

PROGRAMMERING AF KEMISKE BEREGNINGER

BIND 1: ALGOL

Af

Jørgen Kjær

Haldor Topsøe, Hellerup, Danmark

Nr. 1

Copyright 1964
Haldor Topsøe, Chemical Engineers
Hellerup, Denmark

FORORD

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 formaals-tjenligt 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 uhjælpe- ligt fast, hvis man forsøger at lære dem sproget ad teoretisk vej med ord- bøger og grammatik.

Paa et senere stadium af undervisningen maa de finere detaljer 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. løs- ning 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 beregnin- ger bør udføres af kemiingeniører, som har lært ALGOL, eller af ALGOL- specialister, som har lært noget kemi eller faaet det kemiske problem for- klaret 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-procedureerne 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.

Det er hensigten, at denne bog skal være det første bind i en serie af bøger, som i alt væsentlig vil beskrive det kompleks af programmer til kemiske ingeniørberegninger, som er udarbejdet hos Haldor Topsøe. Der er foreløbigt planlagt følgende yderligere bind:

2. Numeriske Metoder.
3. Termodynamiske Beregninger.
4. Beregning af Katalytiske Konvertere.
5. Analyse af Kinetiske Data.
6. Beregning af Reformerprocesser.
7. Beregning af Ammoniakkonvertere.

Bøgerne vil eventuelt ogsaa blive udarbejdet paa engelsk.

Jeg vil gerne takke Preben Sørensen, som har afprøvet alle algoritmer i bogen, og fru Birthe Sørensen for hulning af manuskriptet.

Hellerup, Maj 1964

Jørgen Kjær

INDHOLDSFORTEGNELSE

	Side
1. INDLEDNING	7
2. DEN PROGRAMSTYREDE CIFFERREGNEMASKINE	9
2.1. Bordregnemaskinen	9
2.2. Cifferregnemaskinen	10
2.3. En vigtig forbedring	11
2.4. Betingelser	12
3. PROGRAMMERINGSSPROG	14
3.1. Maskinsprog	14
3.2. ALGOL	15
4. GRUNDBEGREBERNE I ALGOL	17
4.1. Simple sætninger	17
4.2. Et simpelt program	23
4.3. Standardfunktioner	25
4.4. Talsæt og <u>for</u> -sætninger	27
4.5. Betingelsessætninger	40
4.6. Etiketter og hopsætninger	46
5. SIMPLE PROGRAMEKSEMPLER	57
5.1. Beregning af vandgasligevægten	57
5.2. Beregning af trykfald i rør	63
6. PROCEDURER	78
6.1. Procedurebegrebet	78
6.2. Beregning af cylinderrumfang	78
6.3. Summation	85
6.4. Beregning af entalpi	96
6.5. Beregning af vandgasligevægt	102
6.6. Beregning af trykfald i rør	106
6.7. Rekursive procedurer	112
7. ANDRE ALGOLBEGREBER	115
7.1. Blokke	115
7.2. Logiske variable	127
7.3. Skiftespor	135
7.4. Brug af <u>integer</u> og <u>real</u>	138
7.5. Specielle <u>for</u> -sætninger	144
7.6. Brug af navne	150

8.	INDLÆSNING OG TRYKNING	153
8.1.	Læsning af tal	153
8.2.	Læsning af tegn	160
8.3.	Trykning af tal	173
8.4.	Trykning af tekst	181
8.5.	Trykning af tegn	182
9.	BRUG AF TROMLEN	190
9.1.	Disposition af lageret og tromlen	190
9.2.	Automatisk segmentering af programmet	192
9.3.	Tromleprocedurer	196
10.	SPECIELLE KONVENTIONER HOS HALDOR TOPSØE	204
10.1.	Udgaver af oversætterprogrammet	204
10.2.	Linie- og sideskift	209
10.3.	Forsidetrykning	214
10.4.	Behandling af inputdatagrupper	226
10.5.	Behandling af konstante talsæt og tekststreng	242
10.6.	ALGOL Bibliotek	249
11.	REFERENCER	251
12.	STIKORDSREGISTER	252

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 differentiaalligninger 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 detaljer 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:

$$1234 \times 9876$$

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:	Multipliser 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 10^4 tal hver med 8 betydende decimale cifre. Man siger, at hukommelsen (ogsaa kaldet lageret) indeholder 10^4 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 10^4 forskellige celler, har vi ialt $64 \times 10^4 = 65536$ 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

ovennevnte 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.

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:

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

som et eksempel (side 7) paa et ALGOL-udtryk. Vi ønskede at beregne tallet e ved hjælp af dette udtryk. Dette skrives i ALGOL saaledes:

$$e := a + b \times (c + d)$$

Meningen er, at naar maskinen har beregnet talværdien af $a + b \times (c + d)$ 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: $+$, $-$, \times 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 valse på skrivemaskinen.

Det simple udtryk:

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

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 på 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 1₁₀⁻¹² (een gange 10 i potensen -12). Bemærk iøvrigt den snedige måde vi noterer 10-potenser på: det lille 10-tal: ₁₀ skrives med et bestemt tegn på 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

$$p := a + b \times c + d / e - f \uparrow g;$$

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 $e - f \uparrow g$. Vi maa altsaa have nogle faste regler for operationssymbolernes indbyrdes styrke. Reglen er den, at operationerne skal udføres i følgende orden:

1. Potensopløftning (\uparrow)
2. Multiplikation og division (\times , $/$)
3. Addition og subtraktion ($+$, $-$)

I tilfældet ovenfor skal $b \times c$ altsaa beregnes inden vi adderer til a . Ligeledes skal d/e beregnes inden dette led summeres, og $f \uparrow g$ skal være beregnet inden det subtraheres. Som et kuriosum kan vi se paa, hvorledes oversætterprogrammet faktisk nedbryder regneudtrykket til simple 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:

$h := a + b \times c;$

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.:

$p := a + b \times c + d/(e - f/g);$

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:

$a \times b \times c$

$d \times e \times f$

kan dette i ALGOL skrives som enten:

$$q := a \times b \times c / (d \times e \times f);$$

eller:

$$q := a \times b \times c / d / e / f;$$

eller som kombinationer heraf. Derimod er

$$q := a \times b \times c / d \times e \times f$$

ensbetydende med, at man vil have beregnet brøken:

$$\frac{a \times b \times c \times e \times f}{d}$$

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:

$$a^2 + 2 \times a \times b + b^2$$

hvor en begynder ofte fejlagtigt vil skrive:

$$a^2 + 2ab + b^2$$

For at undgaa misforstaaelse bemærker vi, at en celledes indhold kun ændres, naar man gemmer noget i cellen, ikke naar man henter cellens indhold. Efter udførelse af sætningen $a := b$ er a-celledes indhold ændret, men b-celledes indhold er uændret.

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 * DIAMETER2;
```

altsaa højden gange kvadratet paa diameteren 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 * DIAMETER2  
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 deklarereres 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;  
  læs(HØJDE, DIAMETER);  
  RUMFANG := 0.785398 * HØJDE * DIAMETER2;  
  trykvr;  
  tryk({nddd.ddd}, HØJDE, DIAMETER, RUMFANG)  
end;
```

Sætningen:

```
  læs(HØJDE, DIAMETER);
```

vil i det oversatte program bevirke, at maskinen gør sig klar til at indlæse et hulbaand med læseapparatet 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 to sidste sætninger bevirker, at programmet perforerer et hulbaand med det ønskede resultat. Den første af de to sætninger:

trykvr;

bevirker at der perforeres tegnet for ny linie, d.v.s. et tegn, der naar hulbaandet udskrives paa Flexowriteren bevirker, at valsen skubbes tilbage til begyndelsen af en linie samt drejes frem til begyndelsen af næste linie (vr betyder: vogn retur).

Den sidste sætning:

```
tryk({nddd.ddd}, HØJDE, DIAMETER, RUMFANG);
```

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

Naar programmet afprøves med de to ovennævnte tal: 3.1 og 2.9, afleverer maskinen et hulbaand, som ved udskrift paa Flexowriteren giver:

```
3.100  2.900  20.476
```

Det ønskede rumfang er altsaa 20.476.

4.3. Standardfunktioner.

I AIGOL 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)	Fortegnssfunktionen 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 området 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 $\leq E$.

Betegnelsen E i disse eksempler skal antyde, at argumentet til disse funktioner ikke blot kan være et tal eller en variabel, men ogsaa 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: spild, som man iøvrigt ogsaa kan faa, hvis der optræder for store tal ($\text{tal} > 1.34_{10}^{154}$) i maskinen eller forekommer division med nul. Da potensopløftning: a^b for b real beregnes som $\exp(b \times \ln(a))$, kan man her faa ln-udskrift, hvis $a < 0$ eller spild-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 $\cos(E) \neq 0$. 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:

$$\begin{aligned}x[1]*a[1,1] + x[2]*a[1,2] + x[3]*a[1,3] &= a[1,4] \\x[1]*a[2,1] + x[2]*a[2,2] + x[3]*a[2,3] &= a[2,4] \\x[1]*a[3,1] + x[2]*a[3,2] + x[3]*a[3,3] &= a[3,4]\end{aligned}$$

Et talsæt kaldes i ALGOL for et array. 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 (arrays), 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;  
  læs(a, b, c, d, e, f, g, h, i, j);  
  sum := a + b + c + d + e + f + g + h + i + j;  
  trykvr;  
  tryk({ndddd.ddd}, sum)  
end;
```

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

I anden linie: læs(a, b, o.s.v. læser programmet de 10 næste tal fra datastrimlen og gemmer dem i cellerne for a, b, o.s.v.

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

I fjerde linie: trykvr; perforeres tegnet for ny linie.

I femte linie:

```
  tryk({ndddd.ddd}, sum);
```

perforeres 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];  
  læs(A);  
  sum := A[1] + A[2] + A[3] + A[4] + A[5]  
        + A[6] + A[7] + A[8] + A[9] + A[10];  
  trykvr;  
  tryk({\ndddd.dddd.}, sum)  
end;
```

Her har vi erstattet de simple variable: a, b, c, o.s.v. med talsættet A[1:10]. Sætning læs(A); er en forkortet skrivemaade for:

```
læs(A[1], A[2], A[3] ..... A[10]);
```

læs(A) vil bevirke læsning af lige saa mange tal fra data-strimlen, som der er elementer i talsættet. Det først læste tal gemmes i cellen for det første element, o.s.v.

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

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

Her maa vi indføre noget, som tillader at nedskrive summationen 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 $k = 1, 2, \dots, 10$, 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.

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

```
begin
  real sum;
  integer k;
  array A[1:10];
  læs(A);
  sum := 0;
  for k := 1 step 1 until 10 do
    sum := sum + A[k];
  trykvr;
  tryk({ndddd.dddd}, sum)
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];
  sum := 0;
  for k := 1 step 1 until 10 do
    begin
      læs(A[k]);
      sum := sum + A[k]
    end;
  trykvr;
  tryk({ndddd.dddd}, sum)
end;
```

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

```
læs(A[k]);
```

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 gasstrømme, 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];  
  for k := 1 step 1 until 4 do  
    begin  
      læs(Strøm1[k], Strøm2[k]);  
      Blanding[k] := Strøm1[k] + Strøm2[k];  
      trykvr;  
      tryk(⟨ndddd.dd⟩, Strøm1[k], Strøm2[k], Blanding[k])  
    end  
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 varmfylde.

Hvis gassens varmfylde er konstant, C_p kcal/kgmolC, vil entalpien, H kcal/kgmol, vokse lineært med temperaturen, t grader Celcius:

$$H := C_p \times t + I;$$

I er en integrationskonstant, d.v.s. det absolutte indhold af entalpi er for saa vidt ubestemt, indtil man har defineret en passende reference-tilstand. 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: $C_p \times t + I$ ikke være nøjagtigt nok, idet C_p 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:

$$H := a[0] + a[1] \times T + a[2] \times T^2 + a[3] \times T^3 + a[4] \times T^4;$$

Vi siger, at H beregnes af et fjerdegradspolynomium i T med de 5 koef-

ficienter: $a[0]$, $a[1]$,, $a[4]$. 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];
a[0] := -19625.11;
a[1] := 3.359595;
a[2] :=  $8.495905_{10}^{-3}$ ;
a[3] :=  $-7.112638_{10}^{-7}$ ;
a[4] :=  $-2.548678_{10}^{-10}$ ;
læs(T);
H := a[0] + a[1]*T + a[2]*T2 + a[3]*T3 + a[4]*T4;
trykvr;
tryk(⟨-ndddd⟩, H)
end;
```

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

Sætningen: læs(T); sørger for indlæsning af en værdi af T fra datastrimlen. Derefter beregnes H af polynomiet, og der trykkes 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 deklarereres 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];  
  læs(a);  
  læs(m);  
  læs(T);  
  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;
```

```
trykvr;  
tryk({-ndddd}, H)  
end;
```

Programmet begynder med deklaration af heltallene k og n og af de almindelige tal h , H og T . Derefter deklarerer 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 $8 \times 5 = 40$ a -koefficienter.

For ikke at gøre programmet for uoverskueligt lader vi det begynde med at indlæse de 40 a -koefficienter med sætningen $\text{læs}(a)$. Derefter indlæses de 8 m -værdier og endelig som før den ene T -værdi.

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 :=  $T \times h + a[n, k]$ ;
```

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 paaagældende 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 33 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 ikke 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 ikke 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:

array 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]	*	*

hvor de tre elementer mærket med * ønskes udeladt, vil deklarationen array 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 $D[(8-i) \times (i-1)/2 + j]$.

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 deklare-re:

```
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], protesteres der dog ikke, førend man prøver at hente eller gemme eet af elementerne.

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 deklara-tionsbetegnelsen: 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];
  sum := 0;
  for k := 1 step 1 until 10 do
    begin
      læs(A[k]);
      sum := sum + A[k];
      if A[k] > 100 then
        begin
          trykvr;
          tryk({nd}, k);
          tryk({ndddd.ddd}, A[k])
        end
      end;
    trykvr;
    tryk({ndddd.ddd}, sum)
  end;
```


Dette program adskiller sig kun fra den tidligere version, side 32, ved at der er tilføjet sætningen:

```
if A[k] > 100 then  
begin  
    trykvr;  
    tryk({nd}, k);  
    tryk({nddd.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 hulning af tegnet for ny linie, hulning 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 naturlig-

vis 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 program eksempel, 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. 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];
  sum := 0;
  for k := 1 step 1 until 6 do
    begin
      læs(element);
      A[k] := element;
      trykvr;
      tryk({ndd.dd}, element);
      if element < 0 then tryktekst({< Negativ});
      if element > 100 then tryktekst({< Over 100});
      sum := sum + element
    end;
  trykvr;
  trykvr;
  tryk({ndd.dd}, sum);
  if abs(sum-100) > 0.02 then tryktekst({< Fejl});
  forhold := A[1]/A[2];
  if forhold < 1 ∨ forhold > 5 then
    begin
      trykvr;
      tryktekst({<Skævt forhold})
    end
  end;
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 (0.9 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 tryktekst({< 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 tryktekst({< 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:

```
if forhold < 1 v forhold > 5 then
```

er benyttet ALGOL-tegnet for eller: v. Udtrykket a v 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 ikke tilladt at skrive:

if 10 < p < 20 then

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

if 10 < p \wedge p < 20 then

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:

$x < 17.4$

$x \leq 24.2$

$x = 98$

$x \neq 13$

$x > 9.8$

$x > 1.3$

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

Må man ogsaa bruge sammensatte logiske udtryk, hvori indgaar flere relationer, som f.eks. $10 < p \wedge p < 20$, 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.

$a = b$ læses: a ækvivalent med b . Er sand hvis a og b begge er sande eller begge falske, ellers falsk.

Endelig benyttes betegnelsen \neg , a for ikke- a , d.v.s. $\neg a$ 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 kompli- cerede beregninger opstaar der ofte den situation, at det kan være nødven- digt at overspringe visse dele af programmet, eller at gaa et stykke bag- læns 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 beregninger- ne 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 etiket- ter (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 tid- ligere program til summation af 10 tal fra en strimmel (side 32) kan man f.eks. skrive:

```
begin
  real sum;
  integer k;
  array A[1:10];
START: sum := 0;
LÆS:  for k := 1 step 1 until 10 do
Q17:  begin
P89:   læs(A[k]);
MNp:   sum := sum + A[k]
      end;
TRYK: trykvr;
tt5:  tryk({ndddd.dddd}, sum)
      end;
```

Anvendt paa denne maade tjener etiketterne ikke noget egentligt regne-

mæssigt 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];
ST:  sum := 0;
     for k := 1 step 1 until 10 do
       begin
         læs(A[k]);
         sum := sum + A[k]
       end;
     trykvr;
     tryk({ndddd.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-glossen: 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

på strimlen angive, hvor mange talgrupper på 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];
  læs(gruppeantal);
  for n := 1 step 1 until gruppeantal do
  begin
    sum := 0;
    for k := 1 step 1 until 10 do
    begin
      læs(A[k]);
      sum := sum + A[k]
    end;
    trykvr;
    tryk({ndddd.dddd}, sum)
  end
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-over-sætteren optager en etikette to celler af maskinens hukommelse, og det er her ligegyldigt, om den bruges eller ej. I eksemplet side 46 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(r2 + s2);
comment q er afstanden mellem punkterne;
v := q*x;
.....
.....

```

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 43, 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];
ST:  læs(TYPE, A);
      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
  trykvr;
  tryktekst({<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
        trykvr;
        tryktekst({<FEJL I KOMPONENT>});
        tryk({nd}, k)
      end fejludskrift
    end for k;
  go to SLUT;
T3: forhold := A[1]/A[2];
  if forhold < 1 ∨ forhold > 5 then
    begin
      trykvr;
      tryktekst({<SKEVT FORHOLD>})
    end;
  SLUT: end af programmet;
```

Programmet begynder med etiketten: ST og sætningen:

```
læs(TYPE, A);
```

som læser talværdien af TYPE fra strimlen samt de seks elementer af A.

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 hop-sætning: 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: spild paa skrivemaskinen og stop. Det havde været bedre at kode:

```
T3:   if A[2]  $\neq$  0 then forhold := A[1]/A[2];  
      if A[2] = 0  $\vee$  forhold < 1  $\vee$  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];
  læs(TYPE, A);
  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
          trykvr;
          tryktekst({<SUM IKKE 100>})
        end fejludskrift
      end if TYPE 1
    else
      if TYPE = 2 then
        for k := 1 step 1 until 6 do
          begin
            element := A[k];
            if element < 0 ∨ element > 100 then
              begin
                trykvr;
                tryktekst({<FEJL I KOMPONENT>});
                tryk({nd}, k)
              end fejludskrift
            end if TYPE 2
          else
            begin
              forhold := A[1]/A[2];
              if forhold < 1 ∨ forhold > 5 then
                begin
                  trykvr;
                  tryktekst({<SKÆVT FORHOLD>})
                end
              end if
            end
          end if
        end
      end if
    end
  end
```

```
    end fejludskrift  
  end TYPE 3  
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 TYPE3  
  else  
  begin  
    trykvr;  
    tryktekst({<FORKERT TYPE>})  
  end forkert type  
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:

$$\text{entier}(Q/7) \times 7$$

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):

$$\text{sign}(a/b) \times \text{entier}(a/b)$$

Det logiske udtryk:

$$Q:7 \times 7 = Q$$

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 \neq 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 \neq 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];
  læs(TYPE, A);
  if TYPE:2*2  $\neq$  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
          trykvr;
          tryktekst({<SUM IKKE 100})
        end fejludskrift
      end if TYPE 1, 3, 5, 7;
      if TYPE:4*2  $\neq$  TYPE:2 then
        for k := 1 step 1 until 6 do
          begin
            element := A[k];
            if element < 0  $\vee$  element > 100 then
```

```
begin
  trykvr;
  tryktekst({<FEJL I KOMPONENT>});
  tryk({nd}, k)
end fejludskrift
end if TYPE 2, 3, 6, 7;
if TYPE > 4 then
  begin
    forhold := A[1]/A[2];
    if forhold < 1  $\vee$  forhold > 5 then
      begin
        trykvr;
        tryktekst({<SKEVT FORHOLD>})
      end fejludskrift
    end if TYPE 4, 5, 6, 7
  end af programmet;
```


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:



kan foregaa. Hvis en gas indeholdende kulilte (CO) og vanddamp (H₂O) ledes over en passende katalysator i temperaturomraadet 300 - 500 gr. Celsius, vil noget af kulilten omsætte sig med vanddampen til kulsyre (CO₂) og brint (H₂). Reaktionskemaet viser, at 1 molekyle CO reagerer med 1 molekyle H₂O under dannelse af 1 molekyle CO₂ og 1 molekyle H₂. 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 m_{CO} , $m_{\text{H}_2\text{O}}$, m_{CO_2} og m_{H_2} , vil der gælde relationen:

$$K = \frac{m_{\text{CO}_2} \cdot m_{\text{H}_2}}{m_{\text{CO}} \cdot m_{\text{H}_2\text{O}}}$$

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:

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

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 m_{CO} , m_{H_2O} , m_{CO_2} og m_{H_2} , samt at ligevægten ikke er indtraadt, da vil forholdet:

$$\frac{m_{CO_2} \times m_{H_2}}{m_{CO} \times m_{H_2O}}$$

være forskelligt fra K. Hvis ligevægtsindstillingen bevirker, at molbrøken af CO falder med beløbet delta fra m_{CO} til m_{CO} , da vil molbrøken af vanddamp falde med et tilsvarende beløb, og molbrøkerne af CO_2 og H_2 vil begge vokse med beløbet delta. Naar ligevægten er indtraadt, har man da:

$$K = \frac{(m_{CO_2} + \text{delta}) \times (m_{H_2} + \text{delta})}{(m_{CO} - \text{delta}) \times (m_{H_2O} - \text{delta})}$$

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 Δ^2 . Løser vi ligningen med hensyn til Δ , faar vi:

$$\Delta = \frac{K \cdot m_{\text{OCO}} \cdot m_{\text{OH}_2} - m_{\text{OCO}_2} \cdot m_{\text{OH}_2}}{K \cdot (m_{\text{OCO}} + m_{\text{OH}_2}) + m_{\text{OCO}_2} + m_{\text{OH}_2}}$$

Vi begynder beregningerne med at sætte de søgte ligevægtsmolbrøker: m_{CO} , $m_{\text{H}_2\text{O}}$, o.s.v. lig med de opgivne molbrøker: m_{OCO} , m_{OH_2} , o.s.v. Derefter beregnes K af temperaturudtrykket og Δ af ligningen ovenfor. Molbrøkerne korrigeres derefter:

$$\begin{aligned} m_{\text{CO}} &:= m_{\text{CO}} - \Delta; \\ m_{\text{H}_2\text{O}} &:= m_{\text{H}_2\text{O}} - \Delta; \\ m_{\text{CO}_2} &:= m_{\text{CO}_2} + \Delta; \\ m_{\text{H}_2} &:= m_{\text{H}_2} + \Delta; \end{aligned}$$

De nye molbrøker indsættes i formlen for Δ i stedet for de gamle molbrøker, Δ beregnes igen, og molbrøkerne korrigeres, o.s.v. Iterationen afbrydes, naar Δ er blevet mindre end en vis værdi. Skal ligevægtsanalysen passe med en usikkerhed paa 0.01 procent, skal Δ 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 gasblandingsens originale sammensætning. I stedet for at opgive molbrøkerne, m_{OCO} , m_{OH_2} , o.s.v. i den originale blanding, er det mere praktisk at opgive molprocenterne, der er 100 gange større. Vi betegner disse med: M_{CO} , $M_{\text{H}_2\text{O}}$, 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, M_{In} , 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;

læs(t, MCO, MH2O, MCO2, MH2);

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);

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;

trykvr;

tryktekst({<Temperatur, gr. C>});

tryk({ndddd}, t);

trykvr;

tryktekst({< Molprocenter:>});

trykvr;

tryktekst({< Original Ligevægt>});

trykvr;

trykvr;

tryktekst({<Kulilite>});

tryk({ndddd.dd}, MCO, 100*mCO);

trykvr;

tryktekst({<Vand >});

tryk({ndddd.dd}, MH2O, 100*mH2O);

trykvr;

tryktekst({<Kulsyre>});

```
tryk({ndddd.dd}, MCO2, 100*MCO2);  
trykvr;  
tryktekst({<Brint });  
tryk({ndddd.dd}, MH2, 100*MH2);  
trykvr;  
tryktekst({<Inerte });  
tryk({ndddd.dd}, MIn);  
trykvr  
end af programmet;
```

Hvis dette program afprøves med følgende inputstrimmel:

400 30 10 10 50

giver outputstrimlen følgende udskrift:

```
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:

```
trykvr;  
tryktekst({< });  
tryk({ }, t);
```

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

```
tryk({ndddd.dd}, MCO, 100×MCO);
```

skal der trykkes to tal: først MCO og derefter 100×MCO. Begge tal trykkes efter skemaet: nndddd.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 integrationsmetode 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 andengrads-ligninger. I iterationsmetoden ovenfor har vi - maa ske 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 formelen:

$$\Delta P = \rho \times f \times \frac{L \times V \sqrt{2}}{D \times (2g)}$$

med f givet ved:

$$f = \frac{0.184}{NRe^{0.2}}$$

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:

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

med:

G	Gassens massehastighed (kg/(m ² ·h))
μ	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, μ, 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:

$$Pc[1], Pc[2], \dots, Pc[COMP]$$

og den kritiske temperatur:

$$Tc[1], Tc[2], \dots, Tc[COMP]$$

og den kritiske viskositet:

$$myc[1], myc[2], \dots, myc[COMP]$$

Antallet af komponenter betegnes med COMP. Blandingens kritiske tryk, P_{cmix} (atm.), beregnes som:

$$P_{cmix} := Pc[1] \cdot m[1] + Pc[2] \cdot m[2] + \dots + Pc[COMP] \cdot m[COMP]$$

hvor $m[1]$, $m[2]$, o.s.v. er molbrøkerne af de enkelte komponenter. Paa lignende maade findes blandingens kritiske temperatur, T_{cmix} (gr.K):

$$T_{cmix} := Tc[1] \cdot m[1] + Tc[2] \cdot m[2] + \dots + Tc[COMP] \cdot m[COMP]$$

og den kritiske viskositet, myc_{mix} (kg/mh), af blandingen:

$$myc_{mix} := myc[1] \cdot m[1] + myc[2] \cdot m[2] + \dots + myc[COMP] \cdot m[COMP]$$

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:

$$Pr := P/P_{cmix}$$

og analogt for den reducerede temperatur:

$$Tr := T/T_{cmix}$$

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:

$$\text{my} := \text{myr} * \text{mycmix}$$

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:

$$\begin{aligned} \text{myr} &:= a * \text{Tr}^b + c * \text{Tr}^d * \text{Pr} && (0 \leq \text{Pr} \leq 1) \\ \text{myr} &:= a * \text{Tr}^b + c * \text{Tr}^d + e * \text{Tr}^f * (\text{Pr} - 1) && (\text{Pr} > 1) \end{aligned}$$

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:

$$\text{rho0} := \text{rhon}[1] * \text{m}[1] + \text{rhon}[2] * \text{m}[2] + \dots \text{rhon}[\text{COMP}] * \text{m}[\text{COMP}]$$

Den aktuelle vægtfylde bliver saa:

$$\text{rho} := 273.16 / \text{T} * \text{P} * \text{rho0}$$

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 gasblanding, 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 hules derefter paa en hulstrimmel og bruges som inputstrimmel til programmet. Bemærk, at vi opgiver rørdiameteren i mm, medens det ovennævnte for-
melapparat 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
 $\rho_0 \text{ kg/m}^3$, saaledes at 1 kgmol gas vejer $22.415 \times \rho_0 \text{ kg}$. Den strømmende
vægtmængde bliver da $F \times 22.415 \times \rho_0 \text{ kg/h}$. Massehastigheden, G, har dimen-
sionen $\text{kg}/(\text{m}^2 \times \text{h})$, altsaa massen per m^2 tværsnitsareal af røret. Vi skal
altsaa dividere med tværsnitsarealet, A m^2 :

$$G := F \times 22.415 \times \rho_0 / A$$

og A findes af:

$$A := 0.785398 \times D^2$$

Talfaktoren er $\pi/4$ og D regnes i m.

Ved division af G (i $\text{kg}/(\text{m}^2 \times \text{h})$) 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;

læs(Dmm, L, F, t, P, MH2, MN2, MNH3);

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 blandingssegenskaber 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 indgaa i det samlede udtryk. Derefter kommer selve trykfaldsberegningen;

my := myr * mycmix;

A := 0.785398 * D²;

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 * V² / D / (2 * 1.2714₁₀⁸);

comment Talfaktoren 1.2714_{10}^8 er fremkommet ved omregning af tyngdeaccelerationen 9.81 m/sek^2 til $9.81 \times 3600^2 = 1.2714_{10}^8 \text{ m/h}^2$. 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;

$\text{delP} := \text{delP}/9678;$

tryktekst(⟨⟨

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

⟩);

tryk(⟨ndddd.dd⟩, Dmm, L, F, t, P, MH2, MN2, MNH3, delP);

trykvr

end af programmet;

Hvis programmet afprøves med følgende inputstrimmel:

12.7 2 446.13 20 300 75 25 0

ser udskriften af resultatstrimlen 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 inuttal og resultatet. Mellemlregninger trykkes ikke.

Vi gaar nu over til det andet tilfælde, hvori programmet skal kunne regne paa op til 8 gasser som forklaret side 67. 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 hules 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 til-

fælde 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];  
læs(Dmm, L, F, t, P, COMP);  
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  
            læs(CN[i], M[i]);  
            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*xm[1] + 0.80375*xm[2] + 1.24987*xm[3] + 1.33875*xm[4]
      + 1.24965*xm[5] + 1.96346*xm[6] + 1.78202*xm[7] + 0.71573*xm[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*1.2714*10^8*9678);
```

```
tryktekst(⟨⟨
```

D	L	F	t	P	delP
mm	m	kgmol/h	gr.C	atm.	atm.

```
⟩);
```

```
tryk(⟨ndddd.dd⟩, Dmm, L, F, t, P, delP);
```

```
trykvr;
```

```
tryktekst(⟨⟨
```

Komponent Molprocent

```
⟩);
```

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

```
begin
```

```
tryk(⟨ndddd⟩, CN[i]);
```

```
trykml(11);
```

```
tryk(⟨ndd.dd⟩, M[i]);
```

```
trykvr
```

```
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 to sætninger:

```
  læs(CN[i], M[i]);  
  m[CN[i]] := 0.01*M[i];
```

Den første sætning læser en linie med et komponentnummer og den tilhørende molprocent, og den anden 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 real 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: forkert 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:

```
tryk({ndddd}, CN[i]);
trykml(11);
```

erstattes med følgende:

```
j := CN[i];  
tryktekst(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 deklarereres i begyndelsen af programmet. Ogsaa her vil man kunne opnaa en forenkling af programmeringen ved brug af procedurebegrebet.

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 d_{elP} 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:

$$d_{elP} = \frac{\text{konstant}}{P} \times L$$

og dette udtryk kan integreres analytisk til at give:

$$PA^2 - PB^2 = 2 \times \text{konstant} \times L$$

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:

$$\text{RUMFANG} := 0.785398 \times \text{HØJDE} \times \text{DIAMETER}^2;$$

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 * D2;
```

Dette er den saakaldte proceduredeklaration, som fortæller oversætterprogrammet, 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 * D2;
```

som giver den formel, efter hvilken V skal beregnes.

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

```
V := 0.785398 * H * D2;
```

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 kald 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 * D2;
```

udskiftet med de tre parametre H1, Diam6 og V7, som staar i procedurekaldet:

CYL(H1, Diam6, V7);

Proceduresætningen ser da saaledes ud:

V7 := 0.785398 * H1 * Diam6²;

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 * d²;

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

CYL(hh, DW, W);

forårsage udførelsen af sætningen:

W := 0.785398 * hh * DW²;

De tre parametre: H, D og V, som optræder i proceduredeklarationen, kaldes formelle parametre, medens de tre parametre: H1, Diam6 og V7 i procedurekaldet betegnes aktuelle 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 sætning nemlig sætningen:

$$V := 0.785398 \times H \times D^2;$$

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 * D2;
```

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 procedure-sætningen:

```
cyl := 0.785398 * H * D2;
```

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

```
cyl := 0.785398 * H1 * Diam62;
```

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 deklarerer 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+m1/2, 3*k+7*m1/2, 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 + m2 step 1 until 3*k + 7*m2 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 bag-
efter skal bruges af hovedprogrammet. I det foreliggende tilfælde skal den
variable: s ikke kaldes ved value, da dens værdi er et resultat af proce-
durens regning. Gør man det alligevel, vil den aktuelle parameter i hoved-
programmet (altsaa her x, y eller z) slet ikke blive ændret.

I GIER ALGOL kan arrays ikke kaldes ved value. Dette ville ogsaa nor-
malt 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 proceduredeklarationen.

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 summe-
re 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:

$$x := \text{SUM}(M[j], j, 1, 10);$$

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 i stedet for E , kaldes ofte for Jensens device (efter Jørn Jensen, Regnecentralen).

Man vil da forstaa, at procedurekaldet:

$$x := \text{SUM}(M[j], j, 1, 10);$$

har ganske samme effekt som kaldet af den anden SUM-procedure paa side 87:

$$\text{SUM}(M, 1, 10, x);$$

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;

```

Virkingen 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);

Virkingen heraf er, at v bliver beregnet som summen af alle 8x9 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 betonedede 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 proceduredeklaration. 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 36 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 absolute 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 entalpieregningen er iøvrigt mægt til den tidligere udgave side 36 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. I-øvrigt havde det været lidt fikser 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 kulbrinteblending, som skal reformeres. 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],  
      ELLIST[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 er vist i kapitel 10, side 242, hvorledes dette kan gøres.

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 60 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 ca. 0.4 millisekunder at hente en aktuel parameter, som er en simpel variabel og ikke kaldt ved value (kaldt ved name), ser man, at der gaar $16 \times 0.4 = 6.4$ millisekunder ekstra til hver cyklus i beregningen fordi de fire molprocenter er kaldt ved name. Her kan altsaa spares nogen regnetid, hvis man ændrer proceduren, saa 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) + (((1.4752030310-10*T-9.6605186110-7)*T
+ 3.0101792910-3)*T - 1.50650387)*T + 4.94327234103)/T);
M1 := MCO;
M2 := MH20;
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;
MH20 := M2;
MCO2 := M3;
MH2 := M4;
end WGEQ;
```

Vi har maalt regnetiden for de to versioner af ligevægtsproceduren og fundet 54 millisek. og 41 millisek., hvilket tydeligt illustrerer forskellen.

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

```
begin comment Beregning af vandgasligevægt;
  integer i;
  real t;
  array MInput, MEQ[1:4];
  comment library WGEQ;
  læs(t, MInput);
  for i := 1 step 1 until 4 do MEQ[i] := MInput[i];
  WGEQ(t, MEQ[1], MEQ[2], MEQ[3], MEQ[4]);
  trykvr;
  tryktekst(⟨⟨Temperatur, gr.C⟩⟩);
  tryk(⟨ndddd⟩, t);
  trykvr;
  tryktekst(⟨⟨          Molprocenter:⟩⟩);
```



```
trykvr;  
tryktekst({<      Original  Ligevægt}&);  
trykvr;  
for i := 1 step 1 until 4 do  
begin  
    trykvr;  
    tryktekst(if i = 1 then {<Kulilte}&  
    else if i = 2 then {<Vand  &  
    else if i = 3 then {<Kulsyre}&  
    else {<Brint. &);  
    tryk({ndddd.dd}, MInput[i], MEQ[i])  
end for i;  
trykvr  
end af programmet;
```

Der er her indført et par. smaændringer. 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:

```
comment library WGEQ;
```

betyder, at der paa dette sted skal anbringes proceduredeklarationen af WGEQ. Det kan enten være et signal til hulledamen om at hun paa dette sted skal hente en strimmel med WGEQ-proceduren paa og lade Flexowriteren re-perforere denne strimmel ind paa selve programstrimlen. Men det kan ogsaa bruges paa en lidt fikserede maade, idet hulledamen simpelthen kopierer, hvad der staar:

```
comment library WGEQ;
```

Hvis man lader ALGOL-oversætteren oversætte programmet i denne form, faar man en fejludskrift om at der mangler en deklaration af noget i linie 8, nemlig proceduren WGEQ. Det gør vi naturligvis heller ikke. Filosofien er den, at regnemaskinen selv skal indsætte deklarationen, og det sker ved hjælp af et særligt program, som først indlæses i maskinen, og som i sig indeholder alle vore anerkendte procedurer (evt. et passende udvalg af dem)

paa ALGOL-form (ikke oversat). Derefter indlæses den programstrimmel, som indeholder sætningen:

```
comment library WGEQ;
```

og programmet reperforerer nu programstrimlen, indtil den naar denne sætning. Her vil programmet saa hulle proceduredeklarationen for WGEQ og derefter reperforere resten af programmet. Den nye programstrimmel kan derefter oversættes. Brugen af ALGOL-biblioteket er nærmere forklaret i afsnit 10.6.

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 72, 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;  
  læs(Dmm, L, F, t, P, COMP);  
  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 * D2;  
      G := F * 22.415 * rho0 / A;  
      rho := 273.16 / (t + 273.16) * P * rho0;  
      V := G / rho;  
      NRe := D * G / my;  
      f := 0.184 / NRe0.2;
```

```
delP := rho*f*L*V2/D/(2*1.271410*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*Tr0.60 + (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
  læs(CN[i], M[i]);
```

```
      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);
        tryktekst({<
D      L      F      t      P      delP
mm     m  kgmol/h  gr.C  atm.  atm.
});
      tryk({ndddd.dd}, Dmm, L, F, t, P, delP);
      trykvr;
      tryktekst({<
Komponent      Molprocent
});
      for i := 1 step 1 until COMP do
        begin
          tryk({ndddd}, CN[i]);
          trykml(11);
          tryk({nnd.dd}, M[i]);
```

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

Programmet begynder som tidligere med deklaration af en række integer og real 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]
```

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

Derefter deklarerer 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 $0.01 \times M[i]$. 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 blandingssegenskaber: 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 $Pc[i] \times m[i]$ 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 $0.001 \times Dmm$ 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. I kapitel 10 gives flere eksempler paa brugen af saadanne procedurer.

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:

$$v := \text{SUM}(\text{SUM}(Q[j,k], j, 1, 8), k, 1, 9);$$

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:

$$\text{FAC}(n) = 1 \times 2 \times 3 \times \dots \times n$$

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 $n \times \text{FAC}(n-1)$ og $\text{FAC}(n-1)$ skal da igen beregnes som $(n-1) \times \text{FAC}(n-2)$ 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:

$$e := a + b \times (c + d);$$

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 antydnet.

7.1. Blokke.

ALGOL-sproget er opbygget af sætninger. Vi har tidligere set eksempler paa betingede sætninger, hopsætninger, for-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 begin og end:

```
.....  
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 A'er 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 staa. 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 deklarerer 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 paa-gældende variable deklarerer 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.

For simple variable (altsaa integer, real og de senere omtalte boolean) kan man udvide deklARATIONEN med betegnelsen own, 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:

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	
<hr/>			
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 referencelle i den næstyderste 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 $12 \times 3 + 18 = 54$. 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, $p \times 3 + q$, 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 talsæt-elementer sker hver gang maskinen i det oversatte program skal have fat i et element.

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 næsttinderste 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 næsttinder-

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;
    trykvr;
    tryk({ndd}, 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 åbenbart 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 own arrays. Det er navnlig own arrays 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 tryktekst(⟨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.:

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

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 Engländeren 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 -1, men det er ikke tilladt at skrive:

```
begin  
  boolean S, T;  
  S := 1;  
  T := - 1;  
  .....  
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];
  NEG := false;
  for k := 1 step 1 until 10 do
    begin
      læs(element);
      A[k] := element;
      if element < 0 then NEG := true
    end for k;
    if NEG then
      begin
        trykvr;
        tryktekst({<Negativ})
      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 variable. 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];  
  for j := 1 step 1 until 3 do  
    begin  
      NEG[j] := false;  
      for k := 1 step 1 until 10 do  
        begin  
          læs(element);  
          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  
          trykvr;  
          tryktekst({<Negativ:>});  
          for j := 1 step 1 until 3 do if NEG[j] then tryk({ndd}, j)  
        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. deklarerere 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 54.

Paa side 45 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	<u>false</u>	<u>false</u>	<u>true</u>	<u>true</u>	
b2	<u>false</u>	<u>true</u>	<u>false</u>	<u>true</u>	
<hr/>					
-, b1	<u>true</u>	<u>true</u>	<u>false</u>	<u>false</u>	Nægtelse
b1 \wedge b2	<u>false</u>	<u>false</u>	<u>false</u>	<u>true</u>	Og
b1 \vee b2	<u>false</u>	<u>true</u>	<u>true</u>	<u>true</u>	Eller
b1 \Rightarrow b2	<u>true</u>	<u>true</u>	<u>false</u>	<u>true</u>	Implikation
b1 <u>=</u> b2	<u>true</u>	<u>false</u>	<u>false</u>	<u>true</u>	Ækvivalens

Beregning af komplicerede logiske udtryk som:

$$b1 \wedge (b2 = b3 \vee x > 100)$$

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 (<, <, =, >, >, \neq)
3. Nægtelse (-,)
4. Og (\wedge)
5. Eller (\vee)
6. Implikation (\Rightarrow)
7. Ækvivalens (=)

Disse regler er analoge med reglerne for rækkefølgen af operatorer i regneudtryk (side 20).

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 := I8, I19, I67, PQL;
  .....
  .....
  .....
  comment Her tildeles n en værdi ved beregning eller indlæsning;
  .....
  go to S[n];
  .....
I8:  .....
     .....
PQL: .....
     .....
     .....
I19: .....
     .....
     .....
I67: .....
     .....
end;
```

Programmet indeholder heltalsvariablen: n og skiftesporet: S. Den sidste skal deklareres i begyndelsen af en blok, f.eks. som her:

```
switch S := I8, I19, I67, PQL;
```

som bestaar af ALGOL-glosen switch, af navnet: S, af det dynamiske lig-

hedstegn (:=) og endelig en liste med de fire etiketter: L8, L19, L67 og FQL, 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 49 kan ogsaa skrives ved hjælp af et skifte-spor, f.eks. saaledes:

```
begin  
  real sum, forhold, element;  
  integer k, TYPE;  
  array A[1:6];  
  switch SW := T1, T2, T3;  
ST:  læs(TYPE, A);  
      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  
          trykvr;  
          tryktekst({<SUM IKKE 100>})
```



```
end;  
go to SLUT;  
T2: for k := 1 step 1 until 6 do  
  begin  
    element := A[k];  
    if element < 0  $\vee$  element > 100 then  
      begin  
        trykvr;  
        tryktekst(⟨⟨FEJL I KOMPONENT⟩⟩);  
        tryk(⟨nd⟩, k)  
      end fejludskrift  
    end for k;  
    go to SLUT;  
T3: forhold := A[1]/A[2];  
    if forhold < 1  $\vee$  forhold > 5 then  
      begin  
        trykvr;  
        tryktekst(⟨⟨SKEVT FORHOLD⟩⟩)  
      end;  
SLUT: 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 52.

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:

$$7.458_{10}^{-155} < \text{abs}(r) < 1.341_{10}^{154}$$

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:

spild

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 $2^4 \times 1.109375$. I cellen ser det saaledes ud (saakaldt lagring med flydende komma):

0000000100|01.0001110000000000000000000000

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 $2\frac{1}{4} \times 1.0625$, eller:

000000100|01.000100000000000000000000000000

Hvis vi fra dette tal 17 trækker det mindst mulige, som lige netop kan ses paa tallet, bliver det nu til:

000000100|01.000011111111111111111111111111

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. Disse tal gemmes i princippet paa samme maade som real variable, blot sørger maskinen altid for at afrunde tallet til nærmeste heltal, hvis der er mulighed for at tallet ikke er nøjagtigt et helt tal. Et tal som ovennævnte næsten-17 vil altsaa ikke kunne forekomme i en celle reserveret en integer variabel.

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 := ..... ;
```

sker der ingen afrunding. Det gør der heller ikke i:

```
i1 := i2 + i3*i4;
```

fordi højresiden kun indeholder integer variable, samt addition og multiplikation. Afrunding er derfor overflødig. Beslutningen herom træffes af oversætterprogrammet, idet det fremgaar af sætningens form. Naar det oversatte program regner, undersøger maskinen ikke om $i2 + i3 \cdot i4$ virkelig er et eksakt heltal, inden den gemmer det. Hvis der derfor er fejl i programmet, saaledes at $i2$, $i3$ og $i4$ slet ikke er blevet tildelt nogen værdi, kan programmet let faa fat i nogle gamle real variable fra en anden blok, som vi saa det paa side 125. I saa fald vil $i1$ ikke blive gemt som et heltal.

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. I udtrykket $i2 + i3/i4$ skal der ogsaa afrundes, fordi $i3/i4$ altid udføres som en rigtig 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:

$$-536870912 \leq i1 \leq 536870911$$

altsaa ca. en halv milliard. Maskinen giver ikke alarm, hvis en integer variabel kommer udenfor dette interval. Hvis det kun er en lille overskridelse af intervallet, bliver de sidste cifre i tallet forkerter, men er det en større overskridelse, kan resultatet blive helt forkert. Dette ses

af følgende eksempel:

```
begin
  real k;
  integer i1, i2, j;
  i1 := 536870911;
  k := 1;
  for j := 1 step 1 until 7 do
    begin
      i2 := i1*k;
      trykvr;
      tryk({nddddddddddddd}, i2);
      k := 10*k;
    end
  end;
```

Programmet skulle trykke en række tal, som bliver 10 gange større for hver gang. Men resultatet bliver:

```
536870911
5368709104
53687091071
536870910971
970662592502
910532935664
309236596733
```

Da man normalt ikke bruger saa store heltal, har denne faldgrube ikke saa stor praktisk betydning.

Derimod er der en anden fælde ved brugen af real og integer variable i procedurer. Hvis en formel parameter i en procedure af denne bliver tildelt en værdi og parameteren er specificeret til at være real, vil det gaa galt, hvis man i det aktuelle kald har skrevet en integer variabel for denne parameter. I saa fald sker der nemlig ingen afrunding. Som eksempel kan vi tage summationsproceduren fra side 89 :

```
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;
```

Her er den formelle parameter, s, specificeret til at være real. Hvis vi paa trods heraf i det aktuelle kald af proceduren erstatter s med en integer variabel, f.eks.:

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

hvor t er tænkt som integer, sker der ingen afrunding, hver gang der gemmes noget i t-cellen, hvis indhold efter beregningen derfor ikke behøver at være et eksakt heltal. Hvis M-talsættet er et integer array, vil summen, t, dog naturligvis blive et heltal. Det er iøvrigt navnligt ved procedurer for talindlæsning og tromleoverføring (se kapitel 9) at man kan begaa fejl af denne art.

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 det foreliggende tilfælde kunne man altsaa have udeladt:

```
real s;  
array A;
```

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 forholdsvis 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.

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-til type, 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: SÆTNING1 dækker over den sætning, som skal styres af V, saa giver den officielle ALGOL-definition en detaljeret forklaring paa for-sætningens virkemaade, idet man - stadig i ALGOL - omskriver til de følgende simple 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: if (V-C)*sign(B) > 0 da aldrig kan blive opfyldt. Sætningen: go to L3 bliver derfor aldrig udført, og maskinen kører derfor rundt i for-sætningen uendeligt længe, med mindre der i selve SÆTNING 1 findes en hop-

sætning, 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 for-sætningen:

```
for T := TSTART step DELT until TSLUT do
```

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 falde ved brugen af for-sætninger er spørgsmaalet om hvilken værdi, den styrende variabel, V, har naar for-sætningen er afsluttet. Ser vi paa den detaillerede forklaring paa side 144, 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 undefineret. 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 programøren 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 I4;  
.....  
.....  
end for V;  
I4: .....
```

Meningen er aabenbart, at den sidste del af den sammensatte SÆTNING 1 ikke skal udføres for $V = N$. Naar maskinen kommer til I4, 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  $\neq$  N then  
begin  
.....  
.....  
end if  
end for V;  
I5: .....
```

Her er værdien af V udefineret, naar maskinen naar frem til I5, omend vi i GDER ALGOL ved, at $V = N + 1$ (eller 1 for $N < 1$).

Paa side 75 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 saaledes:

```
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 printal:

```
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 $y = 3, 5, 7, \text{ o.s.v.}$ indtil $y^2 > x$. Selve indholdet af for-sætningen består 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.

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 eventuelt 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.

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 24 saa vi en anvendelse af læseprocedu-
ren: læs:

```
læs(HØJDE, DIAMETER);
```

Naar maskinen møder denne sætning 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.

Antallet af aktuelle parametre for læs er vilkaarligt, dog mindst 1. Som aktuelle parametre kan bruges navne paa simple variable (real eller integer) eller talsæt (real eller integer). Der maa gerne staa navne af forskellige typer mellem hinanden og ligeledes navne paa simple variable og talsæt mellem hinanden. Ved indlæsning af talsæt skal elementerne staa i en bestemt rækkefølge. I programmet:

```
begin  
  array A[1:2, 1:3];  
  læs(A);  
  .....  
  .....  
end;
```

skal A-elementerne staa saaledes paa strimlen:

A[1,1] A[1,2] A[1,3] A[2,1] A[2,2] A[2,3]

Der tælles altså først op i det bageste index. Man faar nøjagtig samme resultat, hvis man skriver:

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

men den første skrivemaade bør naturligvis foretrækkes, da den er kortere og hurtigere at udføre for maskinen.

Vi skal nu se paa formen af de tal, som skal staa paa inputstrimlen. I programmet:

```
begin  
  real P, Q, R;  
  læs(P, Q, R);  
  .....  
  .....  
end;
```

kan udskriften af inputstrimlen f.eks. se saaledes ud:

359.17 -318 1.24₁₀-5

Tallene er opbygget af cifrene fra 0 til 9 samt hjælpetegnene:

Plus:	+
Minus:	-
Punktum:	.
10-potens:	10

Proceduren: læs udfører den nødvendige analyse af hvert enkelt 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 læs-proceduren sker analysen naturligvis i et stykke program, skrevet i maskinsprog, for at det kan gaa saa hurtigt som muligt.

Hos Haldor Topsøe bruger vi en konvention for inputtallenes form, som afviger lidt fra den filosofi, som findes i GIER ALGOL II-oversætterprogrammet, saaledes som det er udarbejdet af REGNECENTRALEN. I den originale version af oversætteren kræves det, at der er et synligt tegn til afslutning af hvert tal. De tre tal ovenfor skal altsaa skrives som f.eks.:

359.17; -318; 1.24_{10}^{-5} ;

I stedet for semikolon kan bruges et hvilket som helst andet synligt tegn eller tegnet for ny linie.

Naar man daglig skal hulle store mængder inputmateriale, føler man trang til at undgaa at skulle hulle de mange semikoloner. Hos Haldor Topsøe har vi derfor rettet i oversætterprogrammet, saaledes at det ikke-synlige mellemrumstegn (SPACE) er tilstrækkeligt til at afslutte et tal. Det er derfor nok at skrive:

359.17 -318 1.24_{10}^{-5}

selve rettelsen er beskrevet nøjere i kapitel 10.

I den originale version er SPACE et blindt tegn, d.v.s. at det kan optræde midt i et tal uden forstyrrelse, f.eks.:

4.481 689; 4.486 173; 4.490 661;

I tal med mange betydende cifre er det praktisk med ekstra SPACES inde i tallene for at lette læsningen. Hvis ovennævnte tre tal blev læst med læs(P, Q, R) efter konventionerne hos Haldor Topsøe (SPACE som terminator) ville kun halvdelen af strimlen blive læst, og de variable ville faa værdierne:

P := 4.481;

Q := 689;

R := 4.486;

altsaa helt forkert. Da der kun yderst sjældent forekommer inputmateriale med mange betydende cifre i tekniske beregninger, har denne faldgrube ingen praktisk betydning for os. En lille smule alvorligere for os er det maaske, at vi heller ikke kan tillade SPACE omkring 10-potensen i tal med 10-potens. Vi skal altsaa skrive:

1.24_{10}^{-5}

men ikke:

$1.24 \quad 10^{-5}$

der vil blive opfattet som to tal: 1.24 og 10^{-5} . Det duer heller ikke med:

$1.24_{10} \quad -5$

hvor man vil faa udskriften: talfejl paa skrivemaskinen, fordi det ogsaa her opfattes som to tal: 1.24_{10} (der ikke er tilladt) og -5.

I kapitel 10 omtales andre specielle inputkonventioner hos Haldor Topsøe, f.eks. om indlæsning af uforandrede tal og om afslutning paa data-grupper.

Ved indlæsning af variable af typen integer, f.eks. i programmet:

```
begin  
  integer N;  
  læs(N);  
  .....  
  .....  
end;
```

skal man være opmærksom paa, at det indlæste tal ikke bliver afrundet, før det gemmes i cellen for N. Det er derfor nødvendigt, at der virkelig staar et helt tal paa strimlen. Disse eksempler:

Tal paa strimmel	giver	Tal i maskine
5		5.00000000
4.9		4.89999999
9.00		9.00000000
999		999.00000000
9.99_{10}^2		998.99999618

viser, at man ihvert fald bør undgaa 10-potens paa strimlen. Punktummet behøver ikke at skade.

Foruden proceduren læs findes der en tilsvarende procedurefunktion, som betegnes: læst. Det er en real procedure. I programmet:

```
begin  
  real x;  
  integer y;  
  .....  
  .....  
end;
```

kan vi faa indlæst x og y ved at skrive enten:

```
læs(x, y);
```

eller:

```
x := læst;  
y := læst;
```

og vi har her den fordel, at y bliver afrundet korrekt, fordi læst-proceduren er af type real og y af type integer.

Med læst-proceduren kan man lave forskellige smurrige ting, f.eks. i en sætning som:

```
B[læst, læst] := læst;
```

Hvis inputstrimlen hertil er:

3 7 3.847

bliver virkningen af ovennævnte sætning:

$B[3, 7] := 3.847;$

altsaa at talværdien 3.847 gemmes i B-elementet $B[3, 7]$. Dette betyder, at vi kan bruge en strimmel med selve dette udseende:

$B[3, 7] := 3.847;$

som input til sætningen:

$B[læst, læst] := læst;$

fordi de øvrige symboler paa strimlen er blinde. Men det gaar naturligvis ikke at bruge en inputstrimmel med dette udseende:

$A[3, 7] := 3.847;$

i det haab, at maskinen da vil tildele værdien 3.847 til et A-element. Det bliver stadig B-elementet, der faar tallet, saaledes som der staar i programmet. Bemærk iøvrigt, hvorledes maskinen i sætninger af formen:

$B[P, Q] := R;$

først behandler venstresiden, altsaa beregner (her indlæser) P og Q og udregner nummeret paa B-elementet. Derefter beregnes højresiden og sætningen udføres. Dette skal man være klar over af hensyn til ovennævnte anvendelse og ogsaa af hensyn til sætninger som:

$A[n] := n := n + 1;$

Hvis n er lig med 7, før sætningen udføres, vil tallet 8 blive gemt i n-cellen og i cellen for $A[7]$ (ikke $A[8]$).

Endelig er der en anden procedurefunktion, ved navn: `tast`, som er ganske analog med `læst`, blot tages input nu fra den skrivemaskine, der er tilkoblet maskinen, ikke fra hulstrimmellæseren. Har man altså i programmet sætningen:

```
x := tast;
```

skal man da, naar maskinen i det oversatte program naar frem til denne sætning, sætte sig hen til skrivemaskinen og skrive den ønskede værdi af `x`.

Procedureerne `læst` og (`især`) `tast` bruges kun sjældent i større rutineprogrammer. Til gengæld kan de være meget nyttige ved små eengangsprogrammer, f.eks. til afprøvning.

Har man logiske variable i sit program, og ønsker man at indlæse disses værdi fra en inputstrimmel, kan man godt programmere saaledes:

```
begin  
  real x;  
  boolean y;  
  læs(x, y);  
  .....  
  .....  
end;
```

Man skal da bruge talværdien 1 for `y` sand og -1 for `y` falsk. Inputtet:

```
4.7  -1
```

vil altså give:

```
x := 4.7;
```

og

```
y := false;
```

Man kan nu spørge, hvad der sker, hvis man skriver noget andet end 1

eller -1 paa inputstrimlen for y. Det indlæste tal vil blive gemt i y-cellen paa 2-potensform (flydende tal) som vist f.eks. paa side 138. Men da maskinen undersøger sandhedsværdien af en logisk variabel ved at føle paa det første ciffer i tallet (længst til venstre) ser vi, at alle tal med to-potenser ≥ 0 vil staa som sande, medens tal med to-potenser < 0 staa som falske. Vi faar derfor følgende sammenhæng mellem cellens talværdi, R, og dens logiske værdi, B:

$$B := R = 0 \vee R \geq 1 \vee R < -1;$$

Da dette kan være noget uoverskueligt, plejer vi hos Haldor Topsøe slet ikke at operere med logiske variable i det egentlige inputmateriale, men kun med tal. I den inputspecifikation, som brugeren af et program skal udfylde, kan der f.eks. staa:

(12)..... Tryk gasanalyse: 0: nej, 1: ja

Brugeren skal altsaa skrive 0 eller 1. Tallet indlæses sammen med det øvrige talmateriale og kan underkastes specielle undersøgelser sammen med dette. Naar vi kommer hen i nærheden af det sted i programmet, hvor denne logiske variable skal bruges, kan der f.eks. staa:

```
trykgas := DATA[12] = 1;
```

Talværdien 1 giver altsaa trykgas lig sand, alle andre talværdier giver værdien falsk.

8.2. Læsning af Tegn.

Vi saa, at ved indlæsning af tal skal læseproceduren udføre en analyse af, hvad der staa 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.

Tabel over Hulsymboler

Strimmel	Talværdi	LC	UC
---0-.---	0	SPACE	
---.---0	1	1	∨
---.---0	2	2	x
---0-.---00	3	3	/
---.---0---	4	4	=
---0-.---0-0	5	5	;
---0-.---00-	6	6	[
---.---000	7	7]
---0-.---	8	8	(
---00-.---0	9	9)
---00-.---0-	10	Bruges ikke	
---0-.---00	11	STOP CODE	
---00.---0-	12	END CODE	
---0.---0-0	13	Bolle-aa	
---0.---00-	14	_	
---00.---000	15	Bruges ikke	
---0-.---	16	0	^
---00-.---0	17	<	>
---00-.---0-	18	s	S
---0-.---00	19	t	T
---00-.---0-	20	u	U
---0-.---0-0	21	v	V
---0-.---00-	22	w	W
---00-.---000	23	x	X
---000.---	24	y	Y
---0-0-.---0	25	z	Z
---0-0-.---0-	26	Bruges ikke	
---000.---00	27	'	10
---0-0.---0-	28	CLEAR CODE	
---000.---0-0	29	Rødt skift	
---000.---00-	30	TAB	
---0-0.---000	31	PUNCH OFF	

Strimmel	Talværdi	LC	UC
-0---.---	32	-	+
-0-0-.--0	33	j	J
-0-0-.0-0	34	k	K
-0---.000	35	l	L
-0-0-.0--	36	m	M
-0---.0-0	37	n	N
-0---.00-	38	o	O
-0-0-.000	39	p	P
-0-00.---	40	q	Q
-0--0.---0	41	r	R
-0--0.-0-	42	Bruges ikke	
-0-00.-00	43	ø	Ø
-0--0.0--	44	PUNCH ON	
-0-00.0-0	45	Bruges ikke	
-0-00.00-	46	Bruges ikke	
-0--0.000	47	Bruges ikke	
-000-.---	48	æ	Æ
-00--.---0	49	a	A
-00--.-0-0	50	b	B
-000-.0-0	51	c	C
-00--.0--	52	d	D
-000-.0-0	53	e	E
-000-.00-	54	f	F
-00--.000	55	g	G
-00-0.----	56	h	H
-0000.-0-	57	i	I
-0000.-0-	58	LOWER CASE	
-00-0.-00	59	.	:
-0000.0--	60	UPPER CASE	
-00-0.0-0	61	SUM CODE	
-00-0.00-	62	Sort skift	
-0000.000	63	TAPE FEED	
0---.---	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 (små 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.

Nr. 12. END CODE. Findes dette tegn paa en inputdatastrimmel til et ALGOL-program, standser regnemaskinen, naar den møder tegnet og skriver ordet:

vent

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.

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 en inputdatastrimmel til et ALGOL-program, nulstilles en intern sumcelle. Talværdien af de efterfølgende symboler paa strimlen summeres i denne celle (modulo 128).

Nr. 29. Rødt skift. Skifter farvebaandet paa den tilkoblede skrivemaskine til rødt. Har ingen effekt paa Flexowriteren.

Nr. 30. TAB. Flytter valsen frem til næste tabulatorstop.

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). I den originale version af ALGOL-oversætteren er læseprocedurerne indstillet til paa inputstrimler til ALGOL-programmer at overspringe data mellem PUNCH OFF og PUNCH ON. Denne finesse er fjernet fra de oversætterprogrammer, som bruges hos Haldor Topsøe.

Nr. 44. PUNCH ON. (Se under Nr. 31).

Nr. 58. LOWER CASE. Skifter Flexowriteren (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 en input-datastrimmel til et ALGOL-program, 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:

sumfejl

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.

I GIER ALGOL II findes to forskellige metoder til indlæsning af de enkelte symboler fra strimlen. Den mest anvendte af disse benytter sig af proceduren læstegn, f.eks.:

```
begin
  integer n;
  .....
  n := læstegn;
  .....
```

Dette er en integer procedure uden formelle parametre. Talværdien af læstegn er den, som er givet i tabellen side 162-163, hvis tegnet er i lo-

ver case og forhøjet med 128, hvis tegnet er i upper case. Hvis vi har følgende program:

```
begin  
  integer i;  
  integer array B[1:6];  
  for i := 1 step 1 until 6 do B[i] := læstegn  
end;
```

og lader programmet læse en strimmel med de seks synlige tegn:

D=117;

da vil B-talsættet efter beregningen indeholde:

B[1]	180	D:	52 + 128
B[2]	132	=:	4 + 128
B[3]	1	1:	1
B[4]	1	1:	1
B[5]	7	7:	7
B[6]	133	§:	5 + 128

Denne læseprocedure vil altså automatisk holde regnskab med case-situationen, saaledes at hver gang den møder et casetegn (58 eller 60), gemmes oplysningen om det nye case internt, og den læser endnu et tegn. Dette er f.eks. sket ovenfor mellem B[2] og B[3], hvor maskinen maa have læst et tegn for lower case, og ligeledes mellem B[5] og B[6], hvor der maa have været et tegn for upper case.

De to case-tegn opfattes altså ikke som egentlige tegn og værdien af læstegn kan derfor aldrig være 58 og 60. Følgende andre symboler overspringes ogsaa:

Nr. 12	END CODE
Nr. 28	CLEAR CODE
Nr. 31	PUNCH OFF
Nr. 44	PUNCH ON
Nr. 61	SUM CODE
Nr. 63	TAPE FEED

men nr. 12, 28 og 61 vil udføre deres kontrolfunktion som forklaret ovenfor. Nr. 31 og 44 er helt blinde.

Som et eksempel paa brugen af læstegn viser vi nu en procedure, som udfører omtrent samme funktion som læst-proceduren, 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;  
  integer symbol, decimal;  
  real RESULTAT;  
  start := plus := true;  
  punktum := false;  
  decimal := 0;  
  RESULTAT := 0;  
A:  symbol := læstegn;  
  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  $\wedge$  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  $\wedge$  symbol < 9 then
C:  begin
        RESULTAT := RESULTAT*10 + symbol;
        if punktum then decimal := decimal + 1;
        go to A
    end if ciffer;
    if -, punktum  $\wedge$  symbol = 59 then
        begin
            punktum := true;
            go to A
        end if punktum;
    if symbol = 0  $\vee$  symbol = 128
     $\vee$  symbol = 64  $\vee$  symbol = 192 then
        begin
            if -, plus then RESULTAT := - RESULTAT;
            TAL := RESULTAT*10 $\wedge$ (-decimal);
            go to D
        end if afslutning;
        go to A
    end if not start;
D:  end TAL;
```


Proceduren opererer med de tre 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.

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 := læstegn;

som læser det næste rigtige symbol fra strimlen, d.v.s. efter overspringelse af case-tegn, END CODE, o.s.v. Det indlæste tegn gemmes som et integer i cellen: symbol.

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 $1 \leq \text{symbol} \leq 9$. 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. Da den rent mekaniske læsetid for et enkelt tegn er ca. 2 millisek., og tiden for et parameterfrit procedurekald er ca. 4.7 millisek., vil blot eet procedurekald pr. indlæst symbol bevirke, at regningerne gaar langsommere end indlæsningen.

Foruden proceduren læstegn findes en procedure: tastegn, der er ganske analog, men blot tager input fra skrivemaskinen istedet for fra strimmellæseren.

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 FQ 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 proce-

dure til indlæsning af tallene. Vi antager, at vi staar midt imellem ABC og 317 og læser eet symbol ad gangen med:

```
symbol := læstegn;
```

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. læst-proceduren:

```
j := læst;
```

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. Dette sker med proceduren: sættegn. Hvis vi skriver:

```
sættegn(3);
```

er dette en besked til maskinen om, at vi lader som om, at strimlen er kørt en plads baglæns, samt at næste gang maskinen indlæser et symbol, vil dette netop blive 3-tallet. I det ovennævnte tilfælde ved vi naturligvis ikke, at der staar et 3-tal her, saa vi maa i programmet skrive:

```
sættegn(symbol);
```

forudsat, at det sidst læste symbol virkelig er gemt i cellen med navn: symbol. Er vi i den fortvivlede situation, at vi ikke ved, i hvilken celle det sidst læste symbol er gemt - og det vil jo ske, hvis der kommer to talgrupper efter hinanden og vi bruger læst-proceduren - da er der endnu en udvej. Der er lavet yderligere en standardprocedure ved navn: tegn, hvis talværdi altid er det sidst læste symbol. Vi bør derfor skrive:

```
sættegn(tegn);
```

Baade sættegn og tegn er en integer procedure, men de aktiverer ikke strimmellæseren. Bemærk iøvrigt, at man ikke kan køre flere pladser baglæns paa een gang. Skriver vi:

```
sættegn(A);  
sættegn(B);
```

bliver A-et fortrængt, naar B-et indsættes, idet der kun er plads til at gemme eet tegn.

Naar vi bruger proceduren læstegn, bliver der automatisk holdt regnskab med case-situationen paa inputstrimlen. I nogle tilfælde vil man gerne kunne foretage strimmellæsningen med netop een fysisk hulrække ad gangen, altsaa ogsaa faa case-tegnene m.m. med som talværdier. Dette sker i den anden af de to metoder, som findes. Her bruges en integer procedure ved navn: lyn:

```
n := lyn;
```

Navnet kommer fra betegnelsen for den tilsvarende ordre i maskinsproget: læs fra ydre enhed med nulstilling af registret. I øvrigt er proceduren ogsaa særlig hurtig, fordi den ikke programmeres som et egentligt procedurekald, men direkte som de to nødvendige ordrer i maskinsprog.

Vi omskriver nu TAL-proceduren ovenfor, saaledes at der bruges lyn i stedet for læstegn. Vi maa da selv holde regnskab med case-situationen, f.eks. i en ikke-lokal logisk variabel: lowercase. Sætningen:

```
A: symbol := læstegn;
```

ændres da til følgende sætninger:

```
A: symbol := lyn;  
if symbol = 58 v symbol = 60 then  
begin  
    lowercase := symbol = 58;  
    go to A  
end if casetegn;  
if -, lowercase then symbol := symbol + 128;
```

Til sammenligning kan anføres, at regnetiden for versionen med læstegn er 72 millisek. og 49 millisek. for versionen med lyn for tal af formen:

-12.34 -12.34 -12.34

Som afslutning paa omtalen af indlæsemetoderne er det værd at bemærke, at den simple skrivemaade:

læs(x, y);

er et betydeligt fremskridt sammenlignet med f.eks. FORTRAN. Dette ældre programmeringssprog er baseret paa anvendelsen af hulkort, og man skal der give en lang forklaring paa hvorledes de to tal x og y staar paa hulkortet, altsaa antallet af decimaler, af heltal, og om der er 10-eksponent. Alt sammen oplysninger, der logisk set er overflødige, da maskinen jo selv kan undersøge, hvordan tallene staar.

8.3. Trykning af Tal.

I programmet side 24 var vist et eksempel paa trykning af tal:

```
tryk({nddd.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, f.eks. saaledes:

3.100 2.900 20.476

Proceduren: tryk arbejder med en liste af aktuelle parametre, hvorefter den første altid skal være et saakaldt layout, her:

```
{nddd.ddd}
```

Layoutet er skrevet paa en maade, saaledes at det umiddelbart gengiver tallets form. Her har vi maksimalt fire tal før kommaet: nddd og tre decimaler: .ddd

Det lille n paa første plads fortæller, at vi vil have trykt et heltalsnul. Der maa højst være 1 n, og det skal staa forrest. Tallet 0.123 faas som:

0.123 med {nddd.ddd}
eller .123 med {dddd.ddd}

I layoutet {dddd.ddd} giver det samlede antal d-er (eller n + d) 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:

{nddd}

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
	-.1	- .1	-.1
	.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 cif-fer.

Trykning med 10-potens kan ogsaa specificeres. Følgende layout:

{-d.ddd₁₀-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.d0010+d}  
  1.23510-3  
  12.3510-3  
  123.510-3  
  1.235  
  12.35  
  123.5  
  1.23510+3  
  12.3510+3
```

Reglen er: Hvis der er N nuller i layoutet (her 2), bliver eksponenten et multiplum af N + 1.

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 (1963a).

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;  
  trykvr;  
  for N := 71 step 1 until 130 do  
    begin  
      tryk({nd}, N);  
      trykml(2);  
      if N:10*10 = N then trykvr  
    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 10₁₀¹
10₁₀¹ 10₁₀¹ 10₁₀¹ 10₁₀¹ 11₁₀¹ 11₁₀¹ 11₁₀¹ 11₁₀¹ 11₁₀¹ 11₁₀¹
11₁₀¹ 11₁₀¹ 11₁₀¹ 11₁₀¹ 12₁₀¹ 12₁₀¹ 12₁₀¹ 12₁₀¹ 12₁₀¹ 12₁₀¹
12₁₀¹ 12₁₀¹ 12₁₀¹ 12₁₀¹ 13₁₀¹ 13₁₀¹ 13₁₀¹ 13₁₀¹ 13₁₀¹ 13₁₀¹

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:

tryk(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 tryk-procedurekaldet, gælder for trykningen af alle de følgende parametre i parentesens:

tryk({-nd.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 {-n.dd₁₀-dd}, kan man enten skrive:

tryk({-nd.dd}, A, B);

tryk({-n.dd₁₀-dd}, C);

tryk({-nd.dd}, D, E);

eller lidt fikserer:

tryk({-nd.dd}, A, B, tryk({-n.dd₁₀-dd}, C), D, E);

hvorefter man undgaar at gentage layoutet.

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. Man kan altsaa ikke skrive:

```
A1 := {-nd.dd};
A2 := {-n.dd10-dd};
tryk(A1, A, B);
tryk(A2, C);
tryk(A1, D, E);
```

Dette er forbudt, fordi et layout ikke er en tilladt type variabel, som real, integer, o.s.v.

Men der er andre maeder at klare sig paa. Een maade at gøre det paa er at deklareere en speciel procedure for trykning med hvert af de foreliggende layouts:

```
begin
  real A, B, C, D, E;
  procedure TA1(x);
  value x;
  real x;
  tryk({-nd.dd}, x);
  procedure TA2(x);
  value x;
  real x;
  tryk({-n.dd10-dd}, x);
  .....
  .....
  .....
  TA1(A); TA1(B);
  TA2(C);
  TA1(D); TA1(E);
  .....
  .....
end;
```

Her har vi desværre tabt muligheden for at anbringe trykparenteserne inden i hinanden, samt at bruge flere parametre. Man kan altsaa ikke skrive:

```
TA1(A, B, TA2(C), D, E);
```

Det er forbudt, fordi TA1 og TA2 udtrykkeligt er deklareret til at have netop een parameter.

En anden løsning ser saaledes ud:

```
begin
  real A, B, C, D, E;
  procedure TA3(A1, A2);
  string A1, A2;
  begin
    .....
    .....
    .....
    tryk(A1, A, B, tryk(A2, C), D, E);
    .....
    .....
  end TA3;
  TA3(⟨-nd.dd⟩, ⟨-n.dd10-dd⟩)
end;
```

Her er hele det hidtidige program gjort til en procedure ved navn TA3. Den har to formelle parametre: A1 og A2, som er specificeret til at være af typen string. Denne betegnelse har vi ikke mødt før, men den kan bruges i proceduredeklarationer til at fortælle, at den paagældende formelle parameter er en streng, d.v.s. en vilkaarlig række symboler anbragt imellem de to tegn:

⟨ ⟩

Et layout er netop en saadan streng.

I proceduredeklarationen maa der gerne forekomme sætningen:

```
tryk(A1, A, B, tryk(A2, C), D, E);
```

med de to formelle parametre A1 og A2. Efter deklarationen skriver vi blot sætningen:

```
TA3({-nd.dd}, {-n.dd10-dd});
```

som er et kald af proceduren TA3, og hvorved vi faar indsat de to aktuelle parametre, som er de ønskede layouts.

I eksemplet paa side 60 saa vi, at man godt kan anbringe regneudtryk som parametre i trykproceduren i stedet for blot navnene paa de variable, der skal trykkes:

```
tryk({ndddd.dd}, MCO, 100*MCO);
```

Saadanne regneudtryk kan være vilkaarligt indviklede og f.eks. indeholde kald af procedurefunktioner (deklarerede eller standard-funktioner):

```
tryk({-n.ddd10-dd}, sqrt(SUM(M[j], j, 1, 10)));
```

hvor SUM er tænkt som procedurefunktionen deklareret side 91.

Der findes en anden procedure: skrv, som er ganske analog med tryk, men giver udskrift paa den koblede skrivemaskine i stedet for paa perforatoren. Den kan bruges ved alarमुdskrifter o.l.

Til slut bemærker vi, at man har rejst den indvending mod procedurekald af formen:

```
tryk({-nd.dd}, A, B, C, D, E);
```

at dette ikke er korrekt ALGOL, fordi vi bruger en procedure med et paa forhaand ukendt antal parametre. Det maa indrømmes, at det er ærgerligt, at man ikke selv kan deklarere procedurer af denne art, men da der er tale om en indbygget standardprocedure kodet i maskinsprog, falder indvendingen formentlig bort.

8.4. Trykning af Tekst.

Eksempler paa teksttrykning har vi set i programmet side 43:

```
if element < 0 then tryktekst({< Negativ});
```

Proceduren har navnet: tryktekst, og den aktuelle parameter er en streng anbragt imellem tegnene:

```
{< }</pre></div>

Bemærk, at vi for layoutstrengene benytter:



```
{ }</pre></div>

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.

Kaldet af tryktekst kan indeholde flere parametre, f.eks.:


```
begin  
  real A, B, C;  
  .....  
  .....  
  if B < A ∨ B > C then  
  tryktekst({<B er for }, if B < A then {<lille} else {<stor});  
  .....
```


Her er den ene parameter: {<B er for} medens den anden indeholder en betingelse:


```
if B < A then {<lille} else {<stor}
```


En eller flere af parametrene kan ogsaa være formelle, paa samme maade som vi saa i proceduren TA3, side 179.

En anden maade at trykke tekststrengene paa er at kopiere direkte fra en inputstrimmel. Hertil bruges proceduren trykkopi. Den aktuelle parameter


```


```

hertil er en tekststreng med enten 1 eller 2 egentlige symboler. Eksempel med 1 symbol:

```
trykkopi({<]}) ;
```

Her kopieres inputstrimlen fra det sted den staar i øjeblikket, til den naar symbolet:] .

Eksempel med to symboler:

```
trykkopi({<[ ]}) ;
```

Her vil inputstrimlen blive læst blindt indtil tegnet: [mødes. Derefter kopieres som før, indtil tegnet:] mødes. De to definerende symboler: [og] kopieres ikke. Parameteren i trykkopi kan ogsaa være betinget eller formel.

Man har ogsaa de to procedurer: skrvtekst og skrvkopi analoge med tryktekst og trykkopi, men med skrivemaskineoutput i stedet for perforatoroutput.

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 læstegn kunne læse et symbol ad gangen.

Hertil bruges proceduren tryktegn:

```
tryktegn(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 hules specielt med:

```
        tryktegn(58);  
eller    tryktegn(60);
```

Ønsker vi saaledes at trykke kombinationen:

```
D=117;
```

med brug af tryktegn, kan dette gøres saaledes:

```
tryktegn(60);  
tryktegn(52);  
tryktegn( 4);  
tryktegn(58);  
tryktegn( 1);  
tryktegn( 1);  
tryktegn( 7);  
tryktegn(60);  
tryktegn( 5);
```

Som en mere interessant anvendelse af tryktegn 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;  
  tryktekst(⟨⟨  
    Strimmel          Talværdi          LC          UC  
  ⟩⟩);  
  trykvr;  
  tryktegn(58);  
  for i := 0 step 1 until 64 do  
  begin  
    if i = 0 ∨ i = 32 then HOVED else trykvr;  
    trykml(10);  
    tryktekst(⟨⟨|⟩⟩);
```

```
s := 0;
k := 1;
for j := 1 step 1 until 7 do
begin
  m := k:2x2;
  n := k - m;
  s := s + n;
  HUL[j] := n;
  k := m:2
end for j;
for j := 7 step -1 until 1 do
begin
  tryktegn(if HUL[j] ≠ 0 then 38 else 32);
  if j = 5 then tryktegn (if s:2x2 = s then 38 else 32);
  if j = 4 then tryktegn(59)
end for j;
tryktekst(⟨⟨|⟩⟩);
trykml(10);
tryk(⟨nd⟩, 1);
trykml(10);
if i = 0 then tryktekst(⟨⟨ SPACE⟩⟩)
else
if i = 10 ∨ i = 15 ∨ i = 26 ∨ i = 42 ∨ i = 45 ∨ i = 46 ∨ i = 47
then tryktekst(⟨⟨Bruges ikke⟩⟩)
else
if i = 11 then tryktekst(⟨⟨ STOP CODE⟩⟩)
else
if i = 12 then tryktekst(⟨⟨ END CODE⟩⟩)
else
if i = 13 then tryktekst(⟨⟨ Bolle-aa⟩⟩)
else
if i = 28 then tryktekst(⟨⟨ CLEAR CODE⟩⟩)
else
if i = 29 then tryktekst(⟨⟨ Rødt skift⟩⟩)
else
if i = 30 then tryktekst(⟨⟨ TAB⟩⟩)
else
if i = 31 then tryktekst(⟨⟨ PUNCH OFF⟩⟩)
else
```



```
if i = 44 then tryktekst(<< PUNCH ON>)
else
if i = 58 then tryktekst(<< LOWER CASE>)
else
if i = 60 then tryktekst(<< UPPER CASE>)
else
if i = 61 then tryktekst(<< SUM CODE>)
else
if i = 62 then tryktekst(<< Sort skift>)
else
if i = 63 then tryktekst(<< TAPE FEED>)
else
if i = 64 then tryktekst(<< CAR RET>)
else
begin
    tryktegn(i);
    trykml(if i = 14 then 11 else 10);
    tryktegn(60);
    tryktegn(i);
    tryktegn(58)
end symbol;
if i = 31 then for j := 1 step 1 until 11 do
    trykvr
end for i
end program;
```

Programmet består 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.

Trykning af visse tegn er saa ofte forekommende, at man har indført særlige procedurer herfor. To af disse har en parameter, som siger, hvor mange gange tegnet skal gentages:

trykml (N); Trykker N SPACES
tryktom(N); Trykker N TAPE FEED

Her kan N være et udtryk, f.eks.:

trykml(if N > 0 then 7 else 3×1);

Uden parametre har man følgende procedurer:

Navn	Symbol	Ækvivalent med:
trykstop	STOP CODE	tryktegn(11)
trykende	END CODE	tryktegn(12)
tryklar	CLEAR CODE	tryktegn(28) og nulstilling af sumcelle
tryktab	TAB	tryktegn(30)
trykslut	PUNCH ON	tryktegn(44)
trykvr	CAR RET	tryktegn(64)

Desuden haves proceduren tryksum, der trykker de tre tegn:

```
STOP CODE      tryktegn(11)
SUM CODE       tryktegn(61)
Intern checksum
```

Den vigtigste af de parameterfrie procedurer er trykvr, der huller tegnet for vogn tilbage og ny linie. Hos Haldor Topsøe bruger vi oftest en særlig procedure: LINE(N), som N gange laver trykvr med sideskiftkontrol. Den er beskrevet i afsnit 10.2.

De andre procedurer bruges næsten udelukkende ved hulning af intermediære strimler, d.v.s. outputstrimler, der senere skal bruges som input til et andet program eller en fortsættelse af selve programmet. Hulningen af den intermediære strimmel startes da med:

trykklar

Herved hules CLEAR CODE tegnet og den interne sumcelle nulstilles. Naar hulningen er færdig, afsluttes med:

tryksum

som bevirker hulning af STOP CODE, SUM CODE og den interne checksum. Naar strimlen senere indlæses, kontrolleres checksummen som forklaret side 165. Man bør naturligvis sikre sig, at SUM CODE virkelig bliver læst igen. Det er ikke godt at programmere saaledes:

```
begin
  integer N;
  array A[1:10];
  .....
  .....
  trykklar;
  for N := 1 step 1 until 10 do tryk(←-nd.do), A[N]);
  trykslut;
  .....
  .....
```

for senere at indlæse dette ved hjælp af:

```
begin  
  integer N;  
  array A[1:10];  
  .....  
  .....  
  læs(A);  
  .....  
  .....
```

Hvis der her ikke indlæses mere, naar maskinen slet ikke frem til SUM CODE. Man bør indsette hulning af et ekstra tal, altsaa erstatte trykslut med:

```
trykslut;  
tryk({n}, 0);
```

og erstatte læs(A) med f.eks.:

```
læs(A, N);
```

At N herved bliver nul er jo ligegyldigt, men nu har vi med sikkerhed passeret SUM CODE.

De fleste af de her nævnte procedurer kan bruges som parametre i den egentlige trykprocedure, f.eks.:

```
tryk({-nd.dd0}, A, trykml(7), B, trykvr,  
  if N > 1 then C else D, trykml(2), E + 9xF, trykslut, 0);
```

Se nærmere herom i Naur (1963a).

Endelig bemærker vi, at de fleste tryk-procedurer findes i en tilsvarende skrv-udgave med output paa den tilkoblede skrivemaskine i stedet for paa perforatoren:

tryk-form	skrv-form
tryk	skrv
tryktekst	skrvtekst
trykkopi	skrvkopi
tryktegn	skrvtegn
trykml	skrvml
tryktom	
trykstop	
trykende	
trykklar	
tryktab	skrvtab
trykslut	
trykvr	skrvvr

9. BRUG AF TROMLEN

9.1. Disposition af Lageret og Tromlen.

Vi har tidligere (side 121-123) set et eksempel paa sammenhængen mellem et lille ALGOL-program og de celler i maskinen, som oversættelsesprogrammet har udvalgt til at gemme de forskellige variable i. Data for de forskellige blokke var gemt saaledes:

Blok nr.	Celler
0 (yderst)	813 - 828
1	797 - 812
2	793 - 796
3 (inderst)	789 - 792

Maskinens hurtige hukommelse - det, vi normalt kalder lageret - indeholder 1024 celler nummereret fra 0 til 1023. I et kørende ALGOL-program er den høje ende af lageret altid fast optaget af en række administrative smaaprogrammer, f.eks. til beregning af adressen paa elementer i talsæt. Disse faste programmer fylder celle 836 - 1023. Derefter følger en lille liste med det saakaldte DISPLAY-talsæt, der i det foreliggende tilfælde optager cellerne 832 - 835 (se side 123). Disse celler indeholder oplysninger om, hvor de forskellige blokkes data er anbragt i lageret. Antallet af DISPLAY-celler er udregnet af oversætterprogrammet som det største antal blokke, der paa noget tidspunkt kan være til stede indeni hinanden.

Derefter følger de to celler 829 - 830 med programmets to own variable, og endelig de egentlige blokdata i cellerne 789 - 828. De resterende celler, altsaa fra 0 til 788 er da reserveret til lagring af selve programmet.

De ca. 800 celler, der saaledes er til raadighed til lagring af de nødvendige to typer af information:

1. Program
2. Talmateriale

vil kun tillade anvendelsen af ganske smaa programmer og talmængder. Det har derfor været nødvendigt at udvide maskinens hukommelse. Dette er sket ved at tilføje en saakaldt tromlehukommelse, der indeholder ialt 12800 celler, altsaa ca. 12.5 gange saa mange celler som i det egentlige lager.

Disse nye 12800 celler indgaar dog ikke paa lige fod med de 1024 celler i lageret, idet maskinen kun kan operere direkte paa cellerne i lageret, altsaa opfatte disses indhold som operationer eller tal. Naar tromlecellerne skal udnyttes, maa deres indhold først kopieres fra tromlen til lageret eller omvendt. Denne kopiering eller tromletransport sker i portioner paa 40 celler ad gangen.

I GIER ALGOL er tromlen disponeret saaledes, at hele programmet staar permanent paa tromlen. Hvis der er plads tilovers, kan denne bruges til at gemme talmateriale. Antager vi, at programmet fylder P celler, er tromlen opdelt i de tre omraader:

Omraade	Tromleceller	Anvendelse
1	0-2599	Faste programmer i oversætterens kørende system, m.m.
2	2600 til 12799-P	Lagring af talmateriale.
3	12799-P til 12799	ALGOL-program.

Den øverste ende af tromlen (fra celle 12799 og nedefter) bruges altsaa til lagring af det oversatte ALGOL-program. Den nederste ende af tromlen (celle 0 - 2599) er fast reserveret til standardprocedurer m.m. i det kørende ALGOL-system, samt til programmer i maskinsprog til administration af programmer i maskinsprog (GNL-1). Det mellemliggende omraade staar til raadighed for programmøren til lagring af talmateriale. Hvorledes dette gøres, forklares i afsnit 9.3.

9.2. Automatisk Segmentering af Programmet.

I eksemplet ovenfor saa vi, at cellerne 0 - 788 i maskinens lager var til raadighed til lagring af programmet under dettes kørsel. I virkeligheden er det lidt mere kompliceret, idet antallet af programceller ikke er konstant under kørslen med programmet. Umiddelbart efter starten af programmet arbejder maskinen endnu kun i den yderste blok og kan derfor disponere over cellerne 0 - 812 til programmet. Saa snart maskinen kommer ind i den næstyderste blok, skal cellerne 797 - 812 bruges til talmaterialet heri, og programcellerne maa derfor indskrænkes til 0 - 796. Denne reduktion af programpladsen fortsætter, hver gang der opstaar en ny blok inden i de gamle. Naar maskinen staar i den inderste blok, er programcellerne 0 - 788. Naar beregningerne i den inderste blok er forbi, forlades denne, og maskinen gaar igennem de mellemliggende, indtil den naar sætningen:

a := 2;

som staar i den yderste blok. Paa dette tidspunkt er alle cellerne 0 - 812 til raadighed for programmet, og der er derfor stor chance for, at de gamle tal i cellerne 789 - 812 er kommet til at indeholde program istedet for tal.

Den praktiske administration heraf foregaar nu paa følgende maade. Det komplette program er altid anbragt fast i den øverste ende af tromlen. Det er opdelt i segmenter paa 40 celler, svarende til det antal celler, der kan overføres til lageret ad gangen. Naar maskinen skal begynde at regne, overføres det første segment til de første celler i lageret, altsaa 0 til 39, eller helt korrekt 7 - 46, idet celle 0 - 6 er forbeholdt specielle ting.

Der regnes nu paa programmet i cellerne 7 - 46. Naar maskinen kommer frem til den sidste af disse celler (46), staar her en oplysning om, at segmentet slutter her, samt nummeret paa den tromlecelle, hvori programmet fortsætter. Maskinen hopper nu til et lille program i den høje ende af lageret, og hvori programsegmenteringen administreres. Det lille program har en liste over hvilke segmenter, der til enhver tid er til stede i lageret, samt hvor de staar. I vort tilfælde er det nye segment ikke til stede,

fordi vi lige er begyndt at regne, og administrationen vil derfor overføre dette segment fra tromlen til de næste ledige lagerceller, f.eks. til 48 - 97 (celle 47 er reserveret andet formaal). Derefter hoppes til det sted indenfor omraadet 48 - 97, hvor det rigtige program fortsætter. Det vil normalt ikke være celle 48, fordi den første del af segmentet bruges til at gemme de talkonstanter o.l., der skal bruges i dette segment.

Hvis maskinen kommer frem til celle 97, altsaa til slutningen af andet segment, gentages spøgen her med tredje segment, o.s.v.

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

Hvis et nyt segment skal overføres til lageret, men al lagerplads er udnyttet til gamle segmenter, vil det nye segment blive anbragt i stedet for et af de gamle segmenter, som derfor ikke længere er tilstede. Maskinen tæller, hvor mange gange hvert segment har været brugt, og det er segmentet med lavest prioritet, som bliver udskiftet.

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

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

Det laveste antal lagersegmenter, som maskinen kan køre med, er to. Her vil beregningerne gaa meget langsomt, hvis der hele tiden skal hentes nye segmenter frem. Talmaterialet kan i dette tilfælde brede sig fra celle 99 til ca. 830. Man vil heraf se, at der aldrig kan være mere end ca. 700 tal i lageret ad gangen. Programmet maa derfor være lavet saaledes, at der paa intet sted opereres med flere end ca. 700 simple variable og talsøtelementer. I praksis bliver tallet noget mindre, fordi der ogsaa skal være plads til administrative oplysninger blandt talmaterialet, f.eks. blokreferencer, data for talsøtgrænser, m.m.

I nogle tilfælde kan grænsen paa 700 tal overholdes, hvis man har lavet en fornuftig opdeling i blokke. Store talsæt, som kun bruges inden for et lille omraade af programmet, bør deklarereres i en blok paa dette sted, f.eks.:

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

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

Hvis maskinen under kørsel med et ALGOL-program kommer ind i en blok, hvor det samlede antal variable passerer den øvre grænse paa ca. 700, standser beregningen, og man faar udskriften:

ak

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

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

Iøvrigt bemærkes det, at man paa grund af den automatiske segmentering undertiden kan være ude for det uheld, at et ganske lille stykke af programmet, der gennemløbes mange gange, f.eks. en for-sætning til summation, er blevet placeret paa to forskellige segmenter, altsaa begynder i slutningen af det ene segment og slutter i begyndelsen af det næste. Der tabes herved nogen tid i segmenteringsadministrationen, som kunne være undgaaet, hvis for-sætningen havde ligget helt indenfor det ene segment. Har man

mistanke om, at dette uheld er sket, kan man afhjælpe det ved at indføre en eller flere harmløse sætninger som f.eks.:

```
x := x;
```

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

Et andet forhold, som staar i forbindelse med den automatiske segmentering er følgende:

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

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

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

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

Ogsaa smaa talkonstanter som 2, 3, 4, o.s.v. kan deklarereres saaledes. Tallet 1 behøver man dog ikke at deklarere, da det er indbygget i det kørende ALGOL-program:

Celle 839: 1 (eller true)
Celle 840: -1 (eller false)

9.3. Tromleprocedurer.

Vi har set, at maskinen højst kan operere med ca. 700 tal samtidigt i lageret og helst lidt mindre, hvis det ikke skal gaa for langsomt. Hvis vi derfor skal løse følgende opgave:

Læs 1000 tal fra en strimmel, find deres sum, og gem dem i maskinen til senere anvendelse, f.eks. trykning, da maa vi saubent tage tromlen til hjælp til lagring af tallene.

Vi bruger hertil to standardprocedurer:

til tromle(A);
fra tromle(A);

og en standardvariabel: tromleplads. Opgaven kan løses saaledes:

```
begin  
  integer i, j, top;  
  real SUM;  
  array A[1:100];  
  top := tromleplads;  
  SUM := 0;  
  for i := 1 step 1 until 10 do  
    begin  
      læs(A);  
      til tromle(A);  
      for j := 1 step 1 until 100 do  
        SUM := SUM + A[j]  
    end for i;
```

```
tromleplads := top;
for i := 1 step 1 until 10 do
begin
    trykvr;
    fra tromle(A);
    for j := 1 step 1 until 100 do
    begin
        tryk(⟨-ndd.dd⟩, A[j]);
        if j:10×10 = j then trykvr
    end for j
end for i;
trykvr;
tryk(⟨-ndddd.dd⟩, tryktekst(⟨<SUM:⟩), SUM)
end program;
```

Vi har set, at det oversatte ALGOL-program gemmes i den øverste ende af tromlen. Et program paa f.eks. 1600 celler vil altsaa optage tromlecellerne 11200 - 12799. Den første ledige tromlecelle er derfor nr. 11199.

Naar et oversat ALGOL-program er klar til at starte, har maskinen sørget for, at talværdien af den integer variable: tromleplads er sat lig med nummeret paa den første ledige tromlecelle, altsaa her 11199. I programmet ovenfor har vi sætningen:

```
top := tromleplads;
```

og værdien af top bliver derfor ogsaa 11199. Derefter følger den første forsætning med i:

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

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

```
til tromle(A);
```

overfører talsættet A til tromlen, saaledes at det højeste element anbringes i den tromlecelle, hvis nummer er lig med tromleplads, det næsthøjeste

element i cellen før, o.s.v. For $i = 1$ faar vi altsaa følgende overføring:

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

o.s.v.

```
A[1]   -      -      11100
```

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

Summationen af elementerne sker med sætningen:

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

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

```
tromleplads := top;
```

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

```
fra tromle(A);
```

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

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

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

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

Den lavest tilladelige værdi af tromleplads er 2600. Hvis værdien kommer herunder eller fejlagtigt bliver sat til en værdi højere end startværdien, standser maskinen, og man faar fejludskriften:

tromle ak

paa den tilkoblede skrivemaskine.

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

```
begin
.....
.....
begin comment Blok 1;
  array A[1:100];
  læs(A);
  tromleplads := a;
  til tromle(A);
  .....
end;
.....
.....
begin comment Blok 2;
  array C[1:100];
  tromleplads := a;
```

```
    fra tromle(C);
    .....
end;
.....
begin comment Blok 3;
    array D[1:20], E[1:80];
    tromleplads := a;
    fra tromle(E);
    fra tromle(D);
    .....
end
end program;
```

I Blok 1 indlæses talsæt A[1:100] og gemmes paa tromlen. I Blok 2 kaldes det frem igen som talsæt C[1:100]. I Blok 3 har vi endelig vist, at man ikke behøver at overholde de oprindelige dimensioner af talsættet, idet det oprindelige A-talsæt nu er fordelt paa to: D[1:20] og E[1:80]. Men her skal man naturligvis passe paa, at der ikke kommer kludder i indiceringen. Sammenhængen er her:

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

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

Simple variable kan ikke uden videre overføres ved hjælp af tromlepro-

cedurerne. Den parameter, der bruges i til tromle og fra tromle, skal altid være navnet paa et talsæt. Men man kan jo blot indføre et talsæt paa et enkelt element. Har vi saaledes en Blok 4 i eksemplet ovenfor, hvori en real variabel: P skal sættes lig med A[17], kan vi skrive:

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

Tromleprocedurerne bekymrer sig ikke om typen af de overførte talsæt. Hvis et talsæt er opstaaet som et real array i een blok, gemt paa tromlen og kaldt frem i en anden blok som et integer array, vil der ikke være sket nogen afrunding og man vil kunne faa fejl:

```
begin  
  .....  
  begin  
    array A[1:10];  
    comment A-elementerne udregnes;  
    .....  
    tromleplads := a;  
    til tromle(A);  
    .....  
  end;  
  .....  
  begin  
    integer array N[1:10];  
    tromleplads := a;  
    fra tromle(N);  
    .....  
  end;  
end program;
```

Situationen kan reddes, hvis man efter sætningen:

```
fra tromle(N);
```

tilføjer:

```
for i := 1 step 1 until 10 do N[i] := N[i] + 10-6;
```

De to procedurer til tromle og fra tromle er iøvrigt af formen integer procedure. Deres talværdi er lig med minus det overførte antal elementer. Hvis vi derfor skriver:

```
tromleplads := a1;  
a2 := a1 + til tromle(A);
```

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

Flerdimensionale talsæt behandles af tromleprocedurerne paa samme maade som af læs-proceduren. Efter udførelsen af programmet:

```
begin  
  array A[1:2, 1:3];  
  .....  
  tromleplads := 7006;  
  til tromle(A);  
end;
```

vil A-elementerne staa saaledes paa tromlen:

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

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

Til slut bemærker vi, at den aktuelle værdi af tromleplads gemmes i lagercelle 852.

10. SPECIELLE KONVENTIONER HOS HALDOR TOPSØE

10.1. Udgaver af Oversætterprogrammet.

De rutineberegninger, som for tiden udføres paa GIER-regnemaskinen hos Haldor Topsøe, falder i to grupper:

1. Regning paa programmer skrevet i maskinsprog
2. Regning paa programmer skrevet i ALGOL

Den første gruppe programmer blev udarbejdet i perioden fra maskinens installation i December 1961 til Februar 1963, hvor vi kunne begynde at bruge ALGOL-oversætterprogrammet. Efter dette tidspunkt er næsten alle nye programmer skrevet i ALGOL.

Da de allerfleste af programmerne er udarbejdet af en lille gruppe bestaaende af ganske faa programmører, men bruges af en helt anden og langt større gruppe, maa brugerne med rette kunne forlange, at der ikke er nogen synlig forskel paa programmer skrevet i maskinsprog og programmer skrevet i ALGOL. Konventionerne for input bør derfor være ens for de to typer programmer. Dette krav har vi kunnet opfylde fuldt ud, men kun ved at indføre nogle ganske faa ændringer i oversætterprogrammet. Disse ændringer beskrives nedenfor.

Oversætterprogrammet findes i følgende fire udgaver:

Nr.	ÆNDRINGER				
	PUNCH-ON -OFF er blind	Dato- ind- sætning	SPACE er terminator	Konden- seret program- output	Behand- ler lange pro- grammer
4	ja	nej	nej	nej	nej
5	ja	ja	nej	ja	nej
6	ja	ja	ja	ja	nej
7	ja	ja	ja	ja	ja

Følgende er fælles for de fire udgaver:

Oversætterprogrammet findes som en hulstrimmel i form af et saakaldt kondenseret output efter konventionerne i GNL-1. Betegnelsen GNL-1 dækker over de standardprogrammer til indlæsning og udlæsning m.m. som altid er tilstede i maskinen, idet de findes paa tromlecellerne 0 - 1279, som er aflaaet, saaledes at deres indhold ikke kan ødelægges. GNL-1-systemet er beskrevet af Kjær (1963a) og det er forskelligt fra de to andre systemer: SLIP og HJÆLP, der anvendes ved de øvrige GIER-installationer.

De kondenserede udgaver af ALGOL-oversætteren indlæses i maskinen ligesom ethvert andet rutineprogram.

Behandling af PUNCH-ON-OFF. I den originale version af oversætteren analyseres alle inputstrimler til et oversat program for indhold af de to symboler: PUNCH ON og PUNCH OFF. Forekomsten af disse to symboler har to effekter:

1. Alle symboler mellem PUNCH OFF og PUNCH ON overspringes (ignores).

2. PUNCH ON er terminator for talindlæsning.

Den sidste effekt er den farligste. Hvis vi har talgruppen:

123.45 678.90

paa en datainputstrimmel og hulledegen af vanvare har sat et PUNCH ON symbol mellem 2-tallet og 3-tallet, vil maskinen opfatte dette som:

12 3.45 678.90

Der kommer altsaa et ekstra tal ind i datagruppen og det hele bliver forrykket og forkert.

Forekommer der pludseligt et isoleret PUNCH OFF tegn, ogsaa sat af vanvare, vil maskinen afsøge inputstrimlen for det tilsvarende PUNCH ON tegn, der naturligvis normalt ikke findes. Den læser derfor resten af inputstrimlen og vil endda have mere. Denne fejl kan derfor normalt ikke undgaa operatørens opmærksomhed.

Da man ikke kan se de to tegn for PUNCH OFF og PUNCH ON paa en kontrol-

udskrift af inputmaterialet, har vi simpelthen rettet oversætterprogrammet, saaledes at begge tegn er blinde, ligesom f.eks. TAPE FEED. Blindheden gælder for alle læseprocedurer: læs, læst, læstegn, trykkopi, o.s.v., men ikke for lyn-proceduren.

Datoindlæsning. For alle programmer skrevet i maskinsprog har vi den konvention, at outputrapporten fra hver beregning automatisk bliver dateret af maskinen. Datoen skal ikke staa paa inputstrimlen, men er fast lagret i tre bestemte lagerceller som maaned, dag og aar. De tre tal staar ogsaa i tre tromleceller i den aflaaste del af tromlen. Hver morgen vil operatøren laase op for tromlen, rette datoen og laase igen. Hver gang maskinen nulstilles før en ny beregning, overføres de tre datotal fra tromlen til de tre lagerceller.

For at indføre den samme automatiske datering af beregninger paa programmer skrevet i ALGOL har vi gjort følgende:

ALGOL-programmet skal indeholde en bestemt procedure: COVER1, som varetager trykning af forside og dato. Den er beskrevet i afsnit 10.3 nedenfor. Det væsentlige i proceduren med henblik paa datering er tre sætninger:

```
.....  
begin  
  real month, day, year;  
  .....  
  month := 1960;  
  day := 1961;  
  year := 1962;  
  .....
```

Kunsten er nu at faa ombyttet de tre tal: 1960, 1961, 1962 med den aktuelle værdi af maaned, dag og aar. Det kan ikke gøres under oversættelsen, men maa ske hver gang det oversatte program indlæses i maskinen.

De tre udgaver: 5, 6 og 7 af oversætterprogrammet er derfor ændret saaledes, at naar den egentlige oversættelse er færdig, og det oversatte program staar i den høje ende af tromlen, da træder et lille specialprogram i funktion, som gennemsøger hele det oversatte program og laver en liste over, i hvilke tromleceller de tre tal 1960, 1961 og 1962 staar. Denne liste gemmes i lageret, og derefter sker hulning af det oversatte program

+ listen i kondenseret form. Hver gang det kondenserede program derefter indlæses, og maskinen startes, vil et andet lille specialprogram (som ogsaa er hullet med ud fra lageret) træde i funktion. Det henter de tre tal for den aktuelle dato fra de tre lagerceller og anbringer dem i de tromleceller, som listen angiver. Derefter fortsættes med det normale ALGOL-program, og listen gaar tabt.

Man vil forstaa, at metoden svigter, hvis programmet indeholder de tre indbyggede talkonstanter: 1960, 1961 og 1962 anbragt paa andre steder i programmet med henblik paa normal anvendelse. Disse tal vil da ogsaa blive udskiftet med den aktuelle dato. Skulle noget saadant forekomme, maa man bede om operatørens hjælp til at fjerne de forkerte tromlecellenumre fra listen. De tre tal: 1960, 1961 og 1962 maa derimod gerne bruges som input til programmet, da udskiftningen er sket, naar programmet begynder at regne.

SPACE som terminator. Som forklaret side 155 har vi fundet det praktisk at gøre SPACE til terminator for talindlæsning. Tallet anses altsaa for afsluttet, naar der mødes et SPACE. Synlige tegn som semikolon, bogstaver, og CAR RET fungerer stadig som terminatorer. Ved denne ændring lettes fremstillingen af inputstrimler, og vi faar samme konvention som for inputstrimler til programmer, der er skrevet i maskinsprog.

Denne ændring findes kun i udgave 6 og 7. Selve rettelsen er iøvrigt gjort ved at simulere, at det indlæste SPACE er et TAB-symbol, altsaa tegnet for tabulator. Herved er det kun een celle, der behøver at rettes (celle 14 paa tromlekanal 58). De to indhold af cellen er:

Original version:

HV r- 13 LZ;

Rettet version:

ARSD (846) + 30 LZ

Her staar tallet 30 for TAB-symbolet. Man kan indsætte andre tal her, hvis man har lyst. I programmer oversat før 6. april 1964 staar 64 istedet for 30, saaledes at et SPACE blev konverteret til CAR RET, men dette var sommetider upraktisk, hvorfor CAR RET blev rettet til TAB. Man skal iøvrigt være opmærksom paa at værdien af proceduren: tegn (se side 172) er 30 og ikke 0 efter indlæsning af et tal afsluttet med SPACE.

Konvertering af SPACE til TAB sker kun ved de egentlige læseprocedurer: læs, læst og tast men ikke ved læstegn, trykkopi og lyn.

Kondenseret Program. Naar en oversættelse er afsluttet, vil udgave 5, 6 og 7 (men ikke 4) perforere det oversatte program paa den kondenserede form. Ønskes dette output ikke, kan man koble perforatoren fra, lige efter at den er begyndt at perforere.

Normalt bør man indlæse det kondenserede program i maskinen inden afprøvningen, men det er ikke strengt nødvendigt. Snyder man for indlæsningen, vil datomekanismen dog svigte, og hvis proceduren COVER1 er anvendt, vil outputrapporten blive dateret: January 20₁₀, 1962.

Lange Programmer. Normalt anvendes udgave 6 af oversætterprogrammet, men ved meget store programmer er der ikke plads til oversætterprogrammet og det oversatte program samtidigt. Vi har derfor lavet udgave 7, som kan klare væsentligt større programmer. Her indlæses kun en lille smule af oversætterprogrammet først og derefter hele det ALGOL-program, som skal oversættes. Resten af oversætterprogrammet indlæses derefter automatisk efterhaanden som der bliver brug derfor. Det kondenserede output af det oversatte program er automatisk opdelt i to ligestore strimler for at lette den senere indlæsning.

For fuldstændighedens skyld bemærker vi, at de her foretagne rettelser med SPACE, PUNCH OFF, etc. kun vedrører kørslen af oversatte programmer. I selve ALGOL-programmets sætninger har vi ikke lavet rettelser. Her kan man altsaa skrive:

```
x := 4.567 987;  
y := 1.2310 -5;
```

idet SPACE her er blindt. Ligeledes vil oversætterprogrammet overspringe alt program mellem PUNCH OFF og PUNCH ON, med mindre man har bedt den om at lade være, hvad man iøvrigt altid bør.

Flere af de her indførte rettelser som datoindsættelse og oversættelse af lange programmer vil iøvrigt kunne udføres paa den nye oversætter til GIER ALGOL III uden rettelser (Naur 1964). Vi har i øjeblikket en udgave 8 af ALGOL III-oversætteren, svarende til udgave 4 af ALGOL II-oversætteren, men vi har endnu ikke vedtaget at gaa over til at bruge ALGOL III.

10.2. Linie- og Sideskift.

Naar store mængder beregningsresultater skal udskrives automatisk paa en Flexowriter, er det praktisk, hvis outputstrimlen er hullet, saaledes at det udskrevne automatisk passer med sideinddelingen i de lange papirbaner, der benyttes ved udskriften. Man bør derfor forsyne sine programmer med procedurer, der sørger for en automatisk tælling af linierne paa hver side og skift til ny side, hver gang det er nødvendigt.

Vi bruger normalt proceduren:

LINE(n);

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

-7-

6789-2

Her er 7-tallet sidens nummer, og 6789 er beregningsnummeret, der er opgivet i inputmaterialet. Endelig er 2-tallet sektionsnummeret. Det sidste beregner programmet selv, idet det altid begynder med sektion 1. Ønsker man at gentage beregningen med lidt andre inputtal, indlæses disse i fortsættelse af sektion 1, og maskinen gentager beregningen som sektion 2, o.s.v.

LINE-proceduren maa derfor have adgang til følgende variable:

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

calcno: Beregningsnummer.

sectno: Sektionsnummer.

pageno: Sidenummer.

Man kunne have lavet proceduren med disse formelle parametre:

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

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

Vi har derfor vedtaget, at disse fire parametre skal være ikke-lokale variable, som er deklareret i den yderste blok. For at programmøren kan huske at faa disse variable med, har vi en særlig formular, som han kan udfylde ved programmering af begyndelsen til et program. Formularen er vist paa næste side.

Programmeringsformular.

Program

begin

boolean

integer LRest, calcno, sectno, pageno,

real

array

comment library LINE;

comment library PSHIFT;

sectno := 1;

AA: begin comment cover and headline;

integer i;

array B[0:39];

comment library CENTEXT;

comment library COVER1;

comment library HEADLINE;

if sectno > 1 then go to BB;

COVER1(, ,

}<

}<

for i := 0 step 1 until 39 do B[i] := -1234;

tromleplads := ;

for i := 1 step 1 until do tiltromle (B);

BB: HEADLINE(20, ZZ)

end of cover and headline;

begin comment input block;

boolean

integer

real

array

comment library READ5;

comment library READ6;

READ6(67,)

Vi ser, at den yderste blok indeholder deklarationen:

```
integer LRest, calcno, sectno, pageno;
```

Programmøren kan tilføje flere variable her, ogsaa af de andre typer. Den her anvendte metode er lidt forskellig fra den som bruges ved RISØ-GIER-installationen, se Vilstrup (1963). Her anbringes linie- og side-skiftparametrene i en alleryderste blok, som omgiver programmørens program, og som han iøvrigt ikke har adgang til. Det har den fordel, at man ikke glemmer parametrene, men til gengæld faar man et ekstra blokniveau i programmet og har ingen fordel af den absolutte adressering paa de variable i den yderste blok.

Foruden de fire variable integer, har vi ogsaa deklareret de to procedurer LINE og PSHIFT i den yderste blok. Deklarationen for LINE er:

```
procedure LINE(n);  
value n;  
integer n;  
begin  
  integer ia;  
  LRest := LRest - n;  
  if LRest < 0 then go_to A;  
  for ia := 1 step 1 until n do trykvr;  
  go_to B;  
A:  LRest := LRest + n + 5;  
    LINE(LRest);  
    LRest := 65;  
    trykml(36);  
    tryk({-n}, - pageno);  
    tryktegn(32);  
    trykml(27);  
    tryk({-ndddd}, calcno);  
    if sectno < 10 then tryk({-n}, - sectno)  
    else tryk({-nd}, - sectno);  
    pageno := pageno + 1;  
    LINE(3);  
B: end of LINE(n);
```

Bemærk, at proceduren i programmeringsformularen er deklareret ved:

```
comment library LINE;
```

Det er forklaret nedenfor i afsnit 10.6, hvorledes den rigtige deklaration indsættes istedet for denne kommentar.

LINE virker saaledes: Først formindskes LRest med n. Er resultatet negativt, hoppes til A, hvor sideskiftet udføres, ellers trykkes blot CAR RET n gange, og proceduren er færdig. Skal siden skiftes, reableres LRest og forhøjes yderligere med 5:

```
LRest := LRest + n + 5;
```

og der trykkes CAR RET LRest gange. Bemærk, at dette sker ved et kald af proceduren selv:

```
LINE(LRest);
```

som altsaa bruges rekursivt i egentlig forstand. Det kunne naturligvis ligesaa godt have været gjort med en for-sætning.

Proceduren indsætter nu værdien 65 for LRest, trykker 36 SPACE, trykker sidetallet med et minus foran og bagved, samt 27 SPACE. Derefter trykkes beregningsnummeret, efterfulgt af et minus og sektionsnummeret. Sidetallet øges med 1, og der laves 3 frie linier, igen ved et rekursivt kald af LINE.

Naar man trykker tabeller, vil man gerne sikre sig, at der ikke skiftes side midt i tabellen, undtagen hvis det er en meget stor tabel. Hertil bruges proceduren:

```
PSHIFT(n);
```

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

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

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

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

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

før tabeltrykningen.

I den her viste udgave af LINE er der regnet med 70 linier pr. side, svarende til tætteste linieafstand paa Flexowriteren. Nærværende bog er skrevet med noget større afstand: $70/1.5 = 46.7$ linier pr. side.

Antallet af anslag pr. linie er højst 80. Her findes ingen automatisk kontrol.

10.3. Forsidetrykning.

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

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

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

En typisk forside er vist paa næste side:

Forsideeksempel

May 15, 1964

File No. 358

The Ammonia Co., Gastown

Analysis of Converter Performance

GIER Calculation No. 6789

Calculation of TVA-Type Ammonia Converter

GIER Program TVA-1

Foruden teksten paa forsiden ønsker vi ogsaa, at der over hver beregningssektion kan staa nogle linier med forklarende tekst, som ogsaa skal opgives i inputmaterialet.

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

Inputstrimlen til forsideeksemplet side 215 maa da have set saaledes ud:

Ff

6789

[358

[The Ammonia Co., Gastown

[Analysis of Converter Performance

e

[Data from November, 1963

e

Herefter følger det egentlige talmateriale. Overskriften til sektion 1:

Data from November, 1963

er ikke vist paa forsiden, men kommer øverst paa side 2 i rapporten.

Inputformular

0. Cover Data:

Ff

..... Calculation No.

[..... File No.

[..... Cover Page Text

[..... - - -

[..... - - -

e Stop e

1. Headline Data:

[..... Section Headline Text

[..... - - -

[..... - - -

e Stop e

Inden vi studerer selve forsideproceduren, bemærker vi, at der er en finesse ved behandlingen af de variable tekststrengene paa forsiden og i overskriften. Den første tekststreng:

[The Ammonia Co., Gastown

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

[The Ammonia Co., Gastown

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

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

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

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

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

Proceduren arbejder med et integer array text[1:100], hvori den gemmer de indlæste symboler, saa snart den er kommet ind i en tekststreng. Inden strengen er mødt, analyseres symbolerne kun for stop-e og [. Medens strengen løses, analyseres for CAR RET, der giver hop til c3. Den viste analyse for TAPE FEED er overflødig. Derimod tælles antallet af symboler i strengen, dog saaledes at tegnet for understregning: _, og den lodrette streg: | ikke tælles med, da disse ikke flytter vognen. Naar CAR RET-tegnet er

mødt, huller proceduren saa mange SPACE, som svarer til at der er 80 anslag per linie. Derefter hules hvert tegn ved hjælp af tryktegn-proceduren, idet der indsættes de nødvendige case-tegn. Tekstlinien afsluttes med 1 eller 2 CAR RET.

Forsidetrykningen sker ved hjælp af proceduren COVER1, hvis deklaration er vist nedenfor:

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

```
m13:  if day < 10 then tryk({-n}, day) else tryk({-nd}, day);
      tryktegn(27);
      tryk({-dddd}, year);
      LINE(8);
      tryktekst({<File No. >});
      trykkopi({<[[>});
      LINE(14);
      LRest := LRest - 1;
      sættegn(134);
      CENTEXT(true);
      LINE(1);
      trykml(27);
      tryktekst({<GIER Calculation No.>});
      tryk({-ndddd}, calcno);
      LINE(2);
      trykml(sp1);
      tryktekst(name1);
      LINE(2);
      trykml(sp2);
      tryktekst(name2);
      LINE(100)
      end of COVER-1;
```

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

{<Calculation of TVA-Type Ammonia Converter>

og name2 er programmets forkortelse, her:

{<GIER Program TVA-1>

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

Proceduren udfører iøvrigt følgende:

1. Der læses et enkelt tegn fra strimlen for at fjerne et eventuelt gammelt tegn fra læseapparatet.

2. Værdien af beregningsnummeret, calcno, indlæses med læst-proceduren.

3. De tre ikke-lokale variable tildes de rigtige startværdier:

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

4. Perforering af 50 TAPE FEED og 60 SPACE.

5. Trykning af dato. Her staar først de tre sætninger:

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

Det blev forklaret side 206, at hver gang det oversatte ALGOL-program indlæses i maskinen, vil de rigtige datotal blive indsat i stedet for 1960, 1961 og 1962.

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

```
go_to M[month];
```

hoppes til trykning af maanedens navn. Derefter trykkes dagen med 1 eller 2 cifre, et komma og endelig aarstallet.

6. Trykning af 8 CAR RET med proceduren LINE.

7. Trykning af ordet: File No.

8. Kopiering af sagsnummeret fra inputstrimlen med:

```
trykkopi(⟨⟨[[⟨]);
```

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

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

et ekstra CAR RET-tegn. Til gengæld maa vi da med sætningen:

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

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

```
sættegn(134);
```

simulere, at vi har rykket inputstrimlen en plads tilbage til [-tegnet igen, fordi vi jo allerede har løst dette tegn een gang, nemlig som afslutning paa kopieringen af sagsnummeret.

10. Indlæsning og centertrykning af en eller flere forsidedekststrengene med CENTEXT-proceduren. Der anbringes 2 CAR RET efter hver linie.

11. Trykning af en ekstra CAR RET.

12. Trykning af 27 SPACE.

13. Trykning af den faste tekststreng:

```
{<<GIER Calculation No.>
```

14. Trykning af beregningsnummeret, calcno.

15. Trykning af 2 CAR RET.

16. Trykning af sp1 SPACE.

17. Trykning af programmets fulde navn: name 1.

18. Trykning af 2 CAR RET.

19. Trykning af sp2 SPACE.

20. Trykning af programmets forkortelse: name2.

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

Trykning af tekstlinier som sektionsoverskrifter sker med proceduren HEADLINE, hvis deklaration er:

```
procedure HEADLINE(n, out);
```

```
value n;
```

```
integer n;
```

```
label out;
```

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

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

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

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

sættegn(symb);

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

PSHIFT(n);

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

I programmeringsformularen side 211 findes en blok til trykning af forside og sektionsoverskrift. Den begynder ved etiketten: AA.

De tre nødvendige procedurer, som vi lige har omtalt, er deklareret ved:

```
comment library CENTEXT;  
comment library COVER1;  
comment library HEADLINE;
```

I sektion 1 bruges blokken til trykning af baade forside og sektionsoverskrift, men i de følgende sektioner kun til sektionsoverskrifter. Vi har derfor sætningen:

```
if sectno > 1 then go to BB;
```

som bevirker, at kaldet af COVER1-proceduren overspringes. I formularen er kaldet skrevet:

```
COVER1(      ,      ,  
  <<  
  <<
```

og dette skal udfyldes af programmøren, f.eks. saaledes:

```
COVER1(19, 31,  
  <<Calculation of TVA-Type Ammonia Converter>>,  
  <<GIER Program TVA-1>>);
```

De næste tre sætninger er:

```
for i := 0 step 1 until 39 do B[i] := - 1234;  
tromleplads := ;  
for i := 1 step 1 until do tiltromle (B);
```

De bruges til at indsætte veldefinerede tal i de tromleceller, der er forbeholdt lagring af det egentlige inputmateriale til programmet. Hvis brugeren af programmet glemmer at opgive et af inputtallene, vil programmet regne med et veldefineret tal, ikke et eller andet tilfældigt tal fra den forrige beregning. Dette forklares nærmere i næste afsnit.

Endelig kaldes overskriftproceduren:

```
BB: HEADLINE(20, ZZ);
```

som vil skifte side, hvis der er mindre end 20 linier tilbage paa siden, og som bevirker hop til etiketten: ZZ, naar stoptegnet: z mødes.

10.4. Behandling af Inputdatagrupper.

Det vil ofte være praktisk at opdele det egentlige inputtalmateriale til en beregning i flere grupper, saaledes at man f.eks. har apparatdimensioner samlet i een gruppe, driftsbetingelser i den næste gruppe, o.s.v.

Naar brugeren af programmet skal udføre en beregning, maa han udfylde en tilsvarende inputspecifikation, hvoraf der findes en kopi i vejledningen til programmet. I de fleste tilfælde begynder specifikationen som vist side 217 med tekstmateriale til forside og sektionsoverskrift. Derefter følger specifikationen til det egentlige talmateriale, som f.eks. i følgende, der ikke tilhører et bestemt program, men kun skal illustrere forholdet:

2. Rørdata:

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

3. Driftsbetingelser:

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

z Stop

Lad os antage, at vi har udfyldt inputspecifikationen saaledes:

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

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

Ff

9876

[358

[Beregningseksempel

e

[Basiseksempel

e

100 8 110 90 e

1298.3 500 23 e

e 102 ee

e 104 ee

z

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

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

e ddd 92 ee

e ddd 94 ee

eller:

e q 3 92 ee

e q 3 94 ee

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

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

udfyldning af inputspecifikationer.

Vi har derfor brug for nedenstaaende procedure READ5, som foretager indlæsning af tal med samtidig kontrol af forekomst af bogstaverne d, e og q.

```
procedure READ5(a, p, stope);  
real a;  
integer p;  
label stope;  
begin  
    integer symb;  
L1:   symb := læstegn;  
    if symb = 53 then go_to stope;  
    if symb = 52 then go_to L2;  
    if symb = 40 then p := p + læst;  
    if symb = 16  $\vee$  symb  $\geq$  1  $\wedge$  symb  $\leq$  9  $\vee$  symb = 32  $\vee$  symb = 59  
     $\vee$  symb = 155 then  
    begin  
        sættegn (symb);  
        a := læst;  
        sættegn(tegn);  
        go_to L2  
    end;  
    go_to L1;  
L2: end of READ-5;
```

De tre parametre er:

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

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

```
a := læst;
```

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

```
p := p + læst;
```

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

```
sættegn(symb);
```

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

```
sættegn(tegn);
```

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

Proceduren kan f.eks. bruges saaledes:

```
begin  
  integer i;  
  real X, Y, Z, V;  
  array A[0:3];  
  comment library READ5;  
  .....  
  .....  
AA: for i := 0 step 1 until 4 do  
    READ5(A[i], i, EE);  
EE:  X := A[0];  
     Y := A[1];  
     Z := A[2];  
     V := A[3];  
     .....  
     go_to AA;  
     .....  
end program;
```

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

Her er talmaterialet anbragt i lageret som talsættet: A. For større programmer er det mere praktisk at have det anbragt paa tromlen. Vi har tidligere nævnt, at overføring til og fra tromlen sker i grupper paa 40 tal ad gangen i maskinsprog. Det er derfor praktisk at tænke sig tromlen opdelt i saakaldte kanaler paa 40 tal hver. Vi nummererer cellerne paa hver kanal fra 0 til 39 og kanalerne fra 0 til 319. Værdien af tromleplads for celle C paa kanal K er:

$$40 \times K + C$$

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

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

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

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

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

Da vi ved den senere trykning af inputmaterialet i outputrapporten gerne vil overspringe de inputtal, som er uforandrede fra sidste sektion, er READ6 indrettet til at kontrollere hvilke tal, der er forskellige fra dem i forrige sektion. Den gemmer informationen herom i den sidste celle (39) paa tromlekanalen. Kaldes vi indholdet af denne celle for m, er m = 0, hvis alle tal er uændret fra før. Er m ≠ 0, er et eller flere tal ændret. Er indholdet i celle nr. j ændret, er der adderet 2^j til det oprin-

delige nul i celle 39. Indeholder celle 39 f.eks. tallet 43, skriver vi dette som:

$$1 \times 2^0 + 1 \times 2^1 + 1 \times 2^3 + 1 \times 2^5$$

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

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

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

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

Vi har her begrænset os til kun at bruge cellerne 0 - 28 paa tromlekanalen (+39), fordi vi ellers faar afrundingsfejl i indholdet af celle 39.

Vi kan nu se paa den sidste del af programmeringsformularen side 211. Input af et talmateriale med fire datagrupper til kanalerne 67 - 70 kan skrives som blokken:

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

Vi kan nu ogsaa forstaa meningen med de tre sætninger, som er nævnt side 226. Med fire datagrupper paa kanal 67 - 70 kan disse tre sætninger skrives:

```
for i := 0 step 1 until 39 do B[i] := -1234;  
tromleplads := 2839;  
for i := 1 step 1 until 4 do tiltromle(B);
```

Herved faar vi anbragt tallet -1234 i alle celler paa de fire kanaler 67 - 70. Dette sker lige efter forsideindlæsningen, og der vil derfor være et veldefineret sammenligningsgrundlag for READ6, ogsaa i sektion 1. Tallet -1234 er blot ment som et eller andet usandsynligt tal. Hvis der er mulighed for, at et af inputtallene virkelig kan antage denne værdi, bør man vælge et andet, da værdien ellers ikke vil blive trykt i sektion 1. Man kan ogsaa bruge tallet nul som startværdi, men det er mere sandsynligt, at det vil forekomme som normalt input. Endelig kan man bruge forskellige starttal for de forskellige inputvariable. I dette tilfælde bør det anføres i inputspecifikationen hvilke tal, der paa denne maade er indbygget i programmet, for at brugerne kan udnytte det.

Til slut omtaler vi to procedurer, som bruges ved trykning af inputmateriale med tekstforklaring. Det vil altid være klogt at lade programmet trykke værdien af alle inputtal, for at man kan se, at maskinen har benyttet de rigtige tal. Vi kan f.eks. ønske at faa de to datagrupper side 227 trykt saaledes:

RØRDATA

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

DRIFTSBETINGELSER

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

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

```
begin  
  integer mcount, mfact;  
  array DATA[0:39];  
  comment library TABLE2;
```

```
comment library T3;
TABLE2(67, 10, NP1,
  {<RØRDATA>, 36});
T3({<Rørantal>, 38, {-nddd});
T3({<Højde (m)>, 38, {-nddd.dd});
T3({<Udvendig diameter (mm)>, 25, {-nddd.dd});
T3({<Indvendig diameter (mm)>, 24, {-nddd.dd});
LINE(2);
NP1: TABLE2(68, 10, NP2,
  {<DRIFTSBETINGELSER>, 31});
T3({<Gasmængde (kgmol/h)>, 25, {-nddd00.000});
T3({<Temperatur (gr.C)>, 30, {-nddd});
T3({<Tryk (atm.abs.)>, 33, {-nnd.d0});
LINE(2);
NP2: end blok;
```

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

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

Deklarationerne er:

```
procedure TABLE2(chan, freeline, noprint, headline, space);
integer chan, freeline, space;
label noprint;
string headline;
begin
  tromleplads := 40xchan + 39;
  fratromle(DATA);
  if DATA[39] = 0 then go to noprint;
  PSHIFT(freeline);
  trykml(space);
  tryktekst(headline);
  LINE(2);
```

```
    mcount := 0;
    mfact := 1
end of TABLE-2;

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

Proceduren TABLE2 har de fem formelle parametre:

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

I proceduren T3 har vi parametrene:

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

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

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

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

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

Hvis man under ingen omstændigheder vil have sideskift midt i en tabel trykt med TABLE2, maa værdien af freeline vælges højere end antallet af elementer i datagruppen. Dette giver betydeligt papirspild, hvis der i sektion 2 eller senere kun skal trykkes en enkelt variabel i en tabel. Vi har derfor en variant af TABLE2, som udfører

```
PSHIFT(z + 3);
```

istedet for PSHIFT (freeline) og som selv tæller det faktiske antal linier, z, der skal trykkes. Her er parametren: freeline erstattet med parametren: numit, antallet af elementer i datagruppen. Proceduredeklarationen for denne variant, TABLE4, er:

```
procedure TABLE4(chan, numit, noprint, headline, space);  
integer chan, numit, space;  
label noprint;  
string headline;  
begin  
    integer cell, z, i, p;  
    tromleplads := 40xchan + 39;  
    fratromle (DATA);  
    cell := DATA[39];  
    if cell = 0 then go_to noprint;
```

```
z := 0;
for i := numit - 1 step -1 until 0 do
begin
  p := 2i;
  if cell > p then
  begin
    z := z+1;
    cell := cell - p
  end
end;
PSHIFT(z+3);
trykml(space);
tryktekst(headline);
LINE(2);
mcount := 0;
mfact := 1
end of TABLE-4;
```

De her viste tabeltrykninger gælder for endimensionale tabeller. For flerdimensionale tabeller maa man som regel programmere dette hver gang.

Til slut omtales to specielle procedurer, som bruges ved input og output fra meget store programmer. Hvis et program er blevet saa stort, at det i realiteten ikke kan være i maskinen, kan man ofte klare sig ved at dele programmet i to eller flere afsnit. Vi vil da gerne bibeholde sektionsoptdelingen med udeladelse af trykningen af tal, der er uforandret fra sidste sektion. Dette kan gøres saaledes:

Første programdel læser en ganske normal inputstrimmel, men den trykker ingen forside eller sektionsoverskrifter. Programdelen afleverer en outputstrimmel, som blot indeholder en kopi af forsidedata og sektionsoverskrifter. Denne outputstrimmel, som vi i det følgende kalder den intermediære strimmel, benyttes saa paa sædvanlig maade som inputstrimmel til den anden programdel. Input af det almindelige talmateriale sker med READ5 og READ6 i alle programdele, medens output fra programdel 1 til den intermediære strimmel sker med en særlig procedure (PRINT9), der sørger for de nødvendige stop tegn (e) og dittotegn (d), saaledes at informationen om hvilke tal, der er uændret, ikke gaar tabt i den intermediære strimmel.

Deklarationen for PRINT9 er:

```
procedure PRINT9(max);  
value max;  
integer max;  
begin  
  integer mark, mcount, mfact, pr;  
  real R;  
  array DATA[0:39];  
  chan := chan + 1;  
  tromleplads := 40*chan + 39;  
  fratromle(DATA);  
  mark := DATA[39];  
  if mark ≠ 0 then  
    begin  
      mfact := 1;  
      pr := 0;  
      for mcount := 0 step 1 until max do  
        begin  
          mfact := mfact*2;  
          if (mark - (mark:mfact)*mfact)*2 - mfact ≥ 0 then  
            begin  
              if pr:6x6 = pr then trykvr;  
              pr := pr + 1;  
              R := DATA[mcount];  
              tryk(if abs(R) < 110-10 ∨ abs(R) > 11010 then {-n.ddddd0+dd}  
                else {-n.ddddd0+d}, R);  
              trykml(1)  
            end if print  
          else tryktegn(52)  
        end for mcount  
      end if mark ≠ 0;  
      trykvr;  
      tryktegn(53)  
    end PRINT9;
```

Den formelle parameter, max, angiver det relative nummer paa det sidste element i datagruppen. I det ovennævnte tilfælde med de to inputgrupper for rørdata og driftsbetingelser kan PRINT9 anvendes saaledes:

```
begin comment trykning af intermedisær strimmel;  
  integer chan;  
  comment library PRINT9;  
  chan := 66;  
  PRINT9(3);  
  PRINT9(2)  
end trykning;
```

Her er max henholdsvis 3 og 2, fordi de to grupper som vist side 227 har dataelementerne 0-3 og 0-2. PRINT9 indeholder iøvrigt en ikke-lokal variabel: chan, som af programmøren skal sættes til første kanalnummer minus 1. Proceduren tæller 1 frem heri for hvert kald.

Kopiering af forsidedata og sektionsoverskrift sker med proceduren READ9:

```
procedure READ9(locase, OUT);  
boolean locase;  
label OUT;  
begin  
  boolean text;  
  integer i;  
  text := false;  
L1: i := lyn;  
  tryktegn(i);  
  if i = 25 ^ -, text ^ locase then  
    begin  
      trykstop;  
      tryktom(50);  
      go_to OUT  
    end if z;  
  if i = 58 then locase := true;  
  if i = 60 then locase := false;
```



```
if text then  
begin  
  if i = 64 then text := false  
end  
else  
begin  
  if i = 53  $\wedge$  locase then go_to EX;  
  if i = 6  $\wedge$  -, locase then text := true  
end;  
go_to L1;  
EX: end READ9;
```

Den tilsvarende forsideblok kan se saaledes ud:

```
AA: begin comment kopiering af forside og overskrift;  
  comment library READ9;  
  if sectno > 1 then go_to BB;  
  locase := true;  
  READ9(locase, ZZ);  
  .....  
  comment her kan vi indsætte tallet -1234 eller andre indbyggede  
  værdier;  
  .....  
  .....  
BB:  READ9(locase, ZZ)  
  end kopiering;
```

READ9 indeholder de to parametre: locase og OUT. Den logiske variable locase maa være deklareret i den yderste blok og bruges til at gemme oplysningen om case-situationen paa inputstrimlen. Etiketten: OUT, der her er indsat som ZZ, skal henvise til programmets slutning. Der hoppes hertil efter at stop-zet er mødt. Ved etiketten ZZ bør man iøvrigt programmere en udskrift paa skrivemaskinen, med oplysning om at den intermediære strimmel skal køres videre paa næste programdel.

Opdelingen i mindre programdele kan naturligvis kun gøres, hvis der ikke findes for-sætninger, som omfatter hele programmet.

10.5. Behandling af Konstante Talsæt og Tekststreng.

I programmet til trykfaldsberegning side 106 saa vi et eksempel paa, hvorledes man kan indbygge talværdierne af et todimensionalt talsæt i et program. Vi giver her et lignende eksempel af mere generel karakter.

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

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

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

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

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

Vi anvender hertil proceduren ENT3, hvis deklaration er vist nedenfor. Kun et udsnit af talmaterialet er vist.

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

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

Proceduren ENT3 virker paa samme maade, som blev vist ved D-proceduren i programmet side 106. Her er der indbygget en E-procedure med fem parametre i ENT3. E kaldes en mængde gange, idet der hver gang tælles frem i j:

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

Hvis j er lig med et af de foreliggende identifikationsnumre:

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

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

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

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

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

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

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

Tromleplads:	Element
dplace	A[0,0]
dplace + 1	A[0,1]
dplace + 2	A[0,2]
dplace + 3	A[0,3]
dplace + 4	A[0,4]
dplace + 5	A[1,0]

O.S.V.

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

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

Kaldet af TDYN12 kan se saaledes ud:

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

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

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

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

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

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

Et eksempel paa brug af PRINT7 er:

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

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

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


For hvert kald undersøges det, om:

$$j = \text{CN}[\text{count}]$$

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

$$T(\{\langle \text{Carbon dioxide} \quad \rangle\});$$

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

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

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

10.6. ALGOL Bibliotek.

Vi har flere gange set eksempler paa ALGOL-programmer, i hvilke der forekommer ordene: comment library efterfulgt af navnet paa en procedure. Meningen hermed er, at udnytte regnemaskinen som hjælp til fremstilling af programstrimlerne. Hvis der i programmet staar:

.....

```
comment library LINE;  
comment library PSHIFT;
```

ønsker vi disse to linier udskiftet med de rigtige deklARATIONER for de to procedurer.

Vi har hertil udarbejdet et specielt program i maskinsprog, som i sig indeholder ALGOL-teksten til de vigtigste procedurer. Der er ikke plads til alle procedurer i eet program, saa det findes i flere udgaver: LIB-1, LIB-2, o.s.v.

Har vi et ALGOL-program med comment library for procedurer, som alle findes i LIB-1, er fremgangsmaaden følgende. LIB-1 indlæses i maskinen, og ALGOL-programmet anbringes i læseapparatet. Ved start reperforerer LIB-1 ALGOL-programmet med samtidig indsættelse af de rigtige deklARATIONER i stedet for comment library-linierne. Det saaledes fremkomne komplette ALGOL-program kan derefter oversættes paa oversætterprogrammet.

Hvis en procedure, f.eks. PRINT7, ikke findes i LIB-1, vil programmet udskrive bemærkningen:

```
PRINT7 not available
```

paa den tilkoblede skrivemaskine. I saa fald sker der ikke noget med sætningen:

```
comment library PRINT7;
```

og man maa saa køre programmet igen paa LIB-2, o.s.v.

11. 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).
- 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.: Standard Routines for Input and Output. Intern rapport om GNL-1. Sec. Ed. (1963a).
- Kjær, J.: Thermodynamic Calculations on an Electronic Digital Computer. Akademisk Forlag, København (1963d).
- Naur, P. et al.: A Manual of GIER ALGOL. Regnecentralen (1963a).
- Naur, P.: The design of the GIER ALGOL compiler. Part I. BIT, 3, 124 - 140 (1963b).
- Naur, P. et al.: A Manual of GIER ALGOL III. Regnecentralen (1964).
- Vilstrup, H.: Lærebog i GIER-ALGOL. Akademisk Forlag, København (1963).

12. STIKORDSREGISTER

- abs, 26
- adresse, 11, 15, 124
- afrunding, 20, 42, 140, 157, 174
- ak, 194
- ALGOL-rapport, 8
- arctan, 26
- array, 27, 39, 88, 153
- begin, 23, 116
- betingelse, 12, 40, 69, 177
- blok, 115
- boolean, 127, 159

- calcno, 209
- CAR RET, 165, 186
- case, 164, 172, 182
- celle, 11, 190
- CENTEXT, 218
- checksum, 165, 187
- CLEAR CODE, 164, 186
- comment, 49
- comment library, 105, 249
- cos, 26
- COVER1, 206, 220
- cylinderrumfang, 23, 78

- d, 228
- dato, 206
- deklaration, 23, 28, 33, 38, 80, 116, 135
- dimension, 28, 39

- DISPLAY, 190
- division, 21, 54, 141
- do, 31

- e, 228
- eller(∨), 44, 134
- else, 41, 49
- end, 21, 49, 116
- END CODE, 164, 186
- ENT3, 242
- entalpi, 34, 96, 242
- entier, 26, 54, 141
- etikette, se label
- et-tal, 196
- exp, 26

- fakultet, 112
- false, 128
- Flexowriter, 15, 25
- flydende komma, 138
- forside, 214
- for-sætning, 31, 75, 144
- FORTRAN, 144, 173
- fra tromle, 196

- GIER ALGOL II, 4, 208
- GIER ALGOL III, 4, 208
- global, 99
- GNL-1, 191, 205
- go_to, 47
- grundoperation, 11
- grænser, 29

- HEADLINE, 223

- heltalsdivision, 54
- hop, se go_to
- hukommelse, 10, 191
- hulsymbol, 162
- højreside, 18

- if, 41
- ikke, se nægtelse
- implikation(=>), 45, 134
- index, 28, 39, 123
- indlæsning, 24, 143
- inputspecifikation, 68, 71, 217
- integer, 24, 138, 156
- iteration, 59

- Jensens device, 93

- kald, 80
- kommentar, se comment
- komponentnummer, 67
- kondenseret program, 208

- label, 46, 118, 151
- lager, 11, 190
- layout, 173
- LIB-1, 250
- lighedstegn, 17
- LINE, 187, 209, 212
- linieskift, 209
- ln, 26
- logisk variabel, se boolean
- LOWER CASE, 165

LRest, 209
lyn, 172
læs, 24, 153
læst, 157
læstegn, 165

maskinsprog, 14

name, 91
navn, 150
nægtelse(-), 45, 134

og(^), 45, 134
oversættelse, 15, 204
own, 119

pageno, 209
parameter, 80, 81, 102, 111
parentes, 21
polynomium, 34
potens, 19
printal, 150
PRINT7, 247
PRINT9, 239
procedure, 78
procedurefunktion, 83
programmering, 15
programmeringsformular, 211
PSHIFT, 213
PUNCH OFF, 164, 205
PUNCH ON, 165, 186, 205

q, 228

READ5, 229
READ6, 232
READ9, 240
real, 23, 138
regneudtryk, 19
rekursiv, 112, 213
relation, 45
rækkefølge af operatorer
 20, 134

sectno, 209
segmentering, 192
sideskift, 209
sign, 26
sin, 26
skiftespor, se switch
skrv, 180
skrvkopi, 182
skrvml, 189
skrvtab, 189
skrvtegn, 189
skrvtekst, 182
skrvvr, 189
SPACE, 155, 164, 207
specifikation, 80, 88, 143
spild, 26, 138
sqrt, 26
standardfunktion, 25
step, 31, 144
STOP CODE, 164, 186
string, 179
SUM CODE, 165, 187
sumfejl, 165
summation, 30, 85
switch, 135
sætning, 115
sætning, sammensat, 41, 115

sættegn, 171

T3, 235
TAB, 164, 186
TABLE2, 235
TABLE4, 237
tal, 154
talkonstant, 195
talsæt, se array
tangens, 26
TAPE FEED, 165
tast, 159
TDYN12, 245
tegn, 160, 171, 182
tekst, 181
terminator, 207
then, 41
til tromle, 196
tipotens(_n), 19
totalsystem, 139
tromle, 190
tromle ak, 199
tromleplads, 196
true, 128
tryk, 24, 62, 173
trykende, 186
trykfald, 63, 106
trykklar, 186
trykkopi, 181
trykml, 186
trykning, 24, 173
trykslut, 186
trykstop, 186
tryksum, 186
tryktab, 186
tryktegn, 182
tryktekst, 44, 181

tryktom, 186

trykvr, 24, 186

until, 31, 144

UPPER CASE, 165

value, 88, 103, 143

vandgaslignevægt, 57, 102

venstreside, 18

vent, 164

viskositet, 64

while, 149

z, 224

ækvivalens(=), 45, 134