ADMINISTRATION AF DATA I GIER

Jørgen Kjær

Haldor Topsøe, Vedbæk, Danmark

1968

Copy no.

Haldor Topsøe, Chemical Engineers

Vedbæk, Denmark

FORORD

I den tidligere udsendte bog: Programmering af kemiske Beregninger, Bind 1, ALGOL, fandtes to kapitler om brug af tromlen og om specielle input-outputprocedurer i brug hos Haldor Topsøe. I den anden udgave af ALGOL-bogen (Elementært ALGOL, 1968) er disse to kapitler udeladt, og de præsenteres her i nærværende bog i stærkt ændret og udvidet form.

Bogen omtaler de metoder, som benyttes og har været benyttet hos Haldor Topsøe ved programmering i GIER ALGOL II, ALGOL III og ALGOL 4. Desuden gives en kort omtale af vore planer for et mere avanceret databehandlingssystem, det saakaldte GIPS-system, der idag i det væsentlige kun er paa planlægningsstadiet.

Man kan synes, at det maaske er lidt overflødigt i detailler at gøre rede for systemer, som ikke længere bruges ved fremstilling af nye programmer. Men da vi idag har programmer kørende i alle tre ALGOL-varianter (foruden i GIER maskinkode) er det vigtigt for at kunne vedligeholde, forbedre og omprogrammere disse programmer, at der findes en samlet, skriftlig redegørelse for de vigtigste af de metoder, som har været brugt.

Vedbæk, Februar 1968

Jørgen Kjær

INDHOLDSFORTEGNELSE

1. INDLEDNING

Naar en begynder i programmeringsfaget har lært at bruge de elementære begreber i ALGOL eller et andet lignende programmeringssprog, varer det som regel ikke længe, før han støder paa forskellige nye problemer, som kræver en mere avanceret programmeringsteknik. I en regnemaskine som GIER, hvor ferritlageret kun indeholder 1024 celler, bliver det hurtigt nødvendigt at bruge et saakaldt baggrundslager til lagring af de tal, som i et givet øjeblik ikke behøver at være tilstede i ferritlageret.

I kapitel 2 beskrives, hvorledes man i ALGOL programmerer datatransporter til og fra baggrundslageret, der for GIER-maskinerne hos Haldor Topsøe er en tromle og et pladelager. I samme kapitel beskrives den automatiske programsegmentering i GIER, som hele tiden sørger for overførsel af de nødvendige programdele fra baggrundslageret til ferritlageret.

I kapitel 3 omtales de fleste af de specielle procedurer, der gennem aarene er udviklet hos Haldor Topsøe, for at sikre at de fremstillede beregningsresultater faar et nogenlunde overskueligt og ensartet præg. Det drejer sig dels om procedurer til linieskift og sideskift og til forsyning af rapporterne med en forside, og dels om procedurer til indlæsning af datagrupper og trykning af disse. Opdelingen af inputmaterialet i datagrupper er et andet eksempel paa en mere avanceret programmeringsteknik, som er praktisk og nødvendig, saa snart datamaterialet vokser over en vis størrelse.

Kapitel 4 beskriver forskellige metoder til placering af konstante talsæt i et program. Det var især i ALGOL II og III, at der var behov for specielle procedurer hertil, hvorimod det i ALGOL 4 er let at udføre, fordi man kan oprette navngivne arealer paa pladelageret.

I kapitel 5 gives en kort beskrivelse af forskellige metoder til kontrol af konsistensen af tallene i en datagruppe, især for kemisk-tekniske beregninger.

I kapitel 6 omtales GIPS-systemet.

2. BRUG AF TROMLE OG DISK

2.1. Alment om Baggrundslagre.

I bogen om Elementært ALGOL blev forklaret, hvorledes ciffermaskinens lager (hukommelse) bruges til lagring baade af programmet (opbygget af ordrer) og af talmaterialet. Det lager, der her er tale om, er det saakaldte ferritlager (eller det hurtige lager), hvori hver enkelt celle direkte og med meget stor hastighed kan kommunicere med maskinens centrale registre, hvad enten cellernes indhold skal opfattes som ordrer eller tal.

Da ferritlageret i GIER kun indeholder 1024 celler, og disse skal bruges baade til lagring af program og af talmateriale, kan man let indse, at det kun er ganske smaa programmer, og saadanne med et ganske lille talmateriale, som kan rummes i ferritlageret. For større programmer og for programmer med et stort talmateriale maa der indføres forskellige mekanismer, som sørger for at hente og bringe programdele og talgrupper fra et hjælpelager, det saakaldte baggrundslager.

De vigtigste typer af baggrundslagre er:

1. Magnetbaand
2. Tromle
3. Pladelager (Disk).

Fælles for alle baggrundslagre er, at de indeholder mange celler og at prisen pr. celle er væsentlig lavere end for cellerne i ferritlageret. Den billigste type er magnetbaand, som er velegnet til visse kategorier af opgaver, men som har den ulempe, at det tager lang tid at køre fra den ene ende af baandet til den anden.

Magnetiske tromler og pladelagre er væsentlig dyrere end magnetbaand, men da de roterer med stor hastighed, kan man hurtigt faa fat i en vilkaarlig celle. Sker rotationen med netfrekvensen (50 omdrejninger pr. sek.), vil accesstiden være af størrelsesordenen 0.02 sek. (20 millisek.). For den magnettromle, som er forbundet med den gamle GIER-maskine hos Haldor Topsøe, tager det netop 20 millisek. at overføre en talgruppe paa 40 celler. I den nye GIER-maskine er baggrundslageret et pladelager (disk), hvor der kræves en mekanisk bevægelse af nogle arme, hvis man skal have fat i et

vilkaarligt sted paa disken. Overføringstiden bliver derfor større, nemlig 85 millisek. for indstilling af armene plus 3 millisek. for den egentlige overførsel af 40 celler. Ved overføring af flere grupper paa 40 celler, som kommer lige efter hinanden paa disken, koster det altsaa kun 3 millisek. mere at overføre 40 celler mere.

I GIER ALGOL (II, III, og 4) sker programtransporter fra baggrundslageret helt automatisk efter de principper, som skitseres i det følgende afsnit. Transporter af talmateriale maa derimod programmeres, som beskrevet i afsnit 2.3 og 2.4. Da GIERs grundoperation for transporter til og fra baggrundslageret overfører 40 celler ad gangen, er baade tromle og disk opdelt i saakaldte kanaler, hver indeholdende 40 celler. Kanalerne er nummereret saaledes:

	Kanal-numre	Samlet antal kanaler	Samlet antal celler
Tromle	0-319	320	12800
Disk	0-9599	9600	384000

De 9600 kanaler paa disken er i virkeligheden opdelt i 10 grupper, nummereret fra 0 til 9. Indenfor hver gruppe nummereres kanalerne fra 0 til 959.

2.2. Automatisk Programsegmentering.

Selv om overførslen af program fra baggrundslageret til ferritlageret sker automatisk i GIER ALGOL, uden at man behøver at skrive noget om det i programmet, er det praktisk at vide en lille smule om de anvendte mekanismer. Figur 1: Datatransporter i GIER (side 10) viser princippet i placering af program og talmateriale i ferritlageret og paa baggrundslageret.

Figuren illustrerer forholdene, naar et ALGOL-program er oversat og er ifærd med at udføre de egentlige beregninger. Hvad der sker under den forudgaaende oversættelse interesserer os ikke her; det eneste vi behøver at vide, er at oversættelsen afsluttes med at det oversatte program i sin helhed er lagret paa baggrundslageret. Paa figuren er dette vist som omraadet: G i den ene ende af tromlen eller disken. Dette svarer til forholdene i GIER ALGOL II og III, hvor det oversatte program altid afleveres i den høje ende af tromlen eller disken, altsaa enden med de højeste kanalnumre. I GIER ALGOL 4 vil det oversatte program blive afleveret paa baggrundslageret i et navngivet omraade:

{<work}

som man principielt kan have placeret hvorsomhelst. Hos Haldor Topsøe bruges for tiden ogsaa her de højeste kanalnumre.

Ferritlageret er disponeret saaledes:

Omraade A fra celle 0-15 indeholder et ganske lille program til udførelse af forskellige administrative funktioner. Tallet 15 kan variere noget i de forskellige versioner af ALGOL.

Omraade B er reserveret lagring af større eller mindre dele af det oversatte program.

Omraade C er forbeholdt lagring af tal, nemlig de variable, som i et givet øjeblik er deklareret i de blokke, som man da opholder sig i.

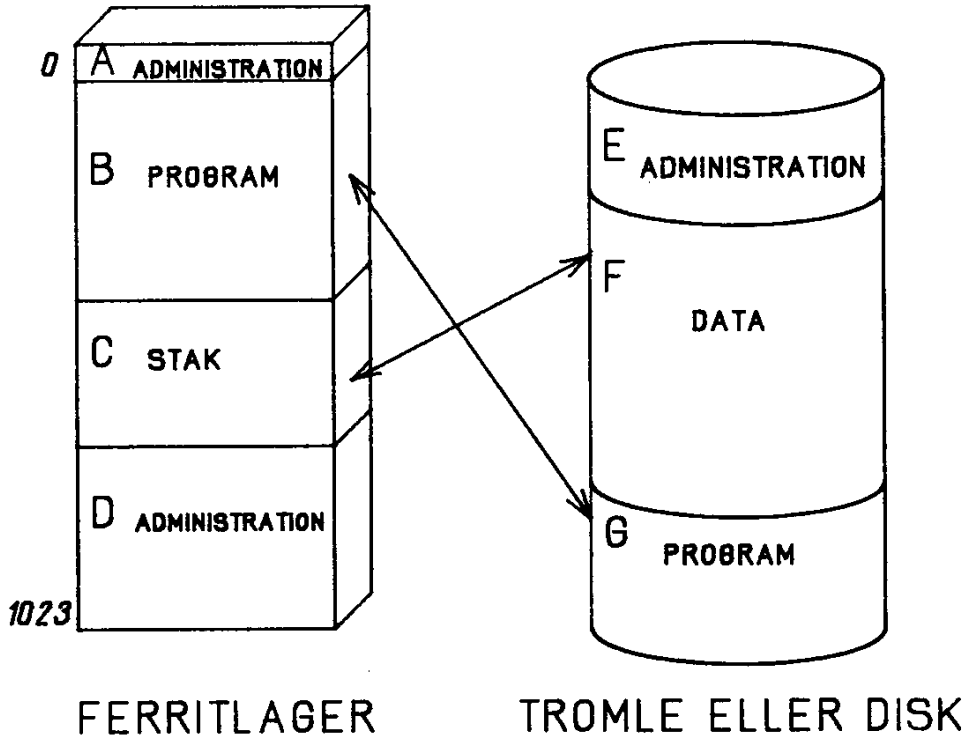
Omraade D fra celle ca. 800 til 1023 indeholder forskellige administrationsprocedurer.

Det er nu vigtigt at være klar over, at talomraadet C, den saakaldte stak, har en variabel størrelse. Hver gang man gaar ind i en blok, vokser stakken, fordi der skal være plads til de nye variable, som er deklareret i denne blok. Paa samme maade bliver stakken mindre, hver gang man forlader en blok. Den effektive størrelse af omraadet B maa derfor variere tilsvarende.

Den praktiske administration heraf er nu følgende. Programdelene paa baggrundslageret overføres i segmenter paa 40 celler ad gangen, idet 40 celler er den mindste gruppe celler, der kan overføres samlet. Naar maskinen begynder at regne, overføres det første segment til de første celler i omraade B, d.v.s. celle 16 - 55.

Der regnes nu paa programmet i cellerne 16 - 55. Naar maskinen kommer frem til den sidste af disse celler (55), staar her en oplysning om, at segmentet slutter her, samt nummeret paa den tromlekanal, hvori programmet fortsætter. Maskinen hopper nu til et lille program i D-omraadet hvori programsegmenteringen administreres. Det lille program har en liste over hvilke segmenter, der til enhver tid er tilstede i lageret, samt hvor de staar. I vort tilfælde er det nye segment ikke til stede, fordi vi lige er begyndt at regne, og administrationen vil derfor overføre dette segment fra tromlen til de næste ledige lagerceller, f.eks. til 57 - 96 (celle 56 er reserveret andet formaal). Derefter hoppes til det sted indenfor omraadet 57 - 96, hvor det rigtige program fortsætter. Det vil normalt ikke være celle 57, fordi den første del af segmentet bruges til at gemme de talkonstanter o.l., der skal bruges i dette segment.

Hvis maskinen kommer frem til celle 96, altsaa til slutningen af andet segment, gentages spøgen her med tredje segment, o.s.v.



DATATRANSPORTER I GIER

FIGUR 1

HALDOR TOPSØE	
	Cust. no 20851 DR.NO.: 449516

Hvis det lille administrationsprogram ved overgang fra eet segment til et andet opdager, at det nye segment allerede staar i lageret, sker der naturligvis ingen tromletransport, men hoppes blot til dette sted.

Hvis et nyt segment skal overføres til lageret, men al lagerplads er udnyttet til gamle segmenter, vil det nye segment blive anbragt i stedet for et af de gamle segmenter, som derfor ikke længere er tilstede. Maskinen forsyner de overførte segmenter med et voksende løbenummer, og det er segmentet med det laveste løbenummer, der bliver udskiftet.

Hver gang talmaterialet vokser, f.eks. ved at man gaar ind i en ny blok, holdes der fornødent regnskab med, hvilke segmenter i lageret, der herved bliver ødelagt. Paa samme maade bliver der mere programplads til raadighed, hver gang talmaterialet indskrænkes. Denne plads bliver udnyttet, men dog med en vis træghed, fordi der normalt vil være stor sandsynlighed for, at den gaar tabt igen et øjeblik efter.

Segmenteringsmetoden indeholder iøvrigt flere finesser, se herom Naur (1963, 1965b).

Det laveste antal lagersegmenter, som maskinen kan køre med er to i ALGOL II og III og fire i ALGOL 4. Her vil beregningerne gaa meget langsomt, hvis der hele tiden skal hentes nye segmenter frem. Talmaterialet kan i dette tilfælde brede sig fra celle 180 til ca. 800. Man vil heraf se, at der aldrig kan være mere end ca. 650 tal i lageret ad gangen. Programmet maa derfor være lavet saaledes, at der paa intet sted opereres med flere end ca. 650 simple variable og talsætelementer. I praksis bliver tallet noget mindre, fordi der ogsaa skal være plads til administrative oplysninger blandt talmaterialet, f.eks. blokreferencer, data for talsætgrænser, m.m.

I nogle tilfælde kan grænsen paa 650 tal overholdes, hvis man sørger for en fornuftig opdeling i blokke. Store talsæt, som kun bruges inden for et lille omraade af programmet, bør deklareres i en blok paa dette sted, f.eks.:

```
begin
   ........
   ........
   begin
      array A[1:100];
      ........
      ........
   end;
   ........
   begin
      array B[1:100];
      ........
      ........
   end;
   ........
end;
```

Men hvis de to talsæt skal bruges samtidigt, maa de naturligvis deklareres i samme blok, eller i blokke, der staar indeni hinanden.

Hvis maskinen under kørsel med et ALGOL-program kommer ind i en blok, hvor det samlede antal variable passerer den øvre grænse paa ca. 650, standser beregningen, og man faar udskriften:

stack

paa den tilkoblede skrivemaskine, som tegn paa at der ikke længere er plads til programmet.

Man kan ogsaa faa fejludskriften:

array

nemlig, hvis der deklareres et meget stort array i blokken. Samme udskrift faas, hvis antallet af elementer i arrayet er nul eller negativt.

Man maa da skrive programmet om, f.eks. saaledes at det faar flere sideordnede blokke. Hjælper dette ikke, maa man gaa over til at bruge tromlelagring af materialet som forklaret i næste afsnit.

Iøvrigt bemærkes det, at man paa grund af den automatiske segmentering undertiden kan være ude for det uheld, at et ganske lille stykke af programmet, der gennemløbes mange gange, f.eks. en for-sætning til summation,

er blevet placeret paa to forskellige segmenter, altsaa begynder i slutningen af det ene segment og slutter i begyndelsen af det næste. Der tabes herved nogen tid i segmenteringsadministrationen, som kunne være undgaaet, hvis for-sætningen havde ligget helt indenfor det ene segment. Har man mistanke om, at dette uheld er sket, kan man afhjælpe det ved at indføre en eller flere harmløse sætninger som f.eks.:

```
x := x;
```

og derved faa skubbet for-sætningen ind i et enkelt segment.

I GIER ALGOL 4 er der indført to særlige tricks, som tillader programmøren at faa lidt kontrol med segmenteringen. Disse tricks findes ikke i ALGOL II og III.

Det første trick illustreres af følgende eksempel:

```
for i := 1 step 1 until N do
begin
   sum := sum + A[i]↑2
end for;
```

Her er der anbragt et understreget for som kommentar mellem end og semikolonnet. Oversætteren opfatter denne kommentar paa en særlig maade, idet den sørger for, at den oversatte for-sætning bliver anbragt paa netop eet segment. Herved kan man undgaa overgang fra eet segment til et andet midt i en for-sætning, som maaske skal gennemløbes mange gange. Herved spares regnetid. Metoden kan kun bruges ved korte for-sætninger af simpel type, se Naur (1967), side 45. Man bør ikke overdrive brugen af dette trick, da programmets længde forøges paa grund af ikke helt opfyldte segmenter.

Det andet trick i ALGOL 4 udnytter en standardvariabel:

```
integer tracks transferred;
```

Denne variabel er automatisk deklareret i ethvert program og er nulstillet, naar programmet begynder at regne. Den øges med een, hver gang et programsegment overføres fra tromlen eller disken. Hvis man forsyner programmet med udskriften af talværdien af denne variable paa passende steder, kan man faa oplysninger om, hvor i programmet segmenteringsmekanismen er mest kritisk for regnetiden.

I forbindelse med segmenteringen maa der ogsaa gøres opmærksom paa følgende:

Da hvert programsegment indledes med en lille liste over de talkonstanter, der skal bruges i dette segment, kan det være meget pladskrævende, hvis et program indeholder mange ens talkonstanter. Har vi f.eks. programmet:

```
begin
   real A, B, D1, D2, Q;
   ........
   A := 3.14159×B;
   ........
   Q := 3.14159×(D1 + D2);
   ........
end;
```

og er talkonstanten: 3.14159 brugt mange steder i programmet, og fordelt over et saa stort omraade, at man maa formode, at det kommer til at fylde flere segmenter, da kan det ofte betale sig at deklarere konstanten som en særlig variabel:

```
begin
   real A, B, D1, D2, Q, pi;
   ........
   pi := 3.14159;
   ........
   A := pi×B;
   ........
   Q := pi×(D1 + D2);
   ........
end;
```

Ogsaa smaa talkonstanter som 2, 3, 4, o.s.v. kan deklareres saaledes. Tallet 1 behøver dog ikke deklareres, da det er fast indbygget i omraade D.

Til slut bemærker vi, at den automatiske programsegmentering kun transporterer segmenter fra baggrundslageret til ferritlageret, ikke omvendt. Dette betyder, at programsegmenternes indhold maa være rent statisk. Ændringer af ordrerne er ikke tilladt. Herved har man afskaaret sig fra at udnytte forskellige former for automatisk modifikation af ordrerne med deraf følgende langsommere regnetid.

2.3. Datatransporter i ALGOL II og III.

Vi har set, at maskinen højst kan operere med ca. 650 tal samtidigt i lageret og helst lidt mindre, hvis det ikke skal gaa for langsomt. Hvis vi derfor skal løse følgende opgave:

Læs 1000 tal fra en strimmel, find deres sum, og gem dem i maskinen til senere anvendelse, f.eks. trykning, da maa vi aabenbart tage baggrundslageret til hjælp.

Hvis vi betragter figur 1, drejer det sig altsaa om transporter mellem omraadet C i ferritlageret (stakken) og omraadet F paa tromlen eller disken. Vi har her brug for to ting:

1. ALGOL-procedurer til den egentlige transport.
2. En metode til nummerering eller navngivning af positioner paa baggrundslageret.

I GIER ALGOL II og III er dette løst paa en relativ simpel maade, medens metoden i GIER ALGOL 4 er mere avanceret. Den sidste beskrives i næste afsnit. Metoden i ALGOL II og III adskiller sig kun ved navnene paa begreberne (dansk i ALGOL II og engelsk i ALGOL III). Der bruges to standardprocedurer til transporterne og een standardvariabel til fixering af positioner paa tromlen eller disken:

	ALGOL II	ALGOL III
Transporter:	til tromle (A)	to drum (A)
	fra tromle (A)	from drum (A)
Standardvariabel:	tromleplads	drumplace

Da virkemaaden er fuldstændig ens i ALGOL II og III kan vi nøjes med at illustrere forholdene i ALGOL III. Den ovennævnte opgave til læsning af 1000 tal kan da løses paa følgende maade:

```
begin
   integer i, j, top;
   real SUM;
   array A[1:100];
   top := drumplace;
   SUM := 0;
   for i := 1 step 1 until 10 do
   begin
      input (A);
      to drum (A);
```

```
        for j := 1 step 1 until 100 do
        SUM := SUM + A[j]
    end for i;
    drumplace := top;
    for i := 1 step 1 until 10 do
    begin
        outcr;
        from drum (A);
        for j := 1 step 1 until 100 do
        begin
            output(‹-ndd.dd›, A[j]);
            if j:10×10 = j then outcr
        end for j
    end for i;
    outcr;
    output(‹ndddd.dd›, outtext(‹SUM:›), SUM)
end program;
```

Bemærk de specielle procedurer til indlæsning og trykning, som er benyttet her i ALGOL III. De er meget nær identiske med dem i ALGOL 4 med undtagelse af input (A), der læser alle de 100 tal, som er deklareret i A. Man kan studere nærmere om ALGOL III i Naur (1965a).

Programmet virker paa følgende maade. Lad os antage, at vi arbejder med en tromlemaskine, hvor antallet af celler paa tromlen er 12800. For nemheds skyld kan vi tænke os, at cellerne er nummereret fra 0 til 12799. Vi har set, at det oversatte ALGOL-program gemmes i den øverste ende af tromlen (omraade G paa figur 1). Hvis programmet fylder f.eks. 1600 celler, vil det optage cellerne 11200-12799. Den øverste ledige tromlecelle i dataomraadet F har derfor nummer 11199.

Naar et oversat ALGOL III program er klar til at starte, har maskinen sørget for, at talværdien af standardvariablen:

```
integer drumplace;
```

er sat lig med nummeret paa den første ledige tromlecelle regnet ovenfra, altsaa her 11199. I programmet ovenfor har vi sætningen:

```
top := drumplace;
```

og værdien af top bliver derfor ogsaa 11199. Derefter følger den første

for-sætning med i:

```
for i := 1 step 1 until 10 do
```

som hver af de 10 gange gør følgende: Med input (A) faar vi indlæst 100 tal fra inputstrimlen og gemt i talsættet A. Sætningen:

```
to drum (A);
```

overfører talsættet A til tromlen, saaledes at det højeste element anbringes i den tromlecelle, hvis nummer er lig med drumplace, det næsthøjeste element i cellen før, o.s.v. For i = 1 faar vi altsaa følgende overføring:

```
A[100] til tromlecelle 11199
A[99]  -       -       11198
A[98]  -       -       11197

o.s.v.

A[1]   -       -       11100
```

Naar transporten er slut, bliver talværdien af drumplace reguleret ned, saaledes at den giver nummeret paa den næste frie tromlecelle, altsaa her: 11099. Værdien af drumplace er altsaa blevet formindsket med antallet af elementer i talsættet. Dette betyder, at for i = 2 vil næste overføring til tromlen ske til cellerne 11000 - 11099, o.s.v. indtil vi har faaet overført alle 1000 tal til tromlecellerne 10200 - 11199.

Summationen af elementerne sker med sætningen:

```
for j := 1 step 1 until 100 do SUM := SUM + A[j];
```

Ønsker vi at underkaste de 1000 tal paa tromlen en nærmere behandling, her illustreret ved at vi trykker tallene igen, da kan det gøres som vist i sidste halvdel af programmet. Først skal vi ved hjælp af sætningen:

```
drumplace := top;
```

genindsætte startværdien af drumplace, altsaa her 11199. Derefter kommer igen en i-for-sætning, som hver gang udfører sætningen:

```
from drum (A);
```

hvorved den tromlecelle, hvis nummer er lig med drumplace, overføres til det højeste A-element, altsaa A[100], den næste lavere tromlecelle gaar til A[99], o.s.v. Efter transporten er værdien af drumplace ogsaa her formindsket med antallet af elementer, altsaa her 100.

Programmet trykker iøvrigt 10 elementer per linie og værdien af summen tilsidst.

Ved hjælp af de to procedurer, to drum og from drum, kan vi faa overført talsæt til og fra tromlen paa den her skitserede maade. Maskinen sørger selv for, at værdien af drumplace er den højst mulige ved programmets start. Hvis programmøren husker at gemme denne startværdi i en passende variabel (her: top), kan man altid styre programmet tilbage til det sted paa tromlen, hvor lagringen begyndte.

Hvad enten man overfører til eller fra tromlen, vil værdien af drumplace blive reduceret med antallet af overførte elementer efter transporten.

Den lavest tilladelige værdi af drumplace for en tromlemaskine afhænger af, om vi bruger ALGOL II eller III, idet administrationsomraadet E i figur 1 er lidt større i ALGOL III end i ALGOL II:

	ALGOL II	ALGOL III
Brugte kanalnumre i E:	0-65	0-79
Laveste frie kanal i F:	66	80
Laveste værdi af drumplace:	2640	3200

Hvis værdien af drumplace kommer under den laveste værdi, eller fejlagtigt bliver sat til en værdi højere end startværdien, standser maskinen, og man faar fejludskriften:

```
drum alas
```

i ALGOL III eller tromle ak i ALGOL II.

For en disk-maskine kørende i ALGOL III er figur 1 ikke helt korrekt, idet omraade E og G er slaaet sammen i det lave omraade og dataomraadet F udgør det øverste omraade:

Omraade	Funktion	Gruppe	drumplace
E+G	Adm. + program	0	(negativ)
F	Data	1-9	0-345599

Vi bemærker, at disken er inddelt i 10 grupper hver med 960 kanaler eller 38400 celler. Værdien af drumplace er her afpasset saaledes, at laveste celle i gruppe 1 har nummer nul. Ogsaa her faar man fejludskriften drum alas, hvis drumplace bliver negativ eller større end 345599.

I eksemplet ovenfor skete transporten til og fra lageret til det samme talsæt: A. Dette behøver ikke at være tilfældet. Den normale brug af tromlen vil være, at man overfører et talsæt til tromlen i en af de første blokke, for senere at hente det frem igen som et andet talsæt i en anden blok. Som f.eks.:

```
begin
   ........
   ........
   begin comment Blok 1;
      array A[1:100];
      input (A);
      drumplace := a;
      to drum (A);
      ........
   end;
   ........
   .......
   begin comment Blok 2;
      array C[1:100];
      drumplace := a;
      from drum (C);
      ........
   end;
   ........
   begin comment Blok 3;
      array D[1:20], E[1:80];
      drumplace := a;
      from drum (E);
      from drum (D);
      ........
   end
end program;
```

I Blok 1 indlæses talsæt A[1:100] og gemmes paa tromlen. I Blok 2

kaldes det frem igen som talsæt C[1:100]. I Blok 3 har vi endelig vist, at man ikke behøver at overholde de oprindelige dimensioner af talsættet, idet det oprindelige A-talsæt nu er fordelt paa to: D[1:20] og E[1:80]. Men her skal man naturligvis passe paa, at der ikke kommer kludder i indiceringen. Sammenhængen er her:

```
D[1]  = A[1]
D[2]  = A[2]
        .
        .
        .
D[20] = A[20]
E[1]  = A[21]
E[2]  = A[22]
        .
        .
        .
E[80] = A[100]
```

Vi ser, at der i de tre blokke er brugt ialt 300 celler til talsætelementer i lageret, men at vi aldrig har haft mere end 100 fremme ad gangen.

Simple variable kan ikke uden videre overføres ved hjælp af tromleprocedurerne. Den parameter, der bruges i to drum og from drum, skal altid være navnet paa et talsæt. Men man kan jo blot indføre et talsæt paa et enkelt element. Har vi saaledes en Blok 4 i eksemplet ovenfor, hvori en real variabel: P skal sættes lig med A[17], kan vi skrive:

```
.....
begin comment Blok 4;
   real P;
   array F[1:1];
   drumplace := a - 83;
   from drum (F);
   P := F[1];
   .....
   .....
end;
```

De to procedurer to drum og from drum er i øvrigt af formen integer procedure. Deres talværdi er lig med minus det overførte antal elementer. Hvis vi derfor skriver:

```
drumplace := a1;
a2 := a1 + to drum(A);
```

faar vi A overført til tromlen, og a2 bliver lig med værdien af drumplace efter transporten.

Flerdimensionale talsæt behandles af tromleprocedurerne paa samme maade som af input-proceduren, hvor tællingen begynder i det sidste index. Efter udførelsen af programmet:

```
begin
   array A[1:2, 1:3];
   ........
   drumplace := 7006;
   to drum(A);
end;
```

vil A-elementerne staa saaledes paa tromlen:

Tromlecelle	Element
7001	A[1,1]
7002	A[1,2]
7003	A[1,3]
7004	A[2,1]
7005	A[2,2]
7006	A[2,3]

Det bemærkes, at proceduren from drum, der overfører talsæt fra tromlen til lageret, ikke ødelægger talsættet paa tromlen. Ethvert talsæt, der er overført til tromlen med proceduren to drum, forbliver intakt, indtil der indsættes nye værdier med to drum igen. Dette er ganske analogt med forholdene i lageret, hvor enhver hente-ordre ikke ændrer indholdet af den celle, hvorfra der hentes.

2.4. Datatransporter i ALGOL 4.

I ALGOL 4 har man forladt den simple pladsbestemmelse paa baggrundslageret ved hjælp af et heltal: drumplace. Man kan her operere med forskellige omraader paa baggrundslageret, hver karakteriseret ved et navn. I det følgende omtales først administrationen af disse arealer og derefter transportprocedurerne.

2.4.1. Arealadministration. Vi beskriver først de helt elementære træk ved arealadministrationen i ALGOL 4 og derefter de mere specielle forhold, som kun i mindre grad har interesse for brugeren.

Naar et program i ALGOL 4 er oversat og man skal begynde at regne paa det, vil det frie dataomraade F paa figur 1 være til raadighed for brugeren til lagring af data, som skal benyttes under beregningen, men som ikke skal gemmes til brug for senere beregninger paa dette program eller andre programmer. Det frie dataomraade er karakteriseret ved et navn, som vi noterer som en tekststreng:

```
{<free}
```

Naar vi skal kommunicere med dette omraade, har vi brug for nogle oplysninger om, hvor paa disken dette areal er placeret. Disse oplysninger faas ved at foretage et kald af standardproceduren: where, f.eks. saaledes:

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

Her er den første parameter en tekststreng, som er navnet paa det omraade, hvis placering paa disken vi søger oplysninger om. Den anden parameter er af typen integer og vil efter kaldet indeholde de ønskede oplysninger. Paa hvilken form disse oplysninger findes, er det strengt taget ikke nødvendigt for os at vide noget om, vi skal blot bruge denne variable (her: FREE) ved senere kald af transportprocedurerne: put og get. For fuldstændighedens skyld kan vi godt afsløre her, at arealordet FREE er opbygget af flere heltal, hvoraf første kanalnummer er placeret i bits 28-39 og det samlede antal kanaler i bits 8-23.

Proceduren where er selv af typen integer, og ved at undersøge dens talværdi efter kaldet, kan man faa visse oplysninger om, hvorvidt ind-

hentningen af informationen er lykkedes. Som eksempel kan vi tage programmet:

```
begin
   integer Q, AREA;
   Q := where(‹NAVN›, AREA);
   ....
end;
```

Q kan her antage følgende værdier:

0 Operationen er lykkedes, AREA indeholder de ønskede oplysninger om arealet med navnet: ‹NAVN›.
1 ‹NAVN› findes ikke i arealkataloget. AREA er uændret.
2 ‹NAVN› findes i kataloget, men beskriver ikke et disk-areal. AREA er ændret.
3 Kataloget er ødelagt. AREA er uændret.

Hvis vi kalder where med arealnavnet ‹free›, vil talværdierne 1, 2 og 3 ikke kunne forekomme, med mindre større eller mindre dele af systemet er ødelagt.

Normalt har man kun brug for med procedurekaldet:

```
where(‹free›, FREE);
```

at faa de ønskede oplysninger om placeringen af det frie omraade: ‹free›. Men i specielle tilfælde kan det være af interesse at faa fat i andre arealer, hvis de findes, eller muligvis selv oprette nye arealer (eller nedlægge disse).

Vore termodynamiske data er i øjeblikket lagret paa disken i et areal ved navn: ‹TDATA›. Skal man have fat i data fra dette areal, maa man paa een eller anden maade spørge:

```
where(‹TDATA›, AREA);
```

enten direkte ved at programmere denne sætning eller indirekte ved at kalde en standardprocedure som FTLIST, der internt spørger om, hvor ‹TDATA› staar (se side 102).

Oprettelse af nye omraader sker med proceduren: reserve, og de nedlægges igen med proceduren: cancel. Et eksempel er:

```
begin
   integer AREA;
   ........
   ........
   reserve(‹NYT›, 50);
   ........
   where(‹NYT›, AREA);
   ........
   ........
   cancel(‹NYT›);
   ........
end;
```

Med procedurekaldet: reserve(‹NYT›, 50) faar vi oprettet et areal med navnet: NYT, og indeholdende 50 kanaler, d.v.s. 2000 celler. Naar vi derefter skal transportere til og fra dette areal, maa vi først spørge om, hvor det staar ved et kald af: where. Endelig nedlægges arealet ved kaldet: cancel(‹NYT›).

De to procedurer reserve og cancel er ogsaa af typen integer, ligesom where. Indretter man procedurekaldene saaledes:

```
Q := reserve(‹NYT›, 50);
Q := cancel(‹NYT›);
```

kan Q antage følgende værdier for reserve:

Q	
0	Reservationen er i orden.
1	‹NYT› er et ikke-tilladt navn.
2	‹NYT› findes allerede i kataloget.
3	Kataloget er fuldt.
4	Antallet af kanaler (her 50) er for stort eller ≤ 0.
5	Kataloget er ødelagt.

og for cancel:

Q	
0	Nedlægningen er i orden.
1	‹NYT› findes ikke i kataloget.
2	‹NYT› beskriver et omraade, som ikke maa nedlægges.
3	Kataloget er ødelagt.

Vi ser heraf, at der er visse arealnavne, som er forbeholdt bestemte arealer eller muligvis helt andre ting, og som man derfor ikke kan reservere eller nedlægge. Følgende arealnavne er fast indbygget i kataloget og har en ganske bestemt betydning:

Navn:	Betydning:
free	Det frie areal paa baggrundslageret.
work	Et særligt arbejdsareal, hvor f.eks. oversætteren afleverer det oversatte program.
date	Et ganske lille omraade paa nogle faa celler i selve kataloget, som indeholder datoen.
image	Det saakaldte ferritlagerbillede, d.v.s. en kopi af ferritlageret overført til 26 kanaler paa disken.

Andre standardnavne i kataloget beskriver ikke arealer men ydre enheder:

Navn:	Betydning:
r	Kodebaandslæseren.
t	Skrivemaskinen som inputmedium.
p	Perforatoren.
l	Linieskriveren.
w	Skrivemaskinen som outputmedium.
x	En særlig notation for outputmediet, brugt af visse administrationsprogrammer.

Iøvrigt henvises til Lauesen(1967) og Naur(1967). I den første af disse bøger beskrives det ogsaa, hvorledes man kan bruge navngivne omraader af disken til at lagre ALGOL-programmer i ikke-oversat form, foretage rettelser i dem, oversætte dem, o.s.v. paa en særdeles praktisk maade.

Muligheden for at oprette navngivne omraader paa disken bør udnyttes, hvor det er praktisk, men ikke overdrives. De nyoprettede arealer tages fra {<free}, som derved bliver mindre og mindre. Ved nedlæggelsen vokser {<free} igen, men kun hvis det nedlagte areal i forvejen støder op til {<free}. Man har dog mulighed for at køre et særligt hjælpeprogram, som komprimerer de tiloversblevne arealer.

2.4.2. Transportprocedurer. I GIER ALGOL 4 hedder de to transportprocedurer:

```
put(A, AREA, TR);
```

og

```
get(A, AREA, TR);
```

De formelle parametre har typerne:

```
integer AREA, TR;
array A;
```

Parametren AREA er det arealord, som man skal have bestemt tidligere ved et kald af where for det paagældende areal. Parametren TR er et heltal, som angiver et kanalnummer indenfor arealet. Hvis arealet er oprettet med f.eks. 50 kanaler, skal TR ligge i omraadet:

$$1 \leq TR \leq 50$$

Endelig er parametren A det array, som skal overføres til eller fra disken. Her er der den yderst vigtige forskel fra ALGOL II og III, at A skal indeholde mindst 40 elementer. Mindre arrays kan ikke overføres med put og get. Der bruges altid et helt antal kanaler til lagring af A. Hvis A ikke er et multiplum af 40, gaar der altsaa nogle celler til spilde ved lagringen. Til lagring af A[1:100] medgaar altsaa tre fulde kanaler.

Programmet side 15 til læsning af 1000 tal kan nu skrives saaledes i ALGOL 4.

```
begin
   integer i, j, FREE;
   real SUM;
   array A[1:100];
   where(‹‹free›, FREE);
   select(8);
   SUM := 0;
   for i := 1 step 1 until 10 do
```

```
    begin
       for j := 1 step 1 until 100 do
       begin
          A[j] := read real;
          SUM := SUM + A[j];
       end for j;
       put(A, FREE, 3×i-2);
    end for i;
    for i := 1 step 1 until 10 do
    begin
       writecr;
       get(A, FREE, 3×i-2);
       for j := 1 step 1 until 100 do
       begin
          write(‹-ddd.dd›, A[j]);
          if j mod 10 = 0 then writecr;
       end for j
    end for i;
    writecr;
    writetext(‹<SUM:›);
    write(‹-ddddd.dd›, SUM);
    writecr;
end program;
```

Programmet begynder med at spørge, hvor det frie areal er:

```
where(‹<free›, FREE);
```

Heltallet FREE indeholder nu den nødvendige information om det frie areals placering. Derefter vælges linieskriveroutput og strimmellæserinput med select (8). SUM-cellen nulstilles, og vi begynder for-sætningen, hvor i tælles op fra 1 til 10. For hver værdi af i begyndes en indre for-sætning, hvor j tælles op til 100. For hver værdi af j indlæses det næste A-element, som derefter summeres. Endelig transporteres hele A-arrayet til disken med:

```
put(A, FREE, 3×i-2);
```

Den sidste parameter i dette kald vil antage værdierne 1, 4, 7, o.s.v.,

saaledes at de første 100 tal bliver anbragt paa kanal 1, 2 og 3, de næste 100 tal paa kanal 4, 5 og 6, o.s.v.

I sidste halvdel af programmet sker trykningen af tallene ved hjælp af ganske analoge for-sætninger med i og j, blot bruges her proceduren: get til tilbagetransporten af A-arrayene.

I dette eksempel har vi overført et talsæt med 100 celler til tre kanaler med ialt 120 celler. Oversætterfabrikanten kan ikke garantere, at de 20 ubrugte celler paa de tre kanaler altid vil være placeret paa nøjagtigt samme sted, ogsaa i eventuelle senere udgaver af oversætteren, saa man gør klogt i ikke at disponere over disse celler til andet formaal. Det sikreste er, altid at operere med talsæt, hvis størrelse er et multiplum af 40, naar disse skal administreres af put og get. Saa bliver kanalerne helt udnyttet og man kender de enkelte tals nøjagtige placering. Det sidste er vigtigt, hvis man senere ønsker at hente dele af talsættet frem.

Ved forkert anvendelse af put og get faar man fejludskriften:

```
error 11
```

og beregningen kan ikke fortsættes. Dette sker, hvis talsættet har mindre end 40 elementer, eller man kommer udenfor det opgivne omraade.

3. INPUT-OUTPUTADMINISTRATION

3.1. Linie- og Sideskift.

3.1.1. Proceduren LINE. Naar store mængder beregningsresultater skal udskrives automatisk paa linieskriveren (eller via en hulstrimmel paa en Flexowriter), er det praktisk, hvis der er en automatisk kontrol med, at det udskrevne passer med sideinddelingen i de lange papirbaner, der bruges ved udskriften. Man bør derfor forsyne sine programmer med procedurer, der sørger for en automatisk tælling af linierne paa hver side og skift til ny side, hver gang det er nødvendigt.

Vi bruger normalt proceduren:

```
LINE(n);
```

som trykker n gange CAR RET, hvis der er plads til det paa siden, ellers laver den saa mange, der er plads til, og sideskift. Det sidste bestaar af nogle frie linier, samt den øverste linie paa den følgende side, og som kan se saaledes ud:

-7- 6789

Her er 7-tallet sidens nummer, og 6789 er beregningsnummeret, der er opgivet i inputmaterialet.

LINE-proceduren maa derfor have adgang til følgende variable:

LRest: Det aktuelle antal linier paa siden, som endnu er til raadighed for trykning.
calcno: Beregningsnummer.
pageno: Sidenummer.

Man kunne have lavet proceduren med disse formelle parametre:

```
LINE(n, LRest, calcno, pageno);
```

Men det er alt for klodset med saa mange parametre, der skal bruges mange gange, og hvor det kun er værdien af n, som ændres i de forskellige kald af LINE.

Vi har derfor vedtaget, at disse tre parametre skal være ikke-lokale variable, som deklareres i den yderste blok.

I det følgende beskriver vi LINE-procedurens funktioner gældende for ALGOL 4. Til slut omtales kort de smaa forskelle, som findes i ALGOL II og III. Selve navnet LINE er det samme i alle tre tilfælde.

I ALGOL 4 kræves en fjerde global variabel, INF, af typen boolean, som ogsaa bør staa i den yderste blok. INF bruges til at gemme de tre heltal:

lbot: Antallet af frie linier fra nederste trykte linie paa siden til sidenummerlinien øverst næste side.

lmax: Antallet af linier, som kan bruges til nyttig udskrift paa en side.

cmax: Det maximale antal bogstaver pr. linie.

De fleste resultatudskrifter sker paa papir i A4-format, for hvilket der gælder:

lbot	lmax	cmax
7	65	72

Summen af lbot og lmax skal være lig det samlede antal fysiske linier pr. side, som her altsaa er 72.

Vi skal nu se, hvorledes de tre tal kan pakkes sammen i en celle i ALGOL 4. En celle indeholder 40 bits i GIER, og vi deler den i 4 lige dele med 10 bits i hver:

bits	0-9	10-19	20-29	30-39
Tal:	ledig	lbot	lmax	cmax

Da 10 bits dækker talomraadet fra 0 til 1023, kan vi pakke de tre tal i en celle ved at skrive:

```
INF := (lbot×1024+lmax)×1024+cmax;
```

Men nu var INF af typen boolean, saa vi maa skrive:

```
INF := boolean((lbot×1024+lmax)×1024+cmax);
```

for at snyde typekontrollen.

Naar vi skal ekstrahere eet af de tre tal fra INF kan dette gøres ved passende divisioner med 1024, men der findes en hurtigere maade at gøre det paa i ALGOL 4, hvor man direkte udnytter nogle af maskinens grundoperationer, nemlig de saakaldte skiftoperationer. Dette noteres i GIER ALGOL 4 med operationen shift. Lad os antage, at vi har et program med den logiske variable, INF, og en anden variabel, B, ligeledes af typen boolean. Hvis vi udfører sætningen:

```
B := INF shift 10;
```

vil B faa et indhold svarende til INFs, men hvor alle bits er rykket 10 pladser til venstre. De bits, som derved rykker ud af ordets venstre ende, puttes ind igen i den højre ende (selve INF-cellen er uændret):

bits	0-9	10-19	20-29	30-39
INF	0	lbot	lmax	cmax
B	lbot	lmax	cmax	0

Hvis skiftantallet er negativt, skiftes til højre. Saaledes giver:

```
B := INF shift -10;
```

følgende resultat:

bits	0-9	10-19	20-29	30-39
INF	0	lbot	lmax	cmax
B	cmax	0	lbot	lmax

Ønsker vi nu at assigne det lmax, der staar i bits 20-29 af INF til en heltalsvariabel af navn lmax, kunne man skrive saaledes:

```
lmax := (integer(INFshift-10))mod 1024;
```

hvorved lmax altsaa først flyttes hen til bits 30-39, men dette er ikke den hurtigste maade, idet vi jo skal udføre en division. Man bør her bruge en ren logisk operation, som er langt hurtigere. Lad os tænke os, at vi har endnu en variabel, MASK, af typen boolean og som har følgende indhold:

bits	0-9	10-19	20-29	30-39
MASK	0	0	0	1023

De sidste 10 bits indeholder altsaa tallet 1023, der i totalsystemet skrives saaledes:

1111111111

altsaa 10 1-taller. GIER ALGOL 4 er nu saaledes indrettet, at hvis vi skriver:

INF∧MASK

faar vi det logiske produkt, ikke blot af bit 0 i de to variable, men af alle 40 bits, d.v.s. vi faar 1-taller, hvor INF og MASK begge har 1-taller, ellers nul. Resultatet af INF∧MASK bliver derfor lig med cmax, opfattet som boolean. Ved at skifte INF før den logiske multiplikation, kan de andre tal ogsaa ekstraheres:

bits:	0-9	10-19	20-29	30-39
INF:	0	lbot	lmax	cmax
MASK:	0	0	0	1023
INF∧MASK:	0	0	0	cmax
(INFshift-10)∧MASK:	0	0	0	lmax
(INFshift-20)∧MASK:	0	0	0	lbot

Vi mangler nu blot at konstruere ordet MASK. Dette kan gøres saaledes:

```
MASK := boolean 1023;
```

Men der er en lidt fiksere maade at skrive det paa i ALGOL 4:

```
MASK := 40 1023;
```

idet det understregede tal 40, efterfulgt af et heltal, betyder 40 bits, i hvilke heltallet er anbragt længst til højre. De nærmere regler for opbygningen af bitmønstre paa denne maade findes i Naur (1967).

Det endelige udtryk for ekstraktion af f.eks. lmax bliver da:

```
lmax := integer((INFshift-10)∧40 1023);
```

Deklarationen af LINE i ALGOL 4 ser saaledes ud:

```
procedure LINE(n);
value n;
integer n;
begin
   if n > LRest then
   begin
      for n := LRest + integer((INF shift -20) ∧ 401023) step -1 until 1 do
      writecr;
      LRest := integer((INF shift -10) ∧ 401023);
      for n := (integer(INF ∧ 40 1023)-4):2 step -1 until 1 do
      writechar(0);
      writechar(58);
      writeinteger(⟨-ddd⟩, - pageno);
      writechar(32);
      pageno := pageno + 1;
      for n := (integer(INF ∧ 40 1023)-4):2 - 6 step -1 until 1 do
      writechar(0);
      writeinteger(⟨dddddd⟩, calcno);
      n := 3
   end;
   LRest := LRest - n;
   for n := n step -1 until 1 do writecr
end LINE;
```

Den formelle parameter, n, er det antal linier, som skal trykkes. Dette vil oftest være 1 eller 2.

LINE virker saaledes. Først undersøges, om n > LRest. Er dette tilfældet, skal der udføres et sideskift, der bestaar af følgende elementer:

1. Tryk LRest + lbot gange writecr.
2. Sæt LRest lig med lmax.
3. Tryk (cmax-4):2 mellemrum.
4. Tryk sidetallet med minus foran og bagefter.
5. Øg sidetallet med 1.
6. Tryk (cmax-4):2-6 mellemrum.
7. Tryk beregningsnummeret.
8. Sæt n lig med 3 (n er kaldt ved value).

Herefter fortsættes paa samme maade, som hvis der ikke skal skiftes side, nemlig med følgende:

9. Træk n fra LRest.
10. Tryk n gange writecr.

Da sideskiftet jo kun udføres een gang pr. side, vil langt de fleste kald af LINE kun give anledning til punkt 9 og 10.

De fire globale variable, LRest, calcno, pageno og INF, maa naturligvis være tildelt rimelige værdier før proceduren LINE kaldes første gang. Dette kan enten gøres direkte, eller indirekte via et kald af den generelle inputprocedure, INPUT1, der omtales i afsnit 3.4.

I ALGOL II og III skal den globale variable INF ikke bruges af LINE, idet den tilsvarende udgave af LINE-proceduren havde indbygget faste værdier af lbot, lmax og cmax:

lbot	lmax	cmax
5	65	80

Her var altsaa 70 linier pr. side og 80 bogstaver pr. linie. Naar disse tal er rettet til 72 og 72 i ALGOL 4, har dette ikke noget af gøre med ALGOL 4, men skyldes hensynet til linieskriveren, der nu er normalt outputmedium.

I ALGOL II og III krævede LINE-proceduren en anden global variabel af typen integer:

sectno: Sektionsnummer

Dette blev trykt yderst tilhøjre i sideskiftslinien, f.eks. saaledes:

Brugen af sektionsnumre er tildels forladt i ALGOL 4, men i ALGOL II og III fungerede det paa den maade, at en beregning altid begyndte som sektion 1, og ville man efter beregningen gentage den med ændringer i enkelte af input-tallene, blev disse ændringer derefter indlæst og beregningen begyndte automatisk forfra som sektion 2, o.s.v. Dette omtales nærmere i afsnit 3.3.

3.1.2. Proceduren PSHIFT. Ved trykning af tabeller vil man sommetider sikre sig, at der ikke skiftes side midt i tabellen, undtagen hvis det er en meget stor tabel. Hertil bruges proceduren:

```
PSHIFT(n);
```

der udfører sideskift, hvis n>LRest. Ellers sker der intet. Deklarationen er:

```
procedure PSHIFT(n);
value n;
integer n;
if n>LRest then LINE(100);
```

Heri vil sætningen LINE(100) naturligvis ikke give 100 nye linier, men netop sideskift, fordi LRest altid vil være mindre end 100.

Skal man trykke en tabel med data for COMP komponenter, hver skrevet paa een linie, og har tabellen yderligere en overskrift paa 5 linier, bør man altsaa skrive sætningen:

```
PSHIFT(COMP + 5);
```

før tabeltrykningen.

3.2. Forsidetrykning i ALGOL II og III.

Vi tilstræber, at alle rutineprogrammer afleverer deres resultat i form af en outputrapport, der saa vidt muligt skal have en ensartet ydre fremtræden. Den skal saaledes helst begynde med en forside, hvorpaa der blot staar følgende oplysninger:

1. Dato for beregningens udførelse.
2. Sags Nr.
3. Nogle forklarende tekstlinier om beregningens formaal el. lign.
4. Beregningsnummeret.
5. Programmets navn og forkortelse.

Som senere forklaret leveres datoen automatisk af maskinen. Punkt 2, 3 og 4 maa staa paa inputstrimlen, mens punkt 5 er indbygget i programmet.

En typisk forside er vist paa næste side:

Forsideeksempel

May 15, 1964

File No. 358

The Ammonia Co., Gastown

Analysis of Converter Performance

GIER Calculation No. 6789

Calculation of TVA-Type Ammonia Converter

GIER Program TVA-1

Brugerne af programmerne skal derfor ved udfyldning af inputspecifikationerne til hulning komme med oplysninger om forsidedata. Nederst paa siden er vist begyndelsen af en inputspecifikation med tomme rubrikker til denne datagruppe. Formularen er standardiseret og bør bruges ved alle programmer. For den øvrige del af inputmaterialet til programmet - altsaa de egentlige tal - er programmøren mere frit stillet.

Inputstrimlen til forsideeksemplet side 37 maa da have set saaledes ud:

```
Ff
6789
[358
[The Ammonia Co., Gastown
[Analysis of Converter Performance
e
```

Inputformular

0. Cover Data:

```
Ff
..............  Calculation No.
[.............  File No.
[.....................................................  Cover Page Text
[.....................................................    -    -    -
[.....................................................    -    -    -
e               Stop e
```

Før vi ser nærmere paa forsidetrykningen, bemærker vi, at der er sket en betydelig ændring i de anvendte procedurer herfor ved overgang fra ALGOL II til III og især til 4. Dette fremgaar af nedenstaaende figur 2.

ALGOL:	II	III	4
DATA:			
Forside	COVER1 CENTEXT	COVER2	INPUT1
Sektions- overskrift	HEADLINE CENTEXT		Bruges ikke
Tal- materiale	READ5 READ6	READ5 READ6	INPUT1

Figur 2
Oversigt over inputprocedurer i ALGOL.

Vi ser af denne oversigt, at forsideindlæsning og -trykning i ALGOL II blev udført af de to procedurer COVER1 og CENTEXT. I ALGOL II og III kunne en beregning opdeles i forskellige sektioner, og hver af disse sektioner kunne forsynes med en vilkaarlig overskrift, som blev kopieret direkte fra inputstrimlen. Trykning af denne sektionsoverskrift skete i ALGOL II med procedurerne HEADLINE og CENTEXT, men i ALGOL III blev de tre procedurer:

COVER1
CENTEXT
HEADLINE

slaaet sammen til een: COVER2, som baade kunne behandle forside og sektionsoverskrift.

Ved overgangen til ALGOL 4 blev muligheden for at lave sektionsoverskrift fjernet fra disse standardprocedurer, men til gengæld blev der indført en ny procedure, INPUT1, som baade kan udføre indlæsning og trykning

af forside samt indlæsning af almindelige datagrupper af normalt talmateriale. Det sidste blev tidligere gjort af procedurerne READ5 og READ6 (se afsnit 3.3.). Under beskrivelsen af INPUT1 i afsnit 3.4 forklares mere om baggrunden for denne stadige forbedring af procedurerne.

3.2.1. Proceduren CENTEXT. Naar vi sammenligner forsideinputtet side 38 med den trykte forside side 37, ser vi, at den første tekststreng:

[The Ammonia Co., Gastown

blev hullet helt ude i venstre side af papiret, men af maskinen automatisk skal flyttes hen paa midten af papiret. Hvis man lader maskinen kopiere tekststrengen ved hjælp af ALGOL II proceduren trykkopi, kan vi ikke faa indsat de nødvendige SPACE før strengen, saaledes at den bliver centreret. Man kunne da lade hulledamen centrere teksten, altsaa skrive:

[The Ammonia Co., Gastown

og kopiere direkte med trykkopi. Men er der mange beregninger hver dag, er det for besværligt og urimeligt at lade hulledamen centrere strengene.

Vi har derfor lavet en særlig procedure, CENTEXT, som læser een eller flere tekststrenge, gemmer dem i lageret, og perforerer dem igen med indsættelse af saa mange SPACE foran, at de kommer til at staa midt paa siden. Deklarationen er:

```
    procedure CENTEXT(double);
    boolean double;
    begin
       boolean locase;
       integer symb, ib, ic;
       integer array text[1:100];
c1:    symb := læstegn;
       if symb = 53 then go to c4;
       if symb ≠ 134 then go to c1;
       ib := ic := 0;
c2:    symb := læstegn;
       if symb = 64 ∨ symb = 192 then go to c3;
       if symb = 63 ∨ symb = 191 then go to c2;
       if symb ≠ 14 ∧ symb ≠ 142 then ic := ic + 1;
```

```
        ib := ib + 1;
        text[ib] := symb;
        go to c2;
c3:     trykml((80 - ic)/2);
        locase := false;
        tryktegn(60);
        for ic := 1 step 1 until ib do
        begin
           symb := text[ic];
           if locase = symb > 128 then
           begin
              locase := -, locase;
              tryktegn(if locase then 58 else 60)
           end;
           if symb ≥ 128 then symb := symb - 128;
           tryktegn(symb)
        end for ic;
        tryktegn(58);
        if double then LINE(2) else LINE(1);
        go to c1;
c4: end of CENTEXT;
```

Da denne og nogle af de følgende procedurer er skrevet i ALGOL II, er det nødvendigt med en forklaring til nogle af de anvendte standardproceduренavne:

ALGOL II	ALGOL 4	Forklaring
læstegn		Som lyn, men med addition af 128 i upper case
tryktegn		writechar
trykml(n)		Findes ikke. Trykker n SPACE.

Proceduren indeholder en enkelt parameter: double, som er af typen boolean. Paa forsiden vil vi gerne have dobbelt linieafstand mellem tekstlinierne, men ikke i sektionsoverskrifterne. Hvis double er sand, faar vi 2 CAR RET efter en tekstlinie, ellers 1. Hver tekststreng skal indledes med en firkantet venstreparentes: [og afsluttes med CAR RET. Hele rækken af tekstlinier skal afsluttes med et stop-e, d.v.s. et lille e. Hvis proceduren straks møder dette stop-e, uden at have mødt symbolet: [, udfører den slet ingen reperforering.

Proceduren arbejder med et integer array text[1:100], hvori den gemmer de indlæste symboler, saa snart den er kommet ind i en tekststreng. Inden strengen er mødt, analyseres symbolerne kun for stop-e og [. Medens strengen læses, analyseres for CAR RET, der giver hop til c3. Den viste analyse for TAPE FEED er overflødig. Derimod tælles antallet af symboler i strengen, dog saaledes at tegnet for understregning: _, og den lodrette streg: | ikke tælles med, da disse ikke flytter vognen. Naar CAR RET-tegnet er mødt, huller proceduren saa mange SPACE, som svarer til at der er 80 anslag per linie. Derefter hulles hvert tegn ved hjælp af tryktegn-proceduren, idet der indsættes de nødvendige case-tegn. Tekstlinien afsluttes med 1 eller 2 CAR RET.

3.2.2. Proceduren COVER1. Selve forsideindlæsningen og trykningen sker i ALGOL II med proceduren COVER1, der har følgende deklaration:

```
    procedure COVER1(sp1, sp2, name1, name2);
    integer sp1, sp2;
    string name1, name2;
    begin
       integer id;
       real month, day, year;
       switch M := m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12;
       id := læstegn;
       calcno := læst;
       sectno := 1;
       pageno := 2;
       LRest := 65;
       tryktom(50);
       trykml(60);
       month := 1960;
       day := 1961;
       year := 1962;
       go to M[month];
m1:    tryktekst(‹<January›); go to m13;
m2:    tryktekst(‹<February›); go to m13;
m3:    tryktekst(‹<March›); go to m13;
m4:    tryktekst(‹<April›); go to m13;
m5:    tryktekst(‹<May›); go to m13;
```

```
m6:    tryktekst(‹<June›); go to m13;
m7:    tryktekst(‹<July›); go to m13;
m8:    tryktekst(‹<August›); go to m13;
m9:    tryktekst(‹<September›); go to m13;
m10:   tryktekst(‹<October›); go to m13;
m11:   tryktekst(‹<November›); go to m13;
m12:   tryktekst(‹<December›);
m13:   if day < 10 then tryk(‹-n›, day) else tryk(‹-nd›, day);
       tryktegn(27);
       tryk(‹-dddd›, year);
       LINE(8);
       tryktekst(‹<File No. ›);
       trykkopi(‹<[[›);
       LINE(14);
       LRest := LRest - 1;
       sættegn(134);
       CENTEXT(true);
       LINE(1);
       trykml(27);
       tryktekst(‹<GIER Calculation No.›);
       tryk(‹-ndddd›, calcno);
       LINE(2);
       trykml(sp1);
       tryktekst(name1);
       LINE(2);
       trykml(sp2);
       tryktekst(name2);
       LINE(100)
   end of COVER-1;
```

Her træffer vi følgende standardprocedurer i ALGOL II:

ALGOL II	ALGOL 4	Forklaring
læst	read real	
tryktom(n)	Findes ikke	Trykker n TAPE FEED
tryktekst	writetext	
tryk	write	
trykkopi(‹<[[›)	Findes ikke	Kopierer papirstrimlen mellem [og [.
sættegn(n)	Findes ikke	Sætter char lig n.

Proceduren har de fire parametre: sp1, sp2, name1 og name2. De to sidste er tekststrenge og er specificeret som: string. Den første: name1, er programmets fulde navn, altsaa i eksemplet ovenfor:

```
‹<Calculation of TVA-Type Ammonia Converter›
```

og name2 er programmets forkortelse, her:

```
‹<GIER Program TVA-1›
```

De to første parametre, sp1 og sp2, er specificeret som integer og angiver det nødvendige antal SPACE for at faa henholdsvis name1 og name2 centreret paa siden. Her er de henholdsvis 19 og 31.

Proceduren udfører iøvrigt følgende:

1. Der læses et enkelt tegn fra strimlen for at fjerne et eventuelt gammelt tegn fra læseapparatet.
2. Værdien af beregningsnummeret, calcno, indlæses med læst-proceduren. Dette svarer til: calcno := read real.
3. De tre ikke-lokale variable tildeles de rigtige startværdier:

```
sectno :=  1    (sektion 1)
pageno :=  2    (side    2)
LRest  := 65    (65 linier til rest paa siden)
```

4. Perforering af 50 TAPE FEED og 60 SPACE.
5. Trykning af dato. Her staar først de tre sætninger:

```
month := 1960;
day   := 1961;
year  := 1962;
```

Det vil føre for vidt at give en tilbundsgaaende forklaring paa disse mærkelige datoer, men kort fortalt virkede de saaledes. Naar et oversat ALGOL II program var fikseret i en binær (kondenseret) strimmel, var denne forsynet med et specielt lille hjælpeprogram, som lige efter indlæsningen hentede datoen fra tromlen og anbragte den i stedet for de tre tal: 1960,

1961 og 1962.

Proceduren har en switch, M, med 12 etiketter, een for hver maaned. Med sætningen:

```
go to M[month];
```

hoppes til trykning af maanedens navn. Derefter trykkes dagen med 1 eller 2 cifre, et komma og endelig aarstallet. Bemærk den meget klodsede maade hvorpaa maanedsnavnet blev trykt: 11 fuldstændigt ens hopsætninger og 13 etiketter. I COVER2, side 47, vises en lidt fiksere maade at gøre det paa, og i INPUT1, side 65, er brugt en case-konstruktion, som maa siges at være helt tilfredsstillende, i hvert fald saa længe man arbejder i ALGOL.

6. Trykning af 8 CAR RET med proceduren LINE.
7. Trykning af ordet: File No.
8. Kopiering af sagsnummeret fra inputstrimlen med:

```
trykkopi(‹<[[›);
```

Sagsnummeret kan ikke indlæses og trykkes som et tal, da der undertiden kan være et appendiks til nummeret, f.eks.: 358-a.

9. Trykning af 15 CAR RET. Her har vi kun skrevet: LINE(14), fordi kopieringen af sagsnummeret mellem de to [-tegn maa formodes at have givet et ekstra CAR RET-tegn. Til gengæld maa vi da med sætningen:

```
LRest := LRest - 1;
```

trække 1 fra i linietælleren. Ligeledes maa vi med sætningen:

```
sættegn(134);
```

simulere, at vi har rykket inputstrimlen en plads tilbage til [-tegnet igen, fordi vi jo allerede har læst dette tegn een gang, nemlig som afslutning paa kopieringen af sagsnummeret. I ALGOL II havde man ikke den automatiske effekt, at det sidst læste tegn altid stod i standardvariablen: char, saa dette maatte programmeres eksplicit.

10. Indlæsning og centertrykning af en eller flere forsidetekststrenge med CENTEXT-proceduren. Der anbringes 2 CAR RET efter hver linie.
11. Trykning af en ekstra CAR RET.
12. Trykning af 27 SPACE.
13. Trykning af den faste tekststreng: ‹<GIER Calculation No.›.

14. Trykning af beregningsnummeret, calcno.

15. Trykning af 2 CAR RET.

16. Trykning af sp1 SPACE.

17. Trykning af programmets fulde navn: name1.

18. Trykning af 2 CAR RET.

19. Trykning af sp2 SPACE.

20. Trykning af programmets forkortelse: name2.

21. Sideskift med LINE(100). Vi befinder os nu øverst side 2 i outputrapporten.

3.2.3. Proceduren HEADLINE. Med denne procedure kan man indlæse og trykke sektionsoverskrifter. Deklarationen er:

```
    procedure HEADLINE(n, out);
    value n;
    integer n;
    label out;
    begin
       integer symb;
L1:    symb := læstegn;
       if symb = 25 then
       begin
          trykstop;
          tryktom(50);
          go to out
       end;
       if symb ≠ 53 ∧ symb ≠ 134 then go to L1;
       sættegn (symb);
       PSHIFT(n);
       trykml(35);
       tryktekst(‹<SECTION›);
       if sectno < 10 then tryk(‹-n›, sectno) else
       tryk(‹-nd›, sectno);
       LINE(2);
       CENTEXT(false);
       LINE(1)
    end of HEADLINE;
```

Proceduren har to parametre: et heltal: n og en etikette: out. Hvis værdien af LRest er mindre end n, vil proceduren lave sideskift. Dette bruges for at undgaa, at den nye sektion begynder for langt nede paa en side.

Brugen af etiketten: out kræver en nærmere forklaring. For programmer skrevet i maskinsprog har vi haft den konvention, at der paa inputstrimlen skal staa et lille z som tegn paa, at beregningen er slut. Da tegnet vil staa paa et saadant sted af strimlen, at programmet er ved at indlæse talmateriale til den næste beregningssektion, maa det være den procedure, der varetager sektionsoverskriften, som analyserer for tilstedeværelse af z-tegnet. Naar tegnet er mødt, hulles et stoptegn med proceduren trykstop (=writechar(11) i ALGOL 4) og 50 TAPE FEED, og programmet hopper til etiketten out. Den tilsvarende aktuelle parameter maa da være en etikette anbragt sidst i programmet i den yderste blok.

Proceduren leder iøvrigt efter tegnene: e og [. Saa snart et af disse mødes, simulerer proceduren at det kan læses een gang til fra strimlen med:

```
sættegn(symb);
```

Derefter begynder den egentlige trykning, idet der eventuelt først skiftes side med:

```
PSHIFT(n);
```

Ordet: SECTION trykkes midt paa siden, efterfulgt af sektionsnummeret. Der hulles 2 CAR RET og proceduren CENTEXT kaldes til kopiering og centrering af overskriftslinierne. Endelig hulles 1 CAR RET.

3.2.4. Proceduren COVER2. Denne procedure er skrevet i ALGOL III og erstatter de ovennævnte tre procedurer: COVER1, CENTEXT og HEADLINE. Deklarationen af COVER2 er:

```
procedure COVER2(length1, length2, name1, name2, RESET, freeline, out);
value length1, length2, freeline;
integer length1, length2, freeline;
string name1, name2;
procedure RESET;
label out;
```

```
begin
   boolean locase;
   integer i, j, symb;
   boolean array DATE[1:3];
   procedure printleft(n);
   value n;
   integer n;
   output(if n < 10 then {-n} else
   if n < 100 then {-nd} else {-ndd}, n);
   procedure CENTEXT(n);
   value n;
   integer n;
   begin
      integer array text[1:160];
c1:   for symb := inchar while symb ≠ 53 ∧ symb ≠ 134 do;
      if symb = 134 then
      begin
         i := j := 0;
         for symb := inchar while
         symb ≠ 64 ∧ symb ≠ 192 do
         begin
            i := i + 1;
            if symb ≠ 14 ∧ symb ≠ 142 then
            j := j + 1;
            text[i] := symb
         end for symb;
         outsp((80-j)/2);
         locase := false;
         outchar(60);
         for j := 1 step 1 until i do
         begin
            symb := text[j];
            if locase ≡ symb > 128 then
            begin
               locase := -, locase;
               outchar(if locase then 58 else 60)
            end if case;
            if symb ≥ 128 then symb := symb -128;
            outchar(symb)
         end for j;
         outchar(58);
```

```
      LINE(n);
      go to c1
   end if symb = 134
end CENTEXT;
procedure TAPEFEED;
for i := 1 step 1 until 50 do outchar(63);
procedure M(name);
string name;
begin
   i := i +1;
   if i = j then outtext(name)
end M;
integer procedure date(n);
value n;
integer n;
date := split(DATE[n], 21, 39, j);
sectno := sectno + 1;
if sectno = 1 then
begin
   drumtop := drumplace;
   RESET;
   i := inchar;
   calcno := inone;
   pageno := 2;
   LRest := 64;
   TAPEFEED;
   outclear;
   outsp(60);
   drumplace := 24x40+30;
   from drum(DATE);
   i := 0;
   date(1);
   M({< January});
   M({< February});
   M({<    March});
   M({<    April});
   M({<      May});
   M({<     June});
   M({<     July});
   M({<   August});
   M({<September});
```

```
   M(‡<  October†);
   M(‡< November†);
   M(‡< December†);
   printleft(date(2));
   outchar(27);
   output(‡-nddd†, date(3));
   LINE(8);
   outtext(‡<File No. †);
   outcopy(‡<[[†);
   LINE(14);
   setchar(134);
   CENTEXT(2);
   LINE(1);
   outsp(27);
   outtext(‡<GIER Calculation No.†);
   output(‡-ndddd†, calcno);
   LINE(2);
   outsp((80-length1)/2);
   outtext(name1);
   LINE(2);
   outsp((80-length2)/2);
   outtext(name2);
   LINE(100);
   drumplace := drumtop
end if sectno = 1;
for symb := inchar while symb ≠ 53 ∧ symb ≠ 134 do
if symb = 25 then
begin
   outsum;
   TAPEFEED;
   go to out
end if z;
setchar(symb);
if sectno ≠ 1 then LINE(3);
PSHIFT(freeline);
outsp(35);
outtext(‡<SECTION†);
```

```
   printleft(sectno);
   LINE(2);
   CENTEXT(1);
   LINE(1)
end COVER2;
```

Her maa vi ogsaa give en liste over standardprocedurer i ALGOL III og deres ækvivalens i ALGOL 4:

ALGOL III	ALGOL 4	Forklaring
output	write	
inchar	Som lyn, men med addition af 128 i upper case.	
outsp(n)	Findes ikke.	Trykker n SPACE
outchar	writechar	
outtext	writetext	
split(B,P,Q,N)	Findes ikke.	Ekstraherer heltallet fra bit P til bit Q i boolean B og assigner det til split og N.
inone	read real	
outclear	Findes ikke.	Ækvivalent med writechar(28).
outtext	writetext	
outcopy(‡<[[‡)	Findes ikke.	Kopierer papirstrimlen mellem [og [.
setchar(n)	Findes ikke.	Sætter char = n.
outsum	Findes ikke.	Svarer omtrent til writechar(61), dog uden trykning af checksum.

Proceduren erstatter COVER1 paa den maade, at medens man i ALGOL II skulle programmere sin forside- og sektionsoverskriftblok saaledes:

```
........
........
    sectno := 1;
AA:begin
      .....
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno = 1 then
      begin
         COVER1(........);
         comment Her anbringes eventuelt nulstilling af talmateriale;
      end;
```

```
      HEADLINE(  ,  )
   end;
```

saa kan dette i ALGOL III skrives lidt simplere:

```
.....
.....
    sectno := 0;
AA: begin
       .....
       procedure RESET;
       begin
          comment Denne procedure foretager eventuel nulstilling;
       end RESET;
       comment library COVER2;
       COVER2(.....)
    end;
```

Proceduren COVER2 indeholder følgende syv formelle parametre:

integer length1:	Antal bogstaver i name1.
integer length2:	Antal bogstaver i name2.
string name1:	Programmets fulde navn.
string name2:	Programmets forkortelse.
procedure RESET:	Proceduren COVER2 kalder proceduren RESET lige før trykningen af forsiden. Den kan bruges til nulstilling af talmateriale, etc.
integer freeline:	Hvis LRest er mindre end freeline, vil COVER2 lave sideskift før trykning af sektionsoverskriften.
label out:	Naar stop-zet mødes, hopper COVER2 til denne etikette.

Følgende globale variable maa være tilstede (deklareret i den yderste blok):

```
integer LRest, calcno, sectno, pageno,
procedure LINE, PSHIFT.
```

Disse er de samme som i COVER1. En ny global variabel er tilføjet:

integer drumtop: Før kaldet af RESET assigner COVER2 startværdien af drumplace til drumtop.

I programstumperne ovenfor er standardprocedurerne indsat ved hjælp af betegnelsen: comment library, der delvis svarer til copy i ALGOL 4. Der er dog den forskel, at comment library kræver en reperforering af den anvendte programstrimmel.

Om regnemaaden af COVER2 kan følgende detailler gives:

En række lokale procedurer er deklareret i begyndelsen af COVER2:

printleft(n): Trykker et venstrenormaliseret heltal med højst tre cifre. Kun minusset trykkes før cifrene.

CENTEXT: Svarer til den tidligere procedure.

TAPEFEED: Trykker 50 TAPEFEED.

M(name): Trykker maanedsnavnet, name, hvis et internt tælleværk passer med maanedens nummer.

date(n): Denne procedure henter datoen frem fra tromlekanal 24, celle 28-30.

Proceduren begynder med at lægge 1 til sectno, som derfor maa være sat til 0 af programmøren i begyndelsen af programmet.

Hvis sectno = 1 udføres forsidetrykningen, ellers kun sektionsoverskriften.

Forsidetrykningen begynder med at sætte drumtop lig drumplace, altsaa nummeret paa den øverste frie tromlecelle. Derefter kaldes RESET til nulstilling af de tromlekanaler, som skal bruges til lagring af inputmaterialet. Derefter fortsættes som i COVER1 med beskedne ændringer. Bemærk brugen af M-proceduren:

```
i := 0;
j := month;
M(‹< January›);
M(‹< February›);
.....
```

M-proceduren øger i med 1 og trykker navnet, hvis i = j.

Efter forsidetrykning følger trykning af sektionsoverskrift, omtrent som udført af HEADLINE.

3.3. Input af Datagrupper i ALGOL II og III.

Saa længe vi kun laver ganske smaa programmer i ALGOL, er der ikke større problemer forbundet med at lade programmet indlæse de ganske faa variable, som det faar brug for, fra en inputstrimmel, f.eks:

```
A := read real;
B := read real;
.........
```

hvor vi har benyttet den simplest mulige indlæseprocedure, her i ALGOL 4. De tilsvarende procedurer hedder i ALGOL II og III henholdsvis læst og inone.

Saa snart programmerne bliver noget større, bliver der adskilligt flere tal, som skal indlæses til hver beregning, og det vil da ofte være praktisk at opdele det egentlige inputmateriale til et program i flere grupper, saaledes at man for beregning af et kemisk apparat f.eks. har apparatdimensionerne samlet i een gruppe, driftsbetingelserne i den næste gruppe, o.s.v.

Naar brugeren af programmet skal udføre en beregning, maa han udfylde en tilsvarende inputspecifikation, hvoraf der findes en kopi som bilag til programvejledningen. Paa side 38 saa vi, hvorledes forsidedata til en normal beregning skulle udfyldes, og vi gengiver nedenfor den komplette inputspecifikation til et meget primitivt program, som kun indeholder to datagrupper af teknisk indhold:

Inputspecifikation

0. Cover Data:

```
Ff
.............Calculation No.
[.............File No.
[.....................................................Cover Page Text
[..................................................... -    -    -
[..................................................... -    -    -
e              Stop e
```

1. Headline Data:

```
[.....................................................Section Headline Text
[..................................................... -       -      -
[..................................................... -       -      -
e              Stop e
```

2. Rørdata:

(0).....	Rørantal
(1).....	Højde (m)
(2).....	Udvendig diameter (mm)
(3).....	Indvendig diameter (mm)
e	Stop e

3. Driftsbetingelser:

(0).....	Gasmængde (kgmol/h)
(1).....	Temperatur (gr.C)
(2).....	Tryk (atm.abs.)
e	Stop e
z	Stop

Lad os antage, at vi har udfyldt inputspecifikationen saaledes:

```
Ff
9876
[358
[Beregningseksempel
e
[Basiseksempel
e
100  8  110  90  e
1298.3  500  23  e
z
```

Der er altsaa regnet med 100 rør i denne beregning. Ønsker vi at gentage beregningen med samme talmateriale, men blot med 102 rør og derefter med 104 rør, kan inputstrimlen for alle 3 beregninger skrives saaledes:

```
Ff
9876
[358
[Beregningseksempel
e
[Basiseksempel
e
100  8  110  90  e
1298.3  500  23  e
e  102  ee
e  104  ee
z
```

Princippet er altsaa det, at hvis det første tal i en datagruppe skal ændres, skriver vi blot det nye tal og derefter stop-eet. Skal datagruppen være helt uændret, skriver vi blot stop-eet. Programmet skal altsaa gemme inputtallene fra den forrige sektion og bruge dem til den næste sektion, hvis de ikke bliver ændret ved indlæsningen.

Hvis det i stedet for rørantallet havde været den indvendige diameter, som skulle have været varieret, f.eks. fra 90 til 92 og 94, da kan de to nye datalinier skrives enten:

```
e  ddd  92  ee
e  ddd  94  ee
```

eller:

```
e  q  3  92  ee
e  q  3  94  ee
```

Her betyder d: ditto, d.v.s. tallet er uændret fra før. Bogstavet q efterfulgt af et helt tal betyder, at vi skal rykke saa mange pladser frem i datagruppen, som dette hele tal angiver.

Brugen af de specielle bogstaver: d, e og q gaar tilbage til GIER-programmer skrevet i maskinsprog og for bogstavet: e, endda helt tilbage til programmer for DASK. Metoden er bibeholdt for GIER-ALGOL-programmer, fordi det er praktisk, og for at bevare kontinuiteten i vore principper for

udfyldning af inputspecifikationer.

Ovenstaaende eksempel svarer til de muligheder, som vi har brugt i ALGOL II og III. Bemærk, at der ikke behøves noget synligt afslutningstegn efter hvert tal, saaledes som der nu kræves i vor brug af ALGOL 4 (komma, semikolon, etc.). Bemærk ogsaa, at vi i ALGOL II og III har indlæsning og trykning af sektionsoverskrift som fast standard, medens dette ikke er muligt i ALGOL 4, i hvert fald ikke, hvis man kun bruger standardinputproceduren INPUT1.

3.3.1. Proceduren READ5. Hvis man læser et inputtal med een af de simple procedurer:

```
A := læst;          ALGOL II
A := inone;         ALGOL III
A := read real;     ALGOL 4
```

har man ingen mulighed for at opdage de specielle bogstaver (d, e og q), som vi gerne ville bruge. Derfor har vi lavet en speciel procedure, READ5, som i ALGOL II har deklarationen:

```
    procedure READ5(a, p, stope);
    real a;
    integer p;
    label stope;
    begin
       integer symb;
L1:    symb := læstegn;
       if symb = 53 then go to stope;
       if symb = 52 then go to L2;
       if symb = 40 then p := p + læst;
       if symb = 16 ∨ symb ≥ 1 ∧ symb ≤ 9 ∨ symb = 32 ∨ symb = 59
       ∨ symb = 155 then
       begin
          sættegn (symb);
          a := læst;
          sættegn(tegn);
          go to L2
       end;
       go to L1;
L2: end of READ-5;
```

De tre parametre er:

```
real a;
integer p;
label stope;
```

Her er a den variable, hvis talværdi skal indlæses. Det sker med sætningen:

```
a := læst;
```

men først læser proceduren strimlen tegn for tegn. Møder den et e, hoppes til etiketten: stope. Møder den et d, hopper den ud af proceduren, idet værdien af a er uændret. Møder den et q, vil den løbende værdi af p blive øget med det næste heltal paa strimlen:

```
p := p + læst;
```

Er det læste tegn noget, som danner begyndelsen til et tal, d.v.s. et ciffer, minus, punktum eller 10-potens, retableres det læste tegn ved:

```
sættegn(symb);
```

og a indlæses derefter. Efter indlæsningen af a maa vi igen skrive:

```
sættegn(tegn);
```

fordi tallet kan være afsluttet med et af de søgte bogstaver: d, e eller q.

Proceduren kan f.eks. bruges saaledes:

```
begin
   integer i;
   real X, Y, Z, V;
   array A[0:3];
   comment library READ5;
   ........
   ........
```

```
AA:     for i := 0 step 1 until 4 do
        READ5(A[i], i, EE);
EE:     X := A[0];
        Y := A[1];
        Z := A[2];
        V := A[3];
        ........
        go to AA;
        ........
    end program;
```

Her findes talmaterialet i programmet som talsættet: A[0:3], og vi indlæser værdierne med i-for-sætningen. Bemærk, at det er nødvendigt at kalde READ5 een gang mere end der er elementer i A, for at være sikker paa, at stop-eet er læst, hvis alle fire inputtal staar paa strimlen. Hvis man synes, at det er upraktisk at betegne talmaterialet med A[0], A[1], o.s.v., da kan man blot gøre som vist, nemlig indføre en simpel variabel: X, Y, Z og V for de enkelte elementer og regne videre med dem.

3.3.2. Proceduren READ6. I eksemplet ovenfor var talmaterialet anbragt i lageret som talsættet A. For større programmer med større datamængder er det nødvendigt at lagre inputtallene paa tromlen eller disken. Vi har tidligere set, at disse er opdelt i kanaler med 40 tal paa hver. Nummererer vi cellerne i hver kanal fra 0 til 39, bliver værdien af tromleplads eller drumplace for celle C paa kanal K:

$$40 \times K + C$$

Vi har da vedtaget, at vi normalt anbringer hver inputdatagruppe paa sin tromlekanal, selv om der ofte vil være betydeligt mindre end 40 tal i hver gruppe. Paa den anden side er der tilfælde, hvor mængden af tal i en datagruppe varierer fra den ene sektion til den næste, f.eks. hvis antallet af komponenter i en gasblanding varierer. Derfor ville det være svært at anbringe datagrupperne helt tæt op ad hinanden paa tromlen uden mulighed for fejltagelse. Den opmærksomme læser vil se, at vi her opererer med et sæt own arrays deklareret i den yderste blok og med grænserne: [0:39], eller rettere [0:29], som vist nedenfor.

Proceduren READ6 indlæser een eller flere datagrupper til en opgivet tromlekanal og de følgende kanaler ved hjælp af READ5. Har vi fire datagrupper, som ønskes indlæst til kanal 67, 68, 69 og 70, skriver vi:

```
READ6(67, 70);
```

Der skal da være 4 stop-eer paa strimlen, og hver datagruppe bliver anbragt fra celle 0 paa sin kanal. Deklarationen af READ6 ser saaledes ud i ALGOL II:

```
procedure READ6(ch1, chn);
integer ch1, chn;
begin
   integer c, m, p, r;
   array old, new[0:39];
   procedure dr;
   tromleplads := 40×c + 39;
   for c := ch1 step 1 until chn do
   begin
      dr;
      fratromle(old);
      dr;
      fratromle(new);
      m := 0;
      for p := 0 step 1 until 28 do
      READ5(new[p], p, L1);
L1:   r:= 1;
      for p := 0 step 1 until 28 do
      begin
         if old[p] ≠ new[p] then m := m + r;
         r := 2×r
      end;
      new[39] := m;
      dr;
      tiltromle(new)
   end for c
end for READ-6;
```

Da vi ved den senere trykning af inputmaterialet i outputrapporten gerne vil overspringe de inputtal, som er uforandrede fra sidste sektion, er READ6 indrettet til at kontrollere hvilke tal, der er forskellige fra dem i forrige sektion. Den gemmer informationen herom i den sidste celle (39) paa tromlekanalen. Kalder vi indholdet af denne celle for m, er m = 0, hvis alle tal er uændret fra før. Er m ≠ 0, er et eller flere tal ændret. Er indholdet i celle nr. j ændret, er der adderet $2\uparrow j$ til det oprindelige nul i celle 39. Indeholder celle 39 f.eks. tallet 43, skriver vi dette som:

$$1\times2\uparrow0 + 1\times2\uparrow1 + 1\times2\uparrow3 + 1\times2\uparrow5$$

og ser heraf, at indholdet i cellerne 0, 1, 3 og 5 er ændret. Denne information udnyttes af procedurerne TABLE2 og T3 omtalt i afsnit 3.5.

READ6 har de to formelle parametre: ch1 og chn, som er numrene paa den første og sidste tromlekanal, til hvilke datagrupperne skal indlæses. Proceduren opererer med de to lokale talsæt:

```
old, new[0:39];
```

For hver datagruppe overføres den tilsvarende tromlekanal til begge disse talsæt, og der læses med READ5 til new[p], hvor p løber fra 0 til 28. Naar stop-eet er mødt, sammenlignes indholdet af de to talsæt, og indholdet af celle 39 beregnes. Derefter føres new tilbage til tromlen.

Naar vi kun udnytter cellerne fra 0 til 28, skyldes det den særlige maade, paa hvilken tal af typen integer behandles i ALGOL II og III, nemlig som flydende tal paa samme maade som tal af typen real. Herved er de ti bits i cellen tabt til eksponenten, og vi kan derfor ikke (uden umaadeligt besvær) gemme et almindeligt heltal i hele cellen.

En inputblok til indlæsning af fire datagrupper til kanalerne 67, 68, 69 og 70 kan programmeres saaledes:

```
begin comment input block;
   comment library READ5;
   comment library READ6;
   READ6(67, 70)
end input;
```

I forsideblokken side 51 er anbragt kommentaren:

Her anbringes eventuelt nulstilling af talmateriale.

Ønsker vi at nulstille de fire kanaler 67-70, kan dette i ALGOL II skrives saaledes:

```
begin
   integer i;
   array B[0:39];
   for i := 0 step 1 until 39 do B[i] := 0;
   tromleplads := 70x40 + 39;
   for i := 1 step 1 until 4 do til tromle(B)
end;
```

I nogle af vore ældre programmer vil man se, at vi ikke sætter tallene paa kanalerne lig med nul, men f.eks. lig med -1234. Dette tal skal opfattes som et slags nonsenstal. Da de nedenfor omtalte trykprocedurer, TABLE2 og T3, er indrettet til at overspringe de celler, som er uændret efter indlæsningen, vil man ikke faa trykt celler med nul, hvis disse er uændret. Da man ofte gerne vil se, at brugeren virkeligt har specificeret et nul i disse celler, er det bedre at bruge et nonsenstal som f.eks. -1234.

3.4. Proceduren INPUT1 (ALGOL 4).

Som tidligere omtalt er proceduren INPUT1 en ALGOL 4 procedure, som baade varetager input og trykning af forside som COVER1 eller COVER2, men ogsaa input af almindelige datagrupper, omtrent som READ6. Input af sektionsoverskrifter er derimod udeladt af INPUT1.

Nedenstaaende gennemgang af INPUT1 tager sigte paa baade at forklare dens funktioner, men ogsaa paa at diskutere forskellige programmeringsmæssige finesser, som er indbygget i den, og som kan være af interesse ogsaa for andet programmeringsarbejde.

Deklarationen af INPUT1 er:

```
integer procedure INPUT1(type, track, area, total, item, name1, ch1,
                         name2, ch2, format);
value ch1, ch2, format;
```

```
boolean type;
integer track, area, total, item, ch1, ch2, format;
string name1, name2;
begin
   boolean first, t, numb;
   integer ch, i, j, cmax, lmax, stope, inf, case, mxdat, no, mxno, ref, ty,
   at, c;
   real r;
   array DATA[0:39];
   boolean array TY[0:7];
   boolean procedure T;
   begin
      boolean b;
      integer i, j, ck;
L:    for i := lyn while i ≠ 6 ∧ i ≠ 53 do;
      if i = 6 then
      begin
         j := ck := ch := 0;
         b := false;
         for i := lyn while i ≠ 64 do
         begin
            if i < 63 then
            begin
               b := b ∨ (boolean i shift ck);
               ck := ck+10;
               if ck = 40 then
               begin
                  DATA[j] := real b;
                  j := j + 1;
                  b := false;
                  ck := 0
               end;
               if i ≠ 58 ∧ i ≠ 60 then ch := ch + 1
            end
         end read text;
```

```
      if ck = 0 then j := j-1 else
      DATA[j] := real b;
      for i := (cmax - ch): 2 step -1 until 1 do
      writechar(0);
      writechar(60);
      for i := 0 step 1 until j do
      begin
         b := boolean DATA[i];
         for ck := 0 step -10 until -30 do
         writechar(integer((b shift ck) ∧ 40 63))
      end for i;
      T := true
   end [ else T := false
end procedure T;
TY[0] := 57575757575757 51;
TY[1] := 51515151515157 57;
TY[2] := 51515151515151 57;
TY[3] := 51515151585156 51;
TY[4] := 51515151515151 57;
TY[5] := 51515151515151 55;
TY[6] := 51515354515151 51;
TY[7] := 51515152575251 51;
t := type;
if t shift 6 then char := 0;
at := j := track;
stope := 0;
mxdat := total;
where(t<inform), inf);
get(DATA, inf, 1);
INF := boolean DATA[format];
cmax := integer (INF ∧ 40 1023);
lmax := integer((INF shift -10) ∧ 40 1023);
select(integer(boolean DATA[4] ∨ boolean DATA[5]));
c := if t then read integer else 0;
```

```
if t shift 2 then
begin comment cover printing;
   LRest := lmax;
   calcno := c;
   for i := 50 step -1 until 1 do writechar(63);
   writechar(72);
   writetext({<HALDOR TOPSØE});
   for i := cmax -31 step -1 until 1 do
   writechar(0);
   writetext(case integer((boolean DATA[0] shift 10) ∧ 40 1023) of (
   {<  January}, {< February}, {<    March},
   {<    April}, {<      May}, {<     June},
   {<     July}, {<   August}, {<September},
   {<  October}, {< November}, {< December}));
   writeinteger({dd}, integer((boolean DATA[0] shift -20) ∧ 40 1023));
   writechar(27);
   writeinteger({dddd}, integer(boolean  DATA[0]  ∧ 20  0 20  m));
   LINE(lmax:9);
   if t shift 1 then
   begin
      writetext({<File No. });
      for i := lyn while i ≠ 6 do;
      writechar(60);
      for i := lyn while i ≠ 64 do
      writechar(i)
   end;
   LINE(lmax:4);
   for i := i while T do LINE(2);
   LINE(2);
   for i := (cmax-27):2 step -1 until 1 do
   writechar(0);
   writetext({<GIER Calculation No.});
   writeinteger({dddddd}, calcno);
   LINE(2);
```

```
      for i := (cmax - ch1):2 step -1 until 1 do
      writechar(0);
      writetext(name1);
      LINE(2);
      for i := (cmax-ch2):2 step -1 until 1 do
      writechar(0);
      writetext(name2);
      pageno := 2;
      char := case := 0;
      LINE(LRest + 1)
   end print front page else
   if t shift 6 then
   begin
      if t shift 1 then
      begin
         for i := lyn while i ≠ 6 do;
         char := 128;
         for i := lyn while i ≠ 64 do
         begin
            if i = 58 ∨ i = 60 then
            char := (i - 58)×64
         end
      end skip text;
      for i := 50 step -1 until 1 do writechar(63);
      writechar(72);
      pageno := 1;
      calcno := c;
      LRest := -integer((INF shift -20) ∧ 40 1023);
      LINE(0)
   end no front page;
   if t shift 5 then
   begin
      cancel(¢<xyz1zyx¢);
      if reserve(¢<xyz1zyx¢, integer(t ∧ 20 0 20 m))≠ 0 then
      begin
         select (integer DATA[7]);
L:       writetext(¢<
Calculation not possible¢);
```

```
            go to L
         end;
         where(‹<xyz1zyx›, area)
      end;
      first := t shift 4;
      if t shift 3 then
      begin comment read data;
         case := if char > 127 then 128 else 0;
L3:      no := 0;
         track := at;
         if first then
         begin
            for i := 0 step 1 until 39 do DATA[i] := 0
         end else get(DATA, area, at);
         mxno := 40;
         ref := 1;
L1:      ch := lyn;
L5:      ty := integer((TY[integer((boolean ch shift -3) ∧ 40 7)]
         shift - integer (boolean ch ∧ 40 7)×5) ∧ 40 15);
         case ty of
         begin
            go to L1;
            begin
               case := (ch - 58)×64;
               go to L1
            end 2;
            begin
               put(DATA, area, at);
               if first then
               for i := 0 step 1 until 39 do DATA[i] := 0;
               at := at + 1;
               total := no;
               i := item;
               for i := i while i > mxno do
               begin
                  if first then put(DATA, area, at);
                  mxno := mxno + 40;
                  at := at + 1
               end;
```

```
            stope := stope + 1;
            go to if stope = mxdat then ZZ else L3
         end 3;
         begin
            numb := false;
            no := no + 1;
            char := 0
         end 4;
         begin
            char := 0;
            numb := false;
            no := no + read integer
         end 5;
         begin
            if mxdat = 0 then mxdat := stope;
            if stope = 0 then at := 0;
            go to ZZ
         end 6;
         begin
            if case ≠ 0 then
            begin
               if ch ≠ 32 then go to L1
            end;
L4:         char := ch + case;
            r := read real;
            numb := true;
            no := no + 1
         end 7;
         go to if case ≠ 0 then L4 else L1
      end case of;
L2:   if no > mxno then
      begin
         put(DATA, area, at);
         at := at + 1;
         if first then
         begin
            for i := 0 step 1 until 39 do DATA[i] := 0
         end else get(DATA, area, at);
```

```
            ref := ref + 40;
            mxno := mxno + 40;
            go to L2
         end new track;
         if numb then DATA[no - ref] := r;
         case := if char > 127 then 128 else 0;
         ch := if char > 127 then char - 128 else char;
         go to L5;
ZZ:      INPUT1 := at;
         track := j;
         total := mxdat;
      end read data;
      type := type ∧ 30113033m;
end INPUT1;
```

3.4.1. Parametre i INPUT1. De ti formelle parametre er:

type boolean: Denne parameter er opbygget af 7 almindelige booleans i bits 0-6 og en integer i bits 20-39:

Bit:	
0	Læs beregningsnummer (0:nej, 1:ja)
1	Læs sagsnr. (0:nej, 1:ja)
2	Læs og tryk forside inklusive stop-e (0:nej, 1:ja)
3	Læs normale inputdatagrupper (0:nej, 1:ja)
4	Nulstil inputkanaler før læsning (0:nej, 1:ja)
5	Reserver et disk-areal (0:nej, 1:ja)
6	Første kald af proceduren i programmet (0:nej, 1:ja)
20-39	Antallet af kanaler, som skal reserveres.

Parametren type er kaldt ved name, og efter kaldet af INPUT1 vil bits 0, 1, 2, 4, 5 og 6 være sat til nul. Dette svarer nemlig til den normale anvendelse af INPUT1, hvor første kald giver forsidetrykning og datainput og de følgende kun datainput.

track integer: Dette er det relative nummer paa den kanal i inputomraadet til hvilken den første inputdatagruppe ønskes læst. Inputomraadet forklares nærmere under næste parameter. Naar INPUT1 forlades, er værdien af

track uændret, men internt peger track altid paa den første kanal i den datagruppe, som er ved at blive indlæst. Hver datagruppe skal afsluttes med et stop-e, men kan godt indeholde mere end 40 celler og vil da optage flere kanaler.

area integer: Denne parameter indeholder arealordet for inputdataomraadet. Ønsker man at bruge omraadet {<free} eller et andet omraade med et eksisterende navn, maa programmøren inden første kald af INPUT1 have beregnet værdien af area ved et kald af where. Ønsker man ikke at have denne beskedne ulejlighed, kan man lade INPUT1 selv reservere et omraade, med et eller andet navn, som vi ikke bekymrer os om. INPUT1 vil da selv kalde where og aflevere arealordet i: area.

total integer: Dette er det samlede antal datagrupper, som ønskes indlæst, altsaa lig med antallet af stop-eer paa strimlen. En særlig anvendelse er den, at man sætter total = 0 før kaldet af INPUT1. Proceduren vil da blive ved at læse datagrupper, indtil den møder et stop-z. Proceduren sætter da total lig med antallet af datagrupper, som er indlæst. Efter indlæsning af hver enkelt datagruppe, d.v.s. umiddelbart efter at stop-eet er mødt, vil INPUT1 midlertidigt sætte total lig med det samlede antal elementer, som er indlæst (eller oversprunget) i denne datagruppe. Denne information kan udnyttes i den følgende parameter.

item integer: Dette er det maksimale antal elementer i en datagruppe. Naar et stop-e er mødt, og total indeholder antallet af faktisk indlæste elementer, vil den plads, der reserveres til datagruppen, svare til det største af item og total, dog mindst 1 kanal. For smaa datagrupper kan man lade item være lig med 40, men for større datagrupper er der rige muligheder for snedige effekter ved at lade item være en integer procedure, der opererer paa track og total som ikke-lokale variable.

name1 string: Programmets fulde navn (som i COVER1 og COVER2).
ch1 integer: Antal bogstaver i name1.
name2 string: Programmets forkortelse.
ch2 integer: Antal bogstaver i name2.
format integer: Det ønskede papirformat i outputrapporten. format = 1 giver A4 og format = 2 giver A3.

INPUT1 integer: Naar proceduren forlades, er INPUT1 lig med det relative nummer paa den første frie kanal efter de benyttede inputkanaler. Hvis

der ikke er indlæst datagrupper før stop-zet er mødt, er INPUT1 lig nul.

Det er nødvendigt at lade de fem vigtige parametre:

type, track, area, total og item

som alle er kaldt ved name, være deklarerede variable, ikke talkonstanter.

3.4.2. Globale variable i INPUT1. Proceduren INPUT1 skal have adgang til de samme ikke-lokale variable som LINE-proceduren:

LRest integer	Det aktuelle antal linier paa siden, som endnu kan udnyttes til trykning.
calcno integer	Beregningsnummeret.
pageno integer	Sidetallet. Det er nummeret paa den næste side, som skal trykkes.
INF boolean	En celle med de sammenpakkede værdier af lbot, lmax, og cmax, se side 30.

Det første kald af INPUT1 vil tildele værdier til disse fire variable.

3.4.3. Informationsomraade til INPUT1. Brugen af informationscellen, INF, er kun en del af den forbedrede informationsbehandling, som er indbygget i INPUT1. For at INPUT1 kan bruges paa den rette maade, er det en forudsætning, at der er oprettet et omraade paa disken med navnet {<inform} og indeholdende een kanal. Paa denne kanal skal følgende data være tilstede:

Celle	Indhold
0	Dato: Bits 0-9: maaned bits 10-19: dag bits 20-39: aar
1	Data for papirformat 1 (A4) bits 0-9: Ledig bits 10-19: lbot bits 20-29: lmax bits 30-39: cmax Disse variable er forklaret side 30, og det er normalt dem, som anbringes i cellen INF.
2	Data for papirformat 2 (A3).

3	Data for papirformat 3 (ledig).
4	Inputmediumbit.
5	Outputmediumbit.
6	Bit for meddelelsesmedium.
7	Bit for fejlmedium.

Cellerne 4-7 indeholder den paagældende bit som et heltal i pos. 39. Meningen med disse bits er, at man ved en selectoperation, f.eks.:

```
select(integer celle[4]);
```

faar valgt det ønskede inputmedium. Ligeledes vil:

```
select(integer(celle[4]∨celle[5]));
```

samtidigt vælge input- og outputmedium. Det er her forudsat, at celle [0:39] er deklareret som et boolean array.

Meddelelsesmediet, hvis bit staar i celle 6, er beregnet til mindre vigtige meddelelser, medens fejlmediet, hvis bit staar i celle 7, er beregnet paa alvorligere fejlmeddelelser, hvor beregningen ikke kan fortsættes.

Normalt behøver hverken brugeren af programmet eller programmøren at bekymre sig om indholdet af omraadet ‹<inform›. Der findes et specielt GIER ALGOL 4 program, INF1, som køres hver morgen af operatøren til indsættelse af den nye dato. De øvrige cellers indhold rettes normalt ikke.

Det bemærkes, at lagringen af datoen i omraadet ‹<inform› blev indført hos Haldor Topsøe paa et tidspunkt, hvor GIER ALGOL 4 lige var taget i brug men hvor det nye hjælpeprogramsystem, HELP3, endnu ikke var udkommet. I HELP 3 er der et særligt diskomraade, som hedder ‹<date›, og som indeholder datoen. Brugen af dette er dog endnu ikke indført i INPUT1.

3.4.4. Tegntyper i INPUT1. INPUT1 har brug for at undersøge de indlæste tegn fra inputmediet for at kunne udføre forskellige operationer alt efter hvilke tegn, den har læst. For at gøre programmeringen heraf lettere og mere overskuelig, er der indført en særlig metode, som er yderst velegnet til denne og lignende opgaver.

Alle tegn inddeles i det nødvendige antal klasser, her 8:

Type 1: SPACE, CAR RET, samt alle synlige og usynlige tegn, som ikke tilhører nogle af de følgende typer.

Type 2: Case tegn: lower case og upper case.
Type 3: e
Type 4: d
Type 5: q
Type 6: z
Type 7: Cifre og andre mulige talelementer.
Type 8: Komma og $_{10}$.

Denne typeinddeling skelner ikke mellem upper og lower case.

Naar et tegn er læst fra strimlen, sker typebestemmelsen ved et opslag i et lokalt katalog: boolean array TY[0:7]. Hver af disse 8 celler er inddelt i 8 dele, hver med 5 bits. Typeværdierne for tegnene fra 0-63 er da lagret saaledes:

Bits:	0-4	5-9	10-14	15-19	20-24	25-29	30-34	35-39
TY[0]	7	6	5	4	3	2	1	0
TY[1]	15	14	13	12	11	10	9	8
.....								
.....								
TY[7]	63	62	61	60	59	58	57	56

Kataloget genereres i begyndelsen af INPUT1 ved 8 sætninger af formen:

```
TY[0] := 5757575757575751;
```

Vi har tidligere side 32 set et eksempel paa den specielle GIER ALGOL 4 metode til sammenpakning af bitmønstre i logiske variable. I nærværende tilfælde har vi 8 understregede 5-taller, hvert efterfulgt af et heltal. Skrivemaaden:

```
57
```

betyder 5 bits, opfyldt med tallet 7 i totalsystemet, altsaa:

```
00111
```

Bitmønstret opbygges fra venstre, saaledes at det første 5 udfylder bits 0-4, det næste 5 bits 5-9, o.s.v. Det sidste 5 efterfulgt af 1-tallet bevirker derfor opfyldning af:

00001

i bits 35-39.

Er værdien af det indlæste tegn: ch, finder vi typen af ch ved et katalogopslag saaledes. Først skal vi finde ud af, hvilken af de 8 TY-celler, der skal bruges. Man kunne tænke sig at skrive:

```
TY[ch:8]
```

men dette kræver en division, der er ret langsom. Endvidere vil ch=64 give TY[8], som slet ikke findes. Skriver vi derimod:

```
TY[integer((boolean ch shift-3)↑40 7)]
```

faar vi dels en meget hurtigere beregning, fordi ch shift-3 direkte svarer til division med 8, dels vil alle tegn ≥ 64 paa grund af den logiske multiplikation med 40 7 blive bragt indenfor katalogets omraade.

Kaldes det valgte TY-index efter formlen ovenfor for IN, faas den endelige typebestemmelse som:

```
ty := integer((TY[IN]shift-integer(boolean ch↑40 7)×5)↑40 15);
```

Her giver boolean ch↑40 7 et resultat, der er ækvivalent med ch mod 8, men er hurtigere at lave. Det fremkomne tal ganges med 5 og dette giver os antallet af højreskift, som TY[IN] skal skiftes. Typeværdien staar da i bits 35-39 og fremmede bits fjernes ved logisk multiplikation med 40 15. Til sidst bevirker det foranstillede integer, at ty opfattes med korrekt type.

3.4.5. Proceduren T i INPUT1. Denne lokale procedure i INPUT1 svarer omtrent til den tidligere procedure CENTEXT, men i forbedret form. Proceduren læser fra inputmediet og leder efter de to tegn:

```
[ og e
```

Hvis parentesen mødes, læses resten af denne tekstlinie, indtil CAR RET mødes. De læste tegn pakkes med 4 tegn i hver celle af det lokale array DATA[0:39], og teksten trykkes igen centreret paa siden. Proceduren T er af typen boolean og sættes til sand, hvis der er mødt tekst og til falsk, hvis der er mødt et e. Der hoppes ud fra T, saa snart der er behandlet een

tekstlinie eller et e.

3.4.6. Forsidetrykning med INPUT1. De indledende manøvrer i INPUT1 bestaar først og fremmest i hentning af informationsomraadet {<inform} til det lokale array DATA[0:39] og ekstraktion af INF, cmax og lmax fra dette omraade. Desuden vælges input og outputmedium ved hjælp af:

```
select(integer(boolean DATA[4]↑boolean DATA[5]));
```

Bemærk den udstrakte anvendelse af typekontrolundertrykningen i denne procedure.

Beregningsnummeret indlæses eller sættes lig nul, hvis type siger, at det ikke skal indlæses. Det gemmes foreløbigt i den lokale variable, c.

Derefter følger forsidetrykning, hvis dette ønskes. Udtrykket:

```
t shift 2
```

er sandt, hvis bit 2 i type er sand, d.v.s. 1. Den variable boolean t er en lokal variabel, som er sat lig type i begyndelsen af beregningen. Forsidetrykningen indeholder følgende elementer:

1. Trykning af 50 TAPE FEED.
2. Trykning af tegnet 72, der paa linieskriveren giver det saakaldte top-of-form, d.v.s. skift til næste side efter en speciel styring, som er indbygget i linieskriveren.
3. Trykning af navnet: HALDOR TOPSØE.
4. Trykning af cmax-31 SPACE.
5. Trykning af maanedsnavn, dato og aar. Disse tal ekstraheres fra celle 0 i informationskanalen ved passende skiftoperationer og logiske multiplikationer.
6. Trykning af lmax:9 CAR RET.
7. Hvis sagsnummeret skal indlæses, trykkes ordet: File No. og sagsnummeret kopieres.
8. Trykning af lmax:4 CAR RET.
9. Trykning af den eller de variable forsidetekstlinier. Dette sker med den beskedent udseende for-sætning:

```
for i := 1 while T do LINE(2);
```

Vi husker, at den logiske procedure, T, er sand hver gang den har læst

en tekststreng, men falsk, saa snart stop-eet er mødt. Hver tekststreng, som T læser og trykker centreret, afsluttes derfor med 2 CAR RET.

10. Trykning af GIER Calculation No. efterfulgt af beregningsnummeret.

11. Trykning af de to tekststrenge, name1 og name2.

12. Skift til næste side.

Skal forsiden ikke trykkes, undersøges om det er første kald af proceduren i programmet (t shift 6). Er dette tilfældet, er der aabenbart tale om et specialprogram, som ikke begynder med en forside. Sagsnummeret overspringes, hvis det skal læses (t shift 1), og der indsættes startværdier af pageno og LRest.

3.4.7. Arealreservering i INPUT1. Er denne reservering bestilt via (t shift 5), sker følgende.

Proceduren bruger et besynderligt navn for det areal, som skal reserveres, nemlig:

```
{<xyz1zyx}
```

Navnet er valgt besynderligt, for at det skal være usandsynligt, at andre brugere kan have valgt det samme navn for et areal, som skal bevares fra een beregning til en anden.

Først slettes arealet af kataloget med:

```
cancel({<xyz1zyx});
```

Dette er nødvendigt, da man ellers ikke kan faa reserveret netop det ønskede antal kanaler. Reserveringen sker med:

```
if reserve({<xyz1zyx}, integer (t↖20 0 20 m)) ≠ 0 then
```

Vi husker, at talværdien af reserve er større end nul efter kaldet, hvis reserveringen ikke er i orden. Det ønskede antal kanaler skrives som:

```
integer (t↖20 0 20 m)
```

hvor vi faar en logisk multiplikation af t med en maske opbygget af 20 nuller i bits 0-19 og 20 1-taller i bits 20-39.

Er reservationen mislykket, faas fejludskriften:

Calculation not possible

paa fejlmediet, der normalt vil være skrivemaskinen. Da INPUT1 ikke indeholder nogen fejludhopsetikette, som den kan hoppe til, gentages fejludskriften, indtil operatøren opdager det og afbryder beregningen.

Er reservationen i orden, beregnes arealordet med:

```
where({<xyz1zyx}, area);
```

3.4.8. Indlæsning af datagrupper i INPUT1. Hovedprincippet er, at hver datagruppe indlæses til en ny kanal med første tal i celle nul. Indlæsningen vil normalt ske til kanalerne:

```
track
track + 1
track + 2
o.s.v.
```

altsaa een datagruppe paa hver kanal, men er en datagruppe større end 40 elementer, faar den lov at bruge to eller flere kanaler.

Vi gennemgaar ikke alle detailler i denne del af proceduren, men de vigtigste dele er følgende.

I princippet indlæses hvert tegn med sætningen:

```
ch := lyn;
```

og der foregaar derefter en typebestemmelse ved et katalogopslag, som forklaret i afsnit 3.4.4. Naar typen, ty, er bestemt, gaar vi ind i en stor case-sætning af formen:

```
case ty of
begin
   ....
```

De otte mulige typer giver anledning til følgende otte mulige behandlinger:

Type 1. Ligegyldigt tegn. Læs næste tegn.

Type 2. Case-tegn. Sæt den lokale variable, case, til 0 i lower case og 128 i upper case. Læs næste tegn.

Type 3. Stop-e. Gem arrayet DATA paa den aktuelle kanal. Hvis flere datagrupper skal indlæses, gøres klar hertil.

Type 4. Ditto-d. Øg tællecellen, no, med 1.

Type 5. q. Øg tællecellen med read integer. Dette er ækvivalent med q gange d.

Type 6. Stop-z. Hop til etiketten ZZ i slutningen af proceduren.

Type 7. Cifre eller andre talelementer. Hvis det indlæste tegn er plus, hoppes straks til læsning af næste tegn. Ellers sættes char lig med ch + case, og tallet indlæses med:

```
r := read real;
```

Tællecellen, no, øges med 1 og en logisk variabel, numb, sættes til sand. Ved behandling af d og q sættes numb til falsk.

Type 8. Specialbehandling af komma (der er blindt) og 10-potens, der giver hop til type 7: talindlæsning.

Efter typeanalysen fortsættes med indlæsning af næste tal og eventuelt overføring til disken.

Ved etiketten ZZ sker den afsluttende behandling af parametrene som omtalt i afsnit 3.4.1.

3.4.9. Simpel anvendelse af INPUT1. Som et eksempel paa en meget simpel brug af INPUT1 viser vi følgende blok fra et program, hvori man ønsker forsidetrykning og indlæsning af 5 inputdatagrupper (med højst 40 elementer i hver gruppe).

```
begin
   boolean INF, type;
   integer AREA, calcno, pageno, LRest;
   copy LINE<
   .......
   .......
   type := 7 m 33 50;
AA:begin comment simple input block;
      integer track, total, item;
      copy INPUT1<
      track := 1;
      total := 5;
      item := 40;
```

```
        if INPUT1 (type, track, AREA, total, item,
        {<Calculation of XYZ System}, 25,
        {<GIER Program XYZ-1}, 18, 1) = 0 then go to ZZ;
    end input;
    .....
    .....
ZZ:end program;
```

Bemærk, at de variable: INF, type, AREA, calcno, pageno og LRest maa være deklareret i den yderste blok. Det samme gælder proceduren: LINE.

Før vi gaar ind i inputblokken, tildeles type den ønskede værdi:

```
type := 7 m 33 50;
```

Herved sættes bits 0-6 lig med 1 og antallet af kanaler, der skal reserveres, til 50. I selve inputblokken indsættes værdierne:

```
track := 1;     Læsning til kanal 1 i AREA.
total := 5;     Læsning af 5 datagrupper.
item := 40;     Højst 40 elementer i hver gruppe.
```

Normalt arbejder INPUT1 ikke med mere end 1 sektion, men ønskes flere, maa programmet indeholde et returhop til AA. En mere avanceret brug af INPUT1 er vist i vejledningen for denne procedure.

3.4.10. Brug af INPUT1 med mærkning. Ved ovennævnte brug af INPUT1 i programmet med forside og 5 datagrupper kan man godt have flere sektioner, men man faar ingen mærkning af de tal, som er ændret i de senere sektioner, saaledes som det skete i READ6 i ALGOL II og III.

Det ville ikke være retfærdigt at sige, at mærkningsmuligheden mangler i INPUT1, for hensigten var den, at vi skulle komme frem til, at den information om datavariation, som hidtil er kommet ind i maskinen i sektion 2, 3, o.s.v. ligesaagodt kunne komme ind i maskinen som en ekstra datagruppe i sektion 1. For at dette kan gennemføres fornuftigt, maa der foretages visse ændringer i de procedurer, der trykker datagruppen igen, som beskrevet i næste afsnit. Da disse ændringer ikke er indført endnu, kan man kun bruge INPUT1 med datamærkning ved at foretage særlige krumspring. Et eksempel herpaa er vist i følgende inputblok, taget fra program TD-5.

```
   sectno := 0;
AA:begin comment input;
      boolean type, cell;
      integer track, total, TDATA, group, no, sum, item;
      real value;
      copy INPUT1 <
      copy CHECK <
      sectno := sectno + 1;
      if sectno = 1 then
      begin
         where(‹<free›, FREE);
         type := 5m21330;
         track := 2;
         total := 5;
         item := 40;
         INPUT1(type, track, FREE, total, item,
         ‹<Calculation of Thermodynamic Properties of Gases›, 48,
         ‹<GIER ALGOL 4 Program TD-5›, 25, 1);
         get(DATA, FREE, 2);
         REAC := DATA[1];
         k := if REAC = 0 then 1 else REAC;
         type := 53350;
         for i := 1 step 1 until k do
         begin
            total := 1;
            track := 6 + i;
            INPUT1(type, track, FREE, total, item,
            ‹<›, 0, ‹<›, 0, 1)
         end for i;
         for i := 6 + k step -1 until 2 do
         begin
            get(DATA, FREE, i);
            cell := false;
            for j := 0 step 1 until 38 do
            if DATA[j] ≠ 0 then
            cell := cell ∨ (11 shift - j);
            DATA[39] := real cell;
```

```
            put(DATA, FREE, i)
         end for i;
         if CHECK(‹<TDATA›) ≠ 0 then go to ZZ;
         where(‹<TDATA›, TDATA);
         get(DATA, TDATA, 1);
         tcalc := integer DATA[0];
         CMAX := (integer DATA[38]) - 2;
      end sectno = 1
      else
      begin
         for i := 2 step 1 until 21 do
         begin
            get(DATA, FREE, i);
            DATA[39] := 0;
            put(DATA, FREE, i)
         end for i;
         sum := 0;
L1:      group := read integer;
         if group > 0 then
         begin
            no := read integer;
            value := read real;
            sum := sum + 1;
            get(DATA, FREE, group);
            DATA[no] := value;
            cell := boolean DATA[39];
            cell := cell ∨ (1 1 shift -no);
            DATA[39] := real cell;
            put(DATA, FREE, group);
            go to L1
         end group > 0;
         if sum > 0 then
         begin
            PSHIFT(20);
            outsp(35);
            writetext(‹<SECTION›);
```

```
            if sectno < 10 then write(«-d», sectno)
            else
            if sectno < 100 then write(«-dd», sectno)
            else
            write(«-ddd», sectno);
            LINE(2)
          end
          else go to ZZ
        end sectno > 1
      end input;
```

Følgende variable og procedurer skal være deklareret i den yderste blok:

```
boolean INF;
integer LRest, calcno, sectno, pageno, FREE,
REAC, i, j, k;
array DATA[0:39];
copy INFORM<
copy LINE<
copy MESS<
```

I denne brug af INPUT1 er sektionsnummeret igen taget i brug, og det nulstilles før inputblokken mødes. Første sætning i inputblokken øger sectno med 1.

Hvis sectno derved bliver lig med 1, foregaar forsidelæsningen og -trykningen paa normal maade, omtrent som eksemplet ovenfor. Her ønsker vi direkte at bruge arealet «<free», og maa derfor ved et kald af where bestemme arealordet: FREE.

I programmet TD-5 har vi den komplikation, at antallet af datagrupper afhænger af antallet af kemiske reaktioner, REAC, i beregningen. Da REAC selv er et inputtal i den første datagruppe, er det praktisk at opdele indlæsningen i to dele. I første del indlæses forside og 5 datagrupper. I anden del indlæses kun REAC datagrupper, hvor der dog er den yderligere komplikation, at for REAC = 0 skal vi alligevel lade som om, at REAC er lig med 1.

Efter første del af indlæsningen maa vi derfor hente REAC frem ved:

```
get(DATA, FREE, 2);
REAC := DATA[1];
```

Derefter sker anden del af indlæsningen med REAC eller 1 kald af INPUT1 hvor parametrene nu er ændret til kun at give indlæsning af 1 datagruppe i hvert kald.

Efter dataindlæsningen udføres mærkningen af celle 39 af hver kanal paa samme maade som i READ6, idet vi dog nu kan gøre det lidt simplere ved brug af shift-operatoren. For hver af inputkanalerne hentes kanalen frem med:

```
get(DATA, FREE, i);
```

Derefter sættes den logiske variable, cell, til falsk, d.v.s. der anbringes nuller i alle bits. Saa udføres for-sætningen:

```
for j := 0 step 1 until 38 do
if DATA[j] ≠ 0 then
cell := cell∨(1 1 shift - j);
```

Hvis celle j paa kanalen er forskellig fra nul, indsættes en bit 1 i bit nr. j i cell. Bemærk, at udtrykket: 1 1 er en celle med et 1-tal i bit nul og resten nuller. I udtrykket:

```
1 1 shift-j
```

er 1-biten flyttet hen i bit j.

Til slut gemmes cell i celle 39:

```
DATA[39] := real cell;
```

og kanalen læses tilbage.

De fem sætninger med kaldet af CHECK er en kontrol af det termodynamiske materiale. Det omtales nærmere i afsnit 4.2.

Hvis sectno er større end 1, er vi aabenbart i een af de efterfølgende sektioner, og dataindlæsningen foregaar nu paa en særlig maade, uden anvendelse af bogstaverne d, e eller q.

Først nulstilles celle 39 paa alle inputkanalerne. Derefter begynder den egentlige indlæsning med at vi nulstiller en sumcelle: sum.

Hvert nyt dataelement skal nu indlæses som tre tal:

```
group := read integer;
no := read integer;
value := read real;
```

Her er group datagruppenummeret og no er det relative nummer paa den paagældende kanal ($0 \leq no \leq 39$). Endelig er value den egentlige talværdi af det nye tal. For hvert indlæst tal øges sum med 1. Desuden sættes DATA [no] lig med value paa kanal: group og bit: no i celle 39 paa denne kanal sættes til 1.

Hvis group = 0, er indlæsningen af data forbi. Hvis der overhovedet er indlæst data, trykkes ordet SECTION og værdien af sektionsnummeret, ellers er beregningen slut, og der hoppes til etiketten ZZ sidst i programmet.

3.4.11. Fremtidige inputprocedurer. Det er et karakteristisk træk i INPUT1, at denne procedure efterhaanden har overtaget funktionen af flere mindre procedurer, som den til gengæld gør paa en fiksere maade. Vore tanker om fremtidige inputprocedurer gaar i retning af at fortsætte denne udvikling, saaledes at inputproceduren udskilles som et helt selvstændigt program, der foretager al inputadministration. Programmøren skal da blot viderebehandle de data, som inputproceduren har overført til disken og ligeledes aflevere resultaterne paa disken. Man kan ogsaa tænke sig trykningen foretaget af et helt specielt program. Disse tanker omtales nærmere i kapitlet om GIPS-systemet.

3.5. Trykning af Datagrupper.

3.5.1. Procedurerne TABLE2 og T3. I dette afsnit omtales to procedurer i ALGOL II til trykning af inputmateriale med tekstforklaring.

Det vil altid være klogt at lade programmet trykke værdien af alle inputtal, for at man kan se, at maskinen har benyttet de rigtige tal. Vi kan f.eks. ønske at faa de to datagrupper side 55 trykt saaledes:

RØRDATA

Rørantal	100
Højde (m)	8.00
Udvendig diameter (mm)	110.00
Indvendig diameter (mm)	90.00

DRIFTSBETINGELSER

Gasmængde (kgmol/h)	1298.30
Temperatur (gr.C)	500
Tryk (atm.abs.)	23.00

Hvis talmaterialet er indlæst med READ6 til kanal 67 og 68, kan en blok til trykning af de to tabeller se saaledes ud:

```
    begin
       integer mcount, mfact;
       array DATA[0:39];
       comment library TABLE2;
       comment library T3;
       TABLE2(67, 10, NP1,
       {<RØRDATA}, 36);
       T3({<Rørantal}, 38, {-ndddd});
       T3({<Højde (m)}, 38, {-nddd.dd});
       T3({<Udvendig diameter (mm)}, 25, {-nddd.dd});
       T3({<Indvendig diameter (mm)}, 24, {-nddd.dd});
       LINE(2);
NP1:   TABLE2(68, 10, NP2,
       {<DRIFTSBETINGELSER}, 31);
       T3({<Gasmængde (kgmol/h)}, 25, {-ndddd00.000});
       T3({<Temperatur (gr.C)}, 30, {-nddd});
       T3({<Tryk (atm.abs.)}, 33, {-ndd.d0});
       LINE(2);
NP2: end blok;
```

Der bruges de to procedurer: TABLE2 og T3, som er deklareret nedenfor, og som iøvrigt kræver de ikke-lokale variable:

```
integer mcount, mfact;
array DATA[0:39];
```

Deklarationerne er:

```
procedure TABLE2(chan, freeline, noprint, headline, space);
integer chan, freeline, space;
label noprint;
string headline;
begin
```

```
   tromleplads := 40×chan + 39;
   fratromle(DATA);
   if DATA[39] = 0 then go to noprint;
   PSHIFT(freeline);
   trykml(space);
   tryktekst(headline);
   LINE(2);
   mcount := 0;
   mfact := 1
end of TABLE-2;

procedure T3(textline, sp, layout);
string textline, layout;
integer sp;
begin
   integer cell;
   cell := DATA[39];
   mfact := mfact×2;
   if (cell - (cell : mfact)×mfact)×2 - mfact ≥ 0
   then
   begin
      tryktekst(textline);
      trykml(sp);
      tryk(layout, DATA[mcount]);
      LINE(1)
   end;
   mcount := mcount + 1
end of T3;
```

Proceduren TABLE2 har de fem formelle parametre:

`integer chan;`	Datagruppens tromlekanalnummer.
`integer freeline;`	Hvis LRest < freeline, skiftes side inden tabeltrykningen.
`integer noprint;`	Der hoppes til denne etikette, hvis alle tal i datagruppen er uændret fra forrige sektion.
`string headline;`	Tabeloverskrift.
`integer space;`	Antal SPACE, der skal trykkes før overskriften.

I proceduren T3 har vi parametrene:

string textline;	Tekstforklaring til den variable.
integer sp;	Antal SPACE efter teksten.
string layout;	Layout til trykning af tallet.

I programmet kalder man først TABLE2, som overfører den paagældende tromlekanal til det ikke-lokale talsæt: DATA. Hvis DATA[39] = 0, skal tabellen slet ikke trykkes, og der hoppes straks til etiketten: noprint. Ellers udføres eventuelt sideskift, overskriften trykkes og der laves to CAR RET. Startværdier indsættes for mcount (0) og for mfact (1).

Derefter skal programmet indeholde et kald af T3 for hver af de variable i datagruppen. T3 ganger mfact med 2 og undersøger om DATA[39] indicerer trykning. Hvis ja, trykkes tekstlinien, antallet af SPACE og DATA [mcount] med det anvendte layout. Der laves ny linie med LINE(1). Hvis nej, udføres ingen trykning. T3 slutter med at øge mcount med 1.

Efter sidste T3-kald maa man skrive LINE(2) for at faa plads inden næste tabel.

Hvis der er specielle forhold ved trykningen af en af de variable, kan dette programmeres mellem de enkelte T3-kald.

TABLE2 og T3 findes ogsaa i ALGOL III.

3.5.2. ALGOL 4 versioner af TABLE2 og T3. I programmet TD-5, hvis specielle anvendelse af INPUT1 er omtalt i afsnit 3.4.10, bruges to særlige versioner af TABLE2 og T3, som er skrevet i ALGOL 4 og som kontrollerer mærkningen i celle 39. Disse versioner er ikke standardprocedurer og kan derfor ikke indsættes med copy.

De to proceduredeklarationer er:

```
procedure TABLE2(chan, freeline, noprint, headline, space);
integer chan, freeline, space;
label noprint;
string headline;
begin
   get(DATA, FREE, chan);
   if DATA[39] = 0 then go to noprint;
   for mcount := 0 step 1 until 38 do
   if DATA[mcount] ≠ 0 then go to L2;
   go to noprint;
```

```
L2:     PSHIFT(freeline);
        outsp(space);
        writetext(headline);
        LINE(2);
        mcount := 0;
    end of TABLE-2;


    procedure T3(textline, sp, layout);
    string textline;
    boolean layout;
    integer sp;
    begin
       boolean cell;
       cell := boolean DATA[39];
       print := cell shift mcount;
       if print then
       begin
          writetext(textline);
          outsp(sp);
          write(layout, DATA[mcount]);
          LINE(1)
       end;
       mcount := mcount + 1
    end of T3;
```

Parametrene i disse versioner af TABLE2 og T3 svarer ganske til parametrene i ALGOL II versionen. Virkemaaden er ogsaa meget nær identisk. Følgende smaating bør bemærkes:

I T3 er parameteren layout specificeret som string i ALGOL II, men i ALGOL 4 skal den være boolean. Den globale integer variable: mfact bruges ikke i ALGOL 4, da man her let kan se, om et tal skal trykkes, nemlig hvis:

```
cell shift mcount
```

er sand. I ALGOL 4 er der til gengæld indført en global parameter af typen boolean:

```
print := cell shift mcount;
```

Herved kan man altid se om det netop passerede T3-kald har været aktivt. Dette kan udnyttes til udskrift af specialtekst, som vist i følgende uddrag af trykprogrammet i TD-5.

```
TABLE2(2, 12, NP1,
{<CALCULATION PARAMETERS}, 29);
T3({<Number of components}, 36, {-ddd});
T3({<Number of reactions, not including special reactions}, 4, {-ddd});
T3({<Number of temperatures}, 33, {-dddd});
T3({<Start value of temperature, deg.C}, 22, {-dddd.dd});
T3({<Temperature increment, deg.C}, 27, {-dddd.dd});
T3({<Number of pressures}, 36, {-dddd});
T3({<Start value of pressure, atm.abs.}, 22, {-dddd.dd00});
T3({<Pressure increment, atm.abs.}, 27, {-dddd.dd00});
T3({<Water gas equilibrium approach, deg.C}, 18, {-dddd.dd});
T3({<Methane reforming equilibrium approach, deg.C}, 10, {-dddd.dd});
T3({<Ammonia equilibrium approach, deg.C}, 20, {-dddd.dd});
T3({<Methanol equilibrium approach, deg.C}, 19, {-dddd.dd});
T3({<Permissible error in equilibrium adjustment, mole per cent},
0, {-d.dd10-dd});
T3({<Viscosity calculation type}, 31, {-dd.d});
if print then
begin
   writetext(case vtype of ({<3: TDYN3}, {<3.1: TDYN3a}, {<21: TDYN21}));
   LINE(1)
end if print;
T3({<Diffusion calculation type}, 31, {-dd.d});
if print then
begin
   writetext(case dtype of ({<4: TDYN4}, {<10: TDYN10}, {<15.1: TDYN15a}));
   LINE(1)
end if print;
LINE(2);
NP1: TABLE2(3, 12, NP2,
{<LOGICAL CALCULATION PARAMETERS}, 25);
writetext({<0: no, 1:yes});
LINE(2);
T3({<Calculate table of enthalpies}, 30, {d});
T3({<Calculate table of specific heats}, 26, {d});
T3({<Calculate table of viscosities}, 29, {d});
```

3.5.3. ALGOL 4 procedurerne T1 og T2. Som sidste version af procedurerne TABLE2 og T3 viser vi de to procedurer T1 og T2, som er standardprocedurer i vort ALGOL 4 system og som kan indsættes i et program med copy.

Deklarationen af de to procedurer er:

```
boolean procedure T1(text, t, a, min, item, s);
value t, a, min, item, s;
boolean s;
integer t, a, min, item;
string text;
begin
   integer i, j;
   skip := s;
   area := a;
   track := t;
   if skip then
   begin
      j := 0;
L2:   get(DATA, area, t);
      for i := 0 step 1 until 39 do
      begin
         if DATA[i] ≠ 0 then go to L1;
         j := j + 1;
         if j = item then
         begin
            track := t + 1;
            T1 := false;
            go to L3
         end
      end;
      t := t + 1;
      go to L2
   end if skip;
L1: get(DATA, area, track);
   track := track + 1;
   count := 0;
   if min > LRest then LINE(min);
   writetext(text);
   LINE(1);
   T1 := true;
L3: end T1;
```

```
boolean procedure T2(lay, text);
boolean lay;
string text;
begin
   if count = 40 then
   begin
      count := 0;
      get(DATA, area, track);
      track := track + 1
   end;
   r := DATA[count];
   if r ≠ 0 ∨ -, skip then
   begin
      LINE(1);
      write(lay, r);
      writetext(text);
      T2 := true
   end else T2 := false;
   count := count + 1
end T2;
```

Følgende globale variable og procedurer skal være deklareret i den blok hvor T1 og T2 bruges, eller i en ydre blok:

```
procedure LINE;
integer LRest, calcno, pageno;
boolean INF;
```

Disse bruges af LINE. Følgende bruges direkte af T1 og T2:

`boolean skip;` Værdien af skip sættes af T1. Hvis skip er sand, overspringes trykning af tal og tekst af T2, for de tal, som er nul.

`integer area;` Arealord for dataområdet.

`integer track;` Sættes af T1 og peger hele tiden paa den efterfølgende kanal i området. Da datagrupperne kan være større end 40 celler, kan track ogsaa øges af T2.

`integer count;` Tæller som den tidligere mcount.

`real r;` Indeholder værdien af det aktuelle tal efter hvert T2-kald.

`array DATA[0:39];` Indeholder den aktuelle datakanal.

Bemærk, at T1 og T2 ikke bruger celle 39 til mærkning af de enkelte tal, som skal trykkes, saaledes som de tidligere versioner gjorde. Med lidt snedighed kan man dog godt bruge mærkningen alligevel, som vist senere.

De formelle parametre i T1 er:

string text; Tekststreng for tabeloverskriften. Den skal være centreret, da antallet af bogstaver ikke længere er en parameter.
integer t; Første relative kanalnummer i datagruppen.
integer a; Arealord for dataomraadet.
integer min; Hvis min>LRest, laves sideskift før tabeltrykningen.
integer item; Antallet af elementer i den aktuelle datagruppe.
boolean s; Værdien af skip sættes lig s.
boolean T1; Sand, hvis mindst eet tal er forskelligt fra nul.

De formelle parametre i T2 er:

boolean lay; Layout for trykning af tallet. Man bør indsætte SPACES forrest i layoutet, hvis disse ikke er ens for alle T2-kald.
string text; Tekststreng med forklaring til det trykte tal. T2 trykker først tallet, derefter teksten, og det kan derfor være nødvendigt at indsætte SPACES i begyndelsen af tekststrengen.
boolean T2; T2 er sand, naar den har trykt et tal og tekstlinien.

Følgende eksempel viser princippet i anvendelsen af T1 og T2:

```
begin comment printing;
   boolean skip, .....;
   integer area, track, count, ...;
   real r, ....;
   array DATA[0:39], ....;
   .
   .
   .
   if T1(t<                MECHANICAL DATA}, 1, FREE, 4, 2, true) then
   begin
      T2(t ddddd}, t<          Catalyst bed diameter (mm)});
      T2(t    -d.dddd}, t<     Fouling factor (sq.mhC/kcal)});
```

```
      LINE(2)
   end;
   if T1(‹<                    CATALYST DATA›, 2, FREE, 10, 47, true) then
   begin
      .
      .
      .
      .
      .
      .
   end;
   .
   .
   .
end printing;
```

Finessen med brug af den globale logiske variable, print, i ALGOL 4 versionen af T3 kan her gøres paa en endnu simplere maade:

```
   ......
   if T2(‹ dddd›, ‹<          Viscosity type›)then
   begin
      writetext(‹<                 ›);
      writetext(case vtype of
                (‹<TDYN3›,
                 ‹<TDYN3a›,
                 ‹<TDYN21›));
      LINE(1);
   end if T2;
   ......
```

Ønsker man at bruge T1 og T2 med mærkningsinformation i celle 39, kan dette gøres ved lige efter T1-kaldet at nulstille de celler i DATA, som ikke skal trykkes.

4. BEHANDLING AF KONSTANTE DATA

4.1. Behandling af konstante talsæt og tekststrenge i ALGOL II og III.

I bogen: Elementært ALGOL (Kjær(1968)), blev der i programmet til trykfaldsberegning side 110 vist et eksempel paa, hvorledes man kan indbygge talværdierne af et todimensionalt talsæt i et program. I det følgende giver vi flere eksempler paa, hvorledes dette blev gjort i ALGOL II og III. I det følgende afsnit vises de tilsvarende metoder i ALGOL 4, som er langt fiksere og derfor bør anbefales.

I mange af vore kemiske programmer findes indbygget data for ca. 70 forskellige grundstoffer og kemiske forbindelser. Ved brugen af programmerne skal man opgive antallet af komponenter, COMP, samt et talsæt:

```
integer array CN[1:COMP];
```

som indeholder identifikationsnumrene for komponenterne. I Kjær (1963d), side 11, er givet en liste over disse identifikationsnumre.

Skal der udføres entalpiberegninger i programmet, skal vi altsaa have opbygget talsættet:

```
array ELIST[1:COMP, 0:4];
```

Hertil anvendte vi i ALGOL II og III proceduren ENT3, hvis deklaration er vist nedenfor. Kun et udsnit af talmaterialet er vist.

```
procedure ENT3(COMP, CN, a);
value COMP;
integer COMP;
integer array CN;
array a;
begin
   integer i, j, k;
   procedure E(a4, a3, a2, a1, a0);
   real a4, a3, a2, a1, a0;
   begin
      for k := 1 step 1 until COMP do
```

```
            begin
               if j = CN[k] then
               begin
                  a[k,4] := a4;
                  a[k,3] := a3;
                  a[k,2] := a2;
                  a[k,1] := a1;
                  a[k,0] := a0;
                  i := i + 1;
                  if i > COMP then go to ex;
                  go to h
               end for if j
            end for k;
h:          j := j + 1
         end of E;
         i := 1;
         j := 0;
E( 8.47048660₁₀-11,-6.59715841₁₀ -7, 2.06310490₁₀ -3, 5.84681959,  -1909.88);
E(-2.93775001₁₀-11, 2.95560760₁₀ -7,-3.20003271₁₀ -4, 7.09122150,  -2093.47);
E(-1.29045340₁₀-10, 3.78724080₁₀ -7, 9.88421689₁₀ -4, 7.26073639, -60059.65);
E(-3.00946330₁₀-10, 1.01872460₁₀ -6,-5.08535561₁₀ -4, 6.97950259,  -2060.42);
E(-1.97268220₁₀-10, 5.37525999₁₀ -7, 2.74513310₁₀ -4, 6.74053200,  19553.15);
E(-2.43717910₁₀-10, 7.38602608₁₀ -7,-2.30511500₁₀ -5, 6.73251480, -28438.67);
E( 5.35761499₁₀-10,-3.01635810₁₀ -6, 7.26509499₁₀ -3, 5.37533410, -96224.66);
o.s.v.
E(-3.41621170₁₀-11,-1.31749150₁₀ -6, 7.37633910₁₀ -3, 4.10767169, -29545.30);
E( 1.33325600₁₀ -9,-8.50179080₁₀ -6, 2.48988169₁₀ -2, 4.96751820, -59719.79);
E(-9.60228883₁₀-10, 1.01140680₁₀ -7, 1.00993700₁₀ -2, 4.81536999, -50428.67);
E( 3.32102010₁₀ -9,-1.47626830₁₀ -5, 3.11525780₁₀ -2,-3.74136710, -13478.86);
E( 3.14402500₁₀ -9,-1.66387121₁₀ -5, 3.86750279₁₀ -2, 0.52722735,  23570.80);
E( 7.83491010₁₀-10,-3.76899430₁₀ -6, 7.38529449₁₀ -3,-1.44499870,   -132.00);
E( 0               ,-4.85670000₁₀ -6, 2.24450000₁₀ -2, 3.69300000, -46027.72);
ex:  end of ENT-3;
```

Proceduren ENT3 virker paa samme maade, som blev vist ved D-proceduren i ovennævnte program. Her er der indbygget en E-procedure med fem parametre i ENT3. E kaldes en mængde gange, idet der hver gang tælles frem i j:

```
j := j + 1;
```

Hvis j er lig med et af de foreliggende identifikationsnumre:

```
j = CN[k]
```

indsættes de fem tal paa dette sted i entalpitalsættet (formel parameter: a). Det aktuelle kald af proceduren kan se saaledes ud:

```
begin
   integer COMP;
   ........
   ........
   begin
      integer array CN[1:COMP];
      array ELIST[1:C MP, 0:4];
      comment library ENT3;
      ........
      ........
      ENT3(COMP, CN, ELIST);
      ........
    end;
    ........
end;
```

Brugen af proceduren ENT3 er en forholdsvis kostbar maade at faa det ønskede talsæt samlet paa. For hvert kald af proceduren E skal der foruden de fem talkonstanter bruges 3 celler i programmet til at etablere selve kaldet. Hvis der er daarlig plads i programmet, kan man undgaa at bruge disse ekstra celler paa følgende maade.

Vi indfører et stort talsæt paa 5×70 elementer indeholdende data for alle indbyggede komponenter:

```
array A[0:69, 0:4];
```

Talsættet deklareres dog ikke noget sted i programmet, idet vi kun ønsker at have det staaende paa tromlen:

Tromleplads:	Element
dplace	A[0,0]
dplace + 1	A[0,1]
dplace + 2	A[0,2]
dplace + 3	A[0,3]
dplace + 4	A[0,4]
dplace + 5	A[1,0]
o.s.v.	

Tromleplads for det første element for komponent nr. 0 er altsaa: dplace. Tænker vi os, at dette talsæt paa en eller anden maade er blevet anbragt paa tromlen, kan vi med proceduren TDYN12 opnaa samme virkning som ved brugen af ENT3 ovenfor. Deklarationen af TDYN12 er:

```
procedure TDYN12(COMP, first, last, dplace, CN, a);
value COMP, first, last, dplace;
integer COMP, first, last, dplace;
integer array CN;
array a;
begin
   integer i, j;
   array R[first:last];
   for i := 1 step 1 until COMP do
   begin
      tromleplads := dplace  +  (last - first + 1)×(CN[i] + 1) - 1;
      fratromle(R);
      for j := first step 1 until last do a[i, j] := R[j]
   end for i
end TDYN12;
```

Kaldet af TDYN12 kan se saaledes ud:

```
begin
   integer COMP, dplace;
   ........
   ........
   begin
      integer array CN[1:COMP];
      array ELIST[1:COMP, 0:4];
      comment library TDYN12;
      ........
      ........
      TDYN12(COMP, 0, 4, dplace, CN, ELIST);
      ........
    end;
    ........
end;
```

TDYN12 gennemsøger CN-listen, og den vil for hver komponent overføre et lille talsæt R[first:last], som derefter bruges til indsættelse af værdierne af a-talsættet (her ELIST).

Det store talsæt paa tromlen kan naturligvis bringes paa plads ved kombineret indlæsning og tromletransport. For at undgaa denne specielle indlæsning i ALGOL, har vi i nogle programmer gjort det, at vi efter oversættelsen af ALGOL-programmet har taget strimlen med det tilsvarende kondenserede program, som oversætteren har afleveret, undersøgt startværdien af tromleplads (kanal 39, celle 30) og indlæst det store talsæt i maskinsprog, saaledes at det kommer til at ligge i den frie ende af tromlen. Man skal da blot huske at programmere en lavere værdi af startværdien af tromleplads i selve programmet.

Trykning af komponentnavne i ALGOL II og III i programmer med f.eks. 70 indbyggede komponenter kan ikke udføres umiddelbart, fordi man ikke kan operere med variable af typen: string og slet ikke med et string array. Hvis man ønsker at bruge korrekt ALGOL, kan problemet løses ved brug af proceduren PRINT7, hvis deklaration er (kun nogle af navnene er medtaget):

```
      procedure PRINT7(COMP, count, CN, PR);
      value COMP;
      integer COMP, count;
      integer array CN;
      procedure PR;
      begin
         integer j, k;
         procedure T (s);
         string s;
         begin
            for count := 1 step 1 until COMP do
            begin
               if j = CN[count] then
               begin
                  tryktekst (s);
                  PR;
                  LINE(1);
                  k := k + 1;
                  if k > COMP then go to ex;
                  go to h
               end for if j
            end for count;
h:          j := j + 1
         end of T;
         k := 1;
         j := 0;
T(‡<Oxygen                ‡);
T(‡<Hydrogen              ‡);
T(‡<Water                 ‡);
T(‡<Nitrogen              ‡);
T(‡<Nitric oxide          ‡);
T(‡<Carbon monoxide       ‡);
T(‡<Carbon dioxide        ‡);
T(‡<Argon                 ‡);
T(‡<Methane               ‡);
o.s.v.
```

```
T({<Formaldehyde          });
T({<Ethanol               });
T({<Methanol              });
T({<Ethylene oxide        });
T({<1,3-Butadiene         });
T({<Carbon                });
T({<Dimethyl ether        });
ex:  end of PRINT-7;
```

Et eksempel paa brug af PRINT7 er:

```
begin
   integer LRest, calcno, sectno, pageno, COMP, count;
   comment library LINE;
   ........
   ........
   begin
      integer array CN[1:COMP];
      array GAS[1:COMP];
      comment library PRINT7;
      procedure PP;
      tryk({-nddd.0000}, GAS[count]);
      ........
      ........
      PRINT7(COMP, count, CN, PP);
      ........
   end
end;
```

PRINT7 skal have adgang til komponentantallet: COMP, en tællevariabel: count, komponentnummerlisten: CN og til en speciel procedure: PR, som maa leveres af programmøren i hvert tilfælde. PRINT7 har en intern procedure: T, som kaldes mange gange, idet der hver gang tælles frem i j:

```
j := j + 1;
```

For hvert kald undersøges det, om:

```
j = CN[count]
```

for count = 1, 2, ... COMP. Er betingelsen opfyldt, d.v.s. j er lig med identifikationsnummeret for en af komponenterne, da vil PRINT7 trykke navnet paa denne komponent, idet denne streng staar som aktuel parameter til T. Altsaa for j = 6:

```
T('<Carbon dioxide          ');
```

Naar navnet er trykt, udfører PRINT7 proceduren PR, altsaa her PP, som er deklareret til at give trykning af GAS[count]. Derefter leverer PRINT7 trykning af 1 CAR RET med proceduren LINE, som altsaa maa findes i programmet.

Der er en lille ulempe ved denne trykkemetode, nemlig den at komponenterne bliver trykt efter stigende identifikationsnummer, altsaa en vis sortering.

Hvis en af komponenterne har et højt identifikationsnummer, gaar der et par sekunder for PRINT7, før den kommer frem til det tilsvarende T-kald. Dette spild kunne have været undgaaet, hvis man havde forsynet PRINT7 med en stor switch eller havde fundet frem til den rigtige tekststreng med en lang betingelsessætning. Proceduren ville dog derved have fyldt mere. I den nuværende form koster hvert T-kald 4 celler (+ 4 celler til selve teksten).

4.2. Behandling af Termodynamiske Data i ALGOL 4.

I ALGOL 4 systemet hos Haldor Topsøe er de termodynamiske data for ca. 80 stoffer lagret permanent paa et omraade af disken ved navn '<TDATA'. Hvert stof har en hel kanal, og indenfor denne er de enkelte data lagret saaledes:

Celle	Symbol	Forklaring
0	number	Formelnummer, d.v.s. et tal ud fra hvilket stoffets formel kan ekstraheres. Se nærmere herom i beskrivelsen til proceduren MCOMP.
1	MW	Molvægt
2	F25	Fri dannelsesenergi ved 25 gr. C (kcal/kgmol)

3	Tc	Kritisk temperatur, gr. K.
4	Pc	Kritisk tryk, atm. abs.
5	myc	Kritisk viskositet, kg/mh
6	kc	Kritisk varmeledningsevne, kcal/mhC.
7	CRMVOL	Kubikroden af molarrumfanget i liter pr. kgmol.
8	BOIL	Kogepunkt i gr. K.
9	omega	Acentricitetsfaktor
10	eps	Lennard-Jones konstant nr. 1
11	sigma	- - - - 2
12-15		Ledig
16-20	BB[1:5]	Beattie-Bridgeman-Koefficienter
21-28	BWR[1:8]	Benedict-Webb-Rubin -
29-34		Ledig
35-39	a[0:4]	Entalpikoefficienter.

Data for en komponent med komponentnummer: CN er lagret paa den relative kanal nr. 2+CN i omraadet {<TDATA}. Data for ilt (nr. 0) er altsaa lagret paa kanal nr. 2. Paa kanal nr. 1 er lagret forskellige administrative oplysninger:

Celle	Indhold
0	calcno: Beregningsnummer for sidste TD-2-beregning.
1-37	Ledig
38	2 + antallet af komponenter, for hvilke der er lagret data i {<TDATA}.
39	Checksum af {<TDATA}-omraadet.

Administration og vedligeholdelse af omraadet {<TDATA} udføres af programmet TD-2, og der henvises til vejledningen for dette program for nærmere detailler.

4.2.1. Proceduren FTLIST. Har man brug for en enkelt termodynamisk egenskab af et enkelt stof, kan man let programmere hentning af dette saaledes:

```
begin
   integer TDATA;
   real Tc;
   array DATA[0:39];
```

```
        where(f<TDATA>, TDATA);
        get (DATA, TDATA, 2 + 10);
        Tc := DATA[3];
    end;
```

Dette program henter egenskab nr. 3 (den kritiske temperatur) for komponent nr. 10 (propan).

Ved beregning af konvertere eller andre typiske kemiske beregninger har man brug for at ekstrahere næsten alle termodynamiske data for de COMP komponenter, der bruges i den foreliggende beregning. Dette kan gøres med proceduren FTLIST, der har følgende deklaration:

```
procedure FTLIST(COMP, CN, t);
value COMP, t;
integer COMP, t;
integer array CN;
begin
   integer i, j, k, TDATA;
   integer array d[0:39];
   array D[0:39], A[1:16, 1:COMP];
   where(f<TDATA>, TDATA);
   for i := 1 step 1 until COMP do
   begin
      get(D, TDATA, 2+CN[i]);
      d[i-1] := integer D[0];
      k := 0;
      for j := 1 step 1 until 11, 35 step 1 until 39 do
      begin
         k := k+1;
         A[k,i] := D[j]
      end for j
   end for i;
   put(d, FREE, t);
   for i := 1 step 1 until 16 do
   begin
      for j := 1 step 1 until COMP do D[j-1] := A[i,j];
      put(D, FREE, t+i)
   end for i
end FTLIST;
```

Proceduren har de tre formelle parametre:

integer COMP: Antallet af komponenter.

integer array CN[1:COMP]: En liste over identifikationsnumrene for de COMP komponenter.

integer t: De ekstraherede termodynamiske data vil blive lagret paa disk-kanal t til t + 16 i omraadet FREE, hvor FREE er en global heltalsvariabel, der skal være beregnet før kaldet af FTLIST.

De ekstraherede data lagres saaledes paa kanalerne:

Kanal	Array
t+0	number[1:COMP]
t+1	MW[1:COMP]
t+2	F25[1:COMP]
t+3	Tc[1:COMP]
t+4	Pc[1:COMP]
t+5	myc[1:COMP]
t+6	kc[1:COMP]
t+7	CRMVOL[1:COMP]
t+8	BOIL[1:COMP]
t+9	omega[1:COMP]
t+10	eps[1:COMP]
t+11	sigma[1:COMP]
t+12	a0[1:COMP]
t+13	a1[1:COMP]
t+14	a2[1:COMP]
t+15	a3[1:COMP]
t+16	a4[1:COMP]

Elementerne i disse arrays anbringes fra celle 0 til COMP-1 paa hver kanal. Dette er en noget ødsel placering, men man har jo lov til at lagre dem tættere bagefter.

4.2.2. Proceduren PNAME. Denne ALGOL 4 procedure erstatter den tidligere PRINT7 og har følgende deklaration (kun noget af den er vist):

```
procedure PNAME(n);
value n;
integer n;
begin
   integer i, j;
   i := 1 + n:3;
   j := 1 + (n mod 3);
   case i of
   begin
      writetext(case j of(
      {<Oxygen} ,{<Hydrogen} ,{<Water}));
      writetext(case j of(
      {<Nitrogen} ,{<Nitric oxide} ,{<Carbon monoxide}));
      writetext(case j of(
      {<Carbon dioxide} ,{<Argon} ,{<Methane}));
      writetext(case j of(
      {<Ethane} ,{<Propane} ,{<n-Butane}));
      ........
      ........
      etc.
      ........
      writetext(case j of(
      {<Chloroethane}, {<Hydrogen chloride}, {<Chlorine}));
      writetext(case j of(
      {<Cyclopropane}, {<2,4-Dimethylpentane}, {<2,2,4-Trimethylpentane}));
   end case of
end PNAME;
```

Den formelle parameter, n, angiver komponentnummeret for den komponent, hvis navn skal trykkes. PNAME trykker ingen SPACE, som PRINT7 gjorde. Søjlen med navne skal derfor staa længst til højre i den ønskede tabel.

5. KONTROL AF DATAGRUPPER

5.1. Programstruktur.

For kemiske og lignende programmer af mellemstørrelsen har vi ofte anvendt følgende inddeling af programmet i blokke:

1. Indlæsning af data.
2. Kontrol af datagrupper.
3. Trykning af inputmateriale.
4. Beregning.
5. Trykning af resultater.

Erfaringen har lært os, at hvis man ikke forsyner sit program med en vis mængde kontrol af det indlæste talmateriale inden de egentlige beregninger paabegyndes, løber man stor risiko for at beregningen afspores paa grund af division med nul eller lignende. Det er derfor klogt at indrette programmet som vist ovenfor, hvor blok nr. 2 udfører en saadan kontrol af talmaterialet. Hvis denne kontrol afslører fejl, bør man sætte en global boolean til sand, trykke inputmaterialet i blok 3, og derefter gaa tilbage til blok 1 for indlæsning af nye data til næste sektion. Er der ingen fejl, fortsættes med blok 4 og 5 til beregning og trykning af resultater.

5.2. Kontrolprocedurer.

Det er klart, at man ikke ved datakontrol kan gardere sig fuldstændigt mod fejl. Smaa fejl kan ikke fanges, ej heller saadanne fejl, der først afsløres et stykke ind i de egentlige beregninger.

5.2.1. Standardproceduren CHECK. Denne standardprocedure hos Haldor Topsøe er lagret paa disken og inkluderes i et program ved at skrive copy CHECK<. Den kontrollerer, at et navngivet dataomraade paa disken ikke er blevet ødelagt. Deklarationen er:

```
integer procedure CHECK(list);
string list;
begin
   integer AREA, i, j, sum, track;
   integer array DATA[0:39];
```

```
   procedure M(t);
   value t;
   integer t;
   begin
      integer s;
      s := MESS(‹<ERROR, CHECK, ›, 7);
      writetext(list);
      writeinteger(‹ddd›, t);
      select(s);
      CHECK := t;
      go to ZZ
   end M;
   CHECK := 0;
   i := where(list, AREA);
   if i = 1 ∨ i = 3 then M(1);
   if get(DATA, AREA, 1) ≠ 0 then M(2);
   sum := 0;
   track := DATA[38];
   for j := 1 step 1 until track do
   begin
      if get(DATA, AREA, j) ≠ 0 then M(3);
      for i := 0 step 1 until 39 do
      sum := sum + DATA[i]
   end for j;
   if sum ≠ 0 then M(4);
ZZ: end CHECK;
```

Der er en enkelt formel parameter:

```
string list
```

Dette skal være navnet paa det disk-areal, som ønskes kontrolleret. Et kald af CHECK er vist paa side 81:

```
if CHECK(‹<TDATA›)≠ 0 then go to ZZ;
```

hvor man har ønsket at kontrollere arealet ‹<TDATA›. Er arealet i orden, er værdien af CHECK = 0.

CHECK forudsætter, at celle 38 paa kanal 1 i det undersøgte omraade indeholder antallet af kanaler, som skal kontrolleres. Det behøver altsaa ikke at være hele arealet. Endvidere er det en forudsætning, at summen af

alle celler paa alle de afprøvede kanaler er blevet reguleret til nul ved at indsætte minus checksummen i en vilkaarlig af disse celler.

CHECK bruger to globale procedurer:

MESS til fejludskrift.

INFORM til at fremskaffe den bit, der definerer fejludskrifts-mediet.

Deklarationerne er:

```
integer procedure MESS(text, t);
string text;
integer t;
begin
   MESS := select(integer(INFORM(t)));
   LINE(1);
   writetext(text)
end MESS;

boolean procedure INFORM(item);
value item;
integer item;
begin
   integer i, inf;
   boolean array B[0:39];
   i := where(«<inform», inf);
   if i = 1 ∨ i = 3 ∨ get(B, inf, 1) ≠ 0 then
   begin
      select(16);
      writecr;
      writetext(«<ERROR, INFORM»);
      go to ZZ
   end if error;
   INFORM := B[item]
end INFORM;
```

Procedurerne MESS og INFORM kan ogsaa inkluderes ved hjælp af copy. Brug af CHECK i den her nævnte form blev indført hos Haldor Topsøe før

HELP3 systemet udkom. Man kan nu, stort set, erstatte CHECK med hjælpeprogrammet: check i HELP3.

5.2.2. Specialproceduren CHECK. I det tidligere nævnte program TD-5 indeholder datakontrolblokken en normalt deklareret procedure, der desværre ogsaa har faaet navnet: CHECK. Den har deklarationen:

```
procedure CHECK(cond, text, n);
boolean cond;
string text;
integer n;
if -, cond then
begin
   integer s;
   if -, error then
   begin
      s := MESS({<Input error}, 7);
      select(s);
      error := true;
   end if first error;
   s := MESS(text, 7);
   write({-dddd}, n);
   select(s)
end CHECK;
```

De tre formelle parametre er:

boolean cond:	En betingelse, som skal være opfyldt af talmaterialet.
string text:	En tekststreng, som udskrives hvis betingelsen ikke er opfyldt.
integer n:	Et heltal, som ogsaa udskrives, hvis der er fejl.

Proceduren arbejder med en global boolean: error, der sættes til falsk i begyndelsen af kontrolblokken. Første gang der optræder en fejl, rettes den til sand, og der udskrives teksten: Input error. Denne CHECK-procedure bruger ogsaa den globale standardprocedure: MESS.

5.2.3. Specialproceduren DISPLAY. Deklarationen herfor er:

```
procedure DISPLAY(group);
value group;
integer group;
begin
   get(D, FREE, group);
   for i := 0 step 1 until 39 do
   d[i] := D[i] + 0.01
end DISPLAY;
```

Den formelle parameter, group, er nummeret paa den datagruppe, som vi nu ønsker at kontrollere. Proceduren arbejder paa de to arrays:

```
array D[0:39];
integer array d[0:39];
```

som er deklareret i kontrolblokken. Kanal nr. group overføres til D, og hver enkelt celle i D overføres derefter til den tilsvarende d-celle. Dette sker her med sætningen:

```
d[i] := D[i] + 0.01;
```

Additionen af 0.01 er strengt taget overflødig i ALGOL 4, fordi D[i] altid vil blive omformet korrekt til en integer med nødvendig afrunding. Det er en reminiscens fra ALGOL II og III, hvori sætningen:

```
d[i] := D[i];
```

ikke gav afrunding, fordi tal af typen integer blev lagret paa samme maade som reals.

Brugen af de to arrays, d og D, er praktisk, fordi nogle af inputtallene vil være heltal, men de indlæses alle som reals og staar paa disk-kanalerne som saadan.

5.2.4. Specialproceduren RANGE. Endelig kan det være praktisk med en hjælpeprocedure, som kontrollerer, om en variabel, x, ligger i omraadet: from $\leq$ x $\leq$ to. Dette gør proceduren RANGE:

```
boolean procedure RANGE(x, from, to);
value x, from, to;
real x, from, to;
RANGE := from ≤ x ∧ x ≤ to;
```

Den er sand, hvis betingelsen er opfyldt, ellers falsk.

5.3. Kontroleksempel.

Følgende uddrag af kontrolblokken i programmet TD-5 viser eksempler paa brugen af kontrolprocedurerne:

```
error := false;
CHECK(RANGE(COMP, 1, 16), {<number of components}, COMP);
CHECK(RANGE(REAC, 0, 16), {<number of reactions}, REAC);
CHECK(vtype=30∨vtype=31∨vtype=210,
{<viscosity calculation type}, vtype);
CHECK(dtype=40∨dtype=100∨dtype=151,
{<diffusion calculation type}, dtype);
DISPLAY(3);
for i := 0 step 1 until 18 do
CHECK(d[i] = 0 ∨ d[i] = 1,
{<logical parameter no.}, i);
DISPLAY(4);
for i := 1 step 1 until COMP do
CHECK(RANGE(d[i-1], 0, CMAX),
{<component number}, i - 1);
DISPLAY(5);
SUM := 0;
for i := COMP - 1 step -1 until 0 do
begin
   M := D[i];
   SUM := SUM + M;
   if (caltyp shift 5) then CHECK(RANGE(M, 0, 100),
   {<mole percentage of component no.}, i)
end for i;
if (caltyp shift 5) then CHECK(RANGE(SUM, 99.99, 100.01),
{<sum of mole percentages}, 0);
.......
etc.
```

De to første kald af CHECK kontrollerer:

$$1 \leq \text{COMP} \leq 16$$
$$0 \leq \text{REAC} \leq 16$$

Derefter kontrolleres de to andre globale variable: vtype og dtype. Efter kaldet:

```
DISPLAY(3);
```

kontrollerer vi, at items 0 til 18 i datagruppe 3 er opgivet som 0 eller 1.

Efter DISPLAY(4) checker vi, at de opgivne komponentnumre ligger i intervallet fra 0 til CMAX, hvor CMAX er det højeste komponentnummer, der for tiden findes i de termodynamiske data. Dette er tidligere ekstraheret fra celle 38 paa kanal 1 i {<TDATA}.

Med DISPLAY(5) faar vi hentet datagruppe 5 frem, som indeholder en gasanalyse. Det kontrolleres, at hver analyse ligger mellem 0 og 100, samt at summen ligger mellem 99.99 og 100.01.

6. GIPS-SYSTEMET

I løbet af de ti aar, der har været udført elektroniske beregninger hos Haldor Topsøe, er der sket en udvikling i omfanget og karakteren af de opgaver, der er behandlet. Der er gjort en lang række erfaringer, med hensyn til hvorledes programmerne bør indrettes, og kørslerne afvikles.

Det saakaldte GIPS-system, hvis navn er en forkortelse af: General Integrated Programming System, er opstaaet ved interne drøftelser i regnemaskineafdelingen hos Haldor Topsøe paa basis af et oplæg fra Preben Sørensen, og indeholder en lang række af de indvundne erfaringer og især ønsker om et mere effektivt programmeringssystem.

Det maa straks bemærkes, at GIPS-systemet ikke er en fuldbyrdet kendsgerning, kun ganske faa af elementerne i systemet er skabt. Størstedelen befinder sig kun i et forberedende stadium. Naar systemet engang er færdigt - eller i hvert fald mere færdigt end det er nu - vil der naturligvis fremkomme særlige beskrivelser af det. Men det er naturligt at afslutte denne bog med en kort omtale af systemets vigtigste elementer. Denne beskrivelse maa naturligvis blive ufuldstændig og især foreløbig. Meget vil sikkert blive ændret, efterhaanden som systemet udbygges i praksis.

Der er følgende hovedpunkter i GIPS-systemets struktur:

1. Fleksibel program- og datastruktur.
2. Brug af udvidede dataformater (databeskrivelser).
3. Formulering af datatransporter og andre sprogafhængige operationer som makro-order, der eventuelt kan underkastes automatisk oversættelse fra eet sprog til et andet.

Disse tre problemer omtales i de følgende afsnit, og der omtales til slut en primitiv implementering af dele af systemet.

6.1. Datafleksibilitet og Programfleksibilitet.

For at forklare, hvad der menes med fleksibilitet i strukturen af data og programmer, kan det især for begyndere i programmeringens kunst være nyttigt at betragte forskellen mellem ganske smaa programmer og større programmer.

Som et helt lille program kan vi tage eksemplet fra bogen om Elementært ALGOL (Kjær(1968)) paa side 25. Programmet beregner rumfanget, R, af en cylinder med højden, H, og diametren, D. Vi kan sige, at programmet opererer paa en datagruppe, bestaaende af de tre elementer:

H
D
R

Programmet bestaar af de tre faser:

1. Indlæsning af H og D.
2. Beregning af R.
3. Trykning af H, D og R.

Indlæsning af H og D foregaar med standardproceduren:

```
H := read real;
D := read real;
```

og analogt for trykningen af H, D og R med: write.

Som næste eksempel kan vi nævne TD-5-programmet, hvis inputblok er vist side 80 i nærværende bog. Her er det talmateriale, som skal indlæses, meget større, og det er praktisk at dele det op i saakaldte datagrupper. Hver datagruppe indeholder nogle variable, som paa naturlig maade hører sammen. Følgende datagrupper bruges i TD-5:

Datagruppe	Indhold
1	Forsidedata.
2	Styretal. 15 simple variable af typen integer og real: Antal komponenter, reaktioner, o.s.v.
3	Logiske parametre. 19 simple variable af typen: boolean.
4	Komponentnumre. Identifikationsnumrene: CN[1:COMP], hvor COMP er antallet af komponenter.
5	Gassammensætning: M[1:COMP].

6 Temperatur-approach, appr[1:REAC], for de REAC reaktioner, der kan forekomme.

7 Støkiometriske koefficienter for reaktionerne: s[1:REAC, 1:COMP].

Denne datastruktur er ganske typisk for programmer af mellemstørrelsen. Foruden forsidedata har vi to datagrupper med simple variable og fire datagrupper med arrays. Blandede datagrupper kan ogsaa let forekomme, saaledes at f.eks. de første elementer i datagruppen er simple variable og resten arrays.

Indlæsning af datagrupper sker med proceduren INPUT1 (eller den tidligere READ6), som lagrer hver datagruppe paa sin disk-kanal, evt. paa flere.

Programmet TD-5 er bygget til at operere paa netop disse syv datagrupper, som altid skal indlæses i den opgivne rækkefølge. Vi kan ikke faa programmet til at acceptere andre datagrupper med en anden kemisk betydning, og vi kan heller ikke faa programmet til at udføre andre operationer paa datagrupperne, end hvad det er bygget til at gøre. Man kan f.eks. ikke indlæse to gassammensætninger i datagruppe 5 og saa haabe, at programmet vil blande de to gasser inden det beregner ligevægtssammensætningen, eller hvad opgaven nu bestaar i. Dette gaar slet ikke: Maskinen vil altid opfatte den datagruppe, som følger efter gruppe 5, som gruppe 6, altsaa som et sæt temperatur-approacher.

6.1.1. Ældre, fleksible programmer. Hos Haldor Topsøe blev alle programmer i begyndelsen lavet uden nogen form for fleksibilitet, d.v.s. efter samme faste struktur som ovennævnte TD-5-program. Dette gælder ogsaa konverterprogrammerne, hvor næsten alle indeholder følgende datagrupper i inputmaterialet:

1. Forsidedata.
2. Mekaniske data for katalysatorafsnittet.
3. Katalysatordata.
4. Data for indgangsgas.
5. Mekaniske data for nedre varmeveksler.
6. Diverse beregningsparametre.

Denne faste struktur af datagrupperne gjorde det f.eks. umuligt at kombinere en konverterberegning med en loop-beregning til nøjagtig indstilling af stofbalancen i hele kredsløbet. Skulle dette regnes nøjagtigt, var man nødt til at udføre flere beregninger efter hinanden, skiftevis paa konverterprogrammet og paa loopprogrammet, og med manuel hulning af input til de nye beregninger.

Det første program med en beskeden fleksibilitet var program R-4 til beregning af varmebalancen i røggassektionen fra en rørreformer. Her kunne opereres med tre forskellige typer af afkølingsformer, hver defineret ved en datagruppe. Rækkefølgen af datagrupperne paa inputstrimlen skulle svare til den fysiske forekomst af afkølingerne, og det første element i disse datagrupper maatte derfor være et typenummer (1, 2, 3) som definerede, hvilken afkølingstype, der var tale om.

I ammoniakkredsløbsprogrammet, AL-5, var indført et delvist fleksibelt beregningsforløb, men et fast inputformat.

I rørreformerprogrammet, R-7, blev brugt en blandet struktur, idet selve rørreformeren blev beregnet ud fra en fast datagruppestruktur, medens apparaterne efter reformeren kunne staa i vilkaarlig rækkefølge paa inputstrimlen.

6.1.2. PLANT-1-programmet. Ved fremkomsten af dette program i 1965 fik man hos Haldor Topsøe det første program med fuld fleksibilitet. Programmet er i det væsentlige lavet af N. Mathiesen og beskrevet i særlige vejledninger. Da fleksibilitetsprincippet i PLANT-1-programmet er næsten identisk med det, som er planlagt for GIPS-systemet, er der ingen grund til at forklare det nærmere her. Vi bemærker blot, at hver datagruppe i PLANT-systemet indledes med to styretal:

1: FSNO, flow-sheet nummer
2. TYNO, typenummer

Her er FSNO et slags løbenummer for datagruppen, idet det svarer til et nummereret knudepunkt i beregningsskemaet. Dette skema kaldes ogsaa beregnings-flow-sheetet, idet det indeholder de samme elementer, som det egentlige proces-flow-sheet, samt nogle yderlige knudepunkter af beregningsmæssig natur (iterationer, tabelbehandling, o.s.v.).

TYNO er et typenummer, der definerer datagruppens betydning, f.eks.:

TYNO	Betydning.
150	Varmevekslerberegning.
205	Reformerberegning.
302	Deling eller blanding af strømme.

PLANT-programmet er internt opdelt i blokke, som hver behandler det tilsvarende typenummer. I den oprindelige version af PLANT-programmet angav flow-sheetnummeret, FSNO, direkte den rækkefølge, i hvilken datagrupperne skulle behandles, dog med mulighed for hop paa ønskede steder. Senere er dette dog gjort endnu mere fleksibelt. PLANT-programmet opererer med en særlig type datagrupper, som kaldes strømdatagrupper, som har deres egen nummerering, og som indeholder data for de forskellige strømme af gasser og vædsker. Det er ikke sikkert, at man i GIPS-systemet kan anerkende disse datagruppers særlige karakter.

PLANT-1-programmet er skrevet i ALGOL III og fungerer som et korrekt ALGOL-program, idet hvert programafsnit staar som en blok indenfor en fælles, ydre ramme, som ogsaa er skrevet i ALGOL. For det planlagte GIPS-system i ALGOL 4 er det hensigten, at hvert programafsnit skal gøres til et helt selvstændigt ALGOL-program. Den fornødne overgang fra det ene ALGOL-program til det næste sker via HELP 3-systemet og med alle relevante data lagret i et fælles dataomraade. Denne organisation vil være et væsentlig fremskridt fra PLANT-1-systemet, hvor oversættelsen af de store ALGOL-programmer giver vanskeligheder.

6.1.3. Kommunikation mellem datagrupper. Den fulde fleksibilitet af PLANT-systemet og af det kommende GIPS-system har den revolutionerende effekt, at en beregningsopgave, der kan sammenstilles af de byggestene, som findes i systemet, nu kan udføres som en enkelt beregning i systemet. Tidligere maatte man enten udføre flere beregninger efter hinanden paa eksisterende smaaprogrammer, med mellemliggende manuel fremstilling af nyt input, eller man maatte bygge et nyt program op af tidligere programafsnit og afprøve programmet inden beregningen kunne udføres. Da fremstillingen af nye programmer - ikke mindst afprøvningen - er en langsom process, er fordelen ved fleksibiliteten iøjnefaldende.

Visse vanskeligheder ved den fleksible struktur er dog ikke elimineret. Det er vigtigt, at man paa simpel og sikker maade kan overføre et eller flere beregnede tal fra een datagruppe til en anden. En beregnet afgangstemperatur fra et apparat skal maaske overføres som indgangstemperatur til et andet apparat eller som inputtal til en matematisk blok, som indstiller temperaturen paa en ønsket værdi.

I PLANT-sytemet er datakommunikationen fra een gruppe til en anden løst paa den maade, at hvis man i inputtet opgiver f.eks. en temperatur som:

370017

betyder dette, at den virkelige temperatur maa søges i datagruppe 37, element nr. 17. Denne metode har vist sin anvendelighed gennem flere aar, men den har den principielle svaghed, at den ikke kan bruges paa tal, hvis virkelige størrelse kan komme op over 10000.

I GIPS-systemet vil man formentlig bruge særlige datagrupper, som dynamisk udfører de ønskede transporter. Begge metoder har den ulempe, at de er baseret paa tal: Hvis ovennævnte sekscifrede tal ved en fejltagelse specificeres som: 380018, vil det normalt være ganske fatalt for beregningens forløb.

6.2. Databeskrivelser.

Brugen af de saakaldte databeskrivelser (data descriptions) i GIPS-systemet er en nydannelse hos Haldor Topsøe. Meningen med dette begreb er følgende: Hvis vi betragter den lille datagruppe med rørdata paa side 55 og det lille programafsnit side 85, som trykker denne datagruppe, vil man se, at hvert element i datagruppen er karakteriseret ved sit løbenummer (0-3) i datagruppen, samt ved en tekststreng, som giver en forklaring paa den variable, og et layout til trykning af tallet. Hvis vi opretter en liste, som indeholder disse oplysninger (og nogle enkelte flere) for alle elementer i datagruppen, faar vi en databeskrivelse. For ovennævnte rørdatagruppe kan databeskrivelsen f.eks. se saaledes ud:

```
[4]
6345,          RØRDATA
2, nTUB, 2,    Rørantal
1, H,    3,    Højde (m)
1, Du,   3,    Udvendig diameter (mm)
1, Di,   3,    Indvendig diameter (mm)
-1
```

Databeskrivelsen indeholder følgende elementer:

1. Størrelsen af datagruppen. Dette skrives her som [4].

2. Et typenummer for datagruppen. Dette er her skrevet som: 6345. Alle datagrupper forsynes med et typenummer i omraadet fra 1111 til 9999. Da denne datagruppe er en rent fiktiv gruppe, er typenummeret her valgt arbitrært; der findes for tiden ingen datagruppe med dette typenummer.

3. Et navn paa datagruppen, som her er: RØRDATA. Dette bruges ved trykning af overskriften paa tabellen med datagruppens tal.

4. En informationslinie for hvert dataelement. En informationslinie bestaar igen af fire dele:

1. Et heltal, som angiver layouttypen. Her anvendes foreløbigt følgende værdier:

1: {-dddddd.dd00}
2: {-dddddd}
3: { -d.ddd$_{10}$-dd}

som sikkert bliver ændret. Layouttype 0 betyder, at den paagældende informationslinie ikke vedrører et dataelement, men er en tekstforklaring til de følgende elementer. Layouttype -1 giver afslutning paa datagruppen.

2. ALGOL-navnet paa den paagældende variable, her: nTUB, H, Du og Di. Dette skal være det navn, under hvilket den variable optræder i det egentlige regneprogram.

3. Et heltal, som angiver typenummeret:

1:	boolean	5:	boolean array
2:	integer	6:	integer array
3:	real	7:	(real) array

For saadanne variable, som er arrays, skal ALGOL-navnet efterfølges af indexgrænser, ligesom ved deklarationer.

4. En tekststreng med forklaring til elementet.

6.2.1. Brug af databeskrivelser ved administration af input-output. Hvis den ovennævnte databeskrivelse gøres til det egentlige grundlag for arbejdet med datagrupper, og forefindes som en strimmel eller en stak hulkort, der naturligvis hele tiden maa revideres og forbedres, da kan man straks automatisere en del af programmeringsarbejdet:

1. Et specielt editorprogram kan generere en inputspecifikation paa basis af databeskrivelsen. Dette program eksisterer endnu ikke, men vil være meget let at lave.

2. Et andet editorprogram (som heller ikke er lavet endnu) vil kunne generere en blok til trykning af datagruppen. Dette kan enten ske paa den normale maade i en blok, som indeholder de nødvendige kald af procedurerne T1 og T2, eller man kan gøre det som et helt separat program med T1- og T2-kald for samtlige datagrupper. Et saadant separat trykprogram kan naturligvis ogsaa bringes til at operere direkte paa databeskrivelserne, hvis disse lagres paa disken paa en eller anden form. Det er ikke afgjort, hvilken af disse muligheder, som vil blive benyttet.

6.2.2. Datatransporter. En af de vigtigste fordele, som vil kunne opnaas ved brugen af databeskrivelser, er følgende. Naar datagruppen med rørdata skal bruges i en blok til udførelse af egentlige beregninger, maa den hentes frem fra disken paa en eller anden maade. I den simplest mulige form kan dette gøres paa følgende maade:

```
begin comment Normal ALGOL 4;
   integer TR, nTUB;
   real H, Du, Di;
   array DATA[0:39];
   .....
   TR := en passende værdi;
   get(DATA, FREE, TR);
   nTUB := DATA[0];
   H := DATA[1];
   Du := DATA[2];
   Di := DATA[3];
   .....
end;
```

I blokken har vi deklareret de fire variable: nTUB, H, Du og Di, som findes i datagruppen, samt et array DATA[0:39], som bruges til transporter.

Transporten til lagret sker med en get-operation, hvori kanalnummeret, TR, skal have en passende værdi, f.eks. fra et katalogopslag. Endelig kræves de fire simple sætninger:

```
nTUB := DATA[0];
H := DATA[1];
o.s.v.
```

Ved store datagrupper, og især saadanne, som indeholder lokale talsæt, bliver der tale om et betydeligt antal sætninger af formen:

```
      := DATA[ ];
DATA[ ] :=      ;
```

og programmeringsarbejdet bliver omfattende og meget kedsommeligt. Man føres derfor til at foreslaa en simplificeret notation. I denne form vil ovenstaaende programdel f.eks. kunne skrives saaledes:

```
begin comment GIPS ALGOL 4;
   integer TR;
   .......
   TR := en passende værdi;
   fetch 6345, TR, nTUB, H, Du, Di;
   .....
end;
```

Her er de fem sætninger i den første programdel erstattet af en enkelt sætning:

```
fetch 6345, TR, nTUB, H, Du, Di;
```

Dette er en saakaldt fetch-makro, og er naturligvis ikke korrekt ALGOL. Det bliver det imidlertid, hvis vi oversætter det paa et særligt oversætterprogram, der kan oversætte fra GIPS ALGOL 4 med makro-ordrer til normalt ALGOL 4, som derefter kan oversættes videre paa den normale oversætter til maskinsprog.

Specialoversætteren (som ikke er lavet endnu), skal udføre følgende operationer:

1. Alle fetch-makroer erstattes af en get-ordre. Analogt skal store-makroer udskiftes med en put-ordre. Den værdi af kanalnummeret, TR, som her er skrevet i fetch-makroen, skal ikke være det egentlige kanalnummer, men det løbenummer, som gælder for den paagældende datagruppe.

2. For hvert navn, som er nævnt i makroen efter de to første parametre, skal der indsættes de nødvendige sætninger til overførslen mellem DATA og de variable og omvendt. DATA skal automatisk indsættes i den yderste blok af programmet.

3. Hvis et navn er nævnt i makroens parameterliste, men ikke er deklareret i begyndelsen af blokken, skal de tilsvarende deklarationer tilføjes her.

Da specialoversætteren (ogsaa kaldet GIPS-editoren) har adgang til databeskrivelserne, kan den derfra ekstrahere de nødvendige oplysninger om de variables navne, deres typer og deres løbenummer i datagruppen.

Ikke alle problemer i forbindelse med brug af databeskrivelser og GIPS-editoren er endeligt afklaret. Der er f.eks. spørgsmaalet om talsæt i datagrupper. Man vil sikkert forlange, at grænserne for talsæt i en datagruppe ogsaa skal være simple variable i den samme datagruppe, men dette vil ikke altid være helt praktisk.

6.2.3. Hoveddatagrupper. Mange datagrupper vil være af ren statisk natur, som den ovennævnte rørdatagruppe eller som matematiske data som f.eks. en ganske almindelig kvadratisk matrix. Disse datagrupper hører ikke paa forhaand til noget bestemt programafsnit, men kan ovenikøbet benyttes af flere forskellige programafsnit.

For hvert programafsnit vil der normalt være en særlig datagruppe, som definerer de vigtigste parametre for kørslen af dette afsnit. Det programafsnit, som udfører en matrixinvertering, kan f.eks. være defineret ved følgende datagruppe:

```
[5]
2111,            MATRIX INVERTERING
   2, V,    2,  Matrixorden
   2, m1,   2,  Datagruppenummer af den originale matrix
   2, m2,   2,        -        -  - inverterede matrix
   3, eps,  3,  Minimum pivotelement
   2, error, 2, Fejlindikator
-1
```

Dette kaldes hoveddatagruppen for matrixinverteringsprogramafsnittet. Programmet skal ogsaa operere paa to sekundære datagrupper, nemlig den originale og den inverterede matrix. De skal opgives med deres datagruppeløbenummer, m1 og m2. Vælges disse identisk, lagres den inverterede matrix oveni den originale.

Man vil formentlig indrette sine programmer saaledes, at naar man gaar ind i et programafsnit, vil løbenummeret for den tilhørende hoveddatagruppe automatisk være assignet til en passende global variabel. Løbenumrene for de sekundære datagrupper maa fremskaffes ved programmering af en fetch-makro for hoveddatagruppen.

6.3. GIPS-Editoren.

GIPS-editorens vigtigste funktion under oversættelsen er at erstatte de anvendte makros (fetch og store) med korrekte ALGOL-sætninger. Hvis man tænkte sig, at alle vore programmer var skrevet i dette GIPS ALGOL med makros, ville det være en forholdsvis let sag at lave tre forskellige versioner af denne editor, nemlig svarende til ALGOL II, ALGOL III og ALGOL 4. Editoren for ALGOL II skulle da erstatte fetch-makros med:

```
tromleplads := noget;
fra tromle (DATA);
nTUB := DATA[0];
........
```

Editoren for ALGOL III skulle indsætte:

```
drumplace := noget;
from drum (DATA);
nTUB := DATA[0];
........
```

Endelig skulle editoren for ALGOL 4 indsætte:

```
get (DATA, FREE, noget);
nTUB := DATA[0];
.....
```

Overgangen fra ALGOL II til ALGOL III eller fra ALGOL III til ALGOL 4 kunne da udføres paa den smertefri og automatiske maade, at man indsætter de nødvendige rettelser i editoren og derefter kører alle programmerne paa editoren. Denne oversættelse behøver ikke at være særlig hurtig, da den kun skal udføres sjældent, nemlig hver gang man skifter fra een ALGOL-variant til en ny. Man vil heller ikke stille krav om, at oversættelsen skal være absolut automatisk; det vil være overkommeligt at foretage enkelte manuelle rettelser efter editorens oversættelse.

De sætninger i et regneprogram, som ikke er datatransportmakroer, vil ofte være af en simpel natur, som f.eks.

```
Kser := kpart×(1-void)×(0.5×h×Dp+krad+kcont)/(0.5×h×Dp+kpart);
```

Sætninger, som kun indeholder de fire simple regningsarter og maaske potensopløftning vil være næsten ens i ALGOL og i f.eks. FORTRAN. Der er kun smaa og trivielle forskelle som:

ALGOL	FORTRAN
:=	=
×	1 stjerne
↑	2 stjerner
;	Bruges ikke

Hvis programmet er skrevet i eet af disse sprog, vil det være let at inkludere oversættelsen af disse tegn fra det ene sprog til det andet i GIPS-editorens funktioner. Visse andre ting vil naturligvis volde vanskeligheder, f.eks. det, at man i FORTRAN ikke kan bruge sammensatte sætninger eller Jensens device i procedurekald.

Vi har endnu ikke afgjort, om vort grundsprog til programmering skal være GIPS-ALGOL eller GIPS-FORTRAN, men det er mest sandsynligt, at det bliver GIPS-FORTRAN. Man kan stille det spørgsmaal, hvorfor vi ikke bare vælger korrekt ALGOL 4 eller korrekt FORTRAN IV. Men her maa man huske paa, at vi ikke blot skal udføre en mængde kemiske beregninger paa programmer skrevet i eet eller andet sprog, men at vi ogsaa skal kunne levere større eller mindre afsnit af disse programmer til forskellige kunder, der meget let kan stille krav om at faa programmerne lavet i et bestemt sprog.

6.4. Implementering af GIPS-Systemet.

I øjeblikket findes der hos Haldor Topsøe følgende dele af det planlagte GIPS-system:

1: Et sæt databeskrivelser, især for matematiske beregninger og konverterberegninger.

2. Foreløbige konventioner for datakatalog, inputmateriale og joblister.

3. Et foreløbigt monitorprogram.

4. Tre foreløbige programmer i ALGOL 4, nemlig til simpel input og output og til matrixinvertering i lagret.

For selve GIPS-editoren er udført en vis mængde af systemanalysen, men dette arbejde er ikke afsluttet. De enkelte programmer maa da programmeres direkte i ALGOL 4.

6.4.1. Databeskrivelser. Disse findes paa strimler mærket DD-1000, DD-2000, o.s.v. Der regnes med følgende 9 klasser:

DD-1000: Input, output, etc.
DD-2000: Matematik og statistik.
DD-3000: Termodynamik og kinetik.
DD-4000: Reserveret til varmevekslere, kolonner og absorbere.
DD-5000: Behandling af strømme.
DD-6000: Kemiske reaktioner.
DD-7000: Reserveret mekaniske beregninger.
DD-8000: Ledig.
DD-9000: Diverse.

Underopdelingen af disse klasser findes i ovennævnte databeskrivelser.

6.4.2. Datalagring. Her er der indført følgende, foreløbige konventioner:

6.4.2.1. Arealnavne. Alt datamateriale placeres i et nyt areal ved navn ⟨<data⟩. De termodynamiske data, som i øjeblikket er lagret i arealet: ⟨<TDATA⟩, forbliver i dette areal, men ved første givne lejlighed vil navnet blive ændret til ⟨<tdata⟩, fordi help-systemet ikke accepterer navne med store bogstaver. Dette kræver ogsaa ændring til ⟨<tdata⟩ i nogle af de nye procedurebeskrivelser i ALGOL 4.

6.4.2.2. Inputstrimler. Konventioner for forsidedata og anden indledende administration vil senere blive fastlagt.

Hver egentlig datagruppe noteres som i den tilsvarende GIPS-databeskrivelse. Hvert tal afsluttes med komma eller CR og datagruppen slutter med et stop-e. Bogstaverne d og q bruges foreløbigt ikke. Stop-eet skal være terminator for sidste tal.

Hver datagruppe lagres fra celle 0 i en kanal, saaledes at celle 0 indeholder den første navngivne variable i databeskrivelsen.

Hver datagruppe skal paa inputstrimlen indledes med 3 ekstra tal, der af indlæseprogrammet lagres et andet sted end blandt det egentlige datamateriale. De 3 tal faar altsaa ikke noget cellenummer, men da det nedenfor afsløres, hvor de 3 tal havner, har man altid mulighed for at rette paa dem under kørslen (fra programmet), hvis det er nødvendigt.

De tre tal er:

DGN: Datagruppenummer
TN : Typenummer
LE : Længde af datagruppen

Datagruppenummeret, DGN, skal ligge i omraadet:

$$1 \leq DGN \leq 399$$

DGN skal opgives af brugeren, og man bør begynde forfra med 1, 2, 3, o.s.v.

De datagrupper, som ikke er input, men som oprettes internt af et kørende program, faar tildelt numre i omraadet:

$$1000 \leq DGN \leq 1399$$

Programmerne bør naturligvis nedlægge disse datagrupper igen, naar de ikke længere bruges, men gør man ikke dette, har man automatisk sikret overlevelse af sine data til senere.

Typenummeret, TN, skal svare til numrene 1111-9999 i GIPS-databeskrivelserne.

Længden, LE, er antallet af kanaler med 40 celler.

6.4.2.3. Datakatalog. Dette bestaar af to dele: specielle oplysninger paa kanal 1 i {<data} samt det egentlige katalog, som indeholder een celle for hver datagruppe. De fordeles paa kanalerne:

Kanal 2-11: Inputdatagrupper
Kanal 12-21: Programgenererede datagrupper.

Informationen om datagruppe DGN er anbragt i celle:

DGN mod 40;

paa kanal t i {<data}, hvor t er bestemt ved:

t := DGN:40 + (if DGN ≥ 1000 then -13 else 2);

Indholdet af informationscellen for datagruppe DGN i datakataloget er pakket saaledes:

TN i bits 0-13
LE - - 14-26
track1 - - 27-39

Her er TN og LE forklaret ovenfor, medens track1 er relativt kanalnummer for første kanal i denne datagruppe, altsaa et tal ≥ 22 med den her benyttede katalogstørrelse (der evt. kan ændres senere).

Bemærk, at tallet DGN ikke gemmes noget sted, jvf. dog nedenfor.

Ovennævnte pakning af informationscellen giver følgende grænser, naar tallene ekstraheres uden fortegn:

TN	0-16383
LE	0-8191
track1	0-8191

6.4.2.4. Specielle data. Kanal 1 i ‹<data› er foreløbigt reserveret følgende data:

Celle:	Indhold
0	count
1	DGN
2	TN
3	LE
4	track1
5	ftrack
6	dg1

Betydningen af count forklares nedenfor. De fire tal: DGN, TN, LE og track1 er simpelthen de lige forklarede tal for den datagruppe, som er hoveddatagruppe for det delprogram, som vi i øjeblikket regner i. Man kan da i programmet hente disse tal frem, hvis man har brug for dem.

ftrack er det første frie kanalnummer i ‹<data›, og dg1 er første frie datagruppenummer i omraadet $1000 \leq dg1 \leq 1399$.

6.4.2.5. Job-lister. Ordretælleren, count, henviser til en jobliste, som er en liste over, hvilke delprogrammer, der skal regnes paa. Databeskrivelsen for en jobliste er:

```
[3 + NJ]
1111, JOBLISTE
2,  NJ,  2,  Antal af jobs
2,  AJ,  2,  Aktuelt jobnummer
2,   R,  2,  Fortsæt med denne jobliste
2,  JL[1:NJ],  6,  Jobliste
-1
```

Hvis ordretælleren, count, i et givet øjeblik har værdien 17, betyder det, at datagruppe nr. 17 er jobliste for det kørende delprogram.

Hvis datagruppe nr. 17 samtidig ser saaledes ud:

```
6   NJ
3   AJ
24  R
1   JL[1]
2   JL[2]
12  JL[3]
13  JL[4]
14  JL[5]
15  JL[6]
```

betyder det, at vi i øjeblikket regner paa det program, hvis hoveddatagruppenummer er JL[AJ], d.v.s. JL[3]: 12. Det er altsaa datagruppe 12, som definerer det i øjeblikket regnende delprogram. Tallet 12 vil da staa i celle 1 paa kanal 1 og typenummeret i celle 2. Disse tal staar naturligvis ogsaa i datakataloget.

Det delprogram, der styrer overgangen fra eet delprogram til det næste, maa tælle frem i AJ, indtil AJ > NJ. Naar dette sker, sættes count = R, og der køres videre med en ny jobliste.

Det er muligt, at det vil vise sig nødvendigt eller i hvert fald praktisk at udvide joblisten, saaledes at der for hvert element i joblisten ogsaa staar et tal, som angiver, hvilket program det skal regnes paa. Dette er i øjeblikket kun givet indirekte ved typenummeret paa den tilsvarende hoveddatagruppe.

6.4.3. Foreløbig Monitor. Til midlertidig afprøvning af systemet er lavet følgende programmer:

Navn	
mon1	Monitor
p1211	SIMPLE INPUT
p1311	SIMPLE OUTPUT
p2111	MATRIX INVERSION

Endvidere er der lavet en variant af HELP3-programmet run: run1.

En beregning startes ved at kalde p1211 fra skrivemaskinen. Programmet læser en strimmel og lagrer tallene i {<data}. Styreparametrene før hver datagruppe fyldes i kataloget. En speciel datagruppe giver nulstilling af kataloget og, hvis det ønskes, reservering af {<data}-arealet. En anden speciel datagruppe giver afslutning paa indlæsningen. Denne datagruppe indeholder ogsaa startværdien af count, som bliver indsat. Derefter hopper p1211 selv til monitorprogrammet, mon1.

mon1 analyserer joblisten og kalder de ønskede programmer frem efterhaanden. Naar sidste jobliste er afsluttet, er count = 0, og monitoren hopper igen til inputprogrammet.

Trykning udføres som et almindeligt delprogram med det tilsvarende datagruppenummer i joblisten.

Det specielle program run1 er fremstillet af program run ved følgende operation paa skrivemaskinen:

```
HP
move, run, free<
res, run1<
```

Herved er skabt et program run1 identisk med run. Derefter oversættes og køres ALGOL-program d-304, som indsætter følgende ændringer i run1:

```
run1-kanalen hentes frem.
Celle 7 rettes fra   ar a8, gr 263 + a
til:                 arn r+28, gr 263 + a
Celle 35 rettes fra  0 til qq 36.39
Kanalen skrives tilbage igen.
```

Virkningen af rettelsen er den, at et ALGOL-program, som er kaldt ved run1, slutter med et hop til kanal 36. Her staar et stykke maskinkode, som gør et programmeret kald af HELP, som derefter kalder mon1 via run1.

Programmet mon1 ser saaledes ud:

```
Program mon1, preliminary monitor.
begin
   message
   Translation of mon1;
   integer FREE, count, DGN, TN, LE, c1, c2, c3, t1,
   t2, t3, NJ, AJ, R, track1;
   boolean word;
   integer array LIST[0:39];
   array DATA[0:39];
   select(17);
   if where(«<data», FREE) ≠ 0 then go to L3;
   get(LIST, FREE, 1);
   count := LIST[0];
L1:if count = 0 then
   begin
      writetext(«<
End of calculation, ready for new input: »);
      lyn;
      TN := 1211;
      go to L2;
   end count = 0;
   c1 := count mod 40;
   t1 := count:40 + (if count ≥ 1000 then -13 else 2);
   get(LIST, FREE, t1);
   t2 := LIST[c1] mod 8192;
   get(DATA, FREE, t2);
   comment start of actual job list;
   NJ := DATA[0];
   DATA[1] := DATA[1] + 1;
   AJ := DATA[1];
   R := DATA[2];
   put(DATA, FREE, t2);
   comment return of updated actual job number;
   if AJ ≤ NJ then
   begin
      c3 := AJ + 2;
      t3 := t2 + c3:40;
      c3 := c3 mod 40;
      if t3 > t2 then get(DATA, FREE, t3);
```

```
    DGN := DATA[c3]
end same job list else
begin
   count := R;
   go to L1;
end new job list;
c2 := DGN mod 40;
t2 := DGN:40 + (if DGN ≥ 1000 then -13 else 2);
get(LIST, FREE, t2);
word := boolean LIST[c2];
comment catalog item in word;
TN := integer((word shift -26) ∧ (40 16383));
LE := integer((word shift -13) ∧ (40 4095));
track1 := (integer word) mod 4096;
get(LIST, FREE, 1);
LIST[0] := count;
LIST[1] := DGN;
LIST[2] := TN;
LIST[3] := LE;
LIST[4] := track1;
put(LIST, FREE, 1);
```

```
L2:code TN;
   2, 44;
   grn  r+15          M  ;  0  cell := 0
   pa    r+8 t+3         ;  1  set tæller
   pa    r+3             ;  2  set shift
   pm   r+14 , gm  r+17  ;  3  DIV := 1000
   pmn  a1   , dl  r+16  ;  4  TN/DIV
   tk     0  +6          ;  5  shift
   ac    r+9 , gm  r+15  ;  6  add to cell, store rem.
   pmn  r+13 , dl  r+11  ;  7  DIV/10
   gr   r+12 , pmn r+13  ;  8  New DIV, rem.
   bt     0 t-1          ;  9  count
   hh    r-6             ;  10  repeat
   arn   r+8 , ac   r+4  ;  11  add end and p
   hs      1             ;  12  help
   hv      0             ;  13  not used
   trun1;                ;  14
   qq      0             ;  15  tp1234;
   qqf,                  ;  16  stop
   qq  1000.39           ;  17
   qq    10.39           ;  18
   qq 10.3+10.9+39.39    ;  19  end + p
   qq                    ;  20  DIV
   qq                    ;  21 rem.
   e;
L3:writetext(<<
Monitor trouble!);
   count := 0;
   go to L1;
   message
   end of mon1 reading;
end mon1;
```

Maskinkoden i mon1 har den funktion, at typenummeret, TN, der er et firecifret heltal, omregnes saaledes, at hvis TN = 9876, vil maskinkoden kalde HELP 3-systemet med følgende ordrer:

```
hs  1
hv  0
trun1
tp9876
qqf,
```

For hvert aktivt typenummer, d.v.s. et typenummer hvor datagruppen er hoveddatagruppe for et program, skal dette program være lagret paa disken i oversat form under det tilsvarende navn, her p9876.

6.4.4. Program p1211. SIMPLE INPUT. Dette program ser saaledes ud:

```
Program p1211.  SIMPLE INPUT.
begin
   message
   Translation of p1211, SIMPLE INPUT;
   integer FREE, ftrack, DGN, TN, LE, dummy, i,
   track1, t, c1, t1, dg1;
   integer array LIST[0:39];
   array DATA[0:39];
   select(16);
   if where(<data>, FREE) = 0 then
   begin
      get(LIST, FREE, 1);
      ftrack := LIST[5];
      dg1 := LIST[6];
   end if not first call;
L1:DGN := read integer;
   TN := read integer;
   LE := read integer;
```

```
if DGN = 0 then
begin comment special input;
   if TN = 0 then
   begin comment end of input;
      get(LIST, FREE, 1);
      LIST[0] := LE;
      LIST[5] := ftrack;
      LIST[6] := dg1;
      put(LIST, FREE, 1);
      code dummy;
      2, 44;
      hs  1  ;  call help
      hv  0  ;  and
      trun1; ;  fetch
      tmon1; ;  mon1
      qqf,   ;
      e;
   end if TN = 0 else
   begin comment start of input;
         if LE > 0 then
         begin
            cancel(‡<data‡);
            if reserve(‡<data‡, LE) ≠ 0 then
            begin
               writetext(‡<
Trouble in reservation of data area‡);
               go to L5;
            end if reserve;
         end if LE > 0;
         if where(‡<data‡, FREE) ≠ 0 then go to L4;
         for i := 0 step 1 until 39 do LIST[i] := 0;
         i := if LE > 0 then LE else 21;
         for i := i step -1 until 1 do
         put(LIST, FREE, i);
         ftrack := 22;
         dg1 := 1000;
```

```
            end TN > 0;
            go to L1;
         end DGN = 0;
         get(LIST, FREE, 1);
         t := track1 := ftrack;
         ftrack := ftrack + LE;
L2:      get(DATA, FREE, t);
         i := 0;
L3:      DATA[i] := read real;
         i := i + 1;
         if i > 39 then
         begin
            put(DATA, FREE, t);
            t := t + 1;
            if char ≠ 53 then go to L2;
         end new track;
         if char ≠ 53 then go to L3;
         put(DATA, FREE, t);
         c1 := DGN mod 40;
         t1 := 2 + DGN:40;
         get(LIST, FREE, t1);
         LIST[c1] := (TN×8192 + LE)×8192 + track1;
         put(LIST, FREE, t1);
         go to L1;
L4:      writetext(‹‹
Data catalog trouble›);
L5:      ;
      message
       end of p1211 reading;
   end program p1211;
```

Indlæsning af en datagruppe starter med indlæsning af de tre specielle tal:

```
DGN := read integer;
 TN := read integer;
 LE := read integer;
```

Hvis DGN = 0, er der tale om en speciel indlæsning, hvis nøjagtige forløb bestemmes af TN.

For TN = 0 er indlæsningen slut, og det tredje tal, LE, bruges som startværdi af ordretælleren, count. Denne indsættes i celle 0 paa kanal 1 og mon1 kaldes med run1 via HELP3.

For DGN = 0 og TN > 0 er der tale om start af input. Hvis LE > 0, nedlægges hele arealet {<data} og oprettes igen med LE kanaler. Kataloget nulstilles.

Den normale indlæsning sker med en sætning:

```
DATA[i] := read real;
```

hvor i > 39 giver skift til næste kanal. Slut paa datagruppen faas af et stop-e, der skal være anbragt umiddelbart efter sidste tal. SPACE er tilladt før e-et, men ikke CAR RET.

6.4.5. Program p1311. SIMPLE OUTPUT. Dette ser saaledes ud:

```
Program p1311. SIMPLE OUTPUT
begin
   message
   Translation of p1311, SIMPLE OUTPUT;
   integer FREE, DGN, t1, N, c1, n, i, first, last,
   track1, c2, t2, j;
   integer array LIST[0:39];
   array DATA[0:39];
   select(8);
   if where({<data}, FREE) ≠ 0 then
   begin
      select(24);
      writetext({<
Data catalog trouble});
      go toL1;
   end if trouble;
   writecr;
   writetext({<SIMPLE OUTPUT});
   writecr;
   get(LIST, FREE, 1);
   DGN := LIST[1];
```

```
t1 := LIST[4];
get(DATA, FREE, t1);
N := DATA[0];
begin comment inner block;
   integer array A[1:N, 1:3];
   c1 := 1;
   for n := 1 step 1 until N do
   for i := 1 step 1 until 3 do
   begin
      A[n, i] := DATA[c1];
      c1 := c1 + 1;
      if c1 > 39 then
      begin
         t1 := t1 + 1;
         get(DATA, FREE, t1);
         c1 := 0;
      end if new track;
   end for n and i;
   for n := 1 step 1 until N do
   begin
      DGN := A[n, 1];
      first := A[n, 2];
      last := A[n, 3];
      writecr;
      writetext({<Data group no.});
      write({-dddd}, DGN);
      writecr;
      writecr;
      c1 := DGN mod 40;
      t1 := DGN:40 + (if DGN ≥ 1000 then -13 else 2);
      get(LIST, FREE, t1);
      track1 := LIST[c1] mod 4096;
      c2 := first mod 40;
      t2 := track1 + first:40;
      get(DATA, FREE, t2);
      j := 0;
```

```
            for i := first step 1 until last do
            begin
               writetext(‹<(›);
               write(‹ddd›, i);
               writetext(‹<)›);
               write(‹  -d.dddddd10-dd›, DATA[c2]);
               c2 := c2 + 1;
               if c2 > 39 then
               begin
                  t2 := t2 + 1;
                  get(DATA, FREE, t2);
                  c2 := 0;
               end new track;
               j := j + 1;
               if j mod 3 = 0 then writecr;
            end for i;
            writecr;
         end for n;
      end inner block;
L1:   ;
      message
      end of p1311 reading;
   end program p1311;
```

Databeskrivelsen til hoveddatagruppen for dette program ser saaledes ud:

```
[1+3×N]
1311,                  SIMPLE OUTPUT
2, N,             2,   Antal datagrupper
2, A[1:N, 1:3], 6,     Datagruppenummer
                       Første element
                       Sidste element
-1
```

Programmet bruger ikke sideskiftkontrol. Først hentes hoveddatagruppen frem. Det antal egentlige datagrupper, N, som skal trykkes, staar i celle 0 i hoveddatagruppen. Resten af denne gruppe er et array:

```
A[1:N, 1:3]
```

hvori A[n, 1] giver datagruppenummeret paa datagruppe nr. n. Indenfor denne datagruppe giver A[n, 2] nummeret paa det første element, der skal trykkes, og A[n, 3] nummeret paa det sidste. Hoveddatagruppen selv kan naturligvis ogsaa indgaa som en af de datagrupper, der skal trykkes.

Trykningen begynder med teksten SIMPLE OUTPUT. For hver datagruppe trykkes gruppenummeret, samt de ønskede elementer med tre pr. linie og deres løbenummer.

6.4.6. Program p2111. MATRIX INVERSION. Den tilsvarende hoveddatagruppe er vist paa side 123. Selve programmet ser saaledes ud:

```
Program p2111. MATRIX INVERSION
begin
   message
   Translation of program p2111. MATRIX INVERSION;
   integer V, m1, m2, error, FREE, DGN, track1, c1, t1, i, j;
   real eps;
   copy INV1 <
   begin
      integer array LIST[0:39];
      array DATA[0:39];
      if where(1<data1, FREE) ≠ 0 then go to L1;
      get(LIST, FREE, 1);
      DGN := LIST[1];
      track1 := LIST[4];
      get(DATA, FREE, track1);
      V := DATA[0];
      m1 := DATA[1];
      m2 := DATA[2];
      eps := DATA[3];
   end data transfer;
   begin
      array a[1:V, 1:V];
      begin
         integer array LIST[0:39];
         array DATA[0:39];
         c1 := m1 mod 40;
         t1 := m1:40 + (if m1 ≥ 1000 then - 13 else 2);
```

```
      get(LIST, FREE, t1);
      t1 := LIST[c1] mod 4096;
      c1 := 2;
      get(DATA, FREE, t1);
      for i := 1 step 1 until V do
      for j := 1 step 1 until V do
      begin
         a[i, j] := DATA[c1];
         c1 := c1 + 1;
         if c1 > 39 then
         begin
            t1 := t1 + 1;
            get(DATA, FREE, t1);
            c1 := 0;
         end if new track;
      end for i, j;
   end transfer block;
   error := INV1(V, a, eps);
   begin
      array DATA[0:39];
      integer array LIST[0:39];
      c1 := m2 mod 40;
      t1 := m2:40 + (if m2 ≥ 1000 then - 13 else 2);
      get(LIST, FREE, t1);
      t1 := LIST[c1] mod 4096;
      c1 := 2;
      get(DATA, FREE, t1);
      for i := 1 step 1 until V do
      for j := 1 step 1 until V do
      begin
         DATA[c1] := a[i, j];
         c1 := c1 + 1;
         if c1 > 39 then
         begin
            put(DATA, FREE, t1);
            t1 := t1 + 1;
```

```
                get(DATA, FREE, t1);
                c1 := 0;
             end if new track;
          end for i, j;
          put(DATA, FREE, t1);
          get(DATA, FREE, track1);
          DATA[4] := error;
          put(DATA, FREE, track1);
       end transfer block;
    end inner block;
    go to L2;
L1:select(17);
    writetext(*<
catalog trouble in matrix inversion*);
    writecr;
    writetext(*<Data group no.*);
    write(*-dddd*, DGN);
L2:
end program p2111;
```

Programmet bestaar i det væsentlige af et kald af proceduren INV1, der inverterer en matrix i lagret. Desuden findes den nødvendige administration af hentningen af den originale matrix og tilbagetransporten af den inverterede matrix til disken. I den endelige version af programmet skal disse transporter naturligvis udføres med passende standardprocedurer.

6.4.7. Typisk beregning. En beregning paa ovennævnte GIPS-programsystem med de tre programmer: p1211, p1311 og p2111 blev udført med følgende inputstrimmel:

```
Ff
0, 1, 100,
1, 1111, 1,
2, 0, 0,
2, 3 e
2, 2111, 1,
3, 4, 5,
1₁₀-20, 0 e
3, 1311, 1, 5,
1, 0, 4,
2, 0, 4,
3, 0, 15,
4, 0, 10,
5, 0, 10 e
4, 2621, 1, 3, 3,
1, 0.5, 0.25,
1, 1, 1,
1, 0, 0 e
5, 2621, 1, 3, 3 e
0, 0, 1,
```

Resultatet saa saaledes ud:

SIMPLE OUTPUT

Data group no. 1

(0)	2.000000	(1)	2.000000	(2)	0.000000
(3)	2.000000	(4)	3.000000		

Data group no. 2

(0)	3.000000	(1)	4.000000	(2)	5.000000
(3)	$1.000000_{10}-20$	(4)	0.000000		

Data group no. 3

(0)	5.000000	(1)	1.000000	(2)	0.000000
(3)	4.000000	(4)	2.000000	(5)	0.000000
(6)	4.000000	(7)	3.000000	(8)	0.000000
(9)	$1.500000_{10}1$	(10)	4.000000	(11)	0.000000
(12)	$1.000000_{10}1$	(13)	5.000000	(14)	0.000000
(15)	$1.000000_{10}1$				

Data group no. 4

(0)	3.000000	(1)	3.000000	(2)	1.000000
(3)	$5.000000_{10}-1$	(4)	$2.500000_{10}-1$	(5)	1.000000
(6)	1.000000	(7)	1.000000	(8)	1.000000
(9)	0.000000	(10)	0.000000		

Data group no. 5

(0)	3.000000	(1)	3.000000	(2)	$-7.450581_{10}-9$
(3)	$-1.862645_{10}-9$	(4)	1.000000	(5)	4.000000
(6)	-1.000000	(7)	-3.000000	(8)	-4.000000
(9)	2.000000	(10)	2.000000		

Datagruppe 1 er joblisten, der umiddelbart efter indlæsningen indeholder:

(0)	2	NJ ,	antal jobs
(1)	0	AJ ,	aktuelt jobnummer
(2)	0	R ,	datagruppenummer for næste jobliste (0 giver stop)
(3)	2	JL[1],	Job nr.1, = datagruppe 2
(4)	3	JL[2],	- - 2, = - 3

Paa det tidspunkt, hvor trykningen foretages, er AJ talt frem til 2.

Datagruppe 2 indeholder matrixinverteringsdata, nemlig:

(0)	3	V,	matrixorden
(1)	4	m1,	datagruppenummer for original matrix
(2)	5	m2,	- - inverteret matrix
(3)	1_{10}-20	eps,	minimum pivot element
(4)	0	error,	fejlindikator

Datagruppe 3 indeholder data for trykprogrammet, nemlig:

(0) 5 N, antal datagrupper

A[1:5, 1:3]:

n	A[n, 1] Datagruppe-nummer	A[n, 2] Første element	A[n, 3] Sidste element
(1-3)	1	0	4
(4-6)	2	0	4
(7-9)	3	0	15
(10-12)	4	0	10
(13-15)	5	0	10

Endelig indeholder datagruppe 4 den originale matrix:

(0)	3	R,	antal	rækker
(1)	3	C,	-	søjler

n	M[n, 1]	M[n, 2]	M[n, 3]
(2-4)	1	0.5	0.25
(5-7)	1	1	1
(8-10)	1	0	0

og datagruppe 5 den inverterede:

(0)	3	R,	antal	rækker
(1)	3	C,	-	søjler

n			
(2-4)	-7.450581_{10}-9	-1.862645_{10}-9	1
(5-7)	4	-1	-3
(8-10)	-4	2	2

Skrevet paa denne primitive form er de trykte resultater naturligvis noget besværlige at tyde.

7. REFERENCER

Kjær, J.: Thermodynamic Calculations on an Electronic Digital Computer. Akademisk Forlag, København (1963d).

Kjær, J.: Elementært ALGOL. Haldor Topsøe (1968).

Lauesen, S. et al.: A Manual of HELP 3. Regnecentralen (1967).

Naur, P.: The design of the GIER ALGOL compiler. Part I. BIT, 3, 124 - 140 (1963).

Naur, P. et al.: A Manual of GIER ALGOL III. Regnecentralen (1965a).

Naur, P.: The performance of a system for automatic segmentation of programs within an ALGOL compiler (GIER ALGOL). Comm. ACM, 8, 671 - 677 (1965b).

Naur, P. et al.: A Manual of GIER ALGOL 4. Regnecentralen (1967).

8. STIKORDSREGISTER