TECHNISCHE HOGESCHOOL EINDHOVEN
Afdeling Algemene Wetenschappen
Onderafdeling der Wiskunde

WISKUNDE 31

bestemd voor

BDK-III

WISKUNDE 49

bestemd voor

N-IV, W-IV, E-IV, T-IV, B-IV

INLEIDING INFORMATICA

Prof. Dr. E.W. Dijkstra

1976

2259 allen Bibl Mey

[Leeszaal]
Centrale Bibliotheek
T.H. Eindhoven

Technische Hogeschool Eindhoven

Onderafdeling der Wiskunde

Wiskunde 31

bestemd voor BDK-III

Wiskunde 49

bestemd voor N-IV, W-IV, E-IV, T-IV, B-IV

Inleiding Informatica

Wij verzoeken U , dit collegedictaat niet mee te nemen buiten de leeszaal en het na lezing terug te leggen op de ladenkasten. Dank U !

Dictaatnr. 2.259 Prijs *f* 2,50

2.259

Inhoudsbeschrijving
WISKUNDE 41 EN 49
Inleiding Informatica
1976

ALGORITHMEN

Inleiding	0
Automaten	2
Rekenautomaten	5
Een eerste programma	7
Voorwaardelijke uitvoering van statements	12
Herhaalde uitvoering van statements	16
Arrays	19
Logische expressies, variabelen en operatoren	23
De opbouw van een groter programma	28

PROGRAMMEREN in ALGOL 60

§1. Assignment statement	2
§2. Identifiers en numbers	3
§3. Simple arithmetic expressions	5
§4. Het type van variables en simple arithmetic expressions	8
§5. Conditional statement	10
§6. While statement	14
§7. Do-until statement	16
§8. For statement	17
§9. Arrays	19
§10. Block structuur	22
§11. Procedure	24
ALPHABETISCHE INDEX	29

JdG, 12 Juli 2005

Leeszaal
Centrale Bibliotheek
T.H. Eindhoven

ALGORITHMEN

Inleiding.

In het begin van de negende eeuw na Christus werkte in de bibliotheek van het toen pas gestichte Bagdad een zekere Muhammed ben Musa, bijgenaamd al-Khwarizmi, die het eerste Arabische boek over algebra schreef. (Het woord "algebra" is een verbastering van de Arabische titel!) Behalve dat schreef hij een handleiding voor het zg. Indisch rekenen, dwz. het rekenen in het tientallig stelsel. Enige eeuwen later heeft dit boek (in Latijnse vertaling) een grote rol gespeeld bij de invoering van het tientallig stelsel in Europa; de bijnaam van de auteur, al-Khwarizmi, leeft tot op de huidige dag voort in het woord "algorithme", dat sedert die tijd in zwang is voor "rekenvoorschrift", in het algemeen voor "handelingsvoorschrift".

Ook al komt het woord algorithme in het dagelijkse leven nu niet zoveel voor, met het idee van een algorithme worden we dagelijks geconfronteerd. Sprekende voorbeelden van algorithmen uit het dagelijkse leven zijn breipatronen, recepten en montagevoorschriften. Wie een ander naar de weg vraagt, vraagt in wezen naar een algorithme: hij vraagt naar een handelingsvoorschrift dat hij kan opvolgen om zijn doel te bereiken.

Aan deze alledaagse algorithmen kunnen wij al een aantal typische eigenschappen illustreren. Essentieel is, dat een of andere totale verrichting ontleed wordt in, dan wel opgebouwd wordt uit een aantal op zichzelf simpelere handelingen: het breien van een truitje wordt uitgedrukt in termen van het breien van toeren, die op hun beurt beschreven worden door een bepaalde opeenvolging van speciale steken, minderingen en meerderingen. Als onderdeel van een recept voor ragout zal men vinden, dat een niet te grote ui gesnipperd en goudbruin gefruit moet worden, etc.

Verder observeren wij, dat hele klassen algorithmen herleid worden tot handelingen uit een repertoire, dat duidelijk bij die klasse hoort, waarbij als regel elke handeling uit het repertoire in allerlei algorithmen voorkomt. Zo zal het deelvoorschrift om een uitje te fruiten als onderdeel in allerlei

Jan 76

recepten voorkomen, zo komen typische breihandelingen (zoveel steken zus, zoveel steken zo, minderen etc.) in allerlei breipatronen voor.

Vervolgens zien wij, dat een algorithme alleen dan als zodanig functioneert, wanneer tussen opsteller en uitvoerder geen misverstand of verschil van mening bestaat over het repertoire handelingen, waaruit de totale verrichting moet worden opgebouwd. Breipatronen en montagevoorschriften zijn in dit opzicht vrij zuivere algorithmen; recepten zijn in dit opzicht al wat troebeler ("zout naar smaak toevoegen" is niet zo duidelijk; het is dan ook geen toeval, dat een gerecht eerder mislukt dan een breiwerk). Bij het vragen naar de weg is de verwarring over wat toegelaten handelingen zijn doorgaans volkomen; ze zijn als algorithme dan ook zelden zonder dubbelzinnigheid uitvoerbaar.

Tenslotte moet een algorithme nog een zekere innerlijke logica hebben. Als er in een recept, waarin nog niet over bouillon geschreven is, staat "Laat vervolgens de bouillon afkoelen", dan is er iets mis, ook al weet de uitvoerder op zich zelf best, hoe hij bouillon volgens de regels van de kunst moet laten afkoelen! Toen ik laat op een wat mistige avond in het Botlek verdwaald, bij een tankstation naar de richting Dordrecht vroeg, kreeg ik te horen "Rechtuit en dan bij de laatste stoplichten linksaf."!

Hebben wiskundigen zich dus al sinds eeuwen met algorithmen bezighouden, in de twintigste eeuw zijn algorithmen opnieuw in het middelpunt van de wiskundige belangstelling komen te staan.

De eerste aansporing daartoe kwam van het grondslagenonderzoek, waarin men zich de vraag ging stellen aan welke eisen een deugdelijk wiskundig bewijs nu eigenlijk moest voldoen. Dit was een soort "wiskundig gewetensonderzoek, dat nodig was geworden omdat men tegen allerlei paradoxen aanstootte. Dat men hierbij algorithmen tot voorwerp van studie maakte is niet zo verwonderlijk als we bedenken, dat een bewijs een speciaal soort handelingsvoorschrift is, nl. een redeneervoorschrift om tot een bepaalde conclusie te komen. De moeilijkheden bleken voort te komen uit bewijsstappen, waarbij niet gezegd werd, hoe je ze moest uitvoeren. Wij zullen deze logische schermutselingen niet verder vervolgen. Ter aanduiding van de aansluiting

met wat volgt vermelden we slechts, dat de gedachteconstructie van een van de geniaalste onderzoekers op dit gebied, Alan M.Turing, onder de naam "Turing Machine" wereldvermaard is geworden.

De tweede aansporing om zich op algorithmen te concentreren kwam door de ontwikkeling van de electronische rekenautomaten (ook wel computers of rekenmachines genaamd). Deze machines zijn nl. in staat om algorithmen (van een bepaalde klasse) in zich op te nemen en vervolgens getrouwelijk uit te voeren. Rekenautomaten zijn dus machines, die algorithmen kunnen uitvoeren; voor algorithmen, die bestemd zijn om door een rekenautomaat uitgevoerd te worden is de naam "programma" in zwang gekomen; het opstellen van programma's heet "programmeren" en degene, die dit doet, heet "programmeur". Door de komst van de rekenautomaten is de belangstelling voor algorithmen drastisch veranderd: van een uiterst theoretische belangstelling is het, door de grote toepassingsmogelijkheden van rekenautomaten en hun economische importantie geworden tot een buitengewoon practische aangelegenheid. Bovendien, was aanvankelijk de belangstelling voor algorithmen in hoofdzaak analytisch, nu is de belangstelling in hoge mate synthetisch: er moeten algorithmen van allerlei soort gemaakt worden, willen wij de mogelijkheden van rekenautomaten kunnen benutten. In de afgelopen jaren heeft het programmeren zich ontwikkeld tot een van de meest creatieve loten aan de wiskundige stam.

Automaten.

Dit college zal in hoofdzaak over programmeren gaan, maar daartoe moeten wij eerst min of meer een indruk geven van wat een rekenautomaat kan, uit welke functionele componenten zo'n machine is opgebouwd. Uit het dagelijkse leven zijn dergelijke machines nauwelijks bekend, we kunnen wel diverse automatische mechanismen noemen, die ieder een of ander aspect illustreren.

Om te beginnen zijn er de roltrappen van de Maastunnel. 's Nachts staan deze roltrappen stil. Wie bij de ingang een neergaande of bij de uitgang een opgaande roltrap oploopt, onderbreekt daarbij een lichtstraal die op een fotocel pleegt te vallen en ten gevolge van deze onderbreking zet de trap zich in beweging en rolt hij (ruim) eenmaal zijn eigen lengte af. Wie zich dan daar-

door laat transporteren en aan de andere kant de trap verlaat merkt, dat even later de trap achter hem weer tot stilstand komt, tenzij inmiddels opnieuw iemand de trap is opgegaan en de straal heeft onderbroken: de roltrap rolt zijn eigen lengte af sinds de laatste onderbreking van de straal. Dit betekent, dat vanaf een bepaald ogenblik -nl. de onderbreking van de straal- het mechanisme automatisch een vaste handeling verricht, autonoom "een vast programma" afwerkt, nl. zijn eigen lengte eenmaal afrollen. Er moet een bepaald aantal treden "verrold" worden; als de trap tot ergens halverwege in de afrolling gevorderd is, ergens halverwege in de afwerking van zijn programma, dan zit kennelijk ergens in het mechanisme een geheugenelement, dat bv. bijhoudt over hoeveel treden nog doorgerold moet worden, totdat de trap weer moet stoppen. Wij kunnen ons voorstellen, dat de roltrap is uitgerust met een zg. teller, een geheugenelement, waarin genoteerd staat het aantal treden, waarover de trap nog rollen moet. Wie de lichtstraal onderbreekt, zorgt daardoor dat dit geheugenelement gevuld wordt met het maximale bedrag (= aantal zichtbare treden van de trap + een beetje extra), als de trap loopt wordt de inhoud van dit geheugenelement per treetransport met 1 verminderd en als de inhoud van dit geheugenelement terugkeert tot nul, stopt de trap.

In de praktijk worden dergelijke geheugenelementen gerealiseerd door een object, dat zich in een groot aantal discrete toestanden kan bevinden, waarbij men met elke toestand van het geheugenelement een waarde associeert. Het element "vullen met een bepaald bedrag" betekent "het element brengen in de met dit bedrag geassocieerde toestand". Typerend voor het hele automatisme is:
1) de aanwezigheid van een dergelijk geheugenelement
2) de beinvloeding van de inhoud van dit geheugenelement door de activiteit van het mechanisme (elke tredeverschuiving gaat gepaard met een vermindering van de inhoud met 1)
3) de beinvloeding van het mechanisme door de inhoud van het geheugenelement (trap stoppen als deze inhoud tot nul is teruggekeerd)
4) het startsignaal (hier de onderbreking van de lichtstraal) waardoor het geheugenelement in de aanvangstoestand gebracht wordt.

Vergeleken bij de processen, die van moderne rekenautomaten gevergd worden, is wat van de roltrap verlangd wordt kinderachtig eenvoudig. Waar de roltrap uitkomt met een enkel geheugenelement, kan men in een moderne rekenautomaat duizenden van dergelijke geheugenelementen aanwijzen, die allemaal

dienen ter vastlegging, ter onderscheiding van de interne toestanden, waarin het rekenproces zich bevinden kan. Mede door dit enorme aantal geheugenelementen is een rekenmachine een heel andere automaat geworden dan de besturing van de roltrap; de functie van de geheugenelementen is in principe echter dezelfde.

Er is een ander, drastisch verschil tussen de moderne rekenautomaat en de roltrap. Wat van de roltrap gevergd wordt is eigenlijk altijd hetzelfde, nl. een standaard reactie op een uniek startsignaal. Van de moderne rekenautomaat verlangt men, dat hij allerlei rekenprocessen kan uitvoeren, allerlei algorithmen in zich kan opnemen en gehoorzamen. Men drukt dit wel uit door te zeggen, dat een rekenautomaat een "general purpose" machine is; het wordt iets duidelijker tot uitdrukking gebracht door te zeggen, dat moderne rekenmachines "programmeerbare automaten" zijn.

Er zijn andere apparaten, waarin we dit aspect van programmeerbaarheid al terugvinden. In de technische sfeer vinden we dit bij het weefgetouw van Jacquard, in de amusementssfeer vinden we dit terug bij het beter bekende draaiorgel. Beide worden gestuurd door een "boek", dat bestaat uit een lange reep kaarten, waarin gaatjes de uit te voeren handelingen voorschrijven. Bij het weefgetouw van Jacquard leggen de gaatjes (dwz. de aan- resp. afwezigheid van gaatjes) vast, welke draden van de schering dit keer omhoog moeten, opdat het uiteindelijke damasten tafelkleed de Franse lelie, dan wel het familiewapen vertoont; bij het draaiorgel leggen de gaatjes vast welke pijpen moeten worden aangeblazen en welke balgen voor trommels en belletjes bediend moeten worden.

Enerzijds zijn deze apparaten specifiek: je kunt het weefgetouw alleen maar voor weven gebruiken en het draaiorgel alleen maar om muziek te maken. Zo is ook de "general purpose" rekenmachine specifiek: je kunt hem rekenprocessen laten uitvoeren.

Anderzijds zijn ze, <u>op hun gebied</u>, tamelijk algemeen! Voor het weefgetouw van Jacquard betekent dit, dat het <u>allerlei</u> patroontjes kan weven, van Franse lelies tot familiewapens toe en juist door die algemeenheid door het boek gestuurd moet worden om te bepalen, wat er dit keer geweven moet worden. Voor

het draaiorgel betekent dit, dat het zoals het uit de fabriek komt, allerlei muziek kan maken, van psalmen en volksliederen tot tophits en juist door die algemeenheid het draaiboek nodig heeft opdat vastligt, wat er dit keer voor muziek gemaakt moet worden. Zo ook voor de moderne programmeerbare rekenautomaat: hij kan allerlei algorithmen uitvoeren en het is juist deze algemeenheid die maakt, dat hij steeds een programma nodig heeft ter beschrijving van de gedragslijn, die dit keer gevolgd moet worden.

Rekenautomaten.

De rekenautomaat kan dus programma's uitvoeren, mits zo'n programma aan de machine is toegevoerd. Analoog aan het boek bij het weefgetouw en het draaiorgel wordt het programma als regel geponst in hetzij ponskaarten (in klein model bekend van de postgiro) dan wel in ponsband (in smal model sinds lang bekend bij het telexverkeer). De rekenmachine is uitgevoerd met een zg. "lezer" (kaartlezer of ponsbandlezer), waardoor de informatie in een pak kaarten dan wel een ponsband kan worden afgetast. Geschiedt dit bij het weefgetouw mechanisch (nl. door tastpinnetjes, die afhankelijk van de aanwezigheid van een gat al of niet worden tegengehouden) en bij het draaiorgel pneumatisch (waar afhankelijk van de aanwezigheid van een gat een pijp wel of niet wordt aangeblazen), bij moderne rekenautomaten geschiedt het aftasten (om der wille van de snelheid) meestal optisch: wel een gat laat meer licht door dan geen gat.

Hier houdt overigens de analogie tussen weefgetouw en draaiorgel enerzijds en rekenautomaat anderzijds op. Weefgetouw en draaiorgel beschikken niet over een groot geheugen (dwz. een groot aantal geheugenelementen) en elke regel van het draaiboek wordt "uitgevoerd" op het moment, dat deze regel onder de aftasters van het leesstation ligt. Bij een draaiorgel heeft dit bv. tot gevolg dat de enige manier om zestien maten muziek te laten herhalen is om deze zestien maten twee keer in het immers doordraaiende boek te laten voorkomen. De rekenautomaat beschikt echter wel over een groot geheugen, dat onder andere gebruikt wordt om eerst de hele algorithme op te nemen, voordat aan de werkelijke uitvoering ervan begonnen wordt. In een programma kunnen we, zoals we later zullen zien, wel aangeven, dat bv. een stukje nog een keer herhaald

moet worden (zoals ook in bladmuziek!).

We zijn inmiddels twee componenten van de rekenmachine tegengekomen, het leesstation via hetwelk het geponste programma aan de machine wordt toegevoerd en het geheugen, waarin dit programma wordt opgenomen. Als het programma in zijn geheel is opgenomen, dan begint het eigenlijke rekenwerk: het handelingsvoorschrift wordt opgevolgd, de berekening wordt uitgevoerd. Hierbij speelt het derde onderdeel van de machine een belangrijke rol, het zg. "rekenorgaan". Tijdens het rekenproces speelt het geheugen een dubbele rol: ten eerste wordt het geraadpleegd omdat het uit te voeren programma er in is opgeslagen, ten tweede wordt het gebruikt als "kladpapier" om voor het rekenproces belangrijke tussenresultaten er zolang in op te slaan. (Dit tweede gebruik van het geheugen is analoog aan het geheugenelement van de roltrap.) Tenslotte bevat de rekenmachine apparatuur voor de vierde functie, nl. het aan de buitenwereld terugmelden van de gevraagde uitkomsten. Dit kan via bandponsers, kaartponsers, regeldrukkers, tekentafels, cathodestraalbuizen etc. (In de wandeling heten dit "uitvoerorganen" omdat via deze apparatuur de informatie weer de machine wordt uitgevoerd; de lezers heten "invoerorganen". Omdat men in het Nederlands ook spreekt over "het uitvoeren van een programma" zou in plaats van invoer- en uitvoerorganen "import- en exportorganen" een prettigere terminologie geweest zijn. In het Engels spreekt men van "Input", "Output" en "Execution".)

Naast een grote rekenautomaat beschikt het Rekencentrum van de THE over een ponskamer. Hierin staat een groot aantal zg. Flexowriters opgesteld. Dit zijn electrische schrijfmachines, waaraan een bandponser is gekoppeld; elke keer, dat men een toets van de schrijfmachine inslaat wordt een bij die toets behorende configuratie van wel en niet gaatjes in de band geponst. Men levert zijn programma netjes geschreven in; deze tekst wordt dan op een Flexowriter een keer overgetypt, waardoor de informatie tevens in een ponsband wordt vastgelegd. Het is deze ponsband, die uiteindelijk aan de rekenmachine wordt aangeboden.

Zoals gezegd mag er tussen de opsteller en de uitvoerder van een algorithme geen misverstand bestaan over het repertoire handelingen, waaruit de

totale verrichting moet worden opgebouwd. Met betrekking tot een programma, dat bestemd is om door een machine te worden uitgevoerd betekent dit, dat de programmatekst aan heel precieze regels moet voldoen, de programmatekst moet zijn geschreven in een taal, waarvan de taalregels heel nauwkeurig gedefinieerd zijn. De automaat van het Rekencentrum van de THE accepteert programma's geschreven in ALGOL 60 (waar ALGOL een afkorting is van ALGOrithmic Language), een internationaal erkende programmeertaal. In dit college zal een gedeelte van deze taal behandeld worden, zoals hij hier bij het Rekencentrum in gebruik is.

Een eerste programma.

Zoals gezegd heeft een rekenmachine

1) een invoerorgaan voor de invoer van programma en gegevens
2) een geheugen voor het onthouden van het programma en de tussenresultaten
3) een rekenorgaan voor het uitvoeren van de bewerkingen
4) een uitvoerorgaan voor het afleveren van de antwoorden.

Wij zullen in ons eerste programma al deze vier onderdelen gebruiken. Wij gaan een programma behandelen, dat twee getallen van een band leest en op de regeldrukker op een regel achter elkaar vier getallen afdrukt, nl. de twee gelezen getallen, hun som en hun verschil. Wij beperken ons hier tot gehele getallen. Dit programma kan alsvolgt luiden:

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

Om te beginnen zien we hier een paar onderstreepte woorden ("begin", "integer" en "end"). Zo'n onderstreept woord moet beschouwd worden als een enkel symbool, het fungeert als een soort van leesteken. Bij het ontwerp van ALGOL 60 werd alras duidelijk, dat -zoals in de meeste wiskundige teksten- er behoefte zou zijn aan meer symbolen dan normaliter op een schrijfmachine aanwezig zijn. Hiervoor is een algemene oplossing gevonden, nl. een onder-

streept woord; het complement van deze conventie is, dat in ALGOL-teksten onderstreping voor geen enkel ander doel gebruikt wordt. Op deze wijze zijn in ALGOL 60 ruim 20 speciale symbolen ingevoerd. (Onderstreping is op een schrijfmachine en in manuscript heel makkelijk te doen, voor een zetter is het echter heel moeilijk: als ALGOL-programma's in druk verschijnen, worden deze woorden daarom niet onderstreept maar vet gedrukt.) De onderstreping is dus niets diepzinnigs, het is louter een notatieafspraak hoe een aantal speciale symbolen er op papier zullen uitzien.

Een programma begint altijd met het symbool "begin" en eindigt altijd met het symbool "end". Zij fungeren als haakjespaar, "begin" is de openingshaak en "end" is de sluitingshaak. Zij worden hier gebruikt om de omvang van de programmatekst aan te geven. (In normale conversatie hebben we geen teken om aan te geven, dat we uitgesproken zijn: je gesprekspartner neemt dat aan op vage indicaties zoals een tijdje stilte van jouw kant!) Ander gebruik van het hakenpaar "begin" - "end" zullen we straks tegenkomen.

De rest van de eerste regel -hij heet "de declaratie van de variabelen genaamd "a1", "a2", "som" en "verschil"- beschrijft het gebruik dat van het geheugen gemaakt zal worden voor het onthouden van tussenresultaten. Hij behelst, dat in dit rekenproces vier zg. "variabelen" voorkomen, dwz. vier geheugenelementen analoog aan het geheugenelement van de roltrap. Dat de declaratie ingeleid is met het symbool "integer" behelst, dat elk der vier geintroduceerde variabelen op elk moment een geheel getal kan onthouden.

Bij de roltrap, die maar een enkel geheugenelement had, kon dit onbenoemd blijven, zoals je in een monogame maatschappij er mee volstaan kunt, je echtgenote als "Vrouw" toe te spreken. Heb je een harem, dan is dat wat onduidelijk en moet je kunnen aangeven welke van je vele vrouwen je toespreekt en daartoe moeten de vrouwen benoemd zijn. In een rekenproces, waarin als regel vele variabelen voorkomen, moeten de variabelen om dezelfde reden benoemd zijn en dat is nu net, wat de declaratie voor ons uitdrukt. De eerste regel drukt niet alleen uit, dat er in dit programma over vier (geheeltallige) variabelen gesproken wordt, hij geeft ook de namen op, onder welke in de programmatekst naar deze variabelen verwezen zal worden.

In een programma hebben namen -de technische term voor hen luidt "identifiers"- geen intrinsieke betekenis. "Identifiers have no inherent meaning, they serve for identifying purposes only.". Het is als in het normale leven, waarin meneer de Groot best een klein mannetje zou kunnen zijn. De programmeur is (in hoge mate, zie onder) vrij in de keuze van de namen, die hij zelf invoert en hij doet er goed aan, om sprekende namen in te voeren en verwarrende namen te vermijden.

(Ter onderstreping van het feit, dat namen geen intrinsieke betekenis hebben, geven we een andere programmatekst, die dezelfde opgave als boven oplost:

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

Dit programma doet precies hetzelfde als het voorafgaande; het enige verschil is dat de som de misleidende naam "product" en het verschil de misleidende naam "som" gekregen heeft. Aangezien de machine wel de regels van ALGOL 60, maar niet die van het Nederlands zijn bijgebracht, wordt dit programma even vrolijk uitgevoerd. Desalniettemin wordt het introduceren van een dergelijke (voor ons!) misleidende nomenclatuur natuurlijk ten zeerste afgeraden.)

Opm.1 Men kan ALGOL 60 beschouwen als een taal, waarvan de syntaxis weliswaar heel streng bepaald is, maar waarvan het vocabulaire min of meer open is; de declaratie fungeert dan als "woordenlijstje" voor het bijbehorende programma.

Opm.2 Naast de door de programmeur voor variabelen ingevoerde namen komen er ook een paar ongedeclareerde namen voor functies (zoals "read") en handelingen (zoals "print") in dit programma voor. Hun betekenis is door conventie bepaald, zij verwijzen naar een arsenaal -de zg. bibliotheek-, dat zonder meer bekend verondersteld mag worden.

In het voorafgaande hebben wij gezegd, dat de programmeur een grote vrijheid heeft in de keuze van namen. Helemaal vrij is hij echter niet. Als naast de namen "a1" en "a2" bijvoorbeeld ook de symbolenreeks "a1 + a2" als naam van een derde variabele zou mogen voorkomen, dan zou niet meer duidelijk zijn of met "a1 + a2" de som van de eerste twee variabelen, dan wel de waarde van de derde variabele bedoeld zou zijn!

(Beperkingen aan de toelaatbaarheid van namen, opdat misverstand vermeden worde, zijn bekend. Toen Odysseus door de Cycloop gevangen was zei hij, dat hij "Niemand" heette. Toen Odysseus de Cycloop het enige oog had uitgestoken en de Cycloop, verblind en brullend van de pijn de andere cyclopen vroeg om de boosdoener te vangen en zij hem vroegen, wie dan wel zijn oog had uitgestoken, antwoordde de verminkte "Niemand heeft het gedaan." Zodat zij schouderophalend huns weegs weer gingen en Odysseus een kans kreeg om te ontsnappen.)

In ALGOL 60 is als naam toegelaten elke rij opgebrouwd uit letters en cijfers, mits niet beginnend met een cijfer. (Eventuele spaties en overgang op een nieuwe regel hebben, als overal in een ALGOL 60 programma, geen betekenis.) Toelaatbaar als naam is "seven up", ontoelaatbaar is "7 up"; toelaatbaar is ook "austin seven" evenals "austin 7", maar de laatste twee namen hebben niets met elkaar te maken: ze zijn verschillend omdat het verschillende symbolenreeksen zijn.

Tot zover over namen. Na de declaratie komen in de text acht zg. "statements". De statements beschrijven de eigenlijke handelingen, die verricht moeten worden. Het is een regel in ALGOL 60, dat op elkaar volgende declaraties en/of statements onderling door een puntkomma gescheiden worden. Deze regel, die enige analogie vertoont met de normale interpunctieregel zinnen met een punt af te sluiten, is ingevoerd om de text eenduidig interpreteerbaar te maken: zonder de statement scheider, nl. de puntkomma, was het bv. niet duidelijk waar de eerste statement zou eindigen en waar de tweede zou beginnen, was het niet duidelijk, dat de letterrij "read a2" gesplitst moet worden in de naam "read" gevolgd door de naam "a2".

Zolang niet anders is aangeduid (zie later) worden statements uitgevoerd in de volgorde, waarin ze in de text voorkomen. De eerste statement

"a1:= read"

(lees "a1 wordt read") behelst, dat op grond van de functie read onder doorschuiven van de band het eerste getal van de band gelezen wordt en dat de gelezen waarde aan de variabele a1 wordt toegekend. Precieser: de waarde van de (in de bibliotheek bekend veronderstelde) functie genaamd "read" is gelijk aan de volgende getalwaarde op de invoerband; tot nader order zal de variabele genaamd "a1" na uitvoering van de eerste statement de zojuist gelezen waarde hebben. (Tot nader order betekent in dit programma: voor de verdere duur van dit programma). Het "wordt-symbool", weergegeven door ":=" heet de toekenningsoperator of "the assignment operator".

Ten gevolge van de eerste statement is het eerste getal van de invoerband gelezen en heeft de variabele a1 (waarvan de waarde voordien ongedefinieerd was) het gelezen getal als waarde gekregen. De volgende statement

"a2:= read"

behelst, dat weer een getal van de band gelezen wordt, maar dat de nu gelezen waarde aan de variabele a2 wordt toegekend. Na de eerste twee statements zijn beide getallen van de invoerband gelezen en toegekend aan de variabelen a1 en a2; onder gebruikmaking van (de namen van) die variabelen kunnen we nu het programma zo vaak als nodig is naar deze twee getallen laten verwijzen. In dit programma gebeurt dit voor beide twee keer, nl. in de volgende statement, waarin de som van de getallen berekend wordt en als waarde wordt gegeven aan de variabele genaamd "som" en in de daarop volgende statement, waarin op analoge wijze het verschil wordt berekend en toegekend.

Na de twee statements "invoer", gevolgd door de twee statements "berekening" volgen in de derde regel vier statements "uitvoer". Hier wordt een beroep gedaan op de (in de bibliotheek bekend veronderstelde) handeling genaamd "print", een handeling met één argument, die zoals de naam suggereert, de meegegeven waarde op papier doet verschijnen. Ten gevolge van dit programma zou bv. gedrukt kunnen worden:

+ 456678 + 243179 + 699857 + 213499 .

Dit programma is nog niet representatief voor het gebruik van variabelen. Na de declaratie, die de variabele heeft geintroduceerd, is zijn waarde ongedefinieerd tot aan de uitvoering van de eerste assignment statement, die een waarde aan hem toekent. We hebben gezegd, dat de variabele dan deze waarde blijft behouden "tot nader order": een variabele verliest nl. zijn oude waarde wanneer er door een assignment statement hernieuwd een waarde aan wordt toegekend. Dit wordt geillustreerd in het volgende programma, dat precies hetzelfde effect heeft als het vorige. Let er op, dat in dit programma maar drie variabelen voorkomen.

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

Voorwaardelijke uitvoering van statements.

Wij hebben eerder gezegd, dat de statements worden uitgevoerd in de volgorde, waarin zij in de text voorkomen, tenzij anders is aangegeven. Wij zullen nu eenvoorbeeld geven, waarin inderdaad "anders is aangegeven". Stel, dat wij een programma moeten schrijven, dat weer twee getallen van de band moet lezen en deze om te beginnen moet printen, maar daarachter de som, wanneer het eerste getal kleiner is dan het tweede, edoch het verschil, wanneer zulks niet het geval is. Bij een opgave als deze weten wij niet van te voren, welke handelingen door de machine moeten worden uitgevoerd, omdat dit mede bepaald is door de getallen die van de band gelezen zullen worden. Wij moeten in het programma op een of andere manier kunnen aangeven, hoe de machine de uit te voeren handeling kiest. Dit kan in ALGOL 60 als volgt:

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

Leeszaal
Centrale Bibliotheek
T.H. Eindhoven

Mogelijke uitvoer van dit programma is

+ 243179	+ 456678	+ 699857	of
+ 456678	+ 243179	+ 213499	.

De structuur van de derde regel van dit programma is

"if voorwaarde then statement 1 else statement 2" ;

de betekenis hiervan is, dat eerst onderzocht wordt, of aan de voorwaarde voldaan is, zo ja dan wordt daarna statement 1 uitgevoerd en statement 2 overgeslagen, zo nee dan wordt statement 1 overgeslagen en statement 2 uitgevoerd. De if - then - else constructie geeft ons de mogelijkheid om in het programma de keuze tussen twee alternatieven voor te schrijven. De hele constructie fungeert weer als statement en wordt derhalve van voorganger en opvolger in de text door een puntkomma gescheiden.

Om aan de variabele genaamd "y" de absolute waarde van de variabele genaamd "x" toe te kennen, kan men in een programma dus schrijven de statement:

"if $x < 0$ then y:= - x else y:= x" .

Stel nu, dat gevraagd wordt om de waarde van de variabele genaamd "x" te vervangen door zijn absolute waarde. Naar analogie zou men kunnen schrijven:

"if $x < 0$ then x:= - x else x:= x" .

Dit is opzichzelf goed in de zin, dat deze statement het gewenste effect bewerkstelligt, maar het staat een beetje onnozel. Het is ook onpractisch, want de assignment statement "x:= x", die aan de variabele "x" de waarde toekent, die hij al heeft, heeft geen enkel effect, maar kost wel rekentijd! Wat je zou willen uitdrukken is "als x negatief is, wissel dan x van teken, anders hoef je niets te doen". Dit is een zoveel voorkomende situatie, dat het in ALGOL 60 toegestaan is om "else statement 2" weg te laten. Men krijgt dan de constructie

"if voorwaarde then statement 1"

die behelst, dat onderzocht wordt, of aan de voorwaarde voldaan is: zo ja,

dan wordt statement 1 wel uitgevoerd, zonee, dan wordt statement 1 overgeslagen. Weer fungeert de hele constructie als statement en wordt hij in de text van voorganger en opvolger door een puntkomma gescheiden. Om de waarde van de variabele genaamd "x" door zijn absolute waarde te vervangen schrijft men dus in zijn programma:

"if $x < 0$ then x:= - x" .

(Vraag. Waarom is bovengegeven statement te verkiezen boven

"if $x \leq 0$ then x:= - x",

hoewel deze ook goed is in de zin, dat hij het gewenste effect bewerkstelligt?)

Bij de if - then en bij de if - then - else constructie hebben we dus de mogelijkheid om aan te geven, dat conditioneel (dwz. afhankelijk van het al dan niet vervuld zijn van een voorwaarde) een statement uitgevoerd, dan wel overgeslagen moet worden. Onmiddellijk rijst de vraag, hoe we hiervan gebruik kunnen maken, wanneer de handeling, die eventueel overgeslagen zou moeten worden, zich in eerste instantie niet door een enkele statement, maar door een rijtje statements laat beschrijven.

Beschouw bv. de opgave "wissel x en y van teken als x negatief is". De variabelen x en y inconditioneel (dwz. onvoorwaardelijk) van teken wisselen, laat zich beschrijven door twee statements achter elkaar:

"x:= - x; y:= - y"

maar we mogen voor de conditionele tekenwisseling, zoals gevraagd, niet schrijven:

"if $x < 0$ then x:= - x; y:= - y"

want dit zou betekenen, dat y altijd van teken wordt gewisseld! We moeten op een of andere manier aangeven dat het duo

"x:= - x; y:= - y"

in zoverre een onscheidbaar geheel vormt, dat ze tesamen wel, of tesamen niet worden uitgevoerd. Het aangeven van een dergelijke groepering is de tweede (en belangrijkste) van het reeds eerder vermelde paar statement-

haken "begin" en "end". Door een rijtje statements door dit hakenpaar te omvatten, zorgt men, dat dit rijtje in de omgeving, waarin het staat, fungeert als een enkele statement. Binnen het hakenpaar worden de "interne" statements als gebruikelijk weer door een puntkomma onderling gescheiden. Wij schrijven voor de conditionele tekenwisseling van x en y daarom:

"if $x < 0$ then begin x:= - x; y:= - y end" .

Het is nu duidelijk, waarom het symbolenpaar "begin" en "end" een hakenpaar genoemd wordt. Vergelijk nl. de algebraische uitdrukkingen

"a * b" en "a *(b + c)" ;

in het tweede product, waarin de laatste factor niet een enkele variabele, maar weer een samengestelde uitdrukking is, gebruikt men haakjes om aan te geven over welk stuk van de formule de beschrijving van de laatste factor zich uitstrekt.

Opm. Veel dubbelzinnigheid in het normale Nederlands is te herleiden tot de afwezigheid van haken. In een chocolaterie in de Jordaan heb ik een voordelige aanbieding chocoladerepen aangeprezen gezien met

"slechts drie voor een kwartje" .

De bedoeling was

"slechts (drie voor een kwartje)" ,

maar de zin is helaas ook scandeerbaar als

"(slechts drie) voor een kwartje",

waarmee het als onvoordelige aanbieding functioneert!

In een recent jaarboek van de Leidse Universiteit staat bij een foto het onderschrift:

"Jean Michelhofje aan de Pieterkerkstraat 10 voor bewoning door studenten herbouwd en gerestaureerd."

Gaat dit over studentenhuisvesting of over een studentenwerkkamp?

Herhaalde uitvoering van statements.

Met de if - then - constructie hebben we de mogelijkheid om uit te drukken, dat iets eventueel gebeuren moet, met de if - then - else - constructie hebben we de mogelijkheid, dat óf het een, óf het ander gebeuren moet, we hebben nog niet de mogelijkheid om uit te drukken, dat iets onder een bepaalde voorwaarde herhaald moet worden uitgevoerd.

Laat ons aannemen, dat er twee positieve getallen van de band gelezen moeten worden, zeg a en d, en dat de rest bepaald moet worden die overblijft bij deling van a door d. Om der wille van het voorbeeld leggen wij ons de beperking op, dat we niet van vermenigvuldiging en deling gebruik mogen maken en dat daarom de rest bepaald moet worden door d "zovaak als het gaat" van a af te trekken. Met de term "zovaak als" is al aangegeven, dat het hier om een herhaling van iets gaat. Wij geven eerst het programma, wij zullen het daarna bespreken.

```
begin integer a, d, r;
      a:= read; d:= read;
      r:= a;
      while r ≥ d do r := r - d;
      print(a); print(d); print(r)
end
```

In de eerste regel van dit programma worden drie integers gedeclareerd, a (voor het deeltal), d (voor de deler) en r (voor de rest). In de volgende regel worden twee getallen van de band gelezen, in volgorde het deeltal en de deler. In de derde regel wordt aan r (de "rest in wording") de waarde van het deeltal toegekend: het is dit getal, dat q maal (als q het ons onbekende quotient is) met de deler d verlaagd moet worden.

In de vierde regel komen we een nieuwe vorm van statement tegen, de zogenaamde while statement. Hierin spelen de nieuwe symbolen while en do een rol. Deze constructie heeft als effect dat de statement volgend op do herhaald zal worden uitgevoerd zolang aan de voorwaarde tussen while en do is voldaan.

In dit voorbeeld luidt die voorwaarde "$r \geq d$". Na afloop van de while statement zal dus het tegendeel gelden, en wel: "$r < d$". Verder is het makkelijk in te zien dat de aanvankelijk geldende relatie $r \geq 0$ door de statement niet wordt aangetast, want r wordt alleen dan met d verminderd als zojuist is vastgesteld dat $r \geq d$ was.

De uiteindelijke waarde van r ontstaat door de waarde van a nul of meer (zeg k) keren met d te verminderen. We hebben nu gevonden een getal r ($0 \leq r < d$) waarvoor geldt: "$r + k * d = a$", en dit is juist de definitie van de rest bij deling van a door d.

Naast de in dit voorbeeld geïntroduceerde while statement kennen we nog een tweede vorm van herhalingsconstructie waarbij de te herhalen statement ten minste één keer zal worden uitgevoerd. Deze heeft de gedaante

do "statement" until "voorwaarde" .

Zodra na afloop van de statement aan de voorwaarde is voldaan, wordt verdere herhaling gestaakt.

De do-until statement is minder vaak toepasbaar en ook vaak minder algemeen dan de while statement, maar beslist essentieel in die gevallen waarin de "voorwaarde" slechts is gedefinieerd als de statement ten minste één keer is uitgevoerd. (zie EWD 242 - 29). In het onderhavige voorbeeld hadden we de vierde regel kunnen schrijven als: do $r := r - d$ until $r < d$
uitsluitend in de veronderstelling dat altijd $a \geq d$ zal gelden. Ga dit zelf na.

NB. Met het element van herhaling hebben we de mogelijkheid geintroduceerd van quasi-algorithmen, dwz. iets dat er als algorithme uitziet, maar bij nader inzien een nimmer eindigende taak beschrijft. Ga na, wat er gebeurt, wanneer a positief, maar d abusievelijk negatief is!

Tegen ons programma, dat de rest bepaalt, kan het bezwaar worden aangevoerd, dat het, hoewel in theorie goed, in de praktijk wel eens erg tijdrovend

zou kunnen zijn, nl. als a vele malen groter is dan d. In een volgende versie zullen we aan dit bezwaar tegemoet komen; wel nemen we daarbij aan, dat verdubbelen en halveren tot het arsenaal toegestane operaties behoren. (Noot: verdubbelen en halveren is in een binaire machine de tweetallige komma een plaats verschuiven en dat is wel iets heel anders dan het product of het quotient van twee willekeurige getallen kunnen bepalen.)

Ons versnellingsplan berust daarop, dat we om in grotere stappen te kunnen werken, eerst de deler zovaak verdubbelen, totdat hij groter is dan het deeltal, vervolgens al deze verdubbelingen door halveringen weer te niet doen, maar en passant r (de "rest in wording")steeds, indien mogelijk, met het heersende veelvoud van de deler te verminderen. We introduceren een nieuwe variabele "dd" (voor "dubbele deler").

```
begin integer a, d, r, dd;
      a:= read; d:= read;
      r:= a; dd:= d;
      while r ≥ dd do dd := 2 * dd;
      while dd ≠ d do
      begin dd := dd / 2;
            if r ≥ dd then r := r - dd
      end;
      print(a); print(d); print(r)
end
```

In de eerste herhalingscyclus wordt dd (die begint met de waarde van de deler d) verdubbeld, zolang "r ≥ dd" geldt; na afloop van de eerste herhalingscyclus geldt dat dus niet meer en geldt dus

"r < dd" .

In de tweede herhalingscyclus wordt tussen begin en end eerst dd gehalveerd. Geldt aanvankelijk

"r < dd" (1)

na halvering van dd geldt dus

"r < 2 * dd" . (2)

In de volgende regel geldt óf $r < dd$ óf $r \geq dd$; in het laatste geval wordt r met dd verlaagd, zodat op grond van (2) weer (1), dwz. $r < dd$ geldt. Dit stuk tussen begin en end handhaaft dus de geldigheid van relatie (1); het stuk wordt herhaald zolang $dd \neq d$ is; na afloop van de tweede herhalingscyclus moet dus $dd = d$ zijn. Uit (1) volgt daarmee

"$r < d$".

Aangezien r alleen met veelvouden van d is afgelaagd en deze aflagingen met dd alleen maar plaatsvinden onder de voorwaarde ("$r \geq dd$" nl.), dat hierdoor geen negatieve waarde van r ontstaat, volgt dat eveneens aan de relatie

"$0 \leq r$"

voldaan is en blijft.

Opgave. Voeg aan een van deze beide programma's een declaratie voor de variabele q en de statements toe, die aan q de waarde geven van het quotient, dat deling van a door d oplevert. Voor het eerste programma is dit het makkelijkst, voor het tweede is het het leukst!

Arrays.

Een van de manieren, waarop men in de analytische meetkunde de coordinaten kan benoemen is met "x", "y" en "z". Een andere manier om dit te doen is met "x_1", "x_2" en "x_3". Het voordeel van de laatste notatie is, dat men dan ook een naam heeft voor de algemene coordinaat, nl. "x_i". Het is duidelijk, dat deze naam alleen maar een van de drie coordinaten aanduidt, mits $i = 1$, $i = 2$ of $i = 3$.

Een analoge mogelijkheid bestaat in ALGOL 60. Om te beginnen is er een notatietechnische kwestie: een ALGOL-text bestaat uit een opeenvolging van karakters, waarin het onaantrekkelijk is om een index, een "subscript" als zodanig aan te geven door hem een beetje lager te schrijven: we willen er geen betekenis aan toekennen, wanneer karakters een beetje hobbelig op de lijn lijken te staan. In plaats van "x_1", "x_2", "x_3", en algemeen "x_i" schrijven we daarom

"x[1]", "x[2]", "x[3]" en algemeen "x[i]" en nog algemener "x[i + 1]"

De vierkante haken heten daarom "subscriptiehaken", individueel heten ze in de wandeling "sub" (= "vierkant open") en "bus" (= "vierkant dicht").

Een dergelijke collectie variabelen heet "een array". In de declaratie van een array moet men aangeven de identifier van het array, het aantal subscripts en per subscript onder- en bovengrens van de subscript. (In dit college zal ik me beperken tot zg. een-dimensionale arrays, dwz. arrays met één subscript, het analogon van vectoren. Er zijn in ALGOL 60 ook meer-dimensionale arrays toegestaan: het twee-dimensionale array, waarin elk element door twee subscripts geidentificeerd wordt, is het analogon van de matrix.) In de declaratie vermeldt men tussen de vierkante haken eerst de ondergrens, dan de bovengrens, onderling gescheiden door een dubbele punt.

In het volgende voorbeeld zullen wij een programma schrijven, dat 100 gehele getallen van de band leest en in omgekeerde volgorde print. In de simpelste versie luidt dit programma bv. als volgt:

```
begin integer i; integer array a[0:99];
      i:= 0;
      while i < 100 do
        begin a[i] := read; i := i + 1 end;
      i := 100;
      while i > 0 do
        begin i := i - 1; print(a[i]) end
end
```

In de eerste regel hebben we behalve de integer i de honderd integers a[0] t/m a[99] gedeclareerd. In de lees-cyclus is i = "het aantal gelezen getallen" en als we afspreken, dat we het array in volgorde van opklimmende subscriptwaarde zullen vullen, tevens = "de subscriptwaarde van het array-element, dat aan de beurt is om de volgende waarde te krijgen". In de print-cyclus is i = "het aantal getallen, dat nog geprint moet worden".

We gaan dit programma nu op twee manieren wijzigen: we willen dat er bij het uitprinten slechts 7 getallen op een regel komen. (Omdat we 100 getallen hebben, geeft dat 14 volle regels plus een 15-de regel met nog 2 getallen.)

We mogen voor de overgang op een nieuwe regel gebruik maken van de statement "nlcr" (dwz. "new line carriage return"), welke handeling in de bibliotheek onder die naam bekend verondersteld wordt. Bovendien willen we op de 16-de regel maximum- en minimumwaarde van de gelezen getallen printen.

We voeren drie extra variabelen in:

j: geeft aan, hoeveel getallen er op de onderhavige regel nog geprint kunnen worden;

max: de (tot nog toe gevonden) grootste waarde;

min: de (tot nog toe gevonden) kleinste waarde.

We geven nu eerst het programma:

```
begin integer i, j, max, min; integer array a[0 : 99];
      i := 0;
      while i < 100 do
        begin a[i] := read; i := i + 1end;
      i := 100; j := 7;
      while i > 0 do
        begin i := i - 1; print(a[i]);
              j := j - 1; if j = 0 then begin nlcr; j := 7 end
        end;
      max := a[0]; i := 1;
      while i < 100 do
        begin if a[i] > max then max := a[i]; i := i + 1 end;
      min := a[0]; i := 1;
      while i < 100 do
        begin if a[i] < min then min := a[i]; i := i + 1 end;
      nlcr; print(max); print(min)
end
```

Zoals gezegd zou "j" de betekenis hebben van het aantal getallen, dat nog op de onderhavige regel geprint kan worden. Aanvankelijk is j = 7; elke keer, dat een getal gerpint wordt, wordt j daarom met 1 afgelaagd; komt daarbij j op nul, dan is de onderhavige regel dus vol, wordt dus de overgang op een nieuwe regel ingelast en wordt j, in overeenstemming met zijn betekenis weer op 7 gezet. (We hebben er in dit programma duidelijk gebruik van gemaakt, dat als we niets over nieuwe regels voorschrijven, het eerste getal, dat door het programma geprint wordt, op het begin van een (nieuwe) regel geprint wordt.)

Na het printen van de 100 getallen wordt het maximum gezocht. Steeds is max = "het maximum van de eerste i getallen". Zolang $i < 100$ is, zijn nog niet alle getallen onderzocht en moet het volgende getal onderzocht worden om te kijken of dat soms groter is dan max, het tot nog toe gevonden maximum; zo ja, dan moet de waarde van max worden aangepast. De volgende cyclus zoekt op analoge wijze het minimum en tenslotte worden de laatste twee uitkomsten op een nieuwe regel geprint.

Als we de getallen niet in omgekeerde volgorde hadden hoeven te printen en alleen maar gevraagd was om maximum en minimum te bepalen, dan hadden we het hele array niet nodig gehad. In plaats daarvan hadden we kunnen volstaan met een enkele variabele (die we gemakshalve maar weer "a" noemen) voor de waarde van het laatstelijk van de band gelezen getal.

```
begin integer i, max, min, a;
      a:= read; i:= 1; max:= a; min:= a;
      while i ≤ 100 do
        begin a := read; i := i + 1;
              if a ≥ max then max := a
                 else if a < min then min := a
        end;
      print(max); print(min)
end
```

Hier is:

a = het laatst gelezen getal van de band;

i = het aantal van de band gelezen getallen;

max = het tot nog toe grootste gevonden getal;
min = het tot nog toe kleinste gevonden getal.

Logische expressies, variabelen en operatoren.

Tussen "if" en "then" zijn wij tegengekomen wat wij "voorwaarden" genoemd hebben. Voorbeelden van dergelijke voorwaarden waren: "$x \leq 0$", "$i < 100$", "$i > 0$", "$j = 0$" etc, en we hebben gesteld, dat onderzocht kan worden of aan een dergelijke voorwaarde voldaan is. In deze sectie gaan we dit onderzoek beschrijven als "berekening", maar onmiddellijk rijst de vraag: "Berekening van wat?".

Als x en y gegeven numerieke waarden zijn, dan weten we wat met "x + y" bedoeld is, nl. weer een numerieke waarde en wel de som. Omdat de som weer een numerieke waarde is heet "+" een zg. "arithmetische operator"; andere voorbeelden van arithmetische operatoren zijn "-", "*" en "/". Arithmetische operatoren opereren op numerieke operanden en leveren een numeriek resultaat af.

Maar er zijn andere operatoren, die wel op numerieke operanden opereren maar niet een numeriek resultaat afleveren. Dit zijn de zg. "relational operators":

$x = y$ "x gelijk y"
$x \neq y$ "x ongelijk y"
$x > y$ "x groter dan y"
$x \geq y$ "x tenminste y"
$x < y$ "x kleiner dan y"
$x \leq y$ "x tenhoogste y".

Deze relational operators verbinden twee numerieke operanden en beschrijven daarmee een voorwaarde, waaraan al of niet voldaan kan zijn. We kunnen ze ook interpreteren als bewering, die al of niet waar is. Het onderzoek of de relatie "x = y" geldt kunnen we ook beschrijven als het vaststellen van de waarheid, of nog pregnanter het berekenen van de waarheidswaarde (truth value) van de bewering "x = y". Hiermee zijn de relational operators geworden, tot

operatoren, die als domein hebben de (geordende) getallenparen en als bereik de mogelijke waarheidswaarden. Dit laatste bereik bevat slechts twee verschillende waarden, die we als "true" resp. "false" noteren.

In plaats van te zeggen, dat aan de voorwaarde "x = x" voldaan is, zeggen we dat de logische expressie "x = x" de waarde true heeft; in plaats van te zeggen, dat aan de voorwaarde "x = x + 1" niet voldaan is, zeggen we, dat de logische expressie "x = x + 1" de waarde false heeft.

We kenden arithmetische expressies (nl. arithmetische variabelen, getallen of arithmetische expressies verbonden door arithmetische operatoren), hiernaast onderkennen we nu de zg. logische expressies (voorshands arithmetische expressies verbonden door een relational operator of een waarheidswaarde). Wat de getallen zijn voor de arithmetische expressies zijn de logische waarden "true" en "false" voor de logische expressies, nl. een opsomming van het waardebereik.

Opm. De betekenis van de if - then - else - constructie kunnen we nu alsvolgt expliceren:
"if true then S1 else S2" is equivalent met "S1" en
"if false then S1 else S2" is equivalent met "S2".

De analogie tussen arithmetische en logische expressies gaat verder. Elke waarde van een arithmetische expressie kan aan een arithmetische variabele worden toegekend. Onder geldigheid van de declaratie

"integer x, y, som, verschil"

mogen (nadat x en y een waarde hebben gekregen) de assignment statements

"som:= x + y; verschil:= x - y"

voorkomen. Geheel analoog mag onder geldigheid van de declaraties

"integer x, y; boolean xgroter, xkleiner"

nadat x en y een waarde hebben gekregen de assignment statements

"xgroter:= x > y; xkleiner:= x < y"

voorkomen.

Zoals de assignment statement "som:= x + y" in de variabele genaamd "som" de som van de nu heersende waarden van x en y fixeert (waarna x en/of y van waarde mogen veranderen), zo legt de assignment statement "xgroter:= x > y" vast of nu de heersende waarde van x groter is dan die van y (waarna x en/of y van waarde zouden mogen veranderen).

In de declaratie zijn we het nieuwe symbool "boolean" (zo genoemd ter ere van George Boole) tegengekomen. De declaratie van variabelen heeft nl. een dubbele functie: als we een variabele declareren wordt niet alleen de hem toegevoegde identifier gedefinieerd, maar met behulp van de zg. declarator ("integer" of "boolean") wordt ook het bijbehorende waardebereik van de variabele vastgelegd.

Naast de relational operators -die arithmetische operanden versmelten tot een logisch resultaat- kent ALGOL 60 bovendien zg. logische operatoren, die op logische operanden opererend een logisch resultaat opleveren. Wij noemen er vier en geven hun definitie in tabellarische vorm.

1. De equivalentie.

true	≡ true	is true
true	≡ false	is false
false	≡ true	is false
false	≡ false	is true

Wat het gelijkteken is voor arithmetische operanden is derhalve het equivalentieteken voor logische operanden. De dubbele voorwaarde "0 < x < 1" kan in ALGOL 60 geschreven worden met behulp van de equivalentie:

$$0 < x \equiv x < 1 \quad .$$

Immers:

ligt x links van het interval, dan is 0 < x false en x < 1 true, zodat de hele expressie false is;

ligt x in het interval, dan zijn beide relaties true, zodat de hele expressie true is;

ligt x rechts van het interval, dan is 0 < x true en x < 1 false, zodat de hele expressie weer false is; (voor een natuurlijkere formulering, zie de

conjunctie).

2. De negatie.

non true is false
non false is true

Zo gelden:

(non $x = y$) $\equiv x \neq y$ en
(non $x < y$) $\equiv x \geq y$ en
(non $x > y$) $\equiv x \leq y$.

3. De conjunctie.

true and true is true
true and false is false
false and true is false
false and false is false

desda
De conjunctie is dan en slechts dan true als beide operanden true zijn. Een andere (en naar mijn smaak natuurlijkere) manier om de dubbele voorwaarde

"$0 < x < 1$"

in ALGOL 60 te schrijven is

"$0 < x$ and $x < 1$".

4. De disjunctie.

true or true is true
true or false is true
false or true is true
false or false is false

De disjunctie is dan en slechts dan true als tenminste een van beide operanden true is. (Hij is dus ook true als beide operanden true zijn; dit in tegenstelling tot de manier, waarop in het Nederlands het woordje "of" gebruikt wordt als in "Wil je een appel of een peer?") Let op de equivalentie:

non (B1 and B2) = (non B1) or (non B2),

in woorden: "Als niet aan beide van twee voorwaarden voldaan is, is aan tenminste een van beide voorwaarden niet voldaan en omgekeerd.".

Verdere voorbeelden van relaties, die gelden, zijn:

$x = y \equiv (x \leq y \text{ and } x \geq y)$ en $x \neq y \equiv (x < y \text{ or } x > y)$.

We beschouwen nu de volgende opgave. Op een band staan voor $0 \leq n \leq 100$ in volgorde $2n+1$ getallen, nl.

$$n, x_1, \ldots, x_n, y_1, \ldots, y_n \quad .$$

(Als $n = 0$, dan bevat de band alleen het getal n; als $n = 1$, dan bevat de band n, x_1, y_1 etc.) Deze band moet worden gelezen en ons programma moet vaststellen of het rijtje x-en elementsgewijs aan het rijtje y's gelijk is. (Hierbij gelden "lege rijtjes", dwz. $n = 0$, als gelijk: ze bevatten nl. geen overeenkomstige elementen, die verschillen!) Zo ja, dan moet het programma "gelijk" printen, zo nee, dan moet het "ongelijk" printen. We geven eerst het programma.

```
begin integer i, n; integer array x, y [1:100]; boolean equal;
      n:= read;
      i:= 0;
      while i < n do
        begin i := i + 1; x[i] := read end;
      i := 0;
      while i < n do
        begin i := i + 1; y[i] := read end;
      i := 0; equal := true;
      while i <n and equal do
        begin i := i + 1; equal := (x[i] = y[i]) end;
      if equal then printtext(≠ gelijk ≠) else printtext(≠ ongelijk ≠)
end
```

In de eerste regel worden (o.a.) twee arrays "x" en "y" gedeclareerd, beide ter lengte 100 (dat is in elk geval lang genoeg). In de eerste cyclus wordt het rijtje x-en elementsgewijs gelezen en in volgorde in het array x opgeslagen; bij de test heeft "i" de betekenis "het aantal inmiddels gelezen

x-en". De volgende cyclus leest op analoge wijze de y's van de band. In de test van de laatste cyclus heeft "i" de betekenis "het aantal inmiddels vergeleken x-y-paren" en "equal" heeft de betekenis "bij de inmiddels vergeleken x-y-paren is geen verschil geconstateerd". De voorwaarde

"i < n and equal"

drukt uit dat om door te gaan met vergelijken van het volgende paar er aan twee voorwaarden voldaan moet zijn: de rijtjes moeten nog niet zijn uitgeput en bovendien moet nog geen verschil geconstateerd zijn. Zodra nl. wel verschil geconstateerd is, is de uitslag bekend en kan de elementsgewijze vergelijking gestaakt worden. In de laatste regel wordt een beroep gedaan op de wederom in de bibliotheek bekend veronderstelde handeling genaamd "printtext", die zijn (ene) argument zoals gebruikelijk meekrijgt tussen ronde haken. Dit argument is voor printtext een zg. "string", de karakters "⊲" en "⊳" heten "string open" en "string close" en worden gebruikt om de lengte van de string aan te geven; zij verschijnen zelf niet op papier, de karakters ertussen wel.

De opbouw van een groter programma.

Als laatste voorbeeld behandelen we de opgave om de eerste 1000 priemgetallen (te beginnen bij 2) uit te rekenen en te printen. In grofste structuur bestaat dit programma uit:

```
begin integer array p[1:1000];
      "vul het array p met de opklimmende priemgetallen";
      "print het array p elementsgewijs uit"
end
```

en vervolgens gaan we deze twee tussen aanhalinstekens benoemde handelingen nader uitwerken. Voor de eerste bedenken we, dat na het eerste priemgetal (=2) alle volgende priemgetallen oneven zijn; het zal blijken, dat we dit gegeven goed kunnen gebruiken en in de eerste uitwerking van "vul het array p met de opklimmende priemgetallen" zullen we het enige even priemgetal apart behandelen. Onder aanname van de geldigheid van de declaratie

"integer k, j"

luidt de uitwerking:

Centrale ... theek
T.U. ...


```
p[1]:= 2; k:= 1; j:= 1;
while k < 1000 do
  begin "verhoog oneven j tot het volgende oneven priemgetal";
        k := k + 1; p[k] := j
end
```

Hier heeft k de betekenis "het aantal ingevulde priemgetallen", j is altijd oneven en (na de eerste slag) gelijk aan het laatst ingevulde priemgetal.

Nu zitten we nog met de handeling "verhoog oneven j tot het volgende oneven priemgetal". Onder geldigheid van de declaratie

"boolean jdeelbaar" kunnen we dit de uitwerking geven:

```
do begin j := j+ 2; " geef aan jdeelbaar de betekenis: j is deelbaar" end
until non jdeelbaar
```

Dat we hier j meteen met stapjes van 2 kunnen laten oplopen danken we aan het feit, dat we ons door de aparte behandeling van het enige even priemgetal verder tot oneven priemgetallen kunnen beperken. Het gebruik van de do-until statement is hier aangewezen omdat de uitspraak "jdeelbaar" betrekking heeft op zoiets als het volgende "candidaat priemgetal". Zonder ten minste één verhoging van j is het begrip "volgende" natuurlijk niet gedefinieerd.

Nu zitten we nog met de handeling "geef aan jdeelbaar de betekenis: j is deelbaar". Het is pas nu, dat enige algebra om de hoek komt kijken. We gaan kijken of we van de (oneven) j een (dus oneven) priemfactor kunnen vinden: de priemfactoren, die we willen proberen staan echter al in het inmiddels gevulde stuk van het array p! Omdat we alleen oneven waarden van j onderzoeken hoeven we p[1] = 2 als mogelijke factor niet te proberen en is p[2] de kleinste factor, die mogelijk in aanmerking komt. Verder heeft elk deelbaar getal tenminste 1 priemfactor, waarvan het kwadraat niet groter is dan dat getal en op grond van deze twee overwegingen kan het zoeken naar factoren van j zich beperken tot het proberen van

$$p[2] \ldots\ldots\ldots\ldots p[ord - 1]$$

als ord de kleinste index is, en dus p[ord] het kleinste priemgetal is, zodat

$$p[ord]^2 > j \quad .$$

Onder aanname van de geldigheid van de declaratie

"integer ord, m"

en de veronderstelling, dat aanvankelijk, toen j = 1 was, ook ord = 1 gemaakt is, kunnen we "geef aan jdeelbaar de betekenis: j is deelbaar" de volgende uitwerking geven:

```
while p[ord] * p[ord] ≤ j do ord := ord + 1;
m := 2; jdeelbaar := false;
while m < ord and non jdeelbaar do
  begin " geef aan jdeelbaar de betekenis: p[m] is een factor van j";
        m := m + 1
  end
```

In de eerste cyclus wordt (zo nodig) ord opgehoogd, daarna worden (zo nodig, dwz. voor $j \geq 9$; ga dat na!) priemgetallen uit de tabel als factor geprobeerd, maar dit zoekproces strekt zich tenhoogste uit over de priemgetallen

$$p[2] \ldots\ldots p[ord - 1]$$

en wordt onmiddellijk afgebroken, zodra een factor van j gevonden is. (In bovenstaande uitwerking worden de priemgetallen in opklimmende volgorde geprobeerd omdat de kans dat een willekeurige oneven j door een priemgetal deelbaar is, groter is naarmate het priemgetal kleiner is.)

Ten leste hebben we de uitwerking van "geef aan jdeelbaar de betekenis: p[m] is een factor van j". Dit is het geval, wanneer het op gehelen afgeronde quotient gelijk is aan het quotient zelf. Het op gehelen afgeronde quotient kunnen we onder aanname van de geldigheid van de declaratie

"integer q"

in ALGOL 60 met een trucje aan de variabele q toekennen door de assignment statement

"q:= j / p[m]"

omdat bij waardetoekenning aan een integer variabele van een resultaat, dat niet noodzakelijkerwijs geheel is, deze afronding is inbegrepen. De uitwerking van "geef aan jdeelbaar de betekenis: j is deelbaar door p[m]" wordt daarmee tot

"q:= j / p[m]; jdeelbaar:= (j = q * p[m])"

Rest ons de uiteindelijke samenstelling van het programma, dwz. het verzamelen van de declaraties en het substitueren van de opeenvolgende uitwerkingen; het printen van de priementabel gebeurt hier met 8 getallen per regel (zie EWD242 - 21).

```
begin integer array p[1:1000]; integer k, j, ord, m, q; boolean jdeelbaar;
    p[1]:= 2; k:= 1; j:= 1; ord:= 1;
    while k < 1000 do
      begin do begin j := j + 2;
                     while p[ord] * p[ord] ≤ j do ord := ord + 1;
                     m := 2; jdeelbaar := false;
                     while m < ord and non jdeelbaar do
                       begin q := j / p[m]; jdeelbaar := (j = q * p[m]);
                             m := m + 1
                       end
               end
            until non jdeelbaar;
            k := k + 1; p[k] := j
      end;

    k := 0; j := 8;
    while k < 1000 do
      begin k := k + 1; print(p[k]);
            j := j - 1; if j = 0 then begin nlcr; j := 8 end
      end
end
```

TECHNISCHE HOGESCHOOL EINDHOVEN
Onderafdeling der Wiskunde

Elementaire cursus programmeren in ALGOL 60

behorende bij de colleges
Wiskunde 31 en Wiskunde 49

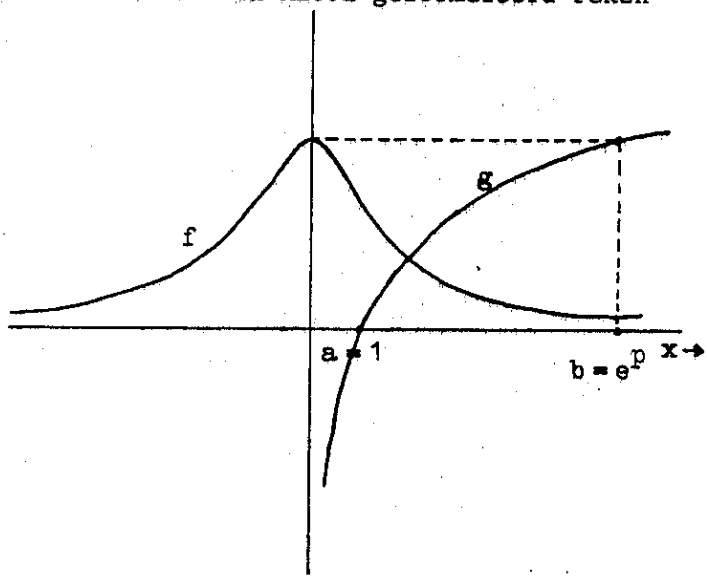
INLEIDING

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 rekenmachine een in ALGOL geformuleerd rekenproces kan uitvoeren.

Voorbeeld.
Beschouw het volgende probleem:
Bepaal in drie decimalen nauwkeurig het snijpunt (x_s, y_s) van de krommen

$f(x) = \frac{p}{1 + x^2}$ en $g(x) = \log x$

bij gegeven positieve waarde van de parameter p.



Een methode om dit probleem op te lossen is de volgende.
Het is zeker dat $f(1) > g(1)$ en $f(e^p) < g(e^p)$.
Zij $a = 1$ en $b = e^p$ dan ligt dus x_s in $[a,b]$.
Zij $c = \frac{1}{2}(a + b)$. Als $f(c) < g(c)$ dan ligt x_s in $[a,c]$.
In dit geval nemen we $b = c$ en a blijft ongewijzigd.
Als echter $f(c) \geq g(c)$ dan ligt x_s in $[c,b]$. Dan nemen we $a = c$ en b blijft ongewijzigd.
In beide gevallen geldt weer x_s ligt in $[a,b]$. De lengte van het nieuwe interval $[a,b]$ is de helft van de lengte van het oude interval $[a,b]$. Dit proces wordt herhaald tot de lengte van $[a,b]$ kleiner dan 10^{-3} is geworden.
Dan is $x_s = \frac{1}{2}(a + b)$ en $y_s = \frac{p}{1 + x_s^2}$ het gevraagde snijpunt met de gewenste nauwkeurigheid.
De formulering van dit rekenproces in ALGOL luidt (voor het geval $p = 2$):

```
begin   real a, b, c, xs, ys, p;
        p := 2; a := 1; b := exp(p);
        do begin c := (a+b) / 2;
                 if ln(c) < p / (1 + c × c) then a := c else b := c
           end
        until (b - a) < .001;
        xs := (a + b) / 2; ys := p / (1 + xs × xs);
        nlcr; print(p); print(xs); print(ys)
end
```

§1. ASSIGNMENT STATEMENT

In het voorbeeld hebben we voor p de waarde 2 genomen. In de ALGOL tekst staat niet p = 2 maar

```
p := 2
```

Het hier gebruikte symbool := (uit te spreken als "wordt") is een toekenningsteken en niet een gelijkteken. Het betekent dat aan de variabele in het linkerlid de waarde 2 wordt toegekend. Een dergelijke formule heet dan ook een assignment statement. Door middel van een assignment statement wordt aan een variabele een waarde toegekend.
Het gebruikelijke symbool = is door het symbool := vervangen om de nadruk te leggen op de asymmetrie. De variabele in het linkerlid krijgt z'n waarde van het rechterlid. Bovendien wordt door het symbool := tot uitdrukking gebracht dat het hier gaat om een handeling die uitgevoerd moet worden en niet, zoals bij het symbool =, om een relatie waaraan al of niet voldaan kan zijn. Zo wordt door de assignment statement

```
i := i + 1
```

de waarde van i met 1 opgehoogd, terwijl de relatie

```
i = i + 1
```

voor geen enkele waarde van i juist is.
Een ander voorbeeld van een assignment statement is

```
xs := (a + b)/2
```

Deze assignment statement geeft aan xs de waarde die verkregen wordt door het rechterlid uit te rekenen. Deze assignment statement is alleen uitvoerbaar als a en b inmiddels (in vorige assignment statements) een waarde hebben gekregen.

§2. IDENTIFIERS EN NUMBERS

In het voorbeeld hebben we variabelen gebruikt welke we namen hebben gegeven (a, b, xs, enz.).
In ALGOL worden identifiers[1] (namen) o.a. gebruikt om variables (variabelen) aan te duiden. In het vervolg zullen we zien dat ook hele processen door identifiers worden aangeduid.

Een identifier is een opeenvolging van letters en cijfers, te beginnen met een letter.

Voorbeelden van identifiers:	Geen identifiers zijn:
wiskunde 31	31
1ste keer	1ste keer
auto	auto's
h2so4	pv = rt
xn	xn - 1
a23	a23.2
den haag	's-gravenhage

In ALGOL geldt de afspraak dat spaties en overgang op een nieuwe regel geen betekenis hebben. Zo mag de identifier "algol 60" ook geschreven worden als "alg o160" of "a l g o l 60".

In het rechterlid van een assignment statement kunnen behalve variabelen ook getallen voorkomen. De getallen worden normaal in het tientallig stelsel genoteerd, maar de schrijfwijze is ook hier gereglementeerd.

Numbers (getallen) worden onderscheiden in twee typen aangegeven door de symbolen integer en real.

N.B. Naast letters, cijfers en symbolen als + , - , (, zijn bepaalde onderstreepte woorden ook ALGOL symbolen. Door de onderstreping verandert het woord in een nieuw ALGOL symbool. Bijvoorbeeld, real moet men niet lezen als een identifier, welke bestaat uit 4 onderstreepte letters maar als één zelfstandig symbool.

1) We gebruiken in de tekst de vaktermen zoals die gebezigd worden in het Revised Report on the Algorithmic Language ALGOL 60.

Een integer is een opeenvolging van cijfers al of niet voorafgegaan door een + of - teken. Een integer is van het type integer.
Ieder ander number is van het type real ook al is z'n waarde geheel (bv. 42 is een integer maar 42.0 is van het type real). Om de notatieregels voor een number van het type real te beschrijven voeren we eerst de begrippen decimal fraction en exponent part in.

Een decimal fraction is een decimale punt gevolgd door één of meer cijfers.
Een exponent part is het symbool $_{10}$ gevolgd door een integer.

Mogelijke vormen van een number van het type real zijn:

1) integer, gevolgd door decimal fraction, gevolgd door exponent part;
2) integer, gevolgd door decimal fraction;
3) integer, gevolgd door exponent part;
4) decimal fraction, eventueel voorafgegaan door een + of - teken, gevolgd door exponent part;
5) decimal fraction, eventueel voorafgegaan door een + of - teken;
6) exponent part, eventueel voorafgegaan door een + of - teken.

Voorbeelden van numbers:	Geen numbers zijn:
-17	17a
$+23.2_{10} + 3$	$23. + 2_{10} + 3$
-14.231	-14.23.1
0.137	$12_{-10}2$
$12_{10}2$	$a_{10}3$
$-.451_{10} - 1$	$2_{10}a$
.137	$15.21_{10}3.5$
+.137	
$_{10} - 3$	$-._{10}3$
$-_{10} - 3$	$12._{10} - 2$
2.0	2.

N.B. Let er op dat $12_{10}2$ van het type real is, al is z'n waarde geheel.

§3. SIMPLE ARITHMETIC EXPRESSIONS

Een simple arithmetic expression kan worden opgebouwd uit variables, numbers, arithmetic operators en ronde haakjes.
Arithmetic operators zijn:

+	(optelling)
−	(aftrekking)
×	(vermenigvuldiging)
/	(deling)
↑	(machtsverheffing)
div	(gehele deling)
mod	(restbepaling)

Een vermenigvuldiging wordt aangegeven door het teken ×. Dit teken mag niet worden weggelaten. Het is dus niet toegestaan het product a × b als ab te schrijven; ab is immers een identifier. Ook a . b is niet toegestaan, de punt wordt alleen als decimale punt gebruikt. In handgeschreven tekst verdient het aanbeveling om in plaats van × te schrijven *, om verwarring met de letter x te voorkomen.
De volgende prioriteitsregels gelden: Machtsverheffing heeft de hoogste prioriteit, dan volgen vermenigvuldiging, deling en restbepaling, en tenslotte optelling en aftrekking. Verder worden de operaties van links naar rechts uitgevoerd.
In expressies wordt zo nodig gebruik gemaakt van ronde haakjes, maar niet van accolades of rechte haken.

De exponent van een machtsverheffing wordt niet hoger geschreven dan het grondtal. Bijvoorbeeld a^2 wordt in ALGOL a ↑ 2.
Komt in de exponent een arithmetic operator voor dan moeten (op grond van de prioriteitsregels) haakjes gebruikt worden. Bijvoorbeeld a^{2p} wordt in ALGOL a ↑(2 × p). Twee arithmetic operators mogen niet naast elkaar staan. Dus voor a^{-2} schrijft men a ↑(-2) en niet a ↑ -2.

Het gebruik van, strikt genomen, overbodige haakjes kan de duidelijkheid van de ALGOL tekst verhogen. Bijvoorbeeld

a × b / c × d is equivalent met ((a × b)/c)× d

Deze laatste schrijfwijze geeft duidelijk aan dat niet bedoeld wordt

(a × b) / (c × d)

Voor gehele getallen is er naast de gewone deling (operator /) ook nog de mogelijkheid van de gehele deling (operator **div**). In het Revised Report wordt deze operator genoteerd als ÷. Als voor twee gehele getallen m en n geldt

$$|m| = q \times |n| + r \quad \text{waarbij } 0 \leq r < |n|$$

dan is

$$m \ \underline{\text{div}} \ n = \text{sign}(m \times n) \times q \quad \text{en} \quad m \ \underline{\text{mod}} \ n = \text{sign}(m \times n) \times r \ .$$

Voorbeelden.

Wiskundige formule	incorrecte expressie	correcte ALGOL expressie
$(a + b)(c + d)$	(a + b)(c + d)	(a + b) × (c + d)
$2n - 1$	2n - 1	2 × n - 1
$\frac{1}{1 + x}$	1 / 1 + x	1 /(1 + x)
$(2^3)^5$	2 ↑(3 ↑ 5)	(2 ↑ 3)↑ 5 of 2 ↑ 3 ↑ 5
2^{3^5}	2 ↑ 3 ↑ 5	2 ↑(3 ↑ 5)
$x \cdot \frac{-1}{(a + b)c}$	x × −1/(a + b) ×c x ×(−1)/(a + b) ×c	x ×(−1)/((a + b) ×c)
$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$	ad - bc	a × d - b × c

Opmerking: in de middelste kolom zijn de 3^e, 4^e, 5^e, 7^e en 8^e expressie wel correct ALGOL, maar geen correcte weergave van de wiskundige formule.

De eenvoudigste vorm van een arithmetic expression is een number of een variable. Er zijn, behalve de behandelde simple arithmetic expressions, nog andere vormen van arithmetic expressions.
We gaan hier verder niet op in.

In §1 is het begrip assignment statement genoemd.
We zijn nu in staat nader te formuleren wat een assignment statement is.

Een assignment statement is een variable, gevolgd door het symbool := ,
gevolgd door een arithmetic expression.

Er bestaat een tweede vorm van de assignment statement (zie blz. 11 onderaan).

§4. HET TYPE VAN VARIABLES EN SIMPLE ARITHMETIC EXPRESSIONS

We hebben gezien dat numbers door hun representatie van een bepaald type zijn, nl. real of integer.
De variables zullen van een bepaald type worden verklaard door middel van een zogenaamde type declaration.
Een dergelijk type declaration bestaat uit het symbool real of integer gevolgd door een of meer identifiers, welke onderling gescheiden zijn door een komma.

Voorbeelden,

```
real a, b, c, xs
integer n, m
```

Declarations worden onderling en van statements gescheiden door een puntkomma. Een puntkomma wordt ook gebruikt om statements te scheiden. Een scheidingsteken is noodzakelijk omdat spaties en overgang op een nieuwe regel geen betekenis hebben in ALGOL.

De integer declaration houdt in dat de zo gedeclareerde variables slechts gehele waarden kunnen krijgen.
Een real gedeclareerde variable kan een willekeurige reële waarde krijgen.
We moeten ons bij real gedeclareerde variables getallen denken, die slechts in eindige precisie gegeven zijn. De waarde van een integer gedeclareerde variable daarentegen wordt exact weergegeven.

De waarde van een simple arithmetic expression is een getal van een bepaald type, dat afhangt van de operators en van het type van de numbers en variables in de expressie.

- a + b, a − b, a × b: is van het type integer als zowel a als b van het type integer zijn, in alle andere gevallen van het type real.
- a / b: is van het type real.
- a div b, a mod b: is alleen gedefinieerd als a en b beide van het type integer zijn, en is dan van het type integer.
- a ↑ b: is van het type integer als a en b beide van het type integer zijn en bovendien b ≥ 0, in alle andere gevallen van het type real.

Bij de assignment statement

variable := arithmetic expression

of

v := AE

is de toekenning duidelijk wanneer v en AE van hetzelfde type zijn.
Het is toegestaan dat het type van het linkerlid en van het rechterlid verschillend zijn.
Is v van het type real en AE van het type integer, dan wordt de waarde van AE als getal van het type real aan v toegekend.
Is v van het type integer en AE van het type real, dan wordt de waarde van AE afgerond op het dichtstbijzijnde gehele getal en als integer aan v toegekend. Is de waarde van AE binnen de rekennauwkeurigheid gelijk aan n + $\frac{1}{2}$, dan is op grond van de eindige precisie van de arithmetiek de waarde van v niet gedefinieerd. Deze kan nl. n zijn maar ook n + 1.

§5. CONDITIONAL STATEMENT

Het komt veel voor dat we het verloop van het programma laten beïnvloeden door het resultaat van de tot nu toe verrichte berekeningen. Willen we b.v. in een programma tot uitdrukking brengen dat de variabele x als waarde moet aannemen het maximum van de waarden van x en y, dan zien we dat als voldaan is aan y > x, dat dan x de waarde van y moet krijgen. De volgende ALGOL statement bewerkstelligt dit doel:

```
if y > x then x := y
```

Dit is een zogenaamde "if statement".

Een if statement bestaat uit een if clause gevolgd door een statement. De if clause bestaat uit het symbool if gevolgd door de voorwaarde op grond waarvan de statement al of niet moet worden uitgevoerd, gevolgd door het symbool then.
De voorwaarde heet een Boolean expression.

Een vorm van een Boolean expression is een relation.

Een relation is een simple arithmetic expression, gevolgd door een relational operator, gevolgd door een simple arithmetic expression.

Relational operators zijn

$<$	(kleiner dan)
$\leq$	(kleiner dan of gelijk aan)
$=$	(gelijk aan)
$\geq$	(groter dan of gelijk aan)
$>$	(groter dan)
$\neq$	(niet gelijk aan)

Voorbeelden.

$a = b$

$x + 1 \leq y \uparrow 2$

$x \times (y \uparrow 2 + z \uparrow 2) > -1$

maar niet $0 \leq x \leq 1$.

De waarde van een Boolean expression kan zijn true of false.
Blijkt na de berekening van de simple arithmetic expressions dat aan de relatie voldaan is, dan krijgt de Boolean expression de waarde true, in het andere geval de waarde false.

Voorbeelden:

Boolean expression	waarde
$1 \geq 2$	false
$a = 2$	true als $a = 2$
	false als $a \neq 2$
$x \geq x + 10$	false voor iedere waarde van x
$(x + 3) \times (x - 2) \geq 0$	true als $x \geq 2$ of $x \leq -3$
	false als $-3 < x < 2$

Naast variables van het type integer of real zijn er in ALGOL ook variables van het type boolean.
Een variable van het type boolean moet gedeclareerd worden door een type declaration. Een dergelijke type declaration bestaat uit het symbool boolean, gevolgd door een of meer identifiers, onderling gescheiden door een komma.
De waarde van een variable van het type boolean is true of false. Zo'n variable krijgt zijn waarde door middel van een assignment statement, waarbij rechts een Boolean expression staat. Dit is de tweede vorm van een assignment statement.

De eenvoudigste vorm van een Boolean expression is true, false of een variable van het type Boolean. Boolean expressions kunnen ook gevormd worden door deze eenvoudige expressions en relations met logical operators te verbinden. Hiervoor verwijzen wij naar de bladzijden 25, 26 en 27 van het diktaat "Algolrithmen".

Voorbeeld.

```
real x, y; boolean b, c, d;
x := 2; y := 3.14;
b := true;
c := x > y;
d := b
```

De Boolean expression in de if clause van een if statement heeft het effect dat de statement volgend op then slechts wordt uitgevoerd, indien de waarde van de Boolean expression true is. Is de waarde false dan wordt de statement volgend op then overgeslagen.

Voorbeeld.
De waarde van de variable x wordt door de if statement

```
if x < 0 then x := - x
```

gelijk aan de absolute waarde van x.

Behalve een assignment statement kan de statement volgend op de if clause ook een compound statement zijn.

Een compound statement bestaat uit een aantal statements, voorafgegaan door het symbool begin en gevolgd door het symbool end. De symbolen begin en end heten statement haken.

Voorbeeld.

```
S0; if B then begin S1; S2 end; S3; S4
```

Heeft B de waarde true, dan worden achtereenvolgens uitgevoerd S0, S1, S2, S3, S4. Heeft B de waarde false, dan worden achtereenvolgens uitgevoerd S0, S3, S4.

Een if statement is één van de twee vormen van een conditional statement die wij behandelen. De andere vorm van een conditional statement is een if statement, gevolgd door het symbool else, gevolgd door een statement.
Ter onderscheiding is het zinvol deze vorm van conditional statement "alternative statement" te noemen.

De alternative statement

```
if B then S1 else S2
```

is gelijk aan de statement S1, als B de waarde true heeft, en gelijk aan de statement S2, als B de waarde false heeft.

Voorbeelden.

1) Voor gegeven waarden van x kan men de berekening van $y = |x + 2|$ en $z = |x + 2| + x$ beschrijven door middel van de statement

```
if x ≥ -2 then begin y := x + 2; z := 2 × x + 2 end
          else begin y := -x - 2; z := -2 end
```

2)
```
S0; if B then begin S1; S2 end else S3; S4
```

Heeft B de waarde true, dan worden achtereenvolgens uitgevoerd S0, S1, S2, S4.

Heeft B de waarde false, dan worden achtereenvolgens uitgevoerd S0, S3, S4.

3) Van de vierkantsvergelijking $x^2 + 2bx + c = 0$ zijn de wortels z_1 en z_2. Gevraagd wordt te berekenen $\max(|z_1|,|z_2|)$.

We veronderstellen dat de variabelen b en c een bepaalde waarde hebben gekregen.

Het volgende stuk programma geeft dan aan de variabele z abs de gevraagde absolute waarde.

```
d := b ↑ 2 - c;
if d < 0 then z abs := c ↑ 0.5
         else if b < 0 then z abs := -b + d ↑ 0.5
                       else z abs :=  b + d ↑ 0.5
```

De statement volgend op then mag geen conditional statement zijn. Dus op het symbool then mag nooit volgen het symbool if. Door middel van de statementhaken begin en end kan men echter van iedere statement een compound statement maken. Op else mag iedere willekeurige statement volgen.

Voorbeeld.

De volgende statement geeft aan max de waarde van het maximum van 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
```

§6. WHILE STATEMENT [1]

Naast de conditionele statement is de mogelijkheid om statements te herhalen een uiterst belangrijk middel om het verloop van het programma te beïnvloeden. Zo'n herhaling wordt ook wel cyclus (programma-lus, Engels: loop) genoemd. De eerste cyclische statement die wij behandelen is de zogenaamde "while statement".

Voorbeeld: [2]

Een programma om $\sum_{n=1}^{1000} \frac{1}{n^2}$ te berekenen en af te drukken zou er als volgt kunnen uitzien:

```
begin integer n ; real som ;
      som := 0 ; n := 0 ;
      while n < 1000 do
         begin n := n + 1; som := som + 1/n ↑ 2 end ;
      print(som)
end
```

Tengevolge van de while statement zal de compound statement in het programma 1000 keer uitgevoerd worden.

1) De while statement en de do-until statement (zie §7) behoren niet officieel tot ALGOL 60. Zij vormen echter enerzijds een dusdanig essentiëel gereedschap voor de programmeur en anderzijds een zo natuurlijke uitbreiding van ALGOL 60 dat zij toch in het kader van deze kursus moeten worden behandeld. In veel ALGOL systemen, waaronder dat op de T.H.E. zijn deze uitbreidingen verwezenlijkt.

2) De betekenis van de statement **print(E)**, waarin E een willekeurige arithmetic expression mag zijn, is "typ de waarde van E".
Om aan een variable x een waarde toe te kennen, die op ponsband staat, zullen we gebruik maken van de assignment statement x := **read**. Deze assignment statement betekent "de waarde van x wordt het volgende getal op de band".

Merk op, dat iedere keer als we een volgende term bij de partiële som optellen, n de betekenis heeft: **"aantal bijgetelde termen".**
Als de som wordt afgedrukt geldt n = 1000. (Ga dit na).

De algemene vorm van de while statement is:

while B do S

hierin staat B voor een Boolean expression en S voor een statement.
De statement volgend op do zal nul of meer keer worden herhaald, en wel zolang de Boolean expression de waarde true oplevert. Heeft voor het begin van de while statement B de waarde false, dan zal S niet worden uitgevoerd en is daarmee de while statement voltooid.
Merk op dat indien S de waarden van de variabelen voorkomend in B niet kan beinvloeden de while statement hetzij equivalent is met een "lege statement" (indien B aanvankelijk de waarde false heeft) en dus had kunnen worden weggelaten, hetzij aanleiding geeft tot een nimmer eindigende herhaling (indien B aanvankelijk true is).

De while statement leent zich in het bijzonder voor cycli waarbij niet aan het begin vast staat hoeveel herhalingen zullen moeten plaats hebben.
Voorbeeld,
een programma dat afdrukt de kleinste macht van 2 groter of gelijk aan een natuurlijk getal (te lezen van een invoerband). Hierbij mag geen gebruik worden gemaakt van deling, vermenigvuldiging of machtsverheffing.

```
begin integer tweem, n ;
      n := read ; tweem := 1 ;
      while tweem < n do tweem := tweem + tweem ;
      print(tweem) ;
end
```

N.B. De statement tussen then en else mag niet een while statement zijn.

Elk ALGOL programma begint met het symbool begin en eindigt met het symbool end. De declarations moeten staan ná begin en vóór de eerste statement. In deze declarations mag iedere identifier slecht één keer voorkomen.

§7. DO-UNTIL STATEMENT *)

De tweede vorm van cyclische statement die wij behandelen lijkt veel op de while statement.
De "do-until statement" heeft de gedaante:

do S until B

De statement S zal tenminste één keer worden uitgevoerd. Is na afloop de waarde van de Boolean expression B false, dan zal S opnieuw worden uitgevoerd, en wel zo vaak totdat B voor de eerste keer de waarde true aanneemt.

Voorbeeld (vgl. met het tweede voorbeeld van §6):
een programma dat afdrukt de grootste macht van 2 kleiner of gelijk een natuurlijk getal n (te lezen van een invoerband). Hierbij mag geen gebruik worden gemaakt van deling, vermenigvuldiging of machtsverheffing.
Analyse: het programma moet een tweemacht door verdubbeling vergroten totdat deze voor de eerste keer n overschrijdt. De vorige tweemacht is dan het gezochte getal.

```
begin integer n, vorige, tweem ;
      n := read ; tweem := 1 ;
      do begin vorige := tweem ;
               tweem := tweem + tweem
         end
      until tweem > n ;
      print(vorige)
end
```

Ga na dat één verhoging van tweem noodzakelijk is omdat voor de beginwaarde van tweem geldt tweem = 1, en voor alle natuurlijke getallen n: 1 ≤ n.
Met de while statement is de volgende oplossing mogelijk:

```
      .............
      vorige := 1 ; tweem := 2 ;
      while tweem ≤ n do
         begin vorige := tweem ; tweem := tweem + tweem end ;
      print(vorige)
      .............
```

*) Zie voetnoot 1) onderaan blz. 14

§8. FOR STATEMENT

In het voorbeeld op blz. 14 moet de statement

```
som := som + 1 / n ↑ 2
```

1000 keer worden uitgevoerd, waarbij n achtereenvolgens de waarde 1,2,3,...,1000 heeft.
Een dergelijk herhaald uitvoeren van een statement kan in ALGOL ook beschreven worden met behulp van een zogenaamde for statement.
Het stukje programma

```
n := 0 ;
while n < 1000 do
   begin n := n + 1 ; som + 1/n ↑ 2 end
```

kan ook worden geformuleerd als:

```
for n := 1 step 1 until 1000 do som := som + 1 / n ↑ 2.
```

Het stukje "for ... do" heet for clause. Het effect van deze for clause is dat de statement volgend op het symbool do 1000 keer wordt uitgevoerd, waarbij n achtereenvolgens de waarde 1,2,3,...,1000 heeft.

Een for statement bestaat uit een for clause gevolgd door een statement.

De statement volgend op een for clause mag iedere willekeurige statement zijn, dus ook bijvoorbeeld een for statement.

Van de verschillende vormen van een for clause behandelen we alleen de volgende vorm

```
for k := ℓ step m until n do
```

waarin k een variable van het type integer is en ℓ, m en n arithmetic expressions (dus bijvoorbeeld variables of numbers) van het type integer zijn.

De uitvoering van de for statement

```
for k := ℓ step m until n do S
```

kunnen we als volgt beschrijven

```
k := ℓ ;
while (n - k) × m ≥ 0 do begin S; k := k + m end
```

Leeszaal
Centrale Bibliotheek
T.H. Eindhoven

Het is dus ook toegestaan dat m negatief is. In dat geval wordt de for statement beëindigd als k < n geworden is.
Verder blijkt: als (n - ℓ) × m < 0 dan wordt de statement S geen enkele keer uitgevoerd.

Voorbeeld.
Bepaal van de reeks

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \qquad |x| < 1$$

de som van de eerste n termen bij gegeven waarden van x en n.
Het berekenen van de som wordt in het volgende stuk ALGOL programma beschreven.

```
som := 0; term := - 1;
for k := 1 step 1 until n do
    begin term := term × (-x);
          som := som + term / k
    end
```

Het gebruik van de for statement is uitsluitend aan te raden voor die cyclische statements waarvan op het moment dat er aan wordt begonnen vaststaat hoeveel herhalingen van de statement zullen plaats hebben.

N.B. De statement tussen then en else mag geen for statement zijn (evenmin als een while statement).

§9. ARRAYS

Een array dient om een rij geïndiceerde variabelen aan te duiden, bijvoorbeeld een vector of een matrix. De verzameling van geïndiceerde variabelen wordt in ALGOL aangeduid door een identifier, genaamd array identifier. Een element van deze verzameling wordt aangeduid door een subscripted variable.

Een subscripted variable bestaat uit een array identifier, gevolgd door het symbool [, gevolgd door één of meer subscripts (indices) onderling gescheiden door een komma, gevolgd door het symbool].

Als subcripts zijn toegestaan willekeurige arithmetic expressions van type integer.

Voorbeelden.

```
x[1]        x[2 × n + 1]
a[1,2]      a[i,j + 1]
b[5,3,-2]   b[1, a2 - 1,x + y + z]
```

Een subscripted variable is een variable van een bepaald type (real, integer of boolean) en kan als zodanig in een arithmetic, resp. Boolean expression voorkomen. Alle subscripted variables van één array zijn van hetzelfde type. Voordat we een array identifier als zodanig mogen gebruiken moet deze identifier gedeclareerd zijn door een array declaration. In een array declaration worden opgegeven het type van de subscripted variables van het array, de array identifier en de grenzen waartussen de subscripts moeten liggen. De onder- en bovengrens voor een subscript worden gescheiden door een dubbele punt; de grenzen voor verschillende subscripts door een komma.

Voorbeelden.

```
real array    v[0 : 4]
integer array a[-1 : 7, 0 : 13]
boolean array b[4 : 5, 3 : 3, -7 : -1]
real array    x[1 : 10], y[1 : 14], z[0 : 5]
```

We beperken ons voorlopig tot het geval dat de grenzen integers zijn.
De declaration is alleen correct als de ondergrens niet groter is dan de bovengrens. Incorrect is bijvoorbeeld de declaration

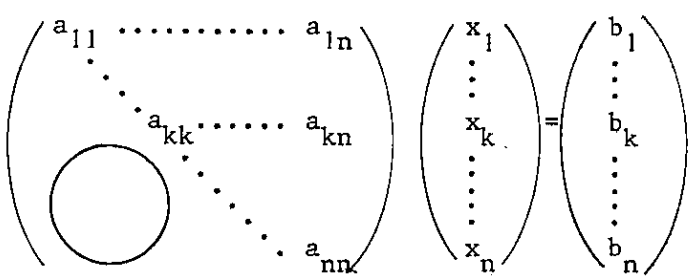
```
real array a[10 : 3, 4 : 7]
```

Een subscripted variable is alleen gedefinieerd als het aantal subscripts overeenstemt met het aantal grenzenparen in de array declaration en als bovendien de waarde van de subscript niet buiten de opgegeven grenzen ligt.

Voorbeeld.
Het volgende stukje programma bepaalt van een gegeven rij geïndiceerde variabelen $x_1, x_2, \ldots, x_{379}$ het maximum en de kleinste index waarvoor dit maximum wordt aangenomen.

```
max := x[1]; k := 1;
for i := 2 step 1 until 379 do
    if x[i] > max then begin k := i; max := x[k] end
```

Voorbeeld.
De oplossing van het volgende stelsel lineair onafhankelijke vergelijkingen



wordt gegeven door

$$x_k = (b_k - \sum_{i=k+1}^{n} a_{ki}x_i)/a_{kk} \qquad k = n, n-1, \ldots, 1$$

Het volgende stuk ALGOL programma berekent bij gegeven waarden van a_{ki} en b_k bovenstaande oplossing.

```
for k := n step - 1 until 1 do
    begin w := b[k];
          for i := k + 1 step 1 until n do w := w - a[k,i] × x[i];
          x[k] := w / a[k,k]
    end
```

Voorbeeld.

Het product van een m × n matrix A en een n × p matrix B is een m × p matrix C waarbij

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

De vermenigvuldiging kan in ALGOL als volgt beschreven worden

```
for i := 1 step 1 until m do
    for j := 1 step 1 until p do
        begin c[i,j] := 0;
              for k := 1 step 1 until n do
                  c[i,j] := c[i,j] + a[i,k] × b[k,j]
        end
```

§10. BLOCK STRUCTUUR

Een compound statement bestaat uit een aantal statements, onderling gescheiden door een puntkomma, voorafgegaan door het symbool begin en gevolgd door het symbool end. Wanneer behalve statements ook declarations tussen begin en end staan, dan noemen we dit geheel een block. De declarations moeten staan tussen het symbool begin en de eerste statement.

Een block bestaat uit het symbool begin, gevolgd door één of meer declarations, gevolgd door één of meer statements, gevolgd door het symbool end.

Een block is een statement, welke in het programma mag staan op iedere plaats waar een compound statement mag staan. Een block is echter essentieel iets anders dan een compound statement. De identifiers, welke in een block gedeclareerd zijn heten lokaal (t.o.v. het block). Dit betekent:

a) dat het object, aangeduid door een dergelijke identifier, buiten het block niet bestaat;
b) dat een object, dat buiten het block door dezelfde identifier worden aangeduid, binnen het block niet bestaat.

Uit a) volgt dat de lokale identifiers van een block buiten dit block vrij ter beschikking staan om gebruikt te worden.
Uit b) volgt dat door een declaration een identifier vrij gemaakt wordt van zijn vroegere betekenis.
Dit heeft grote voordelen. Bijvoorbeeld, als twee personen samen een programma schrijven, dan hoeft de een van de ander niet te weten welke lokale identifiers hij gebruikt.

Bij het verlaten van een block verliezen de lokale variables hun betekenis en ook hun waarde. Dit impliceert dat bij het opnieuw binnenkomen van het block de lokale variables geen waarde hebben.
Identifiers, die niet in een (binnen-)block opnieuw gedeclareerd worden, behouden hun betekenis en eventuele waarde zowel bij het binnenkomen als bij het verlaten van het block. Deze identifiers noemen we niet-lokaal ten opzichte van dit (binnen-)block.

In §9 hebben we ons bij de array grenzen beperkt tot integers. Als array grenzen zijn toegestaan willekeurige arithmetic expressions van het type integer. Als in het begin van een block een array declaration voorkomt dan worden de waarden van de grenzen bij de binnenkomst in het block uitgerekend. Dit impliceert dat de eventueel gebruikte variables in de grenzen reeds een waarde moeten hebben, dus niet lokaal ten opzicht van dit block kunnen zijn.

Voorbeeld.

```
begin integer n ;
      n := read ;
      begin real array a[1 : n], b[1 : n, 1 : n], c[0 : 2 × n + 1];
            ..............
            ..............
      end
end
```

§11. PROCEDURE

Een block is een min of meer afgerond stuk programma. Het kan gebeuren dat zo'n block op verschillende plaatsen in het programma voorkomt. Om niet iedere keer de tekst van dit block te moeten uitschrijven geven we het block aan door een identifier, genaamd procedure identifier. Deze procedure identifier zetten we dan op de plaatsen in het programma waar het block moet worden uitgevoerd. Deze procedure identifier is een statement (en wordt ook van andere statements gescheiden door een puntkomma). Een statement bestaande uit een procedure identifier heet een procedure statement. Een procedure statement is toegestaan op elke plaats waar een compound statement is toegestaan.

Een procedure identifier moet gedeclareerd worden door middel van een procedure declaration. De eenvoudigste vorm van een procedure declaration bestaat uit het symbool procedure, gevolgd door de procedure identifier, gevolgd door een puntkomma, gevolgd door een zogenaamde procedure body.

De procedure body is het block, dat op de plaats van de procedure statement moet worden uitgevoerd.

Voorbeeld.
Op verschillende plaatsen in het programma moeten de waarde van de twee variables a en b verwisseld worden.
De procedure declaration is

```
procedure wissel;
        begin real s;
              s := a; a := b; b := s
        end
```

Een stuk programma waarin de procedure statement voorkomt is

```
a := 1; b := 5; wissel; x := b - a
```

of als we aan a de grootste en aan b de kleinste van de waarden van a en b willen toekennen

```
if a < b then wissel
```

Naast het gebruik van een procedure identifier als procedure statement kan men een procedure identifier ook gebruiken om de waarde van een functie aan

te duiden. We spreken in dit geval van een function designator.

Voorbeeld.
In de assignment statement

```
y := sin(x) + 1
```

is sin(x) een function designator, waarvan de waarde gelijk is aan sin x bij gegeven waarde van x.

Een function designator is toegestaan op elke plaats waar een Boolean of een arithmetic expression is toegestaan. Een function designator moet worden gedeclareerd door middel van een procedure declaration. Bij deze declaration wordt vóór het symbool procedure het type van de function designator aangegeven door een van de symbolen real, integer of Boolean. Het bepalen van de waarde van een function designator geschiedt door het uitvoeren van de procedure body. Daartoe moet in de procedure body een assignment statement voorkomen waarin aan de procedure identifier een waarde wordt toegekend. Dit is de waarde die de function designator krijgt.

Voorbeeld.
In een programma moet de waarde van een polynoom

$$P(x) = a_0x^{10} + a_1x^9 +\ldots+ a_{10}$$

bij verschillende waarden van x worden berekend.
De hiervoor benodigde declarations zijn

```
real x, y; real array a[0 : 10];
real procedure p;
    begin integer i; real s;
        s := a[0];
        for i := 1 step 1 until 10 do s := s × x + a[i];
        p := s
    end
```

Een stukje programma, waarin p als function designator gebruikt wordt is

```
x := 0; y := p; x := 1; y := y + p
```

De variable x is het argument van het polynoom P(x). Aan ieder gebruik van p als function designator (dit noemt men procedure call) moet een assignment statement vooraf gaan die aan x de gewenste waarde geeft.
In plaats hiervan is het ook mogelijk om bij de declaration aan de procedure identifier het argument toe te voegen in de vorm van een zogenaamde formal parameter, die bij de procedure call vervangen wordt door de gewenste waarde.

Een formal parameter is een identifier, die in de procedure declaration voorkomt tussen ronde haken, direct achter de procedure identifier.

Het vorige voorbeeld wordt nu

```
real y; real array a[0 : 10];
real procedure p(x); value x; real x;
     begin integer i; real s;
           s := a[0];
           for i := 1 step 1 until 10 do s := s × x + a[i];
           p := s
     end
```

Het stukje programma wordt nu de volgende statement

```
y := p(0) + p(1)
```

Bij de procedure call staat op de plaats van de formal parameter de zogenaamde actual parameter. Een actual parameter mag een willekeurige expressie zijn. Andere vormen van actual parameters zullen we niet beschouwen.

Voorbeeld.

```
y := 0.70; y := p(y ↑ 2 + 1)
```

Een formal parameter wordt niet buiten de procedure declaration gedeclareerd. Als bij de procedure declaration aan een procedure identifier een formal parameter is toegevoegd, dan wordt vóór de procedure body een nadere omschrijving van deze formal parameter gegeven. Daarbij betekent "value x" dat de waarde van de actual parameter aan x wordt toegekend. Het gedeelte "value x" mag ook worden weggelaten, maar op de consequenties hiervan gaan we verder niet in. Het gedeelte "real x" legt het type van de waarde van de actual parameter vast.

In de procedure body moet een assignment statement voorkomen met de procedure identifier aan de linkerkant (zie blz. 24). Dit geldt ook als de procedure declaration een formal parameter bevat. In deze assignment statement wordt de parameter dus weggelaten.
Dit is het enige geval waarin de procedure identifier in het linkerlid van een assignment statement voorkomt. Het is in dit geval geen function designator. In alle andere gevallen (uitgezonderd bij een declaration) wordt de procedure identifier gebruikt voor een procedure call. Dan mag een eventuele actual parameter niet worden weggelaten.

In de procedure body van het eerste voorbeeld op blz. 24 moeten we de hulpgrootheid s invoeren. We kunnen namelijk geen gebruik maken van P in de statement

```
s := s × x + a[i]
```

omdat de procedure identifier in het rechterlid van een assignment statement een procedure call betekent.

In een procedure declaration mogen meerdere formal parameters tussen de ronde haken voorkomen. Ze worden onderling gescheiden door een komma. Bij de procedure call moet elke formal parameter vervangen worden door een actual parameter.

Voorbeeld.
In een programma moet op verschillende plaatsen het maximum van twee waarden bepaald worden. We kunnen dit doen met behulp van de procedure max.

```
real procedure max(x ,y); value x, y; real x, y;
    begin real s;
          if x > y then s := x else s := y;
          max := s
    end
```

Het gebruik van de function designator als actual parameter is ook toegestaan. Zo wordt door de statement

```
m := max(max(a, b), c)
```

aan m de waarde max(a, b, c) toegekend.

Een aantal function designators behoeft niet te worden gedeclareerd. Dit zijn de zogenaamde standard functions.

abs(E)	voor de absolute waarde van de expressie E
sign(E)	voor het teken van de waarde van E (+1 voor E > 0, 0 voor E = 0, -1 voor E < 0)
sqrt(E)	voor de vierkantswortel van de waarde van E
sin(E)	voor de sinus van de waarde van E
cos(E)	voor de cosinus van de waarde van E
arctan(E)	voor de arctangens van de waarde van E
ln(E)	voor de natuurlijke logarithme van de waarde van E
exp(E)	voor de waarde e^{E}.

In het voorbeeld van de inleiding hebben we reeds gebruik gemaakt van standard functions.

ALPHABETISCHE INDEX

	bladzijde
actual parameter	26, 27
arithmetic expression	6, 9, 19
arithmetic operator	5, 8
array declaration	19, 20, 23
array grenzen	19, 23
array identifier	19
assignment statement	2, 7, 9, 11
begin ..... end	12, 14, 15, 22
block	22, 23, 24
boolean expression	10, 11, 12, 15, 16, 19
compound statement	12, 22
conditional statement	10, 12, 13
declaration	8, 11, 15, 19, 22, 24
do-until statement	16
else	12, 13
for clause	17
formal parameter	26, 27
for statement	17
function designator	25, 27, 28
gehele deling ÷, div	6
identifier	3, 19, 22, 24, 26
if clause	10, 12
if statement	10, 12
logical operator	12
lokaal	22
number	3, 4
print	14
procedure body	24, 25, 27
declaration	24, 26
identifier	24, 25, 27
statement	24

READ 14, 23
relation 10, 11
restbepaling mod 6
simple arithmetic expression 5, 8
standard function 28
subscripted variable 19, 20
ten $_{10}$ 4
type 3, 8, 9, 11, 19, 25, 26
type declaration 8, 11
value 26
variable 3, 8, 11, 19, 22
while statement 14, 15, 16, 18