ATOMENERGIKOMMISSIONEN
Forsøgsanlæg Risø
Regnemaskingruppen

Risø-M-34
Marts 1964
200 eks.

Tillæg til

LÆREBOG I GIER-ALGOL

af

Helge Vilstrup ved Leif Hansson

Indholdsfortegnelse til tillæg.

Tillæg til Appendix

Indledning.

Der foreligger nu en ny udgave af GIERs algoloversætter, den saakaldte GIER-ALGOL III. I forhold til den tidligere benyttede oversætter adskiller den sig ved at være hurtigere, at have indført visse nye muligheder, at have fjernet enkelte lidet anvendte muligheder, samt at anvende engelske gloser for en række standardprocedurer.

Dette tillæg knytter sig til H. Vilstrup: LÆREBOG I GIER-ALGOL, Marts 1963, idet de relevante dele er blevet genoptrykt med de nødvendige ændringer. Enkelte afsnit, navnlig i appendix, er ikke genoptrykt, selv om der forekommer danske navne paa standardprocedurer. Til støtte for oversættelsen skal anføres, at ,,læs-,, og ,,tryk-,, svarer til ,,in-,, og ,,out-,, medens ,,tast-,, og ,,skrv-,, modsvarer ,,type-,, og ,,write-,,.

20. Indlæsning af tal, kommentarer.

Vores færdige ALGOL-program indeholder en række ALGOL-sætninger og konstanter, men ikke de parametre der varierer fra opgave til opgave. Vi har derfor ogsaa standardfunktioner, der sørger for indlæsning af tal.

Som et eksempel kan vi se paa et program til løsning af 2.grads ligninger. Proceduren til dette formaal blev defineret i afsnit 17. Kun procedurehovedet vil blive gentaget her:

```
begin comment programmet løser en komplex 2.grads ligning;

real a,b,c, Z1r, Z1i, Z2r, Z2i, test;
procedure EQ2OR(A,B,C,z1r,z1i,z2r,z2i,INDETERMINATE);
value A,B,C;
real A,B,C,z1r,z1i,z2r,z2i;
label INDETERMINATE;
begin
her skrives procedureblokken fra afsnit 17.
end of EQ2OR;

start:
input (a,b,c);
EQ2OR(a,b,c,Z1r,Z1i,Z2r,Z2i, FEJL);
.............................
Nu udføres trykning af resultater, herom nærmere i næste afsnit
.............................
input(test);
if test ≥ 0 then go to start;
end of program;
```

Programmmet begynder med en kommentar. I ALGOL er det tilladt overalt, hvor der er skrevet ; at tilføje

```
comment vilkaarlige tegn fra flexowriteren;
```

ALGOL-oversætteren ignorerer simpelthen alt fra comment til og med ;. Yderligere er det som her tilladt at tilføje comment ----; efter begin. Bemærk at der ikke maa være et semikolon mellem begin og comment. Dernæst følger deklarationen af alle anvendte variable og proceduren. Umiddelbart efter proceduren begynder det egentlige program. Den 1. sætning er

input (a, b, c);

Naar maskinen (i det oversatte program) møder procedurebetegnelsen input, starter den strimmellæseren, der bør indeholde hulstrimmelen med vores data. Der læses tre tal, og de variable a, b, c faar de tilsvarende talværdier. Paa strimmelen kan der f. eks. staa:

420, 22.9, 37.8, 0,

Tallene skrives efter de konventioner, vi har nævnt i afsnit 5, reelle tal. Som skilletegn mellem tallene kan vi anvende alle normale Flexowritersymboler, der ikke kan indgaa i et tal, altsaa ikke: et ciffer, ., 10, +, -. Blinde symboler er: mellemrum, _.
Der maa gerne være flere skilletegn mellem to tal.

Naar a, b, og c har faaet en værdi svarende til de tre første tal paa hulstrimmelen, gaar maskinen videre til næste sætning. Denne sætning er et procedurekald med indgangsstørrelserne a, b, og c og med resultaterne Z1r, Z1i, Z2r, Z2i.

Vi bemærker, at label FEJL ikke skal deklareres, men defineres i programmet, hvilket forudsættes at ske i een af de prikkede linier. (Derimod skal en label der indgaar som formel parameter i et procedurehoved specificeres i procedurehovedet, saaledes som det ogsaa er vist for INDETERMINATE paa side 4).

Dernæst udføres en udskrift af resultaterne. Herom nærmere i næste afsnit. Nu læses endnu et tal og test faar denne talværdi. Hvis dette tal er ≥ 0, hopper maskinen tilbage til indlæsning af det næste talsæt, hvis tallet er < 0, fortsætter maskinen gennem det sidste end i programmet, beregningen af 2.ordens ligninger er slut for denne gang, og maskinen stopper, klar til næste ALGOL-program.

Hvis vi skal læse et array, kunne det ske paa følgende maade:

```
begin
real array A[1:2, 1:3];
input (A[1,1], A[1,2], A[1,3], A[2,1], A[2,2], A[2,3]);
```

Vi kunne naturligvis ogsaa have anvendt en for-sætning:

```
begin
integer i,j;
real array A[1:2, 1:3];
for i := 1 step 1 until 2 do
for j := 1 step 1 until 3 do
input (A[i,j]);
```

Denne metode har den fordel, at den er uafhængig af antallet af array-komponenter. Endelig har vi endnu en metode, der specielt er indført for at lette indlæsning af arrays, idet vi blot kan skrive:

```
begin
real array A[1:2, 1:3];
input (A);
```

Vi skal blot opgive navnet paa talsættet og faar da indlæst hele talsættet paa samme maade som i de to foregaaende eksempler. Er talsættet flerdimensionalt foregaar indlæsningen paa analog maade.

Fejlreaktion i forbindelse med læsning af ukorrekte datastrimler er nærmere omtalt i appendix, afsnit A6, sammen med andre læseprocedurer.

21. Trykning af resultater.

ALGOL-programmets resultater bliver normalt skrevet ud paa en hulstrimmel af regnemaskinen. Denne hulstrimmel bliver dernæst indsat i Flexowriteren, der skriver resultaterne ud pa papir, nøjagtig som de staar paa hulstrimmelen. Flexowriteren vil normalt indeholde papir i format A4 paa højkant, hvor de enkelte papirark hænger sammen ved hjælp af en perforering.

Naar man skal planlægge sin resultatudskrift, vil det første spørgsmaal være, hvad skal man skrive ud.

Det er klart, at resultaterne skal skrives ud, men det vil oftest være meget nyttigt ogsaa at skrive indgangsparametrene ud. Det er jo disse, der definerer resultaterne, og man kan ikke være fuldstændig sikker paa, at regnemaskinen har læst datastrimmelen korrekt, eller at den rigtige datastrimmel er benyttet. Endvidere kan man hurtigt samle sig saa mange resultatark, at det er overordentlig nyttigt, at indgangsparametrene staar sammen med resultaterne.

Hvis resultaterne fylder mere end een side, vil man have brug for et automatisk skift til den næste side, saaledes at man ikke faar resultater skrevet ud oven i perforeringen, og denne nye side maa indeholde en overskrift, der knytter den sammen med den første side, saa man ikke faar ombyttet resultatarkene.

Sluttelig bør man benytte en række symboler ved begyndelsen og slutningen af hulstrimmelen, saaledes at Flexowriteren stopper, naar hulstrimmelen er gennemlæst, og saaledes at det er muligt at kontrollere (ved automatisk kontrol af tværsummen af samtlige karakterer pa hulstrimmelen), at regnemaskinen virkelig har hullet de ønskede tegn paa hulstrimmelen.

Da en række af disse spørgsmaal gaar igen i de fleste programmer, er det naturligt at benytte en generelt anvendelig administrationsalgoritme til dette formaal.

Saadanne anvendes da ogsaa ved flere GIER-installationer. Herved opnaas at man med et minimum af ulejlighed kan arrangere sin resultatudskrift paa en pæn og praktisk maade.

Nedenfor beskrives en administrationsalgoritme, der er kaldt ADM. I resten af lærebogen forudsættes det, at denne benyttes i forbindelse med trykning af resultater, hvorfor gennemgangen af en række specielle standard-

funktioner til trykning bliver unødvendig og er udeladt. Den læser, der maatte ønske at lære disse nærmere at kende, maa henvises til appendix.

ADM har følgende egenskaber:

1) Programmet skrives paa normal maade som en blok. Denne blok omgives af en ydre blok, der er ADM. ADM bestaar altsaa af to dele, een der kopieres ind paa hulstrimmelen foran programmet, og een der kopieres ind efter programmet. I den første del af ADM indgaar programmets navn og nummer. Disse maa naturligvis ændres fra program til program ved indkopieringen.

2) Naar vi anvender et program, der benytter ADM, vil regnemaskinen begynde med at løbe ind i den første del af denne. Herved trykkes de nødvendige symboler i begyndelsen af hulstrimmelen. Dernæst læser ADM fire tal fra datastrimmelen. Disse tal skal være opgavens nummer og datoen, bestaaende af tre tal, dag, maaned og aar. Datastrimmelen skal altsaa begynde med disse fire tal. (Det forlanges at opgavenummeret er ≤ 999.)

ADM trykker nu en overskrift paa papiret, bestaaende af programmets nummer, opgavens nummer, datoen og programmets navn.

Et eksempel paa en saadan overskrift kan være følgende:

Program nr. 38 - Opgave nr.218 - 27.11.1963
Løsning af komplex ligning A × x↑2 + B × x + C = 0;

3) ADM kontrollerer nu at $1963 \leq$ aarstal ≤ 1968. Hvis dette ikke er tilfældet stoppes efter fejludskrift, saaledes at operatøren kan faa lejlighed til at gribe ind.

4) ADM gemmer værdien af ''drumplace''. Betydningen heraf fremgaar af næste kapitel.

5) Maskinen fortsætter nu ned i det egentlige program. Her maa man definere læsning af parametre, numeriske beregninger og trykning af resultater.

Ved trykning kan vi benytte 3 forskellige standardprocedurer og 2 procedurer i ADM, hvilket vil være tilstrækkeligt til en fuldstændig kontrol af trykningen.

De tre standardprocedurer er

output(), der anvendes til taltrykning
outsp(), der anvendes til trykning af mellemrum
outtext(), der anvendes til trykning af tekst

De to ADM-procedurer er

CR(), der anvendes ved skift til en ny linie og
head, der anvendes ved skift til en ny side.

6) Hvis maskinen under trykningen af resultaterne kommer for langt ned paa en side, vil ADM automatisk skifte til en ny side, med trykning af overskrift og sidetal.

7) Naar beregning og trykning er afsluttet, passerer maskinen ud gennem end i det egentlige program og ind i sidste del af ADM. Denne beordrer indlæsning af det næste tal paa datastrimmelen og betragter det som et nyt opgavenummer. Hvis nummeret er ≥ 0 indlæses ny dato, der skiftes til næste side, drumplace retableres, og maskinen gennemfører beregningen af den nye opgave med nye talværdier. Hvis derimod nummeret er < 0, betragtes beregningen som afsluttet, de nødvendige slutsymboler skrives paa resultatstrimmelen, og maskinen stopper, klar til næste ALGOL-program.

Hvis man under beregningerne konstaterer, at det er umuligt at komme længere i programmet, kan beregningen sluttes straks ved et hop til ,,end of program,, , en label i ADM-algoritmen.
Hvis man konstaterer en fejl, der betyder, at datastrimmelen skal føres frem til næste opgave, kan man udskrive en passende meddelelse til operatøren paa skrivemaskinen(se tekstrykning) og stoppe maskinen ved et kald af ADM-proceduren ,,stop,, .

8) Hvis det ønskes, kan man naar som helst lade regnemaskinen læse resultatstrimmelen til kontrol af, om der er sket nogen fejl i trykningen af resultaterne.

9) Flexowriteren udskriver resultatstrimmelen, nøjagtig som den er defineret i programmet, og stopper udskriften, naar strimmelen er slut.

De fem trykprocedurer har følgende egenskaber:

Sideskift

Procedurekaldet

head;

bevirker at papiret føres frem til næste side, hvor der trykkes overskrift og sidenummer. Proceduren benyttes, hvis man ikke ønsker at det automatiske sideskift skal træde i funktion midt i en sammenhørende udskrift af resultater.

Efter et sideskift kan man uden nogen formaliteter begynde en resultatudskrift, idet maskinen er klar til trykning i venstre margin af papiret.

Der kan da skrives 57 linier, før maskinen automatisk skifter til næste side, idet 5 linier sta r blanke forneden paa papiret.

ADM indeholder en variabel, kaldet linecounter, der altid indeholder det resterende antal linier pa en side, d.v.s. at den varierer mellem 62 og 6.

Tværs over papiret, fra venstre papirkant til højre papirkant er der 99 anslag. Normalt benyttes 13 til venstre margin og der bør benyttes 10 til højre margin, saaledes at man har 76 anslag til disposition.

Ny linie.

Procedurekaldet

CR(n);

bevirker at papiret føres tilbage til venstre margin og føres n linier frem. n betragtes som et heltal, og vi kan paa sædvanlig maade erstatte n med et aritmetisk udtryk.

Trykning af mellemrum.

Procedurekaldet

outsp(n);

bevirker en trykning af mellemslag eller blanke symboler. n betragtes som et heltal, og vi kan pa sædvanlig maade erstatte n med et aritmetisk udtryk.

Teksttrykning.

Procedurekaldet

outtext(‡<Dette kopieres eksakt 2‡);

bevirker at regnemaskinen trykker alt, hvad der staar mellem ‡< og ‡, nøjagtig som det staar skrevet paa papiret og blev hullet pa Flexowriteren.

Der maa kun trykkes een linie tekst i hvert procedurekald, og der maa ikke forekomme skift til en ny linie mellem ‡< og ‡. Hvis man overtræder disse regler, vil linietælling og sideskift ikke virke korrekt.

Man maa her gøre sig klart, at al trykning foregaar fra den øjeblikkelige position pa papiret, saaledes at det vil være forhistorien, der bestemmer, hvor i linien en trykning foregaar. Hvis man har glemt at skrive de nødvendige CR(), vil resten af trykningen saledes foregaa uden for den højre papirkant.

En vanskelighed i forbindelse med anvendelsen af outtext() er, at man i manuskriptet skal meddele sit ønske om, hvormange mellemrum der skal benyttes i teksten. Dette kan f.eks. have betydning i forbindelse med hoveder paa tabeller. Problemet kan løses ved anvendelse af særlige papirark udarbejdet til forma let. En anden mulighed er at skrive ⊥ i stedet for mellemrum. Dette er det eneste symbol, der ikke direkte bliver kopieret, i stedet trykkes mellemrum. Hvis en fejludskrift til operatøren er nødvendig, benyttes writetext(‹< ›); i stedet for outtext.

Taltrykning, layout.

Standardproceduren output anvendes til trykning af tal. Et typisk procedurekald kunne være

output(‹-n.dd10-dd›, a);

der bevirker at talværdien af variabel a trykkes i overensstemmelse med det layout, der opgives som første parameter. I et saadant layout repræsenterer hvert symbol et anslag og er en symbolsk beskrivelse af, hvordan trykningen skal foregaa. Nedenfor følger nogle eksempler paa, hvordan tal trykkes med dette layout:

0	0.00
1	1.00
10	1.00 10 1
-7.655$_{10}$-18	-7.66$_{10}$-18
3.712$_{10}$11	3.71$_{10}$ 11

Vi ser at symbolikken i et saadant layout er følgende:
- betyder, at der skrives et minus, hvis der er et. Hvis tallet er positivt, rykkes det en plads til højre, saaledes at komma staar under komma.
Dernæst følger 3 cifre med komma efter det første ciffer (svarende til n.dd).
Nu følger eventuelt en exponentdel, hvor det samme forhold med minustegnet gælder. (Det første og kun det første symbol er n. Resten af symbolerne skrives som d.)

Følgende 3 layout vil dække de fleste formål:

{-n.___10-dd}. Stregerne efter n betyder at vi kan sætte sa mange d vi ønsker. Hvert d repræsenterer et ciffer. Udskriften er som vist ovenfor.

{-n____}. Stregerne efter n betyder, at vi kan sætte sa mange d, vi ønsker. Hvert d repræsenterer et ciffer.

Layout {-nddd} vil eksempelvis have følgende virkning:

0	0
0.1	0
1	1
1.2	1
-1	-1
3727.6	3728
11259	1126101

Det ses, at vi i dette tilfælde højst anvender 4 cifre foran kommaet. Hvis tallet er for stort anvendes stadig fire cifre, men der tilføjes en exponent. Dette vil normalt være ubehageligt, idet der ikke som i det første layout er sat plads af til en exponentdel, hvorfor resten af linien vil blive forskubbet i forhold til det forventede.

{-n___.___}. Stregerne efter n betyder, at vi kan sætte saa mange d, vi ønsker. Hvert d repræsenterer et ciffer. Det ses, at vi her inkluderer et komma. Bortset herfra svarer layoutet i enhver henseende til det sidste layout.

Layout {-nd.dd} vil eksempelvis have følgende virkning:

0	0.00
0.1	0.10
0.173	0.17
-3.125	-3.13
-375.43	-37.54101
18.163	18.16

Taltrykning, parametre.

Efter layout, et følger en række parametre. Vi kunne godt have skrevet:

output ({-n.dd10-dd}, a,b,c);

Her havde vi fa et trykt talværdien af de tre variable lige efter hinanden. Det er værd at bemærke, at parametrene (her a,b,c) paa sædvanlig maade kan være aritmetiske udtryk.

En speciel egenskab ved proceduren output er imidlertid, at vi ind imellem disse parametre kan anbringe følgende trykprocedurer:

output ( ), outtext( ), outsp ( ).

Enhver af disse trykprocedurer kan ogsaa staa alene og danne en sætning.

Eksempel:

Vi gentager nu programmet fra afsnit 20, til løsning af 2.grads ligninger, idet vi indføjer alle nødvendige sætninger til trykning af resultater.

Den egentlige programblok er omgivet af en ADM-algoritme, ADM-1B. Denne vil til dels være ulæselig, idet den indeholder trykprocedurer, der ikke har været nævnt. Programnummer og navnet paa programmet er omgivet af en bølgelinie. Det er disse størrelser, der skal udskiftes, naar ADM benyttes til andre programmer.

```
comment komplex 2.grads ligning - 22 februar 1964;
begin comment: A.E.K. - ADM 1B - February 14th 1964;
integer pagecounter, linecounter, problem no, day, month, year, drum, a;

procedure head; CR(100);

procedure stop;
begin integer a;
writetext(‡<
stop‡);
a := typechar;
if a ≥ 128 then a := a - 128;
if a = 50 then go to start;
if a = 53 then go to end of program;
end of stop;

procedure CR(a);
value a; integer a;

begin
if linecounter - 6 < a then a := linecounter + 2;
linecounter := linecounter - a;
for a := a - 1 step -1 until 0 do outcr;
```

```
if linecounter < 0 then begin
pagecounter := pagecounter + 1;
linecounter := linecounter + 64;

if pagecounter > 1 then
begin outsp(32); output(¢-ddd¢, -pagecounter, outtext(¢<-¢)) end;
outtext(¢<

A.E.K. - Program nr. 38 - Opgave nr. ¢);
output(¢ndd¢,problem no);
outtext (¢< - ¢);
output(¢nd¢, day, outtext (¢<.¢), month);outtext(¢<.¢); output(¢nddd¢,year);

comment NOW ONE LINE TO BE PRINTED IN EACH HEADING CAN BE WRITTEN;

outtext(¢<
Løsning af komplex ligning A × x↑2 + B × x + C = 0;



¢)
end of linecounter <0
end of CR;

drum := drumplace;
linecounter := 0;
for a := 1 step 1 until 30 do outchar(112); outclear;

start:
drumplace := drum;
pagecounter := 0;
input(problem no);
if problem no < 0 then go to end of program;
input(day,month,year);
for a := 1 step 1 until 30 do outchar(112);
head;
```

```
if year < 1963 ∨ year > 1968 then begin
writetext(‹<
err›); write(‹-nddd10-dd›, year); writetext(‹<< i opg.›); write(‹ndd›, problem no);
stop;
end of test;
comment NOW COMES THE PROGRAM (her slutter 1.del af ADM-1B);

begin
real a, b, c, Z1r, Z1i, Z2r, Z2i, test;

procedure EQ2CR(A, B, C, z1r, z1i, z2r, z2i, INDETERMINATE);
value A, B, C;
real A, B, C, z1r, z1i, z2r, z2i;
label INDETERMINATE;
begin
her skrives blokken fra proceduren i afsnit 17
end of EQ2CR;
writetext(
‹<< A          B          C                x1 , y1                x2 , y2 ›);
CR(1);
start:
CR(1);
input (a, b, c);
output(‹-n.dd10-dd›, a, outsp(3), b, outsp(3), c);
outsp(4);
EQ2CR(a, b, c, Z1r, Z1i, Z2r, Z2i, Fejl);
output(‹-n.dd10-dd›, Z1r, outsp(2), Z1i, outsp(4), Z2r, outsp(2), Z2i);
nyt datasæt:
input(test);
if test ≥ 0 then go to start else go to slut;

Fejl:
outtext(‹<UDEFINERET›);
go to nyt datasæt;
```

```
slut:
end of program;

comment THE FOLLOWING WILL BE INSERTED AFTER THE PROGRAM (2.del af ADM-1B);
go to start;

end of program:
outsum; for a := 1 step 1 until 60 do outchar(112);
end of program;
```

Programmet fungerer pa følgende made:
Maskinen løber igennem det første begin og 1.del af ADM, hvor der læses fire tal og trykkes overskrift etc.

Nu fortsætter maskinen ud i det egentlige program, hvor de nødvendige deklarationer finder sted, løber uden om procedure EQ2CR, trykker overskrift til tabellen over indgangsparametre og resultater og skifter papiret 2 linier frem. Nu læses a, b og c og trykkes ud pa papiret, idet vi som parametre til output foruden a, b og c ogsaa har brugt outsp (3), sa ledes at der kommer en passende afstand mellem resultaterne. Det anvendte layout og teksten i tabelhovedet korresponderer nøje med hinanden.

Efter udskriften af indgangskonstanter fortsætter maskinen med procedurekaldet til beregning af resultaterne, hvorefter disse trykkes ud pa analog ma de.

Nu læses et nyt tal, der er bestemmende for, om beregningen er slut, hvis dette er tilfældet hoppes til slut, maskinen løber ud gennem end of program og ind i 2. del af ADM, hvor der læses endnu et tal, der er bestemmende for, om en ny opgave skal udføres, eller om beregningerne er definitivt slut.

Hvis et talsæt a, b og c er af en sadan art, at vi fa r udhop fra proceduren via den formelle parameter INDETERMINATE, sker dette udhop i virkeligheden til den aktuelle parameter, label Fejl, hvor teksten UDEFINERET trykkes, hvorefter næste datasæt indlæses.

Eksempel paa datastrimmel.

Alt, hvad der følger herefter, kunne være skrevet paa en datastrimmel til ovenstaaende program. Tallene svarer til de tal, vi kontrollerede algoritmen med tidligere i afsnit 14.

```
 1,      22/2,  1963
 A,      B,     C,      test
 0,      0,     2,        0
 0,      4,     8,        0
 2,      0,    -8,        0
 1,    -10,     9,        0
-1,    +10,    -9,       -1
 2,     22/2,  1963
-1,    -4,    -4,         0
 2,    -8,    26,         0
 4,     0,     0,        -1
-1,
```

Bemærk, at det sidste tal skal afsluttes med et skilletegn.

Det vil ofte være meget praktisk og øge driftsikkerheden af et givet program, hvis man udarbejder et skema, i hvilket man kan indføre sine indgangskonstanter. Et eksempel paa et saadant skema er vist paa næste side.

Udskrift af resultater.

Resultaterne udskrives paa nøjagtig den form, vi har defineret i programmet, saaledes som det ses paa side 19 og 20parametrene er delt i to sæt, svarende til to forskellige opgaver, saa det automatiske sideskift kan demonstreres.

22/2 - 1963

Skema til indgangskonstanter.

Program 38

Løsning af komplex ligning A × x↑2 + B × x + C = 0.

..... opgave nr. (hvis -1 da er beregningerne slut)

...../...../..... dag, maaned, aar

Hvis test er < 0 skiftes til nyt opgave nr.

A	B	C	test
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....
.....	.....	.....	.....

Program nr. 38 - Opgave nr. 1 - 22.2.1963

Løsning af komplex ligning A × x↑2 + B × x + C = 0;

A	B	C	x1	y1	x2	y2
0.00	0.00	2.00	UDEFINERET			
0.00	4.00	8.00	-2.00	0.00	-2.00	0.00
2.00	0.00	-8.00	2.00	0.00	-2.00	0.00
1.00	-1.00 10 1	9.00	9.00	0.00	1.00	0.00
-1.00	1.00 10 1	-9.00	9.00	0.00	1.00	0.00

Program nr. 38 - Opgave nr. 2 - 22.2.1963
Løsning af komplex ligning A × x↑2 + B × x + C = 0;

A	B	C	x1 , y1		x2 , y2	
-1.00	-4.00	-4.00	-2.00	0.00	-2.00	0.00
2.00	-8.00	2.60 10 1	2.00	3.00	2.00	-3.00
4.00	0.00	0.00	0.00	0.00	0.00	0.00

Opgave 12:

I afsnit 17 skrev vi en procedure REFA 1 til rodbestemmelse ved hjælp af den modificerede regula falsi metode.

Da man ikke kan være sikker paa, at proceduren er skrevet saaledes, at den udfører de beregninger, man har ønsket, bør den afprøves i et passende program.

Skriv derfor et testprogram til kontrol af, at proceduren fungerer korrekt. REFA 1 skal benyttes til at finde løsningen paa

$$\sin(x) - 0.5 = 0$$

i intervallet 0 til 1 med en nøjagtighed pa først 0.01 og dernæst 0.000001. Sluttelig prøves udgangen WRONG ved beregning af roden af sin(x) i intervallet 0.2 til 1.0. Angiv den forventede udskrift fra dette program.

Naar dette er afsluttet, omskriv da REFA 1 til en ny procedure REFA 2, der virker som REFA 1, men forlanger at funktionen F(z) skal være en real procedure i stedet for et reelt aritmetisk udtryk. Skriv et testprogram til REFA 2, der anvender de samme tests som REFA 1.

Opgave 13:

Nedenfor følger et helt realistisk problem. Skriv det tilsvarende ALGOL-program:
Der ønskes beregnet en tabel paa grundlag af flg. udtryk for l2 og Aber:

l2 = -r×(tg(2×e2)+1/c)/(1-1/c×tg(2×e2)+(tg(2×e2)+1/c)×tg(e2))
hvor
c = r/l1+tg(e1)
(for e2 = -45 grader bliver l2 til r × c/(1+c)).
Aber = - H × r ×(c1 + c2)
hvor
H = l1↑2×l2/(2×r↑3)×(1+(r/l1+tg(e1))↑2)× sqrt(1+(r/l2+tg(e2))↑2)
c1= r↑2× (r/l1+3×tg(e1))/(l1↑2×(1+(r/l1+tg(e1))↑2)↑(3/2))
c2= r↑2× (r/l2+3×tg(e2))/(l2↑2×(1+(r/l2+tg(e2))↑2)↑(3/2))

Parameterværdierne er følgende:
l1 er 50
e1 antager værdierne 0 til 50 grader i skridt paa 5 grader
e2 antager værdierne -20 til -50 grader i skridt paa 10 grader
r antager værdierne 30 til 120 i skridt paa 5.
Resultaterne skal tabelleres i 11 tabeller, een for hver af værdierne af e1, hvis værdi bliver trykt i tabellens hoved. Tabellerne arrangeres paa følgende maade:

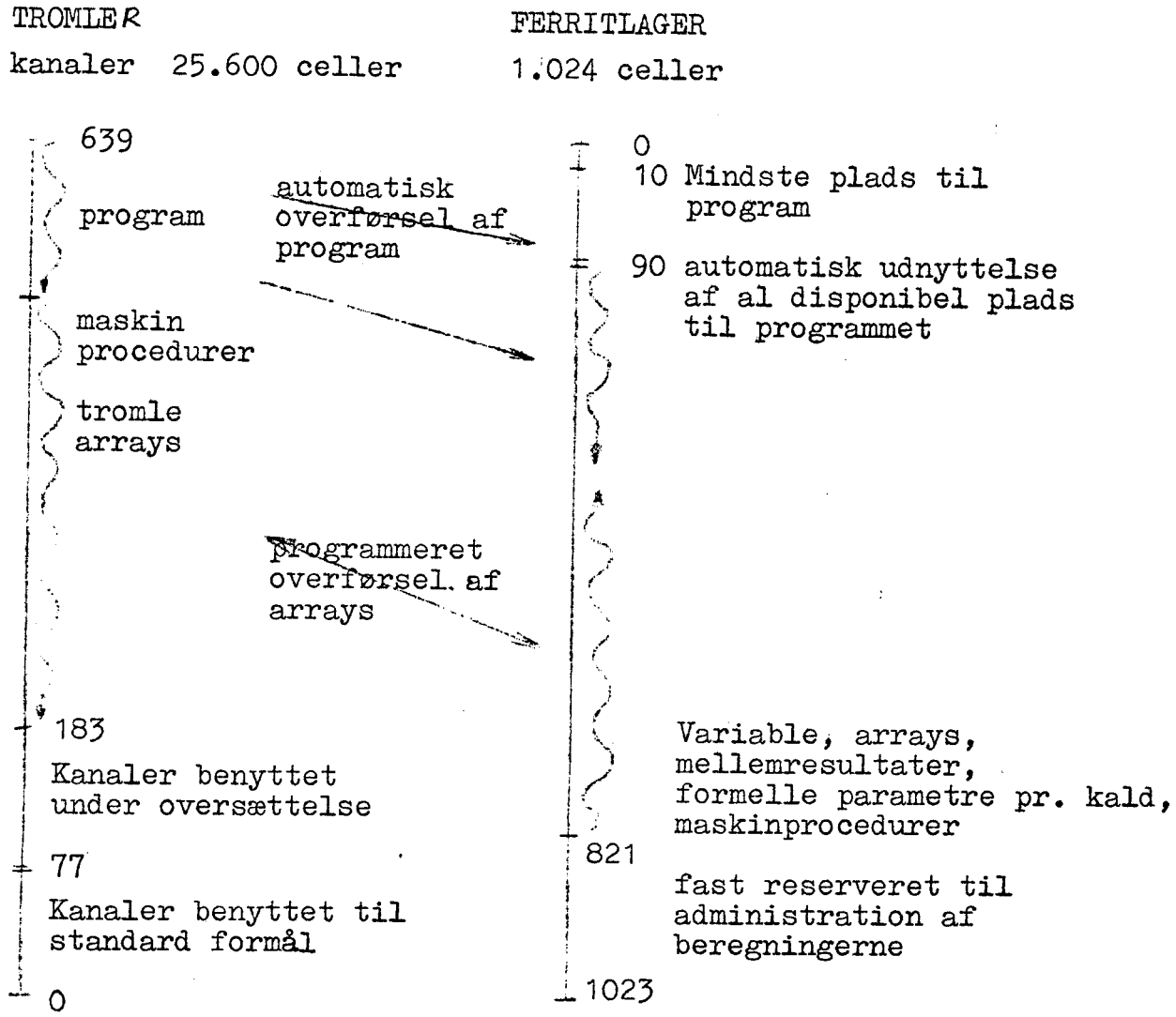
l1 = 50, e1 = 20
e2 = - 20 -30 -40 -50
r l2 Aber l2 Aber o.s.v.
resultaterne er mindre end 1000 og skal trykkes med een decimal.

Programmet skal skrives saaledes, at man anvender en procedure, der kaldes tg og en anden procedure, der kaldes cn, cn benyttes til beregning af c1 og c2. Desuden skal overflødige operander fjernes fra formlerne.

22. Lagring af program og variable.

GIER har i sin grundversion to former for lager. Det ene er arbejdslageret, der bestaar af 1024 lagerceller, hvor hver celle kan rumme eet tal. Det andet er tromlelageret, bestaaende af 640 tromlekanaler, der hver indeholder 40 celler. Tromlelageret rummer altsaa ialt 25600 celler.

GIER-ALGOL benytter de to lagre paa den maade, der er skitseret i nedenstaaende figur.



Oversigt over anvendelsen af lagre i GIER.

Det ses her, at den nederste fjerdedel af tromlelageret indeholder ALGOL-oversætteren under oversættelsen af programmet. Det oversatte program vil blive lagret i den øverste del af tromlelageret.

Naar oversættelsen er afsluttet, er det oversatte program lagret fra en given kanal op til kanal 639 i tromlelageret. (Hvis programmet er saa stort, at det ikke kan være i den disponible del af tromlelageret, vil maskinen skrive ,,program too big,,).

Under beregningerne med det oversatte program, vil en administrationsrutine, der ligger fra celle 821 til 1024 i arbejdslageret, varetage en automatisk overførsel af programmet fra tromlelageret til arbejdslageret.

Pladsreservation i forbindelse med deklarationer, mellemresultater, blokindgange og formelle parameter v.kald, sker fra celle 820 og nedefter. Den resterende plads ned til celle 90 udnyttes automatisk til programmet. Pladsen fra celle 10-89 er den minimale plads, programmet kan optage.

Hvis man prøver et program med arrays af variabel størrelse paa GIER, vil man se, at programmet kører langsommere, jo større disse arrays er, idet der vil ske flere og flere tromleoverførsler, jo mindre plads der er til programmet i arbejdslageret. Der kan her være tale om en faktor af størrelsesorden 10 paa regnetiden. Naar grænsen for de variable passerer celle 90 (d.v.s. at der er plads til ca. 700 tal), kan beregningerne ikke fortsætte, og paa kontrolskrivemaskinen skrives ,,alas,,.

Hvis man har brug for store arrays, kan det derfor være nødvendigt at gemme disse paa tromlen, og kun føre dem over til arbejdslageret, naar de skal benyttes i beregningerne. GIER-ALGOL indeholder saadanne standardprocedurer, der kan benyttes til transport af arrays mellem tromlen og arbejdslageret.

Programmeret flytning af arrays.

I GIER-ALGOL er der to standardprocedurer, ,,to drum,, og ,,from drum,, der overfører data henholdsvis til og fra tromlen. Endvidere er der en heltallig standardvariabel ,,drumplace,, der har med tromle-array administrationen at gøre.

Procedurekaldet

to drum (A);

vil bevirke, at hele array A kopieres over i tromlelageret. Det bemærkes dog, at A stadig eksisterer og optager plads i arbejdslageret. Først naar maskinen forlader den blok, i hvis hoved A blev deklareret, vil A blive slettet fra arbejdslageret, men dette berører ikke tromlelageret.

Korrespondancen mellem tromle og arbejdslager fastlægges paa følgende maade: Kaldets parameter, arraynavnet A, bestemmer hvilken del af ferritlageret, der skal overføres til tromle, og derigennem hvormange celler, der skal overføres. Det sted pa tromlen, hvor overførslen begynder, er bestemt af drumplace. Inden regningerne begynder, far drumplace en talværdi, der svarer til sidste celle pa den sidste ledige kanal i tromlelageret. ADM sørger for, at drumplace altid har denne værdi, nar man begynder par en ny opgave, uanset hvilken værdi den havde, da sidste opgave sluttede. Hvis man ser bort fra, at tromlen fyldes op med tal bagfra, kan man sige at drumplace angiver den første ledige plads par tromlen, naar regningerne begynder. Nar man udfører et kald af ,,to drum,, ændres drumplace, saledes at den ogsa bagefter angiver den første ledige plads par tromlen (to drum og from drum er forøvrigt procedurefunktioner, hvis værdi er lig ændringen i drumplace).

Som følge af denne ændring af drumplace vil nye kald af ,,to drum,, automatisk bevirke lagring paa den første ledige plads par tromlen.

Hvis tromlelagringen passerer et kanal nummer par 183, vil oversætteprogrammet blive ødelagt, og dette ma par ny indlæses før næste oversættelse. (En fejludskrift ,,gone,, gør opmærksom par dette forhold, hvis man har forsømt at indlæse oversætteren inden næste oversættelse).

Hvis tromlelagringen passerer et kanal nummer par 77, vil ogsa standardprocedurer blive ødelagt. I saa fald standser maskinen straks med fejludskriften ,,drum alas,,. Denne fejludskrift fas ogsa, hvis drumplace tilskrives en værdi, der falder inden for det oversatte program.

Læsning fra tromle til arbejdslager sker ved standardproceduren ,,from drum,,.
Procedurekaldet

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

vil saledes bevirke, at elementerne i array A faar tilskrevet nye værdier. A maa være deklareret paa det sted, hvor kaldet star. Afhængig af A,s størrelse og af drumplace,s aktuelle værdi, hentes et bestemt udsnit af tromlen og anbringes i de celler, hvor A star. drumplace ændres pa samme maade som ved to drum. Efter kaldet svarer drumplace altsaa til det næste afsnit af tromlen.

Hvis et array, der tidligere er gemt paa tromlen, skal hentes frem igen, sker det som regel ved, at man inden brugen af ,,to drum,, har haft sætningen d := drumplace; inden ,,from drum,, skrives saa sætningen drumplace := d; d er en integer variabel, der skal være deklareret i en blok, programmet ikke har forladt i mellemtiden.

En fornuftig anvendelse af disse procedurer vil i hvert fald forudsætte, at drumplace, naar man kalder from drum (A), har en værdi, der er den samme, som den havde ved et tidligere kald af to drum (B), hvor A og B er deklareret paa samme maade. Dette kontrolleres ikke af maskinen, og man kan derfor benytte tromleadministrationen paa uortodoks maade ved at beregne en værdi af drumplace ,,midt inde i et array,, eller lignende.

Til støtte herfor kan det oplyses, at to arrays, der indeholder lige mange elementer af samme type fylder det samme. Det er altsaa ikke nødvendigt at de to arrays har de samme grænser, ja end ikke det samme antal indices. Ved ind- og udlæsning til og fra tromle ordnes elementerne paa den maade, der er benyttet ved standardfunktionen input ( ).

Hvis hvert nyt array lagres paa et helt antal kanaler, gaar kanaltransporten hurtigst. Man kan derfor spare nogen regnetid (paa bekostning af plads paa tromle) ved at udvide alle arrays, der skal mellemlagres paa tromlen, saa de har et antal elementer, der er et helt multiplum af 40.

Eksempel:

I dette eksempel læses to arrays fra hulstrimmel, hvorefter de lagres paa tromlen og ombyttes i arbejdslageret.

```
begin
integer a,b;
real array A[1:20, 1:5, 1:2], B[1:10, 1:20];
    input (A,B);
    a := drumplace;
    to drum (A);
L1: b := drumplace;
    to drum (B);
    drumplace := a;
    from drum (B);
L2: drumplace := b;
    from drum (A);
end;
```

Det bemærkes, at sætning L1 og L2 er unødvendige, og det ses at værdien af A[1,2,1] er ombyttet med værdien af B[1,3] og at A[3,1,1] vil korrespondere med B[2,1].

23. Reserverede navne og biblioteksprocedurer.

Følgende navne er reserveret i GIER-ALGOL III til standard-funktioner:

abs	input	setchar
arctan	kbon	sign
char	ln	sin
cos	lyn	split
drumplace	outchar	sqrt
entier	outclear	to drum
exp	outcopy	typechar
from drum	outcr	typein
gier	output	write
gierdrum	outsp	writechar
gierproc	outsum	writecopy
inchar	outtext	writecr
inone	pack	writetext

Med ordet reserveret menes blot, at navnene betragtes som deklareret i en blok uden om programmet, saaledes at man godt kan deklarere disse navne med en anden betydning inde i programmet.

Som det ses, er kun en del af disse funktioner beskrevet i de foregaaende afsnit; resten beskrives i appendix.

Udover de faste standardfunktioner findes en række biblioteksprocedurer og programstykker, der er udformet saa alment, at de uden videre kan kobles ind i programmerne. Som eksempel herpaa har vi den tidligere nævnte algoritme ADM. Yderligere vil de nødvendige matematiske funktioner og numeriske metoder foreligge i procedureform. Som eksempel herpaa har vi Besselfunktioner, exponentialintegral, løsning af lineære ligninger, determinantberegning, numerisk integration, løsning af sammenhørende differentialligninger etc.

I forbindelse med de forskellige GIER-installationer, vil der være et procedurebibliotek, hvor saadanne algoritmer foreligger fuldt beskrevet og afprøvet, saaledes at man simpelthen kan faa de paa gældende hulstrimler kopieret ind paa sin programstrimmel.

Det er naturligvis fordelagtigt at benytte biblioteksprocedurer, hvor det er muligt, idet der ofte er investeret et stort arbejde i at gøre disse saa sikre og effektive som muligt.

Ved Risø GIER forlanges det, at man f.eks. skal skrive

comment library BESS 1, DETERMINANT;

pa det sted, hvor man ønsker at biblioteksprocedurerne BESS 1 og DETERMINANT skal kopieres ind i programmet. Denne kopiering vil da blive foretaget.

Heraf vil det fremgaa, at comment library kun maa skrives paa et sted, hvor en proceduredeklaration er tilladt.

Særlige problemer ved opbygning af programmer.

Ud over hvad der er nævnt tidligere i dette afsnit, kan det anbefales, at man søger kontakt med regnemaskinegruppen, naar man staar overfor at skulle lave et ALGOL-program.

Her vil man kunne faa oplysning om, hvorvidt der er andre, der helt eller delvis arbejder med de samme problemer, eller om problemet maaske allerede foreligger kodet her eller andetsteds. Yderligere kan regnemaskinegruppen eventuelt overtage visse dele af programmeringsarbejdet, hvor disse dele enten har almen interesse eller kræver særlig behandling, og raadgive i forbindelse med matematiske og numeriske problemer.

Endelig kan der være særlige problemer af mere praktisk art, hvor der f.eks. er tale om store datamængder, parameterstudier eller problemer der indeholder arrays af konstanter. Her kan regnemaskinegruppens praktiske erfaring være til nytte, saaledes at den manuelle behandling af data og resultater kan blive væsentlig lettet.

A6. Standardfunktioner der vedrører læsning.

GIER-ALGOL indeholder en hel række af standardfunktioner, der kan benyttes til indlæsning af information til et kørende program. I afsnit A9 (i selve lærebogen) er der givet en oversigt over de forskellige hulsymboler. Disse vil derfor ikke blive defineret nærmere her.

GIER kan indlæse information fra hulstrimmel ved hjælp af in-standardfunktioner og fra skrivemaskine ved hjælp af type-standardfunktioner.

Uanset hvilken af disse læsende standardfunktioner der benyttes, vil læsningen foregaa via en fælles-sorterer. Dette gælder dog ikke "lyn".

Fælles-sortereren vil udelade nogle symboler, lade andre faa en særlig virkning, og lade resten af symbolerne slippe igennem til den aktuelle læsende standardfunktion.

Naar vi i forbindelse med en saadan standardfunktion omtaler symboler, er det altsaa kun de symboler, der er passeret igennem fælles-sortereren.

Fælles-sortererens virkning.

1) Blinde symboler overspringes. Disse er:

Blank strimmel	| . |
Tape feed	| oooo.ooo |
Værdi 127	|ooo o.ooo |
Alle huller	|ooooo.ooo |

2) Alt imellem og inklusive PUNCH OFF og PUNCH ON overspringes. Et PUNCH ON uden forudgaaende PUNCH OFF betragtes som et normalt symbol, der passerer fælles-sortereren.

3) END CODE stopper GIER efter skrivemaskineudskriften

pause

Hvis en af tasterne paa skrivemaskinen derefter trykkes ned, fortsættes indlæsningen, og symbolet betragtes som blindt.

4) Paritetsfejl (d.v.s. et lige antal huller) stopper GIER, og betragtes som en fejl.

5) CLEAR CODE nul-stiller sumcellen for læsning og betragtes iøvrigt som blindt symbol. (se afsnit A7)

6) SUM CODE bevirker, at det næste symbol paa strimmelen sammenlignes med sumcellen for læsning. Ved overensstemmelse betragtes de to symboler som blinde symboler. Ved uoverensstemmelse skriver GIER med rød skrift paa kontrolskrivemaskinen

sum fails

og stopper. Naar en af tasterne paa skrivemaskinen trykkes ned, fortsætter indlæsningen. (Se afsnit A7)

7) CASE-symboler indstiller en intern CASE-celle og behandles iøvrigt som blinde symboler. Ved start er den interne CASE-celle indstillet til LOWER CASE. Ethvert symbol opfattes overensstemmende med CASE-cellens indhold.

char.

char er en integer procedure uden parametre, hvis værdi er talværdien for det sidst læste symbol. Talværdien af et symbol er det tal, der er nævnt i parentes i afsnit A9, for saa vidt symbolet er i LOWER CASE, ellers talværdien + 128. char tager dog ikke hensyn til tegn, der er læst med lyn. char aktiverer ikke nogen indlæseenhed.

setchar.

Procedurekaldet

setchar (160)

vil bevirke, at det første tegn der læses næste gang en læsende procedure bliver aktiveret, vil være det tegn, der korresponderer med 160 (se char), altsaa +, idet dette tegn indskydes før den egentlige læsning fra hulstrimmel eller skrivemaskine.

Værdien af den aktuelle parameter skal svare til et tegn, der kan passere fælles-sortereren.

setchar aktiverer ikke nogen indlæseenhed.

inchar, typechar.

inchar og typechar er begge integer procedure uden parametre. Procedurekaldet

a := inchar;

vil aktivere strimmellæseren, der læser indtil et symbol har passeret fælles-sorteren. a faar en talværdi, der svarer til dette symbol. (se char)

typechar har en tilsvarende virkning.

lyn.

lyn er en integer procedure, der helt svarer til inchar, idet det læste tegn dog ikke passerer fælles-sortereren.

outcopy, writecopy.

outcopy og writecopy kopierer et afsnit af datastrimmelen.

Udskriften sker henholdsvis paa hulstrimmel eller skrivemaskine og omfatter kun de symboler, der passerer gennem fælles-sortereren, samt de nødvendige case-tegn.

Som aktuel parameter i disse procedurekald skal der anvendes en tekststreng bestaaende af eet eller to symboler. Hvis eet symbol anvendes, vil hulstrimmelen blive kopieret indtil indlæsning af det tilsvarende symbol har fundet sted, hvis to symboler anvendes, indlæses hulstrimmelen til og med det første symbol, hvorefter den kopieres indtil det næste symbol. Procedurekaldet

outcopy (‡<[ ]‡);

vil saaledes ved indlæsning af en hulstrimmel, hvorpaa der staar

Heading: [Problem number:]

resultere i en kopiering af

Problem number:

medens procedurekaldet

outcopy (‡<]‡);

ville have resulteret i en kopiering af

Heading: [Problem number:

Det bemærkes, at det er nødvendigt, at styresymboler i procedurekaldet og paa hulstrimmelen er i samme case, og at der ikke forekommer overflødige mellemslag imellem styresymbolerne.

input, inone, typein.

input er nævnt i afsnit 20. inone og typein er real procedures uden parametre, der indlæser eet tal.

De ovennævnte tre procedurer behandler MELLEMRUM, og _som blinde symboler.

Informations-symboler er cifre, ., 10, +, -.

Tallene skal skrives efter de konventioner, der er nævnt i afsnit 5 og 8.

Skilletegn mellem tallene (eet er nok, men der kan godt være flere) er alle andre symboler, nævnt i afsnit A9, for saa vidt de passerer igennem fælles-sortereren.

Hvis man under indlæsningen ønsker at lade talværdien af en indlæst variabel være uændret, kan man undlade at skrive det tilsvarende tal paa hulstrimmelen, og i stedet skrive en ,,input ditto'' der simpelthen er et minustegn.

Saaledes gælder det, at procedurekaldet,

input (a, b, c);

der læser tre tal fra hulstrimmel, vil lade b være uændret, hvis der paa hulstrimmelen staar

3.1415, -, 2.71828,

Hvis man bruger denne mulighed, maa man blot passe, at man ikke har forladt den blok, hvori b er blevet deklareret, idet talværdien for b i saa fald er gaaet tabt.

Input ditto er udefineret, hvis symbolet indlæses med procedurerne inone eller typein.

Hvis et tal er grammatisk ukorrekt, f.eks. 3.3.3, vil GIER indlæse resten af tallet inklusive skilletegnet, hvorefter der med rødt farvebaand udskrives correct input value, end in LC(UC):
paa kontrolskrivemaskinen. Man kan nu skrive det ønskede tal paa skrivemaskinen og slutte i UPPER CASE eller LOWER CASE, saaledes som GIER har forlangt, hvorefter indlæsningen vil fortsætte.

Eksempel:

```
a := inone;
input (V);
for i := inone step inone until V do .....
if typein < 0 then go to finis;
```

A7. Standardfunktioner der vedrører trykning.

GIER-ALGOL indeholder en række out-standardfunktioner, der kan benyttes til trykning af resultater paa hulstrimmel. De fleste af disse findes ogsaa som write-standardfunktioner, der bevirker udskrivning af resultater paa skrivemaskine.

Man vil normalt benytte out-funktionen til resultatudskrift, idet udskriften af resultater paa hulstrimmel er 15 gange hurtigere end skrivemaskineudskrift. Skrivemaskinen benyttes derfor kun til at give særlige oplysninger til operatøren. Enhver output-funktion forudsætter at flexowriter eller skrivemaskine er i LOWER CASE, og efterlader den i LOWER CASE.

Sumkontrol. outclear, outsum.

Det gælder for alle out-standardfunktioner, at talværdien af ethvert symbol, der udskrives paa hulstrimmel, adderes til en særlig sum-kontrol-celle. Procedurekaldet

outclear;

bevirker, at denne sumcelle 0-stilles, og at der hulles et CLEAR CODE symbol paa hulstrimmelen. Ved en senere kontrolindlæsning af strimmelen, vil CLEAR CODE bevirke 0-stilling af sumcellen for læsning.

outsum;

bevirker, at der hulles et STOP CODE symbol, et SUM CODE symbol og et symbol der repræsenterer talværdien af de symboler, der er blevet hullet, efter at programmet er læst ind i maskinen, efter det sidste outclear eller efter det sidste outsum. Saavel outclear som outsum bevirker altsaa en 0-stilling af sumcellen. Ved en senere kontrolindlæsning af hulstrimmelen vil SUM CODE symbolet bevirke en automatisk sammenligning af den næste karakter paa hulstrimmelen med sum-cellen for læsning, saaledes at det kan kontrolleres, om GIER har hullet de rigtige symboler. Sum-kontrollen kan ogsaa benyttes i forbindelse med indlæsning af biblioteksdatastrimler, der f.eks. indeholder materialekonstanter, saaledes at man har sikkerhed for at læsningen forløber korrekt. Hvis operatøren griber ind i valget af udskriftsenhed, giver sumkontrollen forkerte resultater.

outsp.

```
outsp (n);
```

bevirker trykning af n MELLEMRUM. Hvis n $\leq$ 0 springes trykningen over.

outcr, writecr.

```
outcr;
```

bevirker trykning af eet CAR.RETURN symbol. Symbolet betyder:
Før vognen paa Flexowriteren retur til venstre margin og skift til næste linie.

outcopy, writecopy.

Disse procedurer er nævnt i afsnit A6.

outchar, writechar.

```
outchar (n);
```

trykker eet symbol med talværdi n. (se afsnit A9). n betragtes som heltal, afrundes om fornødent og kan være et aritmetisk udtryk. Hvis der ønskes trykt symboler i UPPERCASE, maa der først trykkes et UPPER CASE SYMBOL, og før man trykker med andre output-procedurer, maa man trykke et LOWER CASE.

outtext, writetext.

```
outtext(‡<
a,b,c‡);
```

bevirker en exakt kopiering af tekststrengen, der kan indeholde alle symboler, saavel de der er kendt fra afsnit A9 som alle andre hulkombinationer.

I det ovenstaaende eksempel starter trykningen derfor med en CAR.RETURN og fortsætter med a,b,c.

Som det er nævnt i afsnit A5, kan procedurekaldet indeholde et vilkaarligt antal parametre, formelle saavel som aktuelle.

Forekommer symbolet $\perp$ i tekststrengen, vil det ikke blive kopieret, men i stedet trykt som MELLEMRUM.

output, write.

output(⟨n.dd⟩, a, b);

bevirker trykning af talværdien af a og b i overensstemmelse med det layout, der opgives som første parameter. Proceduren er delvis gennemgået i afsnit 21, som maa gennemlæses. (Se ogsaa afsnit A5).

output kan anvende et vilkaarligt antal parametre. Disse kan være en hvilken som helst af de ovennævnte out- og writeprocedurer (ogsaa output() og write()), eller aritmetiske udtryk.

Et eksempel paa et layout er følgende:

⟨-n␣dd.d00₁₀+d⟩

Hvert symbol betyder eet anslag. Fortegnet foran layoutet og i den eventuelle eksponentdel har følgende betydning:

- betyder at der trykkes - hvis tallet er negativt og MELLEMRUM, hvis det er positivt.

+ betyder at fortegnet altid trykkes.

Hvis der ikke angives fortegn vil der ikke blive sat plads af til det, men det vil blive trykt, hvis tallet er negativt.

± Fortegnet trykkes altid som første anslag, i modsætning til de tre foregaaende tilfælde, hvor det trykkes lige foran tallet.

Det totale antal af n og d er det maksimale antal betydende cifre, hvis der ikke forekommer en eksponentdel i layoutet, og det virkelige antal, hvis der forekommer en eksponentdel.

n kan kun anvendes som det første bogstav og betyder at et 0 foran det decimale komma skal trykkes, medens et saadant vil blive udeladt, hvis d er det første symbol.

Der kan indsættes mellemrum i layoutet mellem n, d og 0 eller det ækvivalente symbol ␣. I begge tilfælde vil mellemrum blive kopieret ved trykningen.

0 kan kun forekomme ved afslutningen af et layout.

I det ovenfor viste eksempel er betydningen af 00, at resultaterne trykkes med fire betydende cifre, saaledes at der er maximalt tre foran kommaet eller tre efter kommaet. Afhængig af tallets størrelsesorden, vil det altsaa blive trykt indenfor de angivne seks positioner dannet af n, d og 0. Forekommer der m nuller i et layout, vil en eventuel eksponent være et tal der er deleligt med m+1.

En eksponentdel kan ikke anvendes uden en taldel og kan ikke indeholde n, 0 eller ±. Hvis eksponenten er 0, trykkes mellemrum. 10 bliver trykt umiddelbart foran fortegnet.

Hvis et tal er for lille til at blive trykt med et layout, der ikke indeholder en eksponentdel, vil der blive trykt 0,er. Hvis det er for stort til et givet layout, der ikke indeholder eksponentdel, vil det blive trykt med dette layout og en passende eksponentdel.

Der er begrænsninger paa det totale antal af n, d, 0 og ±, men disse er ikke nogen hindring for en normal anvendelse af layouts.

Alle tal afrundes korrekt til det antal betydende cifre, der trykkes.

A8. Diverse.

Kapitlet i lærebogen udvides med følgende:

kbon

kbon er en Boolean procedure. Den er true, naar KB lampen paa kontrolbordet er tændt, ellers false. Herved er der mulighed for fra kontrolbordet at gribe ind i beregningernes gang. Se eksemplet side a13.

pack, split

Hver enkelt variabel er som bekendt lagret i en celle i maskinen, og hver celle bestaar af 42 bit, der hver kan have værdien 0 eller 1. Medens der i den del af ALGOL, der hidtil har været behandlet, kun har været mulighed for at behandle et celleindhold som en helhed, er det med Boolean procedure pack og integer procedure split gjort muligt at arbejde med selve bitmønstret.

Lad os antage, at vi ønsker at indsætte følgende bit ,,1100100,, i positionerne 0 til 6 incl. i en Boolean variabel, der hedder mønster, medens resten af ,,mønster,, skal være uforandret. Det gøres med procedurekaldet pack (mønster,0,6,100), idet de viste bit opfattet som et heltal skrevet i 2-talsystemet netop har værdien 100.

Af pack,s aktuelle parametre skal den første altsaa være navnet paa den variable, der skal ændres. Derefter kommer et sæt paa tre parametre, henholdsvis første bit, sidste bit, og et heltal, der angiver, hvad der skal indsættes. De tre parametre kan paa sædvanlig maade skrives som aritmetiske udtryk, der om fornødent bliver automatisk afrundede.

Der kan være flere sæt paa tre parametre i samme kald. De udføres fra venstre til højre, og hvis de dækker ind over hinanden, er det saaledes det sidste, der bliver staaende. Procedurefunktionens værdi er lig det færdige, samlede bitmønster i den variable, der er første parameter, dog paa nær bit 40 og 41, der er udefinerede.

integer procedure split udfører den modsatte funktion af pack. Hvis vi efter det ovenstaaende eksempel pack (mønster,0,6,100) udfører kaldet split (mønster,2,5,i), hvor i er deklareret som integer eller real, far i tilskrevet værdien 2, svarende til bitsammenstillingen ,,0010,,.

Ogsaa ved split kan der følge et vilkaarligt antal sæt paa tre parametre. Af disse tre kan de to første være aritmetiske udtryk. Procedurefunktionens værdi er lig den værdi, den sidste parameter faar.

Den første parameter i split skal være en boolesk variabel eller et boolesk udtryk. Ved booleske udtryk virker de logiske operatorer paa hver enkelt af bittene 0-39, idet 1 svarer til false og 0 til true. Bit 40 og 41 er ikke definerede ved udtryk.

A10. Anvendelse af maskinkode.

I GIER-ALGOL III kan procedurefunktioner skrives i maskinkode. En vejledning i at skrive saadanne procedurer falder uden for denne fremstillings rammer, men der kommer til at foreligge biblioteksprocedurer skrevet i maskinkode, og i dette kapitel skal der forklares, hvorledes man bruger dem.

Maskinordrerne fylder en sammenhængende gruppe celler i stakken paa samme maade som et array. Hvis den plads de staar i, af programmet er reserveret ved deklarationen ,,boolean array A[1:m]'' og proceduren i øvrigt har 3 aktuelle parametre, der her kaldes a, b og c, vil ordrerne gennemløbes ved kaldet:

p := gierproc (A[i],a,b,c);

Herved overføres kontrollen til den ordre, der staar i cellen, der indeholder A[i], og maskinen fortsætter med at ,,udføre arrayet'' indtil beregningerne i dette er færdige. Saa fortsættes paa normal maade med den næste ALGOLsætning. Værdien af ,,i'' maa fremgaa af beskrivelsen. Ligeledes fremgaar det naturligvis af denne, hvad de aktuelle parametre er, og hvilken værdi procedurefunktionen har.

Herefter melder spørgsmaalet sig, hvordan man faar proceduren anbragt i maskinen. Det sker i to tempi. Først læses den op paa tromlen, og derefter kaldes den til stakken. Bibliotheksstrimlen med proceduren begynder med en identifikation, enten et tal eller en streng.

I programmet skrives f. eks. sætningen:

gierdrum (777,a);

eller længde := gierdrum ('<BESS>',b);

Naar algolprogrammet kommer til denne sætning starter strimmellæseren, og hvis det første par strimmelen ikke stemmer overens med henholdsvis ,,777'' eller ,,BESS'' skriver maskinen fejlmeldingen ,,gier'' paa skrivemaskinen. Dette sker ogsaa, hvis strimmelens checksum ikke stemmer. Hvis kaldet af ,,gierdrum'' har ,,0'' som første parameter, foretages der dog ingen kontrol af identifikationen.

Maskinproceduren læses nu op paa tromlen til det sted, der angives af ,,drumplace''s aktuelle værdi. Samtidig ændres ,,drumplace'' paa sædvanlig maade. Ændringen i drumplace, der svarer til maskinprocedurens længde, er samtidig procedurefunktionens værdi. Dens numeriske værdi tilskrives den anden parameter, i eksemplet ovenfor ,,a'' eller ,,b''. Efter det sidste af de to viste sætninger er altsaa længde = -b. ,,gierdrum'' har saaledes ret meget tilfælles med ,,to drum''.

Senere i programmet kan man par ,,sædvanlig,, made læse maskinproceduren ned i et array og derefter hoppe til den.

Vi vil i et eksempel vise et stykke af et program, der varetager disse ting. Fra procedurens beskrivelse ved vi, at regningerne skal begynde i den fjerde celle.

```
begin integer Mplace1,længde 1,i...;
.....
.....
Mplace1 := drumplace;
gierdrum (‡<BESS›,længde1);
.....
.....
begin boolean array BESS[1:længde 1]; real p,n;
drumplace := Mplace 1; from drum (BESS);
.....
for i := 1 step 1 until n do begin
.....
p := gierproc (BESS[4],x);
.....
end af for sætning;
.....
end af array-blok;
.....
end af program;
```

Vi ser, hvorledes vi først gemmer den aktuelle værdi af drumplace, og derefter læser proceduren BESS ind til tromlen ved kaldet af gierdrum. Samtidig faar variablen længde1 en værdi svarende til procedurens længde. længde1 anvendes i deklarationen af det array, BESS, som vi senere læser maskinproceduren ned i. Procedurefunktionen aktiveres ved kaldet gierproc, og regningerne begynder i den fjerde celle af arrayet, BESS[4].

Procedurefunktionen staar til raadighed, indtil maskinen forlader den blok, i hvilken arrayet er deklareret. Sidenhen kan den igen kaldes ind i et passende array med ,,from drum,,. Bemærk at arrayet er deklareret boolean.

Hvis der ingen parametre er bortset fra den første, der angiver adressen, kan man bruge en lidt simplere form end gierproc. Kaldet bliver saa

```
p := gier (BESS[4]);
```

Hvis man kun har brug for proceduren een gang, kan det ogsa gøres

simplere. Hvis ,,gierproc,,s første aktuelle parameter ikke er en variabel, men et tal eller en streng, vil kaldet af den kombinere egenskaber ved ,,gierdrum,, og ,,gierproc,,. Ved kaldet

p := gierproc (‡<BESS‡,x);

startes strimmellæseren, identifikationen kontrolleres og strimmelen indlæses som ved ,,gierdrum,, men til stakken, der udvides med det fornødne område. Umiddelbart derefter udføres maskinkoden, og saa fortsættes med næste sætning i algolprogrammet. Samtidig formindskes stakken igen, og man kan kun gentage proceduren ved paany at indlæse strimmelen. Da der ikke er nogen mulighed for at bestemme indhoppet til maskinproceduren, skal denne være kodet i overensstemmelse med særlige konventioner, og man kan ikke benytte enhver maskinprocedure paa denne maade.

Det er dog uhensigtsmæssigt, at skulle indlæse maskinprocedurestrimlen hver gang programmet skal i maskinen. Nedenfor vises eksemplet fra før i en form, hvor man kun skal gøre det een gang. Eksemplet er udvidet til 5 maskinprocedurer.

```
begin integer array Mplace, længde[1:5]; integer a,b;
a := drumplace;
if kbon then begin
from drum(Mplace); from drum(længde);
Mplace[1] := drumplace; typechar; gierdrum(‡<A‡,længde[1]);
Mplace[2] := drumplace; typechar; gierdrum(‡<B‡, længde[2]);
.................
                                  gierdrum(‡<E‡,længde[5]);
b := a-drumplace; drumplace := a; to drum (Mplace); to drum (længde);
typechar;
gierproc(‡<binout‡,1,a,b);
end of kbon;
drumplace := a;
from drum (Mplace); from drum (længde); drumplace := Mplace[5]-længde[5];
comment her begynder selve programmet;
```

I dette eksempel er oplysningerne om ,,drumplace,,s værdier og længderne samlet i de to arrays, Mplace og længde. Naar man første gang kører sit program, har man KB-lampen tændt, og faar udført de mange indlæsninger af maskinprocedurerne til tromlen. ,,binout,, udskriver den relevante del af tromlen paa en strimmel, saaledes at hele programmet er klar til brug igen, naar denne strimmel er indlæst.

De to første from drum-sætninger har kun til formaal, at stille ,,drumplace,, paa en værdi svarende til, at der er reserveret plads til disse to arrays paa tromlen. Det er nemlig nødvendigt at anbringe dem paa tromlen inden binout-udskriften, og de skal være anbragt et sted, saa man kan ,,finde dem igen,, uden andre oplysninger end dem, der ligger i selve det oversatte program. Derefter indlæses de enkelte maskinprocedurer paa samme ma de som i det tidligere eksempel. ,,b,, svarer til den samlede ændring i ,,drumplace,, fra starten og til vi er færdige med indlæsningen. Det er længden af den del af tromlen, der skal skrives ud med ,,binout,, og det behøves som parameter i denne. Inden udskriften kan finde sted, maa ,,Mplace,, og ,,længde,, gemmes paa tromlen paa den afsatte plads.

Hvis vi ikke lige har foretaget indlæsning af maskinprocedurer, er det nødvendigt at hente ,,Mplace,, og ,,længde,, fra tromlen, og derefter skal ,,drumplace,, stilles, saa den almindelige brug af tromlen til lagring af arrays ikke ødelægger de indlæste procedurer.

typechar-sætningerne giver operatøren tid til at sætte de forskellige strimler i læseren.

A11. binout.

Det er muligt at lave en strimmel, der indeholder en specificeret del af tromlens indhold paa en form, der fylder mindst muligt. Man kan f.eks. have et oversat program paa denne form, hvorved man sparer oversættelsesprocessen, hver gang programmet skal bruges. Det er specielt fordelagtigt, hvis programmet indeholder en række maskinkodede procedurer. Se eksemplet i afsnit A10.

binout er selv en maskinkodet procedure, der indlæses og udføres med kaldet gierproc (‡<binout>, ...).
Den første parameter efter identifikationen, ‡<binout>, er et heltal, der angiver et grundomraade for udskriften. Resten af parametrene angiver to og to (og der skal derfor være et lige antal) en værdi af drumplace og længden af det følgende afsnit af tromlen, d.v.s. længden af det array, der er overført til tromlen ved den paagældende værdi af drumplace. Sammenhængen mellem heltal og grundomraade fremgaar af nedenstaaende skema.

Heltal	Grundomraade
0	Intet
1	Det oversatte program
2	Det oversatte program + oversætterens standardprocedurer
3	Intet
5	Hele oversætteren

Heltal 5 har saaledes ingen interesse for koderen.
En 2-strimmel kan altid køres som program, saa snart den er indlæst.
En 1-strimmel kan kun køres, hvis standardprocedurerne (kanal 39-77) i forvejen er paa tromlen. Dette vil være den normale situation, jvnf. eksemplet side a13.

En 0-strimmel skal have alle nødvendige dele specificeret ved drumplace og længde. Baade 0, 1 og 2-strimler vil efter indlæsning bringe maskinen i run-situation.

En 3-strimmel er beregnet paa senere indlæsning til et program med gierproc eller gierdrum. Identifikationen er den samme som det udskrevne array - der jo formentlig er en maskinprocedure - hele tiden har haft.

0, 1, 2 og 5 kan kun bruges i formen gierproc (‡<binout>, ...), medens 3 ogsaa kan bruges, hvis binout først er indlæst med gierdrum (‡<binout>,a);