TECHNISCHE HOGESCHOOL EINDHOVEN
Afdeling Algemene Wetenschappen
Onderafdeling der Wiskunde

INLEIDING tot de INFORMATICA

Auteur Onbekend

Voorjaarssemester 1979

Technische Hogeschool Eindhoven

Onderafdeling der Wiskunde

Inleiding tot de Informatica

Dictaatnr. 2.269 Prijs *f* 5,-

TECHNISCHE HOGESCHOOL EINDHOVEN

Onderafdeling der Wiskunde

Inleiding tot de Informatica

Voorjaarssemester 1979

Inhoudsopgave

I. Algorithmen

0. Computersystemen

Computersystemen zijn uit de hedendaagse maatschappij niet meer weg te denken, daar zij met grote snelheid en betrouwbaarheid eentonig rekenwerk verrichten, waarvoor anders duizenden rekenaars nodig zouden zijn. Rekenaars, noch computers (rekenautomaten) doen hun werk autonoom; beiden moet verteld worden wat van ze verwacht wordt. Er zijn echter verschillen in de wijze waarop de werkopdrachten aan rekenaars en computersystemen geformuleerd moeten worden:

- rekenaars hebben vaak "aan een half woord" voldoende omdat ze voldoende kritische zin en inzicht bezitten om tijdens hun werk zelf te beslissen hoe zij details zullen verrichten; aan computersystemen moet juist precies vooraf verteld worden wat er kan en moet gebeuren,
- rekenaars controleren zelfstandig hun werk en verbeteren dat indien nodig; computers kunnen niet zelfstandig corrigerend optreden wanneer bijvoorbeeld hun opdrachten onvolledig of dubbelzinnig zijn geweest. (Allerlei klachten en grapjes over computersystemen die de pers halen, zijn dan ook vrijwel altijd toe te schrijven aan onvoldoende en/of onjuiste opdrachten aan een computer door mensen en niet aan de rekenautomaten zelf!)
- voor de communicatie met rekenaars kan zodoende met een mengsel van omgangstaal en wiskundige taal volstaan worden; voor de communicatie met een computer is een precies gedefinieerde taal, een programmeertaal nodig.

Naast deze verschillen zijn er punten van overeenkomst tussen rekenaars en computers; beiden beschikken:

- over een "werkgeheugen" om rekenopdrachten en gegevens vast te houden (bij een computer kan men beter over een "bewaarplaats" spreken omdat het"herinneringsvermogen" zeer beperkt is),
- als verlengstuk van het werkgeheugen over een "achtergrondgeheugen", bij de rekenaar in de vorm van o.a. tabellen en naslagwerken, bij computers in de vorm van magneetbanden, - schijven of - trommels ,
- over een "rekenorgaan" om de uitvoering van de gewone rekenkundige en logische bewerkingen met gegevens uit het werkgeheugen snel uit te voeren,
- over "invoer/uitvoer organen" voor de communicatie met de buitenwereld: bij de rekenaar in de vorm van gezichts-,gehoor-, spraak- en soms tastorganen, bij computers in de vorm van ponsband- of ponskaartmachines, schrijfmachines, regeldrukkers, t.v. schermen, grafieken-schrijvers, enz.

. over een "besturingsorgaan" dat, gehoorzamend aan een programma van werkopdrachten in het werkgeheugen, de samenwerking tussen de hiervoor genoemde organen coördineert.

In ruim twintig jaar tijd is voor computersystemen de belangstelling verschoven van de apparatuur naar de programmatuur, d.w.z. de problemen samenhangende met het proces van de verwerking van gegevens (meestal tot andere gegevens) en in het bijzonder naar de kunst (of kunde?) van het opstellen van opdrachten (meestal "programmeren" genoemd) voor een gewenste gegevensverwerking. Omdat in een bepaalde context gegevens voor de mens informatie bevatten, wordt dit terrein van gegevensverwerking tegenwoordig informatica genoemd. Eigenschappen van rekenautomaten en hun physieke structuur behoren dan tot de randgebieden van de informatica, evenals specifieke toepassingen van informatica-methoden in andere vakgebieden.
Het eerstgenoemde is een randgebied met de informatietechniek.
Bedrijfsinformatica is een ander randgebied; het is een toepassing in het vakgebied van de bedrijfskunde en bedrijfsvoering. Het onderscheid hoofd- en randgebieden is natuurlijk betrekkelijk willekeurig; voor iemand die zich op de informatietechniek concentreert, kan de informatica een randgebied zijn.

Voor een uitvoeriger beschouwing over digitale rekenautomaten zij verwezen naar appendix B.
Naast digitale bestaan ook analoge computers (en hybriede als combinatie van die twee). De eerste zijn op te vatten als geautomatiseerde telramen (getalvoorstelling met discrete eenheden nl. balletjes, de tweede als veredelde rekenlinealen (getalvoorstelling met een continue physische grootheid, nl. een lengte).

1. Processen en hun beschrijvingen

Opgemerkt is reeds dat de informatica zich vooral bezig houdt met problemen, samenhangend met de verwerking van (begin) gegevens tot meestal andere gegevens (verwerking als verzamelnaam voor vergaren, vastleggen, verschaffen en de eigenlijke verwerking of transformatie). Al op het eerste gezicht is er zodoende een overeenkomst met technische processen, waarbij (materiële) grondstoffen verwerkt worden tot tussen- of eindproducten (of soms tot energie in al zijn verschijningsvormen). Deze overeenkomst gaat inderdaad zo ver dat veel gedachten uit de proceskunde over te nemen zijn in de informatica.

Voorbeelden van informatica-processen zijn bijvoorbeeld:

- de uitvoering van (complexe) wiskundige berekeningen, waarbij zowel de grondstoffen als eindproducten getallen zijn (meestal vrij klein in aantal; het aantal uitzonderingssituaties is meestal eveneens klein);
- de uitvoering van administratieve bewerkingen (als bij een girodienst of een magazijnadministratie), waarbij zowel grondstoffen als eindproducten (grote aantallen) getallen en alfabetische teksten zijn; het rekenwerk is wel massaal doch vrij eenvoudig van aard, uitzonderingssituaties zijn vaak talrijk en complex;
- de begeleiding van (technische) processen, waarbij een rekenautomaat in de besturing ingeschakeld is. De grondstoffen zijn hier uit het proces komende, meestal electrische, signalen (en enkele door de toezichthoudende mens ingevoerde instelwaarden). De eindproducten zijn enkele getalwaarden voor rapportage doeleinden en electrische signalen, waarmee het te besturen proces geregeld wordt. Voorbeelden zijn o.a. de besturing van chemische of physische fabrieksprocessen, van cyclotrons, van maanlanders, van fotosatellieten (eindproduct hier foto's van planeten, enz.), patiëntenbewakingssystemen, vliegtuigreserveringssystemen;
- de uitvoering van bibliografische processen, bijvoorbeeld waarbij een rekenautomaat als grondstof trefwoorden accepteert en als eindproduct een lijst van relevante publicaties produceert.

Op details van deze processen hoeft niet te worden ingegaan. Immers, wanneer bij latere beschouwingen een voorbeeld nodig is, wordt dat meestal ontleend aan eenvoudige rekenprocessen, waarmee iedereen wel vertrouwd is.

Karakteristiek voor technische en informatica-processen is hun dynamische karakter, d.w.z. hun afhankelijkheid van de tijd. In de proceskunde is de tijd meestal een continue grootheid. In de informatica is daarentegen als regel de tijd een discrete grootheid; men is namelijk vooral geïnteresseerd in de vordering van een informatica-proces op bepaalde discrete momenten en in veranderingen die tussen die momenten opgetreden zijn. De wijze waarop die veranderingen precies tot stand komen, wordt hierbij buiten beschouwing gelaten.

Voor de beschrijving van een proces in proceskunde en informatica wordt gebruik gemaakt van een verzameling benoemde grootheden (variabelen). Welke variabelen voor een bepaald proces relevant zijn, wordt bepaald door de richting en diepte van belangstelling voor details. Voor de chef van een fabrieksproces is misschien alleen de opbrengst van belang, voor de man die het proces moet regelen zijn wellicht druk en temperatuur van het proces de relevante grootheden, terwijl de man die zich wil verdiepen in het moleculaire procesgebeuren geïnteresseerd zal zijn in de plaats- en snelheidscoördinaten van de bij het proces betrokken moleculen.

De heersende (momentane) waarden van alle grootheden samen definiëren een "toestand". De verzameling van alle mogelijke toestanden wordt de "toestandsruimte" genoemd. Een verandering in de toestandsruimte, een toestandstransformatie, wordt bewerkstelligd door een "actie", d.w.z. iets dat in een eindige tijd de verandering van de waarde van tenminste één toestandsgrootheid met zich meebrengt. Een voorbeeld van zo'n actie is de toekenning van een (nieuwe) waarde aan een toestandsgrootheid.

Een aantal transformaties achter elkaar bewerkstelligt een "sequentieel proces", waarbij dus een rij opeenvolgende toestanden in de toestandsruimte doorlopen wordt. Het eerste element van deze rij heet de begintoestand, het laatste de eindtoestand van een proces. Hoe nauwkeurig men dit proces wil beschrijven hangt weer af van de graad van belangstelling voor het "inwendige" van het proces.
Is men niet in details geïnteresseerd, dan wil men zo'n proces P het liefst opvatten als een (samengestelde) actie die de begintoestand transformeert in de eindtoestand.
Is men daarentegen juist wel in details geïnteresseerd dan wil men het proces beschrijven in termen van "elementaire", d.w.z. niet verder opdeelbare, acties. Deze elementaire acties vormen dan met elkaar een zeker basisrepertoire, dat ten grondslag ligt aan de procesbeschrijving. Wat het basisrepertoire omvat, moet afgesproken worden.

Processen (en acties) spelen zich af in de tijd, zijn dynamisch. De beschrijving van een proces (actie) is echter tijdloos, statisch. Wil een opgestelde procesbeschrijving eenduidig uitvoerbaar zijn, dan moet er overeenstemming zijn tussen opsteller en uitvoerder t.a.v. de taal (-elementen of "opdrachten") waarmee de beschrijving wordt vastgelegd. Is deze taal de wiskundige taal, dan spreekt men van een algorithme (zie voor een beschouwing appendix B § 1) in plaats van procesbeschrijving. Is deze taal aangepast aan een rekenautomaat (als uitvoerder), dan spreekt men van een programma in plaats van een procesbeschrijving. Met een algorithme is voor een rekenaar precies beschreven wat hij uit moet voeren. Met een programma is precies beschreven wat een computersysteem moet verrichten.
De opdracht voor de eerder genoemde toekenning van een waarde aan een variabele wordt genoteerd als:

variabele := (nieuwe) waarde

en wordt assignment statement genoemd.

De notatie voor andere acties komt later aan de orde.

De omstandigheid dat een programmeertaal aangepast moet zijn aan beperkingen van een rekenautomaat maakt het gebruik van een programmeertaal als beschrijvingsmiddel voor een algorithme minder aantrekkelijk. Dit geldt nog eens te meer omdat in het verleden bij het ontwerp van programmeertalen (in meerdere of mindere mate) achteraf minder gelukkig gebleken concepten zijn ingevoerd. Is een programmeertaal echter eenmaal ingeburgerd, dan is het om economische en/of psychologische redenen vrijwel niet mogelijk om nog verbeteringen in die concepten aan te brengen (het invoeren van een geheel nieuwe taal maakt mogelijk nog meer kans).
Aan de andere kant is het gebruik van een dagelijkse omgangstaal ook niet aantrekkelijk als hulpmiddel bij het beschrijven van een algorithme. In hoofdzaak is dit zo omdat beschrijvingen met volzinnen te lang of te vaag zijn.
De hedendaagse tendens is dan ook om voor het ontwerpen van algorithmen en computerprogramma's een "ontwerptaal" te gebruiken waarvan de betekenis (semantiek) formeel, eenvoudig en precies te beschrijven is, evenals de schrijfwijze (syntaxis). Wanneer de ontwerptaal bovendien "goed in het gehoor" ligt, wordt ook het leveren van het correctheidsbewijs van de algorithme of het programma vereenvoudigd. Het omschrijven van een programma in een programmeertaal is dan niet moeilijk meer als men de eigenaardigheden en/of beperkingen van de programmeertaal kent.
In het volgende wordt een ontwerptaal gebruikt om de voornaamste concepten van het programmeren onder de knie te krijgen. Deze ontwerptaal heeft veel gemeen met de programmeertalen Algol 60 en Pascal, zodat het omschrijven in die talen vrij eenvoudig is. Het omschrijven in (oudere) talen als Fortran en Cobol kost door de eigenaardigheden van die talen iets meer moeite.

2. Namen, waarden en typen van variabelen

De beschrijving van toestanden vindt, zoals gezegd, plaats door middel van variabelen (toestandsgrootheden). Om aan een bepaalde variabele te kunnen refereren (d.w.z. er over te kunnen spreken), zal deze van een naam voorzien moeten zijn. In een programma of algorithme hebben namen van variabelen - de Engelse term voor naam is "identifier" - geen intrinsieke, d.w.z. toestandsbepalende betekenis. Dit is als in het gewone leven, waarin heer De Groot best een klein mannetje zou kunnen zijn. Men is in hoge mate vrij in de keuze van de in te voeren namen, maar het is uiteraard verstandig betekenisvolle namen in te voeren en verwarrende namen te vermijden. Om het herkennen van een variabele te vereenvoudigen, wordt

voor het vervolg afgesproken dat

> de naam van een variabele bestaat uit een rij letters en/of cijfers (geen leestekens, maar eventueel wel spaties die geen betekenis hebben), altijd beginnende met een letter.

Toegestaan is dan "seven up", ontoelaatbaar is "7 up". Toelaatbaar zijn "Austin Seven" en "Austin 7", maar deze twee namen duiden wel twee verschillende variabelen aan! (In wiskundige beschouwingen volstaat men meestal met identifiers van 1 letter, maar programma's zijn eenvoudiger te lezen wanneer voor identifiers meer letters gebruikt worden.)

Naast de naam, ter benoeming, (en de later te behandelen structuur) wordt een variabele gekenmerkt door het feit dat een waarde van een variabele een element is van een waardenverzameling en dat zo'n variabele onderworpen kan worden aan bewerkingen (operaties), karakteristiek voor die bepaalde waardenverzameling. Men spreekt van een bepaald "type" variabelen, als ze bij een zekere waardenverzameling behoren. Het eenvoudigst zijn de z.g. "primitieve" typen, aangegeven met integer, real (samen ook arithmetische typen genoemd), boolean (ook wel logisch type genoemd) en character. (De onderstreping wordt gebruikt om aan te geven dat men niet met een naam van een variabele te doen heeft, maar met een symbool voor zijn type-aanduiding.) Op details wordt later ingegaan; voor het moment is voldoende om voor de waarden van variabelen te denken aan getallen, de constateringen waar of onwaar en aan letters, cijfers en andere tekens.

3. Een ontwerptaal voor algorithmen

Slechts voor eenvoudige (reken-)processen zal een algorithme bestaan uit alleen een rij toekenningen. Een voorbeeld van zo'n eenvoudig proces is het verwisselen van de waarden van twee variabelen, genaamd p en q. Het proces hiervoor is, gebruik makend van een hulp-variabele, als volgt te noteren:

h := q
q := p
p := h .

Dat er voor de beschrijving van rekenprocessen als regel meer nodig is, is al te zien aan de volgende twee eenvoudige (reken-)processen.

- Bepaal het aantal keren dat het natuurlijke getal **a** gedeeld kan worden door het natuurlijke getal b (b > 0).
- Vervang het getal p door de absolute waarde van p als $p \leq 0$ en door 2p als $p \geq 0$.

Een algorithme voor het eerste proces is:

. probeer b zo vaak mogelijk van a af te trekken (zonder in het gebied van de negatieve getallen te komen)
. het aantal keren dat deze stap uitgevoerd kan worden is de uitkomst.

Wat formeler is deze algorithme als volgt op te schrijven:

. voer naast a en b twee nieuwe hulpgrootheden q (voor quotiënt) en r (voor rest) in
. beginacties: r := a
q := 0
. zolang $r \geq b$ uit te voeren de acties r := r - b
q := q + 1
. de uiteindelijke q is de gezochte uitkomst.

De juistheid van deze algorithme volgt uit de constatering dat voortdurend geldt: a = q * b + r (een * wordt gebruikt i.p.v. een ×-teken om verwarring met de letter x te voorkomen. Evenmin mag qb worden geschreven omdat dit de naam van een (niet ingevoerde) variabele zou zijn). Het proces is eindig daar de successievelijke waarden van r een monotoon dalende (rekenkundige) rij vormen met een kleinste waarde r waarvoor geldt $0 \leq r < b$. Of er niet een efficiënter algorithme is, wordt hier in het midden gelaten. Aan dit voorbeeld is te zien dat er een behoefte is aan een constructie voor

repetities van het type

zolang "zekere voorwaarde geldt" doe "een aantal acties"

en aan een systematische notatie voor de z.g. "declaratie":

"voer de volgende (hulp-)grootheden in" .

Een algorithme voor het tweede proces zal de volgende gedaante hebben:

als p < 0 dan actie p := -p anders p := 2 * p

als bijzonder geval van

als "zekere voorwaarde geldt" dan "aantal acties" anders "aantal andere acties" .

Onder omstandigheden, met name als de tweede verzameling acties leeg is, zal hiervan alleen nodig zijn het gedeelte

als "zekere voorwaarde geldt" dan "aantal acties".

Van de eerder aangestipte ontwerptaal om algorithmen of programma's op te stellen, zal nu het volgende geëist worden:

. nauwe aansluiting bij de normale wiskundige notaties
. ook enigszins aansluiting bij bestaande programmeertalen, hetgeen o.a. met zich meebrengt dat notaties met indices "onder de regel" of exponenten "boven de regel" vervangen worden door "gelineariseerde" notaties (alles "op de regel"). Voorts moet er rekening mee worden gehouden dat in tegenstelling tot de mens een automaat niet in staat is om eenvoudig uit de context te onderkennen of een spatie niet of wel een betekenis heeft. In het laatste geval moet bijvoorbeeld een spatie (of aantal spaties) of overgang op een nieuwe regel tussen twee acties in de ontwerptaal vervangen worden door een voor de automaat te herkennen symbool. Ook spaties voor en na een programmabeschrijving moeten vervangen worden door eenvoudig herkenbare begin- en eindsymbolen.
. mogelijkheid om verklarend commentaar aan de tekst toe te voegen door dit tussen accoladen te plaatsen.

Van een algorithme in de ontwerptaal zal verwacht worden dat die de volgende opbouw heeft:

begin "declaratie van de relevante probleemvariabelen" ;
"aantal statements dat de acties representeert"
end

waarbij de statements zijn van de soort:

- toekenning: variabele := "te berekenen waarde"
  (te lezen als: de variabele waarvan links de naam staat, krijgt de waarde van wat rechts staat, nadat dit zo nodig berekend is.)
- repetitie: while "voorwaarde" do"aantal statements"od
- alternatief: if "voorwaarde" then "aantal statements" else "aantal statements" fi
  of
  voorwaarde: if "voorwaarde" then "aantal statements" fi
- in/uitvoer:

  read(a)

  (te lezen als: de variabele a krijgt uit de invoerrij de waarde die op dat moment aan de beurt is; de invoerrij schuift één waarde op)

  write(a)

  (te lezen als: bij de uitvoerrij wordt de waarde van wat tussen haakjes staat afgedrukt)

Declaraties onderling en statements onderling worden door punt-komma's (scheidingssymbool) van elkaar gescheiden. De onderstrepingen geven weer aan dat het niet gaat om namen van variabelen, maar om (woord-)symbolen met een heel bepaalde functie. Met het hier geschetste gereedschap zullen nu voor enkele problemen algorithmen worden ontworpen. Later zullen ook opdrachten voor "ingewikkelder" akties (dan de toekenning) bekeken worden.

Opmerking. In de logica wordt bewezen dat er problemen zijn waarvoor geen algorithme is te bedenken. Voor ieder te bedenken algorithme zijn echter de bovenstaande statements voldoende; voor het gemak van de programmeur zullen later nog enkele andere (dus niet beslist noodzakelijke) statements worden ingevoerd.

4.0. Enkele algorithmen

Het voorgaande zal nu worden toegelicht aan de hand van kleine voorbeelden, alvorens in de volgende paragraaf wat nauwkeuriger op enkele begrippen in te gaan.

4.1. Voorbeeld 1: geheeltallige deling en modulobewerking

Dit probleem wordt als volgt nauwkeuriger gesteld. Bepaal van twee natuurlijke getallen, a en b genaamd, het (gehele) quotiënt, q, en de rest, r, bij deling van a door b.
De begin- en eindtoestand van het proces worden bepaald door vier variabelen: de twee in te lezen natuurlijke getallen a en b, het quotiënt, q, en de rest, r. De begintoestand is te karakteriseren door:

> a en b hebben de gegeven waarden, q en r zijn onbepaald.

De eindtoestand wordt beschreven door:

> a en b hebben de gegeven waarden, q bevat het quotiënt en r de rest bij deling van a door b.

Wanneer de operaties "geheel delen" (met als operator div en "rest bepalen" (met als operator mod) beschikbaar zijn, kan de gewenste algorithme direct als volgt worden opgeschreven:

```
begin integer a,b,q,r;
      read(a); read(b);
      q := a div b; r := a mod b;
      write(q); write(r)
end
```

Zij nu echter verondersteld dat de operaties div en mod niet als elementaire acties beschikbaar zijn, maar wel slechts de operaties optellen en aftrekken. Het bepalen van het quotiënt en de rest moet dus uitgedrukt worden in sommen en verschillen. Door intuïtie of ervaring, zoals bijvoorbeeld opgedaan in de vorige paragraaf, is dan het quotiënt q van a en b te vinden als het aantal malen dat b van a afgetrokken kan worden, zodanig dat er een rest r overblijft, die groter is dan of gelijk is aan 0 maar kleiner is dan b. Dit algorithme is als volgt vast te leggen:

```
begin integer a,b,q,r;
      read(a); read(b);
      r := a; q := 0;
      while r ≥ b do r := r - b; q := q + 1 od;
      write(q); write(r);
end
```

Voor de interpretatie van de toestanden in een proces is de betekenis van de variabelen steeds belangrijk. Uit deze betekenis volgen namelijk betrekkingen tussen de variabelen. In bovenstaand voorbeeld geldt steeds de betrekking $a = q * b + r$, die daarom ook wel de invariant van dit proces genoemd wordt. Het onderkennen van de invariant is een machtig hulpmiddel om de correctheid van een repetitie te verifiëren. Uit het feit dat de repetitie beëindigd is, volgt voorts dat aan het eind van de algorithme geldt dat $0 \leq r < b$.
De invariant maakt het veelal zelfs mogelijk om een oplossing te bedenken voor die problemen, waarvoor door intuïtie of ervaring geen oplossing bekend is. De gegeven probleemstelling brengt met zich mee dat een q en een r gevonden moeten worden waarvoor geldt

(1) $a = q * b + r$

(2) $0 \leq r < b$.

Dit stel voorwaarden kan afgezwakt worden door de tweede voorwaarde voorlopig te laten vallen. Proberenderwijs kan nu een oplossing voor (1) gezocht worden. Daarbij blijkt dat aan (1) voldaan wordt door

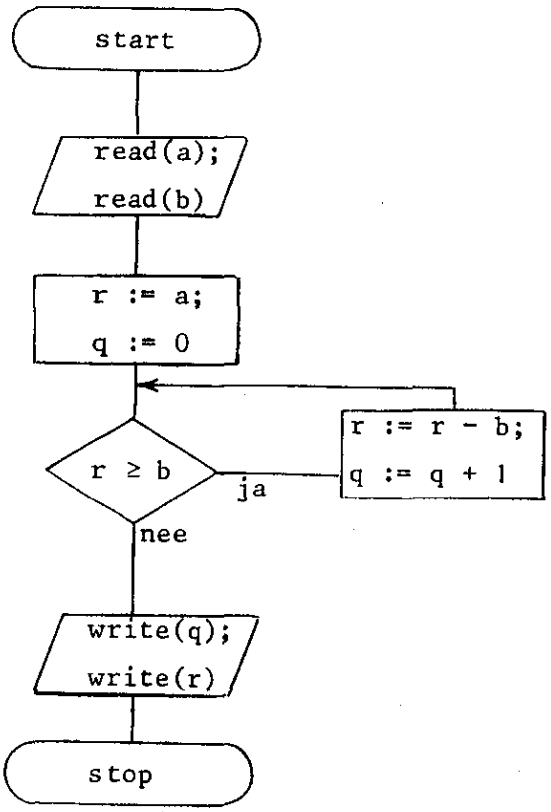
(a) $q = 0$ en $r = a$

(b) $q = 1$ en $r = a - b$

(c) $q = 2$ en $r = a - 2b$.

Aangezien de r-waarden hiermee een monotoon dalende rij vormen, zal op den duur vanzelf aan (2) worden voldaan en het geschetste proces stopt zodra $r < b$. Het proces zelf wordt gekarakteriseerd door het feit dat iedere keer dat q met 1 wordt verhoogd tevens r met b moet worden verminderd.

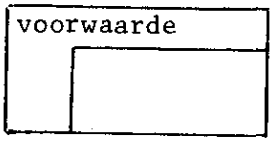
Gezien de eisen die aan onze ontwerptaal gesteld zijn, kan de algorithme nu opgeschreven worden, zoals hiervoor reeds gebeurd is. Opgemerkt kan worden dat met een andere "ontwerptaal" ook een andere schrijfwijze voor de algorithme resulteert. Een voorbeeld hiervan is de "visualisering" met de stroomschemamethode die voor bovenstaand probleem oplevert:



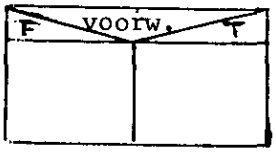
Het grootste bezwaar van deze methode is, hoewel dat voor het behandelde probleem niet naar voren komt, dat men in de verleiding komt om heel "slim" programmablokjes aan door elkaar lopende pijlen te koppelen. Van de zo gevormde 'spaghetti"-programma's is de correctheid nauwelijks te bewijzen omdat dan bij de ingang van de blokjes de begintoestand practisch niet meer vast te stellen is en daarmee niet de uitgangstoestand.

Dit bezwaar wordt ondervangen door de methode van de structuur-diagrammen die de laatste jaren meer in gebruik komt. Hierbij zijn de voornaamste diagramsymbolen de volgende:

- (een rechthoek) voor een rij toekennings - of I/O - opdrachten

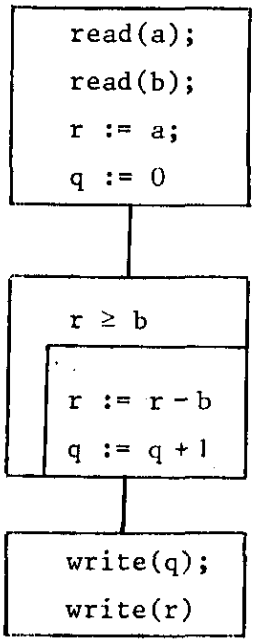


- in de onderste rechthoek komen de statements, die uitgevoerd moeten worden zolang de voorwaarde vervuld is



- in de twee onderste rechthoeken komen de statements behorende bij het <u>false</u>, resp. <u>true</u> zijn van de voorwaarde

Voor het hierboven beschouwde probleem wordt het structuurdiagram



Met de stroomschemamethode heeft de structuurdiagrammethode het bezwaar dat bij ingewikkelde grote programma's de in te vullen rechthoeken steeds kleiner worden of dat rechthoeken op vervolgtekeningen uitgewerkt moeten worden. Wegens de betrekkelijk grotere vrijheid met de aanvankelijk ingevoerde ontwerptaal voor algorithmen, zal deze verder in hoofdzaak gebruikt worden.

4.2. Voorbeeld 2: Grootste gemene deler (gerepresenteerd door de variabele ggd) van twee natuurlijke getallen a en b

De begintoestand van het proces is te karakteriseren door:

a en b hebben de gegeven waarden, ggd is onbepaald

de eindtoestand door:

a en b hebben de gegeven waarden, ggd is de grootste gemene deler.

Om een algorithme op te stellen moet nagedacht worden over de betekenis van de grootste gemene deler van de gegeven getallen a en b. Die betekenis is dat a, resp. b te schrijven is als het product van ggd en een ander geheel getal f1, resp. f2, terwijl voorts f1 en f2 geen gemeenschappelijke deler hebben (behalve de triviale 1). Dit leidt vrij natuurlijk tot het bekijken van het (absolute) verschil van a en b, dat dan te schrijven is als het product van ggd en het (absolute) verschil van f1 en f2. M.a.w. ggd is een gemene deler van a (en b) en het (absolute) verschil van a en b; omdat f1 en f2 geen factor gemeen hebben is ggd tevens de grôôtste gemene deler van het (absolute) verschil tussen a en b. Dit maakt het mogelijk om het oorspronkelijke probleem te "verkleinen" (wat de getallen betreft), omdat de gezochte ggd dus tevens de grootste gemene deler is van het (absolute) verschil van a en b en de kleinste van de gegeven getallen. Hiermee is ook de aanzet van een algorithme verkregen, omdat dit "verkleiningsproces" herhaald kan worden met de na iedere stap verkregen kleinere getallen. Het proces moet stoppen zodra het na iedere stap berekende verschil overeenstemt met het kleinste van de twee getallen (in de volgende stap is dan het verschil 0); dat is dan tevens de gezochte ggd. Alleen gebruik makend van optellingen en aftrekkingen is de algorithme dan uit te schrijven. Wanneer a en b veel in grootte verschillen, is de algorithme echter verre van efficient door het grote aantal uit te voeren stappen. Een efficienter algorithme is te verkrijgen door te bedenken dat hetgeen hierboven geconstateerd is voor het verschil van a en b ook geldt voor de rest die overblijft bij de geheeltallige deling van het grootste door het kleinste van de gegeven getallen. Deze algorithme van Euclides is al uit de oudheid bekend. In onze ontwerptaal heeft die dus de gedaante:

```
begin integer a,b,m,n;
      read(a); read(b); m := a; n := b;
      while m ≠ 0 do if m < n then "verwissel de waarden van m en n" fi;
                        m := m mod n
                  od;
      write(a); write(b); write(n)
end
```

Enig nadenken leert echter dat behalve misschien de eerste keer altijd verwisseling van m en n plaatsvindt, zodat een efficiënter maar minder doorzichtig programma gegeven wordt door:

```
begin integer a,b,m,n,r;
      read(a); read(b);
      if a < b then m := b; n := a else m := a; n := b fi; r := 1;
      while r ≠ 0 do r := m mod n; m := n; n := r od;
      write(a); write(b); write(m)
end
```

De toekenning r := constante ($\neq 0$) is nodig omdat r een waarde $\neq 0$ moet hebben om de repetitie te kunnen beginnen. De repetitie heeft als invariant dat steeds 2 getallen m en n (met m > n) worden berekend die dezelfde ggd hebben als de oorspronkelijke getallen a en b. De algorithme eindigt omdat de r-waarden een monotoon dalende rij van getallen vormen met tenslotte r = 0. Als de mod-bewerking niet als elementaire operatie beschikbaar is, zou die zoals in voorbeeld 1 uit te schrijven zijn; duidelijker wordt de algorithme daardoor zeker niet.

4.3. Voorbeeld 3: Sorteren van een rij letters

Het sorteren van een rij van n letters betekent dat deze letters (waaronder gelijke mogen voorkomen) in de volgorde moeten worden geplaatst waarin ze in het alfabet voorkomen; bijvoorbeeld

$$P,K,W,R,E,K \rightarrow E,K,K,P,R,W\ .$$

Enige tientallen algorithmen zijn bekend om dit probleem op te lossen en met enig nadenken kan men een aantal ervan zelf wel bedenken. Deze methoden maken gebruik van de elementaire actie: het van plaats verwisselen van twee letters, wanneer geconstateerd wordt dat verwisseling moet gebeuren op grond van de plaatsen van die letters in het alfabet. Van

de gezochte algorithme zal nu verlangd worden dat het aantal hulpgrootheden onafhankelijk is van n en klein blijft.
Een heel ruwe vorm van een sorteeralgorithme is:

(1) while "rij is niet gesorteerd" do "voer geschikte verwisseling uit" od.

Wat is echter een geschikte verwisseling en hoe is te constateren of een rij gesorteerd is (of niet)? Wel kan meteen bedacht worden dat in een niet-gesorteerde rij toch opeenvolgend letters voorkomen, die toevallig al in de goede volgorde staan (zoals hierboven de letters K, W en E, K).
Dit kan, denkend aan de manier waarop men in een kaartenbak een aantal kaarten (met daarop een letter) zou kunnen sorteren, leiden tot het idee om de lengte van een rij gesorteerde letters (gemakshalve wordt verder van sorteerrij gesproken) stapsgewijs met 1 te vergroten, totdat de sorteerrij alle letters bevat. De algorithme heeft dan de gedaante

(2) "maak een sorteerrij van 1 letter"
(3) while "lengte sorteerrij ≠ n"
do "verleng de sorteerrij met 1 letter (met handhaving van de sorteervolgorde)" od

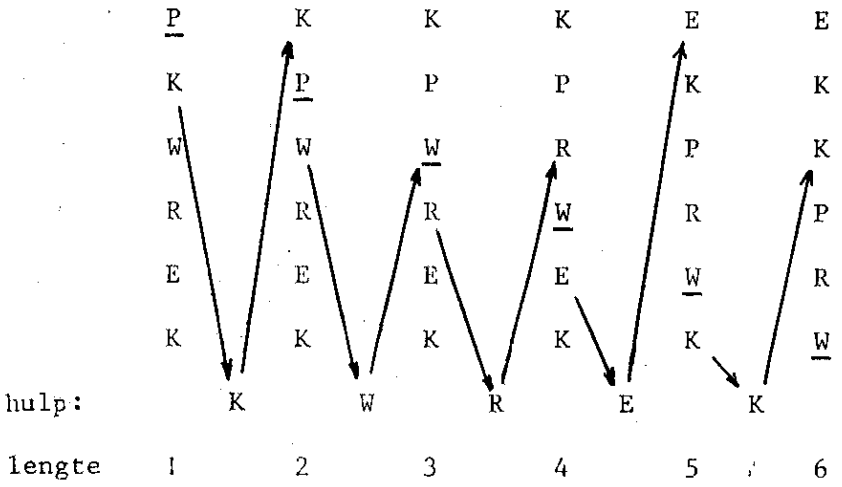
Deze algorithme is niet bruikbaar als programma omdat het computersysteem deze opdrachten niet kent. We zullen de acties dus moeten uitschrijven als een proces. Ook in de verdere ontwikkeling van de algorithme zal deze wijze van werken, met steeds grotere detaillering, gebruikt worden.
Stap 2 is niet moeilijk omdat iedere sorteerrij van 1 letter gesorteerd is; de vraag blijft met welke letter van de oorspronkelijke rij begonnen zal worden. Ook de voorwaarde (3) zal geen zorgen baren als bij uitvoering van (4) maar tevens de lengte van de sorteerrij met 1 opgehoogd wordt.
T.a.v. (4) is duidelijk, dat de toe te voegen letter ontleend moet worden aan die letters uit de oorspronkelijk rij die nog niet een plaats in de sorteerrij hebben gevonden. Een open vraag is nog welk van de in aanmerking komende letters gekozen moet worden. Als keus van de toe te voegen letter wordt genomen de nog niet gesorteerde letter direct volgend op (resp. voorafgaande aan) de reeds gevormde sorteerrij. Voor de handhaving van de sorteervolgorde moet de toe te voegen letter op de goede plaats gebracht worden in een dank zij de vorige stappen reeds goede (maar nog te korte) sorteerrij. Dit betekent, tenzij de toe te voegen letter toevallig achteraan de oude sorteerrij geplaatst kan worden, dat enkele "goede" letters van de oude sorteerrij één plaats moeten opschuiven.

Wordt voor de verdere concretisering aangenomen dat de sorteerrrij begint met de eerste letter, dan krijgt de algorithme de gedaante:

```
"maak een sorteerrij bestaande uit de eerste letter"; lengte := 1;
while lengte ≠ n
      do "voeg letter direct volgend op sorteerrij toe aan de sorteerrij
         met handhaving van sorteervolgorde";
         lengte := lengte + 1
      od
```

Een deelprobleem is nu nog alleen de eis "met handhaving van sorteervolgorde", wanneer de toe te voegen letter niet toevallig lexicografisch volgt na de laatste letter van sorteerrij. Wanneer dat echter niet het geval is, moet op de plaats van de toe te voegen letter de laatste letter van de reeds gevormde sortering terecht komen. Men kan dan denken aan het verwisselen van deze twee letters. Aangezien als regel nog meer letters uit de oude sorteerrij één plaats moeten opschuiven, is het eenvoudiger om de toe te voegen letter meteen opzij te zetten en dan de nodige letters uit sorteerrij één plaats op te schuiven, achteraan beginnend. Wat nodig is volgt dan uit de vergelijking met de opzij gezette letter.

Het tot dusver beschreven proces is als volgt gemakkelijk toe te lichten voor het gegeven voorbeeld:



De algorithme is nu in meer detail te formuleren:

```
"maak een sorteerrij bestaande uit de eerste letter"; lengte := 1;
while lengte ≠ n
    do "zet letter in de positie (lengte + 1) op de plaats hulp";
       "zet een vinger bij de letter in de positie lengte";
       while "door vinger aangewezen letter in het alfabet volgt op
              de letter op de plaats hulp en vinger wijst naar positie
              in sorteerrij"
              do "schuif de door vinger aangewezen letter één plaats op";
                 "verplaats vinger één plaats in de richting van het begin
                 van sorteerrij"
              od;
       "zet letter op de plaats hulp op de door (vinger + 1) bepaalde
       plaats in de sorteerrij";
       lengte := lengte + 1
    od
```

Voor de binnenste repetitie luidt de invariant: "op de plaatsen (vinger + 1) tot en met (lengte + 1) staan letters die allen volgen op de letter op de plaats hulp". Na afloop van deze repetitie geldt dat de door vinger aangewezen letter niet volgt op de letter op de plaats hulp of dat vinger niet meer wijst naar een positie in sorteerrij. (Merk op dat de binnenste repetitie beheerst wordt door twee voorwaarden, die allebei vervuld moeten zijn wil de repetitie niet beëindigd worden.) Door de op de binnenste repetitie volgende actie wordt bereikt dat voor de buitenste repetitie de invariant luidt dat in sorteerrij de letters in de goede volgorde staan en lengte het aantal gesorteerde elementen aangeeft.

Voor het vergelijken van twee letters is een lexicografische ordening nodig of moeten met letters getalwaarden worden geassocieerd zodat dan numerieke vergelijkingsoperatoren gebruikt kunnen worden. Verder is duidelijk dat er behoefte is aan een structuur om daarin rijen letters (i.h.a. gegevens) op te slaan en bepaalde posities in die structuur aan te wijzen.
Uit de wiskunde zijn dergelijke structuren bekend, nl. vectoren en matrices.

Dit zijn verzamelingen elementen, waarbij de verzameling een naam heeft en de elementen benoemd worden door achter die naam één of meer indices te plaatsen, bijv. v_i of m_{ik}. Blijkens het laatste voorbeeld is ook in de ontwerptaal dus behoefte aan een equivalent van vectoren of matrices, die arrays genoemd zullen worden. In de ontwerptaal zullen indices echter niet "half onder de regel" geschreven worden (wegens de problemen om die duidelijk te kunnen lezen), maar tussen rechte haken geplaatst achter de naam van het array; dus v[i] en m[i,k]. De "selectoren" i, resp. i,k wijzen dus een element in de verzameling v, resp. m aan. Naast de tot dusver beschouwde eenvoudige (of scalaire) variabelen die op ieder moment slechts één waarde kunnen aannemen, zijn hier dus op informele wijze de gestructureerde variabelen, zoals v en m ingevoerd. Zulke arrays moeten evenals de eenvoudige variabelen in het begin van een algorithme gedeclareerd worden met

type array identifier[ondergrens: bovengrens, ... , ...]

waarin voor type ingevuld moet worden integer, real, boolean of character overeenkomstig het type van de array componenten. Het aantal keren dat het paar ondergrens : bovengrens herhaald wordt, correspondeert met het aantal indices (voorts geldt dat bovengrens ≥ ondergrens).

Voor de verdere uitwerking van het sorteerprobleem worden nu de volgende grootheden ingevoerd, aannemend dat 100 letters gesorteerd moeten worden:

rij[1 : 100] - een character array om 100 in te lezen characters op te slaan,
lengte - een wijzer naar de laatste ingevulde plaats in het array, resp. de laatste plaats in het gesorteerde deel van het array,
vinger - een wijzer naar het array-element in het reeds gesorteerde deel dat mogelijk voor opschuiving in aanmerking komt,
hulp - een tijdelijke bergplaats voor het in het gesorteerde deel van rij in te voegen element,
schuif - een boolean variabele met de waarde true als een opschuiving moet plaats vinden.

```
begin character array rij[1 : 100]; character hulp;
      integer lengte, vinger; boolean schuif;
      lengte := 0; while lengte < 100 do lengte := lengte + 1; read(rij[lengte])od;
      lengte := 1;
      while lengte < 100
            do hulp := rij[lengte + 1]; vinger := lengte;
               schuif := rij[vinger] > hulp;
               while schuif = true
                     do rij[vinger + 1] := rij[vinger];
                        vinger := vinger - 1;
                        if vinger < 1 then schuif := false
                                      else schuif := rij[vinger] > hulp fi
                     od;
               rij[vinger + 1] := hulp; lengte := lengte + 1
            od
end
```

De in de twee laatste voorbeelden aangegeven wijze om een algorithme op te stellen wordt wel genoemd de stapsgewijze verfijning.

Opmerking. Alvorens dit sorteervoorbeeld te verlaten is het nuttig om nog te bekijken hoeveel bewerkingen uitgevoerd moeten worden voor een rij van n letters. Die bewerkingen zijn het vergelijken van twee letters en het verschuiven van een letter. De aantallen van elk van deze bewerkingen zijn vrijwel even groot, zodat alleen het aantal vergelijkingen bekeken hoeft te worden. Als op een gegeven ogenblik de sorteerrij m elementen bevat ($0 \leq m < n$), dan zijn $\frac{1}{2}(m + 1)$ vergelijkingen gemiddeld nodig om de $(m + 1)$ste letter op zijn plaats te krijgen (iedere positie is even waarschijnlijk). Het totaal aantal vergelijkingen is dan

$$\tfrac{1}{2}(2 + 3 + 4 + \ldots + n) = \tfrac{1}{4}(n - 1)(n + 2) \simeq \tfrac{1}{4}n^2 .$$

Voor kleine n-waarden (tot ca. 20) wint deze algorithme het door zijn eenvoud van meer gecompliceerde sorteermethoden, waarbij het aantal vergelijkingen evenredig is met n log n (en die dus voor grote n-waarden te prefereren zijn).

4.4. Representatie van waarden en algorithmen

Zoals in alle kennisgebieden moet men onderscheid maken tussen begrippen, die vastgelegd worden met axioma's en definities, en hun af te spreken voorstelling of representatie op bijvoorbeeld een "medium" als papier. Zo is een differentiaalquotient gedefinieerd als de limiet van een differentiequotient, terwijl het als regel voorgesteld wordt met bijvoorbeeld $\frac{dx}{dt}$ of $\dot{x}$ of x_t (de laatste representatie heeft het bezwaar dat t ook opgevat kan worden als een index).
Een ander voorbeeld: het gehele getal dertien is gedefinieerd als de opvolger van twaalf, doch kan op papier gerepresenteerd worden met:

13	in het tientallig getalstelsel
15	in het achttallig (of octale) getalstelsel
1101	in het tweetallig (of binaire) getalstelsel
D	in het hexadecimale (of zestientallige getalstelsel : A = tien,...,F = vijftien)
XIII	met Romeinse cijfers.

Men spreekt in al deze gevallen van een digitale representatie omdat het getal wordt voorgesteld met discrete symbolen. Van een analoge representatie spreekt men als bijvoorbeeld afgesproken wordt dat het getal 13 gerepresenteerd wordt door een lijnstuk met een lengte van dertien cm. (dat dit onhandig en niet erg nauwkeurig is, wordt verder in het midden gelaten).
Een voorbeeld van een analoge en discrete representatie met een ander medium dan papier wordt gegeven door de vastlegging van een tijdstip; met wijzers en een wijzerplaat heeft men een analoge representatie, maar tegenwoordig heeft men daarnaast de z.g. "digitale uurwerken", waarbij cijfersymbolen op een schermpje verschijnen.

Bij het werken met rekenautomaten heeft men met veel media te maken. Op (programmeer-) papier wordt een getal "gewoon" (in het tientallige stelsel) gerepresenteerd, binnen de rekenautomaat binair (veelal magnetisch), op een ponsband of ponskaart met bepaalde gatencombinaties (men spreekt van "codes", waarvan een aantal internationaal gestandaardiseerd zijn.
Een algorithme kan men op papier voorstellen met de stroomschemamethode of met structuurdiagrammen of met de bij voorkeur te gebruiken algorithmen ontwerptaal. In deze ontwerptaal fungeert volgens afspraak od als "afsluiter" bij do en evenzo fi als "afsluiter" bij if, d.w.z. wat tussen do en od (dan wel tussen if en fi) staat, moet als een eenheid (samengestelde actie) op-

gevat worden. In de (later te behandelen) programmeertaal Algol 60 zijn echter de afsluiters od en fi niet ingevoerd, maar moet men een samengestelde actie omsluiten met begin ... end, waarmee ook een programma omsloten wordt (wanneer de actie enkelvoudig is, mag men begin ... end achterwege laten).
Voor de representatie van symbolen op een ponskaart of ponsband worden (omdat onderstreping daar niet mogelijk is) meestal afspraken gemaakt van het type: pons begin als 'BEGIN' of als "BEGIN" of misschien als BEGIN (het laatste heeft het bezwaar dat BEGIN dan een "reserved word" is, d.w.z. nooit als naam van een variabele gebruikt mag worden!).
Omdat al deze representatie-kwesties maar details zijn die toch niet bijdragen tot de begrippen, zal er verder weinig aandacht aan geschonken worden. Voor het aanbieden van een programma voor verwerking door een rekenautomaat zijn deze details echter essentieel omdat de automaat geen begrip verlangt (of opbrengt), maar een zich precies houden aan bepaalde afspraken.

5. Constanten, variabelen en expressies

5.1. Typen van waarden

Zoals reeds uit de tot dusver behandelde voorbeelden blijkt, wordt een algorithme of een programma(-ontwerp) opgebouwd met declaraties en opdrachten. Op het repertoire van uit te voeren handelingen of acties, beschreven met behulp van deze opdrachten of statements, zoals de toekenning, de while - do en de if - then - else constructie wordt in de volgende paragraaf teruggekomen.

Hier wordt nader ingegaan op de grootheden, waarvan de waarden onderworpen worden aan of deel uitmaken van de genoemde statements. Een waarde kan zijn een constante of de waarde van een variabele of algemener de waarde van een expressie. Een waarde is van een bepaald type, d.w.z. is een element van een bepaalde waardenverzameling. Van welk type een constante is, blijkt uit de (af te spreken) schrijfwijze van deze constante. Van welk type een variabele is moet daarentegen aangegeven worden in een declaratie, waarin naam en soort van de variabele worden vermeld (en waardoor in het geheugen de nodige plaatsen worden gereserveerd). Het type van een expressie wordt later besproken.
Waarden van gedeclareerde variabelen zijn ongedefinieerd totdat in een algorithme er waarden aan zijn toegekend. De waarde van een expressie is ongedefinieerd als er "ongedefinieerde" variabelen in voorkomen.

Waarden en variabelen kunnen zijn van het type:

- integer — als ze horen tot Z^*, een eindige aaneengesloten deelverzameling van de verzameling, Z, van de gehele getallen,
- real — als ze horen tot Q^*, een eindige deelverzameling der rationale getallen Q,
- complex — als ze horen tot C^*, een eindige deelverzameling der complexe getallen C.
  In deze drie gevallen zijn onder- en bovengrens van de deelverzamelingen voor elk computertype gegeven.
- boolean — als ze slechts de logische of boolean waarden false en true kunnen aannemen. Deze logische waarden zeggen bijv. of zekere uitspraken of beweringen wel of niet waar zijn,
- character — als ze een waarde kunnen aannemen uit een alfabet, bestaande uit een geordende verzameling van letters, cijfers, leestekens

en andere symbolen,

- string als ze als waarde teksten bestaande uit een zeker aantal karakters kunnen aannemen.

In programmeertalen komen in het algemeen wel altijd de typen integer, real en boolean voor; welke van de andere typen voorkomen hangt van de taal af.

Een expressie is een constructie om een nieuwe waarde te berekenen. De eenvoudigste vorm van een expressie is een enkele constante of variabele, doch in het algemeen is een expressie opgebouwd uit constanten, variabelen (functies), ronde haakjes en operatoren. De details hiervan worden in het volgende uiteengezet.

5.2. Arithmetische expressies

Een opgave van arithmetische operatoren en het type van het resultaat van toepassing ervan op operatoren is in bijgaande tabel opgenomen. Hierin zijn de volgende afkortingen gebruikt: i voor integer operand, r voor real operand, c voor een complexe operand en a voor een willekeurige van deze drie operanden en tenslotte ≠ voor ongedefinieerd.

operatie	notatie	operand(-paar)	type resultaat
tekeninversie	-	a	a
optelling	+	ii	i
aftrekking	-	ir,ri,rr	r
vermenigvuldiging	*	ac	c
deling	/	aa	if a ≠ c then r else c fi ≠ als deler = 0
gehèlèndeling	div of //	aa	if a = i then i else ≠ fi ≠ als deler = 0
restbepaling	mod		
machtsverheffing	↑ of **	a^i i > 0	a
		a^i i = 0	a
		a^i i < 0	if a ≠ c then r else c fi ≠ als a = 0
		a^r a > 0	if a ≠ c then r else c fi
		a^r a = 0	if a ≠ c then r else c fi ≠ als r ≤ 0
		a^r a < 0	≠
		a^c	≠

Hierbij moet nog het volgende worden opgemerkt:

- men zij er op bedacht dat een real slechts een rationale benadering is voor een reële variabele, zodat bewerkingen met reals (afrondings-) verschillen kunnen vertonen met de corresponderende bewerkingen voor reële variabelen. Deze problematiek wordt in de numerieke wiskunde in detail bekeken.
- bij uitvoering van een arithmetische operatie kàn het resultaat buiten de grenzen van het betreffende type (dus buiten Z^*, Q^* of C^*) komen te liggen. De rekenautomaat zal dan als regel een "overflow" boodschap produceren.
- voor de gehelendeling en restbepaling (die zoals aangegeven alleen voor integers gedefinieerd zijn) geldt:

  $$m \underline{\text{div}}\ n = \text{sign}(m * n) * q$$

  $$m \underline{\text{mod}}\ n = \text{sign}(m * n) * r$$

  waarin

  $$|m| = q * |n| + r$$

  met

  $$0 \leq r < |n|$$

  en

  $$n \neq 0.$$

- niet in alle programmeertalen zijn de drie arithmetische typen en hun combinaties toegestaan,
- het type van een expressie wordt vastgesteld door toepassing van de regels in de tabel.

Bij de uitwerking van een expressie (d.w.z. de berekening van de waarde) moet rekening worden gehouden met de prioriteitsregels. Deze houden in dat ronde haakjes de hoogste prioriteit hebben (d.w.z. hetgeen tussen een openingshaak en bijbehorende sluithaak staat moet eerst worden berekend), terwijl de prioriteitsvolgorde verder is:

. machtsverheffing, vermenigvuldiging of deling, optelling of aftrekking
. van links naar rechts bij gelijke prioriteit (zoals ook gebruikelijk in de normale wiskundige notatie met uitzondering van de machtsverheffing).

Verifieer de volgende weergave van een aantal expressies

normale schrijfwijze	correcte weergave	incorrecte weergave
$2n - 1$	2*n - 1	2n - 1
$(a + b)(c + d)$	(a + b)*(c + d)	(a + b)(c + d)
$\frac{1}{1 - a}$	1/(1 - a)	1/1 - a
$(2^3)^4$	(2 ↑ 3) ↑ 4	2 ↑ (3 ↑ 4)
2^{3^4}	2 ↑ (3 ↑ 4)	2 ↑ 3 ↑ 4
$p \cdot \frac{-1}{qr}$	p * (-1)/(q*r)	p * -1/q * r
$a^{(2p)}$	a ↑ (2 * p)	a ↑ 2 * p

5.3. Boolean expressies

Op gelijksoortige wijze als bij arithmetische expressies kan men met behulp van boolean operanden, boolean operatoren en ronde haakjes boolean expressies opbouwen. Een opgave van in de meeste programmeertalen voorkomende boolean operatoren en het resultaat ervan voor verschillende typen operanden is in bijgaande tabel opgenomen.

operatie	notatie		a = true	true	false	false
	wisk.	algor.	b = true	false	true	false
negatie	$\neg a$	not a	false	false	true	true
conjunctie	$a \wedge b$	a and b	true	false	false	false
disjunctie	$a \vee b$	a or b	true	true	true	false

Boolean operanden kunnen zijn:

- de boolean constanten true of false,
- boolean variabelen (daartoe als zodanig gedeclareerd) en (de later te behandelen) boolean functies,
- condities ("relations"), in het algemeen bestaande uit twee characters of arithmetische expressies gescheiden door een van de conditie-operatoren = en ≠ (die voor alle typen expressies gebruikt mogen worden) en <, ≤, >, ≥ (die uiteraard niet bij complexe expressies gebruikt mogen worden).

Enkele voorbeelden van condities en boolean expressies zijn:

conditie	waarde
x = x	true
x = x + 1	false
x ≤ x + 1	true
(x + 3) * (x - 2) ≥ 0	false als -3 < x < 2, anders true
a + b > c - 4	hangt af van de waarden van a, b en c

expressie	waarde
x = x and y = y	true
(x + 3)*(x - 2) ≥ 0 and x = 0	false
(x + 3)*(x - 2) ≥ 0 or x < 2	true

Net zoals bij arithmetische expressies is er een prioriteitsvolgorde voor de uitwerking van boolean operatoren, nl. eerst haakjes, dan (in de meeste talen) arithmetische expressies, not, and en or. Het gebruik van i.v.m. deze prioriteitsregel strikt genomen overbodige haakjes moet echter niet nagelaten worden als dit de leesbaarheid bevordert. Bij gelijke prioriteit moet een expressie weer van links naar rechts worden uitgewerkt. Voorbeelden:

boolean expressie	interpretatie
a > b or c < d	(a > b) or (c < d)
a + b > -5 and c - d < e ↑ 2	((a + b) > -5) and ((c - d) < (e ↑ 2))
a and b or c ≠ d	(a and b) or (c ≠ d)

Opmerking

Een volledige tabel van binaire boolean operatoren (in prioriteitsvolgorde; f = false, t = true) is:

operatie	notatie			a : f f t t
	algor.	wiskunde	"omgangstaal"	b : f t f t
conjunctie	a and b	a ∧ b	a èn b	f f f t
disjunctie	a or b	a ∨ b	a of b	f t t t
implicatie	a imp b	a → b	als a dan b	t t f t
equivalentie	a eqv b	a ≡ b	b alleen dan als a	t f f t
differentie		a \ b	a wel, b niet	f f t f
antivalentie		a eor b	òf a, òf b	f t t f
Peircefunctie		a nor b	noch a, noch b	t f f f
Schefferfunctie		a nand b	a niet of b niet	t t t f

Zoals door uitschrijven blijkt, is met de not operator en de eerste twee operatoren de werking van de andere operatoren te representeren. Bijvoorbeeld:

a eqv b door (a and b) or (not a and not b)

a imp b door not a or b

De eerste 4 operatoren komen in verschillende programmeertalen voor. De laatste 2 operatoren hebben de bijzondere eigenschap dat, met ieder van hen alleen, de andere operatoren te representeren zijn.(De nor and nand operatoren worden dan ook gebruikt in elektronische schakelingen, o.a. in een rekenautomaat). Bijvoorbeeld:

not a door a nor a

a and b door (a nor a) nor (b nor b)

a or b door (a nor b) nor (a nor b)

Let tenslotte op het verschil tussen de ("niet - uitsluitende") or en de ("wel - uitsluitende") eor. Vergelijk hiermee het gebruik van het woord "of" in de volgende zinnen:
"Een paspoort of een rijbewijs is nodig om zich te legitimeren".
"De eerste kerstdag valt op dinsdag of woensdag".

5.4. Functies

Zowel bij arithmetische als boolean expressies zijn als operand ook corresponderende typen "functies" te gebruiken die pas later behandeld zullen worden. Het is voorlopig voldoende zo'n operand te omschrijven als de waarde van een functie voor de argumentwaarde, die tussen (ronde) haakjes achter de functienaam geplaatst wordt. Voorbeelden zijn de volgende "standaardfuncties" (ae = arithmetische expressie; ae' = waarde van ae)

abs(ae)	: de absolute waarde van ae'	type real
sqrt(ae)	: de vierkantswortel uit ae'	
exp(ae)	: de exponentiële functie van ae'	
ln(ae)	: de natuurlijke logarithme van ae'	
sin(ae)	: de sinus van ae' (in radialen)	
cos(ae)	: de cosinus van ae' (in radialen)	
arctan(ae)	: de arctan van ae' (hoofdwaarde)	
sign(ae)	: +1 als ae' > 0, 0 als ae' = 0 en -1 als ae' < 0	type integer
entier(ae)	: de grootste integer niet groter dan ae'	

5.5. Enkelvoudige en gestructureerde variabelen

Variabelen, die in een bepaalde toestand slechts één waarde bezitten, noemt men enkelvoudige of scalaire variabelen. Daarentegen zijn gestructureerde of samengestelde variabelen gekenmerkt door het feit dat hun waarden bestaan uit een samenstel van logisch bij elkaar horende "componentwaarden". In principe kan iedere componentwaarde zelf weer een samenstel van waarden zijn. De manier waarop de componenten van een samengestelde variabele met elkaar samenhangen, heet de structuur van de variabele. Zo kan bijvoorbeeld een punt in de driedimensionale ruimte gekarakteriseerd worden door een variabele, te noemen punt, met drie reële componenten (de coördinaten van dat punt).

Een andere samengestelde variabele, aangegeven met de naam "persoon" heeft bijvoorbeeld de componenten: naam, geboorteplaats, geboortedatum. Deze laatste component is zelf weer samengesteld uit de componenten: jaar, maand, dag.

5.5.1. Array

Een array is zo'n gestructureerde variabele, gekarakteriseerd door de volgende eisen:

- iedere component is enkelvoudig
- alle componenten zijn van hetzelfde type
- het aantal componenten is constant
- een component wordt geselecteerd met behulp van een rij geheeltallige indices (het aantal indices heet de dimensie van het array).

Het type van de componenten en de onder- en bovengrens voor iedere index moet bij de declaratie van een array opgegeven worden; bijvoorbeeld in uitbreiding aan de declaratie van scalaire variabelen met:

```
integer array index, jan[0:10], n[-1:0];
boolean array klaar[-10:2,5:9];
real array piet[0:1,1:2,5:8];
```

Met de eerste declaratie worden drie 1-dimensionale arrays met resp. 11, 11 en 2 integer componenten gedeclareerd. De eerste twee arrays zullen geen indexwaarden anders dan 0 t/m 10 kennen, array jan kent slechts -1 en 0. Met de tweede declaratie wordt een 2-dimensionaal boolean array gedeclareerd, waarbij de eerste index kan lopen van -10 t/m 2 en de tweede index van 5 t/m 9 (naar analogie met matrices zou men van rij en kolomindex kunnen spreken). Array klaar bevat dus 13 * 5 = 65 componenten. Met de derde declaratie wordt een 3-dimensionaal array gedeclareerd met real componenten en de opgegeven onder- en bovengrens voor ieder van de indices. Voor elke dimensieaanduiding geldt dat de bovengrens nóóit kleiner mag zijn dan de ondergrens. (Op de plaats van de grenzen mag een arithmetische expressie

staan, wanneer de in de expressie staande variabelen tenminste gedefinieerd zijn, d.w.z. op het moment van de declaratie een waarde bezitten.)

De declaratie van een enkelvoudige of een samengestelde variabele betekent niet een waarde-toekenning aan de variabele of de componenten (ook niet de waarde nul)! De waarde van (elke component van) een array-variabele is ongedefinieerd totdat een waarde is toegekend. De eerste keer dat dit gebeurt heet een initialisatie; deze moèt plaats hebben gevonden voordat een expressie met zo'n variabele of component geëvalueerd wordt (anders is ook de expressie ongedefinieerd).

Wanneer een bepaalde array-component als operand in een expressie nodig is, wordt dit aangegeven met de arraynaam gevolgd door, weer tussen rechte haken, een aantal indexwaarden, gelijk aan de dimensie van het array. Uiteraard moeten de indexwaarden binnen de gedeclareerde grenzen liggen (anders wordt gerefereerd aan een niet bestaande component). Bij het selecteren van een bepaalde component mogen tussen de rechte haken arithmetische expressies staan op de plaats van de indexwaarden (bij evaluatie van die expressies mogen hun (afgeronde) waarden weer niet buiten de declaratiegrenzen liggen). Voorbeelden van correct en incorrect gebruik van array-componenten op basis van de voorgaande declaraties zijn:

correct	incorrect
jan[7]	jan[-1]
n[0]	n[1]
klaar[0,6]	klaar[0]
piet[0,2,7]	piet[0,3,8]
	piet[0,2]
---	---
---	---
a := 1;	a := 2;
q := piet[a,2*a,6*a]	q := piet[a-1,a,a**2]

5.5.2. Record en file

Een record is een andere gestructureerde variabele, gekenmerkt door de volgende eisen:

- componenten mogen van verschillend type zijn en mogen zelf ook weer samengesteld zijn,
- een component wordt geselecteerd met behulp van de unieke nããm van iedere component (in tegenstelling tot arrays kunnen dus geen componenten geselecteerd worden met behulp van expressies).

Nog weer een andere gestructureerde variabele is de (sequentiële) file, gekenmerkt door:

- alle componenten zijn van hetzelfde type (mogen bijvoorbeeld scalaire variabelen, arrays of records zijn).
- het aantal componenten is niet constant.
- selectie is slechts mogelijk van de eerste en van telkens de volgende component; bovendien is er een mogelijkheid om het einde van een file (eof) te constateren.

De enige bewerkingsmogelijkheid van een file correspondeert met de werking van mechanische lees- en schrijfoperaties bij input/output apparatuur en achtergrondgeheugens.

6. Opdrachten en besturingsstructuren

Na de bespreking in de vorige paragraaf van de grootheden die onderworpen worden aan bewerkingen, zal nu nader bekeken worden welke opdrachten voor die bewerkingen te onderscheiden zijn, verdeeld in:

- enkelvoudige opdrachten, waarmee aan een of een aantal variabelen een waarde wordt gegeven. Voorbeelden hiervan zijn de assignment statement en bepaalde procedure statements; de laatste komen pas later aan de orde. Deze opdrachten worden als regel uitgevoerd in de volgorde waarin de statements staan.
- samengestelde opdrachten voor o.a. het herhaald of het selectief uitvoeren van statements. Men spreekt dan ook wel van "sequentiëringsstatements" of van "control structures". Eigenlijk is het feit dat na elkaar geschreven statements (concatenatie) als regel na elkaar uitgevoerd worden ook al een control structure.

6.1. Assignment statement

Ten aanzien van de ("enkelvoudige") assignment statement

variable := expressie (of korter V := EX)

is op te merken dat linker- en rechterlid beide óf van het type boolean óf character óf arithmetisch moeten zijn. Bij arithmetische typen is het toegestaan dat V en EX niet beiden real of integer zijn. Als EX van het type integer is en V van het type real, dan wordt de waarde van EX als getal van het type real aan V toegekend. Is daarentegen V van het type integer en EX van het type real, dat wordt de waarde van EX afgerond op het dichtstbijzijnde gehele getal (uit Z^*), dat dan als integer aan V wordt toegekend.

Wanneer aan een aantal variabelen (allen van hètzelfde type!) eenzelfde waarde moet worden toegekend, dan kan dit met behulp van de zogenoemde "samengestelde" assignment. Voorbeeld:

x := y := z := expressie

Aangezien onder deze variabelen ook geïndiceerde variabelen kunnen voorkomen, geldt de volgende afspraak ten aanzien van de betekenis van deze constructie:

. bepaal van links naar rechts gaande de identiteit van de variabelen,
. bepaal de waarde van de expressie,
. ken deze waarde toe aan de geïdentificeerde variabelen.

Ga aan de hand hiervan na dat

i := 3; i := a[i] := i + 1;

ten gevolge heeft dat a[3] en i beiden 4 worden.

6.1.1. De dummy statement is een lege rij symbolen, zoals bijvoorbeeld tussen twee opeenvolgende punt-komma's of tussen een punt-komma en een daarop volgend end-symbool. Dummy statements hebben geen effect bij de verwerking van een programma en zijn als regel zinloos.

6.2. Herhaalde uitvoering van statements

Voor de while-do statement en de hierna te behandelen do-until statement treft men in programmeertalen alternatieve schrijfwijzen aan, bijvoorbeeld voor de laatste repeat-until. Voor het begrip van deze constructies is dit natuurlijk niet essentieel. Voorts wordt hier de vaak handige for statement behandeld.
Zoals reeds is gebleken, is de while statement uitermate geschikt voor cycli waarbij tijdens het schrijven nog niet vaststaat óf, resp. hóé vaak een aantal statements uitgevoerd, resp. herhaald moet worden. In die gevallen waarbij door de aard van het probleem het in ieder geval zeker is dat een aantal statements tenminste 1 keer moet worden uitgevoerd, is een do-until statement

```
do S until B
```

vaak eenvoudig in het gebruik. De interpretatie hiervan is dat de met S aangegeven groep statements moet worden uitgevoerd; wanneer dan de boolean expressie B false is wordt dit herhaald (dit betekent natuurlijk dat S moet bewerkstelligen dat B een keer true wordt). Een voorbeeld voor het gebruik van dit statement is de nulpuntsbenadering van een functie met de methode van Newton:

```
x1 := start;
do x0 := x1; x1 := x0 - f(x0)/f'(x0)
   until abs(x1 - x0) < eps
```

Bij de toepassingen van de while statement zal opgevallen zijn dat deze vaak het volgende karakter hebben

```
i := 0; while i ≤ n do S; i := i + 1 od
```

Dit is te vervangen door een enkel statement

```
for i := 0 step 1 until n do S od
```

In het algemeen geldt dat de for statement

```
for i := p step q until r do S od
```

waarin i een (integer) variabele is en p,q en r arithmetische expressies (van het type integer) zijn, hetzelfde effect heeft als

```
i := p; while (r - i) * q ≥ 0 do S; i := i + q od
```

De waarden van q en r worden na iedere uitvoering van het do-gedeelte opnieuw berekend! (Het is echter niet verstandig om ingewikkelde q en r expressies - bijv. afhangend van hetgeen in het do-gedeelte gebeurt - te gebruiken, aangezien de algorithmen daardoor vrijwel ondoorgrondelijk worden.)

Een voorbeeld voor het gebruik van dit for statement is de berekening van het product C van een m × n matrix A en een n × p matrix B met

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

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

Een andere vorm van een for statement is die waarbij voor een aantal (niet noodzakelijk equidistante) waarden de berekening van S moet worden uitgevoerd; bijv.

```
for i := ex1, ex2,..., exn do S od
```

waarin i, ex1,...,exn een variabele, resp. expressies van het type integer zijn.

6.3. Selectieve uitvoering van statements

De alternatieve statement

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

geeft een middel om afhankelijk van de waarde van de boolean expressie B

statements S1 of S2 uit te voeren.
In programmeerproblemen komt echter ook vaak een situatie voor als in onderstaand voorbeeld:

als de waarde van een zekere integer variable (of expressie) A
3 is voer dan statement S1 uit,
4 is voer dan statement S2 uit,
15 is voer dan statement S3 uit,
een ander dan de genoemde is, doe dan niets.

Met een keten van alternatieve statements is dit natuurlijk uit te schrijven:

```
if A = 3 then S1
        else if A = 4 then S2
                      else if A = 15 then S3
                           fi
                fi
fi
```

Vooral als het aantal alternatieven groot is, zijn er problemen met het overzichtelijk opschrijven van een algorithme. Om die reden schrijft men daarom bij voorkeur het volgende op, gebruik makend van het taalelement case:

```
case A of
     3 : S1;
     4 : S2;
    15 : S3
esac
```

Zoals bij de eerste schrijfwijze staan S1, S2 en S3 voor een enkel statement of een rij statements en afhankelijk van het resultaat van de evaluatie van de arithmetische (integer) expressie A, wordt hetzij S1, hetzij S2, hetzij S3 uitgevoerd of wordt de case-constructie als een dummy statement opgevat als het resultaat van de evaluatie niet 3, 4 of 15 is. De integers 3, 4 en 15, door een dubbele punt van de statements S1, S2 en S3 gescheiden, worden "labels" genoemd. Binnen een case-statement moeten deze labels uiteraard uniek zijn. (In de literatuur treft men ook enigszins afwijkende definities en benamingen aan. Zo spreekt men ook wel van een "arithmetische" if-constructie en noemt men de gewone if- constructie de "logische" if.

Opmerking. Een goed gebruik van de bladspiegel bevordert zoals in het voorgaande aangegeven is, de leesbaarheid van algorithmen. Ter wille van verticale ruimtebesparing schrijve men

```
while ... do ... od
if ... then ... else ... fi
```

als op de plaats van de puntjes korte teksten staan. Bij lange teksten schrijve men echter

```
while ...          en        if ...
    do ...                       then ...
       ...                            ...
    od                           else ...
                                      ...
                               fi
```

(bij elkaar horende do - od, if - fi en then - else onder elkaar schrijvend).

7. Opstellen van algorithmen

In de volgende paragrafen worden voor een aantal, meestal vrij eenvoudige problemen algorithmen opgesteld. De problemen zijn in drie groepen verzameld. In de eerste groep wordt vrijwel niet "gerekend", doch worden gegeven rijen getallen geïnspecteerd, waarna met sommige getallen iets gedaan wordt. De tweede groep bevat problemen, waarbij meer gerekend wordt. In de derde groep zitten enkele meer gecompliceerde problemen. Bij de eenvoudige problemen is langs intuïtieve weg de algorithme opgesteld; bij de ingewikkelder problemen wordt een formeler beschouwingswijze toegevoegd.

7.1. "Inspectieproblemen"

7.1.1. Gegeven is een rij van 100 integers. Gevraagd wordt het rangnummer van het eerste getal uit die rij dat gelijk is aan het 100-ste getal.

Daar het 100-ste getal het laatst gelezen wordt, moeten de eerste 99 eerst opgeslagen worden, waarvoor een array gekozen wordt. Het 100-ste getal kan dan achtereenvolgens met de getallen uit het array vergeleken worden, welk proces echter gestopt moet worden zodra een gelijkheid wordt geconstateerd. Ook het 100-ste getal wordt maar in het array opgeslagen om het zoekproces in ieder geval te beëindigen. Zowel voor het inlezen van de getallen als voor het vergelijken van getallen is een variabele i nodig met de betekenis: i is het rangnummer van het te lezen, resp. te vergelijken getal. De algorithme is dan:

```
begin integer getal, i; integer array g[1:100];
      i := 1; while i ≤ 100 do read (g[i]); i := i + 1 od;
      getal := g[100];
      i := 1; while getal ≠ g[i] do i := i + 1 od; write(i)
end
```

7.1.2. Gegeven is een rij natuurlijke getallen, afgesloten door een 0. Gevraagd wordt te bepalen hoe vaak elk van de getallen 11 tot en met 20 in deze rij voorkomen.

Uit de opgave is duidelijk dat, totdat een 0 gelezen is, met uitzondering van de verzameling V van de getallen 11 tot en met 20 de getallen slechts ingelezen hoeven te worden. Een globale algorithmevorm is dus

```
while getal ≠ 0 do "als getal ∈ V dan tellen hoe vaak dit getal nu al
                        voorgekomen is";
                   "lees (volgende) getal"
od
```

In een aantal tellers moet dus bijgehouden worden hoe vaak de getallen van V al voorgekomen zijn. Het ligt voor de hand om die tellers bij elkaar te plaatsen in een array met indices 11 tot en met 20, dat oorspronkelijk op 0 geïnitialiseerd is. Iedere keer dat een getal uit V geconstateerd wordt, wordt de corresponderende array component met 1 opgehoogd. Voor de initialisatie van de while-lus moet vooraf het eerste getal van de rij ingelezen worden. De algorithme wordt dus

```
begin integer array tel[11:20]; integer getal, i;
      i := 10; while i < 20 do i := i + 1; tel[i] := 0 od;
      read (getal);
      while getal ≠ 0 do if getal ≥ 11 and getal ≤ 20
                            then tel[getal] := tel[getal] + 1
                         fi;
                         read (getal)
                      od;
      i := 10; while i < 20 do i := i + 1; write (tel[i]) od
end
```

7.1.3. gevraagd wordt om van een rij van 100 in te lezen getallen het grootste en het kleinste af te drukken, evenals het rangnummer van het eerste getal gelijk aan het minimum en het rangnummer van het laatste getal gelijk aan het maximum.

Men zou in de verleiding komen om eerst de 100 getallen in een array op te slaan en dan op zijn gemak max en min (en hun plaats) te zoeken. Nodig is dat niet omdat direct na het inlezen van het eerste getal gesteld kan worden dat dit nu het maximum en het minimum van de tot dusver gelezen rij is. Deze bewering kan na het inlezenvan ieder volgend getal zo nodig herzien worden. Voor het bepalen van de gewenste rangnummers moet bijgehouden worden dat het zoveelste getal onder handen is . De volgende variabelen zullen nodig zijn:

g : het laatst gelezen getal
i : het rangnummer van het laatst ingelezen getal;
imin : het rangnummer van het eerste getal gelijk aan het tot dusver gevonden minimum;
imax : het rangnummer van het laatste getal gelijk aan het tot dusver gevonden maximum;
max, min: de tot dusver gevonden max en min waarden.

```
begin integer i, imin, imax, g, max, min;
      read(g);  max := min := g; i := imax := imin := 1;
      while i < 99 do i := i + 1; read(g);
                    if g < min then min := g; imin := i
                              else if g ≥ max then imax := i;
                                                   max := g
                                   fi
                    fi
                od;
      write(imax); write(max); write(imin); write(min)
end
```

7.1.4. Gegeven is een rij positieve gehele getallen, afgesloten door een 0. De som moet bepaald worden van het 1e, 3e, 6e, 10e,... getal uit de rij.

Ten aanzien van het inlezen worden alle getallen op dezelfde wijze behandeld. Ten aanzien van de verwerking is er echter verschil in behandeling; er zijn "te sommeren" getallen (het 1e, 3e, 6e, 10e,... getal), die bij elkaar opgeteld moeten worden (in bijvoorbeeld een variabele som) en "niet te sommeren" getallen, die verder genegeerd kunnen worden. Een cycle is blijkbaar nodig om de getallen in te lezen, totdat het getal 0 is gelezen. Een globale vorm van de cycle is

```
while gelezen getal ≠ 0
      do if "getal is sommeerbaar"
            then "tel getal op bij som";
                 "bepaal rangnummer van volgend sommeerbaar getal"
         fi;
         "lees volgend getal"
      od
```

Een te sommeren getal is een getal waarvan het rangnummer in de rij tot de "kandidaten" (1,3,6,10,...) behoort. De kandidaten worden als volgt gevonden: De eerste kandidaat is 1; het verschil in rangnummer van de eerste en de tweede kandidaat is 2; het verschil in rangnummer van ieder tweetal opeenvolgende kandidaten is telkens 1 groter dan het verschil in rangnummer van het direct daaraan voorafgaand tweetal opeenvolgende kandidaten.

Het proces is volledig bepaald door de waarden van de volgende variabelen:

getal: het laatst gelezen getal uit de rij;
i : het rangnummer van dit laatst gelezen getal;
k : het rangnummer van het eerstvolgende te sommeren getal;
v : het verschil tussen k en het rangnummer van het voorafgaand te sommeren getal;
som : het totaal van de te sommeren getallen t/m het laatst verwerkte getal van die soort.

Uit deze beschrijving volgt nu direct:

```
begin integer getal, i, k, v, som;
      k := 1; v := 1; som := 0;
      read(getal); i := 1;
      while getal ≠ 0
            do if i = k then som := som + getal;
                             v := v + 1;
                             k := k + v
               fi;
               read(getal); i := i + 1
            od;
      write(som)
end
```

7.1.5. Gegeven is een invoerrij van tweehonderd gehele getallen:

$$x_1, x_2, \ldots, x_{100}, y_1, y_2, \ldots, y_{100} .$$

Wanneer het rijtje x-en elementsgewijs gelijk is aan het rijtje y-en, dan moet "gelijk" worden afgedrukt; anders moet "ongelijk" worden afgedrukt.

Aangezien de eerste y pas na de 100 x-en komt, moeten de x-en wel ingelezen en opgeslagen worden in een array. Daarna kunnen de y-en stuk voor stuk verwerkt worden, d.w.z. iedere y vergeleken met de overeenkomstige x. Een globale vorm van de algorithme is dan:

```
begin .....
      "lees 100 x-en in"
      while "vergelijkingsproces is niet afgelopen"
            do "lees (volgende) y";
               "vergelijk deze y met de corresponderende x"
            od;
      ..........
end
```

Nodig zijn dus de volgende variabelen:

x : array met grenzen [1:100]; hierin worden de eerste honderd getallen uit de rij geplaatst: x[i] bevat x_i,

y : de uit de (tweede helft van de) rij gelezen waarde die met de overeenkomstige x uit de eerste helft vergeleken wordt,

i : - in de eerste while-lus de index van het volgende element van x dat ingelezen wordt,
- daarna rangnummer van het y-getal dat indien nog nodig vergeleken wordt met x_i in het array,

equal: boolean variabele; deze heeft de betekenis dat bij een bepaalde waarde van i ($1 < i \leq 100$), alle paren x,y met index 1 t/m i - 1 gelijk zijn (de waarde van equal is dan true) of dat alleen $x_{i-1} \neq y_{i-1}$ (de waarde van equal is dan false); voor i = 1 krijgt equal de waarde true omdat er aanvankelijk nog geen verschil is geconstateerd.

Het vergelijken van de getallen moet gestopt worden wanneer alle honderd y-en verwerkt zijn ($i > 100$), of wanneer er een verschil is geconstateerd (equal heeft dan de waarde false). Er moet dus nog een volgende y met een x vergeleken worden zolang geldt: $i \leq 100$ and equal. Heeft na beëindiging van de while statement equal de waarde true (dan is dus gestopt omdat $i > 100$), dan zijn de rijen gelijk; heeft equal echter de waarde false dan is verschil geconstateerd. Een derde while-lus is toegevoegd om er voor te zorgen dat alle y-waarden gelezen zijn.

```
begin integer i, y; boolean equal; integer array x[1:100];
      i := 1;
      while i ≤ 100 do read(x[i]); i := i + 1 od;
      i := 1; equal := true;
      while i ≤ 100 and equal
            do read(y);
               equal := y = x[i];
               i := i + 1
            od;
      while i ≤ 100 do read(y); i := i + 1 od;
      if equal then write("gelijk") else write("ongelijk") fi
end
```

7.2. "Numerieke" problemen

Bij opgaven waarbij veel "gerekend" wordt, met name met reals, worden de programmeringsproblemen o.a. bepaald door:

- de opmaak van grote hoeveelheden resultaten in bijvoorbeeld tabelvorm (denk onder andere aan tafels);
- afrondings- en nauwkeurigheidskwesties;
- de doelmatigheid van de benodigde algorithmen;

Overwegingen die een rol spelen bij het maken van tabellen zijn

- uit hoeveel regels moet een blok opgebouwd worden en hoeveel posities nemen kolomscheidingen in?
- hoeveel getallen moeten berekend worden; hoe moeten argumentwaarden over de eerste tabelkolom en de overige kolomkoppen verdeeld worden? Hoeveel regels tabel volgen hieruit?
- uit hoeveel regels moet een blok opgebouwd worden en hoeveel blokken gaan op een pagina gezien de maximale paginahoogte, de regelafstand en eventuele blokscheiders?

Wanneer een antwoord is gevonden op deze vragen is de programmering van de tabelopmaak betrekkelijk eenvoudig, al zijn er veel geneste while-lussen zoals uit bijgaande programmaschets blijkt.

```
begin
      ap := "aantal pagina's per tabel"
      while ap > 0
            do ap := ap - 1; "nieuwe pagina";
               "kopregel(s) maken en afdrukken";
               ab := "aantal blokken per pagina";
               while ab > 0
                     do ab := ab - 1;
                        "blokscheiding maken en afdrukken";
                        ar := "aantal regels per blok";
                        while ar > 0
                              do ar := ar - 1;
                                 "maak en plaats eerste getal op regel";
                                 ag := "aantal getallen per regel";
                                 while ag > 0
                                       do ag := ag - 1;
                                          "plaats kolomscheiding";
                                          "maak en plaats volgend getal";
                                       od;
                                 "druk regel af"
                              od
                     od
            od;
      "nieuwe pagina" {ter voorkoming van beschrijving laatste tabelpagina}
end
```

Binnen in dit programma zit nog de berekening van een functiewaarde. Het zoeken van de meest doelmatige algorithme zal als probleem van de numerieke wiskunde hier niet verder besproken worden. Bij het programmeren van de algorithme moet zo veel mogelijk gebruik worden gemaakt van integers om accumulatie van afrondingsfouten te voorkomen. Als bijvoorbeeld een aantal sinuswaarden voor x = 0(0.1)1 afgedrukt moet worden, moet dat niet gebeuren met

```
x := 0; while x ≤ 1.0 do write(sin(x)); x := x + 0.1 od
```

maar met

```
i := 0; while i ≤ 10 do write(sin(0.1 * i)); i := i + 1 od
```

In het eerste geval worden met x := x + 0.1 door de niet exact representeerbare 0.1 afrondingsfouten geaccumuleerd. Daardoor wordt de while-lus mogelijk te vaak of te weinig doorlopen en wordt op de duur de sinus van een verkeerd argument berekend.
Hoe deze afrondingsfouten kunnen doorwerken op de programmering zal tenslotte nog toegelicht worden in het volgende ingewikkelder voorbeeld.

7.2.1. Sommatie van de Taylorreeks voor arc sin (x)

De Taylorreeks voor arc sin (x) wordt gegeven door (x < 1):

$$(1) \qquad r = x\left(1 + \frac{1}{2}\frac{x^2}{3} + \frac{1.3}{2.4}\frac{x^4}{5} + \frac{1.3.5}{2.4.6}\frac{x^6}{7} + \ldots\right)$$

De problemen voor de programmering zijn in hoofdzaak van numeriek wiskundige aard. In de eerste plaats moet de oneindige som vervangen worden door een sommatie tot die bovengrens waarvoor de restterm kleiner is dan de voor arc sin (x) vereiste nauwkeurigheid. Voor het vervolg wordt verondersteld dat bekend is dat daartoe n termen in de sommatie moeten worden meegenomen.
Uit bovenstaande reeks is af te lezen dat de i-de term van r/x gegeven wordt door $t_1 = 1$ en

$$(2) \qquad t_i = \frac{1.3.\ldots.(2i-3)}{2.4.\ldots.(2i-2)} \cdot \frac{(x^2)^{i-1}}{2i-1} \qquad (i > 1)$$

Hieruit volgt dat:

$$(3) \qquad t_{i+1} = \frac{(2i-1)^2}{2i} \cdot \frac{x^2}{2i+1} \cdot t_i \qquad (i \geq 1)$$

Zou men echter (3) gebruiken om uitgaande van t_1 (=1) de hogere t_i te berekenen dan krijgt men een accumulatie van afrondingsfouten. Om dat te vermijden is het daarom verstandiger om (2) te herschrijven in een vorm waarin zo veel mogelijk met integers gewerkt wordt.

$$t_i = p_i * g_i/(h_i * k_i) \qquad (i > 1)$$

met

$$t_1 = p_1 = g_1 = h_1 = k_1 = 1$$

$$\left.\begin{aligned} p_{i+1} &= p_i * x^2 \\ g_{i+1} &= g_i * k_i \\ h_{i+1} &= h_i * (k_i + 1) \\ k_{i+1} &= k_i + 2 \end{aligned}\right\} \qquad (1 \leq i < n)$$

(Men zou nog kunnen onderzoeken of $(x^2)^i$ niet nauwkeuriger met exp(i * 2 * ln(x)) te berekenen is.)
De algorithme voor de Taylorreeks heeft dan de volgende eenvoudige vorm

```
integer g, h, k, i, n; real som, t, p, r, x, xx;
g := h := k := i := 1; som := t := p := 1,0
xx := x * x;
while i < n
      do p := p * xx;
         g := g * k;
         h := h * (k + 1);
         k := k + 2;
         t := p * g/(h * k);
         som := som + t;
         i := i + 1
      od;
r := som * x
```

7.2.2. Gemiddelde en variantie van een rij reële getallen a_i

Gemiddelde en variantie van een rij getallen zijn gedefinieerd door

$$gem = (\sum_{i=1}^{n} a_i)/n$$

$$var = \sum_{i=1}^{n} (a_i - gem)^2/(n - 1) \ .$$

Uitwerking van het kwadraat in var levert in verband met de definitie van gem op dat

$$var = [\sum_{i=1}^{n} a_i^2 - n * gem^2]/(n - 1) \ .$$

Deze uitdrukking brengt minder bewerkingen met zich mee, zoals eenvoudig is na te gaan. Het is duidelijk dat de kern van de berekening gevormd wordt door

$$soma = \sum_{i=1}^{n} a_i \quad \text{en} \quad somak = \sum_{i=1}^{n} a_i^2$$

omdat dan verder

$$gem = soma/n \quad \text{en} \quad var = (somak - soma^2/n)/(n - 1)$$

(var wordt in deze vorm geschreven om de nauwkeurigheid van de berekening zo hoog mogelijk te houden). Als nu gegeven is dat de getallen n en a_i op het invoermedium staan, is de vraag hoe de algorithme opgesteld moet worden.

Men zou een array kunnen declareren, waarin de getallen a_i ingelezen worden en vervolgens weer uitgelezen voor de berekening van soma en daarna nog eens voor somak. Dit zijn dan drie repetities, die dezelfde structuur hebben. Enig nadenken leert echter dat de invoering van een array overbodig en zelfs inefficiënt is. Als namelijk een a_i-waarde ingelezen is, kan de bijdrage ervan meteen verwerkt worden in soma en somak. In de enkele repetitie waarin dit gebeurt zijn de invarianten dat:

- i het aantal tot dan ingelezen getallen bevat
- soma de som van alle tot dan ingelezen getallen bevat
- somak de som van de kwadraten van alle ingelezen getallen bevat.

De gezochte algorithme is nu vrijwel meteen op te schrijven:

```
begin real ai, soma, somak, gem, var; integer i, n;
      soma := somak := 0; i := 0; read(n);
      while i < n do read(ai);
                     soma := soma + ai;
                     somak := somak + ai * ai;
                     i := i + 1
                  od;
      gem := soma/n;
      var := (somak - soma * soma/n)/(n - 1);
      write(gem); write(var)
end
```

7.3. Grotere voorbeelden

7.3.1. De eerste honderd priemgetallen

Daar priemgetallen gedefinieerd zijn door het feit dat ze alleen deelbaar zijn door 1 en zichzelf, is met enig proberen te constateren dat het begin van een rij van priemgetallen gegeven wordt door

2,3,5,7,11,13,17,... .

Is een priemgetal gevonden, dan is het eerst volgende getal dat op delers onderzocht moet worden twee meer, omdat er afgezien van 2 geen andere even priemgetallen zijn. Bij het zoeken van delers van een mogelijk priemgetal p hoeven voorts slechts de reeds eerder gevonden priemgetallen geprobeerd te worden en niet hun veelvouden. Bij het proces moeten dus gevonden priemgetallen niet alleen afgedrukt worden, maar ook opgeslagen. Daar de kleinste priemgetallen de grootste kans hebben om als deler op te treden kan bij het deelbaarheidsonderzoek het best begonnen worden bij de kleinste priemgetallen. Bij dit proberen van de reeds gevonden priemgetallen als delers hoeft in de rij van reeds gevonden priemgetallen zeker niet verder gekeken te worden dan het grootste priemgetal dat $\leq \sqrt{p}$ is omdat een mogelijk groter priemgetal als deler een factor zou hebben opgeleverd die $\leq \sqrt{p}$ is. Evenmin hoeft verder gekeken te worden als een deler gevonden is. Na de voorgaande vaststellingen blijven er eigenlijk nog twee deelproblemen over, nl. hoe is te constateren dat een reeds gevonden priemgetal, zeg q, een deler is van p of niet? Hoe is te constateren dat q niet groter is dan

$\sqrt{p}$ als geen gebruik wordt gemaakt van de worteltrekking. De antwoorden op deze vragen worden gegeven door het bekijken van de waarde van p mod q (het 0 zijn geeft aan dat q een deler van p is) en het teken van q ** 2 - p.

De structuur van een algorithme is:

```
"plaats 2 als eerste en enige even priemgetal in een array met 100 plaatsen"
while "array is niet vol"
      do gok := gok + 2; (eerste)deler = eerste priemgetal;
         while "verder te proberen"
               do "bepaal gok mod deler en volgende deler" od;
         if"gok een priem is then plaats deze in array" fi
      od
```

Voor het verder detailleren van de algorithme worden de volgende variabelen ingevoerd:

pa[1:100]: een "priemenarray" met 100 plaatsen,
pav : geeft aan dat pav plaatsen van pa reeds gevuld zijn, $1 \leq pav \leq 100$
pad : geeft aan dat pa[pad] als deler van gok geprobeerd wordt, $pad \leq pav$
gok : variabele die mogelijk priemgetal is,
glp : hulpgrootheid ("gok lijkt priem") bij deelbaarheidsonderzoek van gok

Uitgeschreven wordt de gezochte algorithme dan

```
begin integer array pa[1:100]; integer pav, pad, gok, boolean glp;
      pa[1] := 2; pav := gok := 1;
      while pav < 100
            do gok := gok + 2; pad := 1; glp := true;
               while pa[pad]**2 ≤ gok and glp                    1)
                     do glp := gok mod pa[pad] ≠ 0;
                        pad := pad + 1
                     od;
               if glp then pav := pav + 1; pa[pav] := gok; write(gok) fi
            od
end
```

Opmerking. Gezien de conditie pad ≤ pav gaat 1) alleen goed dank zij een stelling uit de getallentheorie die zegt dat elk priemgetal kleiner is dan het kwadraat van zijn voorganger. Daardoor wordt glp eerder false dan dat pad > pav zou worden (en met het laatste pa[pad] ongedefinieerd).

7.3.2. De eerste honderd W-getallen

De verzameling van W-getallen is gedefinieerd door

- 1 hoort tot de verzameling
- als w er toe behoort, dan behoren ook 2w + 1 en 3w + 1 er toe
- geen andere getallen horen er toe.

Gevraagd wordt de eerste honderd W-getallen in volgorde te genereren; n.l. w_1 (= 1) $w_2 \ldots w_{100}$.

Als de eerste getallen gevonden zijn, dan volgt uit de definitie van de verzameling dat voor w_k geldt dat

. er een index i is, zodat

$$w_k = 2w_i + 1 \quad \text{en dus} \quad w_k > 2w_{i-1} + 1 \qquad 1 \le i < k \qquad (1)$$

. en/of er een index j is, zodat

$$w_k = 3w_j + 1 \quad \text{en dus} \quad w_k > 3w_{j-1} + 1 \qquad 1 \le j < k \qquad (2)$$

Omdat de W-getallen in volgorde geproduceerd moeten worden, dienen blijkbaar twee indices i en j te worden bijgehouden die aangeven van welke reeds gevonden W-getallen 2- of 3-vouden + 1 nog te plaatsen zijn. Aannemend dat de twee indices goed zijn bijgehouden, wordt het volgende W-getal bepaald door

$$w_{k+1} = \min(2w_i + 1,\ 3w_j + 1) \ .$$

Zodra een nieuw W-getal gevonden is zal tenminste één index, maar mogelijk 2 indices verhoogd moeten worden. Een verhoging met slechts 1 is toegestaan, omdat geen W-getallen overgeslagen mogen worden.

Een algorithme voor de berekening van de W-getallen heeft dan de volgende vorm:

```
while k < 100 do k := k + 1; "bepaal en plaats w[k]";
                 "verhoog i en/of j met 1"
              od
```

Aangezien w[1] = 1 het eerste W-getal is moet deze lus begonnen worden met

```
w[1] = 1; k := 1; i := 1; j := 1;
```

en de while-lus wordt (met invoering van een hulpvariabele h):

```
while k < 100 do k := k + 1;
                 h := 2 * w[i] + 1;
                 if h > 3 * w[j] + 1
                    then w[k] := 3 * w[j] + 1; j := j + 1
                    else w[k] := h; i := i + 1;
                         if h = 3 * w[j] + 1
                            then j := j + 1
                         fi
                 fi
              od
```

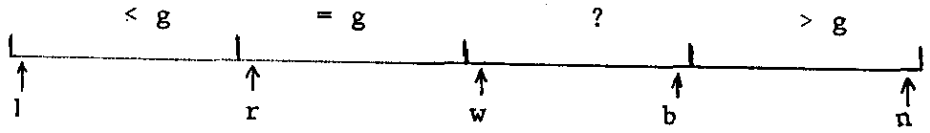
De invariant van deze while-lus is blijkbaar

- w[1],...,w[k] zijn in monotoon stijgende volgorde de k kleinste W-getallen
- i is de kleinste index waarvoor $2 * w[i] + 1 > w[k]$ $\quad 1 \leq i < k$
- j is de kleinste index waarvoor $3 * w[j] + 1 > w[k]$ $\quad 1 \leq j < k$

Een uitbreiding van de probleemstelling, nl. met de voorwaarde dat bijv. ook 5w + 1 tot de W-getallen behoort, vereist een vrij eenvoudige aanpassing van de algorithme. Minder triviaal is de opgave om van een natuurlijk getal te onderzoeken of die tot de W-getallen behoort.

7.3.3. De Nederlandse vlag

Gegeven is een array a[1 : n] met n willekeurige getallen en een getal g. Gevraagd wordt de getallen zo te rangschikken dat links aaneengeschoven de getallen kleiner dan g staan, rechts aaneengeschoven de getallen groter dan g en daar tussen in de mogelijke getallen gelijk aan g. Deze driedeling van de getallen rechtvaardigt de naam waaronder dit probleem bekend is door de drie groepen getallen met de kleuren rood, wit en blauw te identificeren.



Het probleem is ten dele opgelost als de bovenstaande situatie bereikt is (de beginsituatie is hier een bijzonder geval van, nl. w = l en b = n):

a[k] < g	voor $l \leq k < r$
a[k] = g	voor $r \leq k < w$
a[k] niet onderzocht	voor $w \leq k \leq b$
a[k] > g	voor $b < k \leq n$.

Voor een verbetering van deze situatie hoeven a[l] t/m a[r - 1] en a[b + 1] t/m a[n] niet meer beschouwd te worden. Voor a[w] zijn de volgende gevallen te onderscheiden

a[w] < g: het ligt voor de hand om het getal a[w] over te brengen naar de positie r en het daar aanwezige getal naar positie w. Voorts kunnen r en w met 1 vermeerderd worden;

a[w] = g: geen verwisseling van getallen is nodig, doch w kan met 1 vermeerderd worden;

a[w] > g: verwisseling van de getallen a[w] en a[b] ligt dan voor de hand, terwijl b met 1 verminderd kan worden.

Het aantal nog te onderzoeken getallen is tenslotte met 1 verminderd, en het proces is afgelopen zodra w > b. De algorithme wordt

```
r := w := 1; b := n;
while w ≤ b
      do if a[w] < g
            then z := a[r]; a[r] := a[w]; a[w] := z; r := r + 1; w := w + 1
            else if a[w] > g then z := a[b]; a[b] := a[w]; a[w] :=z; b := b - 1
                            else w := w + 1
                 fi
         fi
      od
```

7.3.4. Een loket probleem

Van een loket wordt gebruik gemaakt door klanten genummerd van 1 t/m n, die in volgorde van aankomst bediend worden zodra het loket vrij is. Aankomsttijd en bedieningsduur van klant i worden gegeven door a_i en b_i (> 0). Als het loket opengaat op tijdstip 0 en sluit na vertrek van klant n en als $0 \leq a_1 < a_2 \ldots < a_n$ wat is dan het sluitingstijdstip en de lengte van de langste rij die voor het loket gestaan heeft?

De vraag naar het sluitingstijdstip leidt tot de gedachte om te letten op het vertrektijdstip van de klanten. Voor klant 1 is dat uiteraard $v_1 = a_1 + b_1$, doch voor iedere volgende klant hangt het er van af of hij in een lege rij arriveert (dan is het vertrekmoment voor klant i $v_i = a_i + b_i$), dan wel in een niet-lege rij (dan is het vertrekmoment $v_{i-1} + b_i$). Dit is samen te vatten met

$$v_0 = 0$$

$$v_i = \max(a_i, v_{i-1}) + b_i \qquad 1 \leq i \leq n \ .$$

Hiermee zijn de vertrekmomenten achtereenvolgens te berekenen en v_n geeft meteen het sluitingstijdstip. Op ieder vertrekmoment v_i vermindert de rijlengte r met 1, doch vergeleken met het vorige vertrekmoment v_{i-1} kan de rijlengte intussen toegenomen zijn met alle klanten j waarvoor

$$a_j > v_{i-1} \ \underline{\text{and}} \ a_j \leq v_i \ \underline{\text{and}} \ j \leq n \ .$$

Wegens de monotonie van a_j kan de eerste voorwaarde vervangen worden door "begin met die j-waarde die op het moment v_i nog niet beschouwd was". De

langste rij wordt dan gevonden met $\max(r_i)$ voor $1 \leq i \leq n$.

Een algorithme om de gevraagde grootheden te berekenen is nu:

```
while "nog niet alle klanten afgehandeld"
    do "bereken vertrektijd van klant aan loket";
       if "nog niet alle klanten gearriveerd"
          then "bereken aantal klanten gearriveerd in interval na vertrek
                vorige klant tot en met vertrektijd klant aan loket";
               "tel dit aantal op bij rijlengte ten tijde van vertrek
                vorige klant"
               "pas mogelijk nieuw maximum voor rijlengte aan"
       fi;
       "rijlengte met 1 verminderen"
    od
```

Voor de vastlegging van dit proces zullen naast n, a[i] en b[i] de volgende variabelen nodig zijn:

v : vertrektijd van klant aan loket (een index is niet nodig omdat uiteindelijk alleen de vertrektijd van de laatste klant gevraagd is en alleen de voorlaatste en laatste v van belang zijn)

r : aantal klanten in rij op vertrektijd van klant aan loket (ook hier is geen index nodig omdat slechts de voorlaatste en laatste r van belang zijn)

max : de grootste waargenomen waarde van r

i : het aantal reeds afgehandelde klanten

j : het aantal gearriveerde klanten t/m vertrektijd van klant aan loket

next : hulpgrootheid voor de berekening van j; true als a[j] ≤ v

Uitgeschreven wordt de algorithme:

```
i := v := r := max := 0; j := 1;
while i < n
      do i := i + 1;
         if a[i] ≥ v then v := a[i] + b[i] else v := v + b[i] fi;
         if j ≤ n then next := a[j] ≤ v;
                       while next do j := j + 1; r := r + 1;
                                     if j ≤ n then next := a[j] ≤ v else next := false fi
                                  od;
                       if r > max then max := r fi
         fi;
         r := r - 1
      od
```

De invariant van de lus is, zoals ook reeds volgt uit de betekenis van de variabelen

v = vertrektijd van klant i
j = grootste waarde van k ($1 \leq k \leq n$) waarvoor a[k] ≤ v
r = rijlengte op tijdstip v
max = maximale rijlengte opgetreden t/m tijdstip v

7.4. Aan algorithmen te stellen eisen

Een eerste eis die aan een algorithme gesteld moet worden is dat hij correcte resultaten oplevert bij gebruik onder de omstandigheden waarvoor hij ontworpen is. Een correctheidsbeschouwing zal daarom niet mogen ontbreken bij een ontworpen algorithme of nog beter: bij het ontwerp moet de correctheid steeds in het oog worden gehouden. Een oude methode voor het beproeven van een algorithme, nl. de verwerking van een aantal testgevallen (voor als regel gemakkelijk door te rekenen omstandigheden) is namelijk alles behalve veilig. Geven die namelijk het gewenste resultaat dan is daarmee slechts de afwezigheid van grove blunders en alleen voor de testgevallen de correctheid aangetoond.

Een tweede eis, nl. dat de algorithme doeltreffend ("efficient") is, houdt in dat de totale verwerkingstijd kort moet zijn. In de complexiteitstheorie wordt daartoe onderzocht hoe het aantal bewerkingen in een algorithme afhangt van de "omvang" van het probleem (zie bijv. de behandelde sorteeralgorithme); in de numerieke wiskunde wordt onderzocht hoe het aantal herhalingen van een cyclus zo klein mogelijk kan worden gehouden, gezien

de gewenste nauwkeurigheid, enz.

Naast deze twee eisen moeten echter enkele andere gewenste eigenschappen in het oog worden gehouden, nl.

- de doelmatigheid. Correcte algorithmen worden nl. in de praktijk te hulp geroepen omder omstandigheden waarvoor ze niet gemaakt zijn. Het is daarom meestal verstandig om de aanloop van een algorithme zo in te richten dat daarin gecontroleerd wordt of de omstandigheden inderdaad de juiste zijn. Is dat niet het geval dan hoort de berekening afgebroken te worden na het produceren van een diagnostische boodschap. Voorbeelden: in een vierkantswortelberekening eerst controleren of het argument niet negatief is. Door invoering van overtollige gegevens (redundantie) als controledigits moet men zich wapenen tegen gebruikersfouten, door codes te gebruiken tegen apparatuurfouten.
- de modulariteit. Grote, complexe problemem moeten opgesplitst worden in zo zelfstandig mogelijke deelproblemen. Dit maakt het niet alleen mogelijk om met meer mensen aan een probleem te werken (eventueel met verschillende specialismen), maar bevordert ook de overzichtelijkheid en documentatie, correctheid, de mogelijkheid tot latere aanpassing ("onderhoud") en de herstartbaarheid bij apparatuurfouten.
- de leesbaarheid. Tussen leesbaarheid en doeltreffendheid moet een redelijk compromis getroffen worden (gezien bijvoorbeeld de mogelijkheid tot het invoeren of elimineren van variabelen). Een hulpmiddel is daarbij het toevoegen van commentaar aan de algorithmetekst.

8. Processen en deelprocessen

Bij de bespreking van processen (1.2) is reeds opgemerkt dat men een proces de ene keer wil uitschrijven in termen van elementaire acties, de andere keer echter het liefst opvat als een "ondeelbare" actie. Voor het laatste tracht men dan een geschikte notatie af te spreken. Ook in de behandelde voorbeelden kwam dit steeds naar voren. Bij de geheeltallige deling worden eerst div en mod als ondeelbare acties opgevat en vervolgens uitgeschreven in termen van de (nog elementairder) acties optellen en aftrekken. In het grootste gemene deler voorbeeld werd het verwisselen van de waarden van twee variabelen eerst als ondeelbare actie opgevat , maar daarna uitgeschreven; de mod bewerking werd daarentegen niet verder uitgeschreven.

In het sorteervoorbeeld kwam naar voren hoe verbaal weergegeven gecompliceerde acties stapsgewijs ontleed werden in eenvoudiger en tenslotte in standaard elementaire acties.

Een uitgeschreven deelproces in een procesbeschrijving wordt verduidelijkt door dit deelproces als zodanig te markeren, bijvoorbeeld met behulp van dezelfde begin - end haken die om een algorithme gezet worden. Ook wat tussen do en od bij een while-opdracht staat is op te vatten als een deelprocesbeschrijving. De daarin voorkomende variabelen zijn alleen elders in de algorithme gedeclareerd zodat dit deelproces zich afspeelt in een deelruimte van de totale toestandsruimte.
Vaak komen in de beschrijving van het deelproces "locale" variabelen voor, die alleen een rol spelen binnen het deelproces en niet daar buiten. Het ligt dan voor de hand om de declaratie van deze locale variabelen pas te laten plaatsvinden binnen het deelproces, dus direct na de begin-openingshaak. Zo'n deelproces waarin locale variabelen gedeclareerd zijn wordt een blok genoemd, een deelproces zonder locale variabelen een samengesteld (of compound) statement. In de deelprocesbeschrijving treden ook z.g. globale variabelen uit de omgeving op, variabelen waarvan de waarden in het blok voor de berekening gebruikt worden en waaraan in het blok de resultaten van de berekening worden toegekend, die later in de omgeving kunnen worden gebruikt. Via de locale variabelen kunnen namelijk geen resultaten naar de omgeving worden overgebracht omdat die locale variabelen daar niet bekend zijn. Op de naamgeving van locale variabelen wordt verderop nader ingegaan.

Wanneer een deelproces verschillende keren in een proces voorkomt (eventueel met andere gegevens opererend) wordt dit nog meer verduidelijkt door het niet verschillende keren uit te schrijven, doch dit aan te geven met een z.g. procedure statement, bestaande uit een geschikt gekozen naam met daar achter tussen ronde haakjes de betreffende gegevens. Wat dan precies uitgevoerd wordt, moet net als andere grootheden in het begin van het betreffende programma (of blok) gedeclareerd worden met een proceduredeclaratie. De vorm van deze samengestelde actie wordt later besproken, doch het is duidelijk dat hierin staat hoe uit bepaalde gegevens uit de omgeving de weer voor deze omgeving gewenste resultaten berekend worden.

8.1. Het blok

De schrijver van een blok moet natuurlijk van de schrijver van het omvattende programma horen welke globale variabelen (en hun betekenis) hij kan verwachten. Om het hem bij de naamgeving van zijn locale variabelen zo gemakkelijk mogelijk te maken wordt het in programmeertalen mèt blokstructuren toegestaan dat de namen van die locale variabelen overigens vrij gekozen worden. Een globale variabele uit de omgeving en een locale variabele uit een blok kunnen dan dezelfde naam hebben. Een bezwaar is dat niet wanneer de afspraak gemaakt wordt dat zo'n gemeenschappelijke naam
- buiten het blok geen betrekking zal hebben op de locale variabele
- binnen het blok uitsluitend betrekking zal hebben op de locale variabele.
(Dat een locale variabele slechts betekenis heeft in een blok, wordt ook weergegeven met "de scope van een locale variabele is tot het betreffende blok beperkt").

De globale variabele met dezelfde naam als een locale variabele is binnen het blok onbekend en onbereikbaar. Bij het verlaten van een blok verliezen de locale variabelen hun betekenis en ook hun waarde; ze bestaan niet meer. Dit impliceert dat op het moment van het (opnieuw) binnenkomen van hetzelfde blok (hetgeen mogelijk is daar een blok overal mag staan waar een statement mag staan) de waarden van de locale variabelen (weer) ongedefinieerd zijn. In het voorbeeld

```
begin integer a,b,c;
      ----------
      begin integer c,d,e;
            ----------
      end;
      ----------
end
```

bestaan de variabelen a, b en c van het buitenblok (programma) in het gehele programma, terwijl de variabelen c, d en e (in het binnenblok gedeclareerd) alleen bestaan in dit binnenblok; de variabele c van het buitenblok is in het binnenblok niet bereikbaar; als in het binnenblok een c voorkomt dan is dit de c van het binnenblok; na het verlaten van het binnenblok heeft de c van het buitenblok nog steeds de waarde die hij bezat voordat het binnenblok

werd geactiveerd").
De grenzen in een array-declaratie worden bij activering van het omvattende blok (opnieuw) berekend. Dit geeft de mogelijkheid om dynamisch(d.w.z. tijdens de uitvoering van het programma) de grenzen van een array te bepalen:

```
begin integer n;
      read(n);
      begin real array A[1:n];
            ----------
      end
end
```

Dit is niet te bereiken met bijvoorbeeld:

```
begin integer n; read(n); real array A[1:n] ..... end
```

omdat na een statement geen declaraties mogen volgen (in hetzelfde blok).
Het grote voordeel van de blokstructuur is dat bij de programma-opbouw gedacht kan worden aan acties die later als blokken uit te schrijven zijn.

8.2. De procedure

Bij het begrip procedure spelen drie aspecten een rol:

- Het effect van de procedure, wàt bewerkstelligt de procedure. Diegene die een al gedeclareerde procedure wil gebruiken zal een eenduidige definitie wensen van het effect of resultaat van de procedure. De exacte beschrijving van het effect van de procedure is echter ook van belang voor de "constructeur", want aan de hand hiervan moet hij de procedure "construeren". Men kan om het lezen van een programma te vereenvoudigen, in de naam van de procedure uitdrukken wat het effect is van de procedure.
- De samenhang van de procedure met de omgeving. Deze samenhang wordt gerealiseerd via parameters. In de declaratie van de procedure worden "formele" parameters (namen) vastgelegd. Bij de activering van de procedure worden "actuele" parameters opgegeven. Via deze actuele parameters krijgt de procedure invoerwaarden uit de omgeving (invoerparameters) en kan de procedure de berekende waarden afleveren aan de omgeving (uitvoerparameters).

- De beschrijving van de wijze waarop de procedure de voor het gewenste effect geëiste samenhang tussen de formele parameters tot stand brengt (hoè werkt de procedure). Dit komt neer op een beschrijving van het proces, een schema van acties dat, uitgaande van een bepaalde begintoestand (bepaald door de invoerwaarden) een bepaalde eindtoestand (vastgelegd in de uitvoervariabelen) oplevert. Deze beschrijving van het effect wordt de body van de procedure genoemd. De body is een blok waarin naast de formele parameters ook locale variabelen een rol kunnen spelen.

In het voorgaande is uitgegaan van een algemeen schema om, door invulling van de actuele parameters, te komen tot een bepaalde actie. Ook het omgekeerde is mogelijk. Zo kan het in 4.1 behandelde worden opgevat als de beschrijving van de bepaalde actie: "Bepaal quotiënt en rest van de natuurlijke getallen en leg de resultaten vast in q en r ($a = q * b + r$ and $0 \leq r < b$)". Deze actie is een bijzonder geval van de algemenere actie: "Bepaal quotiënt en rest van twee positieve gehele getallen (de invoerparameters) en ken de resultaten toe aan twee variabelen (de uitvoerparameters)". Als deze algemenere actie, waarin dus geen namen van parameters staan, door de activering van een procedure met bijvoorbeeld de naam divmod wordt gerealiseerd, dan wordt de specifieke actie op q, r, a en b met divmod (q,r,a,b) genoteerd.
Het probleem is nu de procedure divmod te construeren (declareren). In de procesbeschrijving van de procedure worden de invoerparameters deeltal en deler genoemd, de uitvoerparameters quot en rest. Van de parameters moet aangegeven worden van welk type ze zijn, van de invoerparameter deeltal en deler bovendien (d.m.v. value) dat het gaat om de wáárden van deze parameters. De declaratie ziet er als volgt uit:

```
procedure divmod(quot, rest, deeltal, deler);
        value deeltal, deler; integer quot, rest, deeltal, deler;
        begin quot := 0; rest := deeltal;
              while rest ≥ deler
                    do rest := rest - deler; quot := quot + 1 od
        end
```

Uitvoering van de "procedure statement" divmod(q,r,a,b) in een programma als

```
begin integer a,b,q,r; procedure divmod ..... end;
      read(a); read(b);
      divmod(q,r,a,b);
      write(q); write(r)
end
```

houdt de activering in van het in de procedure declaratie beschreven proces. Deze activering bestaat uit twee stappen:

- parameterinitialisering; d.w.z. de namen deeltal en deler staan voortaan voor de wáárden van a, resp. b, en de namen quot en rest voor de variabelen q resp. r, zelf;
- uitvoering van de body: bij deze uitvoering zullen voor deeltal en deler steeds de waarden van a en b genomen worden en bij toekenning aan quot en rest zal in feite een toekenning aan q en r plaatsvinden.

Opmerking 1. Een programma kan tot op zekere hoogte als blok opgevat worden, maar ook als een procedure (zonder parameters). Het verschil is, dat een procedure geactiveerd wordt door een aanroep, terwijl een programma geactiveerd wordt door het aanbieden aan een computersysteem. Een procedure is ingebed in een voor uitvoering zorg dragend programma; een programma in een "omgeving", die voor de uitvoering van het programma zorgt. In deze omgeving zijn ook reeds genoemde (standaard) procedures gedeclareerd, die dan ook in programma's mogen worden gebruikt zonder ze zelf te declareren (zoals o.a. de veel gebruikte read en write).

Opmerking 2. De aanroep van een procedure heeft tot gevolg, dat zoals bij een blok een deelproces wordt uitgevoerd, dat een eigen toestandsruimte heeft. Na uitvoering van het deelproces bestaat deze toestandsruimte niet meer.

8.3. Routine en functieprocedures

In het voorgaande is een beeld geschetst van wat verder "routineprocedures" zullen heten. Uitvoering van een dergelijke procedure heeft tot gevolg dat uit invoerparameters de waarden voor uitvoervariabelen worden berekend volgens het in de body van de procedure vastgelegde patroon. De functie van één uitvoerparameter mag overgenomen worden door de procedurenaam. In dat geval spreekt men van een functieprocedure die overal mag worden aangeroepen op de plaatsen, waar een variabele in een expressie mag staan. De functienaam in zo'n aanroep noemt men een function designator. Een aantal veel gebruikte functieprocedures is in de meeste programmeertalen

beschikbaar met hun standaardnamen.
Daar in een functieprocedure een te berekenen waarde toegekend wordt aan de procedurenaam, moet in de declaratie ervan ook aangegeven worden van welk type deze waarde zal zijn zodat functieprocedures ook wel typed procedures genoemd worden en routineprocedures non-typed . Voorbeelden: (details van deze procedures, zoals het precieze gebruik van value en locale variabelen, zoals in c) s en seen, en het effect van aanroep, moeten nog nader toegelicht worden):

a) een procedure om de grootste van 2 getallen te bepalen:

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

Opmerking. In de body mag nu begin - end weggelaten worden omdat er maar 1 statement is.

b) een functieprocedure om de som van enkele (geheeltallig veronderstelde) elementen van een vector a te bepalen

$$\sum_{i=m}^{n} a_i$$

```
integer procedure som(a,m,n); value m,n; integer m,n; integer array a;
        begin integer s,i;
              s := 0; i := m;
              while i ≤ n do s := s + a[i]; i := i + 1 od;
              som := s
        end
```

c) een functieprocedure om na te gaan of in het array a[h:k] een element voorkomt met de waarde w:

```
boolean procedure found(a,h,k,w); value h,k,w; integer h,k,w; integer array a;
        begin integer i; boolean seen; seen := false; i := h;
              while not seen and i ≤ k do
                                          seen := a[i] = w;
                                          i := i + 1
                                    od;
              found := seen
        end
```

II. Programmeertalen

0. Inleiding

Het belangrijkste van de gebruikte ontwerptaal voor algorithmen is het verschaffen van "denkgereedschap", waarmee men

- eigen gedachten over algorithmen kort kan formuleren,
- deze gedachten aan medemensen kan overdragen,
- de correctheid van algorithmen eenvoudig kan verifiëren,
- onafhankelijk is van de structuur van rekenautomaten.

Voor de communicatie tussen mens en rekenautomaat zijn echter programmeertalen nodig, waarvan men zeker de eerste drie van de net genoemde kenmerken kan verlangen. Deze kunnen echter in strijd zijn met het vierde kenmerk zodat een compromis gevonden moet worden. De voornaamste beperkingen ten gevolge van de structuur van rekenautomaten zijn:

- het onvermogen om andere dan gelineariseerde teksten te lezen en verwerken ("alles op de regel"),
- de "bijziendheid" van rekenautomaten, die maakt dat een rekenautomaat zich op ieder moment slechts in hoofdzaak met de locale tekst kan bezig houden,
- het onvermogen om ambiguiteiten te interpreteren, waar de mens dit wel kan dank zij de context.

Programmeertalen moeten dan ook inspelen op deze beperkingen.

Iedere rekenautomaat kent uiteraard zijn eigen "machinetaal", doch deze is voor gebruik door programmeurs te gedetailleerd en omslachtig vergeleken met de nu in hoofdzaak gebruikte "hogere"programmeertalen of "procedure-talen". Na vooral Britse experimenten met het gebruik van eenvoudige arithmetische expressies en assignments in het begin van de vijftiger jaren, was omstreeks 1957 Fortran (= FORmula TRANSlation) I de eerste procedure-taal die beoogde onafhankelijk te zijn van de structuur van rekenautomaten en het gereedschap te zijn voor numerieke berekeningen. Al was die onafhankelijkheid niet volmaakt en al waren verschillende ingevoerde concepten verre van ideaal, toch evolueerde Fortran I via Fortran II tot de huidige Fortran IV. Deze evolutie behelsde niet de verbetering van onvolkomenheden, doch de toevoeging van nieuwe concepten, met name die van logische expressies en van procedures. Pas in de laatste

jaren komt de vraag op of Fortran niet een goede schoonmaakbeurt moet krijgen, doch gezien de grote investering in Fortran IV programma's loopt men daar niet erg warm voor. Voor technisch werk is Fortran IV de meest gebruikte taal.

De meeste taalontwerpfouten en de machine afhankelijkheid van Fortran zijn verwijderd in Algol 60, dat in het begin van de zestiger jaren zijn eerste standaardisatie kreeg en sindsdien slechts ondergeschikte veranderingen onderging. Algol 60 was evenals Fortran echter vooral gericht op de noden van numeriek wiskundigen; programmeerproblemen op het gebied van vertaalprogramma's en bedrijfssystemen gaven aanleiding tot het ontstaan van recente talen als Algol 68 (een zeer uitgebreide taal) en Pascal (een beperkte uitbreiding van Algol met verschillende datastructuren). Er is een langzaam, resp. snel groeiende belangstelling voor deze twee talen in de wetenschappelijke wereld.
De in de administratieve wereld levende belangstelling voor de verwerking van alfabetische teksten gaf in het begin van de zestiger jaren aanleiding tot het ontstaan van Cobol. Teneinde tegemoet te komen aan de wens om ook mensen zonder middelbare algebra kennis tot programmeur op te leiden is deze taal nogal breedsprakig en het "procedures met formele parameters" concept ontbreekt vrijwel geheel, ondanks de vele, meest kleine, wijzigingen en aanvullingen die Cobol tot nu toe ondervond. Dit neemt niet weg dat Cobol de meest gebruikte programmeertaal is omdat de meeste rekenautomaten ook voor administratief werk worden ingezet. Pogingen om met de taal PL/I (begin zeventiger jaren) de aanhangers van Fortran, Algol en Cobol te bekeren tot het gebruik van één taal voor alle typen toepassingen hebben tot dusver slechts een matig succes omdat PL/I niet alleen maar goede kwaliteiten van die talen in zich verenigt, maar ook enkele slechte.
In de volgende paragrafen worden slechts Algol 60 en enkele van de genoemde programmeertalen in min of meer detail behandeld. Naast deze talen zijn nog vele andere talen ontstaan, waarvan een groot deel op te vatten is als een dialect van de genoemde talen; daarnaast zijn echter voor speciale toepassingen andere talen ontstaan waar hier echter niet verder op ingegaan wordt.

Literatuur

Fortran: definiërend rapport ANS Fortran X 3.9 - 1966, Comm. ACM 7 (1964) 591, 14 (1971) 628;boek: D.D. McCracken - A guide to Fortran IV programming (2-nd ed.).

Algol 60:definiërend rapport: Computer Journal 19 (1976) 364-379;boek: J.J. van Amstel - Algol 60 (Kluwer 1975).

Cobol : definiërend rapport: ANS Cobol X 3.23 - 1974;boek: F. Remmen - Cobol (Academic Service 1977).

Pascal : definiërend rapport: K. Jensen,N. Wirth - Pascal (Springer 1976) boek: idem.

andere talen: J.E. Sammet - Comm. ACM 19 (1976) 655.

1. Algol 60

De behandelde ontwerptaal heeft zo veel gemeen met Algol 60 dat de verschillen snel opgesomd zijn.

1.1. Typen, constanten, variabelen, waarden, expressies

Algol 60 kent officieel de typen boolean, integer , real en string, doch onofficieel is soms ook het type complex beschikbaar. Expressies kunnen worden gevormd met de arithmetische operatoren +, -, *, **, / en div (onofficieel soms ook met mod) en de boolean operatoren not, and, or, imp en eqv. De evaluatieregels zijn zoals behandeld in het vorige hoofdstuk. Als gestructureerde variabele is slechts het array beschikbaar.

In Algol 60 is het begrip string wel ingevoerd als constante doch niet als variabele; er zijn geen bewerkingen op strings gedefinieerd. Strings kunnen zodoende alleen maar als actuele parameters van procedures (bijvoorbeeld bij read en write) gebruikt worden.

De in Algol 60 toegelaten schrijfwijze voor getallen is als volgt kort samen te vatten:

± [natgetal][. natgetal] [$_{10}$ ± natgetal]

met de volgende interpretatie:

± een van de twee tekens wordt desgewenst gekozen;

natgetal natuurlijk getal (inclusief 0);

[...] constructie tussen deze haken mag in zijn geheel ontbreken, doch tenminste één moet gekozen worden;

$_{10}$ symbool voor grondtal 10.

Toegelaten zijn dus getallen als

. 123 - 1.23 + 0123 +$_{10}$-5 - 12.3$_{10}$+ 9

maar niet"getallen"als

. 12.3 ++ 123 - 123. + .$_{10}$ - 5 - 12.3$_{10}$ + 9.8

hetgeen vrijwel aansluit bij onze normale schrijfwijze.

Opmerking. De ponscode voor $_{10}$ kan plaatselijk verschillen.

In uitbreiding aan wat in de wiskunde(en in het vorige hoofdstuk)een expressie is genoemd, kent Algol 60 de zogenaamde "conditionele" expressie (de Algol termen zijn simpele of eenvoudige expressie, resp. expressie):

if BE then E1 else E2

waarin BE een (eventueel weer conditionele) boolean expressie voorstelt, E1 een (eenvoudige!) boolean of arithmetische expressie en E2 een (eenvoudige of conditionele) boolean of arithmetische expressie. E1 en E2 moeten uiteraard beiden òf arithmetisch of boolean zijn, terwijl het else gedeelte nooit mag ontbreken omdat de expressie dan ongedefinieerd zou zijn. De waarde van de expressie wordt bepaald door de waarde van BE. Het gebruik van de conditionele expressie is alleen aan te raden voor eenvoudige gevallen van het type

if SBE then SE1 else SE2

waarin S een afkorting is voor simpele en waarin de SE's betrekkelijk korte expressies zijn (van een conditionele expressie is door deze tussen

ronde haakjes te plaatsen altijd een simpele expressie te maken).
Gebruik in een assignment als

x := if x > 0 then y else z

is dan direct te doorgronden. Hetzelfde wordt echter bereikt met

if x > 0 then x := y else x := z

De conditionele expressie is zodoende voor gebruik in assignments eigenlijk overbodig, maar soms gemakkelijk en overzichtelijk als index van een array-variabele of als actuele parameter in een procedure aanroep (in het laatste geval omdat conditionele statements niet als actuele parameters toegestaan zijn; het is echter te ondervangen door invoering van extra variabelen).

1.2. Opdrachten en besturingsstructuren

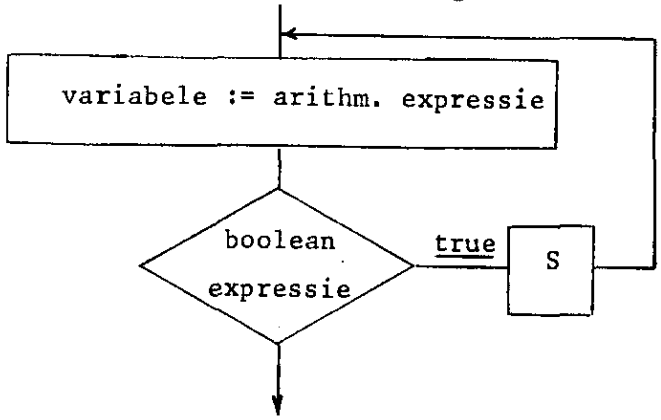
Algol kent als besturingsstructuren eveneens de concatenatie, de repetitie en het alternatief / voorwaardelijk statement (maar niet de selectie met het case-statement), doch de schrijfwijze wijkt enigszins af door het ontbreken van de afsluiters od en fi.

1.2.1. Repetitie

Het ontbreken van de od afsluiter betekent dat een andere afspraak gemaakt is om aan te geven welke groep statements achter de do uitgevoerd moet worden. Die moet tussen een begin en een end symbool geplaatst worden (en heet dan compound statement), tenzij de groep uit slechts één statement bestaat. In het laatste geval mag begin ... end achterwege blijven omdat de afsluitende puntkomma van het statement achter do tevens afsluiting van het repetitie-statement is.
Officieel bestaan de while - do en do - until statements niet in Algol, doch om hun plezierige eigenschappen zijn ze onofficieel veelal toch wel beschikbaar. Wél bestaat officieel de for - statement met de twee reeds genoemde varianten en zelfs een derde variant die niet eerder behandeld is, nl. de vorm

for variabele := arith. expressie while boolean expressie do S

waarvan de interpretatie het best met het volgende schema te verklaren is



Omdat uitvoering van statement(s) S naast de nodige beïnvloeding van de boolean expressie ook de arithmetische expressie kan beïnvloeden, is gebruik van deze variant als regel af te raden. Hooguit is deze te gebruiken, wanneer de while - do repetitie uit het vorige hoofdstuk niet beschikbaar is; een onschuldige assignment in het begin als variabele := (zelfde) variabele is dan aan te raden.
Wat achter do staat, moet altijd als een blok opgevat worden, ook al staat er maar 1 statement.

1.2.2. Alternatief / voorwaardelijk statement

Het ontbreken van de fi afsluiter in officieel Algol 60 betekent in de eerste plaats dat met begin - end haken zo nodig weer compound statements achter then en else geschreven moeten worden. Er is echter nog een andere consequentie die blijkt bij het opschrijven van

if C1 then if C2 then A else B

waarin C1 en C2 boolean expressions zijn en A en B bijvoorbeeld assignment statements. Het zou geïnterpreteerd kunnen worden als een voorwaardelijk statement waarbij na then (na C1) een alternatief statement staat (zie fig. 1), of als een alternatief statement waarbij na then (na C1) een voorwaardelijk statement staat (zie fig. 2).

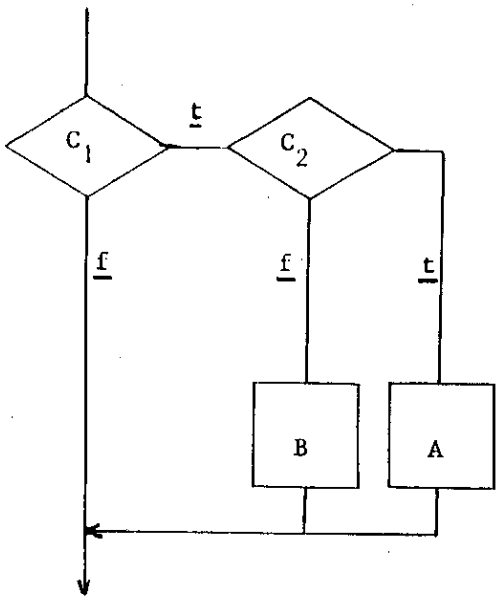


fig. 1

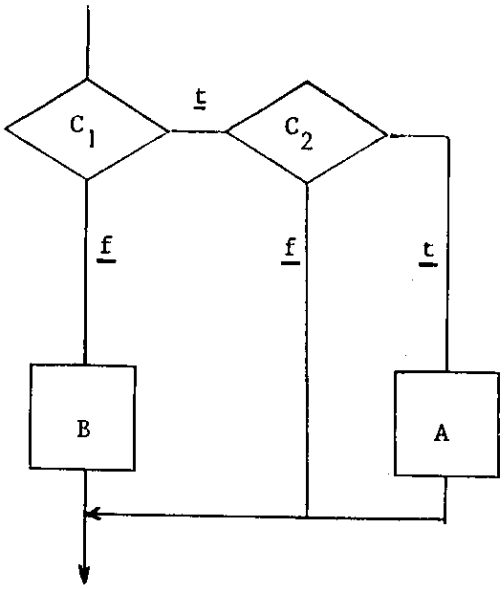


fig. 2

Deze dubbelzinnigheid wordt in Algol 60 opgelost met de volgende afspraak: then kan niet gevolgd worden door een alternatief (voorwaardelijk) statement. Als toch zo'n statement zou moeten volgen, kan men dit door omsluiten met het hakenpaar begin - end omvormen tot een altijd toegestaan compound statement. Zo kunnen de twee hierboven genoemde interpretaties in correct Algol 60 weergegeven worden met:

```
if C1 then begin if C2 then A else B end
```

resp.

```
if C1 then begin if C2 then A end else B
```

Na then mag evenmin een herhalingsopdracht staan; een constructie als

```
if C1 then while C2 do if C2 then A else B
```

is namelijk weer op twee manieren te interpreteren. Ook hier kan gebruik gemaakt worden van het hakenpaar begin - end voor het maken van een eenduidig interpreteerbare constructie.

Opmerking. Onofficieel wordt ook wel de (andere) afspraak gemaakt dat een else behoort bij de dichtstbijzijnde voorafgaande then, hetgeen het gebruik van begin - end haken soms overbodig maakt.

1.2.3. Andere sequentiëringsstatements

Via Fortran heeft Algol 60 uit de machinetalen het goto statement geërfd, waarvan het gebruik tegenwoordig niet meer aanbevolen wordt. De werking van het z.g. "sprong statement"

```
goto label
```

is dat nu de rij statements, beginnend met het statement waarvoor

label:

staat, moet worden uitgevoerd. De label dient uniek te zijn, mag met zekere beperkingen vrij gekozen worden en hoeft officieel niet gedeclareerd te worden. Sprongen van buiten naar het inwendige van een blok zijn echter verboden!
De reden dat dit statement (en hier niet besproken switches) afgeraden wordt is dat het (onoordeelkundig) gebruik gauw aanleiding geeft tot "spaghetti"-programma's waarvan de correctheid nauwelijks te verifiëren is.

1.3. Procedures in Algol 60

1.3.1. Procedure declaratie

In de declaratie van een procedure worden achtereenvolgens geschreven:

(1) procedure

bij routineprocedures of

integer procedure, real procedure of boolean procedure

bij functieprocedures.

(2) De door de programmeur zelf te kiezen naam voor de procedure.

(3) Tussen haakjes en onderling gescheiden door komma's de parameters van de procedure.

(4) Het zogenaamde value-part; hiermede wordt opgegeven welke formele parameters als invoerwaarden gebruikt worden. Uitvoerparameters mogen nóóit in het valuepart staan!

(5) De specificatie van de parameters; hiermede wordt de soort van de formele parameters opgegeven en wel met behulp van zogenaamde specifiers. Als zodanig kunnen optreden: real, integer, boolean, real array, integer array, boolean array, procedure, real procedure, integer procedure en boolean procedure.
De formele parameters dienen gespecificeerd te worden. Op te merken is ook nog, dat in de specificatie alleen de naam van de formele parameter wordt genoemd en niet bijkomende informatie (als arraygrenzen bij een array, of parameter en body bij een procedure).

(6) De body van de procedure. De body is een statement; deze statement zal vaak, zoals in twee van de drie voorbeelden uit de vorige paragraaf, de vorm van een blok hebben.

Na de parameterlijst (3), na de valuelijst (4) en na de specificaties wordt een puntkomma geschreven; bovendien worden de specificaties onderling gescheiden door een puntkomma (real a; integer m).

De variabelen die in de body gedeclareerd zijn, heten locale variabelen. In de body mogen naast formele parameters en locale variabelen ook variabelen uit de omgeving, waarin de procedure is gedeclareerd, gebruikt worden. Dit worden globale variabelen genoemd. Het gebruik van globale variabelen moet als regel afgeraden worden, omdat door de globale variabelen de procedure niet los is te maken van zijn omgeving (wat wel gaat via de parameters).

De waarde berekend voor een function designator, is de waarde die (dynamisch gezien) als laatste wordt toegekend aan de procedurenaam. De procedurenaam mag dus meermalen in de body voorkomen in het linkerlid van een assignment statement. Men had de procedure max uit de vorige paragraaf namelijk ook kunnen declareren als:

```
real procedure max2(x,y); value x,y; real x,y;
        max2 := if x > y then x else y
```

In de procedure som mag men echter niet schrijven som := som + a[i], omdat som dan in een expressie zou voorkomen, hetgeen een nieuwe aanroep van de procedure som zou betekenen.

In het vorige hoofdstuk is een strikte scheiding gesuggereerd tussen invoer- en uitvoerparameters. Het komt echter ook voor dat parameters zowel een invoer- als een uitvoerfunctie vervullen:

```
procedure wissel(a,b); real a,b;
begin real hulp;
      hulp := a; a := b; b := hulp
end
```

In de valuelijst komen van de formele invoerparameters die enkelvoudige variabelen, waarbij van de overeenkomstige actuele parameter slechts de waarde nodig is op het moment van de aanroep van de procedure (zie 1.3.2). De waarde van deze "value-parameters" (men spreekt van een call-by-value) wordt bij de aanroep vastgesteld en aan de formele parameters toegekend (zie 1.3.4). Vandaar ook dat array parameters meestal niet in de valuelijst worden opgenomen, omdat het actuele array dan in het formele array gekopieerd zou moeten worden, hetgeen extra geheugenruimte kost. Voor de niet in de value-lijst opgenomen parameters spreekt men van een call-by-name.

1.3.2. Procedure-aanroep

Een procedure kan alleen aangeroepen worden in het blok (of een binnenblok hiervan), waarin de procedure gedeclareerd is.
Een routineprocedure wordt aangeroepen door middel van een procedurestatement. Deze bestaat uit de naam van de procedure met daarachter de actuele parameters (tussen haakjes en onderling gescheiden door komma's).
Een functieprocedure wordt aangeroepen door middel van een function designator, die in een expressie voorkomt. De function designator bestaat uit een procedurenaam gevolgd door de tussen haakjes staande lijst van actuele parameters.

Bij aanroep moeten alle actuele parameters gedeclareerd zijn en behoren alle invoerparameters een waarde te bezitten. Aantal, type en volgorde van de actuele parameters moeten in overeenstemming zijn met de formele parameters en hun specificaties.
Bij de verwerking van een routine-aanroep door middel van een procedurestatement (bij routineprocedures) en een function designator (bij functieprocedures) wordt als het ware op de plaats van de aanroep in het programma de body van de procedure uitgevoerd. Daarbij moet een correspondentie gelegd worden tussen formele en actuele parameters. Hoe dit gebeurt, wordt behandeld in de volgende paragraaf.

1.3.3. Het parametermechanisme

Voordat de body van de procedure als het ware op de plaats van aanroep wordt uitgevoerd, worden achtereenvolgens de volgende handelingen uitgevoerd:

- de body van de procedure wordt ingebed in een (fictief) blok, waarin de value parameters worden gedeclareerd (volgens de overeenkomstige specificatie) en waarin deze parameters de waarden van de overeenkomstige actuele parameters wordt toegekend;
- de name parameters worden in de body vervangen door de overeenkomstige actuele parameters.

Hierna wordt het fictieve blok uitgevoerd in de omgeving (in het blok) van de aanroep. Dit laatste is van belang als er globale variabelen in de body voorkomen.

De parametervervanging legt aan de actuele parameter de eis op, dat de door bovenstaande handelingen ontstane Algol-tekst correct is (hetgeen bijvoorbeeld niet het geval zou zijn als een name parameter, die in de body in het linkerlid van een assignment statement staat, actueel overeen zou komen met een getal of een expressie!).

Als voorbeeld volgen nu voor de procedure som uit I8.3.1 het fictieve blok en de body die ontstaan bij de aanroep som (b,q,p),

```
begin integer m,n; m := p; n := q;
      begin integer s,i;
            s := 0; i := m;
            while i ≤ n do begin s := s + b[i]; i := i + 1 end;
            som := s
      end body;
end fictieve blok
```

Om het verschil tussen parameters die wel of niet in de valuelijst voorkomen nog eens te illustreren, verifiëre men voor de op het (zeer gezochte) programmadeel

```
begin integer i,z; integer array a[1 : 2];
      procedure P(s,t,u); value ...; integer s,t,u;
      begin i := i + 1; s := s + 1; u := s * t end;
      i := 1; a[1] := 10; a[2] := 12; z := 8;
```

volgende verschillende oproepen van P het hierna opgegeven effect:

oproep	in valuelijst van P	effect i	effect z	opmerking
P(a[i],i,z)	s,t	2	11	
	s	2	22	
	t	2	13	
	-	2	26	
P(i,a[i],z)	s,t	2	10	
	s	2	24	
	t	3	30	
	-	3	-	a[3] is niet gedefinieerd; aanroep niet toegestaan
P(a[i],i,5)				niet toegestane uitvoerparameter
P(4,i,2)	s,t	2	5	
	s	2	10	
	-	2	-	getal als name-parameter niet toegestaan

1.3.4. Jensen's device

Stel dat de volgende procedure gedeclareerd is:

```
real procedure sigma(var,first,last,term);
                value first, last; real term; integer var, first, last;
begin real h;
      h := 0;
      for var := first step 1 until last do h := h + term;
      sigma := h
end
```

Op het eerste gezicht lijkt het alsof deze procedure (last - first + 1) * term berekent (tenminste het actuele equivalent hiervan). Als men echter precies de regels van de vorige paragraaf toepast op de aanroep

```
sigma(n,1,5n + 2)
```

dan ziet men dat door deze aanroep wordt berekend $\sum_{n=1}^{5} n^2$ en door de aanroep

```
sigma(k,10,30,a[k])
```

de waarde wordt berekend van $\sum_{k=10}^{30} a[k]$. Dit komt doordat twee actuele parameter (waarvan de overeenkomstige formele parameters niet in de valuelijst voorkomen) met elkaar samenhangen.

1.3.5. Recursie

In 1.3.1 is voor de procedure som gesteld dat de procedurenaam van een functieprocedure niet in het rechterlid van een assignment statement mag staan, omdat dit een nieuwe aanroep van de procedure inhoudt. Dit laatste was in die procedure niet de bedoeling, maar in het algemeen is het wel toegestaan dat in een procedure een aanroep van de procedure zelf voorkomt. Men spreekt in zo'n geval van een recursieve procedure.
Wanneer is de recursiviteit zinvol te gebruiken? Men kan natuurlijk elk probleem dat recursief gedefinieerd is met behulp van een recursieve procedure oplossen.
Als men als definitie van n! neemt dat n! gelijk is aan n * (n - 1)! voor $n \geq 1$ en gelijk is aan 1 voor $n \leq 0$ dan kan men voor n! als procedure schrijven:

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

Men kan de probleemstelling echter gemakkelijk niet-recursief geven door te definiëren dat n! gelijk is aan 1 * 2 * 3 * ..... * (n - 1) * n voor n ≥ 1 en gelijk is aan 1 voor n ≤ 0. Men kan n! dan berekenen met behulp van:

```
nfac := 1;
for k := 1 step 1 until n do nfac := nfac * k
```

Het volgende voorbeeld laat echter zien dat er problemen zijn waarbij de omzetting van een recursieve oplossingsmethode naar een repetitieve oplossingsmethode niet (zo eenvoudig) gaat.
Stel dat een rij positieve getallen, afgesloten door -1, ingelezen en in omgekeerde volgorde afgedrukt moet worden zonder van een array gebruik te maken. Dit probleem is op te lossen met behulp van het volgende programma:

```
begin procedure printrij;
      begin integer a; read(a);
            if a ≠ - 1 then begin printrij; write(a) end
      end;
      printrij;
end
```

Tenslotte toont het volgende voorbeeld hoe een ander reeds behandeld iteratief proces ook recursief op te schrijven is:

```
integer procedure ggd (m, n); value m, n; integer m, n;
        ggd : = if n > m then ggd (n, m)
                        else if n = 0 then m else ggd (n, m mod n);
```

1.3.6. Voorbeelden

Als toelichting op het voorgaande verschillende functieprocedures om met behulp van het Hornerschema de waarde van een polynoom zoals

$$a_0x^n + a_1x^{n-1} + \ldots\ldots + a_n \; [= ((\ldots\ldots(a_0 * x + a_1) * x + a_2)\ldots)* x + a_n]$$

te berekenen voor gegeven a_i (opgeslagen in een array a[0 : n]) en x-waarden.

a)
```
real procedure pol1;
    begin integer i; real h; i := 0; h := 0;
        while i ≤ n do begin h := x * h + a[i]; i := i + 1 end;
        pol1 := h
    end
```

Deze procedure krijgt zijn informatie via globale variabelen, die uiteraard een waarde moeten hebben gekregen voordat deze functieprocedure aangeroepen wordt met pol1. Door de globale variabelen heeft deze procedure slechts een beperkt nut.

b)
```
real procedure pol2(aa,m,xx); value m,xx; real xx; real array aa; integer m;
    begin integer i; real h; i := 0; h := 0;
        while i ≤ m do begin h := xx * h + aa[i]; i := i + 1 end;
        pol2 := h
    end
```

Deze procedure krijgt met de aanroep pol2(a,n,x) zijn informatie via formele parameters, wat het mogelijk maakt om deze procedure voor vele rijen (polynoom-) coëfficiënten en vele x-en te gebruiken.

c)
```
real procedure pol3(aa,i,m,xx); value m,xx; real aa,xx; integer i,m;
    begin real h; i := 0;h := 0;
        while i ≤ m do begin h := xx * h + aa; i := i + 1 end
        pol3 := h
    end
```

In deze procedure wordt Jensen's device gebruikt bij de aanroep pol 3(a[k],k,n,x).

d)
```
real procedure pol4(aa,m,xx); value m,xx; real xx; real array aa; integer m;
    if m > 0 then pol4 := xx * pol4(aa,m-1,xx) + aa[m]
             else pol4 := aa[m]
```

In deze procedure wordt recursie gebruikt bij de aanroep pol4(a,n,x).

1.3.7. Wanneer en waarom procedures?

In de praktijk, waar men te maken heeft met veel grotere programma's dan hier ten tonele kunnen worden gevoerd, vormen procedures bij uitstek hèt hulpmiddel om een groot programma overzichtelijk te houden. Bovendien kan men procedures ook los van de berekening waarin ze gebruikt worden, onderzoeken op hun correctheid hetgeen ook het correctheidsbewijs voor een groot programma aanzienlijk vereenvoudigt. Deeltaken kunnen zo afzonderlijk opgelost worden.

Bij het isoleren van afzonderlijke procedures kan men dan ook beter geen globale variabelen in de procedurebody laten staan, maar moet men deze bij voorkeur vervangen door formele parameters. Dit brengt wel het gebruik van iets meer parameters met zich mee, maar verhoogt de flexibiliteit en voorkomt eventuele narigheid door onverwachte neveneffecten (vergelijk de verschillende hiervoor behandelde proceduredeclaraties voor de berekening van een polynoomwaarde).

Veel eerder zijn reeds de standaard procedures genoemd. Men moet zich voorstellen dat deze reeds gedeclareerd zijn in een blok dat de programmatekst van een gebruiker omvat, zodat ze zonder meer in die programmatekst aangeroepen kunnen worden. Daarnaast beschikt ieder rekencentrum over een "bibliotheek" van procedures, voor allerlei standaard berekeningen uit de numerieke wiskunde en statistiek. Het gebruik van deze procedurebibliotheek bespaart zeer veel tijd bij de oplossing van praktijkproblemen.

Eigenlijk wel het belangrijkste aspect van procedures is dat zij een uitbreiding geven van het arsenaal van bouwelementen van een programma. Simpele bouwelementen in Algol zijn de verschillende datatypen en verder de assignment, while, if - then - else statements, doch met behulp van procedures kan iedereen op een wijze die hem het beste bevalt een uitbreiding geven aan deze standaard statements. (Uiteraard kan hij ze alleen in zijn eigen programma's gebruiken, zolang ze niet in een procedurebibliotheek opgenomen zijn).

1.3.8. Definitie van Algol 60

In het voorgaande zijn slechts enkele belangrijke facetten van Algol behandeld en dan op informele wijze. Voor een precieze definitie van de syntaxis van Algol zij verwezen naar Appendix A.

Appendix A

Syntaxis van Algol

1. Inleiding

Algol 60 was de eerste programmeertaal waarvoor een bevredigend strenge definitie werd gegeven voor de

- syntaxis; d.i. de beschrijving van de in de taal toegelaten constructies (de grammatica van de taal zou men kunnen zeggen)
- semantiek; d.i. de "betekenis" van die toegelaten constructies in de zin van het resultaat bij uitvoering van het programma.

Voor de semantische beschrijving werd nog gebruik gemaakt van onze gewone spreektaal, doch voor de syntactische beschrijving werd een zogenaamde meta-taal ingevoerd. Deze BNF notatie (als afkorting van Backus Naur Form naar de namen van twee bij de ontwikkeling van Algol betrokken onderzoekers) maakt gebruik van slechts enkele taalbeschrijvings- of "meta"-symbolen (die zelf niet tot de te beschrijven taal behoren), te weten

:: =	voor "bestaat uit" of "gedefinieerd als"
|	voor "(niet uitsluitende) of"
<...>	de omsluitende "metahaken", waartussen de te definiëren grootheid staat.

Voorbeeld

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

<cijfer duo> :: = (<cijfer>, <cijfer>)

<natuurlijk getal> :: = <cijfer> | <natuurlijk getal> <cijfer>

Met behulp van deze definities is te verifiëren dat (9,8), resp. 981 voorbeelden van cijferduo, resp. natuurlijk getal zijn, doch 9.8 en 98.1 niet. Merk op dat de derde definitie een zogenaamde recursieve definitie is omdat rechts weer de linksstaande te definiëren grootheid staat. Een "cirkel-

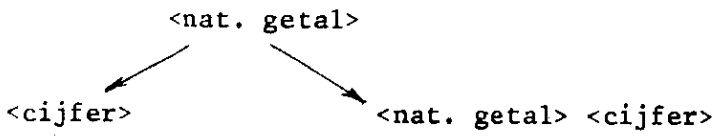
definitie" is het echter niet omdat rechts ook het alternatief <cijfer> genoemd is. Men zou deze recursieve definitie daarom ook kunnen lezen als een verkorte schrijfwijze voor

<cijfer> | <cijfer> <cijfer> | <cijfer> <cijfer> <cijfer> | ... ...

Volgens de definitie is ook 00981 een natuurlijk getal, hoewel men in het dagelijkse leven gewend is om "vooropstaande nullen" (leading zeroes) weg te laten.
In latere jaren zijn nog andere representaties voor syntax definities ontwikkeld, vooral om deze meer aanschouwelijk te maken. Genoemd wordt slechts:

- de syntaxgraaf methode

Voorbeeld



- de stroomdiagram methode

Voorbeeld

Opmerking Nooit tegen de pijlrichting in gaan!

In de volgende paragrafen zal men echter slechts de BNF notatie en de stroomdiagram methode ontmoeten. Voor andere talen (waarin met name het aantal herhalingen in een recursieve definitie sterk beperkt is) dan Algol moet men veelal nog andere metasymbolen invoeren (voor Cobol bijvoorbeeld accoladen om grootheden waaruit één keus gemaakt moet worden; rechte haken om grootheden die 0 of 1 keer mogen voorkomen). Een verdienste van Algol is tenslotte nog dat spaties geen betekenis hebben en dat Algol (dank zij zijn onderstrepingssymboliek) geen keywords kent, dat wil zeggen dat bepaalde woorden als begin en end rustig als identifier gebruikt mogen worden (niet dat dit raadzaam is!).

In het volgende zal men slechts de syntaxis van de in dit college in detail behandelde Algolconstructies tegenkomen; voor een volledige behandeling zij verwezen naar het zogenaamde Revised Report, oorspronkelijk gepubliceerd in o.a. Numerische Mathematik 4 (1963) 420 en C ACM 6 (1963) 1. Om deze syntaxis zo duidelijk en compact mogelijk weer te geven wordt daarbij gedeeltelijk BNF gebruikt, gedeeltelijk de stroomdiagram-notatie. De gebruikte terminologie sluit wel aan bij de syllabus, doch niet in alle details bij het Revised Report. Om ruimte te sparen worden soms afkortingen gebruikt, bijvoorbeeld ident. voor identifier, nat. getal voor natuurlijk getal, enz. In sommige gevallen worden geen formele definities gegeven, maar een informele tussen aanhalingstekens (namelijk wanneer de formele definitie veel schrijf- en leeswerk zou betekenen en de informele definitie niet tot misverstanden kan leiden).

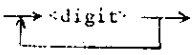
2. Syntaxis van Syllabus Algol

1. Elementaire bouwstenen

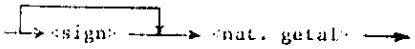
<empty> :: = (d.i. een onbepaald aantal spaties)
<letter> :: = "een van de kleine letters of hoofdletters van een alfabet"
<digit> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<boolean value> :: = true | false
<sign> :: = + | -
<type> :: = integer | real | boolean
<string> :: = "een tussen stringhaken geplaatste rij van letters, cijfers, leestekens en spaties"
<relational operator> :: = < | ≤ | = | ≠ | > | ≥

2. Getallen, identifiers, declaraties en specificaties

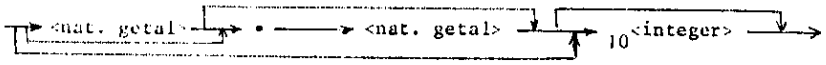
<nat. getal>



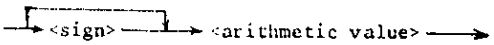
<integer>



<arithmetic value>



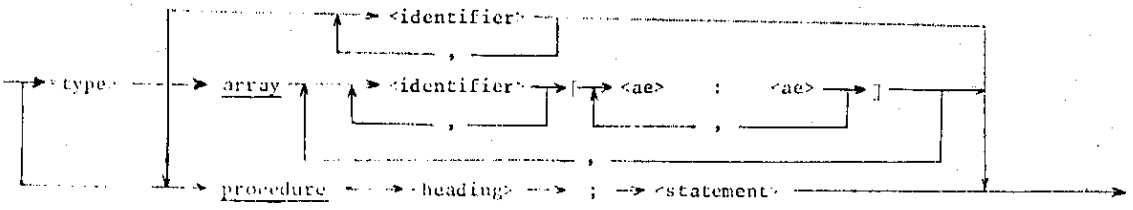
<number>



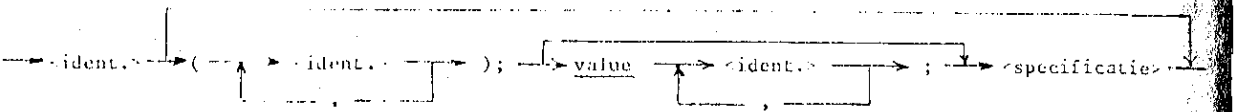
<comparison> ::= <arithmetic expression> <relational operator> <arithmetic expression>

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

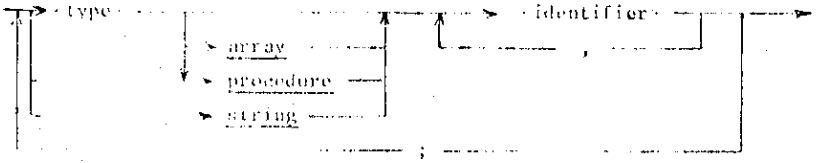
<declaratie>



<heading>



<specificatie>



2.3. Variabelen, parameters, function designators

<variable> → <identifier> [→ <arithmetic expression> (, <arithmetic expression>)*] →

<parameter> :: = <identifier> | <conditional expression> | <string>

<function designator> → <identifier> (→ <parameter> (, <parameter>)*) →

2.4. Expressions

<primary> :: = <value> | <variable> | <function designator> | (<conditional expression>)

Opmerkingen. Waar nodig moet op de juiste plaatsen óf arithmetic óf boolean gelezen worden!
Als boolean primary mag ook een comparison optreden.

<arithmetic factor> → <arithmetic primary> (↑ <arithmetic primary>)* →

<arithmetic term> → <arithmetic factor> (* | / | div <arithmetic factor>)* →

<arithmetic expression> → (<sign>) → <arithmetic term> (+ | - <arithmetic term>)* →

<boolean factor> → (not)* → <boolean primary> (and <boolean primary>)* →

<boolean term> → <boolean factor> (or <boolean factor>)* →

<implicatie> → <boolean term> (impl <boolean term>)* →

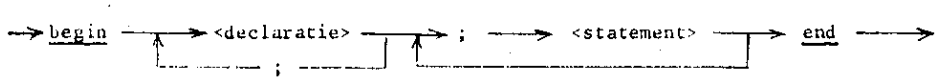
<boolean expression> → <implicatie> (equiv <implicatie>)* →

<conditional expression> → if <boolean expression> then <expression> else <conditional expression> →
<expression> →

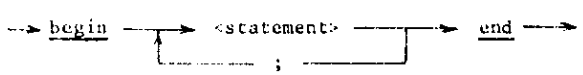
2.5. Program, block, statement

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

<block>



<compound statement>

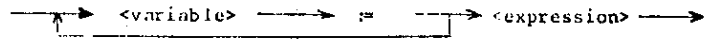


<statement> :: = <unconditional statement> | <conditional statement> | <loop statement>

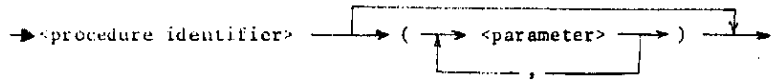
<unconditional statement> :: = <basic statement> | <compound statement> | <block>

<basic statement> :: = <assignment statement> | <procedure statement> | <dummy statement>

<assignment statement>

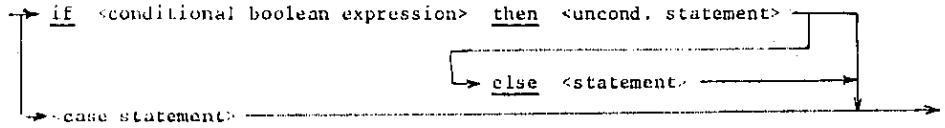


<procedure statement>



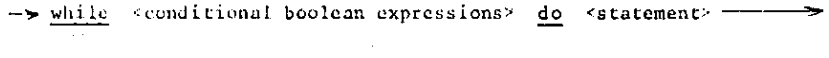
<dummy statement> :: =

<conditional statement>

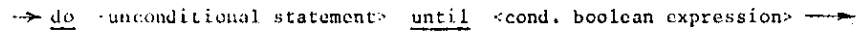


<loop statement> :: = while statement | do statement | for statement

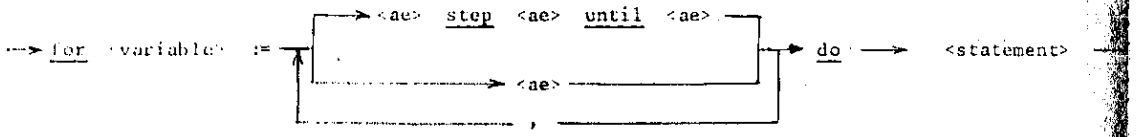
<while statement>



<do statement>



<for statement>



Appendix B

De volgende paragrafen over algorithmen en rekenautomaten zijn ontleend aan een oude syllabus van prof. dr. E.W. Dijkstra over algorithmen.

1. Algorithmen

In het begin van de negende eeuw na Christus werkte in de bibliotheek van het toen pas gestichte Bagdad een zekere Mohammed 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 zogenaamde Indische rekenen, dat wil zeggen het rekenen in het tientallig stelsel. Enige eeuwen 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 "algoritme", dat sedert die tijd in zwang is voor "rekenvoorschrift", in het algemeen voor "handelingsvoorschrift".

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

Aan deze alledaagse algoritmen kunnen wij al een aantal typische eigenschappen illustreren. Essentieel is, dat een of andere totale verrichting ontleed wordt, 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 van ragout zal men vinden, dat een niet te grote ui gesnipperd en bruin gefruit moet worden, etc.

Verder observeren wij, dat hele klassen algoritmen herleid worden tot handelingen uit het repertoire, dat duidelijk bij die klasse hoort, waarbij als regel elke handeling uit het repertoire in allerlei algoritmen voorkomt. Zo zal het deelvoorschrift om een uitje te fruiten als onderdeel in allerlei recepten voorkomen, zo komen typische breihandelingen (zoveel steken zus, zoveel steken zo, minderen etc.) in allerlei breipatronen voor.

Vervolgens zien wij, dat een algoritme 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 algoritmen; 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 algoritmen dan ook zelden zonder dubbelzinnigheid uitvoerbaar.
Tenslotte moet een algoritme 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! Bij het naar de weg vragen is een antwoord als "Rechtuit en dan bij de laatste stoplichten linksaf " niet te gebruiken.

Hebben wiskundigen zich dus al sinds eeuwen met algoritmen beziggehouden, in de twintigste eeuw zijn algoritmen opnieuw in het middelpunt van de wiskundige belangstelling komen 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 algoritmen tot voorwerp van studie maakte is

niet zo verwonderlijk als we bedenken, dat een bewijs een bepaald soort handelingsvoorschrift is, namelijk 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 algoritmen te concentreren kwam door de ontwikkeling van de electronische rekenautomaten (ook wel computers of rekenmachines genaamd). Deze machines zijn namelijk in staat om algoritmen (van een bepaalde klasse) in zich op te nemen en vervolgens getrouwelijk uit te voeren. Rekenautomaten zijn dus machines, die algoritmen kunnen uitvoeren; voor algoritmen, 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 algoritmen drastisch veranderd: van een uiterst theoretische belangstelling is het, door de grote toepassingsmogelijkheden van rekenautomaten en hun economische importantie geworden tot een buitengewone practische aangelegenheid. Bovendien, was aanvankelijk de belangstelling voor algoritmen in hoofdzaak analytisch, nu is de belangstelling in hoge mate synthetisch: er moeten algoritmen van allerlei soort gemaakt worden, willen wij de mogelijkheden van rekenautomaten kunnen benutten.

2. Rekenautomaten

In deze paragraaf zullen we 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.

Te noemen zijn de roltrappen van een voetgangerstunnel. '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 roltrap zich in beweging en rolt hij (ruim) eenmaal zijn eigen lengte af. Wie zich dan daardoor 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 (namelijk de onderbreking van de straal) het mechanisme automatisch een vaste handeling verricht, autonoom "een vast programma" afwerkt, namelijk 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 bijvoorbeeld 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 zogenaamde teller, een geheugenelement, waarin genoteerd staat het aantal treden, waarover de trap nog moet rollen. 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 beïnvloeding 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 beïnvloeding 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 het besturingsmechanisme van een 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, namelijk een standaard reactie op een uniek startsignaal. Van de moderne rekenautomaat verlangt men, dat hij allerlei rekenprocessen kan uitvoeren, allerlei algoritmen 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 (dat wil zeggen de aan- respectievelijk 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, op hun gebied, tamelijk algemeen! Voor het weefgetouw van Jacquard betekent dit, dat het allerlei patroontjes kan weven, van Franse lelies tot familiewapens toe en juist door die algemeenheid door het boek gestuurd moeten 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 algoritmen 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.

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 (bekend van de postgiro) dan wel in ponsband (sinds lang bekend bij het telexverkeer). De rekenmachine is uitgevoerd met een zogenaamde "lezer" (kaartlezer of ponsbandlezer), waardoor de informatie in een pak kaarten dan wel een ponsband kan worden afgetast. Geschiedt dit bij het weefgetouw mechanisch (namelijk 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 orgel beschikken niet over

een groot geheugen (dat wil zeggen 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 bijvoorbeeld 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 algoritme op te nemen, voordat aan de werkelijke uitvoering ervan begonnen wordt. In een programma kunnen we, zoals we later zullen zien, wel aangeven, dat bijvoorbeeld 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 een derde onderdeel van de machine een belangrijke rol, het zogenaamde "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, namelijk het aan de buitenwereld terugmelden van de gevraagde uitkomsten. Dit kan via bandponsers, kaartponsers, regeldrukkers, tekentafels, kathodestraalbuizen 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 prettige terminologie geweest zijn. In het Engels spreekt men van "Input", "Output" en "Execution".)

Appendix C

1. Gegeven is een band met vier getallen a,b,c en d. Maak een programma dat het grootste van deze vier getallen print.

2. Op een band staat een geheel getal n. Maak een programma dat het grootste achtvoud, niet groter dan n, berekent en print.

3. Gegeven is een geheel getal n op de band: $0 \leq n < 1024$.
Bepaal op hoeveel nullen de decimale schrijfwijze van $n!$ eindigt.

4. Gegeven is een band met acht gehele getallen: $x_1, y_1, \ldots, x_4, y_4$.
Een paar (x_i, y_i) stelt de coördinaten van een punt in het platte vlak voor. De getallen op de band zijn zodanig dat geen drietal punten op één rechte ligt.
Maak een programma dat een 0 print indien geen der punten gelegen is binnen de driehoek gevormd door de overige drie punten, en dat een 1 print in alle overige gevallen.

5. Gegeven is een band met twee gehele getallen p en q: $q \neq 0$.
Print eerst het kleinste gehele getal groter of gelijk $\frac{p}{q}$, en daarna het grootste gehele getal kleiner dan $\frac{p}{q}$.

6. Op een band staat een natuurlijk getal n: $2 \leq n \leq 200$. Maak een programma dat een 1 print indien n priem is, en dat een 0 print indien n geen priem is.

7. Een band bevat een natuurlijk getal m: $m \leq 100$.
Een wiel met straal 1 rolt, beginnend bij de oorsprong, over de positieve reële getallenrechte in de richting van oneindig. Maak een programma dat bepaalt hoeveel volledige omwentelingen het wiel heeft gemaakt op het moment dat het punt m aandoet.
Werkt uw programma ook indien $m > 100$?

8. Op een band staan drie gehele getallen. Bepaal hoeveel van deze drie getallen kleiner of gelijk hun gemiddelde zijn.

9. Op een band staat een natuurlijk getal niet groter dan 100. Bepaal de cijfers uit de zeventallige representatie van dat getal.

10. Gegeven is een geheel getal n op een band: $n \geqslant 0$. Maak een programma voor de berekening van n! .

11. Een band bevat 1000 getallen. Bepaal de som van deze getallen.

12. Op een band staan 700 getallen. Bepaal hun product.

13. Gegeven zijn twee gehele getallen k en n op de band:: $0 \leqslant k \leqslant n$. Maak een programma voor de berekening van $\frac{n!}{k!}$.

14. Gegeven zijn twee gehele getallen k en n op de band: $0 \leqslant k \leqslant n$. Schrijf een programma voor de berekening van $\binom{n}{k}$.

15. Maak een programma voor de bepaling van $\sum_{k=0}^{n} \binom{n}{k}$, waarbij n een geheel getal van de band is: $n \geqslant 0$.

16. Schrijf een programma dat voor elke $0 \leqslant i \leqslant 30$ de waarde van 2^i berekent en print.

17. Bereken en print $\sum_{n=1}^{100} \frac{(-1)^n}{n}$.

18. Bereken en print $\sum_{n=0}^{25} \frac{2^n}{n!}$.

19. De rij van Fibonacci $a_0, a_1, a_2, \ldots$ is gedefinieerd door:
$a_0 = 0$, $a_1 = 1$, en $a_i = a_{i-1} + a_{i-2}$ voor $i \geqslant 2$.
Op de band staat een geheel getal n, $n \geqslant 0$. Maak een programma voor de berekening van a_n.

20. Een band bevat het natuurlijke getal n. Bepaal de kleinste geheeltallige waarde van k waarvoor $2^k \geqslant n$.

21. Bereken en print de grootste gehele waarde van n waarvoor $1.57^n + 4.2^n < 5000$, en print de waarde van $1.57^n + 4.2^n$ voor die waarde van n.

22. Een band is van de structuur: $a_0, a_1, \ldots, a_{n-1}, 0$ met alle $a_i > 0$. Bepaal de som der a_i's.

23. Een band met gehele getallen heeft de structuur: $n, m_0, \ldots, m_n$.
Hierbij is $n \geq 2$, en voor elke $0 \leq i < n$: $m_{i+1} - m_i$ even en positief.
Print de rij $g_0, \ldots, g_{n-1}$ die bepaald is door $g_i = \frac{m_{i+1} + m_i}{2}$ voor $0 \leq i < n$.

24. Gegeven is een band met daarop een onbekend aantal, zeg k, natuurlijke getallen $a_0, \ldots, a_{k-1}$, afgesloten door een 0. $(k \geq 0)$.
Bepaal $\sum_{i=0}^{k-1} (k - i)\, a_i$.

25. Een getallenband bevat louter gehele getallen en is van de vorm: $x, n, a_0, a_1, \ldots, a_{n-1}$, met $n \geq 0$. Schrijf een programma dat een 1 print indien een i, $0 \leq i < n$, bestaat waarvoor $x = a_i$, en dat een 0 print in de overige gevallen.

26. Bepaal $\sum_{n=0}^{N} n^2$. Het niet-negatieve gehele getal N staat op de band. Uw programma mag hierbij geen gebruik maken van vermenigvuldiging, deling of machtsverheffing.

27. Gegeven is een band met de getallen a en v: $a < 0$, $v \geq 0$.
Vanuit een der hoekpunten van een rechthoek van 1 bij 2 meter schiet men onder een hoek van 60° met de lange zijde een punt af met een beginsnelheid van v $^m/_{sec}$. Tijdens de beweging ondergaat het punt een eenparige versnelling van a $^m/_{sec^2}$. Als verder de hoek van terugkaatsing gelijk is aan de hoek van inval, er bij botsing geen energie verloren gaat, etc., bereken dan de plaats in de rechthoek waar het punt tot stilstand komt.

28. Op een band staat het natuurlijke getal n, gevolgd door n gehele getallen $a_0, \ldots, a_{n-1}$. Schrijf een programma dat het maximum van de getallen $a_0, \ldots, a_{n-1}$ bepaalt en tevens de kleinste index waarvoor het maximum wordt aangenomen.

29. Gegeven is een band met 100 gehele getallen, $a_1, \ldots, a_{100}$.
Maak een programma dat eerst alle even a_i's print, en daarna alle oneven a_i's.

30. Gegeven is een natuurlijk getal n op de band. Bepaal het langste rijtje van opeenvolgende natuurlijke getallen waarvan de som gelijk is aan n.

31. Gegeven is een natuurlijk getal n op de band. Bepaal de kleinste waarde van a + b, met a en b natuurlijk, waarvoor $a*b \geqslant n$ is.

32. Op een band staat het natuurlijke getal n, gevolgd door n gehele getallen $a_0,\ldots,a_{n-1}$. Maak een programma dat een 1 print indien de rij $a_0,\ldots,a_{n-1}$ monotoon dalend is, en dat een 0 print in de overige gevallen.

33. Op een band staat een geheel getal n, $n \geqslant 0$. Maak een programma dat onderzoekt of de decimale voorstelling van n een 7 bevat.

34. Gegeven is een natuurlijk getal a op de band. Bepaal het grootste natuurlijke getal b waarvan de cijfers in de decimale schrijfwijze een rangschikking zijn van die uit de decimale schrijfwijze van a.

35. Schrijf een programma voor de berekening van

$$\sum_{n=1}^{100} \sum_{k=0}^{n} \frac{1+\sqrt{n^2+k^2}}{k^2+\sqrt{n^2+k^2}} .$$

36. Op een band staat het natuurlijke getal n, gevolgd door n gehele getallen $a_0,\ldots,a_{n-1}$.
Bereken $\max\limits_{i,j} |a_i - a_j|$.

37. Op een band staat een monotoon niet-dalende rij gehele getallen ter lengte 700. Maak een programma voor de bepaling van het aantal verschillende waarden in de rij.

38. Een band bevat een rij bestaande uit 700 gehele getallen. Schrijf een programma ter bepaling van het aantal verschillende waarden in de rij.

39. Op een band staat het natuurlijke getal n. Hoeveel verschillende delers bezit n?

40. Op een band staat een onbekend aantal positieve getallen, gevolgd door het getal -1. Bereken de som van het eerste, derde, vijfde, zevende, etc. positieve getal.

41. Stel vast of een natuurlijk getal n op de band priem is.

42. Op een band staan de natuurlijke getallen $n, a_0, \ldots, a_{n-1}, g$.
Er geldt: $a_0 < a_1 < \ldots < a_{n-1}$.
Gevraagd wordt alle paren i,j af te drukken waarvoor geldt $a_j - a_i = g$.

43. Gegeven is een geheel getal n op de band. Maak een programma dat een 0 afdrukt wanneer n even is en een 1 wanneer n oneven is. De restrictie hierbij is dat optellen en aftrekken de enige toegelaten rekenkundige operaties zijn. Draag zorg voor een efficiënt programma.

44. Op een band staan de gehele getallen $a_0, a_1, \ldots, a_{99}, x$. Er is gegeven dat $a_0 \leqslant a_1 \leqslant \ldots \leqslant a_{99}$. Print een 1 als x voorkomt in de rij $a_0, \ldots, a_{99}$, en print een 0 indien dit niet het geval is.

45. Een band bevat de gehele getallen $n, a_0, \ldots, a_{n-1}$ met $n > 0$.
Print een 1 indien de rij $a_0, \ldots, a_{n-1}$ een palindroom is, d.w.z. $a_i = a_{n-i-1}$ voor alle $0 \leqslant i < n$, en een 0 in andere gevallen.

45'. Gegeven is een band met 1000 natuurlijke getallen. Niet alle getallen van de band zijn onderling gelijk. Bereken het op één na grootste getal op de band, en het aantal malen dat dit voorkomt.

46. Gegeven is een band met 1000 gehele getallen: $a_0, \ldots, a_{999}$. De rij $a_0, \ldots, a_{999}$ is monotoon niet-dalend, en $a_0 \neq a_{999}$. Bereken het op één na grootste getal uit de rij.

47. Een band bevat 1000 gehele getallen: $a_0, \ldots, a_{999}$.
Aan elk getal a_i kennen wij als volgt een gewicht g_i toe:
voor $0 \leqslant i < 999$: $g_i = 0$ als $a_i < a_{i+1}$, $g_i = g_{i+1} + 1$ als $a_i \geqslant a_{i+1}$
voor $i = 999$: $g_i = 0$.
Bereken $\sum_{i=0}^{999} g_i a_i$.

48. Op een band staat een onbekend aantal positieve getallen, gevolgd door het getal -1. Print de som van het $1^e, 3^e, 6^e, 10^e, 15^e, 21^e$, etc. positieve getal.

49. Een band bevat 200 gehele getallen: $a_0, \ldots, a_{99}, b_0, \ldots, b_{99}$.
Gegeven is dat zowel de a-rij als de b-rij monotoon niet-dalend is.
Print een 1 indien er een getal bestaat dat zowel in de a-rij als in de b-rij voorkomt, en een 0 indien dit niet zo is.

50. Schrijf een programma ter berekening van de coëfficiënten van het polynoom $(x + 1)^n$. Hierbij is n een niet-negatief geheel getal, dat op de band staat.

51. Gegeven is een band van de structuur: $n, g_0, g_1, \ldots, g_{n-1}$. Er geldt dat alle getallen geheel zijn, n tenminste gelijk 2 is, en de g-rij monotoon niet-dalend. Maak een programma dat het aantal rijen $m_0 \leqslant \ldots \leqslant m_n$ bepaalt, waarvoor geldt:
m_i geheel voor $0 \leqslant i \leqslant n$, en $g_i = \frac{m_i + m_{i+1}}{2}$ voor alle $0 \leqslant i < n$.

52. Een band bevat 200 getallen: $a_0, b_0, a_1, b_1, \ldots, a_{99}, b_{99}$.
Er geldt:
$a_i \geqslant 0$, $b_i > 0$ voor elke $0 \leqslant i < 100$, en $\sum_{i=0}^{99} a_i = \sum_{i=0}^{99} b_i$.
Rondom een circuit bevinden zich 100 honken, al rondgaande genummerd van 0 t/m 99. Bij honk i is een hoeveelheid benzine a_i aanwezig, terwijl de hoeveelheid benzine benodigd om vanuit i het volgende honk te kunnen bereiken b_i is.
Maak een programma dat alle honken bepaalt van waaruit een auto met een aanvankelijk lege (voldoend grote) benzinetank het circuit geheel kan rondkomen.

53. Gegeven is een band van de structuur: $n, a_0, \ldots, a_{n-1}$. Hierbij is n geheel en niet-negatief. Maak een programma dat al die elementen a_i print, die geen buur hebben waarvan de waarde gelijk is aan die van a_i.

54. Een museum wordt bezocht door 100 personen, genaamd 0 t/m 99. Persoon i betreedt het museum op tijdstip a_i, en verlaat het weer op tijdstip b_i.
De gehele getallen a_i en b_i staan als volgt op de band:
$a_0, a_1, \ldots, a_{99}, b_0, b_1, \ldots, b_{99}$.
Er geldt dat de a-rij monotoon niet-dalend is, en dat $a_i \leqslant b_i$ voor alle $0 \leqslant i < 100$.
Schrijf een programma dat bepaalt gedurende welke hoeveelheid tijd het museum niet leeg is.

55. Schrijf een programma dat de coëfficiënten c_n van het productpolynoom $P(x).Q(x)$ berekent en print; hierbij is
$P(x) = \sum_{i=0}^{p} a_i x^i$ en $Q(x) = \sum_{i=0}^{q} b_i x^i$.
De getallen $p, a_0, \ldots, a_p, q, b_0, \ldots, b_q$ staan in deze volgorde op de band (p en q geheel en niet-negatief).

56. Op een band staat eerst het natuurlijke getal $n \geq 3$, en daarna $2n$ gehele getallen $x_0, y_0, \ldots, x_{n-1}, y_{n-1}$.
Het paar (x_i, y_i) geeft de rechthoekige coördinaten van een punt P_i in het platte vlak. Gegeven is dat de punten $P_0, \ldots, P_{n-1}$ in deze volgorde de rechtsomgaande contour van een convexe n-hoek vormen.
(Een convexe veelhoek is een veelhoek met de eigenschap dat alle hoeken van de veelhoek ten hoogste 180° zijn.)
Toon aan dat de oppervlakte van de convexe n-hoek een $\frac{1}{2}$-voud is, en schrijf vervolgens een programma dat bepaalt hoeveel $\frac{1}{2}$-vouden de oppervlakte bedraagt.

57. Bereken de waarde van de veelterm $y = a_0 x^n + a_1 x^{n-1} + \ldots + a_n$ met behulp van het Horner-schema:
als $b_0 = a_0$, en $b_j = a_j + b_{j-1} x$ voor $1 \leq j \leq n$, dan is $b_n = y$.
De getallen $n, a_0, \ldots, a_n, x$ staan in deze volgorde op de band.

58. De elementen van een 100 bij 100 matrix zijn alle 0 of 1, en staan kolom na kolom op de band. Bepaal voor elke i in het bereik $0 \leq i \leq 100$ het aantal rijen van de matrix met precies i elementen gelijk aan 1.

59. Een band is van de vorm $m, n, a_0, \ldots, a_{n-1}$: alle getallen zijn geheel en minstens 0. Gevraagd wordt een 1 te printen indien elk van de m getallen 0 t/m m-1 voorkomt in de rij $a_0, \ldots, a_{n-1}$, en een 0 te printen indien dit niet zo is.

60. Een rij $(r_1, \ldots, r_n)$ heeft de eigenschap H indien $r_i \geq r_{2i}$ voor alle gehele i met $1 \leq i \leq \frac{n}{2}$, en $r_i \geq r_{2i+1}$ voor alle gehele i met $1 \leq i < \frac{n}{2}$.
Gegeven is een band met 100 gehele getallen: $x, a_1, \ldots a_{99}$. Er geldt dat de rij $(a_1, \ldots, a_{99})$ eigenschap H bezit. Rangschik de getallen van de band zodanig in een integer array h[1:100], dat de rij (h[1],...,h[100]) weer eigenschap H heeft, en print het resultaat.

61. Gegeven is een niet-negatief geheel getal R op de band. Bepaal alle geheeltallige paren (x,y) die voldoen aan:
$x \geq 0, \quad y \geq 0, \quad x \geq y, \quad x^2 + y^2 = R.$
De restrictie hierbij is dat u geen gebruik mag maken van de real-arithmetiek.