RA

stichting
mathematisch
centrum

MC

REKENAFDELING CR 22/71 JUNI

W.P. DE ROEVER

RA

DE PROGRAMMEERTAAL ALGOL 60 AAN DE HAND VAN HET REVISED REPORT

Syllabus bij het Oriënterend Colloquium
Informatica 1970-1971

2e boerhaavestraat 49 amsterdam

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

Inhoud

De programmeertaal ALGOL 60 aan de hand van het Revised Report.

W.P. de Roever.

I. Inleiding.

Voordat tot de bespreking van ALGOL 60 en zijn beschrijving in RR (het Revised Report on the algorithmic language ALGOL 60) wordt overgegaan, volgen enige korte opmerkingen over programma verwerking en rekenmachines.

Globale beschrijving van programma verwerking.

(i) Het programma, eventueel met de bijbehorende gegevens, geschreven in een aan de machine bekende programmeertaal, dus in deze cursus ALGOL 60, wordt gecodeerd.
Dit kan gebeuren op "ponsbanden" of "ponskaarten".

(ii) Teneinde programma plus gegevens de rekenmachine binnen te krijgen, passeren de ponsbanden, waarop programma plus gegevens gecodeerd zijn, de lichtgevoelige cellen en de lichtbundel van de "lezer". Aangezien wel een gaatje in de band meer licht doorlaat dan niet een gaatje, kan de codering zodoende in het geheugen van de rekenmachine worden overgenomen. Dit proces heet het "inlezen" van het programma en zijn gegevens.

(iii) Het programma wordt "geëxecuteerd" door de rekenmachine.

(iv) De resultaten van de executie van het programma worden, indien daartoe expliciet in het programma opdracht toe gegeven is, de rekenmachine "uitgevoerd" bv. afgedrukt door de "regeldrukker". Bovendien wordt het programma zelf nog eens afgedrukt.

In nauw verband hiermee is een rekenmachine in de volgende functionele eenheden te splitsen:

(i) "Invoerorganen", die de invoer van programma plus gegevens, de "input", verzorgen,

(ii) Het "geheugen" dat dient (a) Voor de opslag van programma plus gegevens,
(b) Voor het vastleggen van de voor de executie van het programma benodigde tussenresultaten.

(iii) Het "centrale rekenorgaan", dat alle arithmatische operaties, zoals optelling, aftrekking, vermenigvuldiging en deling, uitvoert.

(iv) "Uitvoerorganen", die het vanuit het geheugen naar de buitenwereld brengen van eindresultaten, de "output", verzorgen.

I.1 Bespreking van een programma.

Als voorbeeld fungeert het volgende programma met bijbehorende gegevens.

```
begin integer a1, a2, antwoord;
      a1 := read; print(a1);
      a2 := read; print(a2);
      antwoord := a1+a2; print(antwoord);
      antwoord := a1-a2; print(antwoord)
end
```

"band met gegevens:" 14 7

In dit programma worden twee getallen van de getallenband gelezen door executie van "a1 := read" en "a2 := read", ze worden afgedrukt door executie van "print(a1)" en "print(a2)", hun som en verschil worden door het centrale rekenorgaan berekend en aan de "naam" antwoord toegekend door executie van "antwoord := a1+a2" en "antwoord := a1-a2" en tenslotte worden de resultaten van deze twee berekeningen afgedrukt door executie van "print(antwoord)".

Vraag: wat is de output van dit programma?

I.2 De basic symbols en identifiers.

Allereerst kan de lezer opvallen dat enkele woorden in dit programma onderstreept zijn nl. begin, integer en end. Dit zijn basissymbolen van ALGOL 60, "basic symbols". Andere basic symbols, die in dit programma voorkomen, zijn:

a,1,2,n,t,w,o,r,d, het kommasymbool, ,;,e,p,i,(,), + en -.

Zo'n onderstreept woord moet opgevat worden als een enkel symbool, een nieuwe letter, net zoals het $\sum$- en $\int$ - teken uit de wiskunde taal of de letters van het alfabet.

De definitie van "basic symbol" staat op blz. 13, sectie 2 van het RR. Beschouw eerst sectie 2.2.1 (secties beginnend met een arabisch cijfer zullen altijd naar secties van het RR verwijzen).
Hierin treft men, binnen het kader van ALGOL 60, de volgende definitie van "digit" (cijfer) aan, die vermoedelijk niet van de U bekende verschilt:

<digit> ::= 0|1|2|3|4|5|6|7|8|9 .

Deze definitie kan men als volgt interpreteren:

<digit> als "de verzameling van alle (syntactische) eenheden van ALGOL 60 waarop het predikaat "digit" van toepassing is",
::= als "bestaat uit"
0|1|2|3|4|5|6|7|8|9 als de opsomming van die verzameling (<digit>) naar zijn elementen.

Zie voor een analoog geval de definitie van "letter", die wel van de bekende definitie afwijkt, nl. "a" en "A" zijn verschillende "letters".

Een "basic symbol" van ALGOL 60 wordt nu door de eerste definitie van sectie 2 gespecificeerd:

<basic symbol> ::= <letter> | <digit> | <logical value> | <delimiter>,

die als volgt kunnen worden geïnterpreteerd:
" de verzameling van alle eenheden van ALGOL 60 waarop "basic symbol" van toepassing is, bestaat uit
de verzameling van alle "letters" (zie 2.1 voor definitie), verenigd met ,"|"
de verzameling van alle "digits" (zie 2.2.1), verenigd met
de verzameling van alle logische waarden, "logical values", (zie 2.2.2) verenigd met
de verzameling van alle "delimiters" (zie 2.3).

Deze wijze van definiëren is vergelijkbaar met regels als A = B ∪ C. Vraagt men zich af welke verzameling A is, dan verlegt A = B ∪ C het probleem naar de aard van B en C.
Voor de volledigheid: < en > heten "metahaakjes".

De vraag of begin een "basic symbol" is, d.w.z. tot de verzameling van "basic symbols" behoort, wordt door bovenstaande regel verlegd naar de vraag of begin een "letter", "digit", "logical value" of "delimiter" is. In 2.3 van het RR staat dan dat een "delimiter" een "bracket" kan zijn en onder de verzameling van "brachets" komt begin voor. Het is dus een "basic symbol".

Vraag: zijn de volgende symbolen basic symbols?

:= , ::= a ab < jan

Als namen treft U in dit programma a1, a2 en antwoord aan.

In ALGOL 60 zijn niet alle namen toegestaan, bv. a+b als naam of m:=n als naam is niet toegestaan. De wel toegestane namen heten "identifiers" en zijn gedefinieerd in 2.4. De formele definitie is te parafraseren als: een "identifier" is elke niet lege rij bestaande uit digits (in 2.2.1 gedefinieerd) en/of letters (in 2.1 gedefinieerd) mits met een letter beginnende. Alvorens op de formele definitie nader in te gaan beschouwen we het volgende, niet in het RR voorkomende, voorbeeld van een definitie:

<ab> ::= <uv> <wij> | <uv>; <wij> end .

Deze zegt dat de verzameling van alle eindige rijen basic symbols waarop "ab " van toepassing is, de vereniging is van twee verzamelingen nl.

(i) die waarvan de elementen in twee gedeelten, kop en staart, te splitsen zijn met "uv" is op de kop en "wij" is op de staart van toepassing

(ii) die waarvan de elementen in vier gedeelten te splitsen zijn, het 2^e en 4^e gedeelte zijn resp. de basic symbols ; en end en op het 1^e en 3^e gedeelte zijn "uv" resp. "wij" van toepassing.

De formele definitie voor "identifier" luidt (2.4.1):

<identifier> ::= <letter>|<identifier><letter>|<identifier><digit>

dus de verzameling van rijen basic symbols die de vereniging is van de verzameling letters en van de verzameling van die rijen basic symbols, waarvan het laatste basic symbol een letter of een digit is en de andere basic symbols op hun beurt weer een identifier vormen.

Om in te zien dat deze definitie correct is, keren we tot onze paraphrase in schrijftaal van de definitie terug. Klaarblijkelijk is elke letter een identifier, want een eindige niet lege rij letters en digits met een letter beginnend. Bestaat een identifier uit meer dan één basic symbol bv. ds70, dan is ds7 ook een identifier, want een eindige niet lege rij digits en letters met een letter beginnend en eveneens zijn ds en d dit.
In de formele definitie wordt deze gang van zaken nu in omgekeerde richting beschreven. Juist omdat als uitgeworpen anker van def. 2.4.1. de verzameling van alle letters (<letter>) als deelverzameling van de verzameling van alle identifiers gegeven is, levert de beschrijving <identifier><letter> op, dat, daar "d" een identifier is, want een letter, d gevolgd door de letter s, ds er ook één is; uit de beschrijving <identifier><digit> kun je dan afleiden dat ds7 een identifiers is -7 is een digit-en dus ook ds70. Omdat <identifier> zowel links als rechts van het ::= teken voorkomt noemen we deze regel "recursief".
U kunt nu 1.1 lezen.
Tenslotte wordt de aandacht erop gevestigd dat de keuze van identifiers voor een programma de werking van dat programma niet beïnvloedt in die zin dat het programma

begin integer som ; som := 1+2; print(som) end

tot dezelfde berekening, hetzelfde proces, binnen de rekenmachine aanleiding geeft als

begin integer product; product := 1+2; print(product)end

Vraag: gegeven de regels <ab> ::= (|[|<ab> |<ab><d> en <d> ::= <digit>.
Is [(((1(37(een "ab"?
En is (12345[een "ab"?

I.3 Functies van begin en end en het puntcommasymbool ; .

Het paar begin en end fungeert om de omvang van het programma, zijn begin en zijn einde, ondubbelzinnig vast te leggen.

Een ander gebruik daarvan, bv. als men in een ALGOL 60 programma het "compound statement"

begin a:=1; b:=2 end

tegenkomt, is al uit het begeleidend college bekend. Daarin fungeren begin en end uitsluitend als haakjespaar en is hun functie vergelijkbaar met het gebruik van haakjes in de expressie a * (b+c); a * b + c is een geheel andere expressie met i.h.a. een geheel andere waarde.
Het symbool ";" scheidt de zinnen van de taal ALGOL 60 van elkaar, zinnen die weer in declaraties en statements kunnen worden verdeeld.
De betekenis van een declaratie of een statement is een actie. Tenzij expliciet aangegeven door een goto statement (van de vorm goto <label>) worden deze acties in de gepresenteerde volgorde uitgevoerd. U kunt zich in dat geval de puntkomma als een "ga maar door" symbool opvatten.

I.4 Inleiding tot declaraties.

Laten we terugkeren tot het gegeven programma.
Na begin ontwaart U de "declaratie"

integer a1,a2, antwoord .

In het voorafgaande is het gebruik van het geheugen voor het onthouden van tussenresultaten vermeld, i.h.a. voor de executie van het programma benodigde gegevens. Een declaratie legt zowel de omvang van de ruimte hiervoor nodig als zijn onderverdeling in eenheden naar de ermee corresponderende identifiers vast.
Uit het basic symbol integer leidt de machine af dat dit ruimte moet zijn voor het opbergen van waarden van het type integer. Niet voor niets komt het woord integer in het basic symbol integer voor; waarden van het type integer representeren nl. een deelverzameling van de U welbekende integers. De verzameling van in de machine te representeren integers is eindig en van machine afhankelijk. Voor de EL X8 van het MC zijn dit alle integers tussen $-2^{26}+1$ en $2^{26}-1$. Uit de noodzaak van specificatie kan men afleiden dat er ook andere soorten waarden, "values" zijn.

Voorbeelden hiervan zijn waarden van het type real, zoals 3.14, en waarden van het type boolean, nl. true en false, d.w.z. waar en onwaar (net zoals 3+5 na evaluatie door de machine de waarde 8 oplevert, levert 3 < 5 na evaluatie door de machine de waarde true op).
De machine leidt uit

integer a1,a2, antwoord

af, dat (i) er 3x ruimte nodig is, nl. ruimte corresponderend met de identifier a1,
ruimte corresponderend met de identifier a2 en
ruimte corresponderend met de identifier antwoord,

(ii) de omvang van deze ruimte de omvang is die nodig is voor het opbergen van waarden van het type integer.

Het effect van deze declaratie is, dat de identifiers a1,a2 en antwoord gaan fungeren als variabelen. Er kunnen waarden aan worden toegekend d.m.v. assignment statements zoals geschetst in het begeleidend college. Dat een identifier niet automatisch een variabele is, kunt U zien aan het gebruik van een identifier als label in de gegeven ALGOL 60 versie van het algorithme van Euclides.

I.5 Inleiding tot de beschrijving van het assignment statement.
Op de declaratie van de variabelen volgen 8 statements.
Het eerste statement is "assignment statement"

a1 := read .

Bij uitvoering van elk assignement statement wordt aan de variabele links van het := teken de waarde van de expressie rechts van het := toegekend. Met het geheugen van de rekenautomaat voor ogen wil dit zeggen dat die uitgerekende waarde in gekodeerde vorm wordt opgeslagen in de geheugenruimte die aan de desbetreffende variabele is toegekend bij zijn declaratie (in dit geval dus executie van integer a1,a2, antwoord).

Hoe ziet een assignment statement er volgens uw tot nu toe opgedane kennis uit? Als een variabele, gevolgd door ":=", gevolgd door een expressie, die bij evaluatie een waarde oplevert. Identifiers die als variabelen fungeren behoren tot de verzameling van ALGOL 60 "variables", in 3.1.1. gedefinieerd. We kunnen onze huidige kennis over de vorm, "syntax" van het assignment statement dus vastleggen in de regel

<assignment statement> ::= <variabele> := <expressie>

die echter niet in deze vorm in het RR voorkomt daar het type van de waarden die de variabele mag bezitten, vastgelegd in de declaratie van deze variabele, in zekere mate met het type van de expressie moet overeenkomen (begin integer i; i := false end is geen correct ALGOL programma).
De betekenis, "semantiek", behorend bij het assignment statement is de "actie" die bij uitvoering ervan wordt ondernomen, d.w.z. de variabele uit het linkerlid wordt gedwongen naar de waarde van de expressie uit het rechterlid te verwijzen (die waarde wordt in het geheugen gezet op de plaats aan de variabele toegekend).
Vervolgens ontwaart U de aanroep van de procedure print, nl.

print(a1) , waarvan de functie, het afdrukken van zijn argument, in dit geval de waarde van a1, U reeds bekend is.

Dit is een voorbeeld van een "procedure statement".

I.6 De bibliotheek van een rekenmachine en de standaard functies.
In de tekst van het als voorbeeld gegeven programma ziet U dat de identifier read niet expliciet in een declaratie wordt ingevoerd.
Op de regel dat elke in een programma gebruikte identifier ook in dat programma gedeclareerd moet zijn, geldt de uitzondering dat dit niet behoeft voor identifiers van procedures uit de bibliotheek van de gebruikte machine. Die bibliotheek moet in ieder geval de standaard functies in 3.2.4 (ga ze na) opgesomd bevatten, doch ook alle voor invoer en uitvoer van gegevens benodigde procedures, zoals read en print.

Men kan dit als volgt visualiseren: aan de lijst van gedeklareerde identifiers wordt bij executie van een programma de standaard lijst van reeds binnen de machine bekende identifiers toegevoegd.
De getallen op de band met gegevens worden één voor één in het geheugen in volgorde van binnenkomst opgeslagen; een wijzer gaat naar het eerste getal verwijzen.
De taak van de procedure read is die getallen in diezelfde volgorde af te lezen. Bij elke aanroep van read wordt het in het geheugen aangewezen (door de wijzer) getal gelezen en wordt de wijzer gezet naar het volgende te lezen getal, indien aanwezig.
Daar read bij aanroep een waarde aflevert heet het een "functieprocedure".

I.7 Variabelen (3.1).
In de wiskunde wordt ook een ander gebruik van veranderlijken gemaakt dan wij tot nu in ALGOL 60 hebben gebruikt.
Een n-dimensionale ruimte bezit b.v. als elementen vectoren, n-tuples $(x_{1,..,}x_i, x_{i+1},...,x_n)$, een matrix kan symbolisch als (a_{ij}), i=1,...,n en j=1,...,m worden aangegeven. In ALGOL 60 noteert men dit als volgt: x[1],...,n[i], x[i+1],...,x[n] en a[i,j]. Merk op dat dit een lineaire notatie is. De ontwerpers van ALGOL 60 wilden geen betekenis toekennen aan het iets lager of iets hoger op de regel staan van basic symbols. Bedenk hierbij dat het programma op de ponsband symbool voor symbool de rekenmachine ingaat. Notaties als x_i, a_{ij}, $\frac{a}{b}$ en c^d zijn dus in ALGOL 60 uit den boze en in plaats daarvan worden de notaties x[i], a[i,j], a/=b, c↑d gebezigd.
De waarden die de indices i en j mogen aannemen, bij a[i,j] dus i=1,...,n en j=1,...,m moeten bij de declaratie van het array a gespecificeerd worden, evenals het type van de waarde door integer, real of boolean, dat aan de array elementen mag worden toegekend. Wanneer a[i,j] als variabele waarden van het type integer toegekend krijgt en i=1,...,10 en j=1,...,20 dan luidt de declaratie van a

integer array a[1:10, 1:20]

Hoe ziet zo'n ALGOL 60 geïndiceerde variabele, "subscripted variabele", er nu precies uit?. Dit is in 3.1, waarin het begrip "variabele" gedefinieerd is, vastgelegd.

Voorbeeld: is a3b17[i,j] een subscripted variabele?

De regel voor subscripted variabele is

<subscripted variabele> ::= <array identifier> [<subscript list>]

en dus, opdat a3b17[i,j] een subscripted variabele is, moet
a3b17 een array identifier en
i,j een subscript list zijn.

Uit de regel <array identifier> ::= <identifier> en
uit het feit dat a3b17 inderdaad een identifier is, volgt dan dat a3b17 een array identifier is.

Uit de regels

<subscript list> ::= <subscript expression> |
<subscript list> , <subscript expression>

en

<subscript expression> ::= <arithmetic expression>

volgt dat, opdat i,j een subscript list weze, i en j arithmetic expressions moeten zijn. Uit 3.3.1 volgt dat een arithmetic expression een variabele kan zijn.

De vraag wordt nu dus: zijn i en j variabelen?

Uit de regels <variabele> ::= <simple variabele>|<subscripted variabele>
<simple variabele> ::= <variabele identifier> (3.1)
<variabele identifier> ::= <identifier>
<identifier> ::= <letter>|<identifier><le ter>|<identifier>
<digit> (2.4.1)

in de regel voor letter (2.1)
volgt dan dat i en j variabelen, dus a3b17 een subscripted variable en ook variabele is.

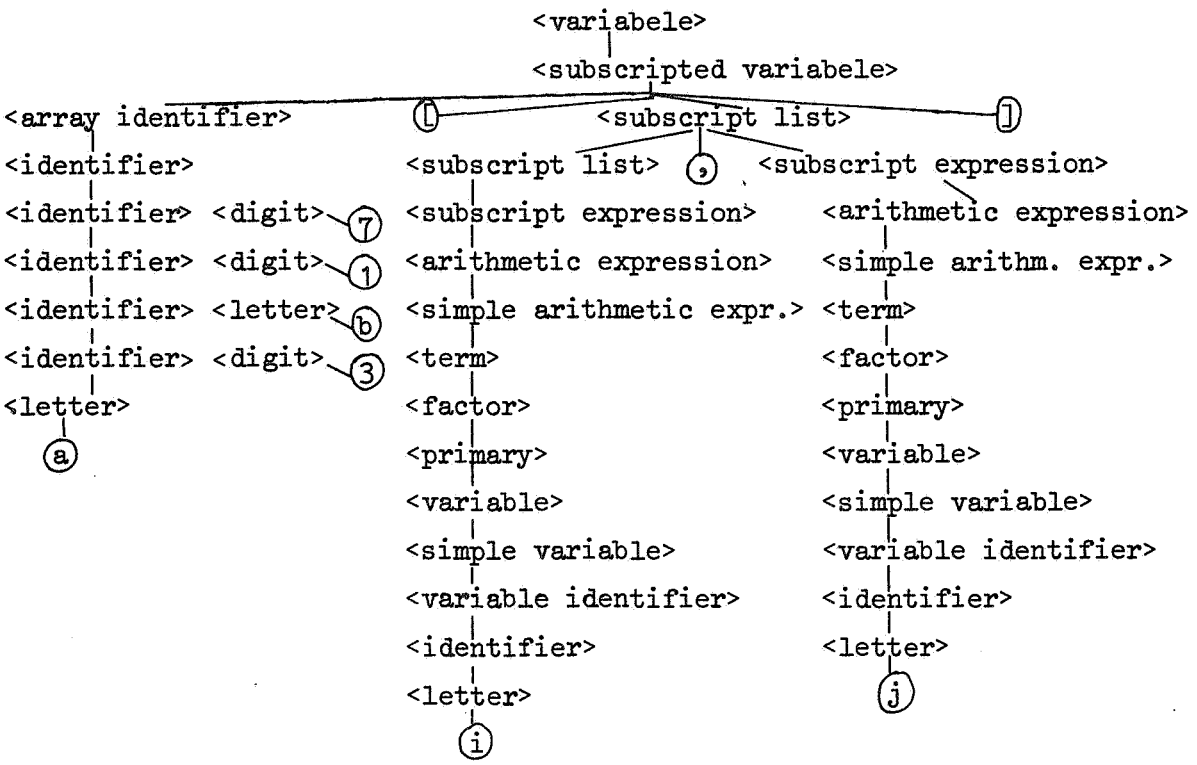
De uitdrukkingen rechts van het ::= teken en gescheiden door het | teken heten "directe producties" van de uitdrukking links van het ::= teken in de desbetreffende regel. Dus i is een directe productie van letter, identifier is een directe productie van zowel variabele identifier als array identifier.

Een directe productie a van b, een directe productie van een uitdrukking c, is een "productie" van c, d.w.z. er moeten dan regels

<c> ::= <b>

en <b> ::= <a> bestaan.

Dus i is een productie van identifier en identifier is een productie van simple variabele is. Een directe productie e van f, een productie van g, is een productie van g. Als we dus zeggen "i is een variabele" dan betekent dit "i is een productie van variabele" hetgeen streng te definiëren is. Hiermede is de aansluiting tussen de syntax-regels en RR en ons spraakgebruik op ondubbelzinnige wijze tot stand gebracht. Omdat i, a3b17, begin, end, + enz. niet verder te analyseren zijn m.b.v. de ALGOL 60 syntax-regels, heten ze "terminale producties". Nauwkeuriger: een "terminale productie" is een productie waarin geen enkel linkerlid van enige syntax-regel uit het RR voorkomt. Dus identifier is geen terminale productie van variabele, doch i wel. De ALGOL 60 syntax-regels heten ook wel productieregels. De afleiding dat a3b17[i,j] een (terminale) productie van variabele is, kan als volgt geschematiseerd worden in een "derivatieboom" voor deze afleiding; merk hierin op dat terminale basic symbols omcirkeld zijn:



De stukken derivatie boom

<arithmetic expression>
|
<simple arithmetic expression>
|
<term>
|
<factor>
|
<primary>
|
<variable>

worden in een volgende sectie besproken.

Vraag: bepaal of de volgende rijen basic symbols variables zijn en teken in het bevestigende geval hun derivatie bomen:

a(1,2) b[3c,5] cd[1,2] a[a[3]]

I.8 Inleiding tot procedures en function designators.

In het begeleidend college hebt u kennis gemaakt met procedure declaraties en procedure aanroepen. Dit geldt i.h.b. voor de procedures rest en ggd op blz.11 van de syllabus van Prof. van de Riet.

Vb.: In een wiskunde boek kun je het volgende aantreffen:

$f(x) = x^7+x^3+1$,

...

...

$x_0 = f(1)+f(3)$.

Wat wil die f(1) nu zeggen?

Indien de waarde van x_0 berekend moet worden, speelt zich bij de lezer, grotendeels onbewust, de volgende algorithme af:

(i) Zoek de definitie van f op.

(ii) Substitueer hierin 1 voor x, dus $f(1) = 1^7+1^3+1$.

(iii) Bereken de waarde van de na substitutie verkregen uitdrukking volgens "de geldende wiskundige conventies".

(iv) Vat f(1) als naam voor die in (iii) verkregen waarde, 3, op.

Wat heeft dit vb. nu met procedure declaraties en "aanroepen" van gedeclareerde procedures d.m.v. procedure statements te maken?
Al programmerende in ALGOL 60 rijst de behoefte aan "afkorting" van op diverse plaatsen van het programma voorkomende onderdelen. Deze onderdelen fungeren a.h.w. als standaardonderdelen van het desbetreffende programma. Opdat afkorting mogelijk worde, voorzien procedure declaraties deze standaardonderdelen van namen, in ALGOL-60 dus identifiers (en d.m.v. een parameterlijst van parameters, indien daar behoefte aan is). Door executie van een procedure declaratie wordt nu de naam van de procedure aan het desbetreffende standaardonderdeel, de "procedure body", verbonden.
Wordt nu zo'n procedure identifier, eventueel van parameterlijst voorzien, in een statement aangeroepen, dan wordt bij executie van deze aanroep het bovenstaande in 4 stappen gegeven algorithme door de rekenmachine uitgevoerd, met dien verstande dat de wiskundige conventies ondubbelzinnig moeten zijn vastgelegd om ze ondubbelzinnig te kunnen toepassen.

Vb.: Na executie van de declaraties van de procedures ggd en rest fungeren de identifiers ggd en rest als identifiers van hun respectievelijke "procedure bodies", d.w.z. hun standaardonderdelen.
Vindt executie van de aanroep ggd (1,5) plaats, dan

(i) zoekt de machine de "procedure body" behorend bij de naam ggd op,

(ii) substitueert hierin 1 voor m en 5 voor n,

(iii) wordt het hierna verkregen programmaonderdeel geëxecuteerd (wat weer executie van de aanroep rest (1,5) ten gevolge heeft),

(iv) wordt door het laatste assignmentstatement van de procedurebody

ggd:= n1

uit te voeren aan ggd (1,5) de waarde van n1, d.w.z. 1, een waarde van het type integer, toegekend; het type van de te assigneren waarde is in de declaratie vermeld door het basic symbol integer voorafgaande aan het basic symbol procedure waarmee de declaratie begint.

In dit voorbeeld werd tijdens executie van de procedure body een waarde aan zijn identifier toegekend door ggd:= n1. Is deze waardetoekenning het geval dan spreekt men i.h.a. van een "functieprocedure". Het type van de aan een functieprocedure toe te kennen waarde staat in zijn declaratie gespecificeerd door de basic symbols

integer, real of boolean aan procedure voorafgaande.

Een aanroep van een functieprocedure in een ALGOL 60 programma heet "function designator" : ggd(1,5) en rest(1,5) zijn dus function designators.

Bij de declaratie van een procedure kunnen parameters gespecificeerd worden in een tussen haakjes staande lijst van parameters genaamd "formal parameter part", die de "formal parameters" bevat.

Vb.: (m,n) treedt zowel bij de declaratie van ggd als van de rest als formal parameter part van ggd(m,n) en rest(m,n) op, met m en n als formal parameters.

Bij aanroep van de desbetreffende (functie)-procedure is voor elke formal parameter uit het formal parameter part een "actual parameter" in het "actual parameter part" van de aanroep (function designator) aanwezig.

Vb.: in de aanroep PRINT(ggd(p,q)) - zie blz. 11 van "Inleiding in de informatika" - is ggd(p,q) de actual parameter van het actual parameter part (ggd(p,q)) van het procedure statement PRINT(ggd(p,q)) en zijn op hun beurt p en q de actual parameters van het actual parameter part (p,q) van de function designator ggd(p,q).

In ALGOL 60 staan 2 parametermechanismen ter beschikking, waarvan wij er één besproken hebben, nl. het mechanisme waarbij iedere in een procedure body voorkomende formal parameter bij aanroep door de ermee corresponderende actual parameter vervangen wordt; dit substitutie proces heet "name replacement" en maakt deel uit van het parametermechanisme genaamd "call by name". Het alternatieve parametermechanisme komt later ter sprake.

Opgave: bestudeer sectie 3.2 van het Revised Report.

II. ALGOL 60 getallen, "numbers", en uitdrukkingen om hen te beschrijven.

II.1 Numbers.

De belangrijkste waarden in ALGOL 60 zijn de ALGOL 60 versie van getallen, de z.g. numbers, gedefinieerd in sectie 2.5.
De regels die de vorm, syntax, van een "number" vastleggen staan in 2.5.1 en kunnen als volgt geparafraseerd worden:
een ALGOL 60 getal, d.w.z. een number, wordt op de volgende manieren geschreven:

(i) op de gebruikelijke wijze en, indien dat nodig is, met een decimaalpunt i.p.v. een komma, d.w.z. als "integer", "decimal fraction" met ev. een teken, "decimal number" met ev. een teken,

bv. .117 -117,25 +0.25 .25

(ii) als integer macht van 10, d.m.v. het aparte, ondeelbare, basic symbol $_{10}$, d.w.z. als "exponent part", ev. met een teken,

bv. $_{10}5$ $-_{10}5$ $-_{10}-5$ $+_{10}+5$

(iii) als een vorm volgens (i) gevold door een vorm volgens (ii) zonder teken ervoor, d.w.z. als "unsigned number", "number",

bv. $-.83_{10}-2$.

De productieregels voor "integer", "decimal fraction", "exponent part", "unsigned number" vindt u in genoemde serie. "Integers" zijn van het type integer, alle andere numbers zijn van het type real.
Voorbeelden van numbers vindt u verder in 2.5.2.

Opgaven.

(i) Ga na dat de volgende rijen basic symbols geen number zijn:

17a $23.+2_{10}+3$ $12-_{10}2$ $-._{10}3$ $12._{10}-2$ 2.

(ii) Teken een derivatie boom voor $-.83_{10}-2$.

(iii) Zorg dat u 2.5 volledig begrijpt.

In ALGOL 60 staan 2 parametermechanismen ter beschikking, waarvan wij er één besproken hebben, nl. het mechanisme waarbij iedere in een procedure body voorkomende formal parameter bij aanroep door de ermee corresponderende actual parameter vervangen wordt; dit substitutie proces heet "name replacement" en maakt deel uit van het parametermechanisme genaamd "call by name". Het alternatieve parametermechanisme komt later ter sprake.

Opgave: bestudeer sectie 3.2 van het Revised Report.

II. ALGOL 60 getallen, "numbers", en uitdrukkingen om hen te beschrijven.

II.1 Numbers.

De belangrijkste waarden in ALGOL 60 zijn de ALGOL 60 versie van getallen, de z.g. numbers, gedefinieerd in sectie 2.5.
De regels die de vorm, syntax, van een "number" vastleggen staan in 2.5.1 en kunnen als volgt geparafraseerd worden:
een ALGOL 60 getal, d.w.z. een number, wordt op de volgende manieren geschreven:

(i) op de gebruikelijke wijze en, indien dat nodig is, met een decimaalpunt i.p.v. een komma, d.w.z. als "integer", "decimal fraction" met ev. een teken, "decimal number" met ev. een teken,

bv. .117 -117,25 +0.25 .25

(ii) als integer macht van 10, d.m.v. het aparte, ondeelbare, basic symbol $_{10}$, d.w.z. als "exponent part", ev. met een teken,

bv. $_{10}5$ $-_{10}5$ $-_{10}-5$ $+_{10}+5$

(iii) als een vorm volgens (i) gevold door een vorm volgens (ii) zonder teken ervoor, d.w.z. als "unsigned number", "number",

bv. $-.83_{10}-2.$

De productieregels voor "integer", "decimal fraction", "exponent part", "unsigned number" vindt u in genoemde serie. "Integers" zijn van het type integer, alle andere numbers zijn van het type real.
Voorbeelden van numbers vindt u verder in 2.5.2.

Opgaven.

(i) Ga na dat de volgende rijen basic symbols geen number zijn:

17a $23.+2_{10}+3$ $12-_{10}2$ $-._{10}3$ $12._{10}-2$ 2.

(ii) Teken een derivatie boom voor $-.83_{10}-2.$

(iii) Zorg dat u 2.5 volledig begrijpt.

II.2 Arithmetische expressies.

In de wiskunde komen algebraïsche uitdrukkingen als

$$(*) \qquad -\frac{a}{b^c} + d \times e$$

voor, i.h.a. uitdrukkingen met getallen als waarde.

In ALGOL 60 wordt het equivalent van zulke algebraïsche expressies door het begrip "simple arithmetic expression" gedekt. Een "simple arithmetic expression" is een regel om een ALGOL 60 getal, "number", te berekenen.

Aangezien in ALGOL 60 het lineairiteits principe geldt - a_i wordt als a[i] en x^y wordt als x↑y genoteerd - wordt (*) in ALGOL 60 de simple arithmetic expression

(**) -a/b↑c+d*e.

Zowel in (*) als in (**) hebben bepaalde operaties voorrang, "precedentie", op andere operaties: in (*) gaan vermenigvuldigingen en delingen aan optelling vooraf.

Deze precedentie, voorrang, van operaties is in ALGOL 60 aangehouden. Hierdoor is bijvoorbeeld de waarde van de simple arithmetic expression a+b*c niet identiek aan de waarde van de simple arithmetic expression (a+b)*c.

ALGOL 60 kent de volgende operaties tussen twee numbers, die hieronder in volgorde van precedentie, van boven naar onder in afnemende mate, staan opgesomd:

(i) machtsverheffing ↑,
(ii) vermenigvuldiging *, deling /, integer deling ÷,
(iii) optelling +, aftrekking -.

Indien in een simple arithmetic expression verschillende operatoren van dezelfde precedentie naast elkaar voorkomen, bijv. in a*b/c, worden de bijbehorende operaties van links naar rechts door de expressie werkende, uitgevoerd: a*b/c specificeert dus hetzelfde number als (a*b)/c.

Een simple arithmetic expression is een regel om een number te berekenen; in bepaalde gevallen is deze number een integer.

De operator ÷ nu is alleen gedefinieerd als beide operanden van het type integer zijn en geeft een integer waarde af verkregen door eerst m.b.v. / te delen en vervolgens naar nul toe tot een integer af te ronden.

In sommige opzichten is ALGOL 60 rijker dan wiskunde taal: ook

(***) if 1>-5 then S+3*Q/A else 2*S+3*q

is een in ALGOL 60 toegelaten regel om een number te berekenen.

Dit, (***), is een vb. van een "arithmetic expression"; dit begrip dekt de meest algemene categorie regels in ALGOL 60 om numbers te berekenen en wordt gedefinieerd door

<arithmetic expression>::= <simple arithmetic expression>/
<if clause><simple arithmetic expression> else
<arithmetic expression>.

Van de "if clause", zie IV, vertel ik u voorlopig alleen dat

if q>-5 then

er een vb. van is.

Alvorens tot definitie van simple arithmetic expression (sae) over te gaan, definieer ik eerst de syntactische eenheden die als operanden voor de eventueel in sae voorkomende operatoren kunnen optreden, de zg. "primaries" door

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

Met primaries als bouwstenen is de in 3.3.1 geg. definitie van "term" te parafraseren door:

(i) een primary is een term,

(ii) een uitdrukking opgebouwd uit de operatoren op twee argumenten ↑, *, /, ÷ met primaries als operanden is een term.

Met termen als bouwstenen is nu een sae te definiëren als:

(i) een term,

(ii) een term met een + of - ervoor (die hier dus als operaties op één

argument worden opgevat),

(iii) een sae gevolgd door een + of - gevolgd door een term.

Vraag: Waarom is de hierondervolgende omschrijving van sae onjuist?

Met primaries als bouwstenen is de in 3.3.1 gegeven definitie van sae te omschrijven als:

(i) een primary is een sae,

(ii) een uitdrukking opgebouwd uit de operatoren ↑, *, /, ÷, +, - met primaries als operanden is een sae.

De waarde van een sae wordt geheel vastgelegd door:

(i) eerst de waarden van zijn primaries te berekenen

(ii) dan pas de eventueel erin gespecificeerde operaties uit te voeren, in volgorde van precedentie, op de berekende "number" waarden van de (primary)-operanden.

Het type van de waarde van een sae is real indien er

(i) tenminste één waarde van het type real in voorkomt,

(ii) indien de sae de operator / bevat,

(iii) indien de sae in de in 3.3.4.3 gespecificeerde gevallen de operator ↑ bevat.

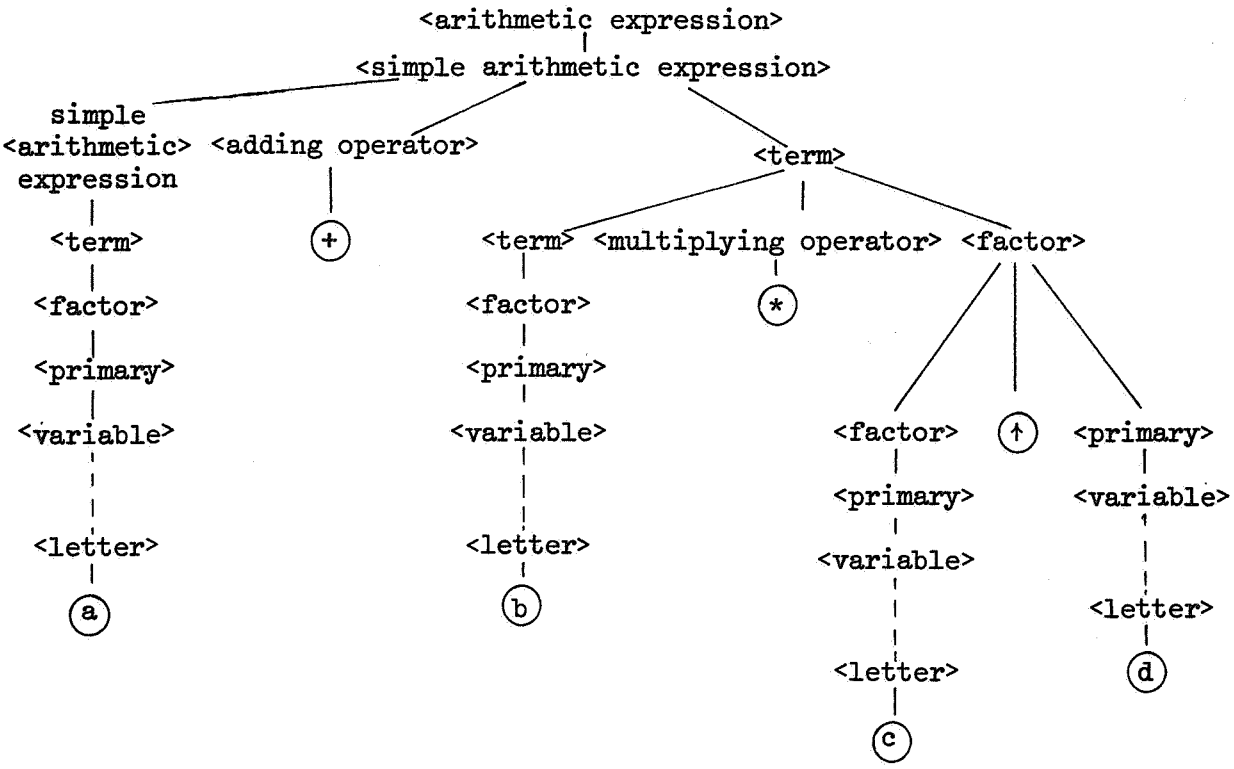
In bepaalde gevallen van machtsverheffing is de waarde van een sae ongedefinieerd; deze zijn in 3.3.4.3 gespecificeerd.

In II.1 heb ik numbers van 2 type, real en integer ingedeeld.
Als in een sae met waarden van het type real wordt geopereerd, worden alle waarden en operaties met een eindige nauwkeurigheid berekend en zullen i.h.a. afrondingsfouten optreden. Dit is onderwerp van studie voor de

"numerieke wiskunde".

Voorbeeld

Schets van derivatie boom voor a+b*c↑d.



Opgaven

1. Ga (m.b.v. 3.3.1) na dat uit dit vb. blijkt dat de opbouw van een arithmethische expressie in dezelfde volgorde plaats vindt als de waardeberekening volgens de precedentie regels.

2. Als a[1,1] = 2, a[1,2] = 1, a[2,1] = 2 en a[2,2] = 2, bereken dan
   a[a[1,2],a[2,1]] * a[a[1,1]↑a[2,2]*a[2,1]÷a[2,2]*a[1,2]÷a[2,1]÷a[1,1],
   if a[2,1]≠2 then 367 else if a[2,2]≤1 then
   a[2,2]↑a[1,2]-7 else 2].

3. Zie 3.2.4 en 3.2.5. F is gedefinieerd door
   2*(entier(z)-sign(z)*entier(abs(z))) - sign(2*sign(z)+1) + 1.
   Wat is de waarde van F voor verschillende waarde van z?

4. Schrijf in ALGOL 60:
   $|x+y^c-3|.e^{\arctan(2/(x^2+y^2))}$, $\sqrt{a^2+b^2+2ab\cos C}$ en $a_{i_1}{}^2{}_1+a_{i_2}{}^2{}_2+3^{-x^2}\sin(\sqrt{x})$.

5. Bestudeer sectie 3.3 van het ALGOL 60 rapport aandachtig en stel er vragen over.

III. Het assignmentstatement.

Voor de inleiding tot het assignmentstatement verwijs ik naar I.5.
Alvorens er op door te gaan, het volgende voorbeeld:

de statements i:= 1; j:= 1; A[k]:= 1

kunnen in ALGOL 60 worden afgekort tot het ene assignment statement

i:= j:= A[k]:= 1.

Met 4.2.1. voor ogen, geldt:

1 i:= en j:= en A[k]:= zijn "left parts" die in i:= j:= A[k]:= tot een "left part list" zijn samengebundeld

2 1 is een "arithmetische expressie"

3 hierdoor voldoet i:= j:= A[k]:= 1 aan de vorm van het assignment statement vastgelegd door

<assignment statement>::= <left part list><boolean expression>/
<left part list><arithmetic expression>

NB: Deze samenbundeling van left part tot een left part list is alleen toegestaan indien de typen van de variabelen in de left parts gelijk zijn.

Opgave
Construeer een derivatie boom van dit voorbeeld uit <assignment statement>.

Wat gebeurt er bij executie van

i:= A[i]:= i+1

aangenomen dat i vóór deze executie de waarde 5 bezit?

De uitvoering van een assignment is een proces dat in 3 stappen verloopt; deze zijn beschreven in de series 4.2.3.1, 4.2.3.2 en 4.2.3.3.
Toepassing van 4.2.3.1 levert op dat i in A[i] de waarde 5 bezit, toepassing van 4.2.3.2 dat i+1 bij evaluatie de waarde 6 oplevert en toepassing van 4.2.3.3 dat zowel aan i als aan A[i] de waarde 6 wordt toegekend.
Dit is dus geen afkorting van i:= i+1; A[i]:= i+1.
Zelfs i:= j:= a+b is geen afkorting van i:= a+b; j:= a+b daar a hierin als functiedesignator kan optreden, die afhangt van de waarde van i.

Vergelijk hiervoor de output van

```
begin real procedure a; a:= b+i; integer i,b,j;
      b:= 2; i:= 4; i:= j:= a+b; print(i);print(j)
end
```

maar met de output van het programma verkregen door hierin

i:= j:= a+b door i:= a+b; j:= a+b

te vervangen.

Opgave

Ook A[a]:= A[a]:= a+b is geen afkorting van A[a]:= a+b; A[a]:= a+b. Ga dit n.a.v. bovenstaand voorbeeld, waarin a als function designator optreedt, na.

In 4.2.4 staat vermeld dat aan een boolean variable of boolean procedure identifier alleen boolean values kunnen worden toegekend en dat aan integer of real variables of aan integer of real procedure identifiers zowel integer als real values kunnen worden toegekend met dien verstande dat afronding kan plaatsvinden bij toekenning van een real value aan een integer variable of procedure identifier.

Opgave

Welk resultaat wordt bereikt door executie van de statements

```
i:= 12; k:= 5; A[i÷k,i-2*k-1]:= A[k-4,i-2*k]:= i:= k+1;
A[i÷k,i-5]:= A[k-i÷2,k-i÷3-1]:= k:= i+1;
```

?

IV. Wanneer is een rij basic symbols een ALGOL 60 programma?

In de "Inleiding tot de informatika" en het voorafgaande zijn reeds meerdere malen voorbeelden van ALGOL 60 programma's gegeven.
Dat dit ook inderdaad ALGOL 60 programma's waren is alleen beweerd; dit is tot dusverre niet aangetoond.
In het Revised Report nu is de vorm van een ALGOL 60 programma zodanig vastgelegd dat aan te tonen valt of een rij basic symbols zo'n programma is.
Dit aantonen valt in twee onderdelen uit een:

(i) De desbetreffende rij moet de vorm van een programma bezitten: een syntactische voorwaarde, die expliciet luidt:
de desbetreffende rij basic symbols moet een terminale produktie van de in 4.1.1. gedefinieerde syntactische categorie "program" zijn.

(ii) De eenmaal als terminale produktie van "program" herkende rij moet voldoen aan die voorwaarden uit het Revised Report, die niet in de erin toegepaste vorm van syntactische regels geperst konden worden en daarom in het Engels moesten worden gegeven; hieronder vallen i.h.b. de semantische voorwaarden.

IV.1. Voorbeeld van een programma.

In deze sectie wordt aangetoond dat de in I.1 met de naam programma aangeduide rij basic symbols een ALGOL 60 programma is, d.w.z. zowel een terminale produktie van program is als aan de op déze specifieke terminale produktie van "program" van toepassing zijnde andere voorwaarden voldoet.

Sla 4.1.1. op; de laatste hierin opgesomde syntactische regel luidt:

<program>::= <block>|<compound tail>

Syntactisch komt het verschil tussen een block en een compound tail hier op neer dat:

(i) in een compound tail tussen het eerste begin en laatste end symbol één of meer door puntkomma's gescheiden statements voorkomen,

(ii) terwijl in een block tussen het eerste begin en laatste end symbol eerst één of meer door puntkomma's gescheiden declaraties en pas daarna de statements voorkomen, die van de laatste declaratie en van elkaar weer door puntkomma's gescheiden zijn.

Lees de Engelse tekst van serie 4.1.1.
De daarin door

L:L:...L:begin D;D;...;D;S;...;S end

beschreven algemene vorm van een block en de door

L:L:...L:begin S;...;S end

beschreven algemene vorm van een compound statement worden gerechtvaardigd door de volgende, in 4.1.1, vermelde regels:

<block>::= <unlabelled block>|<label>:<block>
(De algemene vorm van een block, d.w.z., van een produktie van block is dus: een (eventueel lege) rij elementen van de vorm
<label>:
gevolgd door een unlabelled block.)
<unlabelled block>::= <block head>;<compound tail>
<block head>::= begin <declaration>|<block head>;<declaration>
(De algemene vorm van een block head is dus: een begin symbol gevolgd door één declaratie òf door meer declaraties, die door puntkomma's van elkaar gescheiden zijn.)
<compound tail>::= <statement>end|<statement>;<compound tail>
<compound statement>::= <unlabelled compound>|<label>:
<compound statement>
<unlabelled compound>::= begin <compound tail>

Opgave: ga na dat de boven beschreven algemene vorm ook inderdaad door deze regels gedekt wordt. Ten einde aan te tonen dat de I.1. gegeven rij basic symbols een terminale productie van program is, rest dus, bijvoorbeeld d.m.v. een derivatie boom, aan te tonen dat

(i) integer a1, a2, antwoord
een terminale produktie van declaration, d.w.z. een declaratie, is

en

(ii) a1:= read
print(a1)
a2:= read
print(a2)
antwoord:= a1 + a2
print(antwoord)
antwoord:= a1 - a2
print(antwoord)
elk terminale produkties van statement, d.w.z. statements, zijn.

Wat (ii) betreft:
Van 4 van de 8, naar bewering, statements kunt U m.b.v. de in III behandelde stof aantonen dat het assignment statements zijn.
De andere 4, naar bewering, statements zijn, naar bewering, procedure statements, d.w.z. elk een terminale produktie van procedure statement.
Dit volgt uit 4.7.1. dmv. de regels:

<procedure statement>::= <procedure identifier><actual parameter part>
de in 3.2.1. gegeven regel voor procedure identifier,
<actual parameter part>::= <empty>|(<actual parameter list>)
<actual parameter list>::= <actual parameter>|<actual parameter list>
<parameter delimeter><actual parameter>
<actual parameter>::= <string>|<expression>|<array identifier>
<switch identifier>|<procedure identifier>
de in 3 geg. regel voor expression:
<expression>::= <arithmetic expression>|<Boolean expression>|
<designational expression>
<parameter delimeter>::= ,|)<letter string>:(
en de reeds in II behandelde regels voor arithmetic expression.

Uit de eerste vier in 4.1.1. gegeven regels volgt dan nog dat procedure statement en assignment statement elk een terminale produktie van statements zijn.

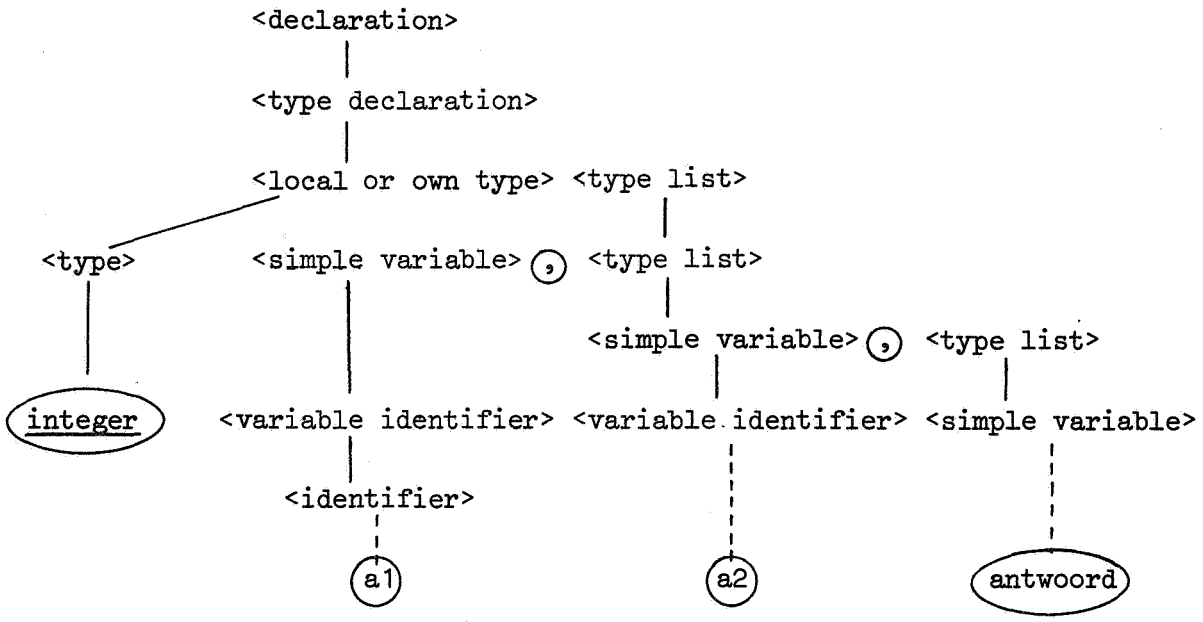
Opgave: ga dit na.

Sla, om (i) aan te tonen, de secties 5 en 5.1.1. op.

Uit de onderstaande derivatieboom, opgebouwd met behulp van regels in deze secties vermeld, volgt dat

integer a1, a2, antwoord

een declaratie is. Ga dit na.



Opgave: teken, op een groot vel papier, een derivatieboom van het besproken programma.

Zoals reeds vermeld zijn bepaalde in het Revised Report neergelegde eisen waaraan een ALGOL 60 programma behoort te voldoen in de Engelse taal en niet in syntactische regels vastgelegd.

Op dit programma zijn van toepassing, behalve de reeds gegeven semantiek van het assignment statement en de reeds ingeleide semantiek van het procedure statement en block:

(Uit sectie 5)

"Apart from labels and formal parameters of procedure declarations and with the possible exception of those for standard functions all identifiers of a program must be declared. No identifier may be declared more than once in any once block head."

"Declarations serve to define certain properties of the quantities used in the program and to associate then with identifiers. A declaration of an identifier is valid for one block. ...
Dynamically this implies the following: at the time of an entry into a block all identifiers declared for the block assume the significance implied by the nature of the declarations given."

(Uit sectie 5.1.3) "Integer declared variables may only assume positive and negative integral values including zero."

Die secties uit MR 81 waarin read en print worden toegelicht, waardoor vastgesteld kan worden of

```
read
print(a1)
print(a2)
print(antwoord)
```

"juiste" aanroepen van procedures zijn, d.w.z. het aantal en type van de parameter kloppen en de procedure identifiers zijn correct.

Opmerking: indien de 8 statements uit het besproken programma willekeurig gepermuteerd worden, verkrijgt U eveneens een correct ALGOL 60 programma (pas de gegeven condities voor "correct programma", die nu gegeven zijn, er maar op toe!). De output ervan kunt U echter i.h.a. niet meer voorspellen, daar

"at the time of entry into a block all identifiers declared for the block assume the significance implied by the nature of the declaration given",

dat wil in dit geval zeggen dat a1, a2, antwoord elk ruimte voor het opbergen van integer toegewezen krijgen. Indien de assignaties aan a1, a2, antwoord niet voorafgaan aan "berekening" van a1, a2, a1+a2, a1-a2 dan zullen de desbetreffende identifiers de reeds op de toegewezen plek aanwezige integer, overgebleven van een vorig programma, als waarde bezitten en die is i.h.a. onbekend.

Het probleem of een programma ook inderdaad datgene uitvoert, wat de schrijver ervan verwacht, kan bij de huidige stand der wetenschap slechts door "trail and error" dmv. executie op een rekenmachine vastgesteld en i.h.a. niet bewezen worden. Deze problematiek staat centraal in de "semantiek van programmeertalen".

IV.2. Semantiek van declaraties en blocks.

Voorbeeld:

```
block 1 | begin     real A, y;
        |           A:= 3.141592654;
        | block 2   | begin real x;
        |           |       x:= 2 ;
        |           |       x:= (x+A/x)/2; print(x);
        |           |       y:= (x+A/x)/2
        |           | end;
        |           print (y)
        | end
```

Dit ALGOL 60 programma heeft de vorm van een block, dat op zijn beurt uit een type declaration en drie statements bestaat, het tweede waarvan weer een block is.
Concentreer uw aandacht nu op de type declaration real x in block 2 en ga over tot het lezen van 5.

"A declaration of an identifier is valid for one block" (i.h.a. onjuist, zoals zal blijken).

"Dynamically this implies the following ...

... assume the significance implied by the nature of the declarations given".

Dus, in dit geval, "implied by the nature of the type declaration real x", dwz. de identifier real x moet waarden van het type real kunnen bezitten (hier slaat 5.1.3 op).

"Identifiers which are not declared for the block, on the other hand, retain their old meaning".

Dus A en y, gedeclareerd in block 1, behouden in block 2 hun oude betekenis.

"At the time of an exit from a block all identifiers which are declared for the block lose their local significance".

Dus, na executie van block 2 verliest x zijn betekenis, hetgeen zoveel zeggen wil als

a) er kan geen assignatie meer aan x plaats vinden.

b) x bezit niet meer de laatste waarde in block 2 aan x geassigneerd.

(x bezit natuurlijk ook voor executie van block 2 geen betekenis, daar die pas "at the time of entry into the block" eraan gegeven wordt).

Terminologie: x is t.o.v. block 2 een lokale variabele;

Variabelen, die in een binnenblock voorkomen met de betekenis eraan gegeven in een buitenblock, het binnenblock omvattend, heten t.o.v. dit binnenblock globale variabelen. Dus: A en y zijn globale variabelen t.o.v. block 2, doch lokale variabelen t.o.v. block 1.

Voorbeeld:

```
                ┌ begin     real A; integer x;
                │           A:= 3.14          ;
                │          ┌begin real x;
block 1 ┌ block 2 ┌       │      x:= 2 ;
        │       │  │       │      x:=(x+A/x)/2; print(x)
        │       │  │       │ end;
        │       │  │        x:= A; print(x)
        │       │  │
        │       │  └ end
        │       │
        │       └
        │
        └
```

In dit ALGOL 60 programma worden geen van de voorwaarden, in 5 genoemd, geschonden. De vraag rijst nu: aan welke x wordt door executie van x:= A geassigneerd, dwz. identificeert de x uit x:= A nu de real variable x uit block 2 of de integer variable x uit block 1?
Hierop slaan de volgende zinnen uit 5:

> ... at the time of entry into a block all identifiers declared for the block assume the significance implied by the nature of the declarations given.
> If these identifiers had already been defined by other declarations outside they are for the time being given a new significance.

Dus: bij binnenkomst in block 2, t.g.v. zijn executie geldt: x "assumes the new significance given by the declaration real x."
"At the time of an exit from a block all identifiers which are declared for the block lose their local significance."

Dus: bij het verlaten van block 2 verliest x de betekenis eraan gegeven door executie van de t.o.v. block 2 locale declaratie real x.
"Outside this block - e.g. block 2 - the particular identifier - x - may be used for other purposes."

Dwz. na het verlaten van block 2, bezit x weer de betekenis eraan gegeven door executie van de t.o.v. block 1 locale declaratie integer x

Tot slot wijs ik er op dat het own begrip, in 5 en 5.1 voorkomende, niet behandeld zal worden, daar het in de praktijk weinig zin bleek te hebben.

Terminologie: dat gedeelte van een programma, waarin een bepaalde declaratie "van kracht is" en dus de corresponderende identifier de in deze declaratie gespecificeerde betekenis bezit, heet de scope van die identifier.

Het scope-probleem voor identifiers staat behalve in 5 ook in 4.1.3 beschreven en wordt in 4.13 ook tot labels uitgebreid:

"A label separated by a colon - dwz. dubbele punt - from a statement, i.e. labelling their statement, behaves as though declared in the head of the smallest embracing block, i.e. the smallest block whose brackets begin and end

Dus: de rij basic symbols

begin goto L; begin integer i; L: PRINTTEXT(‡ onzin ‡) end end

bezit geen betekenis, ofschoon het wel een terminale productie van <program> is.

"In this context a procedure body must be considered as if it were enclosed by begin and end and treated as a block."

Dus: de rij basic symbols

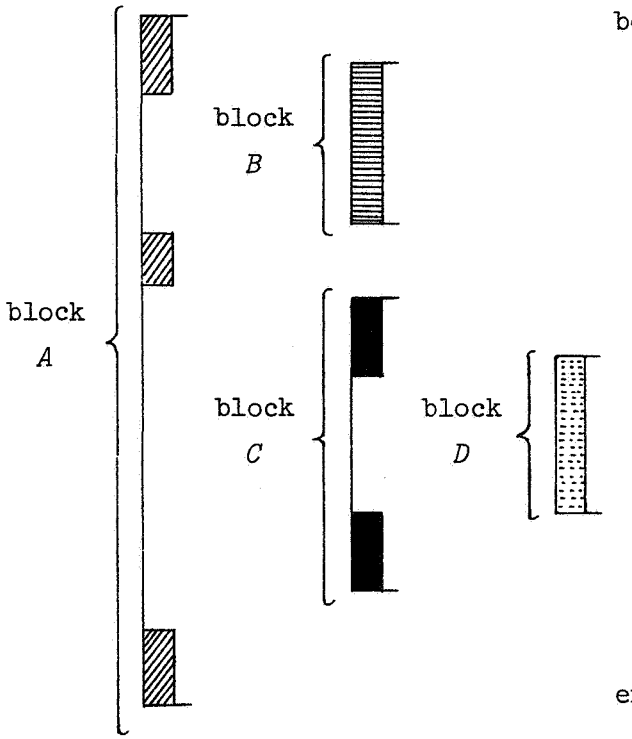
```
begin procedure P;
      L: begin M: PRINTTEXT (‡weer onzin‡) end;
      goto if 3.1768935569 = 3.1768935568 then L else M
end
```

bezit geen betekenis, ofschoon het wel een terminale productie van <program> is, omdat er naar L of naar M kan worden gesprongen en beide labels lokaal zijn t.o.v. de procedure body van P, dwz., derhalve geen betekenis bezitten buiten deze procedure body, waar het goto statement staat.

Merk als detail op dat, vanwege de eindige met elke rekenmachine wisselende nauwkeurigheid waarmee number gerepresenteerd worden, de lezer de waarheidswaarde van de expressie 3.1768935569 = 3.1768935568, ver-

kregen door executie van deze (boolean) expressie op een willekeurige rekenmachine, niet kan vaststellen zonder zich op de hoogte te stellen van de getalrepresentatie van de desbetreffende rekenmachine.

Opgave: Bepaal de scopes van de 18 verschillende variabelen, in de onderstaande block-schets gedeclareerd.




```
begin real a, b, c, d, e, f
  ...
  begin real d, e, f, g
    ...
  end;
  ...
  begin real c, d, h, j
    ...
    begin real b, f, h, k
      ...
    end;
    ...
  end;
  ...
end
```

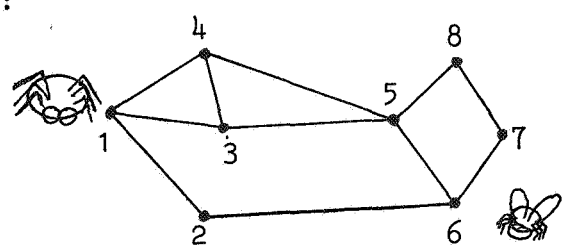
V. De spin en de vlieg.

V.1. Het spel en zijn beschrijving in ALGOL 60.

Het spelletje "de spin en de vlieg" is bij uitstek een voorbeeld van een algorithmisch proces. Bij bespreking ervan worden enkele facetten belicht van o.a. het for- en het conditional-statement. Het spel wordt tussen twee spelers, "spin" en "vlieg" genaamd gespeeld, die zich elk steeds "op een knoop bevinden" en afzonderlijk "een zet doen". Het spelelement bestaat daaruit dat enerzijds de spin de vlieg probeert te "vangen", anderzijds de vlieg aan de spin probeert te ontsnappen.

Definities: een web is een eindige (niet lege) verzameling elementen, genaamd knopen, die paarsgewijs al dan niet met elkaar verbonden zijn; geïsoleerde knopen zijn niet toegelaten,
een zet doen is zich van de ene naar de andere van een paar verbonden knopen verplaatsen,
de spin heeft de vlieg gevangen als zij zich op dezelfde knoop bevinden.

Voorbeeld:



De knopen zijn genummerd van 1 t/m 8.
De spin bevindt zich op knoop 1 en de vlieg op knoop 6.
De spin aan zet op s, 1, kan zich door een zet te doen naar 2, 3 of 4 verplaatsen.
De vlieg aan zet op v, 6, kan zich door een zet te doen naar 2, 5 of 7 verplaatsen.

Probleem: Bepaal de knopenparen (s,v) waarvoor geldt dat er met de spin op s aan zet een strategie voor de spin bestaat, die, welke tegenstrategie de vlieg, in aanvang op v, ook volgt om aan de spin te ontsnappen, de spin de vlieg doet vangen; geef deze strategie.

Dus: de spin aan zet moet een dergelijke zet doen, dat deze belandt op een knoop waarvoor òf geldt de vlieg is nu gevangen òf geldt hoe de vlieg zich ook wendt of keert, vangen zal de spin hem altijd;

de vlieg aan zet moet een dergelijke zet doen, dat deze belandt op een knoop waarvan bekend is dat deze nog mogelijkheid tot ontsnapping aan vangst biedt.

Commentaar: Zij n het aantal knopen van het web.

Van te voren beschouwen wij integer array zet[1:n,1:n] en boolean array verlies[1:n,1:n] als gedeclareerd.

Laat de spin zich op de knoop genummerd door s, de vlieg op de knoop genummerd door v, bevinden. Na uitvoering van het algorithme moet de waarde van zet[s,v] aangeven, indien deze van nul verschilt, dat, met de spin aan zet, de spin de vlieg kan vangen en welke zet de spin moet doen om dit (uiteindelijk) te bereiken.

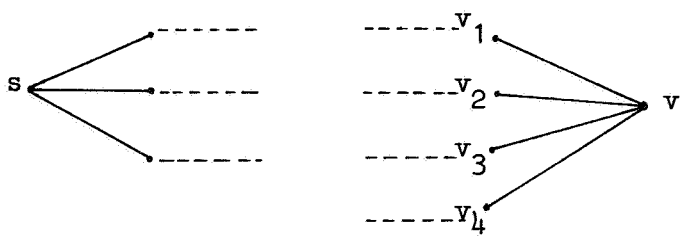
Analoog geldt voor de waarde van verlies[s,v] dat, met de vlieg aan zet, indien deze true is, de vlieg mat staat; anders bestaat er een uit het array zet af te leiden strategie voor de vlieg om te ontsnappen.

Voor alle paren (s,v) worden geëxecuteerd zet[s,v]:= 0 en verlies[s,v]:= false.

Anker: Je weet dat, indien de spin en de vlieg zich op twee met elkaar verbonden knopen bevinden, de spin de vlieg in één zet kan vangen, indien de spin aan zet is.

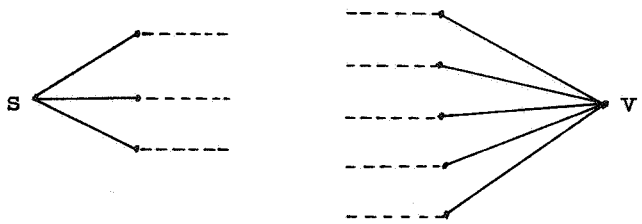
Voor alle met elkaar verbonden knopen geldt dus dat zet[s,v]:= v moet plaatsvinden; goto vlieg en merk op dat de vlieg nu aan zet is.

Vlieg:



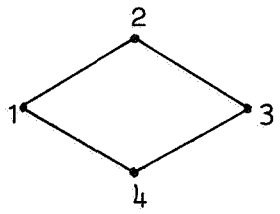
Voor alle paren (s,v) met s ≠ v beschouwen we alle mogelijke zetten voor de vlieg op grond van de reeds opgedane, in array zet vastgelegde, informatie.
De vlieg is aan zet; er wordt vastgesteld of de vlieg nog een zet naar een knoop v' kan doen met zet[s,v'] = 0, dwz., de vlieg kan voorlopig nog ontsnappen.
Is dit niet het geval, dwz., voor alle zetten v' van de vlieg geldt zet[s,v'] ≠ 0, dan moet verlies s,v := true worden geëxecuteerd, daar de vlieg mat staat.
Is er enige subscripted variable verlies[s,v] in deze stap van waarde veranderd, dan goto spin en probeer daarmee de strategie voor de spin te verbeteren; beëindig anders het algorithme.

Spin:



Voor alle paren (s,v), s ≠ v, wordt, op grond van de in boolean array verlies vastgelegde verliesposities van de vlieg vastgesteld of de spin in één zet "een verliespositie van de vlieg bereiken kan", dwz., van s naar s' kan gaan met verlies[s',v] = true. Is dit het geval dan moet kennelijk zet[s,v]:= s' geëxecuteerd worden.
Is enige variabele zet[s,v] van waarde veranderd, dan goto vlieg en probeer daarmee meer verliesposities voor de vlieg te vinden; beëindig anders het algorithme.

Voorbeeld: Beschouw het volgende web:



De uitvoering van het zojuist gegeven algorithme verloopt als volgt:

in anker wordt geëxecuteerd

```
zet[1,2]:= 2  ;  zet[2,1]:= 1  ;
zet[2,3]:= 3  ;  zet[3,2]:= 2  ;
zet[3,4]:= 4  ;  zet[4,3]:= 3  ;
zet[1,4]:= 4  ;  zet[4,1]:= 1  ;
```

in vlieg wordt vervolgens geëxecuteerd

verlies[1,3]:= verlies[3,1]:= verlies[1,4]:= verlies[4,1]:= true

en in spin wordt het algorithme beëindigd.

Na analyse blijkt dat de gedeelten gelabelled anker en spin gecombineerd kunnen worden door ook paren (s,v) met s = v te beschouwen, zoals in de nu volgende ALGOL 60 versie. Het boolean array pad[1:n,1:n] is gedeclareerd om voor alle indices (s,v) aan te kunnen geven of (s,v) al (pad[s,v]:= true) dan niet (pad[s,v]:= false) verbonden zijn.

```
begin integer n; n:= read; comment de waarde van n geeft aantal knopen van
                                    web aan;
      begin integer array zet[1:n,1:n]; boolean array pad, verlies[1:n,1:n];
                                        integer s,v,t; boolean klaar;
web inlezen   : for s:= 1 step 1 until n do for v:= 1 step 1 until n do
                pad[s,v]:= read = 1;
                comment lag op de getallenband een 0 voor dan waren s en v
                        niet verbonden, lag er een 1 voor, dan waren ze dit
                        wel: de waarde van 0=1 is false en de waarde van 1=1
                        is true;
initialisatie: for s:= 1 step 1 until n do for v:= 1 step 1 until n do
                          begin zet[s,v]:= if s≠v then 0 else 1;
                                verlies[s,v]:= s = v end;
spin:           klaar:= true;
                for s:= 1 step 1 until n do
                    for v:= 1 step 1 until n do
                    if zet[s,v]=0 then begin for t:=1 step 1 until n do
                                             if pad[s,t]∧verlies[t,v] then
                                                begin zet[s,v]:= t;
                                                      klaar:= false;
                                                      goto vangst end;
                                       vangst:
                                       end;
                if klaar then goto eind;
vlieg:          klaar:= true;
                for s:= 1 step 1 until n do
                    for v:= 1 step 1 until n do
                    if ¬ verlies[s,v] then begin for t:=1 step 1 until n do
                                                 if pad[t,v]∧zet[s,t]=0
                                                 then goto vrij;
                                                 klaar:= false;
                                                 verlies[s,v]:= true;
                                           vrij:
                                           end;
                if ¬ klaar then goto spin;
eind:           for s:= 1 step 1 until n do
                    for v:= 1 step 1 until v do print(zet[s,v])
end end
```

V.2. Commentaar op semantiek en syntax van het gegeven programma.

Het programma is een block bestaande uit een blockhead gevolgd door een compound tail (4.1.1). Het blockhead bestaat uit een begin symbol en een (type) declaratie. De compound tail bestaat uit een assignment statement, een block, dat als binnenblock zal worden aangeduid, en een end symbol. Merk op dat hierbij gebruik gemaakt is van de equivalentie tussen
; comment <any sequence not containing ;>;
en
;
als in 2.3 vastgelegd.
Het binnenblock bestaat weer uit een blockhead waarvan de declaraties type of array declaraties zijn.

Opgave: Ga na n.a.v. de secties 5, 5.1, 5.2.1, 5.2.2, 5.2.3, 5.2.4.2 dat deze declaraties zowel syntactisch als semantisch correct zijn.

Het eerste statement van het binnenblock is een for statement, waarvan de syntactische ontleding gesuggereerd wordt door

<label>: for <variable>:= <arithmetic expression> step
<arithmetic expression> until <arithmetic expression> do
<statement>

In 4.1.1. staat vastgelegd dat een <for statement> een produktie van <statement> kan zijn, zoals hier het geval is. Dit laatste for statement bevat als statement een assignment statement (4.1.1 en 4.2).

Opgave: ontleedt pad[s,v]:= read = 1; merk i.h.b. op dat de boolean expressie read = 1 een produktie van <relation> is die t.g.v. 3.4.5 een boolse waarde bezit.

Na executie van een for-statement b.v.

for v:= 1 step 1 until n do pad[s,v]:= read = 1

geldt dat de waarde van de "controlled variable" v ongedefinieerd is; als uitzondering hierop geldt, dat, indien de executie beëindigd wordt door

een sprong uit het bevatte statement "naar buiten het for statement", de waarde van de controlled variable dezelfde is als direct vóór executie van het goto statement (4.6.5.). Ga dit na door 4.6.5 ook te lezen.
Van buiten naar binnen een for statement springen is verboden door 4.6.6 (daar de controlled variable dan geen waarde bezit).

Opgave: ontleedt de voorbeelden van 4.6.2 en ga hun executie i.h.b. na n.a.v. 4.6.4.

Van het conditional statement, van de vorm

if BE then S1 else S2 of if BE then S1

wordt vermeld dat een sprong van buiten het conditional statement naar binnen, indien naar een statement binnen S1 of S1 gesprongen wordt, tot gevolg heeft dat tot executie van de relevante gedeelten van S1 wordt overgegaan, na beëindiging waarvan tot executie van het op het conditional statement volgende statement, indien aanwezig, wordt overgegaan, tenzij expliciet een opvolger wordt aangewezen dmv. een verlating van S1 door een goto statement. Hieruit valt af te leiden wat er gebeurd als er naar S2 of naar een statement binnen S2 gesprongen wordt.

Opgave: ga in het geval dat naar een statement binnen S1 gesprongen wordt na, m.b.v. 4.5.1, 4.1.1, 4.6.6 en 4.3.4, dat S1, wil het conditional statement correct zijn, een compound statement moet zijn.

een sprong uit het bevatte statement "naar buiten het for statement", de waarde van de controlled variable dezelfde is als direct vóór executie van het goto statement (4.6.5.). Ga dit na door 4.6.5 ook te lezen.
Van buiten naar binnen een for statement springen is verboden door 4.6.6 (daar de controlled variable dan geen waarde bezit).

Opgave: ontleedt de voorbeelden van 4.6.2 en ga hun executie i.h.b. na n.a.v. 4.6.4.

Van het conditional statement, van de vorm

if BE then S1 else S2 of if BE then S1

wordt vermeld dat een sprong van buiten het conditional statement naar binnen, indien naar een statement binnen S1 of S1 gesprongen wordt, tot gevolg heeft dat tot executie van de relevante gedeelten van S1 wordt overgegaan, na beëindiging waarvan tot executie van het op het conditional statement volgende statement, indien aanwezig, wordt overgegaan, tenzij expliciet een opvolger wordt aangewezen dmv. een verlating van S1 door een goto statement. Hieruit valt af te leiden wat er gebeurd als er naar S2 of naar een statement binnen S2 gesprongen wordt.

Opgave: ga in het geval dat naar een statement binnen S1 gesprongen wordt na, m.b.v. 4.5.1, 4.1.1, 4.6.6 en 4.3.4, dat S1, wil het conditional statement correct zijn, een compound statement moet zijn.

VI. Het procedure statement (4.7).

VI.1. Name replacement.

Op de betekenis van een procedure aanroep is reeds in I.8 ingegaan (recapituleer dus I.8). Deze kwam er ruwweg op neer dat

> de executie van een procedure aanroep neerkwam op de executie van de bijbehorende procedure body, waarin, al naar gelang het voorkomen van parameters, veranderingen waren aangebracht.

Dit is volgens het RR ook de betekenis van executie van het procedure statement (4.7.3.3). Je maakt a.h.w. een kopie van de procedure body op een kladje papier en brengt daar de in 4.7.3.2 en 4.7.3.3 voorgeschreven veranderingen in aan om het effect van executie van het desbetreffende procedure statement na te gaan; dit heet de copy rule.
Deze veranderingen zijn vooral tgv. de scope van de in de body voorkomende identifiers gecompliceerder dan in I.8 neergelegd. Ze worden hieronder gegeven en toegelicht.

——— "Any formal parameter ... called by name ... is replaced, throughout the procedure body by the corresponding actual parameter, after enclosing the latter in parentheses wherever syntactically possible". (4.7.3.2)

Toepassing 1:
```
begin procedure Q(u,v); u:= v*v;
        integer a,b;
        a:= b:= 17; Q(a,a+b); ...
        ...
    end
```

De aanroep Q(a,a+b) heeft ten gevolge dat

```
a:= (a+b)*(a+b)
```

wordt geëxecuteerd en niet

```
a:= a+b*a+b   of (a):= (a+b)*(a+b);
```

dit laatste is syntactisch "not possible".

——— "Possible conflicts between identifiers inserted by this process (of name replacement) and other identifiers already present within the procedure body will be avoided by systematic changes of the formal (zie I.8) or local identifiers involved". (4.7.3.2)

Toepassing 2: Beschouw de volgende programmaschets:

```
begin integer a,b,i,u;
      integer array s[1:10];
      procedure wissel(u,v); integer u,v;
            begin integer s; s:= u:= v; v:= s end;
      ...
      wissel (a,b); ...
      i:= 7; wissel (u,s[i]); ...
      ...
end
```

Bij wissel (a,b) levert de copy rule geen problemen op doch bij wissel (u,s[i]) wel, daar je na de systematische verandering van de formele parameters door de ermee corresponderende actuele parameters

```
begin integer s; s:= u; u:= s[i]; s[i]:= s end
```

verkrijgt.
De scope van de array identifier s van s[i] strekt zich niet meer tot deze rij basic symbols uit, daar de identifier s erin geherdeclareerd wordt als integer.
Zodoende zijn er dus "conflicts" gerezen tussen "identifiers inserted by this process ... and other identifiers already present within the procedure body". Deze worden voorkomen door "systematic changes of the ... local identifiers involved", dwz. de in de procedure body van s gedeclareerde integer s krijgt een willekeurige andere naam, die geen conflicten met zich meebrengt.
Executie van wissel (u,s[i]) komt dus neer op executie van

```
begin integer s1; s1:= u; u:= s[i]; s[i]:= s1 end
```

daar "identifiers have no inherent meaning" (2.4.3).

—— "If the procedure is called from a place outside the scope of any non-local quantity of the procedure body the conflicts between the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable systematic changes of the latter identifiers". (4.7.3.3)

```
Toepassing 3: begin integer a;
                    procedure P; print (a);
                    a:= 1;
                    begin real a; a:= 3.14;
                          P
                    end
              end
```

Rechtstreekse toepassing van de copy rule op de aanroep van P levert print(a) op, waarin a een variable van het type real dreigt te identificeren, daar, ofschoon de procedure body van P de "non-local quantity" a aanroept, ter plekke van de aanroep van P, a ook een "identifiers is whose declaration is valid at the place of the procedure statement", waardoor er een conflict rijst tussen de real variable a en de integer variable a "inserted by this process of (name replacement)". De strekking van de aangehaalde zin uit (4.7.3.3) is nu dat de in het binnenblock gedeclareerde integer a van naam wordt veranderd; wélke naam doet er niet toe, zolang daardoor geen nieuwe conflicten rijzen, daar "identifiers have no inherent meaning".
Als sluitstuk moge nu de eerste zin van 4.7.3.3 gelden:

> "Finally the procedure body, modified as above, is inserted in place of the procedure statement and executed".

Welke klassen ALGOL 60 grootheden mogen nu als actuele parameters in de parameterlijst bij het call-by-name parameter mechanisme worden aangeroepen?
Dat staat in de eerste regel van 4.7.1 neergeschreven; deze luidt:

<actual parameter>::= <string>|
<expression>
<array identifier>|
<switch identifier>|
<procedure identifier>

De klasse van deze parameters kan, doch behoeft niet, in de "specification part" van de procedure declaratie (5.4.1, 5.4.5) worden meegegeven, i.t.t. de handelingen bij gebruik van het call-by-value parameter mechanisme (VI.2).

Voorbeeld 1: De klasse van de actual parameter is <string> :

```
PRINTTEXT (‡OK‡)
```

Voorbeeld 2: De klasse van de actual parameter is <expression> en valt uitéén in de klassen <arithmetic expression> en <designational expression>.
Beschouw hiertoe een aanroep van de functie procedure MOD, die van 2 integers de kleinste niet negatieve rest bij deling van de eerste door de tweede als waarde toegewezen krijgt, echter een "nooduitgang" bezit voor het geval dat deeltal gelijk nul 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 mogelijke aanroep is:

```
MOD(n*(2*n+1),m*(m+7)-3, m is fout) .
```

Merk op dat bij executie van deze aanroep

```
m*(m+7)-3
```

wel 4 keer kan worden geëvalueerd, ofschoon toch steeds dezelfde waarde wordt afgeleverd. Dit is zeer inefficiënt en een treffend voorbeeld waarin het call-by-value parameter mechanisme uitkomst brengt. Zie VI.2.

Voorbeeld 3: Onder de actuele parameters komen er voor van de klassen

<array identifier> en <procedure identifier>.

Beschouw hiertoe aanroepen van

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

Deze procedure berekent een rij waarden van functieprocedure-aanroepen als functieprocedure en rij argumenten gegeven zijn. Als x en y identifiers van real array's zijn van 1 dimensie, met bovengrens $\leq$ 10 en ondergrens $\geq$ 1, dan is

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

een correcte aanroep.

Opmerking: Uit dit voorbeeld blijkt dat de specificaties, door <specifics> in 5.4.1 gegeven, onvolledig zijn en derhalve geen declaraties kunnen zijn, daar b.v. de array grenzen niet worden meegegeven.

VI.2. Jensen's device en het call-by-value parameter mechanisme.

Beschouw een programma waarin procedures met parameters gedeclareerd zijn en veronderstel dat uitsluitend het call-by-name parameter mechanisme wordt toegepast. Indien een expressie als actuele parameter wordt meegegeven - een expressie is een regel om een boolean, integer, real value of een label uit te rekenen; dit volgt o.m. uit 3 - dan kan deze expressie één of meer keren bij executie van de desbetreffende procedure aanroep geëvalueerd moeten worden.
Aanroepen van procedures kunnen in dat geval, wat het substitutie mechanisme betreft, in 2 categorieën verdeeld worden nl.

i) de waarde van de expressie verandert elke keer dat deze berekend wordt,
ii) de waarde van de expressie blijft hetzelfde elke keer dat deze berekend wordt.

Bespreking van categorie i.

Gegeven de declaratie

```
integer procedure Sum(i,ai,n); integer i,ai,n;
begin integer hulp; hulp:= 0;
      for i:= 1 step 1 until n do hulp s = hulp + ai;
      Sum:= hulp
end
```

en beschouw de aanroep Sum(j,1/j,10).
Om de waarde van deze function designator te bepalen moet

```
begin integer hulp; hulp:= 0;
      for j:= 1 step 1 until 10 do hulp:= hulp + 1/j;
      Sum:= hulp
end
```

geëxecuteerd worden;

als j = 1 dan is de waarde van de voor ai gesubst. expr. 1/j .... 1 ,
als j = 2 id. 1/j 1/2,
als j = 3 id. 1/j .... 1/3 enz.

De voor ai gesubstitueerde expressie wordt bij deze aanroep 10 maal geëvalueerd en levert 10 verschillende waarden af doordat de als call-by-name parameter meegegeven expressie 1/j telkens van waarde verandert wanneer een andere call-by-name parameter - j - een andere waarde toegewezen krijgt.

Dit verschijnsel,

> de waarde van een als call-by-name meegegeven parameter expressie verandert van waarde doordat een aan in die expressie voorkomende actuele parameter in de procedure body telkens een andere waarde krijgt toegewezen, heet
>
> "Jensen's device".

Een andere toepassing daarvan is de aanroep

```
Sum(j,a[j],n)                                                    ;
```

het array a[1:n] is voor de aanroep reeds "gevuld".

Opgave: ga na welke waarde er bij evaluatie van deze function designation berekend wordt.

Bespreking van categorie ii.

Dit betreft het geval dat

> de waarde van de voor de formele parameter gesubstitueerde expressie hetzelfde blijft elke keer dat deze geëvalueerd wordt.

De in VI.1, voorbeeld 2, aangehaalde aanroep

```
MOD(n*(2*n+1), m*(m+7)-3, m is fout)
```

van de daar gegeven integer procedure MOD(p,q,L) brengt met zich mee dat,

de voor de formele parameter q gesubstitueerde actuele parameter wel 4 maal geëvalueerd kan worden, niettegenstaande het feit dat "het de bedoeling is" dat 4 maal dezelfde waarde afgeleverd wordt Derhalve is de volgende versie van MOD zowel efficiënter als duidelijker, daar "de bedoeling" er duidelijker uit spreekt (deze is in VI.1 voorbeeld 2 onder woorden gebracht).

```
integer procedure MOD1 (p,q,L); integer p,q; label L;
begin integer ploc, qloc; ploc:= p; gloc:= q;
      begin integer s;
            if qloc = 0 then goto L;
            s:= ploc - ploc : qloc * qloc;
            if s < 0 then MOD1:= s + abs(qloc) else MOD1:= s
end end
```

De als volgt gedeclareerde procedure

```
integer procedure MOD2(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 MOD2:= s + abs(q) else MOD 2:= s
end
```

heeft precies hetzelfde effect bij aanroep met dezelfde parameters als de overeenkomstige aanroep bij gebruik van MOD1;
MOD2 is een afkorting van MOD1, verkregen door p en q in de value lijst (4.7.3.1 en 5.4.3) op te nemen.

Het is nu mogelijk 4.7.3.1 te appreciëren; lees derhalve deze sectie.

Opmerking 1: Als een parameter in de value lijst wordt opgenomen, wordt specificatie van de parameter door het RR in 5.4.5 geëist (daar de typen van de parameter "local to this fictitious block" duidelijk moeten zijn alvorens ze "fictitiously" te declareren).

Opmerking 2: Welke zijn de klassen van de in de value lijst op te nemen parameters?
Dit zijn parameters die waarden afleveren; deze staan in 2.8 beschreven en zijn:

expressies van de typen integer, real, boolean met als specifiers (5.4.1) integer, real, boolean,
arrays van de typen integer, real, boolean met als specifiers integer array, real array of array, boolean array
of designational expressions - expressies die een label als waarde afleveren - met als specifier label.

Opname van een array in de value lijst brengt met zich mee dat in het "fictief" om het procedure body geslagen block een locaal array van dezelfde dimensie en met dezelfde grenzen als het als actuele parameter meegegeven array gedeclareerd wordt en de waarde van dit array "subscripted variable voor subscripted variable" toegewezen krijgt.

VI.3. Recursieve aanroepen van procedures en recursieve procedures.

Beschouw de aanroep

```
Sum(i,Sum(j,a[i,j],n),m)
```

van de in VI.2 behandelde integer procedure Sum(i,ai,n).
Bij evaluatie van deze function designator wordt geëxecuteerd

```
begin integer hulp; hulp:= 0;
      for i:= 1 step 1 until m do hulp:= hulp + Sum(j,a[i,j],n);
      Sum:= hulp
end
```

en voor elk van die m waarden van i wordt, lokaal ten opzichte van dit block,

```
begin integer hulp; hulp:= 0
      for j:= 1 step 1 until n do hulp:= hulp + a[i,j];
      Sum:= hulp
end
```

geëxecuteerd.
Het effect van deze aanroep is, dat de som van de waarden van de subscripted variables a[i,j], i = 1,...,m en j = 1,...,n berekend wordt.

Daar voor evaluatie van deze function designator

meerdere, in elkaar "geneste" keren, de passend gemodificeerde tekst moet worden geïnserreerd

heet dit een recursieve aanroep van de functieprocedure Sum.
Een aanroep van Sum behoeft dit niet te zijn; voor recursieve procedures geldt dit in de regel wel: al in hun procedure body roepen ze zichzelf aan.

Een bekend voorbeeld van een recursieve procedure is de functie procedure voor het berekenen van de faculteit van een getal:

```
integer procedure fac(n); value n; integer;
fac:= if n=0 then 1 else fac(n-1) * n
```

Evaluatie van fac(2) kan de lezer zich als volgt voorstellen:

```
      fac(2)
=  ┌──────────────────────────────────────────────────────────────────
   │  n:= 2
   │  fac:= if n=0 then 1 else fac(n-1) × n
   │      = if 2=0 then 1 else fac(n-1) × n
   │      = if false then 1 else fac(n-1) × n
   │      = fac(n-1) × n
   │      = ┌──────────────────────────────────────────────────────────
   │        │  n':= n-1 = 2-1 = 1
   │        │  fac:= if n'=0 then 1 else fac(n'-1) × n'
   │        │      = if 1=0 then 1 else fac(n'-1) × n'
   │        │      = if false then 1 else fac(n'-1) × n'
   │        │      = fac(n'-1) × n'
   │        │      = ┌──────────────────────────────────────────────────
   │        │        │  n":= n'-1 = 1-1 = 0
   │        │        │  fac:= if n" = 0 then 1 else fac(n"-1) × n"
   │        │        │      = if 0 = 0 then 1 else fac(n"-1) × n"
   │        │        │      = if true then 1 else fac(n"-1) × n"
   │        │        │      = 1
   │        │        └──────────────────────────────────────────────────
   │        │      = 1 × n' = 1 × 1
   │        └──────────────────────────────────────────────────────────
   │      = 1 × n = 1 × 2
   │      = 2
   └──────────────────────────────────────────────────────────────────
= 2
```

Merk op hoe vaak 4.7.3.1 en in zekere zin 4.7.3.2 hierin toegepast zijn.

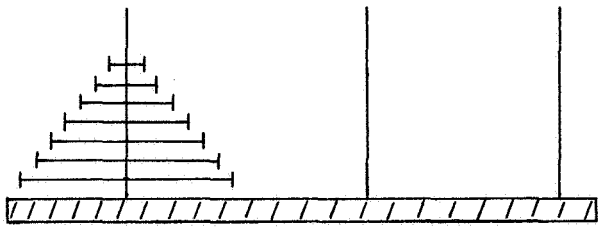
Opgave: Lees 5.4 volledig door.

Merk op dat de in deze sectie behandelde facetten van het procedure mechanisme gerechtvaardigd worden door de laatste zin van 5.4.4:

> "Any occurrence of the procedure identifier within the body of the procedure other than is a left part of an assignment statement denotes activation of the procedure".

Laatste opgave:

In het spel "de drie torens van Hanoi" beschikt men over 3 pinnen op een plankje; op één van die pinnen ligt een van onderen af in afnemende grootte gerangschikte stapel van n schijven:



Het spel bestaat eruit dat men de stapel schijven van de ene pin naar een andere pin brengt, de volgende regels in acht nemende:

(1) per keer één schijf verplaatsen naar een andere pin,
(2) nooit een grotere of een kleinere schijf zetten.

Speel het spel (op papier voor n = 1, n = 2, n = 3 en n = 4 schijven en ontwikkel een strategie waarmee het spel voor k schijven, k ≥ 2, wordt teruggebracht tot het spel voor k-1 schijven, dwz. probeer een recursieve strategie te ontwikkelen.
Pas daarna werpt de lezer een blik op het onderstaande programma, indien hij dat niet al gedaan heeft, probeert dat te begrijpen en probeert in te zien dat dit een schoolvoorbeeld is van een op zijn plaats zijnde, natuurlijke beschrijving van het spelen en de oplossing van dit spel.

```
procedure TORENS VAN HANOI (n,van,naar); value n, van, naar;
          integer n, van, naar;
          if n > 1 then
             begin TORENS VAN HANOI (n-1,van,6-van-naar);
                   TORENS VAN HANOI ( 1 ,van,   naar   );
                   TORENS VAN HANOI (n-1,6-van-naar,naar)
             end
          else
             begin NLCR; print(van); print(naar) end
```