. (Zie boven, bij het voorbeeld van "Sneaky"; wij vermelden dit met zoveel woorden, omdat het officiele rapport het wel suggereert, maar niet expliciet definieert.)

14) Het is niet toegestaan, dat een blok lexicographisch door meer dan 30 blokken omvat wordt. (Hierbij gelden for-statements en procedure lichamen als een nieuw blok.)

15) Parameters in de value lijst worden bij binnenkomst in het procedure lichaam uitgewerkt in volgorde van specificatie. (Dit is van belang, als de uitwerking van de ene werkelijke parameter de waarde van een andere kan beinvloeden.)

16) In plaats van het symbool "Boolean" accepteert de MC-vertaler ook het symbool "boolean"; evenzo "goto" in plaats van "go to".

17) Symbolen, die met behulp van onderstreping worden weergegeven, moeten, indien ze onmiddelijk op elkaar volgen, in de door de Flexowriter uitgetypte tekst door een of meer spaties (of door Tab.), dan wel de overgang op een nieuwe regel van elkaar zijn gescheiden.

18) In de symbolenrij, die na end geskipt wordt, zijn de symbolen "begin", "comment" en de string quotes "⊀" en "⊁" niet toegestaan.

19) Niet alleen de waarde van de lopende variabele van een for clause maar ook de variabele zelf (nl. als deze een geindiceerde variabele is) kan in de statement, volgend op de for clause gewijzigd worden. In de expressies, die voorkomen in de lijstelementen (tussen for en do) vermijde men echter de aanroepen van functieprocedures, die op slinkse wijze de waarde van de lopende variabele, of deze variabele zelf wijzigen.

20) Na de laatste end van het programma accepteert de MC-vertaler geen te skippen symbolen meer.

21) De beperkingen, genoemd in sectie 5.3.5 en de laatste zin van sectie 4.6.5 van het officiele rapport, zijn op de MC-vertaler niet van toepassing.

22) Restrictie 4.7.6 van het officiele rapport is niet van toepassing op de MC-vertaler. Non-localen van het procedure lichaam behouden bij inserering ten gevolge van een procedure-statement de betekenis die zij hebben bij de procedure-declaratie.

SPECIALE INPUT-OUTPUT PROCEDURES

Voor de MC-vertaler staan op het moment van schrijven de volgende procedures zonder declaratie ter beschikking van de programmeur:

"read" is een functie-procedure, waarvan de waarde gelijk is aan die van het eerstvolgende getal op de band; het type hangt af van het aanbod. Deze procedure mag alleen uitgevoerd worden, mits er een getalband in de bandlezer van de X1 ligt. Als dit (wat normaliter het geval is) een 7-gats band in de MC-Flexowritercode is, gelden hiervoor de volgende pons-conventies:

a) De getallen zijn getallen in de normale ALGOL-notatie, al dan niet voorafgagaan door een teken. In scherp contrast met de regels van ALGOL is de lay-out echter aan beperkingen gebonden (zie b en c).

b) Scheiding tussen twee opeenvolgende getallen dan wel afsluiting van het voorafgaande getal, wordt bewerkstelligd door:

1) het teken van het volgende getal ("+" of "-"), of
2) tabulatie of ten minste twee spaties na een cijfer, of
3) overgang op een nieuwe regel, of
4) de komma ",", of
5) commentaar (d.w.z. een reeks ALGOL-symbolen tussen apostrophes), of
6) de zg. brokscheider "?" (zie onder).

c) Een enkele spatie tussen symbolen bewerkstelligt dus geen getalscheiding; hetzelfde geldt voor meer spaties of een tabulatie aansluitend op een teken, omdat dan in elk geval nog een cijfer moet komen.

d) Aan de absolute waarde van de exponent is geen beperking opgelegd; bovendien heeft men de mogelijkheid het getal

"inf"

te ponsen, al dan niet voorzien van een teken. De absolute waarde van de overeenkomstige function designator "read" is dan gelijk aan de maximale absolute waarde van een real-gedeclareerde variabele.

e) De band moet beginnen met een stuk "Tape feed" - opdat de band in de bandlezer gelegd kan worden - en hij moet eindigen met een stuk "Tape feed" - opdat de band correct door de bandlezer gevoerd wordt, zolang er nog significante ponsingen gelezen worden-. Midden op de band is eveneens overal extra "Tape feed" toegestaan (zie echter f); extra stukken "Tape feed" worden door de procedure "read" geskipt.

f) Na "Tape feed" wordt de eerste case-afhankelijke ponsing een hernieuwde case-definitie geeist. Is hieraan niet voldaan, dan wordt de getalband door de procedure "read" geweigerd (de X1 stopt).

g) Erase wordt overal geskipt.

Aan het einde van een getalband moet de zg. brokscheider "?" staan. Dit is een vereiste ponsing; wij hebben hiervoor met opzet een non-ALGOL-symbool gekozen, omdat de aan- of afwezigheid van de brokscheider door het verwerkende ALGOL-programma niet gedetecteerd kan worden. De brokscheider "?" bewerkstelligt, dat het proces van bandlezen, dat door de eerste activering van de procedure "read" in gang is gezet en "autonoom" doorgaat, tot nader order wordt beeindigd. Deze "nadere order" is de aanvraag (via "read") van een getal, nadat alle getallen, die voor de brokscheider op de band staan, door het ALGOL - programma verwerkt zijn. Als slechts een stukje van een getalband ingelezen moet worden, moet men deze stukjes dus door een brokscheider afsluiten. Omissie van het vraagteken aan het einde van de band zou tot gevolg hebben, dat het autonoom werkend bandleesproces niet zou aflopen en dat dientengevolge de band uit de bandlezer zou lopen, iets wat vermeden dient te worden, als men prijs stelt op de correcte voltooiing van het programma.

Voorts mag op de band tussen twee getallen commentaar voorkomen, bestaande uit een reeks ALGOL-symbolen, voorafgegaan en afgesloten door een apostrophe. Dit commentaar wordt door de procedure "read" overgeslagen.

Het is gewenst om op een handcodeband achter de laatste brokscheider een stopcode te ponsen. Deze wordt door de X1 dan niet meer gelezen, maar de stopcode bewerkstelligt, dat bij uittypen van de band (bv. voor controle-doeleinden) de Flexowriter aan het einde gestopt wordt.

Behalve banden, die als boven in Flexowritercode zijn geponst, accepteert de procedure "read" eveneens vijfgatsbanden, geponst op de normale X1-handcodeponser. Het ponsen van deze vijfgatsbanden geschiedt volgens dezelfde conventies, met dien verstande, dat commentaar niet is toegestaan en

a) "Tape feed" wordt vervangen door "roffel S"
b) Spatie wordt vervangen door "U"
c) Tab wordt vervangen door "T"
d) Wagen terug wordt vervangen door "W"
e) "␣" wordt vervangen door "E"
f) "inf" wordt vervangen door "Y"
g) de komma wordt vervangen door "K" en
h) de brokscheider wordt vervangen door "B".

(De lay-out symbolen hebben om der wille van de volledigheid op vijfgatsband een representatie gekregen; waarschijnlijk zal men in dit geval getalscheiding altijd door het teken van het volgende getal bewerkstelligen.)

"stop" is een procedure, die de X1 doet stoppen, b.v. om de operateur gelegenheid te geven, een band in te leggen. Met de toets "BVA" kan de berekening voortgezet worden.

"print(E)" is een procedure, die de waarde van de expressie E op de schrijfmachine uittypt. Is de expressie van type integer, dan wordt een geheel van hoogstens acht decimalen uitgetypt, waarbij non-significante nullen door spaties vervangen worden.
In geval van type real wordt in drijvende komma notatie het breukgedeelte in de rekenprecisie uitgetypt, gevolgd door decimale exponent; in geval de modulus van de exponent groter dan 600 zou zijn, wordt de integer "0" of "inf." uitgetypt. In alle gevallen wordt het getal voorafgegaan door een teken en staat de wagen van de schrijfmachine na afloop aan het begin van de volgende kolom.
Er kunnen maximaal 7 kolommen op een regel getypt worden.

"TAB" is een procedure die bewerkstelligt, dat er een kolom overgeslagen wordt.

"NLCR" is een procedure, die de schrijfmachine naar het begin van de volgende regel stuurt.

"SPACE(E)" is een procedure, die slechts één argument van type integer accepteert; de waarde van E geeft aan, hoeveel spaties er gegeven worden.