STORE TAL

ET GIER DEMONSTRATIONSPROGRAM

Jørgen Kjær

Haldor Topsøe, Vedbæk, Danmark

1967

Copy No.

FORORD

De to tidligere udsendte GIER demonstrationsprogrammer, tændstikspillet NIM og primtalsprogrammet, var illustrationer af to vigtige forhold ved udnyttelsen af en elektronisk cifferregnemaskine: den tilbundsgaaende programmering af alle problemets detailler og maskinens utroligt store regnehastighed.

I nærværende demonstrationsprogram: STORE TAL, er disse to forhold naturligvis ogsaa af betydning, men programmet har ogsaa en vis værdi i sig selv. Ved normale beregninger paa GIER er størrelsen af variable af typen real begrænset til 8-9 betydende cifre og en absolut størrelse paa ca. $1_{10}154$. For variable af typen integer er den øvre grænse ca. $5_{10}11$ i GIER ALGOL 4. Disse grænser er betinget af, hvor mange cifre, der kan gemmes i en enkelt celle i maskinens hukommelse.

Hvis man en sjælden gang, f. eks. af ren nysgerrighed, har brug for at beregne tal med mange betydende cifre, saa som det velkendte tal $2\uparrow 64$:

18446 74407 37095 51616

fra myten om skakspillets opfindelse, kan man ikke uden videre gaa hen til maskinen og faa dette tal beregnet. Der findes ganske vist procedurer til GIER, som tillader beregning med dobbelt nøjagtighed (hvert tal gemmes i to celler) eller flerdobbelt nøjagtighed, men disse kan ikke bruges til enormt store tal.

Programmet STORE TAL tillader beregninger af meget store tal, ikke blot tal, som er for store til at rummes i en enkelt celle, men ogsaa tal, der er for store til paa een gang at være i maskinens hurtige lager. De forskellige regningsarter paa de foreliggende tal specificeres ved hjælp af et simpelt nummersystem.

Programmet er skrevet i GIER ALGOL 4, og beskrivelsen vil senere udkomme paa engelsk.

Virum, Maj 1967
Jørgen Kjær

INDHOLDSFORTEGNELSE

Side

1. INDLEDNING

Med programmet STORE TAL kan man simulere, at GIER-regnemaskinen indeholder registre af meget stor størrelse, saaledes at man paa en bekvem maade kan udføre beregninger paa meget store tal. Det drejer sig her ikke blot om meget store heltal, men ogsaa om tal med mange decimaler. Man opgiver til maskinen, hvor mange decimaler, D, og hvor mange heltalscifre, E, man ønsker at regne med. Programmet simulerer da, at man har en maskine med tre registre, A, B, og C, med plads til lagring af tal af denne størrelse. De regneoperationer, som man ønsker udført, specificeres til maskinen ved hjælp af et ordresystem, hvor tallene 1-22 har en simpel funktion, saa som addition, multiplikation, o.s.v.

I Kapitel 2 beskrives, hvorledes de store tal lagres i maskinen. For tal op til 400 betydende cifre (heltal eller decimaler) lagres hvert tal i lageret med 10 cifre i hver celle, men er der flere betydende cifre, benyttes disken som baggrundslager.

I Kapitel 3 omtales dels de velkendte procedurer til linieskift og sideskift, til udskrift af tekst paa forskellige sprog og til talindlæsning, og dels et par specielle procedurer til kontrol af tallenes størrelse og til fejludskrift, hvis registrene sprænges.

De regneoperationer, som kan udføres paa de tre registre, A, B, og C, er alle deklareret som procedurer. I Kapitel 4 beskrives de simple procedurer, som opererer med eet eller to registre, medens der i Kapitel 5 omtales mere sammensatte procedurer, hvor alle tre registre benyttes, og hvor der i visse tilfælde endog deklareres ekstra, lokale registre til gennemførelse af regningerne.

En særlig gruppe procedurer beskrives i Kapitel 6. Det er procedurer, som beregner en tabel over en funktion. Denne kan være fakultetfunktionen, potensfunktionen a/b hvor a eller b varieres, eller, som en særlig kemisk kuriositet: antallet af isomere, monovalente, alifatiske alkoholer. Den sidste funktion er ogsaa interessant fra et regnemæssigt synspunkt, idet man for at beregne antallet af alkoholer op til N kulstofatomer har brug for at gemme et array med N elementer af store tal.

Kapitel 7 beskriver den procedure, som administrerer udførelsen af de 22 forskellige mulige operationer.

Endelig indeholder Kapitel 8 en beskrivelse af selve programmet og udskrift fra en kørsel. Kapitel 9 indeholder litteraturhenvisninger, og Kapitel 10 er et appendix med en kopi af ALGOL-programmet og af kørselsudskriften.

2. LAGRING AF STORE TAL

2.1. Tal i lageret.

Naar et tal er for stort til at kunne rummes i een celle, opstaar der problemet, paa hvilken form, man skal lade tallet fylde flere celler. Der er flere muligheder. Hvis tallet kun fylder to eller ganske faa celler, vil man normalt bruge samme tallagring, som bruges for en enkelt celle. Hvis vi lagrer heltal med fuld udnyttelse af alle bits i en celle, kan vi i een celle gemme et tal i intervallet:

$$-2\uparrow 39 = -549755813888 \leq N \leq 549755813887 = 2\uparrow 39 - 1$$

Bruger vi to celler til lagring af et heltal med fuld udnyttelse af alle bits, bliver grænserne:

$$-2\uparrow 79 \leq N \leq 2\uparrow 79 - 1$$

fordi alle 40 bits i celle 2 kan udnyttes, i modsætning til celle 1, hvor een bit bruges til fortegn.

Dette system kan udvides til endnu flere celler, og man indser let, at vi paa denne maade faar den tættest mulige pakning af tallene i cellerne, fordi alle bits udnyttes.

Der er imidlertid en væsentlig ulempe ved denne lagringsform. Naar vi skal trykke talværdien af et tal lagret paa denne form, er det nødvendigt at omregne fra det anvendte totalssystemlagring til det normale titalssystem. Denne omregning er meget upraktisk og tidskrævende, naar tallet fylder mange celler.

Vi benytter derfor en anden tallagring, hvor omregningsvanskelighederne helt undgaas. I hver celle lagres kun ti decimale cifre, men paa binær form. Indholdet af en saadan celle kan opfattes som en simpel variabel af typen integer, og som ligger i intervallet:

$$0 \leq N \leq 99999\ \ 99999$$

Naar vi skal lagre et stort tal som $2\uparrow 64$:

18446 74407 37095 51616

kræves der hertil to celler, og lagringen er ganske ensbetydende med, hvad der er udtrykt i følgende program:

```
begin
   integer array A[0:1];
   A[0] := 37095 51616;
   A[1] := 18446 74407
end;
```

For større tal kræves naturligvis flere elementer i A.

Da hver celles indhold er et heltal i omraadet fra 0 til 10↑10 - 1, kan man ogsaa sige, at hver celle indeholder et ciffer i 10↑10 - tal-systemet.

Programmets praktiske administration af heltal og decimaler udføres saaledes. Vi opererer med følgende heltalsvariable i yderste blok:

integer D: det ønskede antal decimaler.

integer E: det ønskede antal heltalscifre.

integer decimals: minus antallet af celler, som kræves til lagring af de D decimaler.

integer limit: antallet af celler, som kræves til lagring af de E heltalscifre, minus 1.

Dette kan belyses nærmere ved et eksempel. Opgiver vi:

```
D = 27
E = 45
```

vil programmet forhøje disse værdier, saaledes at de bliver multipla af 10, d.v.s. D regnes til 30 og E til 50. Vi faar da værdierne af decimals og limit til:

```
decimals := - (30:10);
limit := 50:10 - 1;
```

eller:

```
decimals := - 3;
limit := 4;
```

Vi kunne nu gemme vort store tal som et array: A[4:-3], saaledes at de enkelte elementer indeholder:

A[-3]:	decimaler fra	21 - 30
A[-2]:	- -	11 - 20
A[-1]:	- -	1 - 10
A [0]:	heltalscifre fra	1 - 10
A [1]:	- -	11 - 20
A [2]:	- -	21 - 30
A [3]:	- -	31 - 40
A [4]:	- -	41 - 50

Det er imidlertid mere praktisk at forhøje de anvendte indices, saaledes at laveste indeks altid er nul:

A[0]:	decimaler fra	21 - 30
A[1]:	- -	11 - 20
A[2]:	- -	1 - 10
A[3]:	heltalscifre fra	1 - 10
A[4]:	- -	11 - 20
A[5]:	- -	21 - 30
A[6]:	- -	31 - 40
A[7]:	- -	41 - 50

Indeks for den højeste celle (her 7) beregnes som den globale variable:

```
c39 := limit - decimals;
```

I mange af de anvendte procedurer vil der indgaa en central forsætning af formen:

```
for count := decimals step 1 until limit do
```

eller:

```
for count := 0 step 1 until c39 do
```

I andre procedurer gennemløbes for-sætningen baglæns.

2.2. Tal paa disken.

De tre regneregistre, A, B, og C deklareres som tre arrays:

```
integer array A, B, C[0:c39];
```

Da disse arrays kan fylde mere end hvad lageret kan rumme, hvis c39 er meget stor, er programmet indrettet saaledes, at c39 aldrig bliver større end 39. Naar brugeren har opgivet D og E, beregnes de tilsvarende værdier af limit og decimals som forklaret ovenfor, og vi beregner størrelsen:

```
step := (limit - decimals)÷40 + 1;
```

Værdien af step angiver, hvormange disk-kanaler paa 40 celler, som kræves for at rumme tallet. Er step større end 1, betyder det, at registrene optager mere end 40 celler. Vi beregner derfor den logiske variable, large, som:

```
large := step > 1;
```

og værdien af c39 som:

```
c39 := if large then 39 else limit - decimals;
```

Hvis large er sand, gemmes de tre registre, A, B og C, paa disken, og de forskellige regneprocedurer maa sørge for at hente de enkelte kanaler frem fra disken og gemme dem igen, hvis de ændres. De tre arrays har da dimensionen [0:39] og benyttes som arbejdsceller for de aktuelle segmenter af de store tal. Hvis derimod large er falsk, er c39 mindre end 39, og der benyttes slet ikke disklagring.

Denne skelnen mellem store tal (c39 < 39) og meget store tal (c39 = 39) er nødvendig paa grund af det lille ferritlager i GIER, og det giver mere plads til programmet i lageret, hvis man arbejder med tal, der kun fylder faa celler.

I regneprocedurerne optræder de ovennævnte arrays, A, B og C, som

formelle parametre af typen integer array. Desuden er hvert register karakteriseret ved hjælp af to andre parametre af typen integer:

```
integer size, n;
```

Her er parameteren size et tal, der siger, hvor mange celler i registret, der er fyldt op med cifre. Hvis size = limit, er alle celler fyldt op, og hvis size = decimals, er det kun den mindst betydende celle af tallet, som indeholder cifre forskellig fra nul. Hvis tallet vokser eller aftager, skal procedurerne regulere size tilsvarende op eller ned. Derfor er size normalt kaldt ved name.

Størrelsen n bruges kun, naar large er sand, og da til at fiksere tallets plads paa disken. De tre hovedregistre, A, B og C, har n-værdierne 1, 2 og 3. Programmet reserverer ikke noget særligt disk-omraade til lagring af tallene, men benytter kun det frie omraade af disken. Oplysningerne om dette omraades placering fremskaffes ved procedurekaldet:

```
where({<free}, FREE);
```

og transporter til og fra disken sker da med kald af formen:

```
get(A, FREE, t);
```

eller:

```
put(A, FREE, t);
```

Her skal t være et tal fra 1 og opefter, som giver det relative kanalnummer indenfor omraadet. Et register med parameteren n vil være lagret paa følgende relative kanaler:

```
n×step + 1
n×step + 2
n×step + 3
   .
   .
   .
   .
   .
n×step + step
```

altsaa ialt step kanaler.

Nogle af procedurerne har brug for at oprette hjælperegistre. I proceduren til beregning af tallet pi, kræves saaledes to ekstra registre. Disse deklareres i proceduren:

```
integer array SUM, T1[0:c39];
```

Hvis large er sand, skal vi ogsaa bruge de to n-værdier for de nye registre. Hertil benyttes den globale variable:

```
integer ftrack;
```

der sættes lig 4 ved beregningens begyndelse, og som hele tiden peger paa den første frie n-værdi. Ved deklarationen af SUM og T1 sættes n-værdien for SUM lig med ftrack og for T1 lig med ftrack + 1, og vi øger ftrack med 2. Naar pi-proceduren forlades, trækkes der 2 fra i ftrack.

3. ADMINISTRATIONSPROCEDURER

De fleste af disse procedurer har været omtalt tidligere i NIM-beskrivelsen, Kjær (1966), og i beskrivelsen af primtalprogrammet, Kjær (1967).

3.1. Sideskift.

De to procedurer, NEW PAGE og LINE, benyttes til administration af sideskift og liniekontrol, saaledes at der for hver 70 linier er nogle frie linier ved overgangen til det næste stykke papir.

3.1.1. Proceduren NEW PAGE. Deklarationen er:

```
procedure NEW PAGE;
begin
   for linerest := linerest - 1 while linerest ≥ 0, 69 do writecr;
   writechar(72)
end NEW PAGE;
```

Den globale integer linerest bruges til linietælling.

3.1.2. Proceduren LINE. Deklarationen er:

```
procedure LINE;
if linerest < 8 then NEW PAGE
else
begin
   linerest := linerest - 1;
   writecr
end LINE;
```

Proceduren giver normalt en ny linie (writecr), men hvis linerest er mindre end 8, udføres sideskift.

3.2. Spørgsmaal.

Til kommunikation med maskinen benyttes de tre procedurer: WRITE TEXT, SELECT LANGUAGE og ASK NUMBER. Da der kun er brug for at spørge om ja eller nej et enkelt sted i programmet, er den tidligere procedure: QUESTION ikke medtaget. Hvis man specificerer et negativt antal decimaler (D), opfatter maskinen det som et tegn paa, at beregningen ønskes afsluttet.

3.2.1. Proceduren WRITE TEXT. Deklarationen er:

```
procedure WRITE TEXT(dan, eng, fr, ger);
string dan, eng, fr, ger;
writetext(case lang of (dan, eng, fr, ger));
```

Svarende til de fire mulige værdier af den globale integer lang (1, 2, 3, eller 4) faar man skrevet en tekststreng paa det tilsvarende sprog (dansk, engelsk, fransk eller tysk).

3.2.2. Proceduren SELECT LANGUAGE. Denne benyttes i programmets begyndelse til indstilling af den ønskede værdi af lang. Deklarationen er:

```
procedure SELECT LANGUAGE;
begin
   LINE;
   writetext(
   {<Select language: d: danish, e: english, f: french, g: german.: }); 
   lang := lyn - 51;
   if lang < 1 then lang := 1;
   if lang > 4 then lang := 4;
   LINE;
   WRITE TEXT(
   {<Dansk},
   {<English},
   {<Francais},
   {<Deutsch});
   LINE
end SELECT LANGUAGE;
```

3.2.3. Proceduren ASK NUMBER. Denne procedure bruges til indlæsning af tal fra skrivemaskinen. Da vi kun ønsker at indlæse heltal, er den gjort til en integer procedure. Deklarationen er:

```
integer procedure ASK NUMBER(dan, eng, fr, ger);
string dan, eng, fr, ger;
begin
   LINE;
   WRITE TEXT(dan, eng, fr, ger);
   writetext({<: });
   ASK NUMBER := read integer
end ASK NUMBER;
```

3.3. Kontrol.

Til visse kontrolformaal bruges de to procedurer: ACCEPT og ALARM. De er specielle for nærværende program.

3.3.1. Proceduren ACCEPT. Deklarationen er:

```
procedure ACCEPT(cond);
value cond;
boolean cond;
if -, cond then
begin
   LINE;
   WRITE TEXT(
   {<Brug flere heltalscifre},
   {<Use more integer digits},
   {<Le nombre de chiffres entiers est trop petit},
   {<Zu wenig Ganzzahlstellen});
   go to if show then E1 else E2
end ACCEPT;
```

Den formelle parameter: boolean cond, er en betingelse, som skal være opfyldt, for at et foreliggende tal kan udregnes med den valgte størrelse af registrene. Er betingelsen ikke opfyldt, udskriver programmet en bemærkning om, at man skal bruge flere heltalscifre (stør-

re værdi af E). Der hoppes derefter til een af de to etiketter, E1 eller E2, i hovedprogrammet, hvor man kan vælge en ny værdi af D og E.

Den globale variable, boolean show, har følgende betydning. For at lette demonstrationen er beregningen opdelt i to afsnit. Først indlæser programmet en række ordrenumre fra en papirstrimmel og udfører de tilsvarende regninger. I denne fase er show sat til sand. I den anden fase er show falsk, og man kan indlæse ordretallene fra skrivemaskinen.

3.3.2. Proceduren ALARM. Deklarationen er:

```
procedure ALARM(text);
string text;
begin
   LINE;
   writetext(<Error in: >);
   writetext(text);
   go to if show then E1 else E2
end ALARM;
```

Denne procedure benyttes i forskellige regneprocedurer, hvor der kan forekomme overløb i registrene. Hvis dette er tilfældet, udskrives teksten:

Error in:

samt den tekst, der er givet ved den formelle parameter. Der hoppes derefter til E1 eller E2, ligesom i ACCEPT.

4. SIMPLE REGNEPROCEDURER

Ved simple regneprocedurer forstaar vi her saadanne procedurer, hvori der kun indgaar eet eller to store tal, samt eventuelt et tal af normal størrelse, d.v.s. som kan rummes i een celle.

4.1. Procedurer med eet stort tal.

Vi omtaler her de fire procedurer:

ASSIGN(x, A, asize, na), som tildeler det store register, A, værdien x, hvor x er en normal real.

MULT(A, asize, na, n), som multiplicerer det store tal, A, med den normale heltalsvariable, n.

DIVIDE(A, asize, na, n, empty), som dividerer det store tal, A, med den normale heltalsvariable, n.

PRINT(A, asize, na), som trykker det store register.

Bemærk, at parametren A altid ledsages af de to andre parametre, asize og na, der som forklaret side 12 bestemmer henholdsvis hvor mange celler af registret, som er udfyldt, og nummeret for lagring paa disken.

4.1.1. Proceduren ASSIGN.

Deklarationen herfor er:

```
integer procedure ASSIGN(x, A, asize, na);
value x, na;
integer asize, na;
real x;
integer array A;
begin
   integer c1, c2, t1, t2, cell1, cell2;
   real factor;
   x := abs(x);
   c1 := c39;
   for count := 0 step 1 until c1 do A[count] := 0;
```

```
    if x = 0 then
    begin
       asize := c1 := c2 := cell1 := cell2 := 0;
       go to L1
    end if x = 0;
    asize := entier(0.0434294482×ln(x));
    if asize > limit then ALARM({<ASSIGN});
    factor := MODUL/asize;
    cell1 := entier(x/factor);
    cell2 := (x/factor - cell1)×MODUL;
    c1 := asize - decimals;
    c2 := c1 - 1;
    if c2 < 0 then
    begin
       c2 := c1;
       cell2 := cell1
    end if c2 < 0;
    if c1 < 0 then c1 := c2 := cell1 := cell2 := 0;
L1: if large then
    begin
       t1 := 1 + c1:40;
       t2 := 1 + c2:40;
       c1 := c1 mod 40;
       c2 := c2 mod 40;
       for count := 1 step 1 until step do
       begin
          if count = t1 then
          begin
             A[c1] := cell1;
             if t1 ≠ t2 then
             begin
                put(A, FREE, na×step + t1);
                A[c1] := 0;
                A[c2] := cell2;
                put(A, FREE, na×step + t2)
             end different track
             else
             begin
                A[c2] := cell2;
```

```
            put(A, FREE, na×step + t1)
          end same track;
          A[c1] := A[c2] := 0
        end this track
        else
        put(A, FREE, na×step + count)
      end for count
    end if large
    else
    begin
      A[c1] := cell1;
      A[c2] := cell2
    end core
end ASSIGN;
```

Proceduren overfører talværdien af en normal, simpel variabel (real x) til det store register, A, der noteres som den formelle parameter af typen integer array. Proceduren beregner ogsaa værdien af asize. Hvis large er sand, skal disk-lagringsnummeret, na, være bestemt før kaldet.

Den første sætning i proceduren:

```
x := abs(x);
```

er indsat, fordi vi simpelthen ikke ønsker at regne med negative tal.

Derefter nulstilles alle celler i A-arrayet, idet der højst kan være to af cellerne, som har et indhold forskelligt fra nul. Dette skyldes, at x som real indeholder 8 betydende cifre, og da hver celle rummer 10 decimale cifre, er to celler nok til at reproducere x. Resten af proceduren gaar da ud paa at finde numrene paa disse to celler, samt deres indhold.

Først behandles det specielle tilfælde, at x = 0. Derefter beregnes asize af formlen:

```
asize := entier(0.043429448 2×ln(x));
```

altsaa heltalsdelen af en tiendedel af titalslogaritmen til x. Dette er en umiddelbar følge af, at der kan være 10 decimale cifre i hver celle. Hvis asize er større end limit, faas ALARM-udhop.

Indholdet af de to celler beregnes nu af formlerne:

```
factor := MODUL↑asize;
cell1 := entier(x/factor);
cell2 := (x/factor - cell1)×MODUL;
```

Lad os som eksempel antage, at x er:

```
x = 123 45600 00000
```

Dette giver asize = 1, svarende til at tallet skal gemmes i de to celler:

```
A[0-decimals]:   45600 00000
A[1-decimals]:         123
```

Den globale heltalsvariable, MODUL, har værdien $1_{10}10$, og factor faar derfor i dette tilfælde ogsaa værdien $1_{10}10$. Værdien af cell1 bliver da:

```
entier(123 45600 00000/1 00000 00000)
```

eller: 123. Værdien af cell2 bliver:

```
(123 45600 00000/1 00000 00000 - 123)×1 00000 00000
```

eller: 45600 00000.

Index, c1, for cell1 maa blive asize - decimals, da vi har forskudt disse indices, saaledes at den laveste celle altid har nummer nul. Index, c2, for cell2 bliver een lavere.

Der følger derefter en kontrol paa, om tallet x har været saa lille, at c2 eller maaske baade c1 og c2, er mindre end 0.

Den sidste halvdel af proceduren optages af indsættelsen af værdierne af cell1 og cell2. Hvis large er sand, skal der udføres de nødvendige kanaltransporter til disken idet vi tager hensyn til at cell1 og cell2 kan staa paa to forskellige kanaler. Er large falsk, er operationen simpel:

```
A[c1] := cell1;
A[c2] := cell2;
```

4.1.2. Proceduren MULT. Deklarationen er:

```
integer procedure MULT(A, asize, na, n);
value na, n;
integer asize, na, n;
integer array A;
begin
   integer c, ta, c1;
   carry := c := 0;
   ta := na×step + 1;
   if large then get(A, FREE, ta);
   c1 := limit - decimals;
   for count := 0 step 1 until c1 do
   begin
      cell := A[c];
      code cell, MODUL, carry, n;
      2, 44;
      2, 44;
      2, 44;
      3, 44;
      arn a3, pm a1   ; R := carry, M := cell
      ml p+a4,dl a2   ; RM := (carry+cell×n)/MODUL
      gr  a3, gm a1   ; carry := quotient, cell := rem.
      e               ;
      A[c] := cell;
      c := c + 1;
      if large then
      begin
         if c = 40 then
         begin
            c := 0;
            put(A, FREE, ta);
            ta := ta + 1;
            get(A, FREE, ta)
         end if c = 40
      end if large;
```

```
      if count = asize - decimals then
      begin
         if carry = 0 then go to EX
         else
         if count < c1 then asize := asize + 1
         else ALARM({<MULT})
      end if asize
   end for count;
EX: if large then put(A, FREE, ta)
end MULT;
```

Proceduren multiplicerer det store register, A, med en normal integer parameter, n. Det store register er karakteriseret af de sædvanlige tre parametre:

```
integer array A;
integer asize;
integer na;
```

hvoraf asize er kaldt ved name. Resultatet af multiplikationen gemmes i det samme register, A. Dette er praktisk, f.eks. ved beregning af fakultetsfunktionen.

Det centrale i proceduren er for-sætningen:

```
for count := 0 step 1 until c1 do
```

hvor værdien af c1 er limit - decimals. For hver værdi af count, hentes den tilsvarende A-celle frem:

```
cell := A[c];
```

Index c er normalt lig med count, men hvis large er sand, bliver den sat tilbage til nul, hver gang 40 celler er passeret, ligesom der sker de nødvendige disk-transporter.

Vi skal nu multiplicere cell med n og gemme resultatet tilbage i A[c]. Herved sker der naturligvis menteoverføring, og dette løses paa en praktisk maade ved hjælp af et lille stykke maskinkode paa tre celler. Der benyttes en global integer, carry, til at gemme menten, og denne nulstilles før for-sætningen.

Den første celle maskinkode indeholder ordren:

```
arn  a3,  pm  a1
```

som er ensbetydende med:

```
R := carry;
M := cell;
```

hvor R er maskinens resultatregister og M dens multiplikatorregister. Den anden maskinkodecelle indeholder ordren:

```
ml  p+a4,  dl  a2
```

Her bevirker multiplikationsordren, ml, at vi faar beregnet:

```
n×cell + carry
```

og resultatet staar nu i det forenede R-og-M-register, altsaa med 78 bits nøjagtighed. Denne store nøjagtighed er nødvendig, fordi baade n og cell jo kan have værdien op til $1_{10}10-1$. Efter multiplikationen maa vi dividere med MODUL i dl-ordren:

```
(n×cell + carry)/MODUL
```

Herved fremkommer kvotienten i R-registret og divisionsresten i M-registret. Den tredje ordre:

```
gr  a3,  gm  a1
```

gemmer disse to resultater:

```
carry := R;
cell := M;
```

Det egentlige ALGOL-program fortsætter nu med at gemme cell tilbage i A[c].

I slutningen af for-sætningen har vi betingelsen:

```
if count = asize - decimals then
```

Denne betingelse er sand, naar vi kommer frem til den A-celle, der indeholder det højeste element i A-registret. Der er nu to muligheder. Hvis menten er nul, er A-tallet ikke kommet til at fylde flere celler end før multiplikationen, og vi er færdige. Hvis menten er større end nul, er tallet vokset, og vi øger asize med 1, dog kun hvis asize er mindre end limit, ellers faas ALARM-udhop.

4.1.3. Proceduren DIVIDE. Deklarationen herfor er:

```
integer procedure DIVIDE(A, asize, na, n, empty);
value na, n;
integer asize, na, n;
boolean empty;
integer array A;
begin
   integer c, ta;
   first := true;
   carry := 0;
   c := asize - decimals;
   ta := 1 + c:40 + na×step;
   c := c mod 40;
   if large then get(A, FREE, ta);
   for count := asize step -1 until decimals do
   begin
      cell := A[c];
      code cell, MODUL, carry, n;
      2, 44;
      2, 44;
      2, 44;
      3, 44;
      arn a1, pm  a3 ; R := cell, M := carry
      ml  a2, dl p+a4; RM := (cell+carry×MODUL)/n
      gr  a1, gm  a3 ; cell := quotient, carry := rem.
      e              ;
      A[c] := cell;
      c := c - 1;
```

```
      if large then
      begin
         if c < 0 then
         begin
            c := 39;
            put(A, FREE, ta);
            ta := ta - 1;
            get(A, FREE, ta)
         end if c < 0
      end if large;
      if first then
      begin
         if cell > 0 then first := false
         else
         if asize > decimals then asize := asize - 1
      end if first
   end for count;
   if large then put(A, FREE, ta);
   empty := first ∧ cell = 0
end DIVIDE;
```

Vi har her de samme parametre som i MULT-proceduren, blot betyder n her det tal, som A skal divideres med. En ekstra parameter er tilføjet her: boolean empty, som sættes til sand, hvis A bliver nul ved divisionen. Det er bekvemt at faa denne information ud af hensyn til beregningen af tallet pi, hvor man summerer tre led, som efterhaanden bliver divideret bort.

Den centrale for-sætning i proceduren er her:

```
for count := asize step -1 until decimals do
```

Her begynder vi altsaa med de højeste celler i tallet A. For hver værdi af count henter vi den tilsvarende A-celle:

```
cell := A[c];
```

Ogsaa her har vi et stykke maskinkode paa tre celler, som beregner:

```
(cell + carry×MODUL)/n
```

og som gemmer kvotienten i cell og resten i carry. Ogsaa her skal der være kontrol med, om asize er ændret (formindsket) og de nødvendige disk-transporter, hvis large er sand. Sidst i proceduren beregnes empty som:

```
empty := first ∧ cell = 0;
```

Den logiske variable first er sat til sand ved divisionens begyndelse og skiftes til falsk, saa snart der tilbageføres et A-element forskelligt fra nul.

4.1.4. Proceduren PRINT. Deklarationen er:

```
integer procedure PRINT(A, asize, na);
value asize, na;
integer asize, na;
integer array A;
begin
   boolean first;
   integer DIVISOR, digit, i, space, group, ta, c;
   procedure GROUP(n);
   value n;
   integer n;
   begin
      DIVISOR := MODUL:10;
      space := if first then 0 else 16;
      for i := 1 step 1 until 10 do
      begin
         digit := n:DIVISOR;
         n := n mod DIVISOR;
         if digit ≠ 0 then
         begin
            writechar(digit);
            first := false;
            space := 16
         end
         else writechar(space);
         if i = 5 then writechar(0);
```

```
            DIVISOR := DIVISOR:10
        end for i
    end GROUP;
    first := true;
    group := 0;
    LINE;
    if asize < 0 then asize := 0;
    c := asize - decimals;
    ta := 1 + c:40;
    c := c mod 40;
    if large then get(A, FREE, na×step + ta);
    for count := asize step -1 until decimals do
    begin
        GROUP(A[c]);
        if count = 0 ∧ decimals < 0 then
        begin
            writechar(59);
            first := false
        end
        else writechar(0);
        group := group + 1;
        if (group mod 6 = 0) ∧ count ≠ decimals then LINE;
        c := c - 1;
        if large then
        begin
            if c < 0 then
            begin
                c := 39;
                ta := ta - 1;
                get(A, FREE, na×step + ta)
            end if c < 0
        end if large
    end for count;
end PRINT;
```

Vi har de sædvanlige parametre, A, asize og na, til at bestemme det register, som skal trykkes. Trykningen af de ti decimale cifre i en enkelt celle af A-registret sker med en lokal procedure, GROUP, der trykker indholdet paa formen:

12345 12345

altsaa med et space mellem cifrene 5 og 6. Kaldet af GROUP sker fra for-sætningen:

```
for count := asize step -1 until decimals do
```

Mellem hver gruppe af 5+5 cifre anbringes normalt et space, undtagen paa punktummets plads. Der trykkes højst 6 celler pr. linie, altsaa 60 decimale cifre.

4.2. Procedurer med to store tal.

I denne gruppe procedurer omtales de to procedurer, COPY og ADD.

4.2.1. Proceduren COPY. Deklarationen er:

```
integer procedure COPY(A, asize, na, B, bsize, nb);
value na, nb;
integer asize, na, bsize, nb;
integer array A, B;
begin
   integer c, c1, t1, t2;
   c1 := c39;
   if large then
   begin
      t1 := na×step;
      t2 := nb×step;
      for count := 1 step 1 until step do
      begin
         t1 := t1 + 1;
         t2 := t2 + 1;
         get(A, FREE, t1);
         put(A, FREE, t2)
      end for count
   end if large
```

```
   else
   for c := 0 step 1 until c1 do B[c] := A[c];
   bsize := asize
end COPY;
```

Proceduren udfører den simple operation:

```
B := A;
```

Det store register, B, sættes lig med det store register A. Begge registre er karakteriseret ved de sædvanlige tre parametre:

```
A, asize, na;
B, bsize, nb;
```

Hvis large er sand, overfører proceduren simpelthen alle A-kanaler til B-kanalerne. Er large falsk, har vi den simple for-sætning:

```
for c := 0 step 1 until c39 do B[c] := A[c];
```

I begge tilfælde sættes bsize lig med asize.

4.2.2. Proceduren ADD. Deklarationen er:

```
integer procedure ADD(B, bsize, nb, factor, A, asize, na);
value bsize, nb, factor, na;
integer bsize, nb, factor, asize, na;
integer array A, B;
begin
   integer ta, tb, c;
   if large then
   begin
      ta := tb := 1;
      get(A, FREE, na×step + ta);
      get(B, FREE, nb×step + tb)
   end if large;
   c := - 1;
   carry := 0;
```

```
   for count := decimals step 1 until limit do
   begin
      c := c + 1;
      if c = 40 then
      begin
         c := 0;
         put(A, FREE, na×step + ta);
         ta := tb := ta + 1;
         get(A, FREE, na×step + ta);
         get(B, FREE, nb×step + tb)
      end if c = 40;
      cell := A[c] + factor×B[c] + carry;
      carry := 1;
      for carry := carry -1 while cell < 0 do
      cell := cell + MODUL;
      cell2 := cell:MODUL;
      A[c] := cell - cell2×MODUL;
      carry := carry + cell2;
      if count ≥ bsize ∧ carry = 0 then go to L1
   end for count;
L1: if carry ≠ 0 then ALARM(‡<ADD‡);
   if large then put(A, FREE, na×step + ta);
   asize := limit + 1;
   c := limit - decimals;
   ta := 1 + c:40;
   c := c mod 40;
   if large then get(A, FREE, na×step + ta);
   for asize := asize -1 while asize > decimals do
   begin
      if A[c] ≠ 0 then go to L2;
      c := c - 1;
      if c < 0 then
      begin
         c := 39;
         ta := ta - 1;
         get(A, FREE, na×step + ta)
      end if c < 0
   end for asize;
L2: end ADD;
```

Denne procedure udfører følgende operation paa de to store registre, A og B:

```
A := A + factor×B;
```

hvor factor er en normal integer variabel. Proceduren er først og fremmest lavet med henblik paa factor = 1 og -1, som giver simpel addition og subtraktion.

Den fundamentale for-sætning er:

```
for count := decimals step 1 until limit do
```

Før for-sætningen sætter vi mentecellen, carry, til nul. Indexcellen, c, varierer som count med fornøden korrektion, hvis large er sand.

Det nye indhold af A-cellen beregnes som:

```
cell := A[c] + factor×B[c] + carry
```

Her er det vigtigt, at factor kun har en beskeden størrelse, da man ellers kan faa overløb. Hvis værdien af cell er negativ, adderes MODUL til cell og carry gives en tilsvarende negativ værdi. Naar cell er blevet positiv, divideres med MODUL paa sædvanlig maade.

Naar count har naaet værdien af bsize, kan operationen afbrydes, hvis menten er nul.

Naar den fundamentale for-sætning er gennemløbet, følger en anden for-sætning:

```
for asize := asize -1 while asize > decimals do
```

hvor asize først er sat til limit + 1. Formaalet med denne for-sætning er at faa genindstillet den korrekte værdi af asize.

Det er en væsentlig forudsætning for funktionen af denne procedure, at den nye værdi af A ikke bliver negativ.

5. SAMMENSATTE PROCEDURER

Under denne gruppe procedurer omtales den meget vigtige procedure, LONGMULT, der udfører operationen:

C := A×B;

hvor A, B og C er store registre.

Endvidere har vi de fire andre procedurer:

EXP:	Beregning af eksponentialfunktionen.
PI TO:	Beregning af pi.
SQRT:	Beregning af kvadratrod.
EXPISQN:	Beregning af den meget specielle funktion: exp(PI×sqrt(N)).

Disse fire procedurer opererer alle paa store registre.

5.1. Proceduren LONGMULT.

Deklarationen herfor er:

```
integer procedure LONGMULT(A, asize, na, B, bsize, nb, C, csize, nc);
value asize, na, bsize, nb, nc;
integer asize, na, bsize, nb, csize, nc;
integer array A, B, C;
begin
   integer c, factor, rsize, nr, s, shift, cb, tb, c1, t1, c2, t2, s1,
   s2, s3;
   integer array RES[0:c39];
   nr := ftrack;
   ftrack := ftrack + 1;
   ASSIGN(0, C, csize, nc);
   tb := 1 + nb×step;
   cb := 0;
   if large then get(B, FREE, tb);
```

```
for c := 0 step 1 until bsize - decimals do
begin
   shift := c + decimals;
   if large then
   begin
      factor := B[cb];
      cb := cb + 1;
      if cb = 40 then
      begin
         cb := 0;
         tb := tb + 1;
         get(B, FREE, tb)
      end new B track
   end large
   else
   factor := B[c];
   COPY(A, asize, na, RES, rsize, nr);
   MULT(RES, rsize, nr, factor);
   if shift ≠ 0 then
   begin
      s1 := if shift < 0 then -c else limit - decimals;
      s2 := - sign(shift);
      s3 := if shift < 0 then limit - decimals - c else - c;
      for s := s1 step s2 until s3 do
      begin
         if s < 0 ∨ s > limit - decimals then
         cell := 0
         else
         if large then
         begin
            t1 := nr×step + 1 + s:40;
            c1 := s mod 40;
            get(RES, FREE, t1);
            cell := RES[c1]
         end large
         else
         cell := RES[s];
         c2 := s + shift;
```

```
            if c2 > limit - decimals then
            begin
               if cell ≠ 0 then ALARM(«LONGMULT»)
            end if too big
            else
            if c2 ≥ 0 then
            begin
               if large then
               begin
                  t2 := nr×step + 1 + c2:40;
                  c2 := c2 mod 40;
                  get(RES, FREE, t2);
                  RES[c2] := cell;
                  put(RES, FREE, t2)
               end if large
               else
               RES[c2] := cell
            end if not c2 > limit - decimals
         end for s;
      end if shift ≠ 0;
      rsize := rsize + shift;
      ADD(RES, rsize, nr, 1, C, csize, nc)
   end for c
   ftrack := ftrack - 1;
end LONGMULT;
```

Som nævnt udfører proceduren følgende operation paa de tre store registre, A, B og C:

```
C := A×B;
```

og hvert register har de sædvanlige tre formelle parametre:

```
A, asize, na;
B, bsize, nb;
C, csize, nc;
```

Proceduren opretter et ekstra, lokalt, stort register med deklarationen:

```
integer array RES[0:c39];
```

Den tilsvarende n-værdi for disk-lagringen, nr, sættes lig ftrack.

Proceduren forudsætter, at B-registret indeholder de færreste celler (bsize < asize). Herved faas den hurtigste regnetid, men forudsætningen behøver dog ikke at være opfyldt. I det følgende ser vi kun paa hovedtrækkene i proceduren og ser blandt andet bort fra disk-kanaladministrationen.

C-registret sættes først lig nul.

Den grundlæggende for-sætning er:

```
for c := 0 step 1 until bsize - decimals do
```

som efter tur henter de enkelte B-celler frem som den simple variable, factor:

```
factor := B[c];
```

Nu sættes hjælperegistret, RES, lig med A-registret:

```
COPY(A, asize, na, RES, rsize, nr);
```

og vi multiplicerer RES med factor. Vi faar nu brug for hjælpevariablen, shift:

```
shift := c + decimals;
```

som siger, hvor mange pladser den aktuelle værdi af factor staar til venstre for punktummet i B-registret. Alle celler i RES-registret skal derfor flyttes shift pladser til venstre, inden vi kan addere RES til C-registret. Denne shift-operation er noget kompliceret, fordi shift kan være enten positiv eller negativ, og vi maa naturligvis flytte cellerne paa en saadan maade, at man ikke kommer til at ødelægge celler, der senere skal flyttes. Hertil kommer saa disk-administrationen, hvis large er sand.

Naar skiftet er udført, adderes RES til C:

```
ADD(RES, rsize, nr, 1, csize, nc);
```

5.2. Proceduren EXP.

Deklarationen er:

```
integer procedure EXP(X, xsize, nx, A, asize, na, XN, xnsize, nxn);
value xsize, nx, na, nxn;
integer xsize, nx, asize, na, xnsize, nxn;
integer array X, A, XN;
begin
   boolean out;
   integer tsize, nt, m;
   integer array TERM[0:c39];
   nt := ftrack;
   ftrack := ftrack + 1;
   ASSIGN(1, A, asize, na);
   COPY(X, xsize, nx, TERM, tsize, nt);
   ADD(X, xsize, nx, 1, A, asize, na);
   out := false;
   m := 1;
   for m := m + 1 while -, out do
   begin
      LONGMULT(X, xsize, nx, TERM, tsize, nt, XN, xnsize, nxn);
      COPY(XN, xnsize, nxn, TERM, tsize, nt);
      DIVIDE(TERM, tsize, nt, m, out);
      ADD(TERM, tsize, nt, 1, A, asize, na)
   end for m;
   ftrack := ftrack - 1
end EXP;
```

Proceduren udfører operationen:

$$A := \exp(X);$$

hvor A og X begge er store registre. Proceduren har brug for to ekstra, store registre: XN og TERM, og da der i hovedprogrammet er gjort plads til tre faste, store registre, er det økonomisk at lade et af hjælpe-registrene være identisk med et af registrene i hovedprogrammet. Dette gøres muligt ved at lade XN være en formel parameter, ligesom A og X.

Vi har da de tre store registre som formelle parametre, hver karakteriseret ved de sædvanlige tre størrelser:

```
X, xsize, nx;
A, asize, na;
XN, xnsize, nxn;
```

Det andet store hjælperegister, TERM, oprettes lokalt i proceduren. Grundlaget for beregningen er den velkendte rækkeudvikling:

$$\exp(X) := 1 + X + X\uparrow 2/2 + X\uparrow 3/(2\times 3) + \ldots\ldots$$

Først beregner vi:

```
A := 1;
TERM := X;
A := A + X;
```

som operationer paa de store registre ved kald af procedurerne ASSIGN, COPY og ADD. Derefter sættes tælleværket, m, lig med 1, og vi gennemløber for-sætningen:

```
for m := m + 1 while -, out do
```

For hver værdi af m udføres følgende operationer, noteret paa almindelig ALGOL-form:

```
XN := X×TERM;
TERM := XN;
TERM := TERM/m;
A := A + TERM;
```

Værdien af TERM bliver altsaa multipliceret med X, divideret med m og adderet til A. I selve proceduren sker de fire operationer ved kald af de fire procedurer: LONGMULT, COPY, DIVIDE og ADD.

Her faar vi brug for parametren empty i DIVIDE-proceduren. Den aktuelle parameter er her kaldt: out, og saa snart divisionen med m har fjernet alle betydende cifre fra TERM, sættes out til sand og for-sætningen afbrydes.

5.3. Proceduren PI TO.

Denne procedure har deklarationen:

```
integer procedure PI TO(A, asize, na, T2, t2size, n2, T3, t3size, n3);
value na, n2, n3;
integer asize, na, t2size, n2, t3size, n3;
integer array A, T2, T3;
begin
   boolean out1, out2, out3, out;
   integer factor, m, ns, n1, ssize, t1size;
   integer array SUM, T1[0:c39];
   ns := ftrack;
   n1 := ns + 1;
   ftrack := ftrack + 2;
   ASSIGN(0, A, asize, na);
   ASSIGN(3, T1, t1size, n1);
   out1 := false;
   ASSIGN(24, T2, t2size, n2);
   DIVIDE(T2, t2size, n2, 171, out2);
   ASSIGN(24, T3, t3size, n3);
   DIVIDE(T3, t3size, n3, 1434, out3);
   factor := m := - 1;
   for m := m + 2 while -, out1 do
   begin
      ASSIGN(0, SUM, ssize, ns);
      ADD(T1, t1size, n1, 1, SUM, ssize, ns);
      if -, out2 then
      ADD(T2, t2size, n2, 1, SUM, ssize, ns);
      if -, out3 then
      ADD(T3, t3size, n3, 1, SUM, ssize, ns);
      DIVIDE(SUM, ssize, ns, m, out);
      factor := - factor;
      ADD(SUM, ssize, ns, factor, A, asize, na);
      DIVIDE(T1, t1size, n1, 64, out1);
      if -, out2 then
      DIVIDE(T2, t2size, n2, 3249, out2);
```

```
      if ¬, out3 then
      DIVIDE(T3, t3size, n3, 57121, out3)
   end for m;
   ftrack := ftrack - 2
end PI TO;
```

Proceduren tildeler det store register, A, værdien pi:

```
A := 3.14159 26535 ..........
```

Til beregningen benyttes fire hjælperegistre: T1, T2, T3, og SUM. Af samme grund som nævnt under EXP-proceduren er to af hjælperegistrene gjort til formelle parametre (T2, T3) medens de to andre (T1 og SUM) maa oprettes lokalt i proceduren. Vi faar derfor de sædvanlige 3×3 formelle parametre i proceduren.

Værdien af pi beregnes efter en rækkeudvikling, der er noget mere kompliceret end den, der blev brugt for e-funktionen. Formelapparatet er taget fra en artikel af Shanks og Wrench (1962), som ogsaa anfører en beregnet værdi med 100000 decimaler.

Formlerne er:

$$pi := 24\times\arctan(1/8) + 8\times\arctan(1/57) + 4\times\arctan(1/239);$$

Funktionen arctan bestemmes ved rækkeudviklingen:

$$\arctan(x) := x - x\uparrow 3/3 + x\uparrow 5/5 - x\uparrow 7/7 + \ldots\ldots$$

Vi ser af rækkeudviklingen, at hvis vi sørger for at gemme x, $x\uparrow 3$, $x\uparrow 5$, o.s.v., kan vi beregne det nye led af det forrige ved at multiplicere med $x\uparrow 2$ og dividere med 3, 5, 7, o.s.v. Nu har x de faste værdier:

$$1/8 \qquad 1/57 \qquad 1/239$$

hvorfor $x\uparrow 2$ faar de faste værdier:

$$1/64 \qquad 1/3249 \qquad 1/57121$$

De tre nye led i de tre rækkeudviklinger faas af de forrige potenser ved at dividere med 64, 3249 og 57121. Divisionen med 3, 5, 7, o.s.v. kan udføres efter addition af de tre nye led.

Vi kan nu opstille følgende ALGOL-beskrivelse af proceduren, hvori de store registre noteres som almindelige variable:

```
A := 0;
T1 := 3;
T2 := 24;
T2 := T2/171;
T3 := 24;
T3 := T3/1434;
factor := m := - 1;
for m := m + 2 while -, out1 do
begin
   SUM := T1 + T2 + T3;
   SUM := SUM/m;
   factor := - factor;
   A := A + factor×SUM;
   T1 := T1/64;
   T2 := T2/3249;
   T3 := T3/57121
end for m;
```

I selve proceduren er de forskellige operationer udført som de nødvendige procedurekald med de store registre som parametre. Bemærk, at den første summation:

```
SUM := T1 + T2 + T3;
```

udføres paa den maade, at SUM først nulstilles og T1 adderes. Den yderligere addition af T2 og T3 udføres kun hvis de tilsvarende logiske variable, out2 og out3, er falske. Værdien af out2 og out3 sættes under divisionerne sidst i proceduren. T3 bliver først tom, derefter T2 og til sidst T1, hvorefter beregningen er slut. Bemærk ogsaa, hvorledes sætningen:

```
factor := - factor;
```

lader factor skifte mellem 1 og -1, saaledes at vi faar skiftevis addition og subtraktion.

5.4. Proceduren SQRT.

Deklarationen er:

```
integer procedure SQRT(x, A, asize, na, B, bsize, nb, C, csize, nc);
value x, na, nb, nc;
real x;
integer asize, na, bsize, nb, csize, nc;
integer array A, B, C;
begin
   boolean empty;
   integer xsize, zsize, nx, nz, i, imax;
   integer array X, Z[0:c39];
   nx := ftrack;
   nz := nx + 1;
   ftrack := ftrack + 2;
   ASSIGN(x, X, xsize, nx);
   ASSIGN(sqrt(x), A, asize, na);
   ASSIGN(1/sqrt(x), Z, zsize, nz);
   imax := if asize > zsize then asize else zsize;
   imax := imax - decimals + 1;
   for i := 1 step 1 until imax do
   begin
      LONGMULT(A, asize, na, Z, zsize, nz, B, bsize, nb);
      ASSIGN(2, C, csize, nc);
      ADD(B, bsize, nb, -1, C, csize, nc);
      LONGMULT(Z, zsize, nz, C, csize, nc, B, bsize, nb);
      LONGMULT(B, bsize, nb, X, xsize, nx, C, csize, nc);
      COPY(B, bsize, nb, Z, zsize, nz);
      ADD(C, csize, nc, 1, A, asize, na);
      DIVIDE(A, asize, na, 2, empty)
   end for i;
   ftrack := ftrack - 2;
end SQRT;
```

Proceduren beregner:

```
A := sqrt(x);
```

hvor x er en normal variabel af typen real, og A er et stort register. Som hjælperegistre bruges dels de to formelle parametre, B og C, og dels to lokale registre, X og Z.

Til iterativ beregning af kvadratroden af x bruger man sædvanligvis formlen:

$$a(n+1) = (a(n) + x/a(n))/2$$

hvori a(n) er iteration nr. n. Denne metode kan desværre ikke bruges her, da vi ikke har lavet nogen procedure til division med et meget stort tal.

Vi kan da istedet bruge en metode, som beregner den reciprokke værdi af et opgivet tal, b, ved iteration. Er y(n) iteration nr. n til den reciprokke værdi af b, lyder formlen:

$$y(n+1) = y(n)\times(2 - b\times y(n))$$

Naar denne formel bruges til iteration paa 1/a(n) i den forrige formel, kan man paa denne maade faa en iterativ tilnærmelse til kvadratroden af x uden brug af division med store registre.

Naar procedurens beregningsgang reproduceres ved hjælp af ALGOL-sætninger, hvor de store registre optræder, som om de var simple variable, kan procedurens indhold udtrykkes saaledes:

```
X := x;
A := sqrt(x);
Z := 1/sqrt(x);
for i := 1 step 1 until imax do
begin
   B := A×Z;
   C := 2 - B;
   B := Z×C;
   C := B×X;
   Z := B;
   A := A + C;
   A := A/2
end for i;
```

Det kritiske problem her er antallet af iterationer, imax, som er nødvendigt for at sikre konvergens. Der er ikke her udført noget større arbejde for at finde en sikker værdi af imax, men proceduren er rent foreløbigt indrettet til at sætte imax lig med antallet af celler, som er forskellige fra nul i den første tilnærmelse til sqrt(x) eller 1/sqrt(x). Der vælges den største af de to værdier. Denne metode har vist sig tilfredsstillende, i hvert fald for et beskedent antal celler. Metoden er formentlig konservativ, hvis antallet af celler er meget stort.

5.5. Proceduren EXPISQN.

Dette er en ganske speciel procedure af forsvindende praktisk betydning, men med en vis talteoretisk interesse. Deklarationen er:

```
integer procedure EXPISQN(N, A, asize, na, B, bsize, nb, C, csize, nc);
value N, na, nb, nc;
integer N, asize, na, bsize, nb, csize, nc;
integer array A, B, C;
begin
   integer xsize, nx;
   integer array X[0:c39];
   nx := ftrack;
   ftrack := ftrack + 1;
   PI TO(A, asize, na, B, bsize, nb, C, csize, nc);
   SQRT(N, B, bsize, nb, C, csize, nc, X, xsize, nx);
   LONGMULT(A, asize, na, B, bsize, nb, C, csize, nc);
   EXP(C, csize, nc, A, asize, na, B, bsize, nb);
   ftrack := ftrack - 1
end EXPISQN;
```

Proceduren beregner det ejendommelige udtryk:

$$A := \exp(\pi \times \sqrt{N});$$

N er en simpel, normal, variabel af typen integer. Vi beregner kvadratroden af N i et stort register, assigner pi til et andet stort register, multiplicerer pi med sqrt(N), og beregner endelig eksponentialfunktionen til resultatet. Formelle parametre i proceduren er dels N, dels det store register A med hjælpeparametrene asize og na. Der

bruges endvidere tre store hjælperegistre, hvoraf de to (B og C) er formelle parametre og det tredje (X) oprettes lokalt.

Beregningen foregaar efter skemaet:

```
A := pi;
B := sqrt(N);
C := A×B;
A := exp(C);
```

hvor A, B og C betyder store registre.

Det interessante ved exp(pi×sqrt(N)) er, at for visse værdier af N vil denne funktion antage en værdi, som er meget tæt ved et helt tal. Bowden (1953) anfører eksemplet N = 163, som er:

262 53741 26407 68743.99999 99999 99250 .....

I eksemplet fra en typisk kørsel side 78 viser vi værdien for N = 37:

1991 48647.99997 80464 15400 26939 .....

som ogsaa er meget nær ved et helt tal.

En nærmere teoretisk forklaring paa dette fænomen findes hos Ramanujan (1914), Hardy, Aiyar og Wilson (1927), Hardy (1940) og Hermite (1908), men disse forklaringer er ikke særligt gennemskuelige for matematiske amatører.

Det foreliggende program er særligt velegnet til beregning af denne funktion, fordi baade mellemregningerne og det færdige resultat kræver mange betydende cifre.

6. TABELPROCEDURER

I dette kapitel omtales tre procedurer til beregning af tabeller over de tre funktioner:

FAC(N)	,	fakultetfunktionen
a↑N	,	potensfunktion
N↑b	,	potensfunktion

a og b er opgivne talkonstanter. Værdien af N varieres fra en opgivet startværdi med en opgivet tilvækst til en opgivet slutværdi.

Endelig omtales en fjerde procedure til beregning af antallet af isomere, monovalente alkoholer.

6.1. Proceduren FACTAB.

Deklarationen er:

```
integer procedure FACTAB(from, step, to, A, asize, na);
value from, step, to, na;
integer from, step, to, asize, na;
integer array A;
begin
   integer N, n;
   ACCEPT(limit > 1 + 0.05×to×ln(to));
   ASSIGN(1, A, asize, na);
   for N := 2 step 1 until from -1 do
   MULT(A, asize, na, N);
   n := step -1;
   for N := from step 1 until to do
   begin
      MULT(A, asize, na, N);
      n := n + 1;
      if n = step then
      begin
```

```
        n := 0;
        LINE;
        writetext({<N: });
        write integer({-ddddd}, N);
        writetext({<,  FAC(N): });
        PRINT(A, asize, na)
      end if n
  end for N
end FACTAB;
```

Proceduren beregner en tabel over fakultetsfunktionen:

$$FAC(N) := 1 \times 2 \times 3 \times 4 \ldots\ldots\ldots \times N;$$

Funktionen udskrives for følgende værdier af N:

from
from + step
from + 2×step
.
.
.
.
to

De tre normale, simple heltalsvariable, from, step og to, er formelle parametre. Værdien af FAC(N) fremkommer i det store register, A, karakteriseret ved de tre formelle parametre, A, asize og na.

Proceduren kontrollerer først ved et kald af ACCEPT, at antallet af heltalscifre, E, er tilstrækkeligt stort. Der benyttes den noget konservative betingelse:

$$limit > 1 + 0.05 \times to \times \ln(to);$$

Derefter sættes A lig med 1, og vi beregner FAC(from-1) ved:

```
for N := 2 step 1 until from-1 do
MULT(A, asize, na, N);
```

Derefter fortsætter multiplikationerne med en ny for-sætning:

```
for N := from step 1 until to do
begin
   MULT(A, asize, na, N);
   ......
```

Her er indsat en tælling i hjælpevariablen, n, saaledes at vi faar udskrift af FAC(N) for hver step værdier af N.

6.2. Proceduren POWTAB1.

Deklarationen er:

```
integer procedure POWTAB1(from, step, to, a, A, asize, na);
value from, step, to, a, na;
integer from, step, to, a, asize, na;
integer array A;
begin
   integer N, n;
   ACCEPT(limit > 1 + 0.05×to×ln(a));
   LINE;
   writetext(‹<a: ›);
   write integer(‹-ddddddddddd›, a);
   ASSIGN(1, A, asize, na);
   for N := 1 step 1 until from -1 do
   MULT(A, asize, na, a);
   n := step -1;
   for N := from step 1 until to do
   begin
      MULT(A, asize, na, a);
      n := n + 1;
      if n = step then
      begin
         n := 0;
         LINE;
         writetext(‹<N: ›);
```

```
        write integer(¢-ddddd¢, N);
        writetext(¢<, a↑N: ¢);
        PRINT(A, asize, na)
      end if n
   end for N
end POWTAB1;
```

Proceduren beregner en tabel over a↑N, hvor a er en formel parameter i form af en normal, simpel heltalsvariabel. Værdien af a↑N udskrives for N lig med from, from + step, from + 2×step, ... to, hvor from, step og to ogsaa er tre normale, simple heltalsvariable givet som parametre. Værdien af a↑N fremkommer i det store register A, givet ved de tre formelle parametre, A, asize og na.

Ligesom i FACTAB kontrolleres først E med ACCEPT:

```
limit > 1 + 0.05×to×ln(a);
```

Derefter beregnes a↑(from - 1), og vi faar den egentlige for-sætning:

```
for N := from step 1 until to do
begin
   MULT(A, asize, na, a);
   ....
   ....
```

Værdien af a↑N udskrives for hver step værdier af N.

6.3. Proceduren POWTAB2.

Denne procedure beregner ogsaa en tabel over potensfunktionen, men nu ligger potensen fast og roden varieres: N↑b. Parametrene er de samme som i POWTAB1, blot er a erstattet med b. Deklarationen er:

```
integer procedure POWTAB2(from, step, to, b, A, asize, na);
value from, step, to, b, na;
integer from, step, to, b, asize, na;
integer array A;
```

```
begin
   integer N, n;
   ACCEPT(limit > 1 + 0.05×b×ln(to));
   LINE;
   writetext({<b: });
   write integer({-ddddddddddd}, b);
   for N := from step step until to do
   begin
      ASSIGN(1, A, asize, na);
      for n := 1 step 1 until b do
      MULT(A, asize, na, N);
      LINE;
      writetext({<N: });
      write integer({-ddddddddddd}, N);
      writetext({<, N^b: });
      PRINT(A, asize, na)
   end for N
end POWTAB2;
```

Beregningsgangen er lidt forskellig fra POWTAB1, fordi vi for hver værdi af N skal multiplicere op helt fra 1 til b.

6.4. Proceduren ISOM.

Denne procedure beregner en tabel over antallet af isomere, monovalente, alifatiske alkoholer med et antal kulstofatomer, som varieres fra 1 til N. Tabellen, der trykkes, indeholder tre tal for hver værdi af N: antallet af primære, sekundære og tertiære alkoholer.

Deklarationen er:

```
integer procedure ISOM(N, PRI, psize, np, SEC, ssize,
ns, TER, tsize, nt);
value N, np, ns, nt;
integer N, psize, np, ssize, ns, tsize, nt;
integer array PRI, SEC, TER;
begin
   integer sbase, nu, nv, usize, vsize, k, n, m, i, j, si, sj, q, sk, f;
```

```
boolean empty;
integer array U, V[0:c39];
integer procedure size(n);
value n;
integer n;
begin
   get(U, FREE, sbase + n:40);
   size := U[n mod 40]
end size;
procedure store(n, size);
value n, size;
integer n, size;
begin
   get(U, FREE, sbase + n:40);
   U[n mod 40] := size;
   put(U, FREE, sbase + n:40)
end store;
large := true;
nu := ftrack;
nv := nu + 1;
f:=nv+1;
sbase:=1+step×(1+f+N);
ftrack:=f+1+N+1+N:40;
ASSIGN(1, PRI, psize, f);
store(0, psize);
for n := 1 step 1 until N do
begin
   ASSIGN(0, SEC, ssize, ns);
   ASSIGN(0, TER, tsize, nt);
   m := (n - 1):2;
   for i := 1 step 1 until m do
   begin
      j := n - 1 - i;
      si := size(i);
      sj := size(j);
      if i < j then
      LONGMULT(PRI, si,
      f + i, U, sj, f + j, V, vsize, nv)
      else
```

```
    begin
        ASSIGN(1, U, usize, nu);
        ADD(V, sj, f+ j, 1,
        U, usize, nu);
        LONGMULT(PRI, si,
        f + i, U, usize, nu, V, vsize, nv);
        DIVIDE(V, vsize, nv, 2, empty)
    end i ≥ j;
    ADD(V, vsize, nv, 1, SEC, ssize, ns)
end for i;
m := (n - 2):2;
for i := 1 step 1 until m do
begin
    j := n - 1 - 2×i;
    si := size(i);
    sj := size(j);
    ASSIGN(if i ≠ j then 0 else 2, U, usize, nu);
    ADD(PRI, sj, f + j, 1,
    U, usize, nu);
    LONGMULT(PRI, si,
    f + i, U, usize, nu, V, vsize, nv);
    ADD(U, usize, nu, 1, V, vsize, nv);
    LONGMULT(V, vsize, nv,
    PRI, si, f + i, U, usize, nu);
    DIVIDE(U, usize, nu, if i ≠ j then
    2 else 6, empty);
    ADD(U, usize, nu, 1, TER, tsize, nt)
end for i;
m := (n - 4):3;
for i := 1 step 1 until m do
begin
    q := (n - 2 - i):2;
    for j := i + 1 step 1 until q do
    begin
        k := n - 1 - i - j;
        si := size(i);
        sj := size(j);
        sk := size(k);
```

```
            LONGMULT(PRI,
            sj, f + j, U, sk, f + k, V, vsize, nv);
            LONGMULT(V, vsize,
            nv, PRI, si, f + i, U, usize, nu);
            ADD(U, usize, nu, 1, TER, tsize, nt)
         end for j
      end for i;
      LINE;
      writetext({<N: });
      write integer({-ddddddd}, n);
      LINE;
      writetext({<PRI(N):});
      PRINT(PRI, size(n - 1), f + n - 1);
      LINE;
      writetext({<SEC(N):});
      PRINT(SEC, ssize, ns);
      LINE;
      writetext({<TER(N):});
      PRINT(TER, tsize, nt);
      LINE;
      ADD(TER, tsize, nt, 1, SEC, ssize, ns);
      ADD(PRI, size(n - 1), f + n - 1, 1,
      SEC, ssize, ns);
      COPY(SEC, ssize, ns, U, usize, f + n);
      store(n, usize)
   end for n;
   ftrack := ftrack - (4 + N + N:40)
end ISOM;
```

Proceduren har som formelle parametre dels N, det højeste antal kulstofatomer, som tabellen skal beregnes for, og som er en normal heltalsvariabel, dels tre store registre, hver med de sædvanlige tre parametre:

```
PRI, psize, np;
SEC, ssize, ns;
TER, tsize, nt;
```

som under beregningen indeholder antallet af primære, sekundære og tertiære variable. Desuden bruges to lokale store registre, U og V.

Beregningsmetoden er hentet fra Henze og Blair (1931). I det følgende giver vi en symbolsk ALGOL-beskrivelse af metoden, hvori de store registre, PRI, SEC, TER, U og V noteres, som om de var simple variable. Det totale antal alkoholer findes som summen af primære, sekundære og tertiære alkoholer:

```
TOT := PRI + SEC + TER;
```

Det er her nødvendigt at operere med et array TOT[0:N] af store registre. Det symbolske ALGOL-program er:

```
TOT[0] := 1;
for n := 1 step 1 until N do
begin
   SEC := TER := 0;
   m := (n-1):2;
   for i := 1 step 1 until m do
   begin
      j := n - 1 - i;
      if i < j then V := TOT[i]×TOT[j]
      else
      begin
         U := 1;
         U := U + TOT[j];
         V := U×TOT[i];
         V := V:2
      end i ≥ j;
      SEC := SEC + V
   end for i;
   m := (n-2):2;
   for i := 1 step 1 until m do
   begin
      j := n - 1 - 2×i;
      U := if i ≠ j then 0 else 2;
      U := U + TOT[j];
      V := U×TOT[i];
      V := V + U;
      U := V×TOT[i];
      U := U:(if i ≠ j then 2 else 6);
```

```
      TER := TER + U
    end for i;
    m := (n-4):3;
    for i := 1 step 1 until m do
    begin
      q := (n - 2 - i):2;
      for j := i + 1 step 1 until q do
      begin
        k := n - 1 - i - j;
        V := TOT[j]×TOT[k];
        U := V×TOT[i];
        TER := TER + U
      end for j
    end for i;
    comment Tryk TOT[n-1], SEC og TER;
    TOT[n] := TOT[n-1] + SEC + TER
  end for n;
```

For at kunne operere med talsættet TOT[0:N] paa en nogenlunde bekvem maade, sætter proceduren udtrykkeligt large til sand, saaledes at talsættet altid gemmes paa disken, selv om der maaske er reserveret færre end 40 celler til hvert stort register. Der bruges følgende n-værdier til de nyoprettede registre:

```
U:       ftrack
V:       ftrack + 1
TOT[0]:  f = ftrack + 2
TOT[1]:  f + 1
.
.
.
.
TOT[N]   f + N
```

Endvidere reserveres der 1 + N:40 diskkanaler til lagring af asize--værdierne for talsættet TOT[0:N]. Der er deklareret en særlig procedure, size(n), som henter værdien af asize for TOT[n] fra disken, samt en anden lokal procedure, store(n, size), som lagrer værdien af asize

for TOT[n] paa disken.

I selve proceduren er det vist, hvorledes det symbolske ALGOL-program udtrykkes ved hjælp af de forskellige standardprocedurer i programmet.

7. ORDRESTRUKTUR

Da de forskellige regneprocedurer arbejder temmeligt langsomt, især for meget store tal, er det fuldt ud tilstrækkeligt at anvende en relativ langsom interpretativ metode til at specificere de forskellige mulige beregninger.

Vi benytter derfor et simpelt ordresystem, hvor tallene fra 1-22 har følgende betydning:

No:

```
 1:   A := typein;
 2:   write(A);
 3:   B := A;
 4:   C := A;
 5:   A := B;
 6:   C := B;
 7:   A := C;
 8:   B := C;
 9:   A := A + B;
10:   A := A - B;
11:   A := A×typein;
12:   A := A/typein;
13:   C := A×B;
14:   A := PI;
15:   A := exp(B);
16:   A := sqrt(typein);
17:   A := exp(PI×sqrt(typein));
18:   table of factorial function;
19:   table of a↑N;
20:   table of N↑b;
21:   table of alcohol isomers;
22:   STOP;
```

7.1. Proceduren CALCULATE.

Deklarationen er:

```
procedure CALCULATE;
begin
   integer array A, B, C[0:c39];
   integer procedure next;
   begin
      integer x;
      if show then LINE;
```

```
      writetext({<r := });
      x := read integer;
      if show then write({ddddddddd}, x);
      next := x
   end next;
   integer procedure STOP;
   go to EX;
   procedure ORDER(text, command);
   string text;
   integer command;
   begin
      integer dummy;
      type := type + 1;
      if type = TYPE then
      begin
         writetext(text);
         dummy := command;
         go to NEW
      end if this type
   end ORDER;
   ftrack := 4;
NEW: LINE;
   LINE;
   writetext({<No: });
   TYPE := read integer;
   if show then write ({dd}, TYPE);
   type := 0;
   ORDER({< A := r;}, ASSIGN(next, A, asize, 1));
   ORDER({< write(A);}, PRINT(A, asize, 1));
   ORDER({< B := A;}, COPY(A, asize, 1, B, bsize, 2));
   ORDER({< C := A;}, COPY(A, asize, 1, C, csize, 3));
   ORDER({< A := B;}, COPY(B, bsize, 2, A, asize, 1));
   ORDER({< C := B;}, COPY(B, bsize, 2, C, csize, 3));
   ORDER({< A := C;}, COPY(C, csize, 3, A, asize, 1));
   ORDER({< B := C;}, COPY(C, csize, 3, B, bsize, 2));
   ORDER({< A := A + B;}, ADD(B, bsize, 2, 1, A, asize, 1));
   ORDER({< A := A - B;}, ADD(B, bsize, 2, -1, A, asize, 1));
   ORDER({< A := A*r;}, MULT(A, asize, 1, next));
```

```
   ORDER({< A := A/r;}, DIVIDE(A, asize, 1, next, empty));
   ORDER({< C := A×B;}, LONGMULT(A, asize, 1, B, bsize, 2, C, csize, 3));
   ORDER({< A := PI;}, PI TO(A, asize, 1, B, bsize, 2, C, csize, 3));
   ORDER({< A := exp(B);}, EXP(B, bsize, 2, A, asize, 1, C, csize, 3));
   ORDER({< A := sqrt(r);}, SQRT(next, A, asize, 1, B, bsize, 2, C, csize, 3));
   ORDER({< A := exp(PI×sqrt(r));},
   EXPISQN(next, A, asize, 1, B, bsize, 2, C, csize, 3));
   ORDER({< FACTORIAL TABLE(r, r, r);},
   FACTAB(next, next, next, A, asize, 1));
   ORDER({< POWER TABLE(r, r, r, r↑variable);},
   POWTAB1(next, next, next, next, A, asize, 1));
   ORDER({< POWER TABLE(r, r, r, variable↑r);},
   POWTAB2(next, next, next, next, A, asize, 1));
   ORDER({< ISOMER TABLE(r);},
   ISOM(next, A, asize, 1, B, bsize, 2, C, csize, 3));
   ORDER({< stop}, STOP);
   go to NEW;
EX:end CALCULATE;
```

Proceduren danner den blok, hvori de tre hovedregistre er deklareret:

```
integer array A, B, C[0:c39];
```

Endvidere er der deklareret tre lokale procedurer: next, STOP og ORDER.

Da nogle af de 22 mulige ordrer kræver ekstra tal til definition af operationen, maa vi have en metode til at levere disse. Dette er yderligere kompliceret ved at vi ønsker at køre programmet saaledes, at der først indlæses ordrer og tal fra en strimmel til demonstration af typiske muligheder, og derefter kan de tilstedeværende skrive ordrer og tal paa skrivemaskinen, hvis man ønsker at udføre specielle beregninger. Der benyttes en global logisk variabel, show, til at skelne mellem de to faser. Programmet sætter først show til sand og derefter falsk.

Proceduren next læser et heltal via det valgte medium:

```
x := read integer;
```

Hvis show er sand, skrives der først en ny linie paa skrivemaskinen, og den indlæste talværdi udskrives.

Proceduren STOP foretager blot et hop til slutningen af ordrelisten.

Proceduren ORDER har de to parametre: text og command. Meningen er, at man sidst i proceduren gennemløber en række kald af ORDER:

```
ORDER({<Forklaring 1>, PROCEDURE 1);
ORDER({<Forklaring 2>, PROCEDURE 2);
.....
.....
```

Naar ordrenummeret er indlæst med:

```
TYPE := read integer;
```

sættes et tælleværk, type, lig med nul. Derefter gennemløbes kaldene af ORDER. I hvert kald øges type med 1. Hvis type er forskellig fra TYPE, sker der ikke mere i ORDER, men saa snart de bliver ens, udskrives der den forklarende tekst, og den egentlige operation udføres med:

```
dummy := command;
```

Da alle regneprocedurerne er deklareret som en integer procedure, er denne metode anvendelig, men man kunne naturligvis ogsaa have deklareret dem som almindelige procedurer og skrevet:

```
command;
```

med command specificeret som en procedure. Efter udførelse af proceduren hoppes til etiketten NEW, hvor næste ordrenummer indlæses.

Proceduren begynder som nævnt med indlæsning af ordrenummeret, og det udskrives, hvis show er sand. Derefter gennemløbes ordrelisten. Dette kunne naturligvis ogsaa have været programmeret som en case-sætning.

Nogle af ordrerne kræver yderligere talindlæsning med proceduren next. Dette gælder følgende ordrer:

1: ASSIGN. Der assignes next til A.

11: MULT. Faktoren indlæses som next.

12: DIVIDE. Der divideres med next.

16: SQRT. Vi beregner kvadratroden af next.

17: EXPISQN. Vi beregner exp(pi×sqrt(next)).

18 - 20: Tabelprocedurer. Startværdi, trinlængde og slutværdi indlæses med next. I ordre 19 og 20 indlæses henholdsvis a og b med next.

21: ISOM. Den øvre grænse for antallet af kulstofatomer indlæses med next.

8. PROGRAMMET.

8.1. Programmets opbygning.

En udskrift af ALGOL-programmet er vist i afsnit 10.1.

Den yderste blok indeholder følgende simple variable:

Logiske variable:

first: Hjælpevariabel for PRINT, etc.

empty: Hjælpevariabel ved DIVIDE.

show: Sand, naar programmet indlæser ordrenumre fra papirbaand, falsk, naar ordrenumrene indlæses fra skrivemaskinen.

large: Sand, naar der opereres med tal paa over 400 betydende cifre.

Heltalsvariable:

linerest: Antal frie linier, som er tilbage paa siden.

lang: Sprogtæller: 1: Dansk, 2: Engelsk, 3: Fransk, 4: Tysk.

decimals: Antal celler reserveret til lagring af decimaler i et stort register, med negativt fortegn.

limit: Antal celler reserveret til lagring af heltalscifre er limit + 1.

carry: Mentecelle.

count: Tælleværk for forskellige procedurer.

MODUL: $1_{10}10$.

cell, cell2: Hjælpeceller.

asize, bsize, csize: Gemmer størrelsen af de tre hovedregistre: A, B og C.

type, TYPE: Tælleværker for proceduren CALCULATE.

D, E: Antal decimaler, henholdsvis heltalscifre.

FREE: Beskrivelsesord for disk-arealet: free.

ftrack: Tælleværk for tælling af registernumre paa disken.

step: Antal disk-kanaler per register.

c39: Registrenes størrelse i lageret er [0:c39].

Proceduredeklarationerne er ikke vist i programudskriften her.

Programmet begynder med at indsætte startværdien af linerest og vær-

dien af MODUL.

Sætningen: select(17) vælger skrivemaskinen til input- og output--medium.

Der vælges sprog med SELECT LANGUAGE og udskrives en programbeskrivelse paa tre linier. Endvidere udskrives en ordreliste med forklaring.

Saa udskrives teksten:

Vi lader først maskinen demonstrere nogle eksempler:

Vi sætter show til sand og vælger strimmellæser som input og skrivemaskine som output med select(16). Inputmaterialet er vist umiddelbart efter programmet.

Talmaterialet indlæses med for-sætningen:

for D := read integer while D ≥ 0 do

Naar D bliver negativ, standses for-sætningen. Naar D ≥ 0, indlæses ogsaa værdien af E:

E := read integer;

Værdien af D og E udskrives og omregnes til celleantallene: decimals og limit som vist side 9. Vi beregner ogsaa værdien af step, large og c39.

Saa kaldes proceduren CALCULATE, som indlæser ordrenumrene og udfører regneoperationerne. Naar stop-ordren 22 mødes, hoppes ud af CALCULATE.

Naar for-sætningen møder D = -1, slutter indlæsningen fra papirstrimlen. Vi sætter show til falsk, vælger skrivemaskinen til input og output og skriver sætningen:

Nu kan De forsøge:

Saa følger ny for-sætning, omtrent som i første fase, med indlæsning af D og E fra skrivemaskinen. Beregning af decimals, limit, step, large og c39 er mage til før. Endelig kaldes CALCULATE til indlæsning af ordrenumre og udførelse af beregningerne. Programmet slutter, naar der indlæses et negativt D.

8.2. Diskussion af en kørsel.

En typisk kørselsudskrift er vist i afsnit 10.2.

Den første serie beregninger udføres med 20 decimaler og op til 30 heltalscifre. Vi ser eksempler paa indlæsning, trykning, division, flytning, multiplikation og addition. Endvidere ser vi eksempler paa beregning af pi, e og kvadratroden af 2. Som kontrol beregnes kvadratet paa kvadratroden. Det bemærkes, at beregningerne udføres i registre med 20 decimaler, ikke nødvendigvis med 20 rigtige decimaler. Der er derfor smaa afrundingsfejl.

Der følger derefter en serie beregninger med 300 heltalscifre og ingen decimaler. Vi beregner fakultetsfunktionen for N = 10, 20, ..., 100, og følgende værdier af potensfunktionen:

$$2\uparrow 32,\quad 2\uparrow 64,\quad 2\uparrow 96 \ldots$$
$$2\uparrow 64,\quad 3\uparrow 64,\quad 4\uparrow 64, \ldots\ldots\ldots\ 10\uparrow 64$$

Saa kommer en serie med 1600 heltalscifre og nul decimaler. Hvert register disponerer altsaa over 4 disk-kanaler. Her beregnes fakultetsfunktionen for N = 100, 200, ... 500.

Derefter følger en beskeden tabel over alkoholisomere op til N = 10.

Sidste eksempel i show-fasen er beregning af exp(pi×sqrt(N)) for N = 37.

I anden fase af beregningen har vi beregnet følgende potenser:

$$a\uparrow 50,\quad a\uparrow 100,\quad a\uparrow 150\quad \text{og}\quad a\uparrow 200$$

værdien af a er her valgt til $2\uparrow 32$.

9. REFERENCER

Bowden, B.V.: Faster than thought. A Symposium on Digital Computing Machines, pag. 315 (1953).

Hardy, G.H.: Ramanujan, Cambridge (1940).

Hardy, G.H., Seshu Aiyar, P.V. og Wilson, B.M.: Collected Papers of Srinivasa Ramanujan, Cambridge (1927).

Henze, H.R. og Blair, C.M.: J. Am. Chem. Soc., 53, 3042 (1931).

Hermite, C.: Oeuvres de Charles Hermite, T. 2, pag. 38-82 (1908).

Kjær, J.: Tændstikspillet NIM, Et GIER Demonstrationsprogram, Haldor Topsøe (1966).

Kjær, J.: Primtal, Et GIER Demonstrationsprogram, Haldor Topsøe (1967).

Ramanujan, S.: Modular Equations and Appoximations to pi. Quarterly Journal of Pure and Applied Mathematics, 45, 350-372 (1914).

Shanks, D. og Wrench, J.W.Jr.: Calculation of pi to 100000 decimals. Mathematics of Computation, 16, 76-99 (1962).

10. APPENDIKS

10.1. ALGOL-programmet.

```
Program DEMON-5. Calculation of large numbers.
begin
   boolean first, empty, show, large;
   integer linerest, lang, decimals, limit, carry, count, MODUL, cell,
   cell2, asize, bsize, csize, type, TYPE, D, E, FREE, ftrack, step, c39;
   comment Her anbringes proceduredeklarationerne med comment library
   eller copy;
   linerest := 69;
   MODUL := 10000000000;
   select(17);
   SELECT LANGUAGE;
   LINE;
   WRITE TEXT(
   {<PROGRAM DEMON-5. Beregning af store tal. Programmet simulerer en maskine med
3 registre, A, B og C, som har D decimaler og E cifre før kommaet.
Der anvendes følgende ordresystem:},
   {<PROGRAM DEMON-5. Calculation of large numbers. The program simulates a computer
with 3 registers, A, B, and C, with D decimals and E integer digits.
The following command system is used:},
   {<PROGRAMME DEMON-5. Calcul des nombres eleves. Le programme simule une machine a
3 registres, A, B, et C, avec D decimales et E chiffres entiers.
On utilise les commandes suivantes:},
   {<PROGRAMM DEMON-5. Berechnung von grossen Zahlen. Das Programm simuliert eine
Maschine mit 3 Registern, A, B, und C, mit D Dezimalstellen und E Ganzzahlstellen.
Man verwendet die folgende Befehle:});
   linerest := linerest - 2;
   LINE;
   LINE;
```

```
   writetext(‹<
  No:
   1:     A := typein;        13:     C := AxB;
   2:     write(A);           14:     A := PI;
   3:     B := A;             15:     A := exp(B);
   4:     C := A;             16:     A := sqrt(typein);
   5:     A := B;             17:     A := exp(PIxsqrt(r));
   6:     C := B;             18:     A := table of factorial function;
   7:     A := C;             19:     A := table of a↑N;
   8:     B := C;             20:     A := table of N↑b;
   9:     A := A + B;         21:     A := table of alcohol isomers;
  10:     A := A - B;         22:     STOP;
  11:     A := Axtypein;
  12:     A := A/typein;
›);
   linerest := linerest - 14;
   LINE;
   WRITE TEXT(
   ‹<Vi lader først maskinen demonstrere nogle eksempler:›,
   ‹<We first let the computer show some examples:›,
   ‹<La machine nous donne d abord quelques examples:›,
   ‹<Zuerst zeigt die Maschine einige Beispiele:›);
   show := true;
   select(16);
   for D := read integer while D ≥ 0 do
   begin
      LINE;
      writetext(‹<D:›);
      writeinteger(‹-ddddddd›, D);
      decimals := D;
      if decimals > 0 then
      decimals := -((decimals-1)÷10+1);
      E := read integer;
      LINE;
      writetext(‹<E:›);
      writeinteger(‹-ddddddd›, E);
      limit := (E-1)÷10;
      where(‹<free›, FREE);
      step := (limit - decimals)÷40 + 1;
```

```
      large := step > 1;
      c39 := if large then 39 else limit - decimals;
      CALCULATE;
E1:end for decimals;
   LINE;
   LINE;
   show := false;
   select(17);
   WRITE TEXT(
   {<Nu kan De forsøge:},
   {<Now you may try:},
   {<Maintenant vous pouvez essaier:},
   {<Jetzt koennen Sie versuchen:});
   for D := ASK NUMBER(
   {<Opgiv antal decimåler, D.  -1 er stop},
   {<Specify number of decimals, D.  -1 is stop},
   {<Specifiez le nombre de decimales, D.  -1 est termination},
   {<Bitte, die Anzahl von Dezimalstellen, D, angeben.  -1 ist Schluss})
   while D ≥ 0 do
   begin
      decimals := D;
      if decimals > 0 then
      decimals := -((decimals-1):10 + 1);
      E := ASK NUMBER(
      {<Og antallet af heltalscifre, E},
      {<And the number of integer digits, E},
      {<Et le nombre de chiffres entiers, E},
      {<Und die Anzahl der Ganzzahlstellen, E});
      limit := (E-1):10;
      step := (limit - decimals):40 + 1;
      large := step > 1;
      c39 := if large then 39 else limit - decimals;
```

```
      CALCULATE;
E2:end for decimals
end program;
Input:
20;   30;
 1;  117;   2;
12;    7;   2;   3;  11;   6;   2;
 9;    2;
14;    2;
 1;    1;   2;   3;  15;   2;
16;    2;   2;   3;  13;   7;   2;
22;
 0;  300;
18;   10;  10; 100;
19;   32;  32;  96;   2;
20;    2;   1;  10;  64;
22;
 0; 1600;
18;  100; 100; 500;
22;
 0;  400;
21;   10;
22;
20;   30;
17;   37;    2;
22;
-1;
end of input;
```

10.2. Udskrift fra en kørsel.

```
Select language: d: danish, e: english, f: french, g: german.: d
Dansk

PROGRAM DEMON-5. Beregning af store tal. Programmet simulerer en maskine med
3 registre, A, B og C, som har D decimaler og E cifre før kommaet.
Der anvendes følgende ordresystem:

 No:
  1:    A := typein;        13:    C := A×B;
  2:    write(A);           14:    A := PI;
  3:    B := A;             15:    A := exp(B);
  4:    C := A;             16:    A := sqrt(typein);
  5:    A := B;             17:    A := exp(PI×sqrt(r));
  6:    C := B;             18:    A := table of factorial function;
  7:    A := C;             19:    A := table of a↑N;
  8:    B := C;             20:    A := table of N↑b;
  9:    A := A + B;         21:    A := table of alcohol isomers;
 10:    A := A - B;         22:    STOP;
 11:    A := A×typein;
 12:    A := A/typein;

Vi lader først maskinen demonstrere nogle eksempler:
D:      20
E:      30

No:  1 A := r;
r :=       117

No:  2 write(A);
       117.00000 00000 00000 00000

No: 12 A := A/r;
r :=         7

No:  2 write(A);
       16.71428 57142 85714 28571
```

```
No:  3 B := A;

No: 11 A := Axr;
r :=         6

No:  2 write(A);
        100.28571 42857 14285 71426

No:  9 A := A + B;

No:  2 write(A);
        116.99999 99999 99999 99997

No: 14 A := PI;

No:  2 write(A);
          3.14159 26535 89793 23847

No:  1 A := r;
r :=         1

No:  2 write(A);
          1.00000 00000 00000 00000
No:  3 B := A;

No: 15 A := exp(B);

No:  2 write(A);
          2.71828 18284 59045 23526

No: 16 A := sqrt(r);
r :=         2

No:  2 write(A);
          1.41421 35623 73095 04880

No:  3 B := A;

No: 13 C := AxB;

No:  7 A := C;
```

```
No:  2 write(A);
          1.99999 99999 99999 99998

No: 22 stop
D:       0
E:     300

No: 18 FACTORIAL TABLE(r, r, r);
r :=        10
r :=        10
r :=       100
N:     10,  FAC(N):
   36 28800
N:     20,  FAC(N):
 2432 90200 81766 40000
N:     30,  FAC(N):
        265 25285 98121 91058 63630 84800 00000
N:     40,  FAC(N):
  815 91528 32478 97734 34561 12695 96115 89427 20000 00000
N:     50,  FAC(N):
      30414 09320 17133 78043 61260 81660 64768 84437 76415 68960 51200
00000 00000
N:     60,  FAC(N):
         83 20987 11274 13901 44276 34118 32233 64380 75417 26063 61245
95244 92776 96409 60000 00000 00000
N:     70,  FAC(N):
          1 19785 71669 96989 17960 72783 72168 90987 36458 93814 25464
25857 55536 28646 28009 58278 98453 19680 00000 00000 00000
N:     80,  FAC(N):
 7156 94570 46263 80229 48115 33723 18653 21655 84657 34236 57525 77109
44505 82270 39255 48014 88426 68944 86728 08140 80000 00000 00000 00000
N:     90,  FAC(N):
 1485 71596 44817 61497 30952 27336 20825 73788 55699 61284 68876 69422
16863 70498 53930 94065 87654 59921 31370 88405 96456 17234 46997 81120
00000 00000 00000 00000
N:    100,  FAC(N):
  933 26215 44394 41526 81699 23885 62667 00490 71596 82643 81621 46859
29638 95217 59999 32299 15608 94146 39761 56518 28625 36979 20827 22375
82511 85210 91686 40000 00000 00000 00000 00000
```

```
No: 19 POWER TABLE(r, r, r, r↑variable);
r :=         32
r :=         32
r :=         96
r :=          2
a:            2
N:      32,  a↑N:
42949 67296
N:      64,  a↑N:
18446 74407 37095 51616
N:      96,  a↑N:
 7922 81625 14264 33759 35439 50336

No: 20 POWER TABLE(r, r, r, variable↑r);
r :=          2
r :=          1
r :=         10
r :=         64
b:           64
N:            2,  N↑b:
18446 74407 37095 51616
N:            3,  N↑b:
          3 43368 38202 92512 48465 78490 89281
N:            4,  N↑b:
 3402 82366 92093 84634 63374 60743 17682 11456
N:            5,  N↑b:
      54210 10862 42752 21700 37264 00434 97085 57128 90625
N:            6,  N↑b:
63340 28666 29732 77706 16228 69468 11886 60989 64618 28096
N:            7,  N↑b:
      12197 60487 63583 57001 38573 86256 29718 20755 61529 41312 38401
N:            8,  N↑b:
  627 71017 35386 68076 38357 89423 20766 64161 02355 44446 40345 12896
N:            9,  N↑b:
         11 79018 45777 38583 17152 08728 61412 51866 56782 11592 27584
11090 96961
N:           10,  N↑b:
      10000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000
```

```
No: 22 stop
D:       0
E:    1600

No: 18 FACTORIAL TABLE(r, r, r);
r :=       100
r :=       100
r :=       500
N:    100,  FAC(N):
  933 26215 44394 41526 81699 23885 62667 00490 71596 82643 81621 46859
29638 95217 59999 32299 15608 94146 39761 56518 28625 36979 20827 22375
82511 85210 91686 40000 00000 00000 00000 00000
N:    200,  FAC(N):
      78865 78673 64790 50355 23632 13932 18506 22951 35977 68717 32632
94742 53324 43594 49963 40334 29203 04284 01198 46239 04177 21213 89196
38830 25764 27902 42637 10506 19266 24952 82993 11134 62857 27076 33172
37396 98894 39224 45621 45166 42402 54033 29186 41312 27428 29485 32775
24242 40757 39032 40321 25740 55795 68660 22603 19041 70324 06235 17008
58796 17892 22227 89623 70389 73747 20000 00000 00000 00000 00000 00000
00000 00000 00000 00000
N:    300,  FAC(N):
      30605 75122 16440 63603 53704 61297 26862 93885 88804 17357 69994
16776 74125 94765 33176 71686 74655 15291 42247 75733 49939 14788 87017
26368 86426 39077 59003 15422 68429 27906 97455 98412 25476 93027 19546
04008 01221 57762 52176 85425 59653 56903 50678 87252 64321 89626 42993
65204 57644 88303 88909 75394 34896 25436 05322 59807 76521 27082 24376
39449 12012 86786 75368 30571 22936 81943 64995 64604 98166 45022 77165
00185 17654 64693 40112 22603 47297 24066 33325 85835 06870 15016 97941
68850 35375 21375 54910 28912 64071 57154 83028 22849 37952 63658 01452
35233 15693 64822 33436 79925 45940 95276 82060 80622 32812 38738 38808
17049 60000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000
N:    400,  FAC(N):
 6403 45228 46623 89526 23479 70319 50300 58507 02583 02600 29594 58684
44594 28023 97169 18683 14362 78478 64746 32646 76294 35057 50358 56810
84829 81628 83517 43522 89619 88646 80299 79373 41654 15083 81624 26461
94235 23070 46244 32501 51144 48670 89066 27739 14918 11733 19559 96440
70954 96713 45290 47702 03224 34911 21079 75932 80795 10154 53726 67251
```

```
62787 78900 09349 76376 57103 26350 33153 39653 49868 38683 13393 52024
37378 81577 86791 50631 18587 02618 27016 98197 40062 98302 53085 91298
34616 22723 04558 33952 07596 11505 30223 60868 10433 29725 51948 52674
43223 24386 69948 42240 42325 99805 55161 06359 42376 96139 92319 17134
06385 89965 37970 14782 72066 06320 21737 94720 10321 35662 46138 09077
94230 45973 60699 56759 58360 96158 71512 99138 22286 57857 95493 61617
65448 04532 22007 82581 84008 48436 41559 12294 54275 38480 35583 74518
02267 59000 61399 56014 55952 06127 21119 29181 05032 49100 80000 00000
00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000 00000 00000
N:    500,  FAC(N):
      12201 36825 99111 00687 01238 78542 30469 26253 57434 28031 92842
19241 35883 85845 37315 38819 97605 49644 75022 03281 86301 36164 77148
20358 41633 78722 07817 72004 80785 20515 93292 85477 90757 19393 30603
77296 08590 86270 42917 45478 82424 91272 63443 05670 17327 07694 61062
80231 04526 44218 87878 94657 54777 14986 34943 67781 03764 42740 33827
36539 74713 86477 87849 54384 89595 53753 79904 23241 06127 13269 84327
74571 55463 09977 20278 10145 61081 18837 37095 31016 35632 44329 87029
56389 66289 11658 97476 95720 87926 92887 12817 80070 26517 45077 68410
71962 43903 94362 53642 26052 34945 85012 99185 71501 24870 69615 68141
62535 90566 93423 81300 88562 49246 89156 41267 75654 48188 65065 93847
95177 53608 94005 74523 89403 35798 47636 39449 05313 06232 37490 66445
04882 46650 75946 73586 20746 37925 18420 04593 69692 98102 22639 71952
59719 09452 17823 33175 69345 81508 55233 28207 62820 02340 26269 07898
34245 17120 06207 71464 09794 56116 12762 91459 51237 22991 33401 69552
36385 09428 85592 01872 74337 95173 01458 63575 70828 35578 01587 35432
76888 86801 20399 88238 47021 51467 60544 54076 63535 98417 44304 80128
93831 38968 81639 48746 96588 17504 50692 63653 38175 05547 81286 40000
00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

No: 22 stop
D:      0
E:    400

No: 21 ISOMER TABLE(r);
r :=       10
N:       1
PRI(N):
           1
```

SEC(N):
^

TER(N):
^

N: 2
PRI(N):
^ 1
SEC(N):
^

TER(N):
^

N: 3
PRI(N):
^ 1
SEC(N):
^ 1
TER(N):
^

N: 4
PRI(N):
^ 2
SEC(N):
^ 1
TER(N):
^ 1

N: 5
PRI(N):
^ 4
SEC(N):
^ 3
TER(N):
^ 1

N: 6
PRI(N):
^ 8

```
SEC(N):
 -        6
TER(N):
 -        3

N:        7
PRI(N):
 -       17
SEC(N):
 -       15
TER(N):
 -        7

N:        8
PRI(N):
 -       39
SEC(N):
 -       33
TER(N):
 -       17

N:        9
PRI(N):
 -       89
SEC(N):
 -       82
TER(N):
 -       40

N:       10
PRI(N):
 -      211
SEC(N):
 -      194
TER(N):
 -      102
```

```
No: 22 stop
D:       20
E:       30

No: 17 A := exp(PIxsqrt(r));
r :=        37

No:  2 write(A);
 1991 48647.99997 80464 15400 26939

No: 22 stop

Nu kan De forsøge:
Opgiv antal decimaler, D.  -1 er stop: 0,
Og antallet af heltalscifre, E: 2500,

No: 19 POWER TABLE(r, r, r, r/variable);r := 50, r := 50, r := 200, r := 4294967296,
a:  4294967296
N:     50,  a/N:
         44 46241 64770 94044 62001 68140 65517 36431 58192 34512 13783
93194 18223 09375 36830 69769 15223 89847 82576 17396 94174 85953 52114
10493 83745 10705 64552 83979 31638 50167 01612 81011 95625 85078 62041
59767 30705 69834 50870 39035 93076 12750 83827 26540 55960 65418 17365
26850 35788 89811 39916 27042 32924 68503 14029 87716 16224 87411 87777
95788 92097 02969 04615 32001 91531 13668 62468 94214 88922 05997 88382
82657 21290 29622 02492 02674 74066 98147 05818 56476 50099 60300 38964
18433 21936 00841 64737 75144 51192 92467 88246 55953 89709 57296 16062
63646 45376
N:    100,  a/N:
        1976 90647 89825 63993 65422 64398 37963 34031 53906 82625 77382
89182 65710 15834 06010 93951 12675 62958 48974 61306 30992 94244 70316
46284 28967 96805 75470 50608 90485 92346 00159 01422 93291 02195 10157
40810 57061 66194 81068 84800 32112 98186 93914 60884 52816 61462 33381
43265 44389 74116 40093 67602 54810 38827 24187 83158 73949 54463 18313
77356 57307 01963 73591 69290 83431 87004 53890 61789 27145 61362 37042
73883 84101 31601 01344 26924 66208 48884 61376 21848 96537 94242 99905
38911 51382 46588 84820 03300 08567 61101 73467 99700 34941 59830 09427
19475 06024 97427 19534 14706 03806 82101 70338 96166 32028 39203 64112
08652 63292 24871 86929 24915 18929 14552 00665 47960 69516 12257 86849
```

52991 67071 77130 68944 28954 78867 91499 00427 95482 33003 93640 00764
93977 42106 63557 38284 25752 73030 53752 32721 33980 38718 89299 28113
42082 11131 34100 11356 05446 80947 74099 79279 62721 31886 10112 86792
95697 89492 64046 57366 33925 06505 25409 62862 02773 63124 99143 90269
20337 55536 95204 61624 10311 39550 16195 68814 54777 72710 31259 24797
32508 66583 11685 36159 08352 88130 55872 97178 18314 53887 45781 29700
22381 81376

N: 150, a↑N:

87898 03920 47883 24925 78329 41518 08719 40228 28870 71042 52734
01098 18285 18947 93474 76280 73880 02944 33431 36789 40358 07990 75524
48098 18272 02376 49613 81305 02225 08223 57470 65305 39071 76395 03601
30867 94211 67834 79927 10891 47366 06395 64531 75283 67982 48860 53958
92789 34330 89339 09961 49386 56764 12870 56231 34045 97409 04172 06530
59718 45815 79938 90136 62005 94707 63154 85830 00233 90802 54823 93167
22420 49945 66321 49012 68959 40280 19551 67327 78301 46231 54365 05079
46020 04967 48692 49196 97268 93484 56492 27052 23807 15691 86822 35354
51031 18275 85541 11561 80259 55573 98551 74253 89690 03324 83877 41160
92059 96634 09113 41583 94255 10049 83348 27334 94620 28705 97933 35580
30933 79167 51082 22351 20696 99357 43974 68850 42998 64962 85306 64548
45781 47452 26792 36411 94990 10437 15831 04636 69025 72848 87683 04391
11706 30987 56130 17794 78101 72617 18222 56436 43500 92612 35234 60382
58279 35643 41016 34015 93197 12508 38902 61146 41905 22299 93008 30842
94155 02543 80045 34944 65091 58632 27042 06500 44169 45721 57206 80962
77492 01455 69155 01077 28035 19382 57841 89677 78964 30705 87902 79175
69306 55177 34120 63086 02793 96582 14631 51154 07683 21443 36308 13937
78247 24039 77712 61143 46215 76072 82394 73957 88355 91917 83337 10012
76726 49209 01920 77541 99004 30639 72559 44961 85666 36705 48703 62932
04095 60225 93238 44785 76976 21049 78897 55369 54984 55047 01102 00212
66667 70697 34789 85761 16823 50713 61785 82760 21223 85210 83159 82120
37375 31811 35338 50569 31835 08412 87614 46654 60603 08150 34742 82616
35164 93820 20252 21491 57210 94334 42115 49624 99197 42381 73452 62063
32505 03973 22604 66683 69517 25372 09643 43419 86330 86694 88566 51586
54077 17376

N: 200, a↑N:

39 08159 22664 32387 33174 61428 36148 36731 12676 81070 46341 22725
06674 72076 85355 68430 13838 17204 95886 91174 33070 78182 43450 11844
83028 12729 95124 73215 76513 16662 43694 26519 56617 55211 46060 76508
16799 76758 04172 06793 00544 14257 88999 99682 21814 55907 31711 21585
23758 61916 25968 42648 66953 94453 38537 80206 23210 96898 60956 55234
80067 32061 69578 99389 30646 21394 47900 78445 22654 75093 25302 60932

90694 45117 08573 96111 68420 51110 07278 07029 96021 97553 04683 34392
85228 35681 23658 05447 49345 77299 76877 89272 00578 05058 45692 81027
95984 74814 92880 15751 39205 78725 80485 13369 09371 09613 12514 68207
58995 25393 91748 61917 22044 01099 25365 54254 43334 67577 97401 45331
74411 57668 32347 51173 78300 30067 53803 11411 78372 08922 91860 20884
09354 27421 87687 84951 72143 64478 83790 83826 46195 52328 14452 67002
41066 34029 78244 43148 57725 94698 43406 62589 55213 76811 87058 85537
08406 78104 15837 31029 13142 93532 22481 66923 13560 64885 77076 17678
65710 72087 78727 57211 51671 44969 84455 47824 37658 11792 28842 82324
30765 79850 08269 94409 10879 69017 27016 54935 33858 54192 37394 52891
02194 66400 39871 13901 46945 17482 74843 82033 62034 91170 27739 85087
02854 98619 30279 63906 59011 03098 38995 71129 47553 19519 00699 94199
57287 15779 97339 20151 85546 94093 47042 46331 34190 56713 81265 13775
81786 77705 18591 30300 02603 34804 04803 78452 84233 49388 35344 89638
49864 72605 87064 32652 19489 00609 81900 67654 10832 11709 43392 35540
06135 64249 44564 56395 10323 75578 18092 89920 76476 79659 83067 04845
77942 56504 91079 89823 24928 63540 15437 42404 99247 06852 95256 71271
00057 13066 46256 94704 57811 35744 09288 14052 82687 14804 05082 64376
85344 13838 21755 80528 79567 40468 54193 37070 91945 41655 43913 92625
47975 35062 17540 32918 40282 43756 57845 10890 52788 42826 09693 78835
27288 45075 42718 46667 36270 90471 85999 67377 23112 93406 02704 54109
05956 60344 39294 50215 59935 52543 83198 87090 35361 71354 88670 20994
34928 49139 96584 68967 40313 62649 58871 05269 67617 55570 97165 01891
68531 48260 79439 17084 38199 22088 87812 89029 68582 95053 15773 90213
88539 90717 60414 28857 19601 69709 47204 05328 12974 51190 56694 81031
74745 13400 33033 35172 33611 19313 33799 54007 38438 43950 34058 79987
18732 53376

No: 22 stop
Opgiv antal decimaler, D. -1 er stop: -1
run