ELEMENTÆRT ALGOL

Med eksempler fra kemiske beregninger

Jørgen Kjær

Anden udgave, andet optryk

Haldor Topsøe, Vedbæk, Danmark

1970

Copyright 1970

Haldor Topsøe, Chemical Engineers

Vedbæk, Denmark

Produktion Akademisk Forlag

FORORD TIL ANDEN UDGAVE

Da første udgave af denne bog udkom i 1964 under betegnelsen: Programmering af Kemiske Beregninger, Bind 1, ALGOL, var det hensigten at fortsætte denne serie med andre bind omhandlende numeriske metoder, termodynamiske beregninger, konverterberegninger, o.s.v. Af disse bind er til dato kun udkommet bind 2 og 3 om numeriske metoder. Jeg har ikke opgivet at fortsætte serien, men jeg har indset, at det ikke er muligt for en enkelt person at skabe alle disse programmer, at vedligeholde dem, og at give en udførlig beskrivelse heraf.

En anden vanskelighed ved at fortsætte serien i den oprindeligt planlagte form er følgende. Første udgave af ALGOL-bogen benyttede GIER ALGOL II, medens den engelske udgave af bogen brugte GIER ALGOL III. Siden da er der fremkommet en ny variant: GIER ALGOL 4, der formentlig bliver den endelige version for GIER. Dette ALGOL 4 vil sikkert være meget lig det planlagte ALGOL for den nye RC 4000 maskine, ligesom det er næsten identisk med det nu fremkomne IBM-ALGOL. Hos Haldor Topsøe vil der sidst i 1968 blive opstillet et IBM regneanlæg af typen 360/44, som fortrinsvis vil blive programmeret i FORTRAN, maaske ogsaa i ALGOL.

For ikke at gøre det unødigt indviklet for begyndere, har jeg valgt at begrænse indholdet af nærværende bog til kun at omfatte det elementære ALGOL med et minimum af input-output-procedurer fra GIER ALGOL 4. Tromletransporter er ikke medtaget. Jeg haaber at kunne skrive en særlig bog om vore procedurer til avanceret input og output og til transporter til tromle og pladelager. Denne beskrivelse vil omfatte GIER ALGOL II, III og 4, samt vore planer om det saakaldte GIPS-system.

Vedbæk, Januar 1968

Jørgen Kjær

Andet optryk er uændret fra første optryk 1968.

Vedbæk, Juni 1970

Jørgen Kjær

AF FORORDET TIL FØRSTE UDGAVE

Formaalet med denne bog er at give en beskrivelse af det generelle programmeringssprog ALGOL. Ved fremstillingen er der lagt vægt paa at give eksempler paa programmering af beregninger inden for den kemiske ingeniørvidenskab.

Selv om der i dag findes flere udmærkede lærebøger i ALGOL paa dansk (Andersen (1963), Vilstrup (1963)), har jeg alligevel ment det formaalstjenligt at udarbejde en saadan bog, og jeg har taget særligt hensyn til en række forhold, som jeg mener er vigtige for at lette tilegnelsen af sproget.

1. Fremstillingen følger saa vidt muligt naturmetoden, d.v.s. at man straks begynder at nedskrive simple sætninger i ALGOL, som umiddelbart kan forstaas af læseren, fordi der kun bruges velkendte matematiske symboler (+, -, ×, o.s.v.). Jeg mener, at det er meget vigtigt - i hvert fald i begyndelsen af undervisningen - helt at undgaa opremsning af syntaktiske og grammatiske regler. De fleste mennesker er i stand til automatisk og ubevidst at tilegne sig de væsentligste elementer af et sprog, hvis de placeres i omgivelser, som anvender dette sprog, hvorimod de kører uhjælpeligt fast, hvis man forsøger at lære dem sproget ad teoretisk vej med ordbøger og grammatik.

Paa et senere stadium af undervisningen maa de finere detailler af sproget naturligvis diskuteres, men dette gaar lettere, hvis man i forvejen behersker en elementær version af sproget.

2. Det er vigtigt at illustrere sprogets elementer med smaa eksempler hentet fra det fagomraade, som programmøren beskæftiger sig med. Hvis man kun giver helt neutrale eksempler fra den anvendte matematik, f.eks. løsning af andengradsligninger eller beregning af bestemte integraler, bliver fremstillingen meget farveløs, og man forspilder muligheden for at give programmøren en forstaaelse af, hvorledes matematikken kan anvendes paa f.eks. kemiske problemer.

3. Det er et uafgjort problem, om programmering af kemiske beregninger bør udføres af kemiingeniører, som har lært ALGOL, eller af ALGOL-specialister, som har lært noget kemi eller faaet det kemiske problem forklaret af en kemiker. Jeg har derfor tilrettelagt bogen, saaledes at den

kan læses uden forkundskaber i ALGOL eller kemisk ingeniørvidenskab.

4. Selvom ALGOL er et universelt sprog, vedtaget af en international komite, er der indenfor sprogets rammer mulighed for mindre afvigelser. Nærværende fremstilling er baseret paa det saakaldte GIER ALGOL II, som er udviklet af P. Naur m.fl. til brug for regnemaskinen GIER. Det har for danskere den fordel, at der er benyttet danske betegnelser for standard-procedurerne til indlæsning og trykning, m.v. Der findes ogsaa en version: GIER ALGOL III, som benytter engelske betegnelser, og som iøvrigt er udstyret med forskellige finesser. Det er i øjeblikket uvist, om vi vil gaa over til at bruge GIER ALGOL III paa et senere tidspunkt.

Bogen giver ogsaa en tiltrængt beskrivelse af visse specielle konventioner for ALGOL-programmeringens praktiske udførelse, som gælder ved GIER-installationen hos Haldor Topsøe.

....

Hellerup, Maj 1964

Jørgen Kjær

INDHOLDSFORTEGNELSE

Side

1. INDLEDNING 8
2. DEN PROGRAMSTYREDE CIFFERREGNEMASKINE 10
   2.1. Bordregnemaskinen 10
   2.2. Cifferregnemaskinen 11
   2.3. En vigtig forbedring 12
   2.4. Betingelser 13
3. PROGRAMMERINGSSPROG 15
   3.1. Maskinsprog 15
   3.2. ALGOL 16
4. GRUNDBEGREBERNE I ALGOL 18
   4.1. Simple sætninger 18
   4.2. Et simpelt program 24
   4.3. Standardfunktioner 27
   4.4. Talsæt og <u>for</u>-sætninger 28
   4.5. Betingelsessætninger 43
   4.6. Etiketter og hopsætninger 49
5. SIMPLE PROGRAMEKSEMPLER 61
   5.1. Beregning af vandgasligevægten 61
   5.2. Beregning af trykfald i rør 67
6. PROCEDURER 82
   6.1. Procedurebegrebet 82
   6.2. Beregning af cylinderrumfang 82
   6.3. Summation 89
   6.4. Beregning af entalpi 100
   6.5. Beregning af vandgasligevægt 106
   6.6. Beregning af trykfald i rør 110
   6.7. Rekursive procedurer 117
7. ANDRE ALGOLBEGREBER 119
   7.1. Blokke 119
   7.2. Logiske variable 131
   7.3. Skiftespor 139
   7.4. Brug af <u>integer</u> og <u>real</u> 142
   7.5. Specielle <u>for</u>-sætninger 148
   7.6. Brug af navne 154
   7.7. Case-Konstruktioner 156

8. INDLÆSNING OG TRYKNING 158

8.1. Læsning af tal 158

8.2. Læsning af tegn 161

8.3. Trykning af tal 172

8.4. Trykning af tekst 178

8.5. Trykning af tegn 179

9. REFERENCER 184

10. STIKORDSREGISTER 185

1. INDLEDNING

Den elektroniske cifferregnemaskine er blevet et uundværligt hjælpemiddel ved alle større tekniske og videnskabelige beregninger. Det har dog vist sig i praksis, at det kan være vanskeligt at faa det fulde udbytte af maskinen, fordi den stiller større krav til brugerne, end disse har været klar over. Som ved mange moderne opfindelser er den gode tjener ogsaa en streng herre. En regnemaskine kan erstatte 1000-10000 beregnere, men forlanger til gengæld et planlægnings- og programmeringsarbejde paa højt niveau, som ikke blot kræver specielt kendskab til programmering, men ogsaa en langt mere dybtgaaende viden om det paagældende fagomraade, end man hidtil har kunnet nøjes med.

For at kunne benytte en elektronisk cifferregnemaskine, skal man have programmeret sin beregningsopgave, d.v.s. inden beregningen udføres, skal man paa forhaand fortælle maskinen, hvorledes det skal foregaa. Det sker ved at fodre maskinen med et program, som i sig indeholder alle de formler og regneudtryk, der skal bruges, samt oplysninger om i hvilken rækkefølge, de skal anvendes, antal af iterationer, m.m. Programmet skal af brugeren nedskrives i et sprog med snævre grammatiske regler, som maa overholdes strengt. Brugeren skal altsaa løse to opgaver: udtænke et program, og nedfælde disse tanker i et bestemt sprog.

ALGOL er et saadant sprog, som kan bruges af mennesker til at fortælle elektronregnemaskiner hvilke numeriske processer, de ønsker udført. Det er formentlig eet af de bedste, som findes for tiden, idet det er tilpas universelt til at kunne bruges til vidt forskellige regnemaskiner, uden dog at være saa generelt, at man drukner i en uforstaaelig symbolik.

Det er vigtigt, at læseren gør sig klart, at undervisningen i ALGOL er to sider af den samme sag. Man skal dels lære at planlægge beregninger, d.v.s. at opbygge disse ud fra simple grundelementer: beregning af udtryk, bestemmelse af nulpunkter, løsning af differentialligninger o.m.a., og dels skal man lære den korrekte sproglige formulering af programmeringen. Heldigvis er formen af helt simple regneudtryk i ALGOL som:

$$a + b\times(c + d)$$

saa nær sammenfaldende med almindelig matematisk sprogbrug, at de fleste rent ubevidst accepterer baade form og indhold af et saadant udtryk, og man har derfor et udmærket udgangspunkt for en yderligere udbygning af sproget.

ALGOL er defineret i en officiel rapport: Backus et al. (1960, 1963). Begyndere advares indtrængende mod at læse i - ja blot aabne - denne rapport, idet den er affattet i et sammentrængt sprog, som gør den næsten ulæselig. Med en del erfaring i brug af ALGOL er rapporten uundværlig til at afgøre tvivlsspørgsmaal vedrørende detailler i sprogets form og indhold.

2. DEN PROGRAMSTYREDE CIFFERREGNEMASKINE

Det er muligt at faa en vis forstaaelse af de fundamentale principper i brugen af en cifferregnemaskine ved at foretage en sammenligning med en almindelig, elektro-mekanisk bordregnemaskine, f.eks. af FACIT-typen.

2.1. Bordregnemaskinen.

Maskinen kan udføre de fire regningsarter: addition, subtraktion, multiplikation og division. Skal man f.eks. udføre multiplikationen:

1234×9876

gør man følgende:

1. Nulstil registrene.
2. Indtip det første tal: 1234.
3. Tryk paa multiplikationstasten.
4. Indtip det andet tal. 9876.
5. Tryk paa lighedstegnstasten.

Maskinen regner da, og produktet fremkommer i eet af registrene. Hvis der skal regnes videre paa tallet, maa beregneren normalt skrive det ned paa et stykke papir, før han begynder paa næste udregning.

Vi ser, at maskinen kan siges at have en hukommelse, der tillader den at huske op til to tal (1234 og 9876) og ligeledes at huske een operation (multiplikation). Naar operationen er udført, er operationshukommelsen tom, og man maa begynde forfra med at indtippe nye oplysninger.

2.2. Cifferregnemaskinen.

Her er der tale om langt større hukommelse med plads til lagring af hundreder eller tusinder af tal og operationer. Nogle af de første programstyrede cifferregnemaskiner var forsynet med to adskilte hukommelser: den ene til lagring af tal (begyndelsesværdier, mellemresultater og de endelige resultater) og en operationshukommelse til lagring af de enkelte operationer. Vi kan betragte eksemplet:

Talhukommelse

1:	3.456	(a)
2:	9.876	(b)
3:	12.345	(c)
4:	8.177	(d)
5:	0	(e)

Operationshukommelse

1:	Hent tal 3
2:	Adder tal 4
3:	Multiplicer med tal 2
4:	Adder tal 1
5:	Gem resultat som tal 5

Hvis vi kalder tallene i registrene 1 - 5 i talhukommelsen for a, b, c, d og e, ser vi, at denne konfiguration af operationshukommelsen vil beregne tallet e som udtrykket:

$$a + b\times(c + d)$$

Tilstedeværelsen af talhukommelsen og operationshukommelsen med plads til mange oplysninger er en vigtig forskel fra bordregnemaskinen. At regnehastigheden for de enkelte operationer er mange gange større i elektronregnemaskinen er naturligvis ogsaa en vigtig forskel, og det er netop den store regnehastighed, som gør det nødvendigt at have en stor hukommelse, for at maskinen selv kan udføre omflytninger af mellemresultater. Der

er ikke tid til, at operatøren skal aflæse og nedskrive dem.

I eksemplet ovenfor er vist en talhukommelse med plads til 5 tal med 3 decimaler. I GIER har talhukommelsen plads til 1024 tal hver med 8 betydende decimale cifre. Man siger, at hukommelsen (ogsaa kaldet lageret) indeholder 1024 celler eller æsker til tal.

2.3. En vigtig Forbedring.

I eksemplet ovenfor var der ogsaa 5 pladser i operationshukommelsen. Vi ser, at hver operation er opbygget af to dele, nemlig selve operationen, som her er:

1: HENT
2: ADDER
3: MULTIPLICER
4: ADDER
5: GEM

samt nummeret paa den celle i talhukommelsen, som operationen virker paa, altsaa her: 3, 4, 2, 1 og 5. Disse numre kaldes for operationens adresse.

Hvis maskinen har de fire grundoperationer: adder, subtraher, multiplicer og divider, kan vi vide paa forhaand, at den i hvert fald ogsaa maa have to yderligere operationer: hent og gem, altsaa 6 operationer som minimum. I GIER findes 64 grundoperationer, som vi dog ikke vil omtale nærmere her.

Paa et tidligt tidspunkt af elektronregnemaskinens historie gjorde man den snedige opfindelse, at det slet ikke er nødvendigt at have to forskellige hukommelser, een for tal og een for operationer. Hvis vi har 64 grundoperationer og disse skal kunne operere paa 1024 forskellige celler, har vi ialt 64×1024 = 65536 forskellige kombinationer af operationer. Hver af disse kan udtrykkes ved eet af tallene fra 1 til 65536, og disse tal er lige saa gode tal som alle andre tal, og vi kan jo bare anbringe dem i talhukommelsen. Dette er en vigtig forbedring. Man skal da blot sørge for, at maskinen som første operation faar fat i det rigtige tal i hukommelsen og fortolker det som en operation. Naar denne operation er udført,

vil maskinen normalt gaa videre til næste tal i hukommelsen, fortolke det som en operation, o.s.v. Der kan dog forekomme afbrydelser heri, hvori maskinen hopper tilbage i programmet.

Normalt anbringer man operationstallene i den ene ende af talhukommelsen og de egentlige tal i den anden ende. Man opnaar iøvrigt en meget vigtig effektivisering af maskinens virkemaade derved, at programmet kan udføre beregninger paa selve operationstallene. Skal vi f.eks. addere alle tal fra celle 100 til celle 200, kan det ikke betale sig at lade programmet se saaledes ud:

HENT tal i celle 100
ADDER - - - 101
- - - - 102
o.s.v.

det er alt for klodset. I stedet har man kun een operation, som adderer, og som først ser saaledes ud:

ADDER tal i celle 101

Naar additionen er sket, skal programmet sørge for, at tallet 101 bliver øget til 102, og at operationen kommer til udførelse igen, o.s.v. indtil man er naaet til 200.

Dette, at programmet selv ændrer sine egne operationer, er et meget vigtigt træk i de moderne elektronregnemaskiner.

2.4. Betingelser.

Vi saa ovenfor, at man kan lave et program, som adderer tallene i cellerne 100 til 200 ved efterhaanden at øge adressen i additionsoperationen, indtil man er naaet til 200. Der opstaar da det spørgsmaal, hvorledes maskinen kan vide, at den er naaet til 200. Det er klart, at maskinen paa een eller anden maade maa gøres følsom for betingelser, d.v.s. at den i visse situationer skal kunne afgøre om den skal gøre enten det ene eller det andet. Man plejer at knytte disse betingelser til tallenes fortegn eller til spørgsmaalet om et tal er nul eller ikke. I eksemplet ovenfor

kan man lade maskinen beregne adressen minus 200 hver gang den har adderet og gentage additionen, saalænge denne differens ikke er nul.

Evnen til at adlyde betingelser er ogsaa et meget vigtigt træk ved elektronregnemaskinen, og det er vel nok denne evne, der har bevirket, at man sommetider anvender betegnelsen elektronhjerne. Maskinen kan naturligvis kun adlyde saadanne betingelser, som programmøren har ment det hensigtsmæssigt at anbringe paa passende steder i programmet. Intuitive tænkeprocesser kan næppe udføres paa elektronregnemaskiner, i hvert fald ikke uden en meget stor forøgelse af deres hukommelse.

3. PROGRAMMERINGSSPROG

3.1. Maskinsprog.

Vi saa i sidste kapitel, at beregningen af udtrykket:

$$a + b\times(c + d)$$

kunne foregaa ved hjælp af et lille program af formen:

1: HENT tallet i celle 3
2: ADDER - - - 4
3: MULTIPLICER med tallet i celle 2
4: ADDER tallet i celle 1
5: GEM resultatet i celle 5

De enkelte operationer skal paa passende maade omregnes til talform og gemmes i lageret.

Et program skrevet paa denne form siges at være kodet i maskinsprog, idet programmøren sørger for i hver operationscelle at anbringe talværdien for grundoperationen og den tilhørende adresse efter de konventioner, som gælder for den paagældende maskine. Man kan let forestille sig, at disse konventioner kan variere meget fra een maskine til en anden, idet antallet af grundoperationer og antallet af celler er meget forskelligt. Et program skrevet i maskinsprog for een maskine kan derfor kun sjældent overføres til en anden maskine.

Foruden denne ulempe ved at bruge maskinsprog er der to andre nok saa vigtige.

For det første er det meget upraktisk, at regneprocesserne er opdelt i de helt smaa atomare regneoperationer som de fire regningsarter og hent- og gem-operationerne. En programmør vil meget gerne have lov at skrive f.eks.:

TAG KVADRATRODEN
LØS ET SÆT LINEÆRE LIGNINGER
INTEGRER EFTER SIMPSONS FORMEL
o.s.v.

Disse processer kan opbygges af grundprocesserne, men har man een gang for alle programmeret denne opbygning, er det bekvemt at kunne nøjes med at skrive navnet paa dette kompleks af operationer. Dette, at opfatte en samling af enkelte elementer som en helhed, at give denne helhed et navn, og derefter at bruge dette navn, som om helheden var et udeleligt grundelement, er noget fundamentalt for den menneskelige erkendelse. Det er derfor rimeligt at forvente at problemet dukker op i forbindelse med programmering.

Den anden ulempe ved maskinsproget er brugen af adresser. Det er let at forestille sig, at naar man i hver eneste operation skal opgive nummeret paa den celle, hvori det paagældende tal staar, da er der store muligheder for fejl. Ikke blot kan man komme til at regne paa et forkert tal, men man kan ogsaa komme til at bruge en operationscelle som tal, eller hvad der oftest er endnu værre: bruge et tal som en operation.

3.2. ALGOL.

De her nævnte ulemper ved maskinsprog undgaas næsten fuldstændigt ved programmering i ALGOL. Brugen af adresser forsvinder helt og man kan anvende saa at sige vilkaarlige navne paa de dele af ens program, som man ønsker at maskinen skal opfatte som en helhed.

Inden vi i næste kapitel gaar over til at gennemgaa selve sproget ALGOL, skal vi først se lidt paa programmeringens praktiske gennemførelse.

Processen omfatter følgende faser:

1. Programmøren gennemtænker problemet og udvælger de matematiske metoder, som skal anvendes.

2. Programmet nedskrives i kladde (men helst omhyggeligt) i ALGOL.

3. Programmet renskrives af en hulledame paa en skrivemaskine (Flexowriter), der foruden den almindelige renskrift paa papir afleverer et hulbaand, som er en fuldstændig kopi af hvert eneste bogstav og tegn i papirudskriften. Der læses omhyggeligt korrektur paa udskriften og den (og hulbaandet) rettes, om nødvendigt.

4. Vi sørger nu for at anbringe et ganske specielt program (den saakaldte ALGOL-oversætter) i regnemaskinen.

5. Oversætterprogrammet sættes nu igang med at regne. Det læser

ovennævnte hulstrimmel med ALGOL-programmet, underkaster det en omhyggelig analyse for formelle fejl, d.v.s. saadanne fejl, som umiddelbart strider mod sprogets regler. Oplysninger om disse fejl udskrives paa en automatisk skrivemaskine, der er koblet til regnemaskinen. Hvis der er fejl, standser processen her, og programmøren maa rette fejlene og begynde forfra.

6. Hvis der ikke findes formelle fejl, fortsætter oversætterprogrammet sin analyse og ender med at have oversat ALGOL-programmet til et program i maskinsprog. Efter oversættelsen staar det oversatte program klar i maskinen til at regne paa, men normalt vil man gaa videre med:

7. Oversætterprogrammet afleverer sit arbejde i form af et nyt hulbaand indeholdende det oversatte program i maskinsprog, men hullet i en kondenseret form, som kun er beregnet til at skulle læses af maskinen selv, ikke til udskrift paa en Flexowriter. Dette kondenserede program gemmes til brug ved senere beregninger paa programmet. Disse udføres normalt saaledes:

8. Maskinens hukommelse nulstilles.

9. Maskinen læser det kondenserede program.

10. Ved tryk paa startknappen begynder beregningerne. Disse vil oftest begynde med indlæsning af et vist talmateriale, f.eks. apparatdimensioner eller driftsbetingelser, som er karakteristiske for den specielle beregning, som er ved at blive udført. Maskinen afleverer resultatet af beregningen i form af et hulbaand, som udskrives paa en Flexowriter. Resultatet kan ogsaa udskrives paa en hurtig linieskriver.

Naar ALGOL-programmet een gang er oversat, er det altsaa kun punkt 8 - 10, som bruges ved den daglige anvendelse af programmet.

4. GRUNDBEGREBERNE I ALGOL

I dette kapitel vil vi kun studere de elementære dele af ALGOL, og vi vil hovedsageligt koncentrere os om de egentlige beregninger, idet vi tænker os, at vi staar midt i et program. Bortset fra et enkelt eksempel vil vi ikke behandle problemet om, hvorledes det aktuelle talmateriale kommer ind i og ud af maskinen. Dette forklares specielt i kapitel 8.

4.1. Simple Sætninger.

Vi har tidligere set regneudtrykket:

a + b×(c + d)

som et eksempel (side 8) paa et ALGOL-udtryk. Vi ønskede at beregne tallet e ved hjælp af dette udtryk. Dette skrives i ALGOL saaledes:

e := a + b×(c + d);

Meningen er, at naar maskinen har beregnet talværdien af a + b×(c + d) ved at hente talværdierne af a, b, c og d fra de respektive celler, da skal resultatet gemmes i den celle, som er forbeholdt tallet e. Lad os tage et simplere eksempel:

y := a + b;

Her skal y beregnes som a + b. Semikolonet bruges til at afslutte ALGOL-sætningen og behøver ikke nærmere forklaring. Derimod er der grund til at dvæle ved lighedstegnet, som i ALGOL skrives som kolon, lighedstegn. I virkeligheden findes der to lighedstegn i ALGOL, nemlig det dynamiske lighedstegn, som vi lige har set:

y := a + b;

som befaler, at y skal sættes lig med (d.v.s. beregnes som) a + b, samt et

statisk lighedstegn, f.eks:

z = a + b

som benyttes i betingelser. Her skal z ikke beregnes som a + b, men maskinen skal undersøge, om z allerede er lig med a + b, ja eller nej. Resultatet af sammenligningen er altsaa ja eller nej, og konstruktionen z = a + b vil derfor normalt indgaa i en betingelse. Den kan ikke staa helt alene som y := a + b.

I almindelig matematisk sprogbrug skelner man ikke mellem de to slags lighedstegn, idet dette som regel fremgaar af sammenhængen.

Som følge af det dynamiske indhold af en ALGOL-sætning vil man forstaa, at det er tilladt at skrive f.eks. saaledes:

x := x + 1;

Naar maskinen kommer til denne sætning (i det oversatte program), tager den den aktuelle værdi af x, adderer 1, og gemmer resultatet i x-cellen igen. Man har altsaa øget x med 1.

Resultatet af beregningen af et udtryk kan godt gemmes i flere forskellige celler, f.eks.:

x := y := a + b;

Her beregnes a + b og det fremkomne tal gemmes baade i cellen for x og for y.

I almindelighed bestaar en ALGOL-sætning af en højreside, som er det egentlige regneudtryk, samt een eller flere venstresider, som simpelt hen er en liste over de variable, i hvis celler det fremkomne resultat skal gemmes. Man skal bemærke, at højresiden udregnes først, inden der sker ændring af den eller de variable paa venstre side. Sætningen skal altsaa til en vis grad læses fra højre mod venstre. Tegnet := kan opfattes som en stiliseret pil, der peger mod venstre.

Vi skal nu se lidt nærmere paa formen af de regneudtryk, som kan bruges som højresider i sætninger af denne art. De fire sædvanlige regningsarter noteres med de velkendte tegn: +, -, × og /. Desuden benyttes et specielt tegn for potensopløftning, saaledes at vi ialt har de fem former:

a+b	Addition
a-b	Subtraktion
a×b	Multiplikation
a/b	Division
a↑b	Potensopløftning (a i potensen b)

Den sidste form: a↑b er noget forskellig fra den sædvanlige, hvor man jo anbringer exponenten lidt over linien og ikke bruger ↑-tegnet. Fordelen ved ↑-tegnet er, at det kan skrives med den automatiske skrivemaskine, idet man først skriver: | og derefter: ∧ oveni. Tegnet for: | flytter ikke valsen paa skrivemaskinen.

Den simple sætning:

```
d := a + b + c;
```

kræver tilsyneladende ikke nogen særlig forklaring, idet den variable, d, skal beregnes som summen af a, b og c. Der er dog en lille finesse, som det i enkelte tilfælde kan være vigtigt at være klar over. Sprogets regler siger, at udtrykket a + b + c skal beregnes fra venstre mod højre. Maskinen skal altsaa først tage talværdien af a, dertil lægge værdien af b og endelig addere værdien af c til summen af a og b. Det fremkomne resultat skal gemmes i den celle som er reserveret den variable, d. Man skulle tro, at det var ligegyldigt, i hvilken rækkefølge summationen blev udført, men dette behøver ikke at være tilfældet. Lad os se paa følgende lille programstump:

```
a := 1;
b := 1₁₀-12;
c := -1;
d := a + b + c;
e := a + c + b;
```

Programmet opererer med de fem variable: a, b, c, d og e. I første linie sættes a lig med 1, d.v.s. der gemmes et 1-tal i den celle, som er reserveret a. I anden linie sættes b lig med $1_{10}-12$ (een gange 10 i potensen -12). Bemærk iøvrigt den snedige maade vi noterer 10-potenser paa: det lille 10-tal: $_{10}$ skrives med et bestemt tegn paa skrivemaskinen uden

at dreje paa valsen. I tredje linie sættes c lig med -1. I fjerde linie beregnes d som a + b + c og i femte linie beregnes e som a + c + b. Man skulle paa forhaand tro, at resultatet blev det samme, nemlig $1_{10}-12$. Men det sker ikke, d bliver lig med 0 og e bliver $1_{10}-12$. Forklaringen er den, at maskinen kun regner med en vis nøjagtighed. Hos GIER er denne ca. $1_{10}-9$, d.v.s. naar vi beregner a + b som 1 + $1_{10}-12$ bliver resultatet nøjagtigt 1, idet de $1_{10}-12$ slet ikke kan være med ude til højre i decimalerne af 1-tallet. Naar vi derefter adderer c (-1), bliver resultatet nøjagtigt 0. Hvis vi derimod, som i femte linie, først adderer a og c (1-1) er resultatet nøjagtigt 0, og addition af b giver det rigtige resultat: $1_{10}-12$.

I de fleste ingeniørmæssige beregninger kan man roligt glemme denne faldgrube, men det er klart, at i enkelte tilfælde kan man faa helt forkerte resultater, hvis man ikke er klar over denne afrundingseffekt.

Vi ser nu paa en mere kompliceret sætning:

```
p := a + b×c + d/e - f↑g;
```

Her skal den variable, p, beregnes ud fra udtrykket paa højre side, som indeholder de fem regnesymboler, vi lærte ovenfor. Vi ser, at der staar d/e, men det er ikke paa forhaand klart, om d skal divideres med e eller med e - f↑g. Vi maa altsaa have nogle faste regler for operationssymbolernes indbyrdes styrke. Reglen er den, at operationerne skal udføres i følgende orden:

1. Potensopløftning (↑)
2. Multiplikation og division (×, /)
3. Addition og subtraktion (+, -)

I tilfældet ovenfor skal b×c altsaa beregnes inden vi adderer til a. Ligeledes skal d/e beregnes inden dette led summeres, og f↑g skal være beregnet inden det subtraheres. Som et kuriosum kan vi se paa, hvorledes oversætterprogrammet faktisk nedbryder regneudtrykket til simplere bestanddele for at kunne udføre beregningerne i den rigtige rækkefølge. Bemærk, at udtrykket i princippet beregnes fra venstre mod højre. Programmet bruger en ekstra hjælpevariabel, som vi arbitrært betegner med h, samt et register, R, til dannelse af simple udtryk. Det nedbrudte program ser saaledes ud:

De to første led kan straks behandles, og deres sum anbringes i cellen med hjælpevariablen, h:

```
h := a + b×c;
```

Derefter beregnes d/e i registeret, R:

```
R := d/e;
```

og adderes til h:

```
h := h + R;
```

Endelig beregnes f↑g:

```
R := f↑g;
```

og vi faar det endelige resultat:

```
p := h - R;
```

Bemærk, hvorledes overskueligheden af det oprindelige udtryk er gaaet helt tabt ved denne neddeling af regneprocessen i mindre enheder.

Det er tilladt at anbringe parenteser i regneudtrykkene, og dette er nødvendigt i mange tilfælde. Eks.:

```
p := a + b×c + d/(e - f↑g);
```

Her har vi opnaaet, at tallet d bliver divideret med e - f↑g, idet reglen er den, at talværdien af en parentes skal udregnes inden resultatet indgaar i beregningen af resten af udtrykket.

Til slut et par faldgruber for begyndere. Hvis vi skal beregne en brøk:

```
a×b×c
-----
d×e×f
```

kan dette i ALGOL skrives som enten:

```
q := a×b×c/(d×e×f);
```

eller:

```
q := a×b×c/d/e/f;
```

eller som kombinationer heraf. Derimod er

```
q := a×b×c/d×e×f
```

ensbetydende med, at man vil have beregnet brøken:

```
a×b×c×e×f
---------
    d
```

Multiplikationstegnet kan ikke udelades. Hvis vi skriver:

```
q := abc/def;
```

tror maskinen, at vi har to variable, den ene ved navn abc og den anden ved navn def, og at vi skal beregne abc divideret med def. Man glemmer ogsaa let multiplikationstegnet efter smaa talfaktorer, f.eks. i:

```
a↑2 + 2×a×b + b↑2
```

hvor en begynder ofte fejlagtigt vil skrive:

```
a↑2 + 2ab + b↑2
```

4.2. Et Simpelt Program.

Vi laver nu et sidespring og giver et eksempel paa et komplet ALGOL-program. Vi ønsker et program, som beregner rumfanget af en cylinder med en opgivet højde og diameter.

Selve sætningen, som beregner rumfanget, kan skrives saaledes:

```
RUMFANG := 0.785398×HØJDE×DIAMETER↑2;
```

altsaa højden gange kvadratet paa diametren gange talfaktoren 0.785398, som er pi divideret med 4.

ALGOL-oversætteren kan ikke oversætte en løsrevet sætning som ovenstaaende uden nærmere forklaring. Først og fremmest skal man fortælle maskinen, at der i programmet forekommer de tre variable: RUMFANG, HØJDE og DIAMETER. Dette sker ved at skrive programmet saaledes:

```
begin
   real RUMFANG, HØJDE, DIAMETER;
   RUMFANG := 0.785398×HØJDE×DIAMETER↑2
end;
```

Vi ser, at programmet som helhed er omgivet af betegnelserne: begin og end. Dette behøver formentlig ikke at forklares nærmere, men det bemærkes, at ordene begin og end ogsaa kan forekomme inden i et program, hvor de bruges som parenteser om sætninger eller grupper af sætninger, som skal opfattes som en helhed. Naar oversætterprogrammet har mødt ordet end lige saa mange gange, som det har mødt ordet begin, er det klar over, at det har læst hele programmet.

Betegnelsen:

```
real RUMFANG, HØJDE, DIAMETER;
```

er en saakaldt deklaration, som oplyser oversætterprogrammet om, at der i det foreliggende program kan forekomme tre variable med de tre nævnte navne. Programmet faar altsaa brug for tre celler til at gemme værdierne af de tre variable. Betegnelsen real betyder, at det drejer sig om ganske

almindelige tal, som kan antage reelle værdier. Det sidste er ikke ment som modsætning til komplekse tal, men som modsætning til heltal, idet saadanne variable, som kun bruges til tælling, f.eks. af antallet af komponenter i en gasblanding, bør deklareres at være heltal. ALGOL-betegnelsen herfor er: integer.

Vi udvider nu det ovenstaaende program, saaledes at det ogsaa omfatter indlæsning af de aktuelle værdier af højde og diameter, samt trykning af det beregnede rumfang. Det er her praktisk at lade maskinen trykke ikke blot rumfanget, men ogsaa de indlæste værdier af højde og diameter. Programmet ser nu saaledes ud:

```
begin
   real RUMFANG, HØJDE, DIAMETER;
   select(8);
   HØJDE := read real;
   DIAMETER := read real;
   RUMFANG := 0.785398×HØJDE×DIAMETER↑2;
   writecr;
   write(⟨dddd.ddd⟩, HØJDE, DIAMETER, RUMFANG);
   writecr;
end;
```

Sætningen:

```
select(8);
```

vil i det oversatte program bevirke, at maskinen vælger strimmellæseren som inputmedium og linieskriveren som outputmedium. De to næste sætninger:

```
HØJDE := read real;
DIAMETER := read real;
```

vil i det oversatte program bevirke, at maskinen gør sig klar til at indlæse et hulbaand med læseapparaturet samt læser de to første tal, den møder paa strimlen. Hvis vi til afprøvning af programmet laver en inputstrimmel, hvis udskrift paa Flexowriteren ser saaledes ud:

```
3.1,    2.9
```

vil programmet læse tallet 3.1 og gemme det i den celle, som det har reserveret til den variable: HØJDE. Derefter læser maskinen tallet 2.9 og gemmer det i cellen for DIAMETER.

Efter indlæsning af de to tal gaar programmet videre til næste sætning, hvor værdien af RUMFANG beregnes.

De tre sidste sætninger bevirker, at programmet udskriver det ønskede resultat paa linieskriveren. Den første af de tre sætninger:

```
writecr;
```

bevirker, at der sendes et bestemt signal til linieskriveren, saaledes at papiret rykkes een linie frem og den følgende udskrift begynder helt i venstre side af papiret. Betegnelsen: cr betyder vogn retur (carriage return) og illustrerer den tilsvarende effekt, hvis udskriften havde foregaaet paa en skrivemaskine.

Den næste sætning:

```
write({dddd.ddd}, HØJDE, DIAMETER, RUMFANG);
```

bevirker trykning, d.v.s. udskrift af de tre variable paa linieskriveren. Betegnelsen {dddd.ddd} betyder, at man vil have trykt tallene med tre decimaler (.ddd) og højst fire tal før kommaet (dddd.). Endelig indeholder parentesen i tryksætningen navnene paa de tre variable, som vi ønsker trykt.

Den sidste sætning:

```
writecr;
```

giver igen ny linie og er nødvendig, fordi linieskriveren trykker en hel linie ad gangen. Man kan derfor først se resultatet, naar linien er afsluttet med writecr.

Naar programmet afprøves med de to ovennævnte tal: 3.1 og 2.9, faar man følgende udskrift paa linieskriveren:

```
3.100   2.900  20.476
```

Det ønskede rumfang er altsaa 20.476.

4.3. Standardfunktioner.

I ALGOL er det tilladt uden nærmere forklaring at bruge visse standardfunktioner i de regneudtryk, som man nedskriver. Man kan f.eks. skrive:

cos(a + 2×b)

for cosinus til a + 2b eller:

exp(-E/(R×T))

for e i potensen - E divideret med RT.

Der er 9 standardfunktioner:

abs(E)	Den numeriske værdi af E.
sign(E)	Fortegnsfunktionen af E, d.v.s. +1 for E > 0, 0 for E = 0 og -1 for E < 0.
sqrt(E)	Kvadratroden af E.
sin(E)	Sinus til E.
cos(E)	Cosinus til E. I sin(E) og cos(E) skal E være maalt i rent tal, ikke i grader.
arctan(E)	Hovedværdien af arcus tangens til E, d.v.s. den værdi, som ligger i omraadet fra - pi/2 til pi/2.
ln(E)	Den naturlige logaritme til E.
exp(E)	Eksponentialfunktionen til E, d.v.s. tallet e = 2.718.. opløftet til potensen E.
entier(E)	Det største heltal, som er ≤ E.

Betegnelsen E i disse eksempler skal antyde, at argumentet til disse funktioner ikke blot kan være et tal eller en variabel, men også et udtryk af kompliceret art.

Nogle af disse funktioner er ikke defineret for alle værdier af E. Hvis maskinen under kørsel paa et oversat program kommer ud for et saadant tilfælde, standser den beregningen og skriver en bemærkning paa skrivemaskinen. Forsøg paa beregning af kvadratroden af et negativt tal giver udskriften: sqrt og logaritmen til et negativt tal udskriften: ln. Bemærk:

ln(0) giver ikke fejludskrift. For store værdier af E (d.v.s. E > 354.1982) giver exp(E) fejludskriften: exp. Udskriften: spill vil man faa, hvis der optræder for store tal (tal > $1.34_{10}154$) i maskinen eller forekommer division med nul. Da potensopløftning: a↑b for b <u>real</u> beregnes som exp(b×ln(a)), kan man her faa ln-udskrift, hvis a < 0 eller exp-udskrift, hvis a↑b > $1.34_{10}154$.

Bemærk, at der ikke findes nogen standardfunktion for tangens til E. Den maa man beregne som sin(E)/cos(E) forudsat cos(E) ≠ 0. Det beskrives i kapitel 6, hvorledes man selv kan definere nye funktioner.

Eksempler paa entier(E):

E	3.4	6.9	8	-2.1	-3.9
entier(E)	3	6	8	-3	-4

4.4. Talsæt og for-sætninger.

I mange beregninger har man brug for talsæt, d.v.s. grupper af tal i een eller flere dimensioner. Som eksempel kan vi tage en gasstrøm med følgende mængder i kgmol/hr:

Brint	134.17
Kulilte	26.81
Kuldioxyd	14.89
Vand	169.55

Vi kan betegne de fire elementer i dette talsæt med:

F[1], F[2], F[3] og F[4]

og kan tildele dem de rette værdier ved at skrive:

```
F[1] := 134.17;
F[2] :=  26.81;
F[3] :=  14.89;
F[4] := 169.55;
```

Som et andet eksempel tager vi en todimensional koefficientmatrix:

```
a[1,1]  a[1,2]  a[1,3]  a[1,4]
a[2,1]  a[2,2]  a[2,3]  a[2,4]
a[3,1]  a[3,2]  a[3,3]  a[3,4]
```

som kan forekomme ved løsning af tre ligninger med de tre ubekendte:

```
x[1], x[2] og x[3]
```

Ligningerne er da:

```
x[1]×a[1,1] + x[2]×a[1,2] + x[3]×a[1,3] = a[1,4]
x[1]×a[2,1] + x[2]×a[2,2] + x[3]×a[2,3] = a[2,4]
x[1]×a[3.1] + x[2]×a[3,2] + x[3]×a[3,3] = a[3,4]
```

Et talsæt kaldes i ALGOL for et <u>array</u>. I en almindelig matematisk beskrivelse vil man karakterisere det enkelte elements plads i talsættet ved et index eller en gruppe af indices. Da man ikke kan skrive indices paa de eksisterende automatiske skrivemaskiner uden at skulle dreje valsen med haanden, har man vedtaget, at disse indices i ALGOL skal skrives i en firkantet parentes: [ ] paa samme linie.

Naar der i et ALGOL-program optræder talsæt (<u>arrays</u>), maa disse deklareres, inden man kan benytte dem i programmet. Deklarationen kan se saaledes ud:

```
array F[1:4];
```

for gasmængderne og:

```
array a[1:3, 1:4];
```

for koefficientmatricen og:

```
array x[1:3];
```

for de tre ubekendte. Deklarationen har samme formaal som den tidligere nævnte deklaration af simple variable:

```
real RUMFANG, HØJDE, DIAMETER;
```

nemlig at oversætterprogrammet er klar over, hvilke navne det skal kunne genkende i det følgende, samt hvor mange celler, der skal afsættes plads til i det oversatte program til at gemme de variable.

Deklarationen af et array giver talsættets navn (her F, a og x) og derefter en firkantet parentes, som paa een gang giver oplysning om talsættets dimension samt nedre og øvre grænse for indices i hver dimension. For talsættet F er nedre grænse 1 og øvre grænse 4, saaledes at der kun er tale om de fire elementer: F[1], F[2], F[3] og F[4]. Index-grænserne behøver ikke udtrykkeligt at være givet som tal. Man kan f.eks. have:

```
array y[p:q];
```

Her er nedre grænse p og øvre grænse q. Det er dog nødvendigt, at p og q har faaet tildelt en værdi paa det tidspunkt maskinen i det oversatte program naar frem til deklarationen, ellers vil det beregnede antal pladser q - p + 1 ikke være i overensstemmelse med programmørens ønske. Derimod behøver p og q ikke at kendes under selve oversættelsen.

Som et simpelt eksempel paa brugen af talsæt, vil vi lave et program, som læser 10 tal fra en hulstrimmel, beregner deres sum og trykker den. Uden brug af talsæt kan man programmere saaledes:

```
begin
   real a, b, c, d, e, f, g, h, i, j, sum;
   select(8);
   a := read real;
   b := read real;
   c := read real;
   d := read real;
   e := read real;
   f := read real;
   g := read real;
   h := read real;
   i := read real;
   j := read real;
   sum := a + b + c + d + e + f + g + h + i + j;
```

```
    writecr;
    write({ddddd.dddd}, sum);
    writecr;
end;
```

I første programlinie: <u>real</u> a, b, o.s.v. deklareres de 10 tal: a - j og summen: sum, d.v.s. der afsættes plads til disse 11 variable.

I anden linie: select(8), vælges input fra strimmellæseren og output paa printeren.

I linie 3 til 12 læser programmet de ti næste tal fra datastrimlen og gemmer dem i cellerne for a, b, o.s.v.

I linie 13: sum := a + b +.... o.s.v. beregnes summen.

I linie 14: writecr; laves en ny linie paa linieskriveren.

I linie 15 og 16:

```
write({ddddd.dddd}, sum);
writecr;
```

udskrives den beregnede værdi af summen som et tal med højst 5 cifre før kommaet og med 4 decimaler.

Hvis det ikke drejer sig om 10 tal, men 100 eller 1000, er denne programmeringsmetode upraktisk, fordi man faar alt for mange navne paa de variable. Vi laver nu et program med et talsæt, A, med 10 elementer:

```
begin
    real sum;
    array A[1:10];
    select(8);
    A[1] := read real;
    A[2] := read real;
    A[3] := read real;
    A[4] := read real;
    A[5] := read real;
    A[6] := read real;
    A[7] := read real;
    A[8] := read real;
    A[9] := read real;
    A[10] := read real;
```

```
        sum := A[1] + A[2] + A[3] + A[4] + A[5]
             + A[6] + A[7] + A[8] + A[9] + A[10];
        writecr;
        write(⟨ddddd.dddd⟩, sum);
        writecr;
    end;
```

Her har vi erstattet de simple variable: a, b, c, o.s.v. med talsættet A[1:10].

I denne udformning er programmet stadigvæk upraktisk, idet vi baade har den lange sætning:

```
sum := A[1] + A[2] + ...... A[10];
```

samt den lange række af de 10 sætninger til indlæsning af A-elementerne:

```
A[1] := read real;
A[2] := read real;
o.s.v.
```

Vi ser først paa, hvorledes summationen kan nedskrives i en kort form. Man maa ikke bruge skrivemaaden med prikker:

```
A[2] + ...... + A[10]
```

men man har indført noget, som er praktisk talt ækvivalent hermed, blot i en mere veldefineret form.

Hvis vi begynder med at nulstille den celle, som skal indeholde summen:

```
sum := 0;
```

og derefter udfører sætningen:

```
sum := sum + A[k];
```

10 gange ialt, nemlig for k = 1, 2, ... 10, ser vi, at vi derved faar beregnet den ønskede sum. Programmet skal altsaa have formen:

```
sum := 0;
```

Udfør følgende sætning for k = 1, 2, ..., 10:

```
sum := sum + A[k];
```

Her er den indrammede forklaring ikke korrekt ALGOL. Det bliver det derimod, hvis vi skriver:

```
sum := 0;
for k := 1 step 1 until 10 do
sum := sum + A[k];
```

Sætningen:

```
for k := 1 step 1 until 10 do
sum := sum + A[k];
```

kaldes en for-sætning, fordi den skal udføres flere gange for forskellige værdier af een af de variable. Vi ser, at for-sætningen har den formelle opbygning:

```
for < parameterstyring > do
<egentlig sætning>
```

Den parameterstyring, som staar mellem de to ALGOL-gloser for og do skal indeholde oplysning om hvilken parameter man vil bruge (her: k), samt startværdien (her: 1), trinlængden (ogsaa 1) og slutværdien (her 10). Den egentlige sætning udføres saa een gang for hver af de derved fremkomne værdier af k. Andre former for parameterstyring omtales senere, men denne er den vigtigste.

Naar summationen kan udføres med en for-sætning, kan indlæsningen naturligvis ogsaa:

```
for k := 1 step 1 until 10 do
A[k] := read real;
```

Den endelige form af programmet til addition af 10 tal bliver saa:

```
begin
   real sum;
   integer k;
   array A[1:10];
   select(8);
   for k := 1 step 1 until 10 do
   A[k] := read real;
   sum := 0;
   for k := 1 step 1 until 10 do
   sum := sum + A[k];
   writecr;
   write({ddddd.dddd}, sum);
   writecr;
end;
```

I denne udformning læser programmet de 10 tal, inden det begynder at lægge sammen. Hvis der er mange tal, vil det gaa lidt hurtigere, hvis man lader maskinen addere hvert tal, saa snart den har læst det. Dette kan skrives:

```
begin
   real sum;
   integer k;
   array A[1:10];
   select(8);
   sum := 0;
   for k := 1 step 1 until 10 do
   begin
      A[k] := read real;
      sum := sum + A[k]
   end;
   writecr;
   write({ddddd.dddd}, sum);
   writecr;
end;
```

Vi har her stadigvæk parameterstyringen, hvor k gaar fra 1 til 10, men den sætning, som styres heraf, er sammensat af to sætninger:

```
A[k] := read real;
```

som læser element nr. k, og som før:

```
sum := sum + A[k];
```

der summerer. Da parameterstyringen skal opfatte de to sætninger som en helhed, er de omgivet af begin og end.

Vi giver endnu et par eksempler paa brug af talsæt og for-sætninger.

Vi tænker os først, at vi har et apparat, hvortil der føres to gas-strømme, og vi skal beregne mængden af hver komponent i blandingen. Tallene kan f.eks. være:

	Strøm nr. 1	Strøm nr. 2	Blanding
Brint	134.17,	71.95,	....
Kulilte	26.81,	94.64,	....
Kuldioxyd	14.89,	88.15,	....
Vand	169.55,	61.23,	....

Vi kender tallene i de to første søjler og skal beregne den sidste søjle. Vi antager, at tallene er hullet i den her viste form, saaledes at de paa strimlen staar i rækkefølgen: brint i strøm 1, brint i strøm 2, kulilte i strøm 1, o.s.v.

Programmet kan se saaledes ud:

```
begin
   integer k;
   array Strøm1, Strøm2, Blanding[1:4];
   select(8);
   for k := 1 step 1 until 4 do
   begin
      Strøm1[k] := read real;
      Strøm2[k] := read real;
      Blanding[k] := Strøm1[k] + Strøm2[k];
      writecr;
      write({ddddd.dd}, Strøm1[k], Strøm2[k], Blanding[k])
   end;
   writecr;
end;
```

For hver komponent vil programmet læse mængden af den paagældende komponent i strøm nr. 1 og i strøm nr. 2, og derefter trykke disse tal igen og deres sum. Bemærk den lille finesse ved deklarationen af de tre talsæt:

```
array Strøm1, Strøm2, Blanding[1:4];
```

Naar der findes flere talsæt af samme dimension og samme grænser, behøver disse kun at anføres efter navnet paa det sidste talsæt.

En udskrift af resultatstrimlen fra dette program vil se saaledes ud:

```
134.17   71.95  206.12
 26.81   94.64  121.45
 14.89   88.15  103.04
169.55   61.23  230.78
```

Vi skal senere vise, hvorledes man kan forsyne resultatet med forklarende tekst, f.eks. komponentnavne.

Det andet eksempel handler om beregning af entalpi af gasser og gasblandinger. Entalpien er det samme som varmeindholdet. Entalpien af en gas vokser med stigende temperatur, idet entalpitilvæksten pr. grads temperaturstigning er lig med gassens varmefylde.

Hvis gassens varmefylde er konstant, Cp kcal/kgmolC, vil entalpien, H kcal/kgmol, vokse lineært med temperaturen, t grader Celsius:

```
H := Cp×t + I;
```

I er en integrationskonstant, d.v.s. det absolutte indhold af entalpi er for saa vidt ubestemt, indtil man har defineret en passende referencetilstand. Dette spiller ingen rolle her, da vi normalt kun har brug for forskellen i entalpi ved to forskellige temperaturer, og da gaar værdien af I ud af regningerne.

Til mange anvendelser vil det lineære udtryk: Cp×t + I ikke være nøjagtigt nok, idet Cp faktisk ændrer sig noget med temperaturen. Man plejer derfor at indføre eet eller flere led af højere grad i t, hvorved H lettere kan bringes til at passe med eksperimentelt maalte værdier. Iøvrigt bruger man ofte den absolutte temperatur: $T = t + 273.16$ (grader Kelvin) i stedet for t.

Ved beregninger af entalpi hos Haldor Topsøe medtager vi ofte led op til T i fjerde potens, og H kan da beregnes som:

```
H := a[0] + a[1]×T + a[2]×T↑2 + a[3]×T↑3 + a[4]×T↑4;
```

Vi siger, at H beregnes af et fjerdegradspolynomium i T med de 5 koefficienter: a[0], a[1], ..., a[4]. Et eksempel paa fastlæggelse af koefficienterne er vist i Kjær (1963d), pag. 8. De 5 koefficienter for metan er der beregnet til:

a[0]	-19625.11
a[1]	3.359595
a[2]	$8.495905_{10}-3$
a[3]	$-7.112638_{10}-7$
a[4]	$-2.548678_{10}-10$

Et program til beregning af H kan se saaledes ud for metan:

```
begin
   real H, T;
   array a[0:4];
   select(8);
   a[0] := -19625.11;
   a[1] := 3.359595;
   a[2] := 8.495905⏨-3;
   a[3] := -7.112638⏨-7;
   a[4] := -2.548678⏨-10;
   T := read real;
   H := a[0] + a[1]×T + a[2]×T↑2 + a[3]×T↑3 + a[4]×T↑4;
   writecr;
   write(⟨-ddddd⟩, H);
   writecr;
end;
```

Programmet begynder med at deklarere de to simple variable: H og T og talsættet: a[0:4]. Derefter vælger det de sædvanlige ydre enheder og tildeler de fem a-koefficienter deres rigtige værdier.

Sætningen: T := read real; sørger for indlæsning af en værdi af T fra datastrimlen. Derefter beregnes H af polynomiet, og der trykkes en ny linie og den beregnede værdi af H.

Den lange sætning til beregning af H er upraktisk af flere grunde: For det første er det ikke nødvendigt udtrykkeligt at skrive navnet paa hvert eneste element i talsættet. Det bør gøres ved hjælp af en for-sætning. For det andet er det ikke nødvendigt at skrive T-potenserne udtrykkeligt som T↑4, T↑3, o.s.v., dette giver alt for mange multiplikationer. Den sidste fejl kan rettes, hvis vi skriver H-beregningen saaledes:

```
H := a[0] + T×(a[1] + T×(a[2] + T×(a[3] + T×a[4])));
```

Man ser, at rytmen i beregningen er multiplikation med T og addition af en koefficient. Dette skal gøres ialt 4 gange idet man starter med værdien a[4]. H-beregningen kan da skrives som:

```
H := a[4];
for k := 3 step -1 until 0 do
H := T×H + a[k];
```

Den nye variable, k, maa deklareres som integer i begyndelsen af programmet. For-sætningens egentlige sætning: H := T×H + a[k]; gennemløbes ialt 4 gange, nemlig for k = 3, 2, 1 og 0. Hver gang multipliceres det hidtidige indhold af H-cellen med T, og der adderes elementet a[k]. Summen flyttes tilbage i H-cellen.

Hvis der i stedet for et rent stof som metan foreligger en blanding af f.eks. 8 gasformige komponenter med de tilhørende molbrøker:

m[1], m[2], ..., m[8]

maa programmet ændres:

```
begin
   integer k, n;
   real h, H, T;
   array m[1:8], a[1:8, 0:4];
```

```
select(8);
for n := 1 step 1 until 8 do
for k := 0 step 1 until 4 do
a[n,k] := read real;
for n := 1 step 1 until 8 do
m[n] := read real;
T := read real;
H := 0;
for n := 1 step 1 until 8 do
begin
   h := a[n,4];
   for k := 3 step -1 until 0 do
   h := T×h + a[n,k];
   H := H + h×m[n]
end;
writecr;
write({-ddddd}, H);
writecr;
end;
```

Programmet begynder med deklaration af heltallene k og n og af de almindelige tal h, H og T. Derefter deklareres molbrøktalsættet m med 8 elementer og a-koefficienterne, som vi nu har givet en dimension mere, saaledes at det første index giver nummeret (1-8) paa komponenten og det andet index som før svarer til graden af T. Nu er der altsaa ialt 8×5 = 40 a-koefficienter.

For ikke at gøre programmet for uoverskueligt lader vi det begynde med at indlæse de 40 a-koefficienter med den dobbelte for-sætning:

```
for n := 1 step 1 until 8 do
for k := 0 step 1 until 4 do
a[n,k] := read real;
```

Den dobbelte for-sætning fungerer paa samme maade som den dobbelte entalpi-for-sætning, der forklares nedenfor. Derefter indlæses de 8 værdier af m:

```
for n := 1 step 1 until 8 do
m[n] := read real;
```

Endelig indlæses T-værdien.

H nulstilles først, og derefter kommer der to for-sætninger indeni hinanden. Den inderste styres af k som før:

```
for k := 3 step -1 until 0 do
h := T×h + a[n,k];
```

Den ydre for-sætning styres af n og gennemløbes 8 gange, een for hver komponent. I hvert gennemløb beregnes entalpien, h, af 1 kgmol af den paagældende komponent. Værdien af h multipliceres med molbrøken og adderes i H-cellen. Trykningen af H er uændret.

Til slut et par faldgruber ved talsæt. I eksemplet side 35 optraadte de tre endimensionale talsæt: Strøm1, Strøm2 og Blanding, hver med 4 elementer. Hvert element i talsættet: Blanding blev beregnet ved addition af de tilsvarende elementer i Strøm1 og Strøm2. En matematiker ville sige, at vektoren: Blanding er summen af de to vektorer: Strøm1 og Strøm2. I ALGOL kan man <u>ikke</u> skrive:

```
Blanding := Strøm1 + Strøm2;
```

i det haab at maskinen saa ville addere de to vektorer. Oversætterprogrammet vil straks protestere mod en saadan konstruktion. Med andre ord: programmøren maa udtrykkeligt forklare, hvilke regneregler, som gælder for de talsæt, han har deklareret, f.eks. ved en for-sætning med sætningen:

```
Blanding[k] := Strøm1[k] + Strøm2[k];
```

Vi skal senere se, hvorledes man ved hjælp af procedurebegrebet kan opnaa omtrent den samme effekt som om vektoraddition umiddelbart kunne anvendes.

De anvendte indices for et talsæt skal danne en sammenhængende talrække. Man kan <u>ikke</u> deklarere et array som f.eks. kun indeholder elementerne:

B[2]
B[3]
B[4]
B[12]
B[13]

Hvis man ønsker at bruge netop disse indices: 2, 3, 4, 12 og 13, maa talsættet deklareres som:

<u>array</u> B[2:13];

og programmet vil reservere plads til alle 12 B-celler. De mellemliggende celler: B[5] - B[11] vil da staa tomme, men kan ikke udnyttes til lagring af andre variable.

Den samme skavank optræder ved flerdimensionale talsæt. Har vi f.eks. talsættet:

C[1,1]	C[1,2]	C[1,3]
C[2,1]	C[2,2]	×
C[3,1]	×	×

hvori de tre elementer mærket med × ønskes udeladt, vil deklarationen <u>array</u> C[1:3, 1:3] reservere plads til alle 9 elementer. Her kan man redde situationen ved i stedet at bruge et 1-dimensionalt array, idet man blot sørger for at finde en eentydig forbindelse mellem de to talsæt, f.eks:

C[1,1] svarer til D[1]
C[1,2] - - D[2]
C[1,3] - - D[3]
C[2,1] - - D[4]
C[2,2] - - D[5]
C[3,1] - - D[6]

Man kan vise, at elementet C[i,j] her svarer til elementet D[(8-i)×(i-1)/2 + j].

Programmer med talsæt, hvis dimension ikke kendes paa forhaand, kan ikke uden videre programmeres i ALGOL. Man kan ikke skrive saadan:

array A[1:6, 1:6, ..., 1:6] ialt p dimensioner;

Hvis man ikke ønsker at lave eet program for 1 dimension med A[1:6], et andet for to dimensioner med A[1:6, 1:6] o.s.v., kan man inddele programmet i afsnit: eet med det første array, et andet med det næste, o.s.v. Men man kommer ikke uden om, paa een eller anden maade at angive det højeste antal dimensioner, som kan forekomme. Er dette f.eks. 5, kan man deklarere:

array A[1:n1, 1:n2, 1:n3, 1:n4, 1:n5];

Bruges kun de tre første dimensioner, maa man sætte n1 := n2 := n3 := 6 og n4 := n5 := 1. Men det koster temmelig meget i regnetid at udregne de overflødige indices.

Hvis maskinen i det oversatte program faar fat i et index, som ligger uden for de opgivne grænser, fremkommer der fejludskriften: index paa skrivemaskinen, og beregningen stoppes. Naar maskinen i det oversatte program skal gøre plads til et array a[p:q], beregner den antallet af celler som q - p + 1. Hvis dette antal er negativt, faar man fejludskriften: array og stop. Er antallet 0, f.eks.: a[7:6], faar man ogsaa fejludskriften: array i GIER ALGOL 4, medens man i de tidligere GIER ALGOL II og III ikke fik fejludskrift.

Inden oversættelsen af ALGOL-programmet kan man bede oversætteren om at udelade index-kontrollen. Dette giver en hurtigere regnetid for det oversatte program.

Betegnelsen array er i virkeligheden en forkortelse for real array, idet man har vedtaget, at ordet real her kan udelades. Hvis elementerne skal opfattes som heltal, er deklarationsbetegnelsen: integer array.

4.5. Betingelsessætninger.

I eksemplet med indlæsning af 10 tal og beregning og trykning af deres sum saa vi, hvorledes programmet var i stand til at udføre en tælling fra 1 til 10. Det er klart, at der heri er skjult en egenskab ved programmet, nemlig dets evne til at være følsom for betingelser. Programmet afgør om den variable, k, har naaet værdien 10, ja eller nej.

I andre tilfælde maa de betingelser, som man ønsker at anbringe i programmet, udtrykkes mere direkte. Som eksempel tager vi et program, som skal læse 10 tal fra en hulstrimmel og trykke deres sum, ligesom tidligere, men nu udvider vi programmet, saaledes at det undersøger, om tallene er større end 100. Hver gang programmet finder et tal større end 100, skal det skrive nummeret paa tallet (fra 1 til 10) og værdien af selve tallet. Programmet kan se saaledes ud:

```
begin
   real sum;
   integer k;
   array A[1:10];
   select(8);
   sum := 0;
   for k := 1 step 1 until 10 do
   begin
      A[k] := read real;
      sum := sum + A[k];
      if A[k] > 100 then
      begin
         writecr;
```

```
            write({dd}, k);
            write({ddddd.dddd}, A[k])
         end
      end;
      writecr;
      write({ddddd.dddd}, sum);
      writecr;
   end;
```

Dette program adskiller sig kun fra den tidligere version, side 34, ved at der er tilføjet sætningen:

```
if A[k] > 100 then
begin
   writecr;
   write({dd}, k);
   write({ddddd.dddd}, A[k])
end;
```

Naar maskinen i det oversatte program kommer til denne sætning, sker der følgende. Betingelsen A[k] > 100 undersøges. Hvis A[k] > 100, udfører maskinen den sætning, som kommer efter then, altsaa i dette tilfælde de tre sætninger, som er anbragt mellem begin og end. De tre sætninger bevirker udskrift af tegnet for ny linie, udskrift af værdien af k som et tal med højst to cifre, og endelig trykning af det aktuelle element, A[k], paa samme maade som summen trykkes til sidst.

Hvis A[k] ≤ 100, sker der ingenting, idet maskinen da overspringer de tre sætninger mellem begin og end.

En sætning af denne art kaldes en betingelsessætning. Den kan formelt skrives som:

```
if <logisk udtryk> then <sætning>;
```

og betyder, at hvis det logiske udtryk, som er anbragt mellem if og then er sandt, saa skal sætningen udføres. Er det logiske udtryk falsk, skal sætningen ikke udføres, men springes over, og maskinen gaar videre til næste sætning. Der bliver altsaa her truffet valget mellem at gøre noget

eller at lade være, men betingelsessætningen kan ogsaa udformes saaledes, at der træffes et valg mellem at gøre det ene eller at gøre det andet:

if <logisk udtryk> **then** <sætning 1> **else** <sætning 2>;

Hvis det logiske udtryk er sandt, skal sætning 1 udføres og sætning 2 springes over. Er det logiske udtryk falsk, springes sætning 1 over og sætning 2 udføres.

Hvad vi her har kaldt sætning 1 og sætning 2 kan godt være sammensatte sætninger eller betingelsessætninger. Sammensatte sætninger skal naturligvis omgives af **begin** og **end**, men der gælder den særlige regel, at hvis sætning 1 er en betingelsessætning, skal den have et ekstra sæt **begin** og **end** omkring sig idet ordet **if** ikke maa komme lige efter ordet **then**. Derimod maa der gerne komme et **if** lige efter **else**.

I kemiske og tekniske programmer vil betingelsessætninger normalt kun forekomme hist og her, idet hovedmængden af sætningerne udfører de egentlige numeriske beregninger. Vi giver derfor ikke noget specielt kemisk programeksempel, som i særlig grad illustrerer brugen af betingelsessætninger, men kun et enkelt eksempel paa linie med de foregaaende og henviser iøvrigt læseren til eksempler senere i bogen, hvor betingelsessætninger er brugt i næsten alle programmer.

Vi tænker os et program, der skal kunne læse seks tal fra en hulstrimmel, nemlig en gasanalyse for en ammoniaksyntesegasblanding. Som eksempel kan vi tage analysen:

Brint	67.62
Kvælstof	21.79
Ammoniak	3.00
Argon	4.12
Metan	3.47
Helium	0.00

Hvis strimlen med gasanalysen skal bruges som input (sammen med yderligere tal) til et større program, der f.eks. beregner produktionen i en

ammoniakkonverter, kan det ofte være praktisk at lade maskinen undersøge, om analysen ser rimelig ud. Vi laver et lille program, der læser de seks tal og undersøger, om forskellige betingelser er opfyldt.

For det første vil man forlange, at summen af mængderne i analysen skal give 100 procent. Dette er dog ikke saa lige til som det lyder, idet man maa regne med, at der kan være smaa afrundingsfejl i talmaterialet, d.v.s. maaske en fejl paa 0.01 til 0.02 i summen. Da maskinen iøvrigt regner med 8-9 betydende cifre og der ikke kan undgaas afrundingsfejl ved selve regneprocesserne, har det i hvert fald ikke nogen mening at forlange større nøjagtighed end ca. $1_{10}-6$ procent. Her sætter vi den tilladelige fejl til 0.02.

Dernæst vil vi forlange, at hver analyse skal ligge mellem 0 og 100 procent.

Endelig skal programmet undersøge forholdet mellem brint og kvælstof. Det skal være i nærheden af 3, men vi er tilfreds, hvis det ligger mellem 1 og 5.

Programmet kan se saaledes ud:

```
begin
   real sum, forhold, element;
   integer k;
   array A[1:6];
   select(8);
   sum := 0;
   for k := 1 step 1 until 6 do
   begin
      element := read real;
      A[k] := element;
      writecr;
      write({ddd.dd}, element);
      if element < 0 then writetext({< Negativ});
      if element > 100 then writetext({< Over 100});
      sum := sum + element
   end;
```

```
        writecr;
        writecr;
        write(⸢ddd.dd⸣, sum);
        if abs(sum-100) > 0.02 then writetext(⸢<  Fejl⸣);
        forhold := A[1]/A[2];
        if forhold < 1 ∨ forhold > 5 then
        begin
           writecr;
           writetext(⸢<Skævt forhold⸣)
        end;
        writecr;
    end;
```

Programmet begynder med at deklarere de tre almindelige variable: sum, forhold og element, samt heltallet: k. Talsættet A deklareres til at have seks elementer.

Sumcellen nulstilles, og der følger da den sædvanlige for-sætning, hvor k varieres fra 1 til 6. For hver værdi af k sker følgende. Det næste tal læses fra strimlen og gemmes i cellen med: element og derefter i cellen med A[k]. Da vi faar brug for dette element flere gange, er det praktisk ogsaa at have det som en simpel variabel, da det er meget hurtigere for maskinen at hente en simpel variabel (0.12 millisek.) end at hente en indiceret variabel (0.4 millisek.). Til gengæld koster det altsaa en ekstra celle i lageret.

Derefter trykkes tegnet for ny linie og talværdien af det netop læste element. Saa følger den betingede sætning:

```
if element < 0 then writetext(⸢<  Negativ⸣);
```

som udføres, hvis element < 0, og som bestaar i trykning af den tekst, som er skrevet mellem tegnet: ⸢< og tegnet: ⸣. Den næste betingelsessætning giver en lignende fejludskrift, hvis element > 100. Endelig summeres elementet i sumcellen.

Efter det sjette gennemløb trykkes to nye linier og den beregnede sum. Betingelsessætningen:

```
if abs(sum-100) > 0.02 then writetext(⸢<  Fejl⸣);
```

bevirker trykning af ordet: Fejl, hvis sum > 100.02 eller sum < 99.98.

Derefter beregnes forholdet mellem brint og kvælstof, og vi faar den sidste betingelsessætning, som bevirker trykning af ny linie og teksten: Skævt forhold, hvis forholdet er mindre end 1 eller større end 5.

I konstruktionen:

$$\underline{\text{if}}\ \text{forhold} < 1 \vee \text{forhold} > 5\ \underline{\text{then}}$$

er benyttet ALGOL-tegnet for eller: $\vee$. Udtrykket $a \vee b$ er sandt, hvis a eller b (eller begge) er sand, ellers er det falsk.

En oplagt faldgrube for begyndere findes, naar man skal udtrykke betingelsen, at en variabel ligger i et bestemt interval, f.eks. at p skal ligge mellem 10 og 20:

$$10 \leq p \leq 20$$

Det er i ALGOL <u>ikke</u> tilladt at skrive:

$$\underline{\text{if}}\ 10 \leq p \leq 20\ \underline{\text{then}}\ \ldots\ldots$$

De to relationer: $10 \leq p$ og $p \leq 20$ skal skrives adskilt, men forbundet med ALGOL-tegnet for og:

$$\underline{\text{if}}\ 10 \leq p \wedge p \leq 20\ \underline{\text{then}}\ \ldots\ldots$$

Udtrykket $a \wedge b$ er sandt, hvis baade a og b er sande, ellers er det falsk.

De logiske udtryk, som forekommer i betingelsessætninger, vil normalt være relationer, som kendes fra matematikken:

$$x < 17.4$$
$$x \leq 24.2$$
$$x = 98$$
$$x \neq 13$$
$$x \geq 9.8$$
$$x > 1.3$$

d.v.s. mindre end, mindre end eller lig med, lig med, forskellig fra, større end eller lig med og større end.

Man kan ogsaa bruge sammensatte logiske udtryk, hvori indgaar flere relationer, som f.eks. $10 \leq p \wedge p \leq 20$, der er nævnt oven for. Hvis vi i almindelighed har to relationer, a og b, som hver for sig kan være sand eller falsk, kan vi i ALGOL operere med følgende logiske kombinationer af a og b:

$a \wedge b$ læses:	a og b. Er sand, hvis a og b begge er sande, ellers falsk.
$a \vee b$ læses:	a eller b. Er sand, hvis a eller b eller begge er sande, ellers falsk.
$a \Rightarrow b$ læses:	a implicerer b. Er falsk, hvis a er sand og b falsk, ellers sand.
$a \equiv b$ læses:	a ækvivalent med b. Er sand hvis a og b begge er sande eller begge falske, ellers falsk.

Endelig benyttes betegnelsen -, a for ikke-a, d.v.s. -, a er sand, naar a er falsk og falsk naar a er sand.

Brugen af de logiske operatorer forklares nærmere i kapitel 7.

4.6. Etiketter og Hopsætninger.

Det er karakteristisk for elektroniske beregningsprogrammer, at man som regel ikke kan klare sig med at nedskrive de formler og sætninger, som skal udregnes, i en simpel ubrudt rækkefølge. Ved blot nogenlunde komplicerede beregninger opstaar der ofte den situation, at det kan være nødvendigt at overspringe visse dele af programmet, eller at gaa et stykke baglæns i programmet for at gentage en del af de tidligere sætninger, blot med andre talværdier. Disse forlæns og baglæns hop i programmet vil normalt være betinget af, hvilke talværdier, der fremkommer i løbet af beregningerne eller direkte af hvilke tal, der er opgivet som inputmateriale for den paagældende beregning.

Man har derfor brug for at anbringe en slags stednavne paa de steder i programmet, hvortil man ønsker at hoppe. Disse stednavne betegnes etiketter (**labels**), og man har stort set lov til at vælge betegnelserne frit,

paa samme maade, som vi i det foregaaende har valgt tilfældige navne til vore variable og talsæt. Etiketten skal afsluttes med et kolon.

Der kan anbringes etiketter i begyndelsen af hver sætning. I det tidligere program til summation af 10 tal fra en strimmel (side 34) kan man f.eks. skrive:

```
    begin
       real sum;
       integer k;
       array A[1:10];
       select(8);
START: sum := 0;
LÆS:   for k := 1 step 1 until 10 do
Q17:   begin
P89:      A[k] := read real;
MNp:      sum := sum + A[k]
       end;
TRYK:  writecr;
tt5:   write({ddddd.dddd}, sum);
       writecr;
    end;
```

Anvendt paa denne maade tjener etiketterne ikke noget egentligt regnemæssigt formaal, idet der ikke hoppes til etiketterne nogetsteds. De virker derfor kun som en slags kommentarer, som kan gøre det lettere for programmøren at læse programmet. Maskinen kan absolut ikke forstaa det betydningsmæssige indhold af ord som: START, LÆS, TRYK, o.s.v.

Programmet ovenfor læser 10 tal, trykker deres sum, og standser saa. Ved passende tryk paa startknappen vil programmet begynde forfra og læse 10 nye tal. Hvis vi vil have programmet til automatisk at blive ved, kan vi programmere saaledes:

```
    begin
       real sum;
       integer k;
       array A[1:10];
       select(8);
```

```
ST:     sum := 0;
        for k := 1 step 1 until 10 do
        begin
           A[k] := read real;
           sum := sum + A[k]
        end;
        writecr;
        write({ddddd.dddd}, sum);
        go to ST
     end;
```

Her har vi tilføjet sætningen:

```
go to ST;
```

Naar maskinen naar hertil i det oversatte program, vil den hoppe tilbage til det sted, hvor der staar: ST, d.v.s. den begynder forfra. Sætningen kaldes en hopsætning, og bestaar i sin simpleste form af ALGOL-glosen: go to samt en etikette.

Nu vil programmet blive ved med at læse grupper paa ti tal og trykke deres sum, indtil inputstrimlen slipper op eller operatøren manuelt standser maskinen. Ønsker vi, at maskinen skal stoppe af sig selv, naar den er færdig med alle tallene, maa den paa een eller anden maade have besked om, hvor mange tal, der staar paa strimlen. Vi kan f.eks. lade det første tal paa strimlen angive, hvor mange talgrupper paa 10 elementer, som findes. Man kan da lave en ny for-sætning, f.eks. saaledes:

```
begin
   real sum;
   integer k, gruppeantal, n;
   array A[1:10];
   select(8);
   gruppeantal := read integer;
   for n := 1 step 1 until gruppeantal do
   begin
      sum := 0;
      for k := 1 step 1 until 10 do
      begin
```

```
            A[k] := read real;
            sum := sum + A[k]
        end;
        writecr;
        write(⟨ddddd.dddd⟩, sum);
    end;
    writecr;
end;
```

Vi har her indført en ny heltalsvariabel, n, som bruges til at tælle gruppeantallet. Bemærk, hvorledes vi nu ikke behøver at bruge hopsætningen go to ST, og derfor helt kan undvære etiketten: ST. Det er iøvrigt en udbredt begynderfejl at forsyne programmerne med alt for mange etiketter. Hvis en etikette slet ikke bruges til at hoppe til i et givet program, bør den udelades eller erstattes med en rigtig kommentar. I GIER-ALGOL-oversætteren optager en etikette to celler af maskinens hukommelse, og det er her ligegyldigt, om den bruges eller ej. I eksemplet side 50 tabes der saaledes 14 celler til overflødige etiketter.

En rigtig kommentar skrives som en sætning, der indledes med ordet: comment og slutter med semikolon:

```
...........
...........
a := b + c;
q := sqrt(r↑2 + s↑2);
comment q er afstanden mellem punkterne;
v := q×h;
...........
...........
```

Naar oversætterprogrammet læser dette, vil det simpelthen overspringe hele linien med: comment o.s.v. indtil og med semikolonet. Kommentaren findes altsaa slet ikke i det oversatte program og belaster derfor ikke hukommelsen. Man kan ogsaa anbringe kommentarer efter ordet end og her er det ikke nødvendigt at begynde med ordet comment. Kommentarer efter end afsluttes af semikolon eller else eller end.

Som et andet eksempel paa brug af etiketter og hopsætninger giver vi en variant af programmet side 46, som udførte kontrol af en gasanalyse. Programmet skal stadigvæk læse mængden af de seks komponenter som et tal-

sæt A[1:6], men vi ønsker nu, at der paa inputstrimlen skal staa en oplysning om, hvilken art af kontrolundersøgelse, der skal udføres. Vi vedtager derfor, at inputstrimlen skal begynde med et heltal, som vi kalder TYPE, og som kan have de tre værdier: 1, 2 eller 3. Hvis TYPE = 1, ønsker vi at kontrollere, at summen af procenterne er 100. For TYPE = 2, skal programmet undersøge, om hvert tal ligger i intervallet fra 0 til 100. For TYPE = 3, skal forholdet mellem brint og kvælstof kontrolleres.

Programmet kan se saaledes ud:

```
    begin
       real sum, forhold, element;
       integer k, TYPE;
       array A[1:6];
       select(8);
ST:    TYPE := read integer;
       for k := 1 step 1 until 6 do
       A[k] := read real;
       if TYPE = 2 then go to T2;
       if TYPE = 3 then go to T3;
T1:    sum := 0;
       for k := 1 step 1 until 6 do sum := sum + A[k];
       if abs(sum-100) > 0.02 then
       begin
          writecr;
          writetext({<SUM IKKE 100})
       end;
       go to SLUT;
T2:    for k := 1 step 1 until 6 do
       begin
          element := A[k];
          if element < 0 ∨ element > 100 then
          begin
             writecr;
             writetext({<FEJL I KOMPONENT});
```

```
            write(⟨dd⟩, k)
          end fejludskrift
        end for k;
        go to SLUT;
T3:     forhold := A[1]/A[2];
        if forhold < 1 ∨ forhold > 5 then
        begin
          writecr;
          writetext(⟨<SKÆVT FORHOLD⟩)
        end;
SLUT: writecr;
   end af programmet;
```

Programmet begynder med etiketten: ST og sætningerne:

```
TYPE := read integer;
for k := 1 step 1 until 6 do
A[k] := read real;
```

som læser talværdien af TYPE fra strimlen samt de seks elementer af A. Bemærk, hvorledes indlæsningen af TYPE sker med den særlige betegnelse: read integer.

Resten af programmet er delt i tre dele, een for hver af de tre mulige værdier af TYPE. De tre programdele begynder ved de tre etiketter: T1, T2 og T3. Vi sørger for, at der hoppes til den rigtige etikette ved at indføre de to betingede hopsætninger:

```
if TYPE = 2 then go to T2;
if TYPE = 3 then go to T3;
```

Hvis værdien af TYPE er 2 eller 3, vil eet af disse hop blive udført. I modsat fald sker der ikke noget, d.v.s. maskinen naar frem til etiketten T1. Bemærk, at vi her slet ikke udnytter, at TYPE er lig med 1, d.v.s. hvis vi paa strimlen har skrevet 0 eller 4 eller 117 som talværdien for

TYPE, faar det samme effekt, som hvis vi havde skrevet TYPE = 1.

Ved etiketten T1 udføres programmet for TYPE = 1 med nulstilling af sumcellen, summation ved hjælp af for-sætningen og fejludskrift, hvis afvigelsen fra 100 er større end 0.02. Sidst i dette programafsnit staar sætningen:

```
go to SLUT;
```

som bevirker hop til etiketten: SLUT allersidst i programmet for at vi kan springe over de to programafsnit svarende til TYPE = 2 og 3.

Ved etiketten T2 begynder programmet for TYPE = 2. Det bestaar af en enkelt for-sætning, som kontrollerer hvert af de seks elementer og laver fejludskrift, hvis nødvendigt. Efter for-sætningen kommer den samme hopsætning: go to SLUT; som afsluttede første programafsnit. Bemærk, hvorledes vi har skrevet kommentarer efter de to end-gloser:

```
    end fejludskrift
end for k;
```

Dette er især nyttigt, hvis der er mange begin - end konstruktioner indeni hinanden.

Det tredje programafsnit (for TYPE = 3) begynder ved etiketten: T3. Værdien af forholdet beregnes, og der laves fejludskrift, hvis nødvendigt. Her er det ikke nødvendigt at afslutte med: go to SLUT, fordi afsnittet jo ender ved SLUT. Bemærk iøvrigt, at hvis A[2] = 0, kan beregningen ikke gennemføres, fordi man faar division med nul, som giver fejludskriften: spill paa skrivemaskinen og stop. Det havde været bedre at kode:

```
T3:     if A[2] ≠ 0 then forhold := A[1]/A[2];
        if A[2] = 0 ∨ forhold < 1 ∨ forhold > 5 then
        begin
           .......
           o.s.v.
```

Vi skal nu se paa et par varianter af dette program. Først viser vi, hvorledes vi helt kan undgaa brugen af etiketter:

```
begin
    real sum, forhold, element;
    integer k, TYPE;
    array A[1:6];
    select(8);
    TYPE := read integer;
    for k := 1 step 1 until 6 do
    A[k] := read real;
    if TYPE = 1 then
    begin
        sum := 0;
        for k := 1 step 1 until 6 do sum := sum + A[k];
        if abs(sum-100) > 0.02 then
        begin
            writecr;
            writetext({<SUM IKKE 100})
        end fejludskrift
    end if TYPE 1
    else
    if TYPE = 2 then
    begin
        for k := 1 step 1 until 6 do
        begin
            element := A[k];
            if element < 0 ∨ element > 100 then
            begin
                writecr;
                writetext({<FEJL I KOMPONENT});
                write({dd}, k)
            end fejludskrift
        end for k
    end if TYPE 2
    else
    begin
        forhold := A[1]/A[2];
        if forhold < 1 ∨ forhold > 5 then
        begin
            writecr;
            writetext({<SKÆVT FORHOLD})
        end fejludskrift
```

```
    end TYPE 3;
    writecr;
end af programmet;
```

I dette tilfælde vil TYPE = 1 og TYPE = 2 give den ønskede kontroltype medens den tredje kontroltype faas for alle andre værdier af TYPE. Man har naturligvis lov til at ændre programmet, saaledes at det sidste afsnit kun udføres for TYPE = 3 ved at indføre betingelsen:

```
if TYPE = 3 then
........
```

Der maa da indføres endnu et else med fejludskrift, hvis TYPE ikke har været 1, 2 eller 3:

```
    ........
    end if TYPE 3
    else
    begin
       writecr;
       writetext({<FORKERT TYPE})
    end forkert type;
    writecr;
end af programmet;
```

Den sidste variant af programmet skal illustrere det tilfælde, at vi ønsker at kunne udføre flere af kontrolundersøgelserne samtidigt. For tre undersøgelser bliver der ialt 2↑3 = 8 muligheder, som vi kan betegne med tallene fra 0 til 7, f.eks. saaledes:

TYPE	Programafsnit		
	T1	T2	T3
0	nej	nej	nej
1	ja	nej	nej
2	nej	ja	nej
3	ja	ja	nej
4	nej	nej	ja
5	ja	nej	ja
6	nej	ja	ja
7	ja	ja	ja

Afsnit T1 skal altsaa udføres, hvis TYPE er ulige. Spørgsmaalet er da, hvorledes man udtrykker, at et tal er lige eller ulige. Dette er et specielt tilfælde af problemet om et tal gaar op i et andet. Skal vi undersøge, om tallet 7 gaar op i et givet tal: Q, kan vi beregne:

entier(Q/7)×7

altsaa heltalsdelen af Q divideret med 7 og resultatet derefter multipliceret med 7. Hvis det derved fremkomne tal er lig med det oprindelige Q, gaar 7 op i tallet Q.

Man har iøvrigt indført et særligt tegn for heltalsdivision:

Q:7

Resultatet af en heltalsdivision er selv et heltal, som for a:b defineres som (a og b skal selv være heltal):

sign(a/b)×entier(abs(a/b))

Det logiske udtryk:

Q:7×7 = Q

vil være sandt, hvis 7 gaar op i tallet Q, ellers falsk. Der behøves ingen parentes omkring Q:7, da operationerne udføres fra venstre.

Betingelsen, at TYPE skal være ulige, kan derfor skrives som:

TYPE:2×2 ≠ TYPE

hvilket altsaa kan bruges som betingelse for, at programafsnit T1 skal udføres. Paa lignende maade ser man, at betingelsen for at programafsnit T2 skal udføres, kan skrives som:

TYPE:4×2 ≠ TYPE:2

For programafsnit T3 er det simplest at skrive betingelsen som:

TYPE ≥ 4

Formuleret paa denne maade ser programmet saaledes ud:

```
begin
   real sum, forhold, element;
   integer k, TYPE;
   array A[1:6];
   select(8);
   TYPE := read integer;
   for k := 1 step 1 until 6 do
   A[k] := read real;
   if TYPE:2×2 ≠ TYPE then
   begin
      sum := 0;
      for k := 1 step 1 until 6 do sum := sum + A[k];
      if abs(sum-100) > 0.02 then
      begin
         writecr;
         writetext(⟨<SUM IKKE 100⟩)
      end fejludskrift
   end if TYPE 1, 3, 5, 7;
   if TYPE:4×2 ≠ TYPE:2 then
```

```
    for k := 1 step 1 until 6 do
    begin
       element := A[k];
       if element < 0 ∨ element > 100 then
       begin
          writecr;
          writetext(≮<FEJL I KOMPONENT≯);
          write(≮dd≯, k)
       end fejludskrift
    end if TYPE 2, 3, 6, 7;
    if TYPE ≥ 4 then
    begin
       forhold := A[1]/A[2];
       if forhold < 1 ∨ forhold > 5 then
       begin
          writecr;
          writetext(≮<SKÆVT FORHOLD≯)
       end fejludskrift
    end if TYPE 4, 5, 6, 7;
    writecr;
 end af programmet;
```

I GIER ALGOL 4 kan man ogsaa bruge udtrykket TYPE mod 2 for divisionsresten af TYPE med 2. Hvis TYPE er lige er altsaa TYPE mod 2 = 0.

5. SIMPLE PROGRAMEKSEMPLER

I dette kapitel gives to eksempler paa simple programmer, som kan bruges i praksis. Eksemplerne er givet for at indøve læseren i brugen af de ALGOL-begreber, som allerede er gennemgaaet, inden vi tager fat paa procedurebegrebet i næste kapitel.

De to eksempler giver ogsaa anledning til at berøre problemet om, hvorledes man lægger et program til rette, altsaa de overvejelser, som skal gøres, inden man kan nedskrive programmet i ALGOL.

5.1. Beregning af Vandgasligevægten.

Vi ønsker at lave et program, som kan beregne ligevægtssammensætningen af en gasblanding, i hvilken vandgasreaktionen:

$$CO + H_2O \longleftrightarrow CO_2 + H_2$$

kan foregaa. Hvis en gas indeholdende kulilte (CO) og vanddamp (H_2O) ledes over en passende katalysator i temperaturomraadet 300 - 500 gr. Celsius, vil noget af kulilten omsætte sig med vanddampen til kulsyre (CO_2) og brint (H_2). Reaktionsskemaet viser, at 1 molekyle CO reagerer med 1 molekyle H_2O under dannelse af 1 molekyle CO_2 og 1 molekyle H_2. Man anvender normalt overskud af vanddamp. Processen er reversibel, d.v.s. man faar ikke omsat al CO, selv om der er overskud af vanddamp. Man kan vise, at ved en givet temperatur vil der indstille sig et bestemt forhold mellem molbrøkerne af de deltagende stoffer. Hvis vi betegner molbrøkerne med mCO, $mH2O$, $mCO2$ og $mH2$, vil der gælde relationen:

$$K = \frac{mCO2 \times mH2}{mCO \times mH2O}$$

hvor ligevægtskonstanten, K, kun afhænger af temperaturen. Vi vil ikke forklare denne afhængighed nærmere her, men henvise til Kjær (1963d),

pag. 22 ff. hvorefter K kan skrives paa formen:

$$\ln(K) = a + b/T + c\times\ln(T) + d\times T + e\times T\uparrow 2 + f\times T\uparrow 3$$

Beregning af de 6 talkonstanter a-f er ogsaa forklaret paa ovennævnte sted. T er temperaturen i gr. Kelvin:

$$T = t + 273.16$$

hvor t er temperaturen i gr. Celsius.

Antager vi nu, at vi har en gasblanding med temperaturen, t, og med molbrøkerne mOCO, mOH2O, mOCO2 og mOH2, samt at ligevægten ikke er indtraadt, da vil forholdet:

$$\frac{mOCO2\times mOH2}{mOCO\times mOH2O}$$

være forskelligt fra K. Hvis ligevægtsindstillingen bevirker, at molbrøken af CO falder med beløbet delta fra mOCO til mCO, da vil molbrøken af vanddamp falde med et tilsvarende beløb, og molbrøkerne af CO2 og H2 vil begge vokse med beløbet delta. Naar ligevægten er indtraadt, har man da:

$$K = \frac{(mOCO2+delta)\times(mOH2+delta)}{(mOCO-delta)\times(mOH2O-delta)}$$

Dette er en andengradsligning i den ubekendte: delta. Programmet kan godt indrettes saaledes, at det løser andengradsligningen efter de sædvanlige formler herfor. Der er dog et par smaa vanskeligheder forbundet hermed. Dels vil der normalt være to rødder i andengradsligningen, og man skal da give anvisning paa, hvorledes maskinen skal vælge mellem de to rødder. Dels skal man undersøge det specielle tilfælde, hvor andengradsligningen degenererer til en førstegradsligning. Der kræves altsaa en nærmere analyse, som vi delvis undgaar ved at benytte en iterationsmetode. Vi forudsætter, at delta er en lille størrelse, saaledes at man kan se bort

fra leddene med delta↑2. Løser vi ligningen med hensyn til delta, faar vi:

```
                 KxmOCOxmOH2O-mOCO2xmOH2
delta = ---------------------------
                 Kx(mOCO+mOH2O) + mOCO2+mOH2
```

Vi begynder beregningerne med at sætte de søgte ligevægtsmolbrøker: mCO, mH2O, o.s.v. lig med de opgivne molbrøker: mOCO, mOH2O, o.s.v. Derefter beregnes K af temperaturudtrykket og delta af ligningen ovenfor. Molbrøkerne korrigeres derefter:

```
mCO  := mCO  - delta;
mH2O := mH2O - delta;
mCO2 := mCO2 + delta;
mH2  := mH2  + delta;
```

De nye molbrøker indsættes i formlen for delta i stedet for de gamle molbrøker, delta beregnes igen, og molbrøkerne korrigeres, o.s.v. Iterationen afbrydes, naar delta er blevet mindre end en vis værdi. Skal ligevægtsanalysen passe med en usikkerhed paa 0.01 procent, skal delta altsaa komme under 0.0001.

Vi er nu omtrent klar til at kunne nedskrive programmet i ALGOL. Først maa vi dog fastlægge konventioner for inputmaterialet. Dette skal bestaa af temperaturen, t gr. C, samt gasblandingens originale sammensætning. I stedet for at opgive molbrøkerne, mOCO, mOH2O, o.s.v. i den originale blanding, er det mere praktisk at opgive molprocenterne, der er 100 gange større. Vi betegner disse med: MCO, MH2O, o.s.v. Da der i gassen kan være andre end de fire komponenter, som deltager i selve reaktionen, indfører vi ogsaa molprocenten af inerte, MIn, som dog ikke behøver at indlæses, men som programmet selv kan beregne som differens.

Programmet kan se saaledes ud:

```
begin comment  Beregning af vandgasligevægt;
   real t, T, MCO, MH2O, MCO2,  MH2,  MIn,  mCO,  mH2O, mCO2,  mH2,  K,
        delta;
   select(8);
   t := read real;
   MCO := read real;
   MH2O := read real;
   MCO2 := read real;
   MH2 := read real;
   T := t + 273.16;
   K := exp(-0.768535428×ln(T)  + ((((1.47520303⏨-10×T-9.66051861⏨-7)×T
        +3.01017929⏨-3)×T - 1.50650387)×T + 4.94327234⏨3)/T);
   MIn := 100 - MCO - MH2O - MCO2 - MH2;
   comment Hvis det ønskes, kan man her indlægge en kontrol paa, at MIn
   ligger mellem 0 og 100;
   mCO    := 0.01×MCO;
   mH2O   := 0.01×MH2O;
   mCO2   := 0.01×MCO2;
   mH2    := 0.01×MH2;
ST: delta := (K×mCO×mH2O-mCO2×mH2)/(K×(mCO+mH2O) + mCO2 + mH2);
   mCO  := mCO  - delta;
   mH2O := mH2O - delta;
   mCO2 := mCO2 + delta;
   mH2  := mH2  + delta;
   if abs(delta) > 0.0001 then go to ST;
   comment  Nu følger trykning af resultatet;
   writecr;
   writetext({<Temperatur, gr. C});
   write({ddddd}, t);
   writecr;
   writetext({<            Molprocenter:});
   writecr;
   writetext({<         Original    Ligevægt});
   writecr;
   writecr;
   writetext({<Kulilte});
```

```
        write({dddddd.dd}, MCO, 100×mCO);
        writecr;
        writetext({<Vand    });
        write({dddddd.dd}, MH2O, 100×mH2O);
        writecr;
        writetext({<Kulsyre});
        write({dddddd.dd}, MCO2, 100×mCO2);
        writecr;
        writetext({<Brint   });
        write({dddddd.dd}, MH2, 100×mH2);
        writecr;
        writetext({<Inerte });
        write({dddddd.dd}, MIn);
        writecr
    end af programmet;
```

Hvis dette program afprøves med følgende inputstrimmel:

400, 30, 10, 10, 50,

ser resultatudskriften saaledes ud:

```
Temperatur, gr. C  400
            Molprocenter:
         Original   Ligevægt

Kulilte    30.00     23.41
Vand       10.00      3.41
Kulsyre    10.00     16.59
Brint      50.00     56.59
Inerte      0.00
```

Vi knytter følgende bemærkninger til programmet.

I sætningen for beregning af ligevægtskonstanten, K, har vi indsat talværdier for koefficienterne beregnet som beskrevet i Kjær (1963d), pag. 22.

Det centrale i programmet er de seks sætninger, som begynder ved etiketten: ST, og som slutter med den betingede hopsætning:

```
if abs(delta) > 0.0001 then go to ST;
```

Disse seks sætninger vil blive gentaget, indtil delta bliver tilstrækkelig lille.

I trykprogrammet har vi allerede forklaret betydningen af:

```
writecr;
writetext(‹< ›);
write(‹ ›, t);
```

som giver henholdsvis trykning af tegnet for ny linie, trykning af tekst, og trykning af talværdien af den variable, t. I sætningen:

```
write(‹dddddd.dd›, MCO, 100×mCO);
```

skal der trykkes to tal: først MCO og derefter 100×mCO. Begge tal trykkes efter skemaet: dddddd.dd, d.v.s. med højst 6 cifre før kommaet og med 2 decimaler. Her kan der naturligvis højst forekomme 2 cifre før kommaet (evt. 3 i 100.00), men vi ønsker alligevel plads til 6 cifre, fordi vi derved automatisk faar placeret de to tal MCO og 100×mCO i en passende afstand fra hinanden.

Den her anvendte iterationsmetode virker tilfredsstillende, undtagen naar man er meget langt fra ligevægten. I saa fald kan man risikere, at processen slet ikke konvergerer. Der er forskellige metoder til at raade bod herpaa, f.eks. kan man indføre en øvre grænse for delta. Man slipper altsaa ikke helt for matematiske bekymringer ved at bruge den improviserede iterationsmetode. I praksis maa det anbefales at bruge den konventionelle metode til løsning af andengradsligninger. I iterationsmetoden ovenfor har vi - maaske uden at være klar over det - benyttet Newtons metode til løsning af en ulineær ligning, hvori man i nærheden af den søgte rod erstatter funktionen med tangenten til kurven. Man kan let angive, under hvilke betingelser denne metode konvergerer. Der er grund til i almindelighed at advare imod at opfinde smaa matematiske fiduser under programmeringsarbejdet, idet det ofte viser sig, at saadanne fiduser er veldefinerede metoder fra den anvendte matematiks omraade, og man bør derfor prøve

at finde ud af, om metoden er beskrevet i en lærebog over numeriske metoder, for at drage nytte af den specielle viden med hensyn til konvergens o.l. som andre allerede har fundet ud af. Paa den anden side er mange af disse numeriske metoder netop opfundet og udarbejdet af ikke-matematikere, som var i bekneb for en metode til løsning af et praktisk problem, saa hvis det er en ny metode, man har opfundet, findes der naturligvis intet beskrevet om den.

Programmet ovenfor kan let udvides, saaledes at man faar en tabel over ligevægtssammensætningen ved forskellige temperaturer. Der kræves da to nye inputtal: antallet af temperaturer og temperaturtilvæksten. En tilsvarende for-sætning maa da indføres, ligesom trykprogrammet maa ændres. Læseren kan forsøge dette som et øvelseseksempel.

5.2. Beregning af Trykfald i Rør.

Vi skal nu forsøge at lave et program, som kan beregne trykfaldet i et rør, hvori der strømmer en gasblanding. Inden vi kan nedskrive programmet i ALGOL, er der to vigtige spørgsmaal, som maa opklares: Vi maa finde en passende formel til beregning af trykfaldet, og vi maa afgøre, om programmet skal kunne behandle alle mulige gasblandinger eller kun bestemte typer heraf. Det sidste punkt er vigtigt for fastlæggelsen af formen af inputmaterialet.

Vi ser først paa formlen for trykfaldet. Lad os antage, at vi vælger formlen:

$$\mathrm{delP} = \mathrm{rho} \times f \times \frac{L \times V \uparrow 2}{D \times (2g)}$$

med f givet ved:

$$f = \frac{0.184}{\mathrm{NRe} \uparrow 0.2}$$

hvori vi benytter betegnelserne:

delP Trykfald (kg/m↑2)
rho Gasvægtfylde (kg/m↑3)
L Rørlængde (m)
D Rørdiameter (m)
V Gashastighed (m/h)
g Tyngdeaccelerationen (m/h↑2)
f Friktionsfaktor (dimensionsløs)
NRe Reynolds tal (dimensionsløs)

Reynolds tal er givet ved:

$$NRe = \frac{D \times G}{my}$$

med:

G Gassens massehastighed (kg/(m↑2×h))
my Gassens viskositet (kg/mh)

Det bemærkes her, at friktionsfaktoren, f, er givet ved udtrykket 0.184/NRe↑0.2, medens man ved manuelle beregninger ofte nøjes med at aflæse f paa en kurve. Elektronregnemaskinen kan ikke paa simpel vis aflæse kurver, og man maa derfor ved programmeringen reproducere eventuelle kurver ved passende formler. Der findes veldefinerede matematiske metoder til at udtrykke funktioner af een eller flere variable som f.eks. polynomier. Man kan naturligvis ogsaa nøjes med at lade en tabel over funktionsværdierne indgaa i programmet og saa anvende interpolation.

Vi mangler nu formler for vægtfylden, rho, og viskositeten, my, af gasblandingen. Man kan indrette programmet saaledes, at disse to størrelser opgives i inputmaterialet, men dette er ikke praktisk, fordi den manuelle beregning af viskositeten er en lille smule besværlig og næsten mere omfattende end selve beregningen af trykfaldet. Det maa være en hovedregel for anvendelsen af elektroniske cifferregnemaskiner til rutineopgaver, at beregningen programmeres til bunds, saaledes at den manuelle forbehandling af talmaterialet er saa lille som muligt, helst nul.

Viskositeten af gasblandinger kan beregnes efter det generaliserede diagram i Hougen og Watson (1947), p. 871. For hver komponent i blandingen skal man kende det kritiske tryk:

Pc[1], Pc[2], ..., Pc[COMP]

og den kritiske temperatur:

Tc[1], Tc[2], ..., Tc[COMP]

og den kritiske viskositet:

myc[1], myc[2], ..., myc[COMP]

Antallet af komponenter betegnes med COMP. Blandingens kritiske tryk, Pcmix (atm.), beregnes som:

Pcmix := Pc[1]×m[1] + Pc[2]×m[2] + ... + Pc[COMP]×m[COMP]

hvori m[1], m[2], o.s.v. er molbrøkerne af de enkelte komponenter. Paa lignende maade findes blandingens kritiske temperatur, Tcmix (gr.K):

Tcmix := Tc[1]×m[1] + Tc[2]×m[2] + ... + Tc[COMP]×m[COMP]

og den kritiske viskositet, mycmix (kg/mh), af blandingen:

mycmix := myc[1]×m[1] + myc[2]×m[2] + ... + myc[COMP]×m[COMP]

Disse formler for de kritiske egenskaber af blandingen er kun nogenlunde rigtige.

Naar Hougen og Watsons diagram skal bruges, beregner man først det saakaldte reducerede tryk, Pr, som forholdet mellem det aktuelle tryk og blandingens kritiske tryk:

Pr := P/Pcmix

og analogt for den reducerede temperatur:

Tr := T/Tcmix

T er gassens aktuelle temperatur i gr.K.

Pr og Tr benyttes som indgangsværdier i diagrammet, hvorved man finder den reducerede viskositet, myr. Heraf findes den aktuelle viskositet af blandingen som:

```
my := myr×mycmix
```

Diagrammet giver altsaa funktionen myr i afhængighed af de to variable: Pr og Tr. Det er vist i Kjær (1963d), p. 36, hvorledes funktionen myr kan gengives nogenlunde ved formlerne:

```
myr := a×Tr↑b + c×Tr↑d×Pr        (0≤Pr≤1)
myr := a×Tr↑b + c×Tr↑d + e×Tr↑f×(Pr-1)      (Pr>1)
```

Talværdien for de 6 talkoefficienter a - f findes i ALGOL-programmet nedenfor.

Normalvægtfylden, rho0, af gasblandingen findes ved summation af normalvægtfylderne, rhon, af de enkelte komponenter:

```
rho0 := rhon[1]×m[1] + rhon[2]×m[2] + ... rhon[COMP]×m[COMP]
```

Den aktuelle vægtfylde bliver saa:

```
rho := 273.16/T×P×rho0
```

Vi mangler nu at definere hvilke komponenter, der maa forekomme i gasblandingen ved beregningen. Programmet kan indrettes paa to principielt forskellige maader. Ved den første arbejder vi med en fast komponentliste, f.eks:

Brint
Kvælstof
Ammoniak

Gasanalysen i inputspecifikationen skal da altid opgives som 3 tal, der giver molprocenterne af de 3 stoffer i den nævnte rækkefølge. Er nogle af stofferne ikke til stede, skriver man et nul paa deres plads.

Ved den anden metode kan man have mange flere komponenter i blandingen, og rækkefølgen af stofferne er ligegyldig. Man vedtager da en liste over alle de komponenter, som maa forekomme i blandingen, f.eks:

1 Brint
2 Vand
3 Kvælstof
4 Kvælstofilte
5 Kulilte
6 Kulsyre
7 Argon
8 Metan

Inputmaterialet skal da indeholde antallet, COMP, af komponenter i gasblandingen, samt en nummerliste. Skrives denne liste som:

5
6
1
2
8

betyder det, at der i den paagældende beregning skal regnes med de fem komponenter:

5 Kulilte
6 Kulsyre
1 Brint
2 Vand
8 Metan

Mængderne af de enkelte komponenter skal da anbringes i samme rækkefølge i inputmaterialet.

Vi laver først programmet i den version, som forudsætter en fast komponentliste med netop de tre stoffer: brint, kvælstof og ammoniak. Den tilhørende inputspecifikation kan se saaledes ud:

(1)........... Rørdiameter (mm)
(2)........... Rørlængde (m)
(3)........... Gasmængde (kgmol/h)
(4)........... Temperatur (gr.C)
(5)........... Tryk (atm.abs.)
(6)........... Molprocent brint
(7)........... Molprocent kvælstof
(8)........... Molprocent ammoniak

Brugeren af programmet skal altsaa udfylde dette skema, og de otte tal hulles derefter paa en hulstrimmel og bruges som inputstrimmel til programmet. Bemærk, at vi opgiver rørdiameteren i mm, medens det ovennævnte formelapparat regner med meter. Vi skal altsaa huske at dividere med 1000 i programmet. Man skal opgive den strømmende gasmængde, F, i enheden kgmol/h, hvilket formentlig er mere praktisk end at opgive gashastigheden, V. Sammenhængen mellem F, V og G findes saaledes:

1 kgmol gas fylder 22.415 m↑3 ved 0 gr.C og 1 atm. Normalvægtfylden er rho0 kg/m↑3, saaledes at 1 kgmol gas vejer 22.415×rho0 kg. Den strømmende vægtmængde bliver da F×22.415×rho0 kg/h. Massehastigheden, G, har dimensionen kg/(m↑2×h), altsaa massen per m↑2 tværsnitsareal af røret. Vi skal altsaa dividere med tværsnitsarealet, A m↑2:

G := F×22.415×rho0/A

og A findes af:

A := 0.785398×D↑2

Talfaktoren er pi/4 og D regnes i m.

Ved division af G (i kg/(m↑2×h)) med den aktuelle vægtfylde rho (i kg/m↑3) faar vi gashastigheden V (i m/h):

V := G/rho

Programmet kan nu skrives saaledes:

```
begin comment  Beregning af trykfald;
   real Dmm, L, F,  t,  P, MH2, MN2, MNH3, mh2, mn2, mnh3, D, T, Pcmix,
        Tcmix, mycmix, rho0, Pr, Tr, myr, my,  A,  G,  rho,  V, NRe, f,
        delP;
   select(8);
   Dmm := read real;
   L := read real;
   F := read real;
   t := read real;
   P := read real;
   MH2 := read real;
   MN2 := read real;
   MNH3 := read real;
   comment  Efter indlæsning  af  data omregnes fra  mm  til  m  og fra
            molprocent til molbrøk;
   D     := 0.001×Dmm;
   mh2   := 0.01×MH2;
   mn2   := 0.01×MN2;
   mnh3  := 0.01×MNH3;
   T     := t + 273.16;
   comment  Nu beregnes de kritiske blandingsegenskaber og vægtfylden;
   Pcmix  := 20.8×      mh2 +  33.5×     mn2 + 111.5×      mnh3;
   Tcmix  := 41.26×     mh2 + 126.06×    mn2 + 405.56×     mnh3;
   mycmix :=  0.0125× mh2 +   0.0655× mn2 +    0.1116×  mnh3;
   rho0   :=  0.08994×mh2 +   1.24987×mn2 +    0.759848×mnh3;
   comment  Derefter beregnes my;
   Pr  := P/Pcmix;
   Tr  := T/Tcmix;
   myr := 0.64×Tr↑0.60  +  (if Pr  > 1 then 1.43×Tr↑(-3.98) + 0.275×Tr↑
          (-1.54)×(Pr-1) else 1.43×Tr↑(-3.98)×Pr);
   comment  Bemærk,  hvorledes  vi  her har anbragt betingelsen if then
            else i et regneudtryk.  Det skal omgives af en parentes, og
            værdien af parentesens indhold beregnes, inden det kan ind-
            gaa i det samlede udtryk.  Derefter kommer selve trykfalds-
            beregningen;
```

```
my    := myr×mycmix;
A     := 0.785398×D↑2;
G     := F×22.415×rho0/A;
rho   := 273.16/T×P×rho0;
V     := G/rho;
NRe   := D×G/my;
f     := 0.184/NRe↑0.2;
delP  := rho×f×L×V↑2/D/(2×1.2714₁₀8);
comment Talfaktoren 1.2714₁₀8 er fremkommet ved omregning af tyngde-
        accelerationen 9.81 m/sek↑2 til 9.81×3600↑2 = 1.2714₁₀8
        m/h↑2. Trykfaldet, delP, har nu enheden kg/m↑2. Vil vi
        have det som kg/cm↑2, skal vi dividere med 10000, eller som
        atm. da med 9678;
delP  := delP/9678;
writetext(⟨<
D       L      F       t       P         Molprocenter:       delP
mm      m    kgmol/h  gr.C    atm.     H2      N2     NH3    atm.

⟩);
  write(⟨ddddd.dd⟩, Dmm, L, F, t, P, MH2, MN2, MNH3, delP);
  writecr
end af programmet;
```

Hvis programmet afprøves med følgende inputstrimmel:

12.7, 2, 446.13, 20, 300, 75, 25, 0,

ser udskriften paa linieskriveren saaledes ud:

```
D       L      F       t       P         Molprocenter:       delP
mm      m    kgmol/h  gr.C    atm.     H2      N2     NH3    atm.

12.70   2.00  446.13  20.00  300.00   75.00   25.00   0.00   4.18
```

Trykprogrammet er her indskrænket til det mindst mulige: to tekstlinier med symboler og enheder, og en linie med de 8 inputtal og resultatet. Mellemregninger trykkes ikke.

Vi gaar nu over til det andet tilfælde, hvori programmet skal kunne regne paa op til 8 gasser som forklaret side 71. Inputspecifikationen kan nu laves saaledes:

(1)........... Rørdiameter (mm)
(2)........... Rørlængde (m)
(3)........... Gasmængde (kgmol/h)
(4)........... Temperatur (gr.C)
(5)........... Tryk(atm.abs.)
(6)........... Antal komponenter, COMP

For hver af de COMP komponenter skrives:

	Komponent nummer	Molprocent
(7,8)	....	....
(9,10)	....	....
(11,12)	....	....
(13,14)	....	....
(15,16)	....	....
(17,18)	....	....
(19,20)	....	....
(21,22)	....	....

Numrene tilvenstre for rubrikkerne fortæller hulledamen, i hvilken rækkefølge tallene skal hulles paa strimlen. Har vi en gas med de 3 komponenter kulilte, brint og vand, kan inputmaterialet f.eks. se saaledes ud:

12.7
2
10
500
10
3
5, 20
1, 30
2, 50

altsaa 20 procent CO, 30 procent H2 og 50 procent H2O.

Programmet maa nu ændres, saaledes at der bruges en komponentnummerliste, CN, som er et integer array (talsæt af heltal). I ovennævnte tilfælde er CN[1] = 5, CN[2] = 1 og CN[3] = 2. Dette array er deklareret med grænserne:

CN[1:COMP]

og vi laver et tilsvarende array for molprocenterne: M[1:COMP]. Derimod lader vi molbrøkerne være et komplet array med alle 8 elementer: m[1:8]. Hvis programmet havde indeholdt mange flere end 8 indbyggede komponenter, f.eks. 60 - 70 komponenter, var det naturligvis ikke praktisk med et saa stort array, men det er ikke saa let at programmere med et array m[1:COMP] uden at benytte sig af procedurebegrebet, som vi jo ikke har lært om endnu.

I denne version kan programmet se saaledes ud:

```
begin comment Beregning af trykfald;
   integer COMP, i;
   real Dmm, L, F, t, P, D, T, Pcmix, Tcmix, mycmix, rho0, Pr, Tr, myr,
        my, A, G, rho, V, NRe, f, delP;
   array m[1:8];
   select(8);
   Dmm := read real;
   L := read real;
   F := read real;
   t := read real;
   P := read real;
   COMP := read integer;
   begin
      integer array CN[1:COMP];
      array M[1:COMP];
      D := 0.001×Dmm;
      for i := 1 step 1 until 8 do m[i] := 0;
      for i := 1 step 1 until COMP do
      begin
         CN[i] := read integer;
         M[i] := read real;
         m[CN[i]] := 0.01×M[i]
      end for i;
```

```
        T := t + 273.16;
        Pcmix :=  20.80×  m[1]+218.40×   m[2]+ 33.50×   m[3]+ 65.00×   m[4]
                + 35.00×  m[5]+ 73.00×   m[6]+ 48.00×   m[7]+ 45.80×   m[8];
        Tcmix :=  41.26×  m[1]+647.31×   m[2]+126.06×   m[3]+179.16×   m[4]
                +134.16×  m[5]+304.26×   m[6]+151.16×   m[7]+190.66×   m[8];
       mycmix :=   0.0125×m[1]+  0.1786× m[2]+  0.0655× m[3]+  0.0923× m[4]
                +  0.0685×m[5]+  0.1235× m[6]+  0.0952× m[7]+  0.0580× m[8];
        rho0  :=  0.08994×m[1]+  0.80375×m[2]+  1.24987×m[3]+  1.33875×m[4]
               +  1.24965×m[5]+  1.96346×m[6]+  1.78202×m[7]+  0.71573×m[8];
        Pr := P/Pcmix;
        Tr := T/Tcmix;
        myr := 0.64×Tr↑0.60 + (if Pr > 1 then
        1.43×Tr↑(-3.98) + 0.275×Tr↑(-1.54)×(Pr-1)
        else 1.43×Tr↑(-3.98)×Pr);
        my := myr×mycmix;
        A    := 0.785398×D↑2;
        G    := F×22.415×rho0/A;
        rho  := 273.16/T×P×rho0;
        V    := G/rho;
        NRe  := D×G/my;
        f    := 0.184/NRe↑0.2;
        delP := rho×f×L×V↑2/D/(2×1.2714₁₀8×9678);
        writetext(⟨<
   D        L        F         t        P       delP
   mm       m     kgmol/h    gr.C     atm.      atm.

⟩);
        write(⟨ddddd.dd⟩, Dmm, L, F, t, P, delP);
        writecr;
        writetext(⟨<
Komponent     Molprocent

⟩);
        for i := 1 step 1 until COMP do
        begin
           write(⟨ddddd⟩, CN[i]);
           writetext(⟨<           ⟩);
           write(⟨ddd.dd⟩, M[i]);
```

```
            writecr
         end for i
      end indre blok
   end program;
```

Programmet begynder med deklarationen af de to heltalsvariable: COMP og i, og af de 22 reelle variable og talsættet m[1:8]. Derefter indlæses værdien af Dmm, L, F, t, P og COMP. Bemærk, at vi paa dette tidspunkt endnu ikke har deklareret de to talsæt CN[1:COMP] og M[1:COMP]. Det kan vi ikke gøre, fordi den faktiske talværdi af COMP ikke var kendt ved programmets begyndelse, og maskinen kunne da ikke vide, hvor mange celler, som skulle reserveres til de to talsæt. Efter indlæsningen af talværdien af COMP er vi i stand til at foretage deklarationen, og det sker i de tre linier:

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

Det er vigtigt at bemærke, at vi her maa indføre et ekstra begin, som modsvares af det næstsidste end i programmet:

```
end indre blok
```

Dette ekstra sæt begin - end er nødvendigt, fordi deklarationen kun kan staa i begyndelsen af et programafsnit, et forhold som vil blive forklaret nærmere i kapitel 7.

Efter omregning af Dmm til D kommer en for-sætning:

```
for i := 1 step 1 until 8 do m[i] := 0;
```

som nulstiller de 8 molbrøkceller. Derefter kommer en ny for-sætning, hvor i tælles op til COMP, og som indeholder de tre sætninger:

```
CN[i] := read integer;
M[i] := read real;
m[CN[i]] := 0.01×M[i];
```

De to første sætninger læser en linie med et komponentnummer og den tilhørende molprocent, og den sidste sætning sørger for at gemme molbrøken i den rigtige m-celle.

Beregningen af blandingsegenskaberne:

```
Pcmix := 20.80×m[1] + 218.40×m[2] + o.s.v.
```

fylder temmelig meget skrevet ned paa denne maade. Man kan gøre det lidt simplere, hvis man skriver saaledes:

```
Pcmix := 0;
i := 0;
for R := 20.80, 218.40, 33.50, 65.00, 35.00, 73.00, 48.00,
45.48 do
begin
   i := i + 1;
   Pcmix := Pcmix + R×m[i]
end for R;
```

Størrelsen R maa da være deklareret for <u>real</u> i begyndelsen af programmet. Vi har her brugt en for-sætning, men paa en lidt anden maade end tidligere. Styringen bestaar af en liste af talværdier, som den variable R skal antage. Den sætning, som skal styres, bestaar af de to sætninger:

```
i := i + 1;
Pcmix := Pcmix + R×m[i];
```

Først sættes R lig med 20.80 og de to sætninger udføres. Derefter sættes R lig med 218.40 og sætningerne udføres igen, o.s.v. ialt 8 gange, den sidste for R lig med 45.48.

Bemærk iøvrigt at de to sætninger:

```
Pcmix := 0;
i := 0;
```

ikke kan skrives som een sætning:

```
Pcmix := i := 0;
```

fordi de to variable (Pcmix og i) er af forskellig type (real og integer). Naar maskinen har beregnet et udtryk og skal gemme talværdien af resultatet i en celle reserveret for en integer variabel, afrundes resultatet til nærmeste hele tal. Er det en real, sker der naturligvis ingen afrunding. Derfor maa der ikke staa variable af forskellig type paa venstre side i en sætning, da maskinen saa kommer i vildrede med om den skal afrunde udtrykket eller ej. I det ovennævnte tilfælde er det beregnede udtryk et nul, og man skulle tro, at der ikke var noget problem, men oversætterprogrammet er ikke saa raffineret, at det foretager en speciel undersøgelse af, om højresiden paa forhaand vides at være nul (eller et andet kendt heltal). Man faar derfor fejludskriften: type.

De egentlige beregninger i trykfaldsprogrammet forløber iøvrigt lige som i den første version af programmet. I sætningen til beregning af delP staar der i nævneren udtrykket:

$$2\times 1.2714_{10}8\times 9678$$

Man kan her spørge, om programmøren burde have beregnet talværdien af dette produkt og skrevet dette i stedet. Det er ikke nødvendigt, fordi maskinen under selve oversættelsen vil udregne produktet og anbringe det klar til brug i det oversatte program.

Med det ovennævnte inputmateriale faar man følgende resultatudskrift:

D	L	F	t	P	delP
mm	m	kgmol/h	gr.C	atm.	atm.
12.70	2.00	10.00	500.00	10.00	0.66

Komponent	Molprocent
5	20.00
1	30.00
2	50.00

Trykprogrammet skriver kun numrene paa komponenterne, altsaa de samme numre, som er opgivet som input. De rigtige navne skrives ikke. Ønsker

man dette gjort, kan man ændre programmet, saaledes at sætningerne:

```
write(⟨ddddd⟩, CN[i]);
writetext(⟨<          ⟩);
```

erstattes med følgende:

```
j := CN[i];
writetext(if j = 1 then ⟨<Brint          ⟩
     else if j = 2 then ⟨<Vand           ⟩
     else if j = 3 then ⟨<Kvælstof       ⟩
     o.s.v.

     else if j = 7 then ⟨<Argon          ⟩
     else ⟨<Metan         ⟩);
```

Den nye *integer* variable: j maa deklareres i begyndelsen af programmet. Ogsaa her vil man kunne opnaa en forenkling af programmeringen ved brug af procedurebegrebet, eller ved hjælp af en case-konstruktion som nævnt side 179.

Til sidst et par bemærkninger om selve beregningsmetoden for trykfaldet. Med det her benyttede formelapparat er det aabenbart en forudsætning, at trykket og viskositeten ikke ændrer sig væsentligt gennem røret, saaledes at delP kan udregnes paa een gang uden at korrigere vægtfylden ned gennem røret. Er denne forudsætning ikke opfyldt, maa der udarbejdes et helt nyt program, hvor røret opdeles i smaa stykker, og trykfaldet beregnes ved numerisk integration. I øvrigt kan man med en vis tilnærmelse regne trykfaldet for omvendt proportional med trykket:

$$delP = \frac{konstant}{P}\times L$$

og dette udtryk kan integreres analytisk til at give:

$$PA\uparrow 2 - PB\uparrow 2 = 2\times konstant\times L$$

hvor PA og PB er indgangs- og udgangstrykket. Programmet kan let ændres, saaledes at man i stedet bruger denne afhængighed.

6. PROCEDURER

6.1. Procedurebegrebet.

Ved programmering af større programmer vil man ofte opdage, at man paa forskellige steder i programmet har brug for at anbringe programstumper, som er ens eller næsten ens. Man faar da let den tanke, om det ikke var muligt at nøjes med at skrive saadanne programstumper eet sted, og saa de øvrige steder, hvor de skal bruges, blot at henvise dertil f.eks. med et bestemt navn. Dette opnaas ved brugen af procedurer. Ogsaa her vil vi illustrere begrebet ved hjælp af eksempler, og først efterhaanden give de nøjagtige regler for anvendelsen.

6.2. Beregning af Cylinderrumfang.

Vi tænker os, at vi har et stort program, hvori der mange forskellige steder forekommer beregning af et cylinderrumfang efter den velkendte formel:

```
RUMFANG := 0.785398×HØJDE×DIAMETER↑2;
```

Programmet kan i princippet se saaledes ud:

```
begin
   real V7, H1, Diam6, Vol, P, d, W, hh, DW;
   ........
   ........
   V7 := 0.785398×H1×Diam6↑2;
   ........
   ........
   ........
```

```
    Vol := 0.785398×P×d↑2;
    ........
    ........
    ........
    W := 0.785398×hh×DW↑2;
    ........
    ........
end;
```

Her skal vi altsaa tre steder beregne rumfanget af en cylinder, nemlig med de tre højder: H1, P og hh og med de tre diametre: Diam6, d og DW. De beregnede rumfang hedder: V7, Vol og W.

Vi omskriver nu programmet, saaledes at det indeholder en procedure til beregning af cylinderrumfanget. Vi vælger navnet: CYL til denne procedure, og programmet kan nu se saaledes ud:

```
begin
    real V7, H1, Diam6, Vol, P, d, W, hh, DW;
    procedure CYL(H, D, V);
    real H, D, V;
    V := 0.785398×H×D↑2;
    ........
    ........
    CYL(H1, Diam6, V7);
    ........
    ........
    ........
    CYL(P, d, Vol);
    ........
    ........
    ........
    CYL(hh, DW, W);
    ........
    ........
end;
```

Vi ser, at der forrest i programmet er indsat de tre linier:

```
procedure CYL(H, D, V);
real H, D, V;
V := 0.785398×H×D↑2;
```

Dette er den saakaldte proceduredeklaration, som fortæller oversætter-programmet, at der i det foreliggende program forekommer en procedure ved navn: CYL. Proceduren opererer paa de tre parametre: H, D og V, som i linie 2:

```
real H, D, V;
```

er deklareret (eller mere korrekt: specificeret) til at være almindelige, reelle variable. Linie 3 er selve kernen i proceduren:

```
V := 0.785398×H×D↑2;
```

som giver den formel, efter hvilken V skal beregnes.

Proceduredeklarationen bliver oversat paa passende maade af oversætter-programmet, men i det oversatte program kommer sætningen:

```
V := 0.785398×H×D↑2;
```

ikke direkte til udførelse som saadan. Der sker først noget, naar vi i det oversatte program naar frem til sætningen:

```
CYL(H1, Diam6, V7);
```

Vi siger, at denne sætning er et <u>kald</u> af proceduren: CYL. Der sker da følgende: De tre parametre: H, D og V, som staar i proceduredeklarationen:

```
procedure CYL(H, D, V);
```

bliver i selve proceduresætningen:

```
V := 0.785398×H×D↑2;
```

udskiftet med de tre parametre H1, Diam6 og V7, som staar i procedure-kaldet:

```
CYL(H1, Diam6, V7);
```

Proceduresætningen ser da saaledes ud:

```
V7 := 0.785398×H1×Diam6↑2;
```

og denne sætning udføres nu.

Maskinen regner derefter videre paa det stykke program, som er antydet efter det første kald af CYL, indtil den naar frem til det næste kald:

```
CYL(P, d, Vol);
```

Her gentages hele spøgen, men saaledes at H, D og V nu udskiftes med P, d og Vol, og det er nu sætningen:

```
Vol := 0.785398×P×d↑2;
```

som bliver bragt til udførelse. Endelig vil det tredje kald af CYL:

```
CYL(hh, DW, W);
```

foraarsage udførelsen af sætningen:

```
W := 0.785398×hh×DW↑2;
```

De tre parametre: H, D og V, som optræder i proceduredeklarationen, kaldes <u>formelle</u> parametre, medens de tre parametre: H1, Diam6 og V7 i procedurekaldet betegnes <u>aktuelle</u> parametre. Som resume kan vi altsaa sige, at der ved et procedurekald sker følgende:

I proceduresætningen udskiftes de formelle parametre med de aktuelle parametre, og den derved fremkomne sætning udføres.

Det er vigtigt at være klar over forskellen paa formelle og aktuelle parametre. Den aktuelle parameter: H1 er et symbol, som henviser til et tal, nemlig det tal, som i det givne øjeblik staar i den celle, som er reserveret den variable: H1. Den formelle parameter: H er ogsaa et symbol, men det henviser ikke til et tal, men til et andet symbol, nemlig til den aktuelle parameter, som er benyttet i det foreliggende kald af proceduren.

Man ser altsaa, at de formelle parametre betegner en højere grad af abstraktion end de aktuelle parametre, det er skyggevariable, som først faar liv (d.v.s. udskiftes med en rigtig variabel) hver gang, der sker et kald af proceduren.

Paa den maade som vi her har deklareret proceduren: CYL, kan man sige, at navnet: CYL staar som en forkortet betegnelse for en <u>sætning</u> nemlig sætningen:

```
V := 0.785398×H×D↑2;
```

efter behørig udskiftning af de formelle parametre med de aktuelle. Man kan ogsaa deklarere en procedure paa en lidt anden maade, nemlig saaledes at dens navn staar som betegnelse for en variabel, hvis værdi fremkommer ved beregning af proceduresætningen. Dette forstaas vist lettere ved at se paa det samme eksempel som før, nu blot skrevet paa den anden maade:

```
begin
   real V7, H1, Diam6, Vol, P, d, W, hh, DW;
   real procedure cyl(H,D);
   real H, D;
   cyl := 0.785398×H×D↑2;
   ........
   ........
   V7 := cyl(H1, Diam6);
   ........
   ........
   ........
   Vol := cyl(P, d);
   ........
   ........
   ........
   W := cyl(hh, DW);
   ........
   ........
end;
```

Proceduredeklarationen ser nu lidt anderledes ud:

```
real procedure cyl(H, D);
real H, D;
cyl := 0.785398×H×D↑2;
```

Der er nu kun to formelle parametre: H og D, medens procedurens navn: cyl nu til en vis grad har erstattet den tidligere formelle parameter: V. Bemærk, at der nu staar: real procedure i stedet for blot procedure. Dette betyder, at der nu er reserveret en celle til talværdien af cyl. Naar cyl-proceduren har været kaldt, indeholder cyl-cellen denne talværdi.

Selve procedurekaldene har nu ogsaa faaet en lidt anden form:

```
V7 := cyl(H1, Diam6);
```

Men virkningen er i realiteten den samme. Naar maskinen kommer til dette kald i det oversatte program, bliver de formelle parametre: H og D i proceduresætningen:

```
cyl := 0.785398×H×D↑2;
```

udskiftet med de aktuelle parametre, saaledes at den nu lyder:

```
cyl := 0.785398×H1×Diam6↑2;
```

Denne sætning udføres nu, og cyl-cellen har da faaet tildelt sin talværdi, og denne gemmes derefter i V7-cellen, som om der stod:

```
V7 := cyl;
```

Bemærk, at den her benyttede cyl-procedure er ganske analog med standardfunktionerne, som dog kun har een parameter. Man kan f.eks. skrive:

```
V7 := sin(H1);
```

hvilket naturligvis giver en anden talværdi, men sin-proceduren er formelt deklareret som en real procedure sin(x). Procedurer af form som cyl, sin, etc., betegnes ofte procedurefunktioner.

Hvad enten man bruger CYL-proceduren eller cyl-proceduren bliver tal-

resultaterne det samme. De to metoder har fordele og ulemper, som vi kan illustrere saaledes:

Lad os antage, at vi kun har brug for at beregne differensen:

```
Dif := Vol - W;
```

men ikke for Vol og W hver for sig. Det er da en fordel at bruge cyl-proceduren, idet vi da blot skriver:

```
Dif := cyl(P, d) - cyl(hh, DW);
```

Har vi derimod brug for alle tre variable: Dif, Vol og W, kan man skrive:

```
Vol := cyl(P, d);
W   := cyl(hh, DW);
Dif := Vol - W;
```

eller med CYL-proceduren:

```
CYL(P, d, Vol);
CYL(hh, DW, W);
Dif := Vol - W;
```

Her er de to procedurer lige gode. I andre tilfælde kan CYL-formen være at foretrække, f.eks. hvis den skal indgaa som en udskiftelig procedure i andre procedurer, saaledes at procedurens navn bruges som en formel parameter.

For at undgaa misforstaaelse, bemærker vi, at en procedurefunktion godt kan staa alene som en sætning uden at indgaa i et udtryk:

```
begin
   real x, y;
   real procedure P(a);
   real a;
   begin
      y := 3;
```

```
      P := a↑2
   end P;
   x := P(17);
   ........
   P(12);
   ........
   x := P(98)
end;
```

Sætningen:

```
P(12);
```

udfører nu kun det at sætte y lig 3. Den beregner ogsaa kvadratet paa 12, men resultatet bruges ikke.

6.3. Summation.

Vi har tidligere set eksempler paa programmer, som beregnede summen af en række indlæste tal. Vi skal nu lave en procedure, som udfører selve summationen (men ikke indlæsningen) af en række elementer i et array. Hvis vi foreløbig tænker os, at vi altid opererer paa det samme array, f.eks. A[1:10], som jo kan være mængderne af komponenterne i en gasblanding, kan proceduredeklarationen se saaledes ud:

```
procedure SUM(s);
real s;
begin
   integer i;
   s := 0;
   for i := 1 step 1 until 10 do s := s + A[i]
end SUM;
```

Kaldene af proceduren kan se saaledes ud:

```
........
........
........
SUM(t);
........
........
........
SUM(q);
........
........
```

Proceduredeklarationen begynder med linien:

```
procedure SUM(s);
```

som fortæller os, at der nu følger en proceduredeklaration, at proceduren bærer navnet: SUM, samt at den indeholder een formel parameter, her betegnet: s.

I næste linie:

```
real s;
```

specificeres den formelle parameter til at være af typen real. Derefter følger den egentlige sætning i proceduren:

```
begin
   integer i;
   s := 0;
   for i := 1 step 1 until 10 do s := s + A[i]
end;
```

Dette skal opfattes som een sætning, fordi den er omgivet af begin og end; men det er ogsaa en sammensat sætning, som bestaar af de to sætninger:

```
s := 0;
```

som nulstiller sumcellen: s, og af

```
for i := 1 step 1 until 10 do s := s + A[i];
```

der er den egentlige for-sætning, som udfører summationen. Vi ser, at der er brug for en hjælpevariabel: i, til at tælle i for-sætningen. Denne integer deklareres lige efter begin.

I de to kald af proceduren:

```
SUM(t);
........
........
SUM(q);
```

bliver summen først beregnet i cellen for den variable: t og ved det næste kald i cellen for q. De to variable, t og q, maa være deklareret i hovedprogrammet.

Vi generaliserer nu deklarationen af SUM-proceduren, saaledes at vi kan summere elementerne i et vilkaarligt, eendimensionalt array fra element nr. p til element nr. q. Et program med en saadan proceduredeklaration og et par kald af proceduren kan se saaledes ud:

```
begin
   real x, y;
   array M[1:10], N[7:19];
   procedure SUM(A, p, q, s);
   integer p, q;
   real s;
   array A;
   begin
      integer i;
      s := 0;
      for i := p step 1 until q do s := s + A[i]
   end SUM;
   ........
   ........
```

```
    SUM(M, 1, 10, x);
    ........
    ........
    SUM(N, 11, 15, y);
    ........
    ........
end;
```

Proceduredeklarationen indeholder nu 4 formelle parametre: A, p, q og s. Ligesom før specificeres s som real, og vi specificerer nu p og q som integer. Bemærk, at A specificeres som et array:

```
array A;
```

men der angives ikke grænser for A, fordi A kun er en formel parameter, som ved kaldet af proceduren udskiftes med et rigtigt array, hvis grænser maa være deklareret tidligere.

I det viste tilfælde kaldes proceduren to gange, først med sætningen:

```
SUM(M, 1, 10, x);
```

som bevirker, at vi faar beregnet summen af de ti første elementer i M og resultatet gemt i cellen for x. Det andet kald er:

```
SUM(N, 11, 15, y);
```

Her summerer vi kun nogle af N-elementerne, nemlig fra N[11] til N[15] og resultatet gemmes i y-cellen.

Vi skal nu forklare en finesse ved procedurebegrebet, som erfaringsmæssigt volder begyndere stor vanskelighed. Det er det saakaldte value-begreb. Vi forklarer det i tilknytning til den sidste version af SUM-proceduren, og vi bemærker først, at en aktuel parameter (af typen real eller integer) godt kan være et kompliceret udtryk, f.eks. som i dette kald:

```
SUM(P, k+m↑2, 3×k+7×m↑2, z);
```

Her maa hovedprogrammet aabenbart indeholde et array P med kendte grænser,

en real z, og to variable: k og m, der formentlig er deklareret som integer.

Ved kaldet af proceduren SUM, sker der den sædvanlige udskiftning af de formelle parametre med de aktuelle, saaledes at selve proceduresætningen nu har formen:

```
begin
   integer i;
   z := 0;
   for i := k + m↑2 step 1 until 3×k + 7×m↑2 do
   z := z + P[i]
end;
```

Hver gang for-sætningen har adderet 1 til den variable: i, skal den undersøge, om i derved er blevet større end udtrykket:

```
3×k + 7×m↑2
```

d.v.s. den skal beregne talværdien af dette udtryk lige saa mange gange (+1), som der er elementer, der skal adderes. Dette er aabenbart en masse overflødig regning, idet udtrykket 3×k + 7×m↑2 her er ment som et konstant tal. Vi ønsker derfor at udvide procedurebegrebet saaledes, at nogle af de formelle parametre kan mærkes paa en saadan maade, at deres talværdi udregnes een gang for alle, nemlig inden de egentlige sætninger i proceduren udføres.

I det foreliggende tilfælde vil vi gerne have de to formelle parametre, p og q mærket paa denne maade. Man siger, at de to variable kaldes ved value, (called by value) og det skrives saaledes:

```
procedure SUM(A, p, q, s);
value p, q;
integer p, q;
real s;
array A;
begin
   integer i;
   s := 0;
```

```
    for i := p step 1 until q do s := s + A[i]
end SUM;
```

Vi ser, at der nu er kommet en ekstra linie med:

```
value p, q;
```

Den anbringes *før* specifikationerne og udsiger, hvilke formelle parametre, som kaldes ved value.

Proceduren kunne ogsaa have været deklareret saaledes, uden brug af *value*:

```
procedure SUM(A, p, q, s);
integer p, q;
real s;
array A;
begin
   integer i, pp, qq;
   pp := p;
   qq := q;
   s := 0;
   for i := pp step 1 until qq do s := s + A[i]
end SUM;
```

De to sidste udgaver af SUM-proceduren er fuldstændigt ækvivalente og illustrerer, hvad der sker, naar en formel parameter er kaldt ved *value*. Det, at p og q er kaldt ved *value*, betyder, at der oprettes to ekstra celler (her illustreret med pp og qq), som er lokale for denne procedure (d.v.s. ikke tilgængelige udefra), samt at proceduren allerførst sætter pp lig med p og qq lig med q. Endelig er p og q overalt i den egentlige proceduresætning erstattet med pp og qq. Paa denne maade skal de komplicerede udtryk for p og q som fandtes i de aktuelle parametre kun beregnes een gang.

Naar formelle parametre kaldes ved *value*, sørger programmet automatisk for at oprette de ekstra, lokale celler for de paagældende variable, samt at deres værdi udregnes inden den egentlige procedureberegning begynder.

Reglen for brug af *value* begrebet er da følgende: Brug *altid* *value*

undtagen for saadanne variable, som proceduren selv beregner, og som bagefter skal bruges af hovedprogrammet. I det foreliggende tilfælde skal den variable: s ikke kaldes ved value, da dens værdi er et resultat af procedurens regning. Gør man det alligevel, vil den aktuelle parameter i hovedprogrammet (altsaa her x, y eller z) slet ikke blive ændret.

I GIER ALGOL kan arrays ikke kaldes ved value. Dette ville ogsaa normalt være meget upraktisk, idet proceduren da skulle oprette et helt ekstra array af samme størrelse som det aktuelle array og dette vil oftest være spild af lagerplads.

En parameter, som ikke er kaldt ved value, siges at være kaldt ved name. Dette skrives dog ikke i procedurededeklarationen.

Ved hjælp af de ovennævnte SUM-procedurer kan vi summere elementer i et eendimensionalt talsæt. Men hvis talsættet har mere end een dimension, gaar det ikke. Dette skyldes, at proceduren indeholder formen:

```
A[i]
```

for det element, der skal summeres, og derved er vi bundet til kun at bruge eendimensionale talsæt. Hvis vi ønsker en procedure, som baade kan summere eendimensionale talsæt som hidtil og ogsaa todimensionale talsæt, kan vi deklarere proceduren saaledes:

```
real procedure SUM(E, i, p, q);
value p, q;
integer i, p, q;
real E;
begin
   real S;
   S := 0;
   for i := p step 1 until q do  S := S + E;
   SUM := S
end SUM;
```

De aktuelle kald af denne procedure kan f.eks. foregaa i et program af dette udseende:

```
begin
   integer j, k;
   real x, y, z, v;
   array M[1:10], N[7:19], Q[1:8, 1:9];
   ........
   ........
   ........
   ........
   x := SUM(M[j], j, 1, 10);
   ........
   ........
   y := SUM(N[j], j, 11, 15);
   ........
   ........
   z := SUM(Q[j,3], j, 1, 8);
   ........
   ........
   v := SUM(SUM(Q[j,k], j, 1, 8), k, 1, 9);
   ........
   ........
end;
```

Proceduredeklarationen er nu lavet som en procedurefunktion, analogt med den tidligere cyl-funktion. Derfor udgaar den formelle parameter, s, som bliver overflødig, idet vi nu skal skrive:

```
x := SUM (......);
```

i stedet for:

```
SUM (......, x);
```

Den tidligere formelle parameter af type array: A, bruges ikke mere. I stedet har vi de to nye formelle parametre:

```
real E;
integer i;
```

Den egentlige summationssætning i proceduren er:

```
S := S + E;
```

hvor S er en lokal sumcelle. Det, vi skal addere til S kaldes altsaa blot E, og det er da meningen, at den aktuelle parameter svarende til E skal skrives som et element i et array, f.eks. som i procedurekaldet:

```
x := SUM(M[j], j, 1, 10);
```

Naar dette procedurekald skal udføres, udskiftes de formelle parametre med de aktuelle, og proceduresætningen faar formen:

```
begin
   real S;
   S := 0;
   for j := 1 step 1 until 10 do S := S + M[j];
   SUM := S
end;
```

Vi ser, at det er nødvendigt med den formelle parameter: i (som her udskiftes med den aktuelle parameter: j), fordi proceduren skal bruge dette i (eller rettere j) til at lade M[j] variere. Denne snedige fidus, at proceduren indeholder en tilsyneladende simpel variabel (E) uden indices, samt en integer variabel (i), som bruges til at producere elementer af en indiceret variabel i stedet for E, kaldes ofte for Jensens device (efter Jørn Jensen, Regnecentralen).

Man vil da forstaa, at procedurekaldet:

```
x := SUM(M[j], j, 1, 10);
```

har ganske samme effekt som kaldet af den anden SUM-procedure paa side 92:

```
SUM(M, 1, 10, x);
```

Bemærk dog, at vi i den seneste udgave af SUM maa have deklareret en variabel: j i hovedprogrammet. Ligeledes er det nye kald:

```
y := SUM(N[j], j, 11, 15);
```

ækvivalent med det gamle:

```
SUM(N, 11, 15, y);
```

Ser vi paa det tredje kald af den seneste SUM-procedure:

```
z := SUM(Q[j,3], j, 1, 8);
```

vil proceduresætningen efter udskiftning af de formelle parametre blive til:

```
begin
   real S;
   S := 0;
   for j := 1 step 1 until 8 do S := S + Q[j,3];
   SUM := S
end;
```

Virkningen heraf er, at vi faar summeret den tredje søjle:

```
Q[1,3], Q[2,3], ......, Q[8,3]
```

i det todimensionale Q-array.

Det sidste eksempel er:

```
v := SUM(SUM(Q[j,k], j, 1, 8), k, 1, 9);
```

Virkningen heraf er, at v bliver beregnet som summen af alle 8×9 elementer i Q-arrayet. Hvis vi paa sædvanlig maade erstatter de formelle parametre med de aktuelle, faar vi fra den yderste SUM-procedure-parentes:

```
begin
   real S;
   S := 0;
   for k := 1 step 1 until 9 do
```

```
        S := S + SUM(Q[j,k], j, 1, 8);
        SUM := S
    end;
```

Den inderste SUM-procedure-parentes:

```
SUM(Q[j,k], j, 1, 8);
```

er ganske analog med beregningen af z ovenfor og har som værdi summen af søjle nr. k i talsættet. Den endelige værdi af v bliver derfor summen af alle søjler, d.v.s. summen af samtlige elementer. Vi siger, at proceduren er brugt rekursivt, som f.eks. i:

```
v := sin(sin(x));
```

og det er her karakteristisk, at programmet midt i beregningen af en bestemt procedure midlertidigt maa afbryde denne beregning for at udføre et nyt kald af den samme procedure. Programmet er naturligvis forsynet med den nødvendige administration til at gennemføre dette uden at tabe traaden i beregningen. Foruden denne rekursive anvendelse af procedurer, har man ogsaa egentlige rekursive procedurer, som omtales i afsnit 6.7.

Det er iøvrigt vigtigt at bemærke, at vi maa have en speciel, lokal sumcelle: S til summationen i sætningen:

```
S := S + E
```

Her kan ikke staa:

```
SUM := SUM + E
```

da dette ville bevirke, at proceduren kaldte sig selv i det uendelige. Iøvrigt ville oversætterprogrammet protestere mod at der mangler en parentes med parametre efter det sidste SUM.

6.4. Beregning af Entalpi.

I dette og de to følgende afsnit giver vi nogle kemisk betonede eksempler paa anvendelse af procedurer.

Vi har set, at den vigtigste fordel ved brugen af procedurer er den, at hvis der i et større program er afsnit, som er ens eller næsten ens, behøver dette programafsnit kun at staa eet sted i programmet som en procedure-deklaration. De enkelte afsnit bringes da til udførelse ved et kald af proceduren, d.v.s. at man skriver dens navn og en liste over de aktuelle parametre. Ved at variere de aktuelle parametre kan man faa proceduren til at arbejde med forskellige variable og derved faa korrigeret for de mindre forskelle i de oprindelige afsnit.

Denne filosofi kan udvides, saaledes at en proceduredeklaration, som een gang for alle er programmeret og afprøvet, kan indføres uændret i forskellige programmer, og i hvert af disse bringes til udførelse ved eet eller flere kald. Paa denne maade opnaar man en besparelse i programmerings- og afprøvningstid, fordi de vigtigste dele af programmet (beregning af entalpi, kemisk ligevægt, o.s.v.) uden videre kan overtages fra et ældre program. Man kan paa denne maade ogsaa udnytte procedurer, som har været offentliggjort i tidsskrifter, og som ofte har gennemgaaet en meget kritisk afprøvning. I saadanne tilfælde er det fundamentale princip med udskiftning af de formelle parametre med aktuelle særligt værdifuldt, idet man jo normalt ikke kan regne med, at andre programmører har brugt de samme betegnelser paa de variable, som man selv ønsker at bruge.

Vi giver nu et eksempel paa en saadan standardprocedure til beregning af entalpien af ideale gasblandinger. Paa side 38 blev vist et lille program, som beregnede entalpien, H kcal/kgmol, af en gasblanding med 8 komponenter. Molbrøkerne heraf er givet som talsættet m[1:8], der indlæses som talmateriale fra en inputstrimmel. Beregningen udføres for den absolutte temperatur, T gr.K, der ogsaa indlæses. De 8×5 koefficienter i entalpipolynomierne betegnes som talsættet a[1:8, 0:4], der ogsaa indlæses. Vi laver nu en procedure, som indeholder de fire formelle parametre: H, a, m og t. Her er t temperaturen i gr.C, som er mere praktisk at anvende, medens H, a og m er forklaret ovenfor. Proceduredeklarationen herfor kan se saaledes ud:

```
procedure ENT1(H, a, m, t);
real H, t;
array a, m;
begin
   integer i, j;
   real h, T;
   T := 273.16 + t;
   H := 0;
   for i := 1 step 1 until COMP do
   begin
      if m[i] = 0 then go to L1;
      h := a[i, 4];
      for j := 3 step -1 until 0 do
      h := h×T + a[i,j];
      H := H + h×m[i];
L1:   end
end ENT1;
```

Her er et eksempel paa et program, hvori denne procedure kaldes:

```
begin
   integer COMP, COMPMAX;
   real HRI, HOS, HRO, tginlet, OStemp, toutlet;
   comment  Her anbringes proceduredeklarationen for ENT1;
   ........
   ........
   ........
   comment  Her fastlægges værdien af COMPMAX;
   ........
   ........
   begin
      array INLET, OUTLET[1:COMPMAX], OSFEED[1:9],
            ELIST[1:COMPMAX, 0:4];
      ........
      ........
      ........
```

```
comment Her fremskaffes talværdierne for elementerne i de fire
        arrays: INLET, OUTLET, OSFEED og ELIST;
........
COMP := COMPMAX;
ENT1(HRI, ELIST, INLET, tginlet);
ENT1(HRO, ELIST, OUTLET, toutlet);
COMP := 9;
ENT1(HOS, ELIST, OSFEED, OStemp);
........
........
end
end;
```

Vi ser først paa procedure-deklarationen. Af de fire formelle parametre specificeres H og t som real og a og m som array. Man kunne godt have kaldt t ved value:

```
value t;
```

men da t kun forekommer een gang i proceduren:

```
T := 273.16 + t;
```

og der derefter regnes videre med den lokale variable: T, er der ingen virkelig besparelse ved at kalde t ved value.

Selve entalpiberegningen er iøvrigt mage til den tidligere udgave side 39 med to for-sætninger indeni hinanden, hvor den inderste tæller koefficienterne fra 3 ned til 0 og den yderste tæller komponentantallet. Der er dog den forskel, at vi i det første eksempel altid regnede med 8 komponenter:

```
for n := 1 step 1 until 8 do
```

medens vi i ENT1-proceduren skriver:

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

COMP er aabenbart en integer variabel, men den findes ikke blandt de formelle parametre i proceduren og er heller ikke deklareret sammen med de lokale variable: i, j, h og T. Dette er et eksempel paa brugen af en saakaldt global (ikke-lokal) variabel. Den er deklareret i hovedprogrammet sammen med en anden variabel: COMPMAX:

```
integer COMP, COMPMAX;
```

Forekomsten af den slags globale variable i en procedure kan være risikabelt, fordi programmøren kan glemme at deklarere den i hovedprogrammet, og han er iøvrigt tvunget til at bruge netop dette navn: COMP, det kan ikke udskiftes, som hvis det havde været en formel parameter. Grunden til at vi har gjort COMP til en global variabel er den, at man i store programmer normalt vil have mange forskellige procedurer af termodynamisk og lignende karakter, som alle indeholder for-sætninger, som skal tælle paa antallet af tilstedeværende komponenter. Det er derfor praktisk kun at bruge een eneste celle til at opbevare komponentantallet: COMP. Alternativet havde været at deklarere entalpiproceduren saaledes:

```
procedure ENT1(COMP, H, a, m, t);
value COMP;
integer COMP;
real H, t;
array a, m;
........
o.s.v.
```

men saa vil hvert procedurekald komme til at indeholde en parameter mere.

ENT1-proceduren indeholder en anden forskel fra den tidligere version, nemlig sætningen:

```
if m[i] = 0 then go to L1;
```

Denne er indført, fordi nogle af molbrøkerne sommetider kan være nul, især hvis man arbejder med en stor, fast komponentliste. Det kan da betale sig at overspringe polynomieberegningen af h for saadanne komponenter. Iøvrigt havde det været lidt fiksere at undgaa brugen af L1-etiketten saaledes:

```
for i := 1 step 1 until COMP do
if m[i] ≠ 0 then
begin
   h := a[i,4];
   for j := 3 step -1 until 0 do
   h := h×T + a[i,j];
   H := H + h×m[i]
end;
```

I det viste programeksempel, hvori proceduren ENT1 kaldes tre gange, optræder de to integer variable: COMP og COMPMAX. Brugen af COMP er nødvendig fordi denne variable staar i ENT1-proceduren. Den variable: COMPMAX er ment til at betegne det maximale antal komponenter, som kan forekomme i nogen gasblanding i den foreliggende beregning. Eksemplet er hentet fra beregning af en reformer, hvor det er meningen, at programmet selv skal finde COMPMAX ud fra oplysninger om hvilke komponenter, der findes i den kulbrinteblanding, som skal reformes. Programmet beregner, hvilke lavere kulbrinter, som kan opstaa ved nedbrydning heraf. Denne beregning har vi symboliseret ved:

```
comment Her fastlægges værdien af COMPMAX;
```

Naar værdien af COMPMAX er fastslaaet, kan programmet afsætte plads til de arrays, som skal indeholde mængderne af de forskellige komponenter i de gasstrømme, som findes i apparatet. Det sker ved deklarationen:

```
array INLET, OUTLET[1:COMPMAX], OSFEED[1:9],
      ELIST[1:COMPMAX, 0:4];
```

Her er INLET og OUTLET listerne over komponentmængder i indgangsgassen og afgangsgassen fra reformeren. Desuden optræder gasblandingen: OSFEED, som er en iltholdig gasblanding, som skal tilføres apparatet, og som vides ikke at indeholde højere kulbrinter. Dens elementantal er derfor begrænset til 9, medens COMPMAX er større end 9. Programmet er lavet saaledes, at de første 9 komponenter i INLET og OUTLET altid er de samme og identiske med komponenterne i OSFEED. Hvad der findes herudover i INLET og OUTLET varierer fra beregning til beregning, og findes hver gang af programmet.

Talsættet ELIST er koefficienterne i entalpipolynomierne. Ogsaa her er de første 9 komponenter altid de samme, medens de efterfølgende bestemmes fra gang til gang.

Naar de fire talsæt er deklareret, vil programmet paa passende maade finde deres værdier. Dette er ikke medtaget her, men man kan tænke sig at INLET og OSFEED er inputmateriale. OUTLET maa fremkomme ved reforming af INLET, evt. med indstilling af metan- og vandgasligevægt, altsaa noget programmet kan beregne. Programmet maa ogsaa fremskaffe værdierne af koefficienterne i ELIST, idet det er upraktisk at skulle indlæse dette som egentligt inputmateriale.

Det første kald af ENT1 er:

```
ENT1(HRI, ELIST, INLET, tginlet);
```

Herved beregnes entalpien, HRI, af indgangsgassen repræsenteret ved talsættet: INLET og svarende til indgangstemperaturen: tginlet. Det fremgaar ikke af programmet, om INLET er molbrøkerne af de enkelte komponenter eller de absolutte mængder i kgmol/h, som strømmer igennem apparatet. Hvis det sidste er tilfældet (hvad det faktisk er), bliver entalpien aabenbart beregnet med enheden kcal/h.

Umiddelbart før dette kald har vi med sætningen:

```
COMP := COMPMAX;
```

sørget for at tildele COMP den rigtige værdi. Det næste kald er ganske analogt:

```
ENT1(HRO, ELIST, OUTLET, toutlet);
```

nu er det blot entalpien, HRO, af afgangsgassen, givet ved sammensætningen: OUTLET og med temperaturen toutlet. I det sidste procedurekald:

```
ENT1(HOS, ELIST, OSFEED, OStemp);
```

skal vi beregne iltblandingens entalpi, HOS, men da talsættet OSFEED kun indeholder 9 elementer, maa vi inden dette kald sætte COMP lig med 9:

```
COMP := 9;
```

Hvis ENT1 senere i programmet skal bruges igen paa det fulde antal komponenter, bør man sætte COMP lig med COMPMAX igen efter det tredje kald for ikke at glemme det senere.

Bemærk, at vi har brugt den samme parameter: ELIST for entalpikoefficienterne i alle tre kald i dette program. Man kunne da helt have udeladt denne parameter og gjort den til en global variabel i proceduren. Vi har dog andre programmer, hvor det har vist sig praktisk at bibeholde denne parameter, f.eks. hvis der optræder gasstrømme med helt forskellige sammensætninger. Som hovedregel kan man sige, at der ikke maa findes overflødige parametre i standardprocedurer, men det vil i praksis altid blive et kompromis, fordi en standardprocedure bør være tilpas generelt udformet, saaledes at den virkelig kan bruges i mange forskellige programmer. Herved kan det være nødvendigt at tage nogle flere parametre med end strengt nødvendigt for brugen i et isoleret program.

6.5. Beregning af Vandgasligevægt.

Programmet side 64 til beregning af vandgasligevægten kan ret let omdannes til en procedure. I programmet indlæste vi temperaturen, t, og molprocenterne:

```
MCO, MH2O, MCO2, MH2
```

af de fire deltagende komponenter. Det vil være naturligt, at lade de samme fem variable være formelle parametre i proceduren. Den kan ogsaa aflevere de beregnede ligevægtsprocenter i de samme ovennævnte fire variable molprocenter.

Proceduren kan da se saaledes ud:

```
procedure WGEQ(t, MCO, MH2O, MCO2, MH2);
real t, MCO, MH2O, MCO2, MH2;
begin
   real T, K, delta;
   T := t + 273.16;
   K := exp(-0.768535428×ln(T)  + ((((1.47520303₁₀-10×T-9.66051861₁₀-7)×T
     + 3.01017929₁₀-3)×T  -  1.50650387)×T  +  4.94327234₁₀3)/T);
ST: delta := (K×MCO×MH2O - MCO2×MH2)/(K×(MCO + MH2O) + MCO2 + MH2);
   MCO := MCO - delta;
   MH2O := MH2O - delta;
   MCO2 := MCO2 + delta;
   MH2 := MH2 + delta;
   if abs(delta) > 0.01 then go to ST
end WGEQ;
```

Proceduren indeholder de samme bestanddele som tidligere: beregning af K og den iterative indstilling af ligevægten. Da vi her forudsætter, at molmængderne: MCO, MH2O, o.s.v. er molprocenter, har vi ændret tolerancen paa delta fra 0.0001 til 0.01. Hvis man ønsker, at proceduren skal kunne bruges for molprocenter, molbrøker og absolutte molmængder, vil det være klogt at indføre tolerancen som en formel parameter.

Ingen af parametrene er her kaldt ved value. For temperaturen, t, gælder det samme som for entalpiproceduren. De fire molprocenter kan ikke kaldes ved value, fordi vi skal have ligevægtsværdierne ud af proceduren. Da det koster nogen ekstra tid at hente en aktuel parameter, som er en simpel variabel og ikke kaldt ved value (kaldt ved name), kan der spares nogen regnetid, hvis man ændrer proceduren, saaledes at hovedparten af beregningerne sker med lokale variable, f.eks. saaledes:

```
procedure WGEQ(t, MCO, MH2O, MCO2, MH2);
real t, MCO, MH2O, MCO2, MH2;
begin
   real T, K, delta, M1, M2, M3, M4;
   T := t + 273.16;
```

```
        K := exp(-0.768535428×ln(T)  + ((((1.47520303_10-10×T-9.66051861_10-7)×T
          + 3.01017929_10-3)×T - 1.50650387)×T + 4.94327234_10 3)/T);
        M1  := MCO;
        M2  := MH2O;
        M3  := MCO2;
        M4  := MH2;
ST:     delta := (K×M1×M2 - M3×M4)/(K×(M1 + M2) + M3 + M4);
        M1  := M1 - delta;
        M2  := M2 - delta;
        M3  := M3 + delta;
        M4  := M4 + delta;
        if abs(delta) > 0.01 then go to ST;
        MCO  := M1;
        MH2O  := M2;
        MCO2  := M3;
        MH2   := M4
    end WGEQ;
```

Hvis proceduren skal bruges i et program mage til det tidligere viste, kan dette skrives saaledes:

```
    begin comment  Beregning af vandgasligevægt;
        integer i;
        real t;
        array MInput, MEQ[1:4];
        copy WGEQ <
        select(8);
        t := read real;
        for i := 1 step 1 until 4 do
        MInput[i] := read real;
        for i := 1 step 1 until 4 do MEQ[i] := MInput[i];
```

```
WGEQ(t, MEQ[1], MEQ[2], MEQ[3], MEQ[4]);
writecr;
writetext(‹<Temperatur, gr.C›);
write(‹ddddd›, t);
writecr;
writetext(‹<            Molprocenter:›);
writecr;
writetext(‹<         Original   Ligevægt›);
writecr;
for i := 1 step 1 until 4 do
begin
   writecr;
   writetext(if i = 1 then ‹<Kulilte›
   else if i = 2 then ‹<Vand    ›
   else if i = 3 then ‹<Kulsyre›
   else ‹<Brint  ›);
   write(‹dddddd.dd›, MInput[i], MEQ[i])
end for i;
writecr
end af programmet;
```

Der er her indført et par smaaændringer. Molprocenterne gemmes som to arrays: MInput, MEQ[1:4]. Vi ser bort fra inerte. Resultattrykningen er nu skrevet som en for-sætning med tælling i den variable: i.

Sætningen:

```
copy WGEQ <
```

betyder, at der paa dette sted egentlig skulle anbringes proceduredeklarationen af WGEQ. Man kan have en aftale med hulledamen om, at hun paa dette sted skal hente en strimmel med WGEQ-proceduren paa og lade Flexowriteren reperforere denne strimmel ind paa selve programstrimlen. Men det kan ogsaa gøres paa en lidt fiksere maade, idet hulledamen simpelthen kopierer, hvad der staar:

```
copy WGEQ <
```

Hvis programmet oversættes paa GIER med GIER ALGOL 4 oversætteren,

saaledes som denne fungerer i forbindelse med det saakaldte HELP 3 system, som er et system af forskellige hjælpeprogrammer, sker der følgende: Indlæsningen af programstrimlen standses automatisk, naar ordet copy mødes. Programsystemet begynder derefter at afsøge et internt katalog over forskellige navne. Hvis det her møder navnet WGEQ, og kataloget indeholder den fornødne bekræftelse paa, at navnet WGEQ er et navngivet omraade paa maskinens pladelager(disk), vil oversætteren automatisk skifte over til at hente programteksten fra dette omraade. Her skal der altsaa staa proceduredeklarationen for WGEQ, hvis det skal fungere efter hensigten. Kopieringen fra disk-omraadet afsluttes af ordet finis. Hvis navnet WGEQ ikke findes i kataloget, faar man fejludskriften:

copy

og oversættelsen kan ikke gennemføres. Det samme sker, hvis navnet WGEQ findes i kataloget, men betegner noget andet end programtekst. De nærmere regler for brug af copy og HELP 3 systemet er beskrevet af Naur (1967) og Lauesen (1967).

Hos Haldor Topsøe kan man ved at bede om en speciel liste over GIER ALGOL 4 procedurer se, hvilke procedurer, der paa denne maade er tilgængelige via copy. Dette er ikke tilfældet med proceduren WGEQ, der kun er beregnet som en illustration.

6.6. Beregning af Trykfald i Rør.

Som et yderligere eksempel paa den praktiske brug af procedurer giver vi nu en omskrivning af trykfaldsprogrammet paa side 76, hvor vi har indført forskellige procedurer:

```
begin comment beregning af trykfald;
   integer COMP, i, j, k;
   real Dmm, L, F, t, P, Pcmix, Tcmix, mycmix, rho0, my, delP;
```

```
select(8);
Dmm := read real;
L := read real;
F := read real;
t := read real;
P := read real;
COMP := read integer;
begin
   integer array CN[1:COMP];
   array m, M, Pc, Tc, myc, rhon[1:COMP];
   procedure TRYKF(D, L, F, t, P, rho0, my, delP);
   value D, L, F, t, P, rho0, my;
   real D, L, F, t, P, rho0, my, delP;
   begin
      real A, G, rho, V, NRe, f;
      A := 0.785398×D↑2;
      G := F×22.415×rho0/A;
      rho := 273.16/(t+273.16)×P×rho0;
      V := G/rho;
      NRe := D×G/my;
      f := 0.184/NRe↑0.2;
      delP := rho×f×L×V↑2/D/(2×1.2714⏨8×9678)
   end TRYKF;
   procedure VISC(Tcmix, Pcmix, mycmix, t, P, my);
   value Tcmix, Pcmix, mycmix, t, P;
   real Tcmix, Pcmix, mycmix, t, P, my;
   begin
      real T, Tr, Pr, myr;
      T := t + 273.16;
      Tr := T/Tcmix;
      Pr := P/Pcmix;
      myr := 0.64×Tr↑0.60 + (if Pr > 1 then 1.43×Tr↑(-3.98) +
      0.275×Tr↑(-1.54)×(Pr-1)
      else 1.43×Tr↑(-3.98)×Pr);
```

```
               my := myr×mycmix
            end VISC;
            procedure D(a1, a2, a3, a4);
            value a1, a2, a3, a4;
            real a1, a2, a3, a4;
            begin
               for k := 1 step 1 until COMP do
               if j = CN[k] then
               begin
                  rhon[k] := a1;
                  Tc[k] := a2;
                  Pc[k] := a3;
                  myc[k] := a4;
                  i := i + 1;
                  if i > COMP then go to L1;
                  go to h
               end if j og for k;
h:             j := j + 1
            end D;
            comment Nu er proceduredeklarationerne slut og det egentlige
                    program begynder;
            for i := 1 step 1 until COMP do
            begin
               CN[i] := read integer;
               M[i] := read real;
               m[i] := 0.01×M[i]
            end for i;
            i := j := 1;
            D(0.08994,  41.26,  20.80, 0.0125);
            D(0.80375, 647.31, 218.40, 0.1786);
            D(1.24987, 126.06,  33.50, 0.0655);
            D(1.33875, 179.16,  65.00, 0.0923);
            D(1.24965, 134.16,  35.00, 0.0685);
            D(1.96346, 304.26,  73.00, 0.1235);
            D(1.78202, 151.16,  48.00, 0.0952);
            D(0.71573, 190.66,  45.80, 0.0580);
L1:         Pcmix := Tcmix := mycmix := rho0 := 0;
```

```
        for i := 1 step 1 until COMP do
        begin
           Pcmix := Pcmix + Pc[i]×m[i];
           Tcmix := Tcmix + Tc[i]×m[i];
           mycmix := mycmix + myc[i]×m[i];
           rho0 := rho0 + rhon[i]×m[i]
        end for i;
        VISC(Tcmix, Pcmix, mycmix, t, P, my);
        TRYKF(0.001×Dmm, L, F, t, P, rho0, my, delP);
        writetext({<
    D       L       F        t       P      delP
    mm      m    kgmol/h   gr.C    atm.    atm.

}>);
        write({ddddd.dd}, Dmm, L, F, t, P, delP);
        writecr;
        writetext({<
Komponent      Molprocent

}>);
        for i := 1 step 1 until COMP do
        begin
           write({ddddd}, CN[i]);
           writetext({<            }>);
           write({ddd.dd}, M[i]);
           writecr
        end for i
     end indre blok
  end program;
```

Programmet begynder som tidligere med deklaration af en række <u>integer</u> og <u>real</u> variable. Der er ikke saa mange som tidligere, fordi nogle af de variable nu har fundet plads som lokale variable i procedurerne.

Derefter indlæses de seks variable: Dmm, L, F, t, P og COMP. Saa følger en række nye deklarationer, nemlig af de talsæt, som indeholder COMP elementer, og som først kan deklareres nu, hvor vi kender værdien af COMP.

Bemærk, at vi har flyttet det tidligere array m[1:8] ind paa denne plads som m[1:COMP], fordi vi nu paa grund af de nye procedurer ikke behøver at operere med det komplette array m[1:8]. Til gengæld har vi indført fire nye arrays:

Pc, Tc, myc, rhon[1:COMP]

hvori element nr. i er det kritiske tryk, den kritiske temperatur, den kritiske viskositet og normalvægtfylden af komponent nr. i.

Derefter deklareres tre procedurer: TRYKF, VISC og D.

Proceduren TRYKF arbejder med de otte formelle parametre: D, L, F, t, P, rho0, my og delP, som har den samme betydning som de tilsvarende variable i første version af programmet. Proceduren arbejder med de lokale hjælpevariable: A, G, rho, V, NRe og f, som ogsaa er magen til tidligere. Formlerne er ogsaa de samme. Vi ser at trykfaldet, delP, beregnes under forudsætning af at normalvægtfylden, rho0, og viskositeten, my, allerede er udregnet.

Viskositeten beregnes af proceduren VISC, som har de formelle parametre: Tcmix, Pcmix, mycmix, t, P og my. Disse har ogsaa samme betydning som tidligere. Proceduren har de fire lokale variable: T, Tr, Pr og myr, og den bestaar iøvrigt af de samme formler som tidligere, nemlig beregning af Tr og Pr, samt udregning af den lange formel for myr. Endelig findes my som myr×mycmix.

Den sidste procedure, D, kræver en nærmere forklaring. For hver af de otte komponenter, som er fast indbygget i programmet, skal dette indeholde de fire data: Pc, Tc, myc og rhon, altsaa ialt 32 tal. For komponenten med komponentnummer 1 (brint) er de fire data:

rhon	Tc	Pc	myc
0.08994	41.26	20.80	0.0125

Hvis nu f.eks. den tredje komponent netop er brint, skal programmet udføre sætningerne:

```
rhon[3] := 0.08994;
Tc  [3] := 41.26;
Pc  [3] := 20.80;
myc [3] := 0.0125;
```

Dette gør vi ved hjælp af proceduren D, som har fire formelle parametre:

```
procedure D(a1, a2, a3, a4);
```

og som af hovedprogrammet kaldes otte gange:

```
D(0.08994, 41.26, 20.80, 0.0125);
D(0.80375, 647.31, 218.40, 0.1786);
```

o.s.v. op til:

```
D(0.71573, 190.66, 45.80, 0.0580);
```

Det første D-kald indeholder de fire tal for komponent nr. 1, det næste tallene for komponent nr. 2, o.s.v. Proceduren arbejder med to globale integer variable: i og j. Værdien af j sættes til 1 før det første D-kald.

I hvert D-kald sker følgende. Der udføres en for-sætning:

```
for k := 1 step 1 until COMP do
```

For hver værdi af k undersøges det, om CN[k] er lig med den aktuelle værdi af j. Saa snart programmet er kommet frem til en saadan værdi af k, at CN[k] er lig med j, betyder det, at de fire aktuelle værdier af parametrene: a1, a2, a3 og a4 aabenbart er de søgte tal for komponent nr. k. Proceduren indsætter derefter disse værdier ved hjælp af sætningerne:

```
rhon[k] := a1;
Tc  [k] := a2;
Pc  [k] := a3;
myc [k] := a4;
```

For-sætningen afbrydes derefter, og proceduren slutter med at tælle j een frem:

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

Der er iøvrigt en ekstra tælling i den variable: i, som bevirker, at saa snart programmet har faaet indsat værdierne for alle COMP komponenter, overspringes de sidste D-kald.

Vi er nu færdig med de tre proceduredeklarationer og begynder paa det egentlige program. Ligesom sidst indlæses de COMP talpar:

CN[i] og M[i]

m[i] beregnes som 0.01×M[i]. Derefter sættes de to tælleværker: i og j til 1, og vi faar de otte kald af D-proceduren. Naar dette er forbi, skal vi beregne de fire blandingsegenskaber: Pcmix, Tcmix, mycmix og rho0. Det sker ved først at nulstille de fire variable og derefter at lade en for-sætning, hvor i løber fra 1 til COMP, addere Pc[i]×m[i] til Pcmix og analogt for de andre variable.

Derefter kaldes proceduren: VISC, som beregner viskositeten, my, og proceduren: TRYKF, der beregner det søgte trykfald: delP. Bemærk, at vi ikke har indført nogen speciel variabel for diameteren med enhed meter. Den indlæste diameter i mm: Dmm, indsættes som 0.001×Dmm i procedurekaldet.

Trykprogrammet er uændret fra den tidligere version.

Det fremgaar af de to procedurekald af VISC og TRYKF, at man gerne maa bruge de samme navne paa de aktuelle parametre, som for de formelle parametre, naar blot de tilsvarende navne er deklareret i hovedprogrammet. De formelle parametre bruges jo kun i proceduredeklarationen, hvorfor der ikke kan opstaa misforstaaelser.

Man ser, at programmet fylder en hel del mere skrevet med procedurer end uden, men dette er forstaaeligt, da de to vigtigste procedurer jo kun kaldes een gang hver.

Den komplicerede anvendelse af D-proceduren er nødvendig, fordi man i ALGOL desværre ikke har nogen simpel metode til at generere talsæt med faste elementer. Til daglig brug har vi hos Haldor Topsøe lagret de vigtigste termodynamiske konstanter i et fast, navngivet omraade paa disken, hvorfra de kan hentes frem med særlige procedurer. Disse omtales dog ikke nærmere her.

6.7. Rekursive Procedurer.

I afsnit 6.3 saa vi et eksempel paa en SUM-procedure, som kunne bringes til at kalde sig selv, hvis man anvendte den saaledes:

```
v := SUM(SUM(Q[j,k], j, 1, 8), k, 1, 9);
```

Vi skal nu give et eksempel paa en endnu mere raffineret form for rekursive procedurer, nemlig hvor det allerede i selve proceduredeklarationen udtrykkeligt er skrevet, at proceduren skal kalde sig selv.

Vi vil lave en procedurefunktion, der beregner fakultetsfunktionen:

$$FAC(n) = 1\times 2\times 3\times \ldots\ldots \times n$$

En normal, ikke-rekursiv deklaration herpaa kan være:

```
integer procedure FAC(n);
value n;
integer n;
begin
   integer i, p;
   p := 1;
   for i := 2 step 1 until n do p := i×p;
   FAC := p
end FAC;
```

Der er de to lokale variable: i og p. Først sættes p lig 1, og en for-sætning sørger derefter for at gange med 2, 3, ... n.

En tilsvarende rekursiv procedure kan skrives saaledes:

```
integer procedure FAC(n);
value n;
integer n;
FAC := if n = 1 then 1 else n×FAC(n-1);
```

Her beregnes FAC som n×FAC(n-1) og FAC(n-1) skal da igen beregnes som

(n-1)×FAC(n-2) o.s.v., indtil vi naar ned til 1. Først da bliver den sidste faktor sat til 1, og den kalder ikke sig selv mere. Vi ser, at den rekursive udgave af FAC fylder meget mindre end den ikke-rekursive udgave. Dette kan være en fordel i visse tilfælde, hvor man ønsker at gøre sit program saa kort som muligt. Derimod vil den rekursive beregning tage meget længere tid end den ikke-rekursive, fordi proceduren jo skal kaldes n gange, og det koster ca. 5 millisek. for hver gang. Regnetiden for FAC (10) er 14 millisek. i den ikke-rekursive udgave og 96 millisek. i den rekursive. Brugen af rekursive procedurer kan derfor normalt ikke anbefales, hvis man kan klare sig paa anden maade. Det er iøvrigt kun faa oversættere (heriblandt GIER-oversætteren), som tillader brugen af rekursive procedurer.

Ved tekniske beregninger har man normalt ingen glæde af rekursive procedurer. Dette kan man derimod have ved en mere avanceret programmering. Som eksempel kan vi tage selve oversætterprogrammet, der jo i det væsentlige udfører symbolmanipulation, d.v.s. den transformerer en streng af symboler til en anden streng af symboler. For at kunne gøre dette, maa programmet have adgang til definitionerne paa ALGOL-sproget og paa det ønskede maskinsprog. Definitionerne paa ALGOL kan ofte udtrykkes rekursivt. Naar oversætterprogrammet f.eks. skal analysere sætningen:

```
e := a + b×(c + d);
```

skal de enkelte navne og operatorer (+, ×, o.s.v.) jo forekomme efter et bestemt system. Naar lighedstegnet er passeret, skal oversætteren gøre klar til at modtage et regneudtryk. Her er a det første led, plustegnet er en tilladt operator, o.s.v., men saa snart venstreparentesen er mødt, skal programmet igen gøre klar til at modtage et helt regneudtryk. Analysen af regneudtrykket er altsaa rekursiv.

7. ANDRE ALGOLBEGREBER.

I dette kapitel omtaler vi visse ALGOL-begreber, som ikke har været nævnt tidligere, eller kun løst antydet.

7.1. Blokke.

ALGOL-sproget er opbygget af sætninger. Vi har tidligere set eksempler paa betingede sætninger, hopsætninger, <u>for</u>-sætninger, proceduresætninger og de almindelige sætninger, som beregner værdien af et udtryk.

Sætningerne adskilles med semikolon:

```
........
Sætning;
Sætning;
Sætning;
........
```

Vi har ogsaa set, hvorledes man kan danne saakaldte sammensatte sætninger ved at omgive en række almindelige sætninger med <u>begin</u> og <u>end</u>:

```
........
Sætning;
Sætning;
begin
   Sætning;
   Sætning;
   Sætning
end;
Sætning;
........
```

Hvis man i begyndelsen af en sammensat sætning anbringer een eller flere deklarationer, kaldes den sammensatte sætning en blok:

```
........
Sætning;
Sætning;
begin
   Deklaration;
   Sætning;
   Sætning;
   Sætning
end;
Sætning;
........
```

En deklaration kan omfatte simple variable, talsæt og procedurer:

```
begin
   integer a;
   real b;
   array c[1:117];
   procedure d(n);
   ........
   ........
   ........
end;
```

Hertil kommer ogsaa deklarationer af logiske variable (boolean) og skiftespor (switch), som omtales i de følgende afsnit.

Reglen er nu den, at naar vi har deklareret nogle størrelser i en blok (her: a, b, c og d), da gælder disse navne indenfor denne blok - altsaa mellem det tilhørende begin og end - men ikke udenfor.

Man kan godt have flere blokke indeni hinanden. Hver gang maskinen paa sin vej gennem programmet via et begin passerer ind i en ny blok, faar den herved adgang til at regne paa de variable, som er deklareret her. Omvendt, naar den forlader en blok via det tilsvarende end, taber den adgangen til at regne paa de variable, der fandtes i den blok.

Hvis vi har et program med denne struktur:

```
begin
   real A, B;
   A := 1;
   B := 2;
   begin
      real A;
      A := 3;
      B := 5
   end;
   A := 4
end;
```

ser vi, at den yderste blok indeholder de to variable: A og B. Desuden er der en indre blok, som ogsaa indeholder en variabel ved navn A. De to Aer er i praksis to forskellige celler i maskinen. Saalænge vi befinder os i den yderste blok, altsaa f.eks. i sætningerne:

```
A := 1;
B := 2;
A := 4;
```

da er det det yderste A, som menes, og som programmet regner med. Naar vi derimod gaar ind i den inderste blok, som i sætningen:

```
A := 3;
```

da er det det inderste A, som menes, og vi har slet ikke adgang til det yderste A, før vi igen kommer ud i den yderste blok. Den inderste blok indeholder ogsaa sætningen:

```
B := 5;
```

og her menes det B, som er deklareret i den yderste blok, da der jo kun findes eet B i programmet. Vi kan derfor altid fra en indre blok arbejde med de variable, som er deklareret i en ydre blok, forudsat at navnene ikke er identiske med variable deklareret i den indre blok.

Medens man kun kan gaa ind i en blok via det tilhørende begin, behøver

man ikke nødvendigvis at gaa ud af blokken via det tilsvarende end. Man kan ogsaa benytte sig af en hopsætning, som fører til en etikette i en ydre blok. Naar maskinen hopper til den ydre blok, taber den adgangen til de variable, der er deklareret i den indre blok, ganske som om den var gaaet ud via end. Samtidigt retableres ogsaa adgangen til de ydre variable, som har samme navn som de indre.

For etiketter (labels) benyttes ingen deklaration i begyndelsen af den blok, hvori de gælder. Man skriver blot etiketten paa det sted, den skal staa. De skal altsaa opfattes som lokale for den inderste blok, i hvilken de staar. I programmet:

```
     begin
        real A;
        ........
Q:      A := 17;
R:      go to Q;
        ........
        begin
           real B;
           ........
Q:         B := 19;
S:         go to Q;
           ........
T:         go to R;
           ........
        end
     end;
```

vil den første sætning:

```
R: go to Q;
```

bevirke, at der hoppes til sætningen:

```
Q: A := 17;
```

medens den anden sætning:

```
S: go to Q;
```

bevirker hop til sætningen:

```
Q: B := 19;
```

fordi Q-et i den ydre blok ikke er tilgængeligt fra den indre blok paa grund af navnesammenfaldet. Sætningen:

```
T: go to R;
```

bevirker hop til R i den ydre blok.

Der er ikke noget i vejen for at man kan skrive ALGOL-programmer, hvor alle variable deklareres helt i begyndelsen af programmet, som da kun bestaar af een eneste blok. Naar man alligevel vælger at inddele programmet i blokke, er det dels for at gøre det mere overskueligt, men ogsaa for bedre at udnytte pladsen i maskinen. Hvis en gruppe variable kun benyttes i en lille del af programmet, bør denne del gøres til en blok, og de paagældende variable deklareres i begyndelsen af blokken. Man vil huske, at der for arrays gælder den særlige regel, at hvis de øvre og nedre grænser for de tilhørende indices ikke er skrevet som tal, men som navne paa variable eller som udtryk indeholdende variable, f.eks.:

```
array P[1:k], Q[n:n↑2 + 1];
```

da maa disse variable (her k og n) være deklareret i en ydre blok. Værdien af k og n maa ogsaa være beregnet i den ydre blok. Bemærk iøvrigt, at maskinen automatisk afrunder udtryk som n↑2 + 1 til et helt tal, hvis udtrykket er af en saadan form, at det ikke er garanteret at være et helt tal.

For simple variable (altsaa <u>integer</u>, <u>real</u> og de senere omtalte <u>boolean</u>) kan man udvide deklarationen med betegnelsen <u>own</u>, f.eks.:

```
........
........
begin
   real A;
   own real B;
   ........
   ........
```

```
end;
........
........
```

Meningen hermed er den, at hvis programmet gaar ind i denne blok flere gange, da vil B hver gang have samme værdi, som da blokken blev forladt sidste gang. Naar programmet befinder sig i denne blok, kan den regne paa de to variable A og B. Naar blokken forlades, er A og B ikke tilgængelige. Næste gang, programmet gaar ind i blokken, vil værdien af A være ubestemt, fordi den tilsvarende celle maaske har været brugt til andre variable udenfor blokken. Værdien af B er derimod uforandret fra den værdi, den havde, da blokken blev forladt sidste gang.

Den opmærksomme læser vil nu spørge om, hvad værdi B har, første gang blokken benyttes. Dette er desværre ubestemt. Programmøren maa selv holde regnskab med hvornaar blokken bruges første gang. Man vil derfor forstaa, at det kun er en begrænset glæde, man kan have af at bruge own-betegnelsen. I GIER-ALGOL oversættes own paa den simple maade, at alle own-variable fra alle blokke i programmet anbringes, som om de var deklareret i den yderste blok, naturligvis med den væsentlige forskel, at man ikke kan bruge dem i den yderste blok, men kun i den blok, hvor hver især er deklareret.

I GIER-ALGOL er det kun simple variable, ikke talsæt som kan være own.

Vi vil nu vise et eksempel paa, hvorledes GIER ALGOL-oversætteren anbringer de forskellige variable i maskinen. (Dette eksempel kan eventuelt overspringes ved en første gennemlæsning). Vi ser paa programmet:

```
begin
   integer a, b;
   real c, d, e;
   array f[11:15];
   for a := 11 step 1 until 15 do f[a] := a + 6;
   b := 3;
   c := 4;
   d := 5;
   e := 6;
   begin
      own real g, h;
      real i, j;
      array k[11:12, 17:19];
```

```
        g := 12;
        h := 13;
        i := 14;
        j := 15;
        for a := 11 step 1 until 12 do
        for b := 17 step 1 until 19 do k[a,b] := 100×a + b;
        begin
           real l, m;
           l := 21;
           m := 22;
           begin
              real n, o;
              n := i + l + j + m;
              o := i + j + l + m;
              o := o + 1
           end
        end
     end;
     a := 2
  end;
```

Programmet indeholder fire blokke indeni hinanden. Hvis vi standser maskinen i dens udregninger umiddelbart efter at den har udført sætningen:

```
o := o + 1;
```

og hvor den altsaa staar i den inderste blok med adgang til de variable i alle blokniveauer, da vil talmaterialet være fordelt saaledes i maskinens celler (fordelingen svarer til forholdende i ALGOL II, men er omtrent uændret i ALGOL 4):

Celle nr.	Variabel	Talværdi	Relativ adresse
789	n	72	- 2
790	o	73	- 1
791	Reference:	795	
792	Blokstart:	789	
793	l	21	- 2
794	m	22	- 1
795	Reference:	811	
796	Blokstart:	793	
797	k[11,17]	1117	-14
798	k[11,18]	1118	-13
799	k[11,19]	1119	-12
800	k[12,17]	1217	-11
801	k[12,18]	1218	-10
802	k[12,19]	1219	- 9
803	Arbejdscelle		- 8
804	Arbejdscelle		- 7
805	i	14	- 6
806	j	15	- 5
807	Data for k	810 og 803	- 4
808	Første element	50	- 3
809	Antal elementer	6	- 2
810	Koefficient	3	- 1
811	Reference:	827	
812	Blokstart:	797	
813	f[11]	17	-14
814	f[12]	18	-13
815	f[13]	19	-12
816	f[14]	20	-11
817	f[15]	21	-10
818	Arbejdscelle		- 9
819	a	13	- 8
820	b	20	- 7
821	c	4	- 6
822	d	5	- 5
823	e	6	- 4
824	Data for f	827 og 818	- 3

825	Første element	11	- 2
826	Antal elementer	5	- 1
827	Reference:	0	
828	Blokstart:	813	
829	g	12	
830	h	13	
831	Bruges internt		
832	DISPLAY[3]	791	
833	DISPLAY[2]	795	
834	DISPLAY[1]	811	
835	DISPLAY[0]	827	

Hvis vi begynder med at se paa den inderste blok, saa er de to variable: n og o anbragt i cellerne 789 og 790. De to celler 791 og 792 indeholder administrative oplysninger om denne blok. Celle 791 indeholder en saakaldt reference, nemlig adressen (her 795) paa den tilsvarende referencecelle i den næstinderste blok. Celle 792 indeholder tallet 789, som er adressen paa den celle, hvori denne bloks data begynder. Disse to celler findes i alle fire blokniveauer, lige over den punkterede linie. For den næstinderste blok er de to variable: l og m anbragt i cellerne 793 og 794.

I den næstyderste blok er de to real variable: i og j anbragt i celle 805 og 806. De seks elementer i talsættet: k findes i cellerne 797 - 802. Iøvrigt bruges cellerne 807 - 810 til at gemme nærmere oplysninger om dette talsæt. Da der er to indices i talsættet, sker beregningen af adressen paa et element k[p,q] ved at danne udtrykket:

$$p \times 3 + q$$

Tallet 3 kommer fra at der er tre mulige værdier for q. Dette 3-tal staar i celle 810. Skal vi beregne adressen paa elementet k[12, 18], faar vi først 12×3 + 18 = 54. Denne foreløbige adresse bliver kontrolleret ved hjælp af de to tal i cellerne 808 og 809 (50 og 6). Først subtraheres 50, og resultatet: 4 skal være positivt eller 0. Derefter subtraheres 6 og resultatet: -2, skal være negativt. Endelig adderes det andet tal (803) i celle 807 og vi faar: 803 - 2 = 801, som er adressen paa det søgte element. Vi ser, at det kontrolleres, at den beregnede adresse, p×3 + q, giver een af k-cellerne fra 797 til 802, men det kontrolleres ikke at p og q

hver for sig ligger i det rigtige omraade. Adressekontrollen for talsætelementer sker hver gang maskinen i det oversatte program skal have fat i et element. Som nævnt side 43 kan man faa oversætterprogrammet til at udelade indexkontrollen.

I den yderste blok har vi f-talsættet i cellerne 813 - 817 og de simple variable: a til e i cellerne 819 - 823. De administrative oplysninger om f-talsættet findes i cellerne 824 - 826. Bemærk, at de to own variable: g og h, er anbragt i cellerne 829 og 830.

De anførte relative adresser (sidste søjle i oversigten) bruges paa den maade, at hvis maskinen f.eks. staar i den næstyderste blok, og den skal have fat i den simple variable: j, beregnes denne adresse (806) som p - 5, hvori p altsaa er 811. Tallet 811 er referencen for den aktuelle blok. Cellerne 832 - 835 indeholder en liste over referencerne for de forskellige blokke. Hvis maskinen derimod skal bruge en variabel, som er deklareret i en ydre blok, bliver adresseberegningen lidt mere besværlig. Er vi i den næstinderste blok, findes adressen paa den variable: m som:

```
adresse := p - 1;
```

hvor p er 795. Skal vi fra samme blok have fat i den variable: j, finder maskinen adressen saaledes:

```
s := 811;
adresse := s - 5;
```

Maskinen opererer altsaa med de to variable: p og s, som i virkeligheden er to specielle registre (ikke almindelige celler). Referencen for den aktuelle blok (her 795) er altid fast anbragt i p-registret og ændres først, naar maskinen ændrer aktuel blok. Referencen for de ydre blokke maa derfor anbringes i et andet register, før adressen kan udregnes. Indholdet af s-registret skal derfor indsættes af maskinen før den kan regne paa ikke-lokale variable, og indholdet skal ændres, hver gang der skiftes fra ikke-lokale variable deklareret i een ydre blok til ikke-lokale variable deklareret i en anden ydre blok. Som eksempel kan vi tage sætningen:

```
n := i + l + j + m;
```

hvor i og j er deklareret i den næstyderste blok og l og m i den næstinder-

ste, medens sætningen staar i den inderste blok. Programmet herfor er:

```
s := 795;
Hent l fra s - 2
s := 811;
Adder i fra s - 6
Adder j fra s - 5
s := 795;
Adder m fra s - 1
Gem n i p - 1
```

Kort sagt: hvis man hele tiden veksler mellem variable fra mange forskellige blokniveauer, tabes der en hel del plads og tid til de ordrer, som ændrer s-registret.

For variable i den yderste blok benyttes iøvrigt ofte absolutte adresser.

I eksemplet ovenfor var alle blokke anbragt inden i hinanden. Normalt vil man ogsaa have sideordnede blokke, som f.eks.:

```
begin
   real A, B, C;
   A := 1;
   B := 2;
   begin
      real D, E, F;
      D := 4;
      E := 5;
      F := 6
   end;
   C := 3;
   begin
      real G, H, I;
      H := 7;
      writecr;
      write(⟨ddd⟩, G, H, I)
   end
end;
```

Her har vi den ydre blok med de tre variable: A, B og C. Derefter kommer en indre blok med de tre variable: D, E og F. Denne blok forsvinder igen og derefter kommer der en anden indre blok med de tre variable: G, H og I. I den første indre blok har vi adgang til: A, B, C, D, E og F og i den anden indre blok til A, B, C, G, H og I.

Den anden blok indeholder sætningen:

```
H := 7;
```

samt trykning af ny linie og trykning af talværdierne af G, H og I. Man kan se, at programmet slet ikke indeholder sætninger, som tildeler G og I nogen værdi. Dette er aabenbart en programmeringsfejl. Programmet trykker tallene:

```
4   7   6
```

som værdier for G, H og I. Hvis man sammenligner de to blokke, ser man, at de begge indeholder tre variable, idet D, E og F svarer til G, H og I. Da blokkene er helt ens og sideordnede, vil D, E og F staa i de samme celler som G, H og I. Dette er grunden til at vi faar trykt G = 4 og I = 6. Eksemplet illustrerer en typisk programmeringsfejl, men man maa endelig ikke tro, at der herved er aabnet mulighed for bevidst at overføre tal fra een blok til en anden, sideordnet blok. Hvis de to sideordnede blokke ikke er helt ens, vil det normalt være helt umuligt at overse, hvad der svarer til hvad i de to blokke, især hvis der findes celler med administrativt indhold som talsætgrænser, etc.

Som bemærket ovenfor, tillader GIER ALGOL ikke anvendelsen af <u>own</u> <u>arrays</u>. Det er navnlig <u>own</u> <u>arrays</u> med variable grænser, som volder vanskeligheder:

```
begin
   own array A[P:Q];
   .........
```

Hvis P og Q varierer mellem hvert kald af denne blok, er det svært at finde en fornuftig maade at oversætte programmet paa.

7.2. Logiske Variable.

I afsnit 4.5 blev omtalt forskellige logiske operationer, som vi nu vil forklare nøjere, samt indføre egentlige logiske variable.

Hvis vi har en betingelsessætning, f.eks.:

```
if x > 100 then writetext(‹<FEJL›);
```

der jo vil bevirke udskrift af ordet: FEJL, hvis x > 100, har betingelsen ofte form af en relation mellem talstørrelser og variable, her: $x > 100$. Saadanne relationer kan godt være et langt, kompliceret udtryk f.eks.:

$$A + 3\times C - 4\times D \geq (P + Q)\uparrow 2$$

Hvis man har et stort program, hvori denne betingelse optræder i mange betingelsessætninger, vil det være en fordel, om man kan nøjes med at udregne betingelsen første gang den forekommer, gemme værdien af betingelsen i en celle i en eller anden form, og derefter de følgende gange blot hente værdien frem fra cellen igen, uden at foretage nogen beregning.

En betingelse kan kun have een af to værdier: sand eller falsk. Man har derfor indført en særlig type variable, som kun kan antage de to værdier: sand eller falsk. Denne type variable betegnes: boolean efter Englænderen Boole, der for ca. 100 aar siden grundlagde regnereglerne for logiske variable.

Denne type variable skal deklareres ligesom integer og real:

```
begin
   real A, B;
   boolean S, T;
   ........
   ........
end;
```

Naar vi skal gemme værdien sand eller falsk af en boolean i en celle, maa det ske i form af et tal, men der er ikke nogen konvention for hvilket tal, der svarer til sand, og hvilket til falsk. I GIER ALGOL har man valgt

at lade sand være - 1 og falsk 0, men det er ikke tilladt at skrive:

```
begin
   boolean S, T;
   S := -1;
   T :=  0;
   ........
end;
```

I stedet skal man bruge de to særlige ALGOL-gloser: true og false:

```
begin
   boolean S, T;
   S := true;
   T := false;
   ........
end;
```

Sætningen:

```
S := true;
```

er den simpleste form for en sætning, som tildeler en logisk variabel en værdi. Normalt vil der paa højre side staa en relation, f.eks.:

```
S := x > 100;
```

Man ser ofte begyndere skrive saaledes:

```
if x > 100 then S := true
else S := false;
```

men dette er alt for klodset, naar man lige saa godt kan skrive:

```
S := x > 100;
```

Som et simpelt eksempel laver vi nu et program, som læser 10 tal fra

en strimmel og undersøger, om der er negative tal iblandt:

```
begin
   integer k;
   real element;
   boolean NEG;
   array A[1:10];
   select(8);
   NEG := false;
   for k := 1 step 1 until 10 do
   begin
      element := read real;
      A[k] := element;
      if element < 0 then NEG := true
   end for k;
   if NEG then
   begin
      writecr;
      writetext({<Negativ});
      writecr;
   end if NEG
end program;
```

Vi har her en logisk variabel: NEG, som sættes til falsk i den første sætning:

```
NEG := false;
```

Derefter kommer for-sætningen, som 10 gange læser et element, gemmer det i A[k]-cellen, og undersøger dets fortegn. Hvis elementet er negativt, sættes NEG til sand. Naar for-sætningen er forbi, faar man fejludskrift, hvis NEG er sand. Bemærk, at her kan vi ikke erstatte sætningen:

```
if element < 0 then NEG := true;
```

med:

```
NEG := element < 0;
```

fordi NEG da kun vil være sand, hvis det sidste element var negativt.

Man kan ogsaa have logiske arrays, som maa deklareres som almindelige arrays:

```
boolean array S[1:N];
```

Vi kan f.eks. udvide programmet ovenfor, saaledes at der nu indlæses tre talgrupper hver med 10 tal. For hver af de tre grupper skal man have en boolean som angiver, om der har været negative tal imellem. Det sker i et boolean array:

```
boolean array NEG[1:3];
```

Programmet er:

```
begin
   integer j, k;
   real element;
   boolean array NEG[1:3];
   array A[1:3, 1:10];
   select(8);
   for j := 1 step 1 until 3 do
   begin
      NEG[j] := false;
      for k := 1 step 1 until 10 do
      begin
         element := read real;
         A[j,k] := element;
         if element < 0 then NEG[j] := true
      end for k
   end for j;
   if NEG[1] ∨ NEG[2] ∨ NEG[3] then
   begin
      writecr;
      writetext(≮<Negativ:≯);
      for j := 1 step 1 until 3 do if NEG[j] then write(≮ddd≯, j);
      writecr;
   end if
end program;
```

Hvis der er negative tal i f.eks. gruppe 2 og 3, faar man udskriften:

```
Negativ:  2  3
```

Man kan ogsaa have procedurer med logiske variable som parametre. Hvis vi skal lave to procedurer til regning med endimensionale talsæt:

```
array A, B, C[1:n];
```

saaledes at den ene procedure skal addere A og B i en for-sætning:

```
for i := 1 step 1 until n do C[i] := A[i] + B[i];
```

medens den anden procedure skal subtrahere:

```
for i := 1 step 1 until n do C[i] := A[i] - B[i];
```

vil det normalt kunne betale sig at slaa de to procedurer sammen til een og lade en logisk variabel vise, om det er addition eller subtraktion. Her er først de to procedurer uden anvendelse af logisk variabel:

```
procedure ADD(n, A, B, C);
value n;
integer n;
array A, B, C;
begin
   integer i;
   for i := 1 step 1 until n do C[i] := A[i] + B[i]
end ADD;

procedure SUB(n, A, B, C);
value n;
integer n;
array A, B, C;
begin
   integer i;
```

```
   for i := 1 step 1 until n do C[i] := A[i] - B[i]
end SUB;
```

Og her er den fælles procedure:

```
procedure ADDSUB(plus, n, A, B, C);
value plus, n;
boolean plus;
integer n;
array A, B, C;
begin
   integer i;
   for i := 1 step 1 until n do
   C[i] := A[i] + (if plus then B[i] else - B[i])
end ADDSUB;
```

Som eksempel paa brugen heraf kan vi give et program, hvori vi skiftevis skal addere og subtrahere talsættet T:

```
begin
   integer j;
   boolean PLUS;
   array S, T[1:10];
   comment Her deklareres ADDSUB;
   PLUS := true;
   comment Her indsættes startværdier af S;
   for j := 1 step 1 until 17 do
   begin
      comment Her beregnes værdierne af T;
      ADDSUB(PLUS, 10, S, T, S);
      PLUS := ¬ PLUS
   end for j
end program;
```

Programmet indeholder den logiske variable: PLUS, som først sættes til sand:

```
PLUS := true;
```

Derefter udføres for-sætningen med j 17 gange, idet der hver gang først beregnes nye værdier af T, og derefter kaldes ADDSUB. Da PLUS første gang var sat til sand, vil det første kald give addition. Sidst i for-sætningen udføres sætningen:

```
PLUS := -, PLUS;
```

Vi husker, at tegnet: -, betyder ikke. Hver gang sætningen udføres, vil værdien af PLUS skifte, første gang fra sand til falsk, næste gang fra falsk til sand, o.s.v.

Ved det andet kald af ADDSUB faar vi derfor subtraktion, ved det tredje addition igen, o.s.v.

Ligesom man kan have en real procedure kan man ogsaa have en boolean procedure. Vi kan f.eks. deklarere en saadan procedure, som afgør om et foreliggende tal er lige eller ulige:

```
boolean procedure LIGE(n);
value n;
integer n;
LIGE := n:2×2 = n;
```

Princippet er det samme som nævnt side 58. I GIER ALGOL 4 kan man ogsaa skrive betingelsen at et tal er lige som:

```
LIGE := n mod 2 = 0;
```

idet modulo-operatoren, mod, giver divisionsresten af tallet n ved division med 2. I almindelighed giver N mod M divisionsresten af N ved division med M. Modulo-operatoren er ikke officielt tilladt i ALGOL, men er indført af praktiske grunde i GIER ALGOL 4.

Paa side 49 omtalte vi de forskellige logiske operatorer. Hvis vi har et program med de to logiske variable b1 og b2:

```
begin
   boolean b1, b2;
```

```
        ........
        ........
    end;
```

vil man kunne anvende udtryk af formen:

b1 <operator> b2

idet de fleste logiske operatorer er tosidige, altsaa sammenknytter to variable. Dette gælder dog ikke nægtelsen: -, b1, som kun kræver een variabel. De fem operatorers betydning fremgaar af nedenstaaende tabel, hvor vi har angivet alle fire kombinationer af sand og falsk for b1 og b2.

Logisk Udtryk	Logisk værdi				Betegnelse
b1	false	false	true	true	
b2	false	true	false	true	
-, b1	true	true	false	false	Nægtelse
b1 ∧ b2	false	false	false	true	Og
b1 ∨ b2	false	true	true	true	Eller
b1 => b2	true	true	false	true	Implikation
b1 ≡ b2	true	false	false	true	Ækvivalens

Beregning af komplicerede logiske udtryk som:

b1 ∧ (b2≡b3 ∨ x > 100)

sker ligesom ved regneudtryk fra venstre til højre, og saaledes at parenteser udregnes før beregningen fortsættes. Iøvrigt skal operationerne udføres i følgende rækkefølge:

1. Regneudtryk
2. Relationer (<, ≤, =, ≥, >, ≠)
3. Nægtelse (-,)
4. Og (∧)
5. Eller (∨)
6. Implikation (=>)
7. Ækvivalens (≡)

Disse regler er analoge med reglerne for rækkefølgen af operatorer i regneudtryk (side 21).

7.3. Skiftespor.

Vi har set, at man kan have talsæt af typen real, integer og boolean. De tilsvarende variable optræder altsaa med indices. Hvis man har et program med mange etiketter, kan der undertiden opstaa et behov for ogsaa at forsyne etiketter med indices. Dette findes dog ikke i ALGOL, men man har noget, som minder herom. Det kaldes for skiftespor (switch) og kan illustreres med programmet:

```
begin
    integer n;
    switch S := L8, L19, L67, PQL;
    ........
    ........
    ........
    comment Her tildeles n en værdi ved beregning eller indlæsning;
    ........
    go to S[n];
    ........
L8: ........
    ........
PQL: ........
    ........
    ........
L19: ........
    ........
L67: ........
    ........
end;
```

Programmet indeholder heltalsvariablen: n og skiftesporet: S. Den sidste skal deklareres i begyndelsen af en blok, f.eks. som her:

```
switch S := L8, L19, L67, PQL;
```

som bestaar af ALGOL-glosen switch, af navnet: S, af det dynamiske lighedstegn (:=) og endelig en liste med de fire etiketter: L8, L19, L67 og PQL, som alle skal findes i programmet.

Ved selve brugen af S kan man f.eks. skrive:

```
go to S[n];
```

Hvis den øjeblikkelige talværdi af n er 3, vil denne sætning være ækvivalent med:

```
go to L67;
```

idet programmet opsøger den tredje etikette i deklarationen for S, altsaa her L67. Denne hop-sætning udføres derefter.

Værdien af n kan her være: 1, 2, 3 eller 4. Er værdien forskellig herfra, udføres hopsætningen ikke. Hvis man bruger et udtryk i stedet for n:

```
go to S[1 + a×b/c];
```

vil udtrykket blive afrundet til nærmeste hele tal, inden etiketten beregnes.

Programeksemplet paa side 53 kan ogsaa skrives ved hjælp af et skiftespor, f.eks. saaledes:

```
begin
    real sum, forhold, element;
    integer k, TYPE;
    array A[1:6];
    switch SW := T1, T2, T3;
```

```
ST:     select(8);
        TYPE := read integer;
        for k := 1 step 1 until 6 do
        A[k] := read real;
        go to SW[TYPE];
T1:     sum := 0;
        for k := 1 step 1 until 6 do sum := sum + A[k];
        if abs(sum-100) > 0.02 then
        begin
           writecr;
           writetext({<SUM IKKE 100})
        end;
        go to SLUT;
T2:     for k := 1 step 1 until 6 do
        begin
           element := A[k];
           if element < 0 ∨ element > 100 then
           begin
              writecr;
              writetext({<FEJL I KOMPONENT});
              write({dd}, k)
           end fejludskrift
        end for k;
        go to SLUT;
T3:     forhold := A[1]/A[2];
        if forhold < 1 ∨ forhold > 5 then
        begin
           writecr;
           writetext({<SKÆVT FORHOLD})
        end;
SLUT:   writecr;
     end af programmet;
```

Vi har tidligere gjort opmærksom paa, at det er uøkonomisk at bruge mange etiketter. Dette gælder derfor ogsaa for brugen af skiftespor. For programmet ovenfor bør man foretrække udgaven uden etiketter, som er vist side 56.

7.4. Brug af integer og real.

Vi skal nu diskutere lidt nøjere forskellen paa de to talformer: integer og real.

En variabel, r, af typen real er i GIER begrænset til at ligge i intervallet:

$$7.458_{10}-155 < abs(r) < 1.341_{10}154$$

eller at være eksakt nul. Er tallet forskelligt fra nul, opbevares det med en nøjagtighed svarende til 8-9 decimaler. Hvis vi udregner:

r := r + dr;

hvor r og dr er real, er den mindste, mulige relative ændring i r mellem $2_{10}-9$ og $4_{10}-9$. Hvis dr er mindre end svarende hertil, vil værdien af r være nøjagtig den samme efter sætningen: r := r + dr; som før sætningen.

Hvis en real variabel overskrider grænsen: $1.341_{10}154$, vil maskinen stoppe beregningen og skrive ordet:

spill

paa skrivemaskinen. Det samme sker, hvis man forsøger at dividere med nul. Hvis en real bliver numerisk mindre end $7.458_{10}-155$, sættes den til eksakt nul.

Naar talværdien af en real variabel skal gemmes i en celle, skriver maskinen det som produktet af en 2-potens og et tal mellem 1 og 2. Har vi f.eks. tallet 17.75, skrives det som 16×1.109375 eller 2↑4×1.109375. I cellen ser det saaledes ud (saakaldt lagring med flydende komma):

0000000100|01.000111000000000000000000000

Den første fjerdedel af cellen er forbeholdt 2-potensen, hvoraf man kun gemmer 4-tallet, ikke 2-tallet. 4-tallet er:

0000000100

idet man i totalsystemet tæller saaledes:

Totalsystem	Titalsystem
1	1
10	2
11	3
100	4
101	5

o.s.v.

Taldelen 1.109375 repræsenteres af:

1.00011100000 ...... o.s.v.

der skal opfattes som:

1 hel
+ 0 halve
+ 0 fjerdedele
+ 0 ottendedele
+ 1 sekstendedel
+ 1 toogtredivtedel
+ 1 fireogtresindstyvendedel
o.s.v.

Ændrer vi tallet 17.75 til 17, skrives det som 2|4×1.0625, eller:

0000000100|01.0001000000000000000000000000

Hvis vi fra dette tal 17 trækker det mindst mulige, som lige netop kan ses paa tallet, bliver det nu til:

0000000100|01.0000111111111111111111111111

De mange 1-taller fremkommer ved menteoverføring, fordi vi skal trække 1 fra paa sidste plads og maa laane hele vejen op. Naar man i praksis skal

regne med en real variabel, som det her viste 17 eller næsten-17, spiller det normalt ingen rolle, hvilket af de to ovenfor viste celleindhold man har med at gøre, fordi forskellen er saa lille. Vanskeligheden opstaar, naar vi skal sammenligne to tal. Hvis vi har de to real variable r1 og r2 og faar udført sætningerne:

```
r1 := 17;
r2 := næsten-17;
```

vil en senere sammenligning, som f.eks.:

```
if r1 = r2 then .....;
```

altid give som resultat, at r1 og r2 er forskellige.

Naar vi skal udføre tællinger, som i for-sætninger o. lign., vil det være bekvemt at kunne bruge en type variable, hvor der altid automatisk er sørget for, at hvis tallet fremkommer ved en udregning, at det saa vil blive afrundet til det nærmeste heltal. Denne type variable kaldes integer. Medens tal af typen real i GIER gemmes paa ovennævnte maade, benyttes der en anden tallagring for tal af typen integer i GIER ALGOL 4. Her udnyttes alle 40 bits i cellen til heltallet, som staar med enheden i sidste bit tilhøjre (bit 39). Eksempler:

```
 4: 0000000000000000000000000000000000000100
 3: 0000000000000000000000000000000000000011
 2: 0000000000000000000000000000000000000010
 1: 0000000000000000000000000000000000000001
 0: 0000000000000000000000000000000000000000
-1: 1111111111111111111111111111111111111111
-2: 1111111111111111111111111111111111111110
```

Naar tallene gemmes paa denne maade, vil de altid være eksakt hele tal, fordi der simpelthen ikke er afsat plads til at gemme decimalerne.

De to forskellige slags tallagring giver noget besvær for oversætterprogrammet, især i sætninger, som indeholder begge typer tal. Her kræves en omregning mellem de to lagringsformer.

Dette kan illustreres med nogle eksempler. Vi har et program af formen:

```
begin
   integer i1, i2, i3, i4;
   real r1, r2;
   ........
   ........
end;
```

Ved sætninger af formen:

```
r1 := udtryk af ren real type;
```

sker der ingen omregning. Det gør der heller ikke i:

```
i1 := i2 + i3×i4;
```

fordi højresiden kun indeholder integer variable, samt addition og multiplikation. Omregning er derfor overflødig. Beslutningen herom træffes af oversætterprogrammet, idet det fremgaar af sætningens form.

I følgende tilfælde finder der afrunding sted, inden i1 gemmes:

```
i1 := r1;
i1 := r1 + i1;
i1 := i2 + i3/i4;
```

Hvis højresiden indeholder real variable, skal der afrundes, fordi hele højresiden udregnes som real. I udtrykket i2 + i3/i4 skal der ogsaa afrundes, fordi i3/i4 altid udføres som en real division, d.v.s. eventuelle decimaler medtages. Skriver vi derimod:

```
i1 := i2 + i3:i4;
```

sker der ingen afrunding, fordi i3:i4 altid er et heltal.

Afrunding af et udtryk, U, sker ved at man beregner entier (U + 0.5). Der adderes altsaa 0.5 til udtrykket, hvorefter man tager det største hele tal, som er mindre end eller lig med resultatet.

For integer variable gælder følgende grænser:

$$-2\uparrow 39 = -549\ 755\ 813\ 888 \leq i1 \leq 549\ 755\ 813\ 887 = 2\uparrow 39-1$$

altsaa ca. en halv billion. Maskinen giver ikke alarm, hvis en integer variabel kommer udenfor dette interval som følge af en ren integer addition eller subtraktion. Resultatet bliver da komplet meningsløst, uden at man faar det at vide. Anvendes ren integer multiplikation:

i1 := i2×i3;

faar man fejludskriften: mult, under programmets kørsel, hvis i2×i3 kommer udenfor det tilladte integer interval. De nøjere regler for fejludskrift ved blandede udtryk er beskrevet i Naur(1967).

For procedurer gælder følgende hovedregel. For parametre kaldt ved value er det tilladt at lade en aktuel real svare til en formel integer og omvendt. Oversætterprogrammet vil da indsætte de nødvendige omregninger. Hvis en parameter er kaldt ved name, skal de aktuelle parametre være af samme type som de formelle, ellers faas fejludskrift under oversættelsen.

Det er værd at bemærke, at den officielle definition paa ALGOL kun forlanger, at de formelle parametre, som er kaldt ved value, skal specificeres i proceduredeklarationen.

I GIER ALGOL skal man dog specificere alle formelle parametre, og det samme krav stilles af de fleste andre ALGOL-oversættere. Det ville have været forholdsvist kostbart i regnetid og pladskrav, hvis man tillod udeladelse af specifikationerne, fordi programmet da under kørsel med det oversatte program hele tiden skulle undersøge, hvilken type, der foreligger.

I GIER ALGOL 4 faar man fejludskriften:

formal

hvis man under kørslen af et oversat program forsøger at assigne til en formel parameter, som er kaldt ved name, og som ikke er en variabel, men et udtryk.

I GIER ALGOL 4 er der en særlig mulighed for at snyde den indbyggede typekontrol i oversætteren, hvilket i enkelte tilfælde kan være praktisk. For de nærmere regler herfor henvises til Naur (1967). Som et noget enfoldigt eksempel herpaa anføres følgende program, hvori vi har et array paa 40 elementer, A[1:40], hvis størrelse netop svarer til hvad der kan transporteres frem og tilbage mellem baggrundslageret og selve lageret. Hvordan disse transporter udføres, interesserer os ikke her, vi antager blot at det kan lade sig gøre. Endvidere antager vi, at de 40 elementer alle skal være af typen <u>real</u>, men at vi meget gerne vil have, at element nr. 17 bliver af typen <u>integer</u>, fordi der deri skal staa et stort heltal. Dette kan programmeres saaledes:

```
begin
   integer i, j;
   array A[1:40];
   select(8);
   for i := 1 step 1 until 16 do A[i] := read real;
   j := read integer;
   A[17] := real j;
   for i := 18 step 1 until 40 do A[i] := read real;
   comment Nu kan hele arrayet A f.eks. overføres til disken og
   senere hentes frem igen.  Skal vi derefter have fat i heltallet
   A[17] maa man skrive:;
   .......
   j := integer A[17];
   .....
end program;
```

Det interessante er her de to sætninger:

```
A[17] := real j;

j := integer A[17];
```

Betegnelsen: real j betyder altsaa, at den variable j skal opfattes som om den var af typen real, for at det bliver lovligt at gemme den i cellen A[17]. Bemærk, at j ikke bliver omregnet til at have formen real. Paa samme maade betyder integer A[17], at cellen A[17] ikke skal omregnes til integer form.

Disse finesser er af væsentlig større betydning ved operationer med variable af typen boolean, hvorved man har mulighed for at operere paa de enkelte bits i en celle. Det bemærkes, at disse foranstillede typekontroleliminatorer ikke findes i det officielle ALGOL.

7.5. Specielle for-sætninger.

I dette afsnit skal vi se paa en lidt nøjagtigere definition af de tre typer af for-sætninger, som findes i ALGOL.

Den almindeligste type er step-until typen, som vi har brugt mange gange i det foregaaende. Har vi f.eks. programstykket:

```
L1:   for V := A step B until C do
L2:   SÆTNING 1;
L3:   SÆTNING 2;
      ........
```

hvori betegnelsen: SÆTNING 1 dækker over den sætning, som skal styres af V, saa giver den officielle ALGOL-definition en detailleret forklaring paa for-sætningens virkemaade, idet man - stadig i ALGOL - omskriver til de følgende simplere elementer:

```
L1:   V := A;
L1a:  if (V-C)×sign(B) > 0 then go to L3;
L2:   SÆTNING 1;
      V := V + B;
      go to L1a;
L3:   SÆTNING 2;
      ........
```

I de fleste tidligere eksempler har vi haft A = 1 og B = 1, men alle muligheder er naturligvis tilladt. Især kan B være negativ og behøver ikke at være netop 1 eller -1. Hvis B er positiv, og startværdien A er større end slutværdien, C, vil SÆTNING 1 slet ikke blive udført. Det eneste, der sker, er at V bliver sat lig med startværdien. Dette er iøvrigt i modsætning til kodesproget FORTRAN, hvor SÆTNING 1 altid vil blive udført mindst een gang.

Bemærk iøvrigt, at man aldrig maa sætte B lig med nul, fordi betingelsen: <u>if</u> (V-C)×sign(B) > 0 da aldrig kan blive opfyldt. Sætningen: <u>go to</u> L3 bliver derfor aldrig udført, og maskinen kører derfor rundt i forsætningen uendeligt længe, med mindre der i selve SÆTNING 1 findes en hopsætning, som fører ud af for-sætningen. Dette er en oplagt fælde for begyndere. Hvis vi laver et program, som skal trykke en tabel over visse egenskaber ved forskellige temperaturer og lader beregningen styre af forsætningen:

<u>for</u> T := TSTART <u>step</u> DELT <u>until</u> TSLUT <u>do</u>

og lader brugeren af programmet opgive de aktuelle værdier af TSTART, DELT og TSLUT, kan der ske fejl, hvis man kun ønsker tabellen beregnet for en enkelt temperatur, f.eks. 500 gr. Hvis brugeren specificerer:

```
TSTART := 500;
DELT   :=   0;
TSLUT  := 500;
```

bliver maskinen aldrig færdig med at regne. Specificerer han derimod:

```
TSTART := 500;
DELT   :=   1;
TSLUT  := 500;
```

gaar det godt, og vi faar netop een tabelværdi. Kun de færreste brugere vil finde det logisk, at man skal opgive DELT > 0, naar temperaturen faktisk ikke skal øges. Det vil derfor være klogere af programmøren, hvis han lader antallet af tabelværdier, N, være input i stedet for TSLUT. Man skal da skrive for-sætningen saaledes:

```
for i := 1 step 1 until N do
begin
   T := TSTART + (i-1)×DELT;
   ........
   ........
```

Nu kan vi som input opgive:

```
TSTART := 500;
DELT   :=   0;
N      :=   1;
```

Naar N = 1, er værdien af DELT ligegyldig.

En anden fælde ved brugen af for-sætninger er spørgsmaalet om hvilken værdi, den styrende variable, V, har naar for-sætningen er afsluttet. Ser vi paa den detaillerede forklaring paa side 148, kommer vi til det resultat, at for for-sætningen:

```
for V := 1 step 1 until N do
```

vil V have værdien N + 1, naar for-sætningen er afsluttet, undtagen hvis N < 1, da er V = 1. Ligeledes for

```
for V := N step -1 until 1 do
```

vil V være nul, naar for-sætningen er færdig, eller N, hvis N < 1.

Den variable, V, vil altsaa normalt køre et trin for langt eller være lig med startværdien, hvis SÆTNING 1 slet ikke udføres.

Den officielle ALGOL-definition siger imidlertid, at naar for-sætningen er færdig, er værdien af V udefineret. Det betyder, at de forskellige forfattere af ALGOL-oversætterprogrammer er frit stillet. Man kan derfor risikere, at nogle oversætterprogrammer giver V = N efter udførelsen af

```
for V := 1 step 1 until N do
```

i stedet for V = N + 1, eller maaske en helt tredje mulighed. Hvis programmøren ønsker at bruge den variable V efter for-sætningen, maa han derfor udtrykkeligt tildele den en værdi:

```
for V := 1 step 1 until N do
begin
   ........
   ........
   ........
end for V;
V := N;
........
........
```

Hvis man hopper ud af SÆTNING 1 inden for-sætningen er helt færdig, er V naturligvis veldefineret. Man kan f.eks. skrive:

```
    for V := 1 step 1 until N do
    begin
       ........
       ........
       if V = N then go to L4;
       ........
       ........
    end for V;
L4: ........
```

Meningen er aabenbart, at den sidste del af den sammensatte SÆTNING 1 ikke skal udføres for V = N. Naar maskinen kommer til L4, er V = N (eller 1, hvis N < 1).

Det samme kan ogsaa kodes saaledes:

```
    for V := 1 step 1 until N do
    begin
       ........
       ........
       if V ≠ N then
       begin
          ........
          ........
       end if
    end for V;
L5: ........
```

Her er værdien af V udefineret, naar maskinen naar frem til L5, omend vi i GIER ALGOL ved, at V = N + 1 (eller 1 for N < 1).

Paa side 79 saa vi et eksempel paa den anden type for-sætning, som blot indeholder en liste over de værdier, V skal antage. Formen heraf er:

```
L1:     for V := P, Q, R do
L2:     SÆTNING 1;
L3:     SÆTNING 2;
```

Dette udfører maskinen paa en maade, der omtrent svarer til :

```
L1:     begin
           integer i;
           switch SW := L1a, L1b, L2a;
           V := P;
           i := 1;
           go to L2;
L1a:       V := Q;
           i := 2;
           go to L2;
L1b:       V := R;
           i := 3;
L2:        SÆTNING1;
           go to SW[i];
L2a:    end;
L3:     SÆTNING2;
```

Man ser, at V først sættes lig P og SÆTNING 1 udføres. Derefter sættes V lig med Q og SÆTNING 1 udføres, o.s.v. Naar listen: P, Q, R er udtømt, gaar maskinen videre med programmet efter for-sætningen. Værdien af V er ogsaa udefineret her, men i GIER ALGOL kan man regne med at V er lig med det sidste element i listen, altsaa her R.

Man kan ogsaa have kombinationer af de to typer for-sætninger, f.eks.:

```
for V := 1 step 1 until 6, 8, 13, 15 do
```

eller:

```
for V := 1, 5, 10 step 10 until 100, 200 step 100 until 1000 do
```

De forskellige typer af styring adskilles altsaa med et komma.

Den tredje type for-sætning har ikke været vist tidligere. Den har formen:

```
L1:    for V := E while F do
L2:    SÆTNING1;
L3:    SÆTNING2;
```

Her er F et logisk udtryk, som altsaa kan være sand eller falsk. Udførelsen sker saaledes:

```
L1:    V := E;
       if ¬, F then go to L3;
L2:    SÆTNING1;
       go to L1;
L3:    SÆTNING2;
```

Bemærk, at V sættes lig med E i begyndelsen af hvert gennemløb. E vil da normalt være et udtryk, som faar V til at variere, f.eks.:

```
        V := Vstart - 1;
        for V := V + 1 while F do
L2:     SÆTNING1;
```

Her bliver V øget med 1 i hvert gennemløb. Det logiske udtryk, F, vil normalt blive indstillet af, hvad der foregaar i SÆTNING 1, men startværdien af F maa naturligvis være true, hvis for-sætningen overhovedet skal komme i gang.

Her er et eksempel paa brug af en for-sætning med while. Det er en procedure, som til et opgivet, ulige tal, x, finder det næste højere ulige primtal:

```
    integer procedure PRIM1(x);
    integer x;
    begin
       integer y;
A:     PRIM1 := x := x + 2;
       y := 1;
       for y := y + 2 while y×y ≤ x do
       if x:y×y = x then go to A
    end;
```

Først øges x med 2 og hjælpevariablen y sættes til 1. Derefter udføres for-sætningen for y = 3, 5, 7, o.s.v. indtil y↑2 > x. Selve indholdet af for-sætningen bestaar blot i en undersøgelse af om y gaar op i x, d.v.s. om x:y×y = x. Hvis dette er tilfældet, er x ikke et primtal og vi gaar tilbage til A og tager næste værdi af x. Hvis maskinen derimod naar igennem for-sætningen, saaledes at y↑2 > x og intet af de tidligere y-er er gaaet op i x, da er x et primtal, og proceduren er slut. Bemærk, at betingelsen x:y×y = x i GIER ALGOL 4 ogsaa kan skrives som x mod y = 0.

7.6. Brug af Navne.

Vi har hidtil døbt vore variable, procedurer, etiketter, o.s.v. med mere eller mindre vilkaarlige navne uden at give regler for, hvordan disse maa dannes. Reglen er den, at et navn skal opbygges af bogstaver og even-

tuelt tal, men at det første skal være et bogstav. Andre tegn som bindestreg, punktum o.s.v. maa ikke forekomme. Baade store og smaa bogstaver er tilladt. Eksempler paa tilladte navne:

```
P
A17
PRIM1
Temperatur
```

Og paa ikke-tilladte:

```
Runge-Kutta
22a
Mr.X
```

Mellemrum i navne ignoreres. Maskinen vil altsaa opfatte:

```
TEMP17
```

og

```
TEM  P17
```

som navnet paa den samme variable.

I GIER ALGOL kan bruges vilkaarligt lange navne, og det samlede antal forskellige, deklarerede navne (+ antallet af indbyggede standardprocedurenavne) maa ikke overstige 512. Hvis et navn indeholder over 6 tegn, bruger GIER ALGOL-oversætteren to celler til at gemme navnet under oversættelsen, og det er antallet af celler til at gemme navne i, som ikke maa overskride 512.

I det officielle ALGOL er det tilladt at bruge ikke blot almindelige navne, men ogsaa heltal som etiketter:

```
117:    A := B + C;
121:    Q := 0;
..................
```

I GIER ALGOL maa man ikke bruge heltal som etiketter, kun navne. Heltalsetiketter kan nemlig føre til en vis usikkerhed under oversættelsen.

Hvis vi deklarerer proceduren:

```
procedure P(A, B, Q);
real A, B;
procedure Q;
begin
   ........
   ........
   Q(3);
   ........
end P;
```

kan man ikke vide, om 3-tallet i sætningen: Q(3) er et tal eller en etikette. Dette vil afhænge af, hvad den formelle procedure Q bliver erstattet af i det aktuelle kald.

De fleste oversætterprogrammer tillader ikke heltalsetiketter.

7.7. Case-konstruktioner.

I det foregaaende er der flere gange nævnt eksempler paa finesser i GIER ALGOL 4, som ikke er officielt ALGOL. En meget vigtig af disse nydannelser er den saakaldte case-konstruktion, som findes i to former, svarende til henholdsvis udtryk og sætninger.

Et caseudtryk kan se saaledes ud:

```
a := case k of (P, Q, R + S×T);
```

Det er højresiden, som er det egentlige caseudtryk, og det kan defineres ved et almindeligt ALGOL-udtryk:

```
a := if k = 1 then P else
     if k = 2 then Q else
     if k = 3 then R + S×T else ALARM;
```

Caseudtrykket er en generalisering af den normale if-then-else konstruktion, hvor man ved at undersøge talværdien af den styrende heltals-

variable (her k) straks faar fat i det rigtige element i listen:

(P, Q, R + S×T).

Hvis k er mindre end 1 eller større end antallet af elementer i listen (her 3) faas en alarmudskrift under kørslen.

En casesætning kan se saaledes ud:

```
case k of
begin
   A := B + C;
   Q := R×S;
   begin
      Y := T↑2;
      A := 49
   end
end case;
```

Omskrevet til korrekt ALGOL er dette ækvivalent med:

```
if k = 1 then A := B + C
else
if k = 2 then Q := R×S
else
if k = 3 then
begin
   Y := T↑2;
   A := 49
end
else ALARM;
```

Her faas ogsaa fejludskrift for k < 1 eller k > 3. Bemærk, at det første case har altid nummer 1.

Det anbefales at bruge case-konstruktioner i GIER ALGOL 4, fordi de er hurtigere end if-then-konstruktioner.

8. INDLÆSNING OG TRYKNING

I det officielle ALGOL findes ingen vedtægter for, hvorledes man nedskriver indlæsning af talmateriale og trykning af resultater. Det er overladt til de enkelte forfattere af ALGOL-oversætterprogrammer at indbygge de nødvendige standardprocedurer i oversætteren. Flere af disse procedurer for GIER ALGOL har været omtalt tidligere, men bliver nu defineret nøjere.

8.1. Læsning af Tal.

Allerede i programmet paa side 25 saa vi en anvendelse af læseproceduren: read real:

```
HØJDE := read real;
DIAMETER := read real;
```

Naar maskinen møder disse sætninger i det oversatte program, vil den læse de to næste tal paa inputstrimlen og gemme det første tal i cellen for den variable: HØJDE og det næste tal i cellen for DIAMETER.

Proceduren read real er beregnet til læsning af eet tal ad gangen. Der findes i GIER ALGOL 4 en anden procedure:

```
read general(          );
```

til læsning af arrays, men denne kan normalt ikke anbefales. I stedet bør man bruge een eller flere for-sætninger, f.eks.:

```
begin
   integer i, j;
   array A[1:2, 1:3];
   select(8);
   for i := 1 step 1 until 2 do
   for j := 1 step 1 until 3 do
   A[i,j] := read real;
```

```
    ........
    ........
end;
```

Vi skal nu se paa formen af de tal, som skal staa paa inputstrimlen. I programmet:

```
begin
    real P, Q, R;
    select(8);
    P := read real;
    Q := read real;
    R := read real;
    ........
    ........
end;
```

kan udskriften af inputstrimlen f.eks. se saaledes ud:

$359.17, \quad -318, \quad 1.24_{10}-5,$

Tallene er opbygget af cifrene fra 0 til 9 samt hjælpetegnene:

Plus:	+
Minus:	-
Punktum:	.
10-potens:	$_{10}$

Proceduren: read real udfører den nødvendige analyse af hvert tegn paa hulstrimlen, og kan derved afgøre, hvornaar den har faaet den information, som svarer til de tre tal, P, Q og R. Vi skal nedenfor se et eksempel paa, hvorledes denne analyse ogsaa kan skrives i ALGOL, men i denne procedure sker analysen naturligvis i et stykke program, skrevet i maskinsprog, for at det kan gaa saa hurtigt som muligt.

I eksemplet ovenfor blev hvert tal afsluttet med et komma. Man kan ogsaa bruge andre synlige tegn som afslutningstegn, f.eks. semikolon. Vognretur er ogsaa terminator. Saadan er reglerne i GIER ALGOL 4. Hos Haldor Topsøe har vi i de tidligere oversætterprogrammer for ALGOL II og III foretaget saadanne ændringer, at tegnet SPACE (mellemrum) var tilstrækkeligt til talafslutning. Det har dog været for besværligt at opretholde denne særlige ordning, hvorfor den nu er forladt.

Ved indlæsning af variable af typen integer, f.eks. i programmet:

```
begin
   integer N;
   select(8);
   N := read integer;
   ........
   ........
end;
```

kan man bruge proceduren: read integer, som er væsentlig hurtigere end read real. Hvis man skriver:

```
N := read real;
```

sker der automatisk afrunding og omregning, idet det indlæste først opfattes som real. Ved meget store værdier af N faar man ikke den fulde nøjagtighed af N paa denne maade, men bortset herfra kan metoden godt bruges.

I det foregaaende har vi normalt regnet med input fra strimmellæseren og output paa linieskriveren. Denne kombination vælges med sætningen:

```
select(8);
```

Ønsker man input fra den koblede skrivemaskine og stadig output paa linieskriveren, maa man vælge:

```
select(9);
```

8.2. Læsning af Tegn.

Vi saa, at ved indlæsning af tal skal læseproceduren udføre en analyse af, hvad der staar paa strimlen for at erkende, hvor tallet begynder og ender, hvad der er taldel og hvad potens, o.s.v. Maskinen læser altsaa adskillige tal- og tegn-symboler og benytter den een gang for alle fastlagte analysemetode af tallet.

I visse tilfælde har man brug for en anden slags analyse af det indlæste tal, en simplere eller en mere kompliceret. Dette maa programmøren selv kode i ALGOL og han har derfor brug for en procedure, som kun læser eet tegn fra strimlen. Han maa ogsaa sætte sig ind i hvorledes de enkelte tal og tegn gengives paa strimlen som bestemte hulrækker. Dette er vist paa nedenstaaende tabel.

Det bemærkes, at tabellen kun giver udseendet af de tegn, som en Flexowriter kan læse og skrive. Linieskriveren har yderligere nogle specielle tegn, som dog kun sjældent anvendes. Se herom beskrivelsen til GIER Program DEMON-8, som illustrerer specielle anvendelser af linieskriveren.

Tabel over Hulsymboler

<table>
<tr><th>Strimmel</th><th>Talværdi</th><th>LC</th><th>UC</th></tr>
<tr><td>|---o-.---|</td><td>0</td><td colspan="2">SPACE</td></tr>
<tr><td>|-----.--o|</td><td>1</td><td>1</td><td>∨</td></tr>
<tr><td>|-----.-o-|</td><td>2</td><td>2</td><td>×</td></tr>
<tr><td>|---o-.-oo|</td><td>3</td><td>3</td><td>/</td></tr>
<tr><td>|-----.o--|</td><td>4</td><td>4</td><td>=</td></tr>
<tr><td>|---o-.o-o|</td><td>5</td><td>5</td><td>;</td></tr>
<tr><td>|---o-.oo-|</td><td>6</td><td>6</td><td>[</td></tr>
<tr><td>|-----.ooo|</td><td>7</td><td>7</td><td>]</td></tr>
<tr><td>|----o.---|</td><td>8</td><td>8</td><td>(</td></tr>
<tr><td>|---oo.--o|</td><td>9</td><td>9</td><td>)</td></tr>
<tr><td>|---oo.-o-|</td><td>10</td><td colspan="2">Bruges ikke</td></tr>
<tr><td>|----o.-oo|</td><td>11</td><td colspan="2">STOP CODE</td></tr>
<tr><td>|---oo.o--|</td><td>12</td><td colspan="2">END CODE</td></tr>
<tr><td>|----o.o-o|</td><td>13</td><td colspan="2">Bolle-aa</td></tr>
<tr><td>|----o.oo-|</td><td>14</td><td>_</td><td>|</td></tr>
<tr><td>|---oo.ooo|</td><td>15</td><td colspan="2">Bruges ikke</td></tr>
<tr><td>|--o--.---|</td><td>16</td><td>0</td><td>∧</td></tr>
<tr><td>|--oo-.--o|</td><td>17</td><td><</td><td>></td></tr>
<tr><td>|--oo-.-o-|</td><td>18</td><td>s</td><td>S</td></tr>
<tr><td>|--o--.-oo|</td><td>19</td><td>t</td><td>T</td></tr>
<tr><td>|--oo-.o--|</td><td>20</td><td>u</td><td>U</td></tr>
<tr><td>|--o--.o-o|</td><td>21</td><td>v</td><td>V</td></tr>
<tr><td>|--o--.oo-|</td><td>22</td><td>w</td><td>W</td></tr>
<tr><td>|--oo-.ooo|</td><td>23</td><td>x</td><td>X</td></tr>
<tr><td>|--ooo.---|</td><td>24</td><td>y</td><td>Y</td></tr>
<tr><td>|--o-o.--o|</td><td>25</td><td>z</td><td>Z</td></tr>
<tr><td>|--o-o.-o-|</td><td>26</td><td colspan="2">Bruges ikke</td></tr>
<tr><td>|--ooo.-oo|</td><td>27</td><td>,</td><td>$_{10}$</td></tr>
<tr><td>|--o-o.o--|</td><td>28</td><td colspan="2">CLEAR CODE</td></tr>
<tr><td>|--ooo.o-o|</td><td>29</td><td colspan="2">Rødt skift</td></tr>
<tr><td>|--ooo.oo-|</td><td>30</td><td colspan="2">TAB</td></tr>
<tr><td>|--o-o.ooo|</td><td>31</td><td colspan="2">PUNCH OFF</td></tr>
</table>

Strimmel	Talværdi	LC	UC
|-o---.---|	32	-	+
|-o-o-.--o|	33	j	J
|-o-o-.-o-|	34	k	K
|-o---.-oo|	35	l	L
|-o-o-.o--|	36	m	M
|-o---.o-o|	37	n	N
|-o---.oo-|	38	o	O
|-o-o-.ooo|	39	p	P
|-o-oo.---|	40	q	Q
|-o--o.--o|	41	r	R
|-o--o.-o-|	42	Bruges ikke	
|-o-oo.-oo|	43	ø	Ø
|-o--o.o--|	44	PUNCH ON	
|-o-oo.o-o|	45	Bruges ikke	
|-o-oo.oo-|	46	Bruges ikke	
|-o--o.ooo|	47	Bruges ikke	
|-ooo-.---|	48	æ	Æ
|-oo--.--o|	49	a	A
|-oo--.-o-|	50	b	B
|-ooo-.-oo|	51	c	C
|-oo--.o--|	52	d	D
|-ooo-.o-o|	53	e	E
|-ooo-.oo-|	54	f	F
|-oo--.ooo|	55	g	G
|-oo-o.---|	56	h	H
|-oooo.--o|	57	i	I
|-oooo.-o-|	58	LOWER CASE	
|-oo-o.-oo|	59	.	:
|-oooo.o--|	60	UPPER CASE	
|-oo-o.o-o|	61	SUM CODE	
|-oo-o.oo-|	62	Sort skift	
|-oooo.ooo|	63	TAPE FEED	
|o----.---|	64	CAR RET	

Der er plads til 8 hulrækker paa tværs af strimlen. Hertil kommer en ekstra række med smaa fremføringshuller. Rækken yderst til venstre er altid forbeholdt tegnet for: vogn tilbage og ny linie (CAR RET) og række nr. 4 fra venstre bruges til paritetskontrol, d.v.s. antallet af huller paa tværs af en række skal være ulige, og for de tegn, hvor det er lige, anbringes automatisk et ekstra paritetshul. Herved kan fejl ved hulningen ofte opdages.

De seks tilbageværende rækker giver ialt 2↑6 = 64 kombinationer. I virkeligheden er de fleste tegn tvetydige, svarende til at skrivemaskinen kan staa i lower case (smaa bogstaver) eller upper case (store bogstaver), altsaa ialt 128 muligheder. Visse tegn er dog de samme i begge cases, saaledes at antallet er noget mindre.

Nogle af symbolerne i tabellen kræver nærmere forklaring.

Nr. 0. SPACE. Dette er et mellemrum. Tegnet har samme effekt i lower case (LC) som i upper case (UC).

Nr. 11. STOP CODE. Tegn til Flexowriteren om at stoppe. Har ingen effekt paa linieskriveren.

Nr. 12. END CODE. Findes dette tegn paa et vilkaarligt sted i et ALGOL-program, standser regnemaskinen, naar den møder tegnet og skriver ordet:

pause

paa den tilkoblede skrivemaskine. Maskinen starter igen ved nedtrykning af en vilkaarlig tast paa skrivemaskinen.

Nr. 13. Tegnet for bolle-aa findes ikke paa Flexowriteren, men kun paa den skrivemaskine, som er tilkoblet regnemaskinen, samt paa linieskriveren.

Nr. 14. Tegnet for _ og | flytter ikke valsen. Bruges til sammensatte tegn: ∧⃒, (⃒,)⃒, :̲, etc. samt de understregede ALGOL-gloser.

Nr. 28. CLEAR CODE. Naar regnemaskinen møder dette tegn paa selve programstrimlen til et ALGOL-program, nulstilles en intern sumcelle. Talværdien af de efterfølgende symboler paa strimlen summeres i denne celle.

Nr. 29. Rødt skift. Skifter farvebaandet paa den tilkoblede skrivemaskine til rødt. Har ingen effekt paa Flexowriteren eller linieskriveren.

Nr. 30. TAB. Flytter valsen frem til næste tabulatorstop. Bør normalt ikke benyttes, da linieskriverens tabulatorfunktion er anderledes.

Nr. 31. PUNCH OFF. Naar Flexowriteren automatisk reperforerer en hulstrimmel, kan den indstilles til at overspringe, hvad der staar imellem PUNCH OFF og PUNCH ON (Nr. 44). Oversætteren til GIER ALGOL 4 kan bringes til paa samme maade at overspringe alt mellem PUNCH OFF og PUNCH ON, men dette benyttes normalt ikke, saaledes at de to tegn er blinde.

Nr. 44. PUNCH ON. (Se under Nr. 31).

Nr. 58. LOWER CASE. Skifter Flexowriteren, linieskriveren eller skrivemaskinen til udskrift i lower case. Ens i LC og UC.

Nr. 60. UPPER CASE. Skifter til upper case.

Nr. 61. SUM CODE. Naar regnemaskinen møder dette tegn paa ALGOL-programstrimlen, læser den ogsaa det næste tegn fra strimlen, og dette tegn skal være lig med indholdet af den interne sumcelle, som er nævnt under Nr. 28. Passer checksummen ikke, udskrives ordet:

sum

paa den tilkoblede skrivemaskine. Dette indicerer en hullefejl eller en læsefejl. Maskinen starter igen ved nedtrykning af en vilkaarlig tast paa skrivemaskinen.

Nr. 62. Sort skift. Analog med nr. 29, blot skiftes til sort.

Nr. 63. TAPE FEED. Der bruges gerne ca. 50 af disse tegn i begyndelsen og i slutningen af en strimmel, for at give en bekvem indledning og afslutning paa strimlen.

Nr. 64. CAR RET. Tegn for vogn tilbage og ny linie.

De hulkombinationer, der svarer til 65 - 127, har ingen effekt paa Flexowriteren og skrivemaskinen, og bruges derfor normalt ikke som output. For linieskriveren har enkelte af disse tegn betydning, f.eks. nr. 72, der giver TOP OF FORM, d.v.s. at der rykkes frem til øverst paa næste side. Nr. 80 giver trykning af en linie (som nr. 64), men uden at der rykkes frem til næste linie. Herved kan man f.eks. trykke accenter og tegnene _ og | ved at trykke linien a to gange.

I GIER ALGOL 4 findes kun een metode til indlæsning af de enkelte symboler fra strimlen. Det er proceduren lyn:

```
begin
   integer n;
   ........
   n := lyn;
   ........
```

Dette er en integer procedure uden formelle parametre. Talværdien af lyn er den, som er angivet i tabellen side 162-163, uden hensyn til om tegnet er i lower case eller i upper case.

Hvis vi har følgende program:

```
begin
   integer i;
   integer array B[1:10];
   select(8);
   for i := 1 step 1 until 10 do B[i] := lyn;
end;
```

og lader programmet læse en strimmel med de seks synlige tegn:

D=117;

da vil B-talsættet efter beregningen kunne indeholde:

B[1]	60	UPPER CASE
B[2]	52	D
B[3]	4	=
B[4]	58	LOWER CASE
B[5]	1	1
B[6]	1	1
B[7]	7	7
B[8]	60	UPPER CASE
B[9]	5	;

Vi ser, at case-tegnene behandles ganske som de øvrige tegn. Bemærk, at man af udskriften:

D=117;

naturligvis ikke kan se, hvor mange case-tegn der faktisk er placeret paa de steder af strimlen, hvor der skiftes case. Hulledamen kan godt have anbragt to eller tre case-tegn, hvor der kun behøves eet, ligesom der iøvrigt kan være placeret overflødige case-tegn, TAPE FEED tegn o.l. overalt paa strimlen.

Som et eksempel paa brugen af lyn viser vi nu en procedure, som udfører omtrent den samme funktion som read real, altsaa læser det næste, sammensatte tal fra strimlen. Tallet antages at kunne indeholde baade fortegn (±) og decimalpunktum men ikke 10-potens. Vi antager for simpelheds skyld, at tallet afsluttes med SPACE eller CAR RET, samt at disse ikke forekommer inden i tallet.

```
    real procedure TAL;
    begin
       boolean start, plus, punktum, lowercase;
       integer symbol, decimal;
       real RESULTAT;
       start := plus := lowercase := true;
       punktum := false;
       decimal := 0;
       RESULTAT := 0;
A:     symbol := lyn;
       if symbol = 58 ∨ symbol = 60 then
       begin
          lowercase := symbol = 58;
          go to A
       end if casetegn;
       if ¬, lowercase then symbol := symbol + 128;
       if start then
       begin comment  Vi leder først efter fortegn, punktum eller cifre;
          if symbol = 32 then
          begin
             plus := false;
             go to A
          end behandling af minus.  Plustegn undersøges ikke;
```

```
          if symbol = 59 then
          begin
             punktum := true;
             start := false;
             go to A
          end behandling af punktum;
          if symbol = 16 then
          begin
             symbol := 0;
             go to B
          end behandling af nul;
          if symbol ≥ 1 ∧ symbol ≤ 9 then
B:        begin
             start := false;
             RESULTAT := symbol;
             go to A
          end behandling af ciffer;
          go to A
       end if start
       else
       begin comment  Nu leder vi efter cifre, efter punktum, hvis det
                      ikke er kommet, eller efter afslutning;
          if symbol = 16 then
          begin
             symbol := 0;
             go to C
          end behandling af nul;
          if symbol ≥ 1 ∧ symbol ≤ 9 then
C:        begin
             RESULTAT := RESULTAT×10 + symbol;
             if punktum then decimal := decimal + 1;
             go to A
          end if ciffer;
          if ¬, punktum ∧ symbol = 59 then
          begin
             punktum := true;
             go to A
          end if punktum;
```

```
            if symbol = 0 ∨ symbol = 128
            ∨ symbol = 64 ∨ symbol = 192 then
            begin
               if ¬, plus then RESULTAT := - RESULTAT;
               TAL := RESULTAT×10↑(-decimal);
               go to D
            end if afslutning;
            go to A
         end if not start;
D:  end TAL;
```

Proceduren opererer med de fire logiske variable, hvis værdi sættes inden indlæsningen begynder:

start: Er sand, indtil det første ciffer eller punktum er mødt.
plus: Er sand, indtil der eventuelt mødes et minustegn.
punktum: Er falsk, indtil der eventuelt mødes et punktum.
lowercase: Er sand i lower case situationen og falsk i upper case.

Endvidere bruges to heltalsvariable:

symbol: Heri gemmes det sidst indlæste tegn.
decimal: Antallet af decimaler i tallet. Sættes først til nul.

Endelig gemmes talværdien af det hidtil indlæste i den real variable: RESULTAT, som ogsaa nulstilles først.

Selve indlæsningen foregaar ved hjælp af sætningen:

```
A:  symbol := lyn;
```

som læser det næste symbol fra strimlen. Det indlæste tegn gemmes som et integer i cellen: symbol. Til at holde kontrol med case-situationen bruger vi den logiske variable, lowercase. Hver gang tegnene 58 eller 60 mødes, sættes lowercase igen. Det er normal praksis, at man derefter internt korrigerer talværdien af det indlæste symbol ved at addere 128, hvis tegnet staar i upper case. Paa denne maade bliver tegnenes værdi entydig i den interne repræsentation.

Hvis vi er i startsituationen, altsaa endnu ikke har faaet noget ciffer eller punktum, vil programmet nu lede efter minustegn ved hjælp af betingelsen:

if symbol = 32 **then**

og efter punktum med:

if symbol = 59 **then**

Cifrene 1 til 9 har talværdierne fra 1 til 9, men cifret nul har værdien 16 i den anvendte Flexowriter-kode. Dette er naturligvis lavet, for at man kan faa en effektiv checksumkontrol, ogsaa paa nullerne. Vi skal derfor baade undersøge om symbol = 16 og om $1 \leq$ symbol ≤ 9. Naar det første ciffer er fundet, sættes start til falsk, og cifferet gemmes i resultatcellen.

Naar vi har passeret startsituationen, er den vigtigste analyse naturligvis for nye cifre. Hver gang et nyt ciffer er kommet ind i symbol-cellen, udføres sætningen:

RESULTAT := RESULTAT×10 + symbol;

Talafslutningen kan være:

0:	SPACE	i lower case
128:	-	- upper -
64:	CAR RET	- lower -
192:	-	- upper -

Naar et af disse symboler mødes, skifter vi fortegn paa RESULTAT-cellen, hvis der er mødt et minus, og vi korrigerer for decimalerne ved at dividere med 10↓decimal.

Denne TAL-procedure er naturligvis langt fra at være optimalt programmeret. F.eks. udføres ciffer-undersøgelsen to steder, hvilket kunne have været gjort med en hjælpeprocedure. Man skal dog være forsigtig med at bruge alt for mange procedurekald indeni en læseprocedure, da det let kan tage for lang tid.

Naar man selv laver programmer eller procedurer til symbolindlæsning af tal eller tekst eller blandinger heraf, savner man sommetider muligheden for at kunne køre strimmellæseren baglæns, bare en enkelt række. Hvis man f.eks. har blandinger af tekst og tal i vilkaarlig rækkefølge:

ABC, 317, PQ, 239, 129, MML, 98,

kan man lave en særlig procedure, som indlæser teksten og gør et eller andet ved den, men det er ikke strengt nødvendigt at lave en speciel procedure til indlæsning af tallene. Vi antager, at vi staar midt imellem ABC og 317 og læser eet symbol ad gangen med:

```
symbol := lyn;
```

Saa snart symbol antager værdien af et bogstav, kan man udføre sin specielle tekstindlæsning, men naar symbol svarer til et ciffer, altsaa her 3-tallet, ville det være lidt lettere, hvis man blot kunne faa hele tallet ind med f.eks. read real proceduren:

```
j := read real;
```

men det gaar ikke, da vi jo har tabt 3-tallet og j derfor kun bliver lig 17.

Det var derfor fristende at kunne køre læseapparatet tilbage, saaledes at 3-tallet kom frem igen, men det kan man ikke. Derimod kan maskinen simulere, at strimlen køres een plads baglæns. Og dette gør maskinen i virkeligheden altid, nemlig paa følgende maade. Der findes i GIER ALGOL 4 en standardvariabel af typen integer:

```
integer char;
```

som ikke skal deklareres af brugeren, men som altid er deklareret automatisk i ethvert ALGOL-program. Denne variable har den ganske specielle funktion, at efter ethvert kald af de sammensatte inputprocedurer: read real og read integer vil talværdien af det sidst læste tegn være gemt i den variable: char. Hvis tegnet var i upper case, er der adderet 128 til char, saaledes at værdien er entydig. Endvidere er der den meget væsentlige finesse, at read real og read integer altid begynder med at tage det

første tegn, de skal læse, fra char, ikke fra det faktisk tilsluttede ydre medium, som først benyttes ved læsning af tegn nr. 2 og de følgende. Hvis vi læser følgende talrække:

23.49, -1207; 117a

med programmet:

```
begin
   real A, B, C;
   select(8);
   A := read real;
   B := read real;
   C := read real;
   ......
end;
```

vil talværdien af char efter indlæsning af 23.49 til cellen for A være lig med 27 (talværdien for komma). Naar B derefter indlæses med: B := read real bliver det første tegn, som læses, igen 27, derefter mellemrummene før -1207, og saa selve tallet. Efter læsning af B er char lig med 5 + 128 = 133, idet semikolon har talværdien 5, hvortil skal lægges 128 for upper case. Efter læsning af 117 er char lig med 49 (værdien af a).

8.3. Trykning af Tal.

I programmet side 25 var vist et eksempel paa trykning af tal:

```
write(⟨dddd.ddd⟩, HØJDE, DIAMETER, RUMFANG);
```

Naar maskinen møder denne sætning i det oversatte program, vil den perforere en hulstrimmel med den aktuelle talværdi af de tre variable: HØJDE,

DIAMETER og RUMFANG, eller skrive tallene paa skrivemaskinen eller linie-skriveren, f.eks. saaledes:

3.100 2.900 20.476

Proceduren: write arbejder med en liste af aktuelle parametre, hvoraf den første altid skal være et saakaldt layout, her:

{dddd.ddd}

Layoutet er skrevet paa en maade, saaledes at det umiddelbart gengiver tallets form. Her har vi maksimalt fire tal før kommaet: dddd og tre decimaler: .ddd

I layoutet {dddd.ddd} giver det samlede antal d-er det ønskede antal betydende cifre. Tallet: 1234.567 trykkes altsaa med 7 betydende cifre. Er dette for meget, kan man erstatte nogle af d-erne med nuller (bagfra):

{dddd.ddd}	giver	1234.567
{dddd.dd0}	-	1234.57
{dddd.d00}	-	1234.6
{dddd.000}	-	1235
{ddd0.000}	-	1230
o.s.v.		

Bemærk, at tallene altid afrundes korrekt. De anslag, som ikke bliver udnyttet til ciffertrykning eller trykning af decimalpunktum, fyldes automatisk op med SPACEs. Efter 1234.6 anbringes saaledes to SPACEs.

Ved trykning af heltal udelades punktum:

{dddd}

De her viste eksempler paa forskellige layouts kan kun bruges for positive tal. Kan tallene være negative, har man valget mellem tre skrivemaader:

layout:	⟨-ddd.d⟩	⟨±ddd.d⟩	⟨+ddd.d⟩
Trykning:	123.4	+123.4	+123.4
	-12.3	- 12.3	-12.3
	1.2	+ 1.2	+1.2
	-0.1	- 0.1	-0.1
	0.0	+ 0.0	+0.0

Vi anvender normalt den første af disse. Reglerne er:

1. Minus giver trykning af minus umiddelbart foran negative tal og af SPACE foran positive tal.

2. Plus-minus giver trykning af plus foran positive og minus foran negative tal. Tegnet trykkes som det allerførste anslag paa de pladser, der staar til raadighed for trykningen.

3. Plus trykker ligeledes begge tegn men umiddelbart foran første ciffer.

Trykning med 10-potens kan ogsaa specificeres. Følgende layout:

⟨-d.ddd$_{10}$-dd⟩

giver trykning som:

1.234$_{10}$-12
-2.345 $_{10}$-3
3.456 $_{10}$ 4
-4.567

Man kan ogsaa faa begrænset antallet af mulige potenser, saaledes at der f.eks. kun forekommer: $_{10}$-6, $_{10}$-3, $_{10}$+3, $_{10}$+6, o.s.v. Eksempel:

{-ddd.d00$_{10}$+d}
  1.235$_{10}$-3
 12.35 $_{10}$-3
123.5 $_{10}$-3
  1.235
 12.35
123.5
  1.235$_{10}$+3
 12.35 $_{10}$+3

Reglen er: Hvis der er N nuller i layoutet (her 2), bliver eksponenten et multiplum af N + 1.

Det er tilladt at anbringe SPACES i begyndelsen af et layout, hvilket er særligt bekvemt for at faa den nødvendige afstand mellem søjler af tal. Disse SPACE var ikke tilladt i GIER ALGOL II og III. Eksempel:

write({ ddd.dd}, A, B);

giver trykning af formen:

123.45 987.65

Endelig kan man faa anbragt ekstra SPACEs indeni tallet:

{-ddd ddd.ddd ddd}
 123 456.789 123

Der kan højst disponeres over 15 pladser før decimalpunktummet og 15 efter. De nøjagtige regler herfor findes i Naur (1967).

Hvis man specificerer trykning af et tal, som er for stort til at kunne trykkes med det foreliggende layout, bliver der automatisk paaført tallet en 10-potens (eller antallet af cifre i denne bliver forhøjet), saaledes at man faar trykt den rigtige værdi af tallet. De saaledes trykte tal vil fylde mere, end hvad der svarer til det originale layout. Som eksempel kan vi tage programmet:

```
begin
   integer N;
   select(8);
   writecr;
   for N := 71 step 1 until 130 do
   begin
      write({dd}, N);
      writetext({<  });
      if N mod 10 = 0 then writecr
   end
end;
```

Ved kørsel heraf faar man nedenstaaende resultatudskrift:

71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 $10_{10}1$
$10_{10}1$ $10_{10}1$ $10_{10}1$ $10_{10}1$ $11_{10}1$ $11_{10}1$ $11_{10}1$ $11_{10}1$ $11_{10}1$ $11_{10}1$
$11_{10}1$ $11_{10}1$ $11_{10}1$ $11_{10}1$ $12_{10}1$ $12_{10}1$ $12_{10}1$ $12_{10}1$ $12_{10}1$ $12_{10}1$
$12_{10}1$ $12_{10}1$ $12_{10}1$ $12_{10}1$ $13_{10}1$ $13_{10}1$ $13_{10}1$ $13_{10}1$ $13_{10}1$ $13_{10}1$

Naar N bliver større end 99, sætter maskinen en 10-potens paa, men da der kun er to betydende cifre, gaar enerne i tallet naturligvis tabt.

Et layout kan iøvrigt godt skrives som et udtryk med en betingelse:

```
write(if A < 10 then {d} else {dd}, A);
```

Layoutet er her:

```
if A < 10 then {d} else {dd}
```

og man kan indbygge flere if-konstruktioner, hvis man ønsker det.

Det layout, der anbringes som første parameter i write-procedurekaldet, gælder for trykningen af alle de følgende parametre i parentesen:

```
write({-dd.dd}, A, B, C, D, E);
```

altsaa for alle fem variable: A - E. Hvis en enkelt af disse skal trykkes med et specielt layout, f.eks. C som {-d.dd$_{10}$-dd}, maa man skrive:

```
write(⟨-dd.dd⟩, A, B);
write(⟨-d.dd₁₀-dd⟩, C);
write(⟨-dd.dd⟩, D, E);
```

Har man et stort program, i hvilket der skal trykkes mange variable, men med et beskedent antal forskellige layouts, savner man muligheden for at kunne kalde et layout for noget, at give det et navn, som f.eks. en variabel. Dette er ikke tilladt i det officielle ALGOL. Men i GIER ALGOL 4 kan vi gøre det. Vi kan f.eks. skrive:

```
A1 := ⟨-dd.dd⟩;
A2 := ⟨-d.dd₁₀-dd⟩;
write(A1, A, B);
write(A2, C);
write(A1, D, E);
```

med nøjagtig samme effekt som programmet ovenfor.

I eksemplet paa side 65 saa vi, at man godt kan anbringe regneudtryk som parametre i write-proceduren i stedet for blot navnene paa de variable, der skal trykkes:

```
write(⟨dddddd.dd⟩, MCO, 100×mCO);
```

Saadanne regneudtryk kan være vilkaarligt indviklede og f.eks. indeholde kald af procedurefunktioner (deklarerede eller standard-funktioner):

```
write(⟨-d.ddd₁₀-dd⟩, sqrt(SUM(M[j], j, 1, 10)));
```

hvor SUM er tænkt som procedurefunktionen deklareret side 95.

Der findes i GIER ALGOL 4 en særlig hurtig procedure til trykning af heltal:

```
write integer(⟨-dddd⟩, A);
```

hvor layoutets form er mere begrænset end for write, og hvor der kun maa være een parameter efter layoutet.

Hvis det første d i layoutet for write eller write integer erstattes med et p, bliver der fyldt op med nuller før decimalpunktummet:

```
write({pdd}, A);
```

giver trykningen:

```
007
```

for A = 7.

Normalt output paa linieskriveren faas ved i programmets begyndelse at vælge:

```
select(8);
```

Output paa skrivemaskinen faas ved select(16) og paa perforatoren med select(32). Ved passende addition af de tre tal 8, 16 og 32 kan man faa samtidigt output paa flere medier.

8.4. Trykning af Tekst.

Eksempler paa teksttrykning har vi set i programmet side 46:

```
if element < 0 then writetext({< Negativ});
```

Proceduren har navnet: writetext, og den aktuelle parameter er en saakaldt tekststreng anbragt imellem tegnene:

```
{<   }
```

Bemærk, at vi for layoutstrenge benytter:

```
{   }
```

Der er ingen logisk forskel paa de to slags strenge, det er blot for at lette oversætterprogrammets arbejde, at man gerne vil skelne imellem dem.

Der kan kun bruges een parameter i writetext. Men parametren kan indeholde betingelser:

```
writetext(if A then {<stor} else {<lille});
```

eller case-konstruktioner:

```
writetext(case j of
        {<Brint       },
        {<Vand        },
        {<Kvælstof    },
        .
        .
        .
        .
        .
        .
        .
        {<Metan       });
```

som er ækvivalent med sætningen side 81.

8.5. Trykning af Tegn.

I sjældnere tilfælde kan man ikke klare sig med standardprocedurerne til trykning af tal og tekst, men ønsker selv at opbygge lignende procedurer efter andre retningslinier. Man har da brug for en procedure, som kan trykke eet symbol ad gangen, ligesom vi med proceduren lyn kunne læse et symbol ad gangen.

Hertil bruges proceduren writechar:

```
writechar(n);
```

Den nødvendige værdi af n fremgaar af tabellen side 162-163. Her bruger vi dog ikke det kneb at addere 128 til n for at vise at vi ønsker tegnet i upper case. Alle casetegn maa programmeres specielt med:

```
            writechar(58);
eller       writechar(60);
```

Ønsker vi saaledes at trykke kombinationen:

```
D=117;
```

med brug af writechar, kan dette gøres saaledes:

```
writechar(60);
writechar(52);
writechar( 4);
writechar(58);
writechar( 1);
writechar( 1);
writechar( 7);
writechar(60);
writechar( 5);
```

Som en mere interessant anvendelse af writechar giver vi nedenfor et program, der fremstiller tabellen side 162-163:

```
begin
   integer i, j, k, m, n, s;
   integer array HUL[1:7];
   procedure HOVED;
   writetext(⟨<
       Strimmel          Talværdi       LC          UC

>);
```

```
select(8);
writecr;
writechar(58);
for i := 0 step 1 until 64 do
begin
   if i = 0 ∨ i = 32 then HOVED else writecr;
   writetext(⟨<           |⟩);
   s := 0;
   k := i;
   for j := 1 step 1 until 7 do
   begin
      m := k:2×2;
      n := k - m;
      s := s + n;
      HUL[j] := n;
      k := m:2
   end for j;
   for j := 7 step -1 until 1 do
   begin
      writechar(if HUL[j] ≠ 0 then 38 else 32);
      if j = 5 then writechar(if s mod 2 = 0 then 38 else  32);
      if j = 4 then writechar(59)
   end for j;
   writetext(⟨<|           ⟩);
   write(⟨dd⟩, i);
   writetext(⟨<           ⟩);
   if i = 0 then writetext(⟨<   SPACE⟩)
   else
   if i = 10 ∨ i = 15 ∨ i = 26 ∨ i = 42 ∨ i = 45 ∨ i = 46 ∨ i = 47
   then writetext(⟨<Bruges ikke⟩)
   else
   if i = 11 then writetext(⟨<  STOP CODE⟩)
   else
```

```
            if i = 12 then writetext({<   END CODE})
            else
            if i = 13 then writetext({<   Bolle-aa})
            else
            if i = 28 then writetext({< CLEAR CODE})
            else
            if i = 29 then writetext({< Rødt skift})
            else
            if i = 30 then writetext({<    TAB})
            else
            if i = 31 then writetext({<  PUNCH OFF})
            else
            if i = 44 then writetext({<   PUNCH ON})
            else
            if i = 58 then writetext({< LOWER CASE})
            else
            if i = 60 then writetext({< UPPER CASE})
            else
            if i = 61 then writetext({<   SUM CODE})
            else
            if i = 62 then writetext({< Sort skift})
            else
            if i = 63 then writetext({<  TAPE FEED})
            else
            if i = 64 then writetext({<   CAR RET})
            else
            begin
               writechar(i);
               writetext({<          });
               if i = 14 then writetext({< });
               writechar(60);
               writechar(i);
               writechar(58)
            end symbol;
            if i = 31 then for j := 1 step 1 until 11 do
            writecr
         end for i
      end program;
```

Programmet bestaar af en ydre for-sætning:

for i := 0 step 1 until 64 do

som styrer trykningen af de 65 linier. For hver linie er der først een for-sætning:

for j := 1 step 1 until 7 do

som beregner, hvor der er huller. Trykningen af strimlens udseende sker med en anden for-sætning:

for j := 7 step -1 until 1 do

hvor paritetshullet indsættes med betingelsen:

if j = 5 then

og fremføringshullet med:

if j = 4 then

Trykningen af de mange specialtekster kunne godt have været gjort mere kortfattet ved hjælp af procedurer.

9. REFERENCER

Andersen, Chr.: ALGOL, DASK-ALGOL og GIER-ALGOL, Akademisk Forlag, København (1963).

Backus, J.W., et al.: Report on the Algorithmic Language ALGOL 60 (ed. P. Naur), Regnecentralen (1960), og rettet optryk (1964).

Backus, J.W., et al.: Revised Report on the Algorithmic Language ALGOL 60, Comm. ACM, 6, No. 1, 1 - 17 (1963).

Hougen, O. A., and Watson, K. M.: Chemical Process Principles, III, Kinetics and Catalysis, New York (1947).

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

Lauesen, S. et al.: A Manual of HELP3. Regnecentralen (1967).

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

Vilstrup, H.: Lærebog i GIER-ALGOL. Akademisk Forlag, København (1963).

10. STIKORDSREGISTER

abs, 27
adresse, 12, 16, 128
afrunding, 21, 46, 144, 160, 173
ALGOL-rapport, 9
arctan, 27
array, 28, 41, 92, 158

begin, 24, 120
betingelse, 13, 43, 73, 179
blok, 119
boolean, 131

CAR RET, 165
case, 165, 169
case, 156, 179
celle, 12
char, 171
checksum, 165
CLEAR CODE, 164
comment, 52
copy, 109
cos, 27,
cylinderrumfang, 24, 82

deklaration, 24, 29, 36 41, 84, 120, 139
dimension, 30, 42
division, 21, 58, 145
do, 33

eller(∨), 48, 138
else, 45, 52
end, 24, 52, 120
END CODE, 164
ENT1, 101
entalpi, 36, 100
entier, 27, 58, 145
etikette, se label
exp, 27

fakultet, 117
false, 132
finis, 110
Flexowriter, 16
flydende komma, 142
formal, 146
for-sætning, 33, 79, 148

global, 103
go to, 51, 122
grundoperation, 12
grænser, 29, 92, 123

HELP3, 110
heltalsdivision, 58, 145
hop, se go to
hukommelse, 11
hulsymbol, 162
højreside, 19

if, 44
ikke, se nægtelse
implikation(=>), 49, 138
index, 30, 43, 127
indlæsning, 25, 158
inputspecifikation, 72, 75
integer, 25, 142, 147
iteration, 62

Jensens device, 97

kald, 84
kommentar, se comment
komponentnummer, 71

label, 49, 122, 155
lager, 12
layout, 173
lighedstegn, 18
linieskriver, 27
ln, 27
logisk variabel, se boolean
LOWER CASE, 165
lyn, 165

maskinsprog, 15
mod, 60, 137

name, 95, 146
navn, 154
nægtelse(¬,), 49, 138

og(∧), 48, 138
oversættelse, 16
own, 123

parameter, 84, 85, 107, 146

parentes, 22
paritet, 164
pause, 164
polynomium, 37
potens, 20
primtal, 154
procedure, 82
procedurefunktion, 86
programmering, 16
PUNCH OFF, 165
PUNCH ON, 165

read general, 158
read integer, 51, 160
read real, 25, 158
real, 24, 142, 147
regneudtryk, 20
rekursiv, 117
relation, 48
rækkefølge af operatorer, 21, 138

select, 25
sign, 27
sin, 27
skiftespor, se switch
SPACE, 160, 164
specifikation, 84, 92, 146
spill, 28, 142
sqrt, 27
standardfunktion, 27
step, 33, 148
STOP CODE, 164
sum, 165
SUM CODE, 165
summation, 30, 89
switch, 139
sætning, 119
sætning, sammensat, 44, 119

TAB, 165
tal, 158
talsæt, se array
tangens, 28
TAPE FEED, 165
tegn, 161, 179
tekst, 178
terminator, 160
then, 44
tipotens($_{10}$), 20, 174
totalsystem, 143
true, 132
trykfald, 67, 110
trykning, 25, 172
typekontrol, 147

until, 33, 148
UPPER CASE, 165

value, 93, 107, 146
vandgasligevægt, 61, 106
venstreside, 19
viskositet, 68, 114

while, 153
write, 25, 66, 172
writechar, 180
writecr, 25
write integer, 177
writetext, 47, 178

ækvivalens(≡), 49, 138