STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

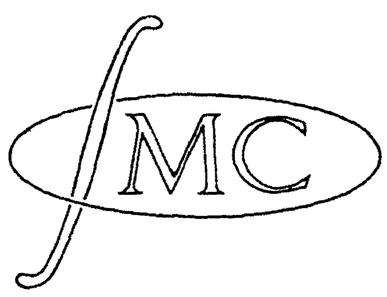
---

REKENAFDELING

Korte inhoud van de voordracht, op
10 september 1963 te houden door

Ch. Harmse

ter gelegenheid van het bezoek van Belgische wiskundigen



The Mathematical Centre at Amsterdam, founded the 11th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.),by the Municipality of Amsterdam and by several industries.

Korte inhoud van de voordracht, op 10 september 1963 te houden door Ch. Harmse, ter gelegenheid van het bezoek van Belgische wiskundigen:

Als ergens een rekenprobleem wordt gesteld, valt de arbeid van het "rekenen" uiteen in twee groepen van werkzaamheden:

1e. inventieve werkzaamheden, omvattende het opstellen van een reken-schema of programma; wij zullen dit PROGRAMMEREN noemen.

2e. automatische afwikkeling van het programma, d.i. CIJFEREN.

Tot programmeren is slechts een betrekkelijk klein aantal mensen in staat, maar de cijferaar behoeft slechts over zeer weinig kundigheden te beschikken, daar hij zich van telramen en/of tabellen kan bedienen. Bij gegeven nauwkeurigheid van die apparatuur is de kwaliteit van het afgeleverde antwoord dan ook uitsluitend bepaald door het programma.

Daarom is de programmeur ook de enige, die verantwoordelijkheid draagt zowel voor de nauwkeurigheid van het antwoord als voor de tijd die er nodig is om het antwoord te becijferen. Hij is ook degene, die er voor heeft te zorgen, dat het programma bestand is tegen "bijzondere gevallen": een schema om $x$ te berekenen uit $x = y/z$ moet instructies bevatten voor het geval $z = 0$ is.

Dat wij het cijferen als een arbeid van mindere orde beschouwen, blijkt bijvoorbeeld hieruit: Doorgaans zijn we er al mee tevreden, als op de vraag naar de waarde van $x$ als antwoord een formule wordt afgeleverd, die $x$ expliciet uitdrukt in bekend veronderstelde grootheden; d.w.z.: met het geven van een schema voor de becijfering beschouwen we het probleem als opgelost, omdat we er van overtuigd zijn dat, als in een bepaald geval uit de praktijk een concreet numeriek antwoord nodig is, de eerste de beste die een formule kan lezen de becijfering zal kunnen uitvoeren.

Gaan we met de taakverdeling tussen programmeur en cijferaar tot het uiterste, dan blijkt dat alleen de programmeur denkwerk verricht; de cijferaar mag in het geheel niet denken: hij behoort niets anders te doen dan de opdrachten waaruit het programma is opgebouwd in de juiste volgorde uitvoeren.

Dat we dit automatische werk kunnen laten verrichten door een automaat is thans wel algemeen bekend; minder bekend is, in welke vorm we aan zo'n computer de opdrachten mededelen, op welke manier hij er kennis van neemt, wat hij doet ter uitvoering er van en hoe hij de resultaten van zijn cijfer-arbeid aan ons vertelt. In het begin van het computer-tijdperk waren het alleen degenen, die "beroepshalve" programmeur waren, die hier iets van wisten en die het ook wel moesten weten om in staat te zijn, de rekenprogramma's op te stellen en vervolgens de machine te bedienen. In kort bestek de zojuist opgesomde vraagpunten te behandelen heeft nauwelijks zin: de informatie, die we hierover zouden kunnen geven, zou een vloed van nieuwe vragen doen rijzen, welker beantwoording voor een goed begrip noodzakelijk maar uit gebrek aan tijd volkomen onmogelijk zou zijn.

Wij zullen daarom volstaan met hierover alleen te vertellen, dat de opdrachten gegeven worden volgens een, voor elke computer andere, code die doorgaans bestaat uit een combinatie van enkele letters en cijfers en dat de aldus gecodeerde opdrachten, bijvoorbeeld in een band of in ponskaarten, worden geponst. De X1 kan zowel ponsband als ponskaarten verwerken, maar de X1 van het Mathematisch Centrum is alleen met een bandlezer uitgerust. Met behulp daarvan "leest" de machine de opdrachten van de band, waarna hij deze opbergt in het geheugen. Op dezelfde manier kunnen we getallen (bijvoorbeeld de elementen van een matrix, waarop we zekere operaties willen laten uitvoeren) naar het geheugen laten overbrengen.

Staan het programma en de daarbij behorende getallen eenmaal in het geheugen, dan kunnen we de computer er toe brengen, het programma opdracht voor opdracht te gaan uitvoeren. Ook daarbij wordt het geheugen gebruikt, namelijk om tussen- en eindresultaten op te bergen, waarvoor het programma de noodzakelijke instructies moet bevatten. Eveneens zal het opdrachten moeten bevatten die tot gevolg hebben dat de resultaten die we op papier wensen te zien, door de aan de computer gekoppelde schrijfmachine worden uitgetypt. Een andere mogelijkheid is, ze te laten ponsen in een band (c.q. in ponskaarten) en dan deze band door een daartoe ingerichte electrische schrijfmachine, de Flexowriter, te laten uittypen. Zo'n ponsband met resultaten kan, eventueel, ook weer door de X1 zelf gelezen worden, waardoor de mogelijkheid gegeven is, de resultaten van een zeker programma verder te laten verwerken door een volgend programma, dat misschien pas over lange tijd aan de orde komt — zelfs zonder dat we z e l f van die resultaten kennis hebben genomen.

Wat we nu wel even zeer duidelijk in het licht moeten stellen is, dat de computer tot geen enkele werkzaamheid in staat is, zolang er niet een programma in zijn geheugen staat: hij is dan zelfs niet in staat, een band te lezen. De noodzakelijke consequentie hiervan is, dat het allereerste programma op een a n d e r e manier in het geheugen moet worden gebracht en de machine het vermogen tot bandlezen zal moeten geven. In de X1 is dit verwezenlijkt doordat een deel van het geheugen reeds door de constructeur bezet is met een aantal vaste programma's, die het bandlezen, typen, ponsen en nog een aantal andere handelingen verzorgen en die, door het indrukken van bepaalde schakelaars, aktief worden gemaakt. Maar ook in de programma's, die we voor de X1 opstellen, kunnen we bepaalde opdrachten opnemen die eveneens het aktief worden van zo'n vast programma veroorzaken.

Met opzet zijn we, vrijwel stilzwijgend, voorbijgegaan aan de vorm van de opdrachtcode. Het zal, uit wat we verteld hebben, reeds voldoende duidelijk zijn geworden, dat het hanteren van die code een gecompliceerde zaak is, die men niet in een paar uurtjes leert: inderdaad zijn daarmede maanden gemoeid. Het heeft dus al heel weinig zin, uitvoerig over die opdrachtcode te gaan spreken en sinds enkele jaren is het ook niet meer nodig die code te kennen om met de computer te kunnen werken want men heeft wegen gezocht en gevonden om hieraan te ontkomen.

Hiervoor zijn een paar zeer goede redenen aan te voeren (waarbij niet ingegaan zal worden op de vraag, of het inderdaad juist deze redenen waren die tot de nieuwe ontwikkeling hebben geleid):

Naarmate de computers s n e l l e r worden (dus: per tijdseenheid meer opdrachten kunnen afwerken) worden steeds ingewikkelder en/of omvangrijker problemen aan de orde gesteld ter berekening, zodat men nu berekeningen programmeert, die nog maar een tiental jaren geleden als veel te tijdrovend buiten beschouwing moesten worden gelaten. Maar om dergelijke programma's te schrijven heeft de programmeur tijden nodig, die het langzamerhand twijfelachtig gaan maken of het nog zin heeft om zo door te gaan: als iemand eerst een half jaar of meer moet besteden aan het schrijven van zijn programma wordt de kans groot, dat het dan geen nut meer heeft om het uit te voeren, ook al zou de computer de berekening misschien in weinige minuten kunnen uitvoeren als het programma eenmaal klaar is.

Bezien wij de zaak nu eens van een andere kant: de programmeur kent de eigenschappen van zijn computer goed, nemen we aan, en hij is bepaald geen kleine jongen in het overzetten van een gesteld probleem in een efficient programma – maar dan moet het probleem ook gesteld zijn, dwz de programmeur moet v o l l e d i g op de hoogte zijn van de vraag of de vragen waar het om gaat, welke bijzondere situaties zich kunnen voordoen, enz. Nemen we eens aan, dat bijvoorbeeld een sterrenkundige een theorie heeft ontwikkeld, die hij door numerieke becijferingen wil toetsen, dan zal de programmeur slechts dan tot een aanvaardbaar programma kunnen komen, als hij zich in die theorie verdiept, er de mogelijke consequenties van overziet even scherp als de sterrenkundige zelf. Waar zal hij de tijd vandaan halen om zich achtereenvolgens te verdiepen in de astronomische, economische, technische, mathematisch-statistische of weerkundige theorieen en problemen die ten grondslag liggen aan de van hem geeiste programma's; en hoe bekwaam zijn degenen die deze programma's van hem eisen, in het formuleren van de gegevens die hij nodig heeft?

Deze laatste vraag roept een andere op: Welke middelen hebben wij voor een dergelijke formulering? Het antwoord zal moeten luiden: "De wiskundige taal", en daarmede zal dan bedoeld zijn de symboliek, die de wiskundige gebruikt om zijn gedachtengangen aan het papier toe te vertrouwen. Maar we weten wel, dat het daarmee niet zo fraai gesteld is als het woord W I S – kunde suggereert. In de beroemde formule van De Moivre voor de n–de machts wortel uit 1 komt een symbool "k" voor en we begrijpen, door jarenlange training, dat daarmede een geheel getal wordt bedoeld. Op de formule volgt dan doorgaans nog tussen haakjes de "mededeling": k = 0, 1, 2, ..... n – 1; de gedachtengang is blijkbaar, dat eerst gelezen moet worden wat a c h t e r de formule staat, voor men weet wat de formule zelf betekent. Wij hebben met dergelijke merkwaardigheden doorgaans niet veel moeite – het kan zijn dat we in een wiskundig betoog eens struikelen en de bedoeling van de auteur niet onmiddellijk doorzien, maar meestal vinden we de draad al gauw terug. Het grote bezwaar blijft evenwel, dat we dan wel m e n e n de werkelijke bedoeling te hebben doorgrond, maar dat het in feite niet goed mogelijk is, daarover objektieve zekerheid te krijgen. Als we ergens de symbolen x = a tegenkomen, dan kan daarmede van alles bedoeld zijn en een van die betekenissen als DE bedoeling aan te nemen, heeft meer van gis– dan van wiskunde.

Men zou nu kunnen voorstellen, degene die het probleem stelt te leren daar z e l f een programma voor te ontwerpen, maar zoals gebleken is in wat wij al eerder zeiden, is dat een verre van eenvoudige zaak: de man is in dat vak nu eenmaal niet deskundig en als hij een programma schrijft dat fout is (wat blijkens de ervaring vrijwel altijd het geval zal zijn) komt toch op de vakman-programmeur de taak neer, de fout of fouten op te sporen en elke programmeur weet, dat dat nog veel moeilijker is dan zelf een programma schrijven, omdat van vele opdrachten in het (foute) programma wel het onmiddellijke effekt bekend is, maar de diepere bedoeling die de auteur daarmee heeft gehad in het duister blijft. Bovendien: de sterrenkundige, die we daarnet ten tonele voerden, zou niet slechts een, maar vele codes moeten beheersen, want niet al zijn problemen zullen door dezelfde computer berekend worden.

Wanneer we nu de aangevoerde punten nog eens overzien, komen we als vanzelf tot de conclusie, dat degene, die het probleem stelt, het zal moeten stellen in de vorm van een rekenschema, dat in een niet al te moeilijk aan te leren taal o n d u b b e l z i n n i g voorschrijft wat er achtereenvolgens gebeuren moet. De bedoelde taal moet dus voldoen aan de eis, dat men er de gang van een rekenproces volkomen in beschrijven kan en moet bovendien in zoverre "universeel" zijn, dat degene, die zich van de taal bedient, niets behoeft te weten van de computer, die uiteindelijk de berekening zal uitvoeren. Het aldus opgestelde schema kan hij dan aan een programmeur geven, die het omzet in de bij z i j n computer behorende opdrachtcode.

Maar wat zien we nu gebeuren? Degene, die tot dusverre "programmeur" was en d e n k w e r k verrichtte, is teruggedrongen naar een positie die niet veel verschilt van die van cijferaar: immers, hij moet een hem gegeven s c h e m a, dat is dus: een p r o g r a m m a, omzetten in de machinecode van zijn computer en dat is blijkbaar een proces dat .. geautomatiseerd kan worden. Het denkwerk is al verricht door degene, die het schema maakte, de opdrachtgever.

Maar dan gaan we natuurlijk een stap verder: we automatiseren het "vertaalproces" inderdaad.

In concreto: Er is zo'n taal ontwikkeld, namelijk ALGOL 60 (er zijn er meer, bijvoorbeeld FORTRAN en COBOL, die we hier evenwel buiten beschouwing zullen laten). De grammatica en semantiek van ALGOL 60 liggen volkomen vast en er zijn geen dubbelzinnigheden. (Deze laatste bewering is wel iets te boud gesproken: zodra het eerste rapport over ALGOL 60 de wiskundigen bereikte stonden er slimme lieden op die aantoonden dat de nagestreefde ondubbelzinnigheid nog niet bereikt was maar zulke schoonheidsfouten worden wanneer ze gesignaleerd zijn, met zorg ge-elimineerd; wij doen daarom maar, a l s o f ALGOL 60 reeds alle ideale eigenschappen heeft, die men deze taal toedenkt.)

Voor elke computer wordt nu een programma geschreven (uiteraard in de code van die computer) dat in staat is, een in ALGOL geschreven programma om te zetten in de code van diezelfde computer (als we zeggen, dat een programma iets doet, dan bedoelen we altijd dat de computer dat doet indien dit programma in zijn geheugen staat). Dit programma, dat dus vertaalt van ALGOL 60 naar machinecode, noemen we VERTALER. Het v e r t a a l d e programma, object-programma genaamd, kan hetzij rechtstreeks door de computer in zijn geheugen worden geplaatst, hetzij afgeleverd worden op een ponsband om in het geheugen te worden gezet als we er aan toe zijn om het te laten "draaien", zoals dat genoemd wordt.

Hoe effektief dit proces werkt zult U straks zien bij de demonstratie die wij op de X1 zullen geven. Wij geven nu een paar voorbeelden van programma's in ALGOL, die ook door degenen, die de taal niet kennen vrijwel geheel zullen worden begrepen. Deze programma's zullen we vanmiddag door de X1 laten vertalen en daarna draaien.

Wij hopen, U in deze voordracht een denkbeeld te hebben gegeven van de wijze waarop een computer heden ten dage gebruikt wordt. Op enkele punten willen wij nog graag even in het bijzonder de nadruk leggen:

1e. Een computer mist alle intelligentie, initiatief en/of belangstelling en volgt alleen passief datgene wat de constructeur (in het bouwschema) en de programmeur (in het programma) hem hebben voorgeschreven.

2e. Een computer die in goede staat verkeert, maakt geen fouten: een rekenresultaat, dat wij fout noemen, is de logische consequentie van een door de programmeur gemaakte fout.

3e. De enige eigenschappen, die de computer voor heeft op de mens, zijn:

   a. zijn zeer grote (en nog steeds toenemende) snelheid;

   b. zijn nauwgezetheid, waardoor hij niet de fouten maakt van een menselijke cijferaar;

   c. zijn vermogen, om tempo en nauwgezetheid lange tijd vol te houden zonder "vermoeid" te raken en zonder ooit zijn gedachten te laten afdwalen.

```
begin   comment Demonstratie-programma  Tovervierkanten.  Ch. Harmse, 23 aug 1963;

        integer r, k, rt, kt, rh, kh; boolean eerste; integer array T[0 : 3, 0 : 3], H[0 : 7, 0 : 3];

        eerste:= true; for r:= 0 step 1 until 3 do
        begin   for k:= 0 step 1 until 3 do
                begin   if r = k then T[3 - r, 3 - k]:= 4 × r + k
                        else if r + k = 3 then T[k, r]:= 4 × r + k
                                          else T[r, k]:= 4 × r + k

                end
        end;

        H[0, 0]:= H[1, 0]:= H[2, 1]:= H[3, 2]:= H[4, 1]:= H[5, 2]:= H[6, 3]:= H[7, 3]:= 0;
        H[0, 1]:= H[1, 2]:= H[2, 0]:= H[3, 0]:= H[4, 3]:= H[5, 3]:= H[6, 1]:= H[7, 2]:= 1;
        H[0, 2]:= H[1, 1]:= H[2, 3]:= H[3, 3]:= H[4, 0]:= H[5, 0]:= H[6, 2]:= H[7, 1]:= 2;
        H[0, 3]:= H[1, 3]:= H[2, 2]:= H[3, 1]:= H[4, 2]:= H[5, 1]:= H[6, 0]:= H[7, 0]:= 3;

AA:     for rh:= 0 step 1 until 7 do
        begin   NLCR; NLCR; for r:= 0 step 1 until 3 do
                begin   rt:= H[rh, r]; for kh:= 0 step 1 until 7 do
                        begin   for k:= 0 step 1 until 3 do ABSFIXT (2, 0, T[rt, H[kh, k]]);
                                SPACE (3)
                        end; NLCR
                end
        end;

        if eerste then
        begin   integer a;
                eerste:= false;
                for r:= 1 step 1 until 3 do
                begin   for k:= 0 step 1 until r - 1 do
                        begin   a:= T[r, k];
                                T[r, k]:= T[k, r];
                                T[k, r]:= a
                        end
                end; goto AA
        end
end
```

```
begin   comment Demonstratie-programma MATRIXVERMENIGVULDIGING. Ch.Harmse, 29 aug 63.

        De te vermenigvuldigen matrices moeten rij na rij geponst zijn en voorafgegaan
        worden door het aantal kolommen en het aantal rijen. De elementen zijn integers.
        Het programma leest eerst matrix A, met p rijen, daarna B, met q kolommen, mits
        p = q. Is hieraan niet voldaan dan wordt de tweede band geweigerd. Dan kan alsnog
        de juiste band ingelezen worden met BVA. Tijdens het bandlezen worden de matrices
        uitgetypt. Als ook de tweede band gelezen is stopt de X1. Geeft men dan BVA, dan
        wordt de produktmatrix C = B × A uitgetypt. Nadat dit voltooid is, stopt de X1 op-
        nieuw. Als dan weer BVA gegeven wordt, dan wordt de produktmatrix opnieuw berekend
        maar opgeborgen inplaats van getypt. Na afloop daarvan wordt "Klaar" getypt. De
        bedoeling is, het tijdverschil te laten zien tussen rekenen met en zonder typen;

        integer n, p;
        n:= read; p:= read;

        begin   integer k, r, m; integer array A[1 : p, 1 : n];
                NLCR; PRINTTEXT (⊦Matrix A⊧); NLCR; for r:= 1 step 1 until p do
                begin   NLCR; for k:= 1 step 1 until n do
                        begin A[r, k]:= read; FIXT (8, 0, A[r, k]) end
                end; NLCR;

        XX:     stop;
                if read ≠ p then begin NLCR; PRINTTEXT (⊦Verkeerde band⊧); goto XX end;

                m:= read;

                begin   integer t, s; integer array B[1 : m, 1 : p], C[1 : m, 1 : n];
                        NLCR; PRINTTEXT (⊦Matrix B⊧); NLCR; for r:= 1 step 1 until m do
                        begin   NLCR; for k:= 1 step 1 until p do
                                begin B[r, k]:= read; FIXT (8, 0, B[r, k]) end
                        end; NLCR;

                        NLCR; PRINTTEXT (⊦Produktmatrix C = BA⊧); NLCR;
                        for r:= 1 step 1 until m do
                        begin   NLCR; for k:= 1 step 1 until n do
                                begin   s:= 0; for t:= 1 step 1 until p do
                                        s:= B[r, t] × A[t, k] + s;
                                        FIXT (8, 0, s)
                                end
                        end;

                        NLCR; NLCR;

        AA:             stop; for r:= 1 step 1 until m do
                        begin   for k:= 1 step 1 until n do
                                begin   s:= 0;
                                        for t:= 1 step 1 until p do
                                                s:= B[r, t] × A[t, k] + s;
                                        C[r, k]:= s
                                end
                        end;

                        PRINTTEXT (⊦Klaar⊧); NLCR; goto AA
                end
        end
end
```

begin comment Demonstratie-programma BENADERING VAN PI. Ch. Harmse, 29 aug 63.

Uitgaande van de regelmatige in- en omgeschreven drie-, vier- en vijfhoeken bij een cirkel met middellijn = 1, wordt pi ingesloten tussen steeds nauwere grenzen door voortdurende verdubbeling van de aantallen zijden. Als a de zijde van de ingeschreven en A die van de omgeschreven n-hoek voorstelt, dan wordt die verdubbeling verkregen door

$$a := \sqrt{(1 - \sqrt{(1 - a \uparrow 2)})/2}$$

$$A := (\sqrt{(A \uparrow 2 + 1)} - 1)/A$$

Daar, blijkens de eerste formule, het vierkant van a eerder bekend is dan a, en voor de volgende verdubbeling juist dat vierkant weer nodig is, wordt dat bewaard in plaats van a zelf, wat rekentijd spaart en de nauwkeurigheid bevordert.

Voor de output is een procedure "schrijf" ingevoerd, die a berekent uit zijn kwadraat. Daarna wordt het aantal zijden uitgetypt, gevolgd door de intussen berekende omtrekken van de beide n-hoeken. Deze procedure test ook, of het verschil tussen beide omtrekken kleiner is dan epsilon, in welk geval de waarde van epsilon wordt getypt en het programma stopt. Epsilon is een macht van tien, die door de gebruiker kan worden ingesteld in de console: als in de vier minst-significante schakelaars het getal m < 16 staat, dan is eps. = $10 \uparrow (-m)$. Als het programma gestopt is, omdat het verschil beneden epsilon is gekomen, dan kan het eventueel met BVA worden doorgestart nadat een kleinere waarde van epsilon ingesteld is;

```
        integer k; integer array n[3 : 5]; real array aa, A[3 : 5];

        procedure schrijf;
        begin   integer aantal; real a, in, om, eps;
                NLCR; eps:= 10 ↑ (-(XEEN (15))); a:= sqrt (aa[k]);
                aantal:= n[k]; ABSFIXT (5, 0, aantal); TAB;
                in:= aantal × a; ABSFIXT (1, 7, in); SPACE (2);
                om:= aantal × A[k]; ABSFIXT (1, 7, om);
                if om - in < eps then
                begin SPACE (5); ABSFIXT (0, 7, eps); NLCR; stop end
        end schrijf;

        aa[3]:= 0.75;     aa[4]:= 0.5;    aa[5]:= (5 - sqrt (5))/8;
        A[3]:= sqrt (3); A[4]:= 1;        A[5]:= sqrt (5 - sqrt (20));

        for k:= 3, 4, 5 do begin n[k]:= k; schrijf end;

volgende: for k:= 3, 4, 5 do
        begin   n[k]:= 2 × n[k];
                aa[k]:= (1 - sqrt (1 - aa[k]))/2;
                A[k]:=  (sqrt (A[k] ↑ 2 + 1) - 1)/A[k]; schrijf
        end; goto volgende
end
```

```
begin   comment Demonstratie-programma LETTER-VERHOUDING. Ch. Harmse, 30 aug 1963.
        Het programma leest een band, waarin de een of andere tekst geponst is
        en typt vervolgens het aantal letters, dat in deze tekst voorkomt, ge-
        volgd door een tabel. In de tabel geeft elke regel een letter van het
        alfabet, gevolgd door het aantal malen dat deze letter is aangetroffen
        en het percentage van het totale aantal letters. Aan het einde wordt
        ter controle het totaal van de percentages gegeven. Tijdens het lezen
        van de band controleert het programma nog niet, of het gelezen symbool
        een letter is of iets anders: dit zou aanzienlijk meer tijd kosten dan
        de gevolgde methode, namelijk alle symbolen te tellen en pas na afloop
        de letters uit te zeven. De rij van gehele getallen: 97, 98, 115, 100,
        ..... 55, 56, 41 die men in het programma ziet staan, zijn de getallen
        die men verkrijgt als men de ponsband-symbolen voor a, b, c, d, ... x,
        y, z opvat als tweetallig geschreven getallen (wat het programma dus
        inderdaad doet) terwijl het getal 11 dat men ook tegenkomt, het evenzo
        opgevatte symbool "stopcode" representeert. Een stopcode kan om aller-
        lei redenen in een band voorkomen zonder noodzakelijk het einde van de
        band aan te geven en daarom is het voorkomen van een enkele stopcode
        geen geschikte maatstaf voor het stoppen van het bandleesproces. Het
        programma test daarom wel o f het gelezen symbool een stopcode is,
        maar houdt alleen dan met bandlezen op als er t w e e stopcodes on-
        middellijk na elkaar worden aangetroffen: is, na een stopcode, het
        volgende symbool iets anders (ook blank) dan wordt de gelezen stopcode
        weer vergeten;

integer i, smbl, wzr, tot, test; integer array L[0 : 127, 1 : 2], A[1 : 26];

procedure PR (letter); string letter;
begin   integer aantal, code;
        NLCR; TAB; PRINTTEXT (letter); i:= i + 1; code:= A[i];
        aantal:= L[code, 1]; ABSFIXT (8, 0, aantal);
        L[code, 2]:= 100 000 × aantal/tot;
        ABSFIXT (2, 2, L[code, 2]/1000); test:= L[code, 2] + test
end PR;

NLCR; PRINTTEXT (‹Demonstratie-programma LETTERVERHOUDING›); NLCR; NLCR;

wzr:= 0;
for i:= 97, 98, 115, 100, 117, 118, 103, 104, 121, 81, 82, 67, 84, 69, 70,
        87, 88, 73, 50, 35, 52, 37 , 38, 55, 56, 41 do
        begin wzr:= wzr + 1; A[wzr]:= i end;
for wzr:= 0 step 1 until 127 do L[wzr, 1]:= 0;

AA:     smbl:= RE7BIT;
if L[11, 1] = 0 then begin BB: L[smbl, 1]:= L[smbl, 1] + 1; goto AA end
                else if smbl ≠ 11 then  begin L[11, 1]:= 0; goto BB end;

tot:= 0;
for i:= 1 step 1 until 26 do tot:= L[A[i], 1] + tot;
PRINTTEXT (‹Totaal: ›); ABSFIXT (8, 0, tot); PRINTTEXT (‹letters›); NLCR;
test:= i:= 0; NLCR;

PR (‹A›); PR (‹B›); PR (‹C›); PR (‹D›); PR (‹E›); PR (‹F›); PR (‹G›); PR (‹H›);
PR (‹I›); PR (‹J›); PR (‹K›); PR (‹L›); PR (‹M›); PR (‹N›); PR (‹O›); PR (‹P›);
PR (‹Q›); PR (‹R›); PR (‹S›); PR (‹T›); PR (‹U›); PR (‹V›); PR (‹W›); PR (‹X›);
PR (‹Y›); PR (‹Z›);

NLCR; NLCR; ABSFIXT (3, 2, test/1000); NLCR
end
```

```
begin   comment Demonstratie-programma M.Potters,
        permutaties van n elementen.  Code-nr MP 290863/3853;
        integer M; boolean ponsen;
        procedure PERM (n, R, USE); value n; integer n; array R; procedure USE;
        begin   integer w, k, i; integer array t[1 : n];
                for k:= 1 step 1 until n do t[k]:= 1;
AA:             USE (n, R); k:= 1;
BB:             w:= R[n - k + 1];
                for i:= n - k + 2 step 1 until n do R[i - 1]:= R[i];
                R[n]:= w;
                t[k]:= t[k] + 1;  if t[k] ≤ k then goto AA;
                t[k]:= 1; k:= k + 1; if k ≤ n then goto BB
        end PERM;
        procedure output (N, A); value N; integer N; array A;
        begin   integer i;
                if ¬ ponsen then NLCR;
                for i:= 1 step 1 until N do
                begin if ponsen then ABSFIXP (2, 0, A[i])
                                else ABSFIXT (2, 0, A[i]) end
        end output;

        M:= XEEN (31); ponsen:= XEEN (-0) < 0;
        begin   integer k; integer array EL[1 : M];
                if ponsen then begin RUNOUT; PUNLCR end else NLCR;
                for k:= 1 step 1 until M do EL[k]:= k;
                PERM (M, EL, output);
                if ponsen then begin TAPEND; STOPCODE end
        end
end
```