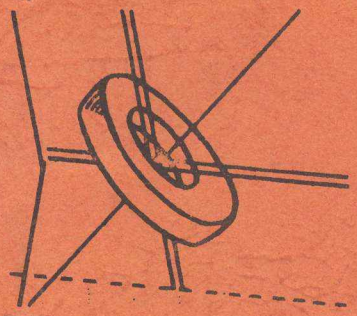


DATO, Tønder.

RUN COMAL

AF BØRGE R. CHRISTENSEN.

RUN C O M A L

er skrevet som en introduktion til sproget COMAL II, der er anden version af sproget COMAL, beskrevet af undertegnede i samarbejde med lektor Benedict Løfstedt, DAIMI, Aarhus Universitet. COMAL II blev først implementeret til kørsel på RC 7000 (NOVA 1200) under SOS af Knud Christensen og Per Christiansen ved Tønder Statsseminariums dataafdeling. Som værtssprog er benyttet DATA GENERALS's Extended BASIC, der er indeholdt i COMAL II som et ægte delsprog. Et program, som er skrevet i Extended BASIC kan altså uden modifikationer af nogen art udføres af COMAL II-fortolkeren (Extended BASIC er beskrevet i EXTENDED BASIC USER'S MANUAL, Fourth Revision - September 1973 - , som er udgivet af DATA GENERAL CORPORATION).

Nærværende introduktion er skrevet til brug for gymnasier, HF og seminarier, men kan iøvrigt bruges af alle, som ønsker at anvende COMAL II ved skrivning af programmer.

Studielektor W.Kjellberg-Christensen, Nakskov Gymnasium, har gennemlæst manuskriptet, og hans bemærkninger har givet anledning til, at mange fejl er blevet rettet og adskillige mangler udbedret. Jeg benytter lejligheden til at takke ham herfor.

Tønder, april 1976.

Børge R. Christensen.

INDHOLDSFORTEGNELSE.

KAP. I.
GRUNDLÆGGENDE COMAL SÆTNINGER.

1. Konstanter.

Et program til en datamat er et sæt instruktioner til behandling af en samling tal, bogstaver eller andre symboler. En sådan samling kalder vi en datamængde, og dens enkelte elementer benævnes data. I mange problemer består datamængden udelukkende af tal, og vi vil derfor først beskæftige os med numeriske data. I COMAL kan man også behandle ikke-numeriske data, men vi udskyder en systematisk gennemgang af dette emne til et senere kapitel.
Hvis vi har givet et bestemt tal, som fx. 13, -17.7 eller 185.05, vil vi omtale dette som en konstant. På grund af indre begrænsninger i datamaterne er det ikke tilladt at bruge enhver konstant i COMAL. I den her omtalte version må man således ikke have mere end 6 betydende cifre i en konstant. Nedenfor vil vi se, hvorledes man angiver meget store og meget små tal.
En konstant kan enten være positiv eller negativ. Hvis konstanten er positiv, kan man efter behag sætte et plus (+) foran den eller udelade det. Hvis konstanten derimod er negativ, skal der anbringes et minus (-) foran den. Skilletegnet mellem heldel og brøkdel i en decimalbrøk er et punktum (ikke et komma!!!). I COMAL kan man fx. skrive 34.678 eller -7.3, mens forsøg på at indlæse de nævnte tal ved brug af symbolerne 34,678 og -7,3 vil føre til alvorlige misforståelser. Ved skrivning af ægte decimalbrøker er det ikke nødvendigt at anbringe et nul foran decimalpunktum'et. Til angivelse af konstanten 0.567 behøver man kun anvende symbolet .567. Symboler som 0.567 eller 000.567 er dog tilladt og vil blive forstået korrekt af systemet.
Når man skal skrive meget store eller meget små tal, kan man anvende den såkaldte eksponential-notation. Som bekendt kan tallet 12000 også skrives $12 \cdot 10^3$ eller $1.2 \cdot 10^4$,

altså som en konstant multipliceret med en passende potens af 10. I COMAL angives potenser af ti ved bogstavet E efterfulgt af eksponenten. Det lige nævnte tal kan fx. skrives således: 12E3 eller 1.2E4. På lignende måde kan vi skrive .00015 som 15E-5 eller 1.5E-4. Konstanten, som følger efter E, kaldes eksponentdelen af tallet. Når systemet udskriver et tal i eksponentialnotation, står punktum'et altid på den principale plads, og eksponenten er et helt tal mellem -79 og 75.

2. Variable.

De tal og tegn, som optræder i et program, gemmes af datamaten i dennes lager. Vi vil ikke komme nærmere ind på, hvorledes dette lager er organiseret, men nævner blot, at man ved hjælp af COMAL kan indrette lageret, så det kan indeholde såvel enkelte konstanter, mængder af sammenhørende konstanter som enkelte tegn og mængder af sammenhørende tegn. I dette afsnit beskæftiger vi os kun med lagerafsnit eller lagerceller, som kan indeholde enkelte konstanter. En sådan lagercelle kan af programmøren tildeles et navn, og optræder derefter som en variabel i programmet. Et navn for en variabel kan bestå af fra ét til otte tegn, af hvilke det første skal være et bogstav, mens evt. følgende tegn kan være bogstaver eller cifre. Hvis man således i et COMAL-program skriver: LET ANTAL=5, meddeler man derved systemet, at det skal reservere en plads i lageret (en lagercelle), som under resten af programmet kan angives ved navnet ANTAL, samt at man straks ønsker tallet 5 lagret på denne plads.

3. Udtryk.

Konstanter og/eller variable forbundet med de sædvanlige regnetegn (aritmetiske operatorer) - fx. 4+5 eller Y+45 - kaldes udtryk eller formler. Når man skriver udtrykket Y+45, tilkendegiver man, at man ønsker 45 adderet til den konstant, der er lagret i den variable, hvis navn er Y. Addition og subtraktion angives ved de sædvanlige tegn:

+ og -. Multiplikation angives ved en stjerne: *, division ved en skråstreg: /, og potensopløftning ved en pil, som peger opad: ↑. Vi har således: 4*3=12, 20/5=4 og 3↑2=9. Læg mærke til, at dersom vi multiplicerer en konstant med en konstant, en konstant med en variabel eller to variable med hinanden, skal tegnet * forekomme. Man må altså ikke, således som det ofte er tilfældet i sædvanlig regnenotation, udelade gangetegnet.
Et udtryk kan indeholde flere operatorer. Disse udføres af datamaten efter et bestemt prioriteringssystem:

1. Udtrykket afsøges fra venstre mod højre, og alle potensopløftninger udføres i den rækkefølge, hvori de forekommer. Således vil udtrykket: 2-3↑2/4↑3 først blive reduceret til 2-9/4↑3 og derpå til 2-9/64.

2. Når alle potensopløftninger er udført, afsøges udtrykket atter fra venstre mod højre, og multiplikationer og divisioner udføres i den orden, hvori de er anført. Således vil udtrykket: 8*4/2*3 først blive reduceret til 32/2*3, derpå til 16*3, som til slut udregnes til 48.

3. Det nu foreliggende udtryk afsøges pånyt fra venstre mod højre, og alle additioner og subtraktioner udføres i den orden, i hvilken de forefindes. Et udtryk som fx. 2+3-4+5 reduceres først til 5-4+5, og derpå til 1+5, som sluttelig udregnes til 6.

Ved at anvende disse regler på udtrykket: 2+3*4↑2/6-7 ser vi, at dette først reduceres til 2+3*16/6-7, derpå til 2+48/6-7, så til 2+8-7 og endelig til 3. Parenteser kan bruges til at ændre på rækkefølgen af udregningerne i et udtryk efter matematikkens sædvanlige regler. Et udtryk må gerne indeholde flere parenteser, og parenteser inden i parenteser er også tilladt. Den almindelige regel er, at indholdet af de inderste parenteser udregnes først under anvendelse af reglerne 1., 2. og 3. ovenfor. Når indholdet af en parentes er udregnet, slettes parentesen, og processen fortsættes, indtil hele

udtrykket er udregnet. Vi har således, at 12/3*2=8, mens 12/(3*2)=12/6=2.
En parentes, som mangler begyndelse eller afslutning, vil bevirke, at systemet afgiver en fejlmelding.
Vær opmærksom på betydningen af et minus-tegn foran et udtryk. Medmindre man har anvendt en parentes, som kan ændre udregningernes rækkefølge, vil minus-tegnet blive prioriteret som subtraktion efter de regler, vi har anført ovenfor. Således er -4↑2=-(4↑2)=-16, hvorimod (-4)↑2 udregnes til 16.

4. Opbygningen af et program.

Vi vil nu begynde at se på, hvorledes et program opbygges, og vi vil betragte følgende eksempel (betydningen af de enkelte sætninger forklares nedenfor):

```
10 LET TAL1=5; TAL2=6*TAL1+3
20 PRINT TAL2
```

Det lille program består af to linjer, der er forsynet med hver sit linjenummer. Et sådant linjenummer skal altid stå først på linjen og kan være et helt tal fra og med 1 til og med 9999.
Linjenumrene bestemmer i hvilken rækkefølge datamaten bringer sætningerne til udførelse. Datamaten læser linjer i et program, efterhånden som de bliver skrevet, og ordner dem derpå internt efter voksende linjenumre. Det er således ikke nødvendigt at skrive de enkelte linjer op i den rækkefølge, hvori de ønskes udført, blot man sørger for, at de linjer, der skal udføres først, har de laveste linjenumre, og de linjer, der skal udføres sidst, har de højeste linjenumre. Hvis to linjer i et program har samme linjenummer, vil den linje, der er skrevet først blive slettet og erstattet med den anden. Disse regler er meget nemme at anvende, idet de tillader os både at rette allerede skrevne linjer og at indsætte nye under skrivningen af programmet. Det er en god skik at nummerere linjerne i et program således: 10, 20, 30, ... , idet man derved skaffer sig god plads til at man senere kan indsætte

supplerende linjer i programmet, hvis man får brug for det. En linje kan slettes ved, at man skrive en tom sætning med samme linjenummer, som den linje, der ønskes slettet.
Når man er færdig med at skrive en linje, trykker man på den taste, der er mærket RETURN; så vil systemet læse sætningen og gemme den, hvis den er fejlfri. Hvis der er fejl i sætningen, afgiver systemet en fejlmelding, og sletter derpå linjen af sit lager.
Hvis man er ved at skrive en linje og opdager, at man har lavet en fejl, som er så graverende, at man ønsker at skrive sætningen om, trykker man blot på den taste, der er mærket ESC (escape), så vil systemet skifte linje uden at læse den sætning, man netop har skrevet. Hvis man ønsker at rette i den linje, man er ved at skrive, trykker man på den taste, der er mærket RUBOUT. Systemet vil da slette det tegn, man netop har indtastet. Hvis man trykker to gange på RUBOUT, slettes de to sidst indtastede tegn osv. På den måde kan man slette så mange tegn, man ønsker, og skrive andre tegn i stedet. Hver gang man trykker på RUBOUT, kvitterer systemet med at skrive tegnet: ←, hvis man skriver på en Teletype, mens blinkeren rykker tilbage, hvis det er en skærmterminal, man skriver fra.

5. LET-sætningen.

LET-sætningen bruges, når man ønsker at anbringe en bestemt konstant i en given lagercelle. Man udtrykker også dette ved at sige, at man tildeler eller tilskriver en variabel en bestemt værdi. Sætningen

```
LET ANTAL=25
```

bevirker, at den variable ANTAL tildeles værdien 25 (konstanten 25 lagres i cellen ANTAL). Sætningen bruges også, når man ønsker at tildele en variabel resultatet af en udregning. Således vil sætningen:

```
LET Y=2*3/4
```

bevirke, at konstanten 1.5 lagres i cellen Y.
Et udtryk på højre side af lighedstegnet kan indeholde variable, når blot disse har fået tildelt en værdi tidligere i programmet.
For tydeligt at forstå, hvad dette vil sige, betragter vi sætningen

```
LET X=Y+3
```

Denne sætning kan læses således: "addér 3 til indholdet af Y og anbring resultet i X". Denne udlægning indebærer, at lighedstegnet ikke får helt samme betydning som i den sædvanlige matematik. Således er sætningerne i linjerne:

```
10 LET X=5
20 LET X=X+16
```

helt korrekte i COMAL og vil resultere i, at den variable X har fået tildelt værdien 21, når linje 20 er udført. Sætningen i linje 20 siger nemlig, at man ønsker 16 lagt til indholdet i X - som er 5 - og resultatet anbragt i X. Man må her være opmærksom på, at det oprindelige indhold i X går tabt. Man siger også, at det bliver overskrevet. (Sammenlign med en båndoptager: når et bånd indspilles, bliver det, som i forvejen står på båndet, slettet).
Af det netop anførte kan man slutte, at sætningen

```
LET 3=X
```

er forkert opbygget, da 3 ikke er tilladt som navn for en lagercelle (variabel). En sådan sætning vil fremkalde en fejlmelding fra systemet. Vær omhyggelig med at anbringe de variable i den rigtige rækkefølge, når en konstant skal skrives fra én celle ind i en anden. Sætningen

```
LET X=Y
```

tildeler den variable X samme værdi, som tidligere er blevet tildelt Y. Konstanten Y påvirkes ikke heraf, men den oprindelige konstant i X går tabt (bliver overskrevet).

Sætningen

```
LET Y=X
```

iværksætter naturligvis den omvendte operation.
I COMAL er det tilladt at skrive flere tildelinger på samme linje; man skal blot adskille de enkelte tildelinger med tegnet ; (semikolon). Således vil sætningen

```
LET X=5; Y=8/X; A=8*X+5
```

virke som tre LET-sætninger efter hinanden.

6. READ, DATA og RESTORE sætningerne.

LET-sætningen gør det muligt for os at tildele en variabel en given værdi, men den bliver besværlig at anvende, hvis vi skal indlæse et stort antal konstanter.
Denne vanskelighed kan vi overvinde ved at bruge DATA og READ sætningerne. Lad os tænke os, at vi ønsker konstanterne 10, 15, 20 og 30 tildelt de variable A, B, C og D hhv. Dette kan udføres af programmet:

```
10 DATA 10,15,20,30
20 READ A,B,C,D
```

Disse sætninger bevirker, at den variable A får tildelt værdien 10, B får tildelt værdien 15, C får tildelt værdien 20 og D får tildelt værdien 30. Vi kan opnå det samme resultat ved at bruge programmet:

```
50 READ A
55 READ B
57 READ C,D
10 DATA 10,15
20 DATA 20,30
```

Vi kan have lige så mange DATA-sætninger, som vi ønsker, i et program, og de kan anbringes hvor som helst i programmet. De enkelte konstanter skal adskilles ved kommaer, men der må ikke anbringes et komma før den første konstant og efter den sidste konstant i listen efter DATA.

For at forstå sammenhængen mellem DATA og READ-sætningerne, må vi se lidt på, hvad der sker i datamaten, når disse sætninger bringes til udførelse. Først opsøger systemet samtlige DATA-sætninger i programmet og opstiller alle de konstanter, der optræder i disse sætninger, i én enkelt liste. I denne liste står konstanterne opført i den orden, i hvilken de forekommer i programmet. Endvidere optræder internt i datamaten en "viser", som peger på den første konstant i listen. Når datamaten under udførelse af programmet møder en READ-sætning, får den første variabel i denne sætning tildelt værdien af den konstant, viseren peger på, og samtidigt flyttes viseren hen til den næste konstant i rækken. Hvis der forekommer endnu en variabel i READ-sætningen, får denne tildelt værdien af den konstant, viseren nu peger på, og samtidig rykker viseren frem til den næste konstant på listen, og således går det videre. Når den sidste variabel i READ-sætningen har fået tildelt en værdi, bliver viseren stående, indtil der eventuelt optræder en ny READ-sætning, hvorefter tildelingen fortsættes på den netop beskrevne måde. Hvis mængden af konstanter på listen bliver brugt op, inden samtlige variable i READ-sætningerne har fået tildelt en værdi, standser datamaten udførelsen af programmet og afgiver en melding om, at den ikke har flere data. Hvis man tænker på dette billede af en liste med konstanter og en viser, der peger dem ud, når man arbejder med DATA og READ-sætninger, skulle det ikke volde større vanskeligheder at anvende dem på rette måde.

Eksempel 6.1.
Lad os betragte programmet

```
10 READ A,B,A1
20 DATA 5,7,-10
30 DATA 105
```

Da sætningerne i linje 20 og 30 er DATA-sætninger, udføres de først, selvom de har højere numre end READ-sætningen. Systemet opstiller en liste med konstanterne fra samtlige DATA-sætninger og sætter en viser på den første:

```
↓
5  7  -10  105
```

Når programmet udføres, møder systemet READ-sætningen, og den første variabel A får tildelt værdien 5, hvorefter viseren flyttes hen til konstanten 7:

```
   ↓
5  7  -10  105
```

Derefter får B tildelt værdien 7 og viseren flyttes:

```
      ↓
5  7  -10  105
```

Endelig tildeles den variable A1 værdien -10 og viseren flytter hen til konstanten 105. Her bliver viseren stående, indtil der optræder evt. andre READ-sætninger. □

Det kan være, man ønsker at bruge de samme data mere end én gang i et program. I så fald kan man gøre brug af sætningen RESTORE, som har til følge, at viseren går tilbage til den første konstant på listen. Sætningen kan bruges uden hensyn til, hvor viseren befinder sig.

Eksempel 6.2.
Lad os tænke os, at programmet fra eksempel 6.1 fortsættes således:

```
40 RESTORE
50 READ P,Q,R4,S
```

Når linje 50 er udført, vil de variable P, Q, R4 og S have fået tildelt konstanterne hhv. 5, 7, -10 og 105 (forklar dette). □

7. INPUT-sætningen.

Når man arbejder fra en terminal, har man mulighed for at indlæse konstanter direkte fra tastaturet ved at anvende sætninger som fx.:

```
INPUT ANTAL,PRIS,NUMMER
```

Når programmet, som indeholder denne sætning, udføres,

vil udførelsen standse, når sætningen er fortolket, og systemet skriver ? på terminalen. Dette spørgsmålstegn fortæller brugeren, at der nu skal indtastes nogle talværdier, og i det aktuelle tilfælde skal der åbenbart indtastes tre konstanter, som vil blive indlæst i hhv. ANTAL, PRIS og NUMMER. Man kan fx. skrive:

```
88, 85.5, 6
```

og derpå trykke på RETURN-tasten. Derefter fortsætter udførelsen af programmet med ANTAL=88, PRIS=85.5 og NUMMER=6. Hvis man ikke selv har skrevet programmet, eller det er længe siden, man har skrevet det, kan det volde vanskeligheder at finde ud af, hvad systemet mener med spørgsmålstegnet. Der står fx. ikke noget om, hvor mange tal, programmet skal bruge som inddata. I så fald kan man se efter i programlisten (programteksten), men man har også den mulighed at anbringe en vejledende tekst i INPUT-sætningen. Det kunne fx. se således ud:

```
INPUT "ANTAL, PRIS OG NUMMER: ",ANTAL,PRIS,NUMMER
```

Når programudførelsen når til denne sætning, skriver systemet ikke mere ?, men derimod den tekst, som er skrevet i anførselstegn. Man får altså udskriften:

```
ANTAL, PRIS OG NUMMER:
```

hvorpå systemet går i venteposition. Brugeren kan nu tydeligt se, at der skal indtastes tre konstanter, og at den første vil blive indlæst i ANTAL, den anden i PRIS og den tredje i NUMMER. Mellem anførselstegnene kan man skrive hvad som helst undtagen naturligvis anførselstegnet ("), der anvendes som termineringstegn for teksten.

8. PRINT-sætningen.

Vi har set, hvorledes man får indlæst konstanter i datamaten, og hvordan man får den til at udføre visse regneoperationer. Vi skal nu se, hvorledes man får datamaten til at meddele os sine resultater. Hertil benytter man

PRINT-sætningen. Denne sætning består af ordet PRINT efterfulgt af en liste over det, man ønsker udskrevet. Sætningen

```
PRINT ANTAL,SALGSPR
```

vil bevirke, at systemet udskriver de konstanter, som er tildelt de variable ANTAL og SALGSPR (indholdet af lagercellerne ANTAL og SALGSPR).
Foruden de variable kan PRINT-sætninger indeholde udtryk, konstanter og tekst. Hvis der optræder et udtryk i en PRINT-sætning, vil systemet udregne den og udskrive resultatet. En tekst er en hvilken som helst tegnfølge, som er sat i anførselstegn.

Eksempel 8.1.
Følgende PRINT-sætning:

```
PRINT "RESULTATET AF UDREGNINGEN ER:";5*X/100
```

indeholder dels en tegnfølge - en tekst - og dels et udtryk. Teksten og udtrykket er adskilt ved et semikolon. Lad os tænke os, at X har værdien 7. Når sætningen bringes til udførelse, vil udtrykket blive udregnet til .35, og man vil få følgende udskrift:

```
RESULTATET AF UDREGNINGEN ER: .35
```

Virkningen af tegnet semikolon er altså, at der sættes ét mellemrum mellem teksten og den udregnede konstant. Man kan også anbringe et komma mellem teksten og udtrykket:

```
PRINT "RESULTATET AF UDREGNINGEN ER:",5*X/100
```

Systemet vil nu give os følgende udskrift, hvis format bestemmes af, at linjen er opdelt i en række kolonner på hver 14 tabulatorslag. Til illustration af opdelingen er er kolonnerne markeret under udskriften:

```
RESULTATET AF UDREGNINGEN ER:                .35
↑             ↑             ↑              ↑
1            14            28             42
```

Den måde, på hvilken man får udskrevet de enkelte dele

af PRINT-sætningen, bestemmes af formatstyringen i COMAL. I de mest udviklede versioner af COMAL er denne formatstyring ret kompliceret og indeholder en række tekniske detaljer. Vi vil udskyde behandlingen af disse til et senere kapitel, og på dette sted nøjes med at give nogle eksempler på brugen af de simple PRINT-sætninger.

Eksempel 8.2.
Programmet

```
10 LET PRIS=3.8; MOMSPCT=15
20 INPUT "VAREMAENGDE (KG.): ",VAEGT
30 PRINT "VAREN UDEN MOMS KOSTER: ";VAEGT*3.8;"KR."
40 MOMSFAK=(100+MOMSPCT)/100
50 PRINT "VAREN MED MOMS KOSTER: ";VAEGT*3.8*MOMSFAK;"KR."
```

Når dette program bliver kørt, fremkommer først udskriften:

```
VAREMAENGDE (KG.):
```

Som omtalt i afsnit 7 skal operatøren indtaste et tal og trykke på RETURN-tasten. Vi kan tænke os, at vedkommende har skrevet 20. Den variable VAEGT har altså fået tildelt konstanten 20. Datamaten fortsætter derpå med resten af programmet, og vi får følgende udskrift:

```
VAREN UDEN MOMS KOSTER:  76 KR.
VAREN MED MOMS KOSTER:  87.4 KR.
```

□

Eksemplet belyser følgende: Vi kan blande tekster og udtryk i en PRINT-sætning. I linje 30 har vi således sammenstillingen: tekst; udtryk; tekst, og det samme gælder linje 50. Den tomme PRINT-sætning i linje 40 bevirker, at vi får et ekstra linjeskift mellem de to udskrevne linjer (datamaten skriver faktisk en tom sætning!).

Eksempel 8.3.
Lad os betragte programmet

```
10 PRINT "OLE OLSEN KØRER IGEN"
20 PRINT "OG HAR NU FAAET SIN EGEN BANE I VOJENS."
```

Når dette program udføres, får man følgende udskrift:

```
OLE OLSEN KØRER IGEN
OG HAR NU FAAET SIN EGEN BANE I VOJENS.
```

Hvis man ændrer linje 10 til:

```
10 PRINT "OLE OLSEN KØRER IGEN ";
```

(hvori består ændringerne?), får man følgende udskrift:

```
OLE OLSEN KØRER IGEN OG HAR NU FAAET SIN EGEN BANE I VOJENS.
```

□

Eksemplet viser, at tegnet semikolon (;) i slutningen af en PRINT-sætning bevirker, at der ikke skiftes linje før den næste PRINT-sætning udføres. Ved hjælp af semikolon kan man altså sammenkæde udskriften fra to eller flere PRINT-sætninger. Et komma har en lignende virkning, men på tilsvarende måde som i eksempel 8.1 får man en større afstand mellem udskriften fra de to linjer.

9. Kommandoerne RUN, LIST, NEW, AUTO og RENUMBER.

Når man har skrevet sit program færdigt og mener, det er fejlfrit, er man klar til at foretage en prøvekørsel af det. Man skriver da, uden linjenummer, ordet RUN fra tastaturet og trykker derpå på RETURN-tasten. Datamaten vil da udføre programmet, hvis der ikke er fejl i det. Der findes visse fejltyper, som først kan opdages under kørslen. Hvis programmet er behæftet med sådanne, vil udførelsen standse og en melding blive udskrevet. Når programmet er udført, skriver systemet END AT xxxx, hvor xxxx angiver nummeret på den linje, der er udført som den sidste.
Hvis man ønsker at få en udskrift af sit program, skriver man ordet LIST, uden linjenummer, fra tastaturet og trykker på RETURN-tasten. Systemet vil da udskrive en liste

over programmet. Denne liste kan man undersøge nærmere og rette eventuelle fejl i programmet eller foretage tilføjelser som beskrevet i afsnit 4.
Ord som RUN og LIST, der skrives uden linjenummer og udføres straks, men ikke gemmes i datamaten på samme måde som et program, kaldes kommandoer. Flere kommandoer er nævnt nedenfor, og senere beskrives andre.
Hvis man ønsker at slette samtlige programlinjer, man har indlæst i lageret, bruger man kommandoen NEW. Systemet vil da fjerne alle henvisninger til det program, man ønsker slettet, og man kan begynde at arbejde med et nyt program uden at risikere, at det bliver blandet med uønskede sætninger fra gamle programmer.
Når man indtaster et nyt program, til hvilket man naturligvis har skrevet en velgennemtænkt kladde, kan man benytte en kommando, som bevirker, at systemet selv sætter linjenumre under indtastningen. Hvis man benytter kommandoen AUTO, vil systemet svare med at skrive:

0010

og vente på at brugeren skal indtaste den eller de sætninger, der skal stå i linjen. Når dette er sket, og brugeren trykker vognretur, skriver systemet:

0020

og venter igen på brugeren. Således fortsættes med numrene 0030, 0040, osv. indtil brugeren trykker på ESC, hvorved systemet går tilbage til den sædvanlige tilstand, i hvilken brugeren selv skal indtaste numrene. Kommandoen kan modificeres, så brugeren selv bestemmer hvilket linjenummer, systemet skal skrive først og med hvilke mellemrum numrene skal angives. Hvis vi fx. ønsker, at programmet eller programafsnittet skal begynde med linje nummer 100 og i øvrigt nummereres således: 100, 105, 110, 115 osv., skriver vi kommandoen AUTO 100,5. Man kan også skrive fx. AUTO 20, og i så fald vil systemet sætte linjenumre på denne måde: 0020, 0040, 0060 osv.
Linjenumrene i et program kan ændres efter at programmet

er skrevet færdigt ved at man bruger kommandoen RENUMBER. Når brugeren afgiver denne kommando, bliver linjerne i programmet automatisk nummereret således: 10, 20, 30 osv., uanset hvilke numre, de i forvejen har haft. RENUMBER kan modificeres på samme måde som AUTO. Hvis man fx. skriver RENUMBER 5000,10, bliver programmets linjer nummereret således: 5000, 5010, 5020 osv., og hvis man skriver fx. RENUMBER 30, bliver linjenumrene således: 30, 60, 90 osv.

RESUMÉ

I dette kapitel har vi hørt om:

Konstanter, variable, udtryk, aritmetiske operatorer, LET-sætninger, READ-sætninger, DATA-sætninger, RESTORE-sætningen, INPUT-sætninger og kommandoerne RUN, LIST, NEW, AUTO og RENUMBER.

LET-sætninger er opbygget således:

LET var_1=<$udtr_1$>; var_2=<$udtr_2$> ; ...; var_n=<$udtr_n$>

hvor var_1, var_2, ..., var_n er en række variabelnavne, mens <$udtr_1$>, <$udtr_2$>, ..., <$udtr_n$> er konstanter, variable eller udtryk. En LET-sætning kan indeholde så mange tildelinger, som linjelængden tillader.

DATA-sætninger er opbygget således:

DATA $konst_1$, $konst_2$, ..., $konst_n$

hvor $konst_1$, $konst_2$, ..., $konst_n$ er konstanter (tal).

READ-sætninger er opbygget således:

READ var_1, var_2, ..., var_n

hvor var_1, var_2, ..., var_n er en række variabelnavne.

RESTORE-sætningen er opbygget således:

RESTORE

og bevirker, at viseren i datalisten går tilbage til det første element i listen.

INPUT-sætninger er opbygget således:

INPUT "<tekst>", var_1, var_2, ..., var_n

hvor <tekst> er en vilkårlig række af tegn, blandt hvilke " dog ikke må forekomme, og var_1, var_2, ..., var_n er en række variabelnavne. Hvis den vejlendede tekst udelades, ser INPUT-sætningen således ud:

INPUT var_1, var_2, ..., var_n

og man bemærker, at der ikke er noget komma (,) foran det første variabelnavn i listen.

PRINT-sætninger er opbygget således:

PRINT xx,(;) yy,(;) ...,(;) zz

hvor xx, yy og zz er konstanter, variable, udtryk eller tekster i anførselstegn. Med ,(;) er vist, at skilletegnet efter eget valg kan være komma eller semikolon. En mere detaljeret beskrivelse af de to tegns virkning findes på side 11. PRINT-sætninger kan afsluttes med , eller ; hvorved et linjeskift underbindes.

Kommandoen RUN bevirker, at programmet udføres, kommandoen LIST bevirker, at der udskrives en programliste og kommandoen NEW bevirker, at alle hidtil indlæste programlinjer slettes.
Kommandoen AUTO bevirker, at systemet selv sætter linjenumre under indtastningen af programmet. Nummereringen bliver således: 10, 20, 30 osv., indtil brugeren trykker på ESC. Kommandoen

AUTO mm,nn

hvor mm og nn er hele, positive tal, bevirker at systemet sætter linjenumre således: mm, mm+nn, mm+2nn osv., indtil brugeren trykker ESC.
RENUMBER-kommandoen bevirker, at systemet nummererer linjerne i et allerede skrevet program således: 10, 20, 30 osv, uanset hvilke numre linjerne i forvejen har haft. Kommandoen

RENUMBER mm,nn

virker analog med AUTO mm,nn.

KAP. II.
FORGRENINGER OG LØKKER.

10. Relationsoperatorer og logiske operatorer.

De COMAL-sætninger, vi skal gennemgå i dette kapitel, er nødvendige, så snart vi bevæger os ud over de allermest banale beregningsopgaver. Sætningerne gør det muligt for os at modificere udførelsen af et program ved fx. at bruge forskellige dele af programmet eller gentage dele af det, afhængig af om nærmere angivne betingelser er opfyldt eller ej.

Eksempel 10.1.
Ved salg af en bestemt vare opkræver et firma ekspeditionsgebyr for alle ordrer, som er under 100 kr. Dette ekspeditionsgebyr er på 15 kr. Den pågældende vare sælges i bestemte færdigpakkede enheder. Vi ønsker at skrive et program, der med antal solgte enheder som inddata udskriver det beløb, kunden skal betale. Enhedsprisen skal anføres i en LET-sætning, således at man let kan rette den ved prisændringer.
Det ønskede program kan se således ud:

```
10 LET PRIS=13.85
20 INPUT "ANTAL SOLGTE ENHEDER: ",ANTALENH
30 LET SALGSPR=ANTALENH*PRIS
40 IF SALGSPR<100 THEN DO
50   PRINT "SALGSPRIS INCL. GEBYR: ";SALGSPR+15;"KR."
60 ELSE
70   PRINT "SALGSPRIS: ";SALGSPR;"KR."
80 ENDIF
```

I linje 10 læses enhedsprisen ind i den variable PRIS, og i linje 20 vil systemet vente på at få indtastet antallet af solgte enheder, hvorpå udførelsen af programmet går videre med udregningen af salgsprisen i linje 30. For at forstå resten af programmet vil vi betragte linje 40, linje 60 og linje 80. Disse tre linjer indeholder sætningerne hhv. IF SALGSPR<100 THEN DO, ELSE og ENDIF, som virker på følgende måde: Hvis det er sandt, at indholdet i SALGSPR er mindre end 100, udføres sætningen i

linje 50, ellers - altså hvis indholdet i SALGSPR ikke er mindre end 100 - udføres sætningen i linje 70. Dette forløb er netop det af brugeren ønskede. Hvis salgsprisen er mindre end 100 kr., lægges 15 kr. til denne; men hvis salgsprisen ikke er mindre end 100 kr., lægges ikke noget ekspeditionsgebyr til.
□

Sætninger, hvis indhold er afgørende for, hvorledes programmets øvrige dele afvikles, kaldes styresætninger. Den overordnede rolle, styresætningerne spiller, viser sig i programlisten ved, at de styrede sætninger (i eks. 10.1 linje 50 og 70) er indrykkede i forhold til styresætningerne.
Styresætningen i eksemplets linje 40 indeholder det åbne udsagn: SALGSPR<100, og det er åbenbart sandhedsværdien af dette udsagn, der er afgørende for, hvilken virkning de tre styresætninger får. Tegnet: <, som forekommer i det åbne udsagn, kaldes en relationsoperator. I COMAL har vi seks relationsoperatorer til rådighed. Disse operatorer og deres betydning er anført herunder:

=	er lig med
<	er mindre end
>	er større end
<=	er mindre end eller lig med
>=	er større end eller lig med
<>	er forskellig fra

De tre første symboler svarer helt til de symboler, vi er vant til i matematikken. De tre sidste afviger lidt fra det sædvanlige på grund af de begrænsninger, tastaturet pålægger os.

Eksempel 10.2.
Vi vender tilbage til problemet fra eksempel 10.1. Firmaets ledelse vedtager nemlig, at kontokunder ikke skal betale ekspeditionsgebyr. Programmet må derfor ændres. Efter at have overvejet situationen, skriver programmøren følgende COMAL-program:

```
10 LET PRIS=13.85
20 INPUT "ANTAL SOLGTE ENHEDER: ",ANTALENH
30 INPUT "HAR KUNDEN KONTO HOS OS (JA=1, NEJ=0) ",KONTO
40 LET SALGSPR=ANTALENH*PRIS
50 IF SALGSPR<100 AND KONTO=0 THEN DO
60   PRINT "SALGSPRIS INCL. GEBYR: ";SALGSPR+15;"KR."
70 ELSE
80   PRINT "SALGSPRIS: ";SALGSPR;"KR."
90 ENDIF
```

Styresætningen i linje 50 indeholder nu et sammensat udsagn. Idet AND betegner den logiske konjunktion, har vi, at udsagnet SALGSPR<100 AND KONTO=0 er sandt, netop når SALGSPR er mindre end 100 og kunden ikke er kontokunde. Hvis derimod salgsprisen er mere end 100 kr. eller kunden ikke har konto hos firmaet, er det sammensatte udsagn falsk, og der lægges ikke noget ekspeditionsgebyr til salgsprisen. □

Ordet: AND, som optræder i det sammensatte udsagn i eksemplets linje 50, betegner en logisk operator. I COMAL har vi to logiske operatorer, nemlig

AND, der angiver det logiske "og", samt
OR, der angiver det logiske "eller".

Vi minder om, at to udsagn, der er sammensat med AND, giver et sandt udsagn, hvis og kun hvis begge udsagn er sande, og at to udsagn, der er sammensat med OR, giver et falsk udsagn, hvis og kun hvis begge udsagn er falske.

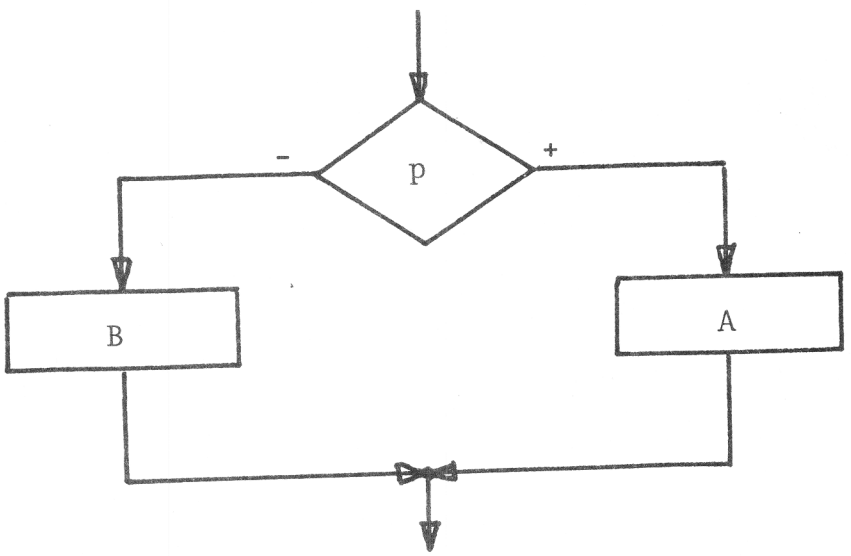
11. Programmer med forgreninger.

I eksempel 10.1 og 10.2 havde vi at gøre med to programmer, hvor bestemte dele kun blev udført under visse betingelser. I begge tilfælde blev udførelsen bestemt af tre sammenhørende styresætninger, nemlig:

```
IF p THEN DO
  [---A---]
ELSE DO
  [---B---]
ENDIF
```

hvor p betegner et - simpelt eller sammensat - åbent

eller sammensat - åbent udsagn. Sætningerne virker på følgende måde: Hvis udsagnet p er sandt, udføres linjen eller linjerne mellem IP p THEN DO og ELSE, og hvis p er falsk, udføres linjen eller linjerne mellem ELSE og ENDIF.
Forløbet af et programafsnit, som indeholder IF p THEN DO, ELSE og ENDIF, kan illustreres ved følgende rutediagram:



A er det programafsnit, der beskrives ved linjerne mellem IF p THEN DO og ELSE, og B er det programafsnit, der beskrives ved linjerne mellem ELSE og ENDIF.

Eksempel 11.1.
Ved fotokopiering på en bestemt maskine betales kopiafgiften efter følgende regler: For de første 5 kopier er afgiften 36 øre pr. stk., for de næste 20 kopier (altså fra og med nr. 6 til og med nr. 25) er afgiften 15 øre pr. stk., og for de overskydende kopier er afgiften 6 øre pr. stk. Vi ønsker at skrive et program, der med antal kopier som inddata kan udregne og udskrive den samlede afgift for kopierne (i kr.).
Det bemærkes, at de fleste serier er på mellem 10 og 20 kopier for den institution, hvor maskinen er installeret.
Følgende program kan løse opgaven:

```
 10 INPUT "ANTAL KOPIER: ",ANTAL
 20 IF ANTAL>=6 AND ANTAL<=25 THEN DO
 30   KOPIAFG=180+(ANTAL-5)*15
 40 ELSE
 50   IF ANTAL<6 THEN DO
 60     KOPIAFG=36*ANTAL
 70   ELSE
 80     KOPIAFG=480+(ANTAL-25)*6
 90   ENDIF
100 ENDIF
110 PRINT "AFGIFTEN FOR";ANTAL;"KOPIER ER: ";KOPIAFG/100;"KR."
```

I dette program er vist, hvorledes man kan have en forgrening inden i en forgrening. Den første forgrening indledes med linje 20. Hvis det er sandt, at ANTAL er større end eller lig med 6 og mindre end eller lig med 25, sættes KOPIAFG lig med 180 (for de første 5 kopier) plus prisen for de øvrige kopier, som er 15 øre pr. stk. Når denne kopiafgift er udregnet, fortsættes udførelsen af programmet med sætningen efter det yderste ENDIF, altså i linje 110, og resultatet udskrives.
Hvis antallet af kopier ikke er større end eller lig med 6 og mindre end eller lig med 25, udføres den del af programmet, som ligger mellem linje 40 og linje 100. Her skelnes atter mellem to tilfælde: Antal kopier mindre end 6 eller antal kopier større end 25. Hvis antallet er mindre end 6, udføres sætningen i linje 60 (svarende til 36 øre pr. kopi), og ellers udføres sætningen i linje 80. Dette programafsnit afsluttes med det inderste ENDIF. Da institutionens forbrug som regel ligger i serier på mellem 10 og 20 kopier, vil det åbne udsagn i styresætningen i linje 20 ofte være sandt, og i så fald udføres afsnittet fra og med linje 50 til og med linje 90 slet ikke, og det er klart, at udførelsen af programmet bliver hurtigere, desto færre linjer, der skal udføres.
□

Eksemplet viser, hvorledes man kan have flere forgreninger inden i hinanden. I COMAL II kan man have indtil 7 forgreninger inden i hinanden. Dette er tilstrækkeligt til de fleste formål.

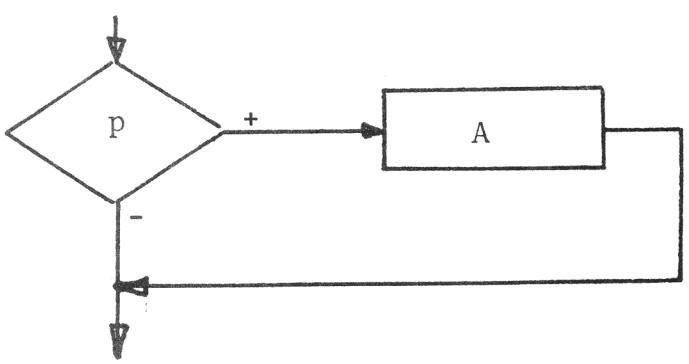
Eksempel 11.2.
Ved administrativ databehandling har man ofte brug for at sortere en datamængde, fx. tal efter størrelse eller navne i alfabetisk orden. Det følgende program udfører den simplest tænkelige sortering, idet det sørger for, at to indlæste forskellige tal bliver udlæst således, at det mindste af tallene altid står først.

```
10 INPUT "TO FORSKELLIGE TAL: ",A,B
20 IF A>B THEN DO
30   LET H=A; A=B; B=H
40 ENDIF
50 PRINT A,B
```

Programmet indeholder styresætningerne IF A>B THEN DO og ENDIF, men ikke noget ELSE. Hvis A er større end B, udføres sætningerne i linje 30, og resultatet er, at konstanterne i A og B ombyttes, således at det mindste af tallene kommer til at stå i A. Hvis det mindste af tallene i forvejen står i A, er der ingen grund til at foretage sig andet end at skrive tallene ud, og dette udføres i linje 50.
□

Som vist i eksemplet, kan styresætningerne IF p THEN DO og ENDIF optræde uden alternativet ELSE. I så fald udføres linjen eller linjerne mellem IF p THEN DO og ENDIF, hvis og kun hvis p er sand, og iøvrigt går udførelsen af programmet under alle omstændigheder videre med linjen efter ENDIF.
Strukturen af denne styring er vist i rutediagrammet herunder. Ved siden af diagrammet er programstrukturen vist:

```
IF p THEN DO
  [---A---]
ENDIF
```



12. Programmer med løkker.

Ved de programmer, vi indtil nu har betragtet, er ingen delproces blevet udført mere end én gang under udførelsen af totalprocessen. Vi skal nu se på programmer til udførelse af processer, hvor én eller flere dele udføres flere gange under afviklingen af programmet. Der er tale om to hovedtyper, hvis struktur vil fremgå af det følgende.

Eksempel 12.1.

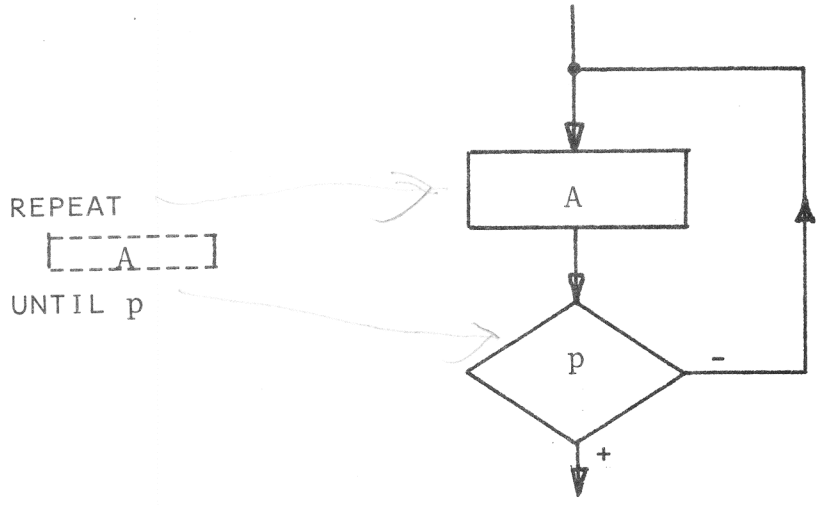
En mand ønsker at spare op til udbetalingen i et hus. Denne udbetaling er på 25.000 kr., og manden kan undvære 2000 kr. pr. kvartal. Dette beløb indsætter han i en bank, som tilskriver renter hvert kvartal. Den pågældende bank giver 12% p.a. for den type opsparing, manden præsterer. Vi ønsker et program, som kan udregne og udskrive, hvormange kvartaler, det vil vare, inden manden har penge nok til udbetalingen. Programmet kan se således ud:

```
10 LET KAPITAL=0; TEL=0
20 REPEAT
30   LET KAPITAL=KAPITAL*103/100+2000; TEL=TEL+1
40 UNTIL KAPITAL>=25000
50 PRINT "OPSPARINGEN VARER: ";TEL;"KVARTALER."
```

Den variable KAPITAL indeholder den opsparede kapital med tilskrevne renter og TEL indeholder det antal kvartaler, der er gået, siden opsparingen begyndte. I linje 10 sættes kapital og tæller til 0, og i linje 20, 30 og 40 efterligner datamaten opsparingen, idet den gentager (repeterer) linje 30, indtil kapitalen er større end eller lig med 25000 (KAPITAL>=25000). Hver gang linje 30 udføres, sker der følgende: Der tilskrives 3% rente til den kapital, som hidtil har stået på kontoen, og de 2000 kr., der indsættes, tilskrives kapitalen. Endvidere øges TEL med 1. Når KAPITAL er på mindst 25000 kr., udskrives det antal kvartaler, opsparingen har varet (tælleren).
Dette forløb, som i virkeligheden vil vare flere år, kan datamaskinen gennemspille på brøkdele af sekunder og således give værdifulde oplysninger, som formentlig

indvirker på de planer, manden har med sine penge. Der er ingen tvivl om, at datamaten arbejder med formaliserede repræsentationer af kendsgerninger og forestillinger, som både omformes og meddeles ved processen. □

I eksemplet er vist, hvorledes en del af et program (nemlig sætningerne i linje 30) udføres flere gange, afhængig af om en bestemt betingelse er opfyldt eller ej. Vi har at gøre med to nye styresætninger: REPEAT og UNTIL p, hvor p er et åbent udsagn, og de to sætninger virker på den måde, at linjen eller linjerne mellem dem repeteres, indtil p er sand. Strukturen af denne styring er vist i rutediagrammet herunder. Ved siden af diagrammet er programstrukturen vist:



Programafsnittet A bliver gentaget, indtil p har værdien sand.

Eksempel 12.2.
Vi vil udbygge programmet fra eksempel 12.1, således at vi kan indtaste forskellige udbetalinger og indskud. Endvidere vil vi indrette det således, at hvis der går mere end 10 år (40 kvartaler), inden udbetalingen kan nås, skal datamaten skrive dette og derpå standse udregningerne.
Vi får følgende program:

```
10 INPUT "UDBETALINGEN ER: ",UDBETAL
20 INPUT "DER INDBETALES PR. KVARTAL: ", INDBET
30 LET KAPITAL=0; TEL=0
40 REPEAT
50   LET KAPITAL=KAPITAL*103/100+INDBET; TEL=TEL+1
60 UNTIL KAPITAL>=UDBETAL OR TEL=40
70 IF TEL=40 THEN DO
80   PRINT "DE BØR OVERVEJE DERES PLANER,"
90   PRINT "DER VIL GAA OVER TI AAR,"
100  PRINT "FØR DE HAR TIL UDBETALINGEN."
110 ELSE
120  PRINT "OPSPARINGEN VARER: ";TEL;"KVARTALER."
130 ENDIF
```

Vi vil overlade det til læseren at gennemgå programmet og undersøge, om det fungerer, som det skal.

□

Eksempel 12.3.

En ægpakkemaskine lægger æg ned i ægbakker 6 ad gangen. Vi vil skrive et program, som efterligner denne proces. Som inddata for programmet vil vi have antallet af æg, som skal pakkes, og som uddata ønsker vi det antal bakker, der afleveres færdigtpakkede af maskinen.
Vi forsøger med følgende program:

```
10 INPUT "ANTAL AEG: ",E
20 LET B=0
30 REPEAT
40   LET E=E-6; B=B+1
50 UNTIL E=0
60 PRINT "DER BLEV PAKKET:";B;"BAKKER."
```

Man skal være heldig, hvis det program virker! Hvis antallet af æg ikke tilfældigvis er et multiplum af 6, bliver E aldrig 0, og programmet vil fortsætte i det uendelige, eller - nok så rimeligt - indtil operatøren trykker ESC.
Vi forsøger at ændre linje 50 til:

```
50 UNTIL E<6
```

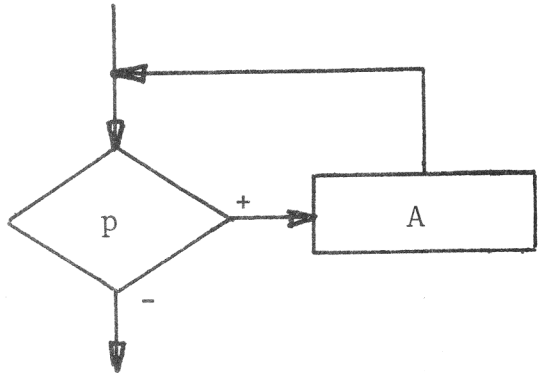
Umiddelbart ser dette bedre ud, og i de fleste tilfælde vil programmet give en korrekt efterligning af den virkelige proces: Maskinen pakker, indtil der er mindre end 6 æg tilbage. Så vil den standse, da den ikke kan pakke flere hele bakker. Imidlertid er der stadig ét tilfælde, i

hvilket programmet ikke kan bruges, nemlig hvis antallet af æg fra begyndelsen er mindre end 6! Linje 40 vil blive udført én gang, idet systemet først i linje 50 opdager, at E er mindre end 6. Derpå udføres linje 60 med det resultat, at det ser ud, som om der er blevet pakket én bakke med æg!

□

Eksemplet viser, at man ikke altid kan have styresætningen med det åbne udsagn anbragt sidst i forløbet. Derfor har man endnu et sæt styresætninger i COMAL II, som kan bruges, når man ønsker en del af et program udført flere gange i samme kørsel. Disse styresætninger er anført herunder, og strukturen af styringen er vist i rutediagrammet ved siden af:

```
WHILE p DO
  [---A---]
ENDWHILE
```



I WHILE p DO er p et åbent udsagn, og sålænge p er sand, bliver delprogrammet A, som beskrives i linjen eller linjerne mellem WHILE p DO og ENDWHILE, repeteret.

Eksempel 12.4.
Med de netop indførte styresætninger kan vi gøre vor "ægpakkesimulator" færdig. Vi skriver:

```
10 INPUT "ANTAL AEG: ",E
20 LET B=0
30 WHILE E>=6 DO
40   LET E=E-6; B=B+1
50 ENDWHILE
60 PRINT "DER BLEV PAKKET: ";B;"BAKKER."
```

Nu vil systemet allerede i linje 30 undersøge, om der er

æg nok til endnu en bakke, og hvis ikke det er tilfældet, fortsætter udførelsen af programmet med linje 60, hvor det korrekte antal bakker bliver udskrevet.

□

De dele af et program, som gentages, kaldes også løkker, og man omtaler de to programstrukturer, vi netop har gennemgået, som løkkestrukturer.

Eksempel 12.5.
Tværsummen af et positivt, helt tal er summen af tallets cifre. Programmet herunder bestemmer tværsummen af alle hele, positive tal, som er større end eller lig med 1000 og mindre end 5000.

```
 10 C3=0
 20 REPEAT
 30   C3=C3+1; C2=-1
 40   REPEAT
 50     C2=C2+1; C1=-1
 60     REPEAT
 70       C1=C1+1; C0=-1
 80       REPEAT
 90         C0=C0+1; T=1000*C3+100*C2+10*C1+C0
100         PRINT "TVAERSUMMEN AF:";T;"ER:";C3+C2+C1+C0
110       UNTIL C0=9
120     UNTIL C1=9
130   UNTIL C2=9
140 UNTIL C3=4
```

Programmet virker på samme måde som en flerecifret tæller (fx. en kilometertæller i en bil). I den inderste løkke (80 - 110) tælles enerne: C0=C0+1, i den næstinderste løkke (60 - 120) tælles tierne: C1=C1+1 osv. Vi opfordrer læseren til at gå programmet igennem for at se efter, om det virker, som det skal. Eksemplet har først og fremmest til formål at demonstrere, at man kan have flere løkker inden i hinanden. Den samme proces kan udføres af et program med fire WHILE-løkker inden i hinanden, og man kan for den sags skyld blande WHILE- og REPEAT-løkker, så fx. hver anden løkke bliver en WHILE-løkke og de øvrige REPEAT-løkker.

□

I COMAL II kan man have indtil 4 løkker af hver type inden i hinanden, uafhængig af rækkefølgen. Et program kan altså indeholde i alt 8 løkker inden i hinanden, og disse kan være "blandede" WHILE- og REPEAT-løkker. Man skal blot passe på, at der ikke bliver mere end 4 af hver slags. I de enkelte løkker kan man, uanset på hvilket niveau løkken findes, have forgreninger. Hvis man har forgreninger inden i løkker, som selv er dele af forgreninger, skal man blot se efter, at der ikke bliver mere end ialt 7 forgreninger i dybden (jvf. p. 20).
Sammenlagt har man altså mulighed for at nå op på ialt 15 niveauer med de styrestrukturer, vi har gennemgået i det foregående. Senere får vi, som tidligere nævnt, endnu en løkkestruktur, hvorved der føjes endnu 4 niveauer til. Dette vil være tilstrækkeligt til de fleste formål.

Eksempel 12.6.
Følgende eksempel er en udbygning af programmet fra eksempel 12.5, idet alle tal, i hvilke blot ét ciffer er 5, fjernes fra tværsumstabellen. Formålet med eksemplet er det ene at vise, hvorledes flere forgreninger og løkker er lagt inden i hinanden.

```
 10 C3=0
 20 REPEAT
 30 | C3=C3+1; C2=-1
 40 | WHILE C2<9 DO
 50 | | C2=C2+1; C1=-1
 60 | | IF C2><5 THEN DO
 70 | | | REPEAT
 80 | | | | C1=C1+1; C0=-1
 90 | | | | IF C1><5 THEN DO
100 | | | | | WHILE C0<9 DO
110 | | | | | | C0=C0+1; T=1000*C3+100*C2+10*C1+C0
120 | | | | | | IF C0><5 THEN DO
130 | | | | | | | PRINT "TVS. AF";T;"ER:";C3+C2+C1+C0
140 | | | | | | ENDIF
150 | | | | | ENDWHILE
160 | | | | ENDIF
170 | | | UNTIL C1=9
180 | | ENDIF
190 | ENDWHILE
200 UNTIL C3=4
```

De forskellige niveauer er markerede ved lodrette linjer og nummereret fra 1 til 7. Denne teknik er for øvrigt altid anvendelig, hvis man har med programmer at gøre, hvor der er mange styrestrukturer inden i hinanden.

□

I forbindelse med en styrestruktur optræder altid et åbent udsagn. Sådanne åbne udsagn kaldes ofte Boolske udtryk (efter den engelske matematiker George Boole). Denne terminologi vil ofte blive benyttet i det følgende. Vi kan herefter tale om dels aritmetiske udtryk (formler), der har en talværdi, og dels om Boolske udtryk, der har en sandhedsværdi.

RESUMÉ.

I dette kapitel har vi hørt om:

Relationsoperatorer, logiske operatorer, Boolske udtryk og følgende styrestrukturer:

(1) IF p THEN DO ... ELSE ... ENDIF

(2) IF p THEN DO ... ENDIF

(3) REPEAT ... UNTIL p

(4) WHILE p DO ... ENDWHILE.

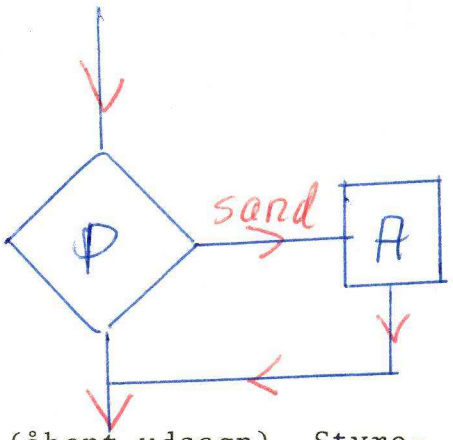
(1) er opbygget således:

```
IF p THEN DO
  [-----------]
  [     A     ]
  [-----------]
ELSE
  [-----------]
  [     B     ]
  [-----------]
ENDIF
```

hvor p er et Boolsk udtryk (åbent udsagn). Styresætningerne virker således: Hvis p er sand, udføres programdelen A mellem IF p THEN DO og ELSE, og hvis p er falsk, udføres programdelen B mellem ELSE og ENDIF. I begge tilfælde fortsættes programudførelsen med sætningen i linjen efter ENDIF.

(2) er opbygget således:

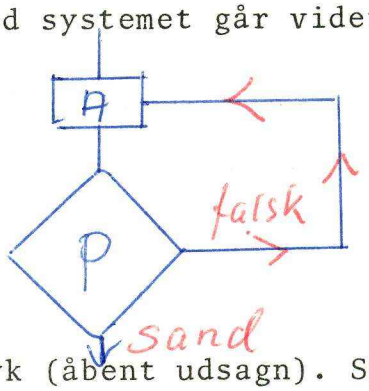
```
IF p THEN DO
[----------]
[    A     ]
[----------]
ENDIF
```



hvor p er et Boolsk udtryk (åbent udsagn). Styresætningerne virker således: Hvis p har værdien sand, udføres programdelen A, og derpå fortsættes udførelsen af programmet med linjen efter ENDIF. Hvis p har værdien falsk, udføres A ikke, med systemet går videre med linjen efter ENDIF.

(3) er opbygget således:

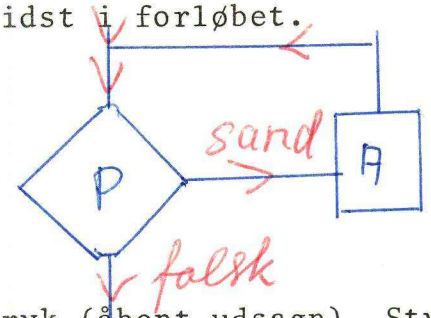
```
REPEAT
[----------]
[    A     ]
[----------]
UNTIL p
```



hvor p er et Boolsk udtryk (åbent udsagn). Styresætningerne virker således: Programdelen A gentages, indtil p har værdien sand. Når p er sand, fortsættes udførelsen med linjen efter UNTIL p.
Algoritmen A udføres altid mindst én gang, idet styringen er anbragt til sidst i forløbet.

(4) er opbygget således:

```
WHILE p DO
[----------]
[    A     ]
[----------]
ENDWHILE
```



hvor p er et Boolsk udtryk (åbent udsagn). Styresætningerne virker således: Programdelen A gentages, sålænge p er sand. Hvis p er falsk, fortsættes udførelsen af programmet med linjen efter ENDWHILE. Hvis p er falsk, når udførelsen af programmet når til WHILE-sætningen, udføres A overhovedet ikke, men systemet fortsætter uden videre med at udføre linjen efter ENDWHILE.

KAP. III.
INDICEREDE VARIABLE. FUNKTIONER.

13. Enkelt indicerede variable (Vektorer).

I de foregående afsnit har vi kun betragtet sådanne problemer, som kunne løses ved brug af simple variable. Selv om en simpel variabel kun kan indeholde én konstant ad gangen, kan man alligevel med lidt omtanke anvende simple variable ved behandling af store datamængder.

Eksempel 13.1.
Ved de fleste statistiske beregninger udgøres inddatamængden af et sæt talværdier, observationssættet, der fx. kan være fremkommet ved målinger eller tællinger.
Ved beregningerne vil man næsten altid få brug for at bestemme gennemsnittet af såvel tallene i observationssættet som af disse tals kvadrater.
Herunder er vist to programmer, der med et observationssæt (legemshøjderne for 25 børn i en 4.klasse), bestemmer og udskriver de nævnte gennemsnit. I det første program er observationssættet angivet i en række DATA-sætninger (fx. markeret på stregkort), og i det andet program skal observationssættet indtastes fra en terminal (on-line):

```
 10 DATA 141,135,128,136,143,150,140,145,137,131
 20 DATA 139,136,148,130,140,143,137,132,143,130
 30 DATA 135,145,142,138,145
 40 LET SUM=0; KVSUM=0; ANTAL=0
 50 REPEAT
 60   READ HØJDE
 70   LET SUM=SUM+HØJDE; KVSUM=KVSUM+HØJDE*HØJDE
 80   LET ANTAL=ANTAL+1
 90 UNTIL ANTAL=25
100 PRINT SUM/ANTAL,KVSUM/ANTAL
```

On-Line

```
 10 LET SUM=0; KVSUM=0; ANTAL=0
 20 REPEAT
 30   INPUT HØJDE
 40   LET SUM=SUM+HØJDE; KVSUM=KVSUM+HØJDE*HØJDE
 50   LET ANTAL=ANTAL+1
 60 UNTIL ANTAL=25
 70 PRINT SUM/ANTAL,KVSUM/ANTAL
```

□

I begge programmer fra eksemplet bliver tallene indlæst ét for ét i den rækkefølge, i hvilken de er anført i DATA-sætningerne eller på den liste, operatøren indtaster efter. Et tal, der er indlæst i HØJDE og behandlet, bliver slettet i HØJDE, når det næste tal hentes ind. Endvidere bemærker vi, at tallene i DATA-sætningerne kun kan læses af programmet på én bestemt måde: først 141, så 135, derpå 128 osv. (jvf. billedet med viseren side 8 og 9). Når man er nået til tallet 150 i linje 10, kan man fx. ikke springe tilbage til tallet 135; men indlæsningen vil uden videre fortsætte med tallet 140. En datamængde, som er arrangeret på denne måde, kaldes en lineær liste. En lineær liste er således karakteriseret ved, at man kun kan få adgang til de enkelte data i den fra en ende af og i en bestemt rækkefølge. Man kan ganske vist bruge RESTORE-sætningen (jvf. side 9) selvom listen ikke er læst til ende; men det betyder blot, at man atter må læse listen forfra i den engang fastlagte rækkefølge. Ved mange af de opgaver, der søges løst ved hjælp af datamater, er det imidlertid ikke tilstrækkeligt at have adgang til datamængden på denne måde. Man indfører derfor en ny slags variable: de nummererede eller indicerede variable. I COMAL II forekommer to slags indicerede variable: enkelt indicerede og dobbelt indicerede. Vi vil begynde med at se på enkelt indicerede variable.

En enkelt indiceret variabel angives ved et navn og et indeks (nummer), som anbringes i en parentes efter navnet. Således er PARTI(2), SKEMA(15) og S(105) korrekte betegnelser for indicerede variable. Man må skelne nøje mellem angivelser som fx. PARTI2 og PARTI(2). Det første er navn for en simpel variabel, hvor 2-tallet er en del af selve navnet, mens det andet betegner en indiceret variabel, hvis navn er PARTI og hvis indeks er 2.

I COMAL II kan indeks kun antage hele, ikke-negative talværdier: 0, 1, 2, 3, osv. Hvis man angiver en indiceret variabel således: A(7.36), vil systemet af sig selv omskrive indeks til 7 (altså heldelen af 7.36).

Fordelen ved at benytte indicerede variable frem for simple variable bliver dog først tydelig, når man tilføjer, at indeks kan angives ved en variabel eller et udtryk, hvis værdi er eller kan udregnes til et ikke-negativt, helt tal. For at forstå, hvad dette betyder, vil vi gennemgå følgende lille program:

```
10 DATA 5,9,6,3,7,8
20 LET I=0
30 REPEAT
40   READ X(I)
50   LET I=I+1
60 UNTIL I=6
```

Som man vil se, skal linjerne 40 og 50 gentages, indtil I er lig med 6. Når linje 40 udføres første gang, sker der flere ting: For det første afsætter systemet automatisk 11 lagerpladser, som får navnene X(0), X(1), X(2), ..., X(10). Dernæst, da sætningen er en READ-sætning og I har værdien 0, indlæses konstanten 5 i den variable X(0). I linje 50 sættes I = 0 + 1 = 1, og da I ikke er 6, udføres linje 40 og 50 atter. Ved udførelsen af linje 40 får den variable X(1) tildelt værdien 9, og ved udførelsen af linje 50 sættes I = 1 + 1 = 2. Da I stadig ikke er lig med 6, udføres linjerne 40 og 50 endnu engang. Således fortsættes, indtil I er lig med 6. Resultatet af kørslen er, at de variable X(0), X(1), X(2), X(3), X(4) og X(5) har fået tildelt tallene hhv.: 5, 9, 6, 3, 7 og 8.
Hvis vi til programmet ovenfor føjer linjen:

```
70 PRINT X(3)
```

vil vi få udskriften

```
6
```

hvilket viser, at systemet har information om, at der findes en variabel med navnet X(3), og at denne variabel indeholder konstanten 6. Det kan bemærkes, at de tilovers-blevne 5 variable: X(6), X(7), X(8), X(9) og X(10) alle indeholder konstanten 0.

En datamængde, der er indlæst i en række enkelt indicerede variable med samme variabelnavn, kaldes en vektor. De enkelte tal i mængden kaldes vektorens komponenter. Vektorens dimension er dens største indeks. Ønsker man at angive vektoren under ét, betegner man den ved det variabelnavn, der står før indeks.
I eksemplet ovenfor er der altså tale om en vektor X med følgende komponenter: 5, 9, 6, 3, 7, 8, 0, 0, 0, 0, 0. Vektorens dimension er 10.
Vi bemærker, at vektoren X har fem overflødige komponenter, nemlig de fem 0'er. Dette er ikke nogen rationel udnyttelse af datamatens lager, og man kan let fjerne dem, idet man på forhånd giver systemet oplysninger om, hvor mange komponenter, man ønsker at læse ind i vektoren X. Hertil anvender man følgende linje:

```
5 DIM X(5)
```

Når denne linje udføres, afsætter systemet netop seks lagerpladser med navnene X(0), X(1), X(2), X(3), X(4) og X(5), og i linje 40 sker der nu blot en indlæsning af vektorens komponenter.
DIM-sætninger (dimensionerings-sætninger) anvendes også, når man ønsker at afsætte flere end 11 pladser til en vektor.

Eksempel 13.2.
Vi vender tilbage til eksempel 13.1 og angiver nu et tredje program, som kan udføre de i eksemplet nævnte gennemsnitsberegninger:

```
10 DIM HØJDE(24)
20 LET SUM=0; KVSUM=0; NUMMER=0
30 REPEAT
40   INPUT HØJDE(NUMMER)
50   LET SUM=SUM+HØJDE(NUMMER); KVSUM=KVSUM+HØJDE(NUMMER)↑2
60   LET NUMMER=NUMMER+1
70 UNTIL NUMMER=25
80 PRINT SUM, KVSUM
```

I linje 10 afsættes plads til en vektor HØJDE med dimension 24 (plads til 25 komponenter), og i linje 40 indtastes

de 25 tal, som udgør observationssættet. I linje 50 beregnes som før summen af de indtastede tal og summen af de indtastede tals kvadrater. I linje 60 øges (opdateres) tælleren NUMMER med 1. Denne tæller benyttes også til at angive indeks for den komponent af HØJDE, der indlæses. Den er afgørende forskel på dette program og de to programmer fra eksempel 13.1 er imidlertid, at samtlige talværdier fra observationssættet er lagret i vektoren HØJDE. når programmet er udført. Man kan derefter arbejde videre med observationssættet, hvis man ønsker det. Således vil de følgende linjer give en kontroludskrift af de indtastede talværdier:

```
100 LET I=0
110 WHILE I<25 DO
120   PRINT HØJDE(I);
130   I=I+1
140 ENDWHILE
```

□

Eksempel 13.3.

I et udsalgskatalog fra et varehus er anført 100 varegrupper. Hver varegruppe har et nummer, og ud for hvert nummer er anført styktal og stykpris. For hver uge foretages en opgørelse over værdien af restlageret. Vi ønsker at skrive et program, som kan udregne denne værdi. Vi tænker os, at inddata for programmet er anført i en tabel som denne:

VARENUMMER	ANTAL	STYKPRIS
NR	ANTAL(NR)	PRIS(NR)
1	234	3.20
2	504	4.70
.	.	.
.	.	.
.	.	.
100	19	345.25

Vi indretter det altså således, at vektoren ANTAL indeholder styktallene for samtlige varenumre og vektoren PRIS indeholder stykpriserne for samtlige varenumre. En vares nummer er samtidig indeks for de komponenter, der angiver

hhv. dens styktal og stykpris. Således indeholder fx. den variable ANTAL(5) styktallet for vare nummer 5, og den variable PRIS(5) er tildelt den pågældende vares pris. Den samlede værdi af vare nummer 5 er da ANTAL(5)*PRIS(5).
Vi kan nu skrive et program, der med styktal og stykpriser som inddata beregner og udskriver varelagerets øjeblikkelige værdi:

```
10 DIM ANTAL(100), PRIS(100)
20 LET NR=1; VAERDI=0
30 WHILE NR<=100 DO
40   INPUT "STYKTAL OG STYKPRIS: ",ANTAL(NR),PRIS(NR)
50   LET VAERDI=VAERDI+ANTAL(NR)*PRIS(NR); NR=NR+1
60 ENDWHILE
70 PRINT "VARELAGERETS VAERDI: ";VAERDI;"KR."
```

□

I det netop anførte eksempel har vi dimensioneret de to vektorer ANTAL og PRIS i samme sætning. Man kan i en DIM-sætning have så mange dimensioneringer, som linjelængden tillader.

14. Dobbelt indicerede variable (matricer).

Som nævnt i forrige afsnit, findes der to slags indicerede variable i COMAL II: enkelt indicerede og dobbelt indicerede. I dette afsnit vil vi gennemgå de dobbelt indicerede variable.
En dobbelt indiceret variabel angives ved et navn og to indekser (numre), som anbringes i en parentes efter navnet. Mellem de to indekser anbringes et komma (,). Eksempler på dobbelt indicerede variable er: TABEL(8,3), OBS(0,0), Q(5,17) og X(118,42). For hvert af de to indekser gælder samme regler, som for indekset i en enkelt indiceret variabel (jvf. side 31 og 32).
En datamængde, som er indlæst i en samling dobbelt indicerede variable med samme variabelnavn, kaldes en matrix (flertal: matricer), og de enkelte data i mængden kaldes matricens komponenter. Matricens dimension er et ordnet talpar (n,m), hvor n angiver det største indeks, der kan optræde på første plads i parentesen, og m angiver det største indeks, der kan optræde på anden plads i parentesen. Matricens navn er det variabelnavn, der står

foran parentesen med indekserne.
På tilsvarende måde som for vektorer kan man meddele systemet, hvormange lagerpladser man ønsker afsat til en given matrix, ved at bruge DIM-sætningen. Hvis man fx. skriver:

```
DIM P(20,30)
```

reserverer systemet lagerplads til 21×31 komponenter (0 medregnes til indekserne på samme måde som for vektorer (jvf. side 32)). Hvis dimensionen for en matrix ikke er angivet i en DIM-sætning, afsætter systemet automatisk 11×11 pladser i lageret, når matricen første gang bliver registreret. Således vil sætningen

```
LET EKS(3,7)=35
```

bevirke, at der afsættes 121 pladser i lageret til matricen EKS. Endvidere får EKS(3,7) tildelt værdien 35, mens alle øvrige variable EKS(I,J) får tildelt værdien 0.

Eksempel 14.1.
I fortsættelse af eksempel 13.1 tænker vi os, at varehuset (fx. af hensyn til forskellige rabatordninger) ønsker at inddele de 100 varenumre i forskellige grupper. Man indretter det således, at der bliver 5 grupper med 20 varenumre i hver. Når en bestemt vare herefter angives således: (3,12), betyder det, at varen er i gruppe 3, og at den indenfor denne gruppe har nummer 12.
Vi vil skrive et nyt program, som kan udregne værdien af varelageret. Vi indretter det således, at det antal enheder, der findes af de forskellige varenumre, indlæses i en matrix ANTAL, og at priserne for de forskellige varenumre indlæses i en matrix PRIS. De to tal, som angiver hhv. gruppe og nummer for varen, bruger vi som indekser for de to matricer. Antal enheder af varen (3,12) skal altså stå i den variable ANTAL(3,12) og enhedsprisen for varen (3,12) skal stå i den variable PRIS(3,12). Den samlede værdi af varen (3,12) er derefter ANTAL(3,12)*PRIS(3,12).

Programmet kan se således ud:

```
 10 DIM ANTAL(5,20), PRIS(5,20)
 20 LET GR=1; VAERDI=0
 30 WHILE GR<=5 DO
 40   LET NR=1
 50   WHILE NR<=20 DO
 60     INPUT "STYKTAL OG STYKPRIS: ",ANTAL(GR,NR),PRIS(GR,NR)
 70     LET VAERDI=VAERDI+ANTAL(GR,NR)*PRIS(GR,NR)
 80     LET NR=NR+1
 90   ENDWHILE
100   LET GR=GR+1
110 ENDWHILE
120 PRINT "VARELAGERETS VAERDI: ";VAERDI;"KR."
```

Når linje 10 udføres, afsætter systemet 6×21 pladser til matricen ANTAL og 6×21 pladser til matricen PRIS. Når linjerne 20-50 er udført, har GR og NR begge værdien 1, og linjerne 60, 70 og 80 gentages, sålænge NR<=20. Herved indlæses antal og priser i de variable ANTAL(1,1), ANTAL(1,2), ANTAL(1,3), ..., ANTAL(1,20) og PRIS(1,1), PRIS(1,2), PRIS(1,3), ..., PRIS(1,20) hhv., og den samlede værdi af de tilsvarende varer indlæses i VAERDI. Derefter har NR værdien 21, og udførelsen går videre med linje 100, hvor GR sættes til 2. Da det Boolske udtryk: GR<=5 er sandt, udføres WHILE-løkken fra linje 30 til linje 100 for anden gang. I linje 40 sættes NR atter til 1, og udførelsen af linjerne 60, 70 og 80 gentages, sålænge NR<=20, med det resultat, at der indlæses antal og priser i de variable ANTAL(2,1), ANTAL(2,2), ..., ANTAL(2,20) og PRIS(2,1), PRIS(2,2),..., PRIS(2,20) hhv. Værdien af varerne opsalderes stadig i den variable VAERDI. Derpå sættes GR lig med 3, og den yderste WHILE-løkke udføres for tredje gang. Således fortsættes, indtil GR er lig med 5 og NR er lig med 20, og værdien af varen (5,20) er lagt til værdien af de øvrige varer. Idet NR udregnes til 21 i linje 80 og GR derpå sættes til 6 i linje 100, styres programudførelsen ikke længere af de to løkker, og linje 120 bliver udført med det resultat, at varelagerets samlede værdi, som nu er opsummeret i VAERDI, bliver udskrevet.

□

15. FOR..NEXT-sætninger.

I programmer med indicerede variable har vi brugt REPEAT-

og WHILE-løkker i forbindelse med opdateringen af indekserne (jvf. eksempel 13.2, 13.3 og 14.1). I mange sammenhæng og meget ofte i programmer med indicerede variable kan WHILE-løkker beskrives på en mere bekvem måde ved brug af FOR...NEXT-sætningerne, hvis betydning vi fastlægger ved følgende:

(i) Programmet: kan erstattes af:

```
LET I=x                          FOR I=x TO y
WHILE I<=y DO                      [----A-----]
  [----A-----]                     NEXT I
  LET I=I+1
ENDWHILE
```

hvor x og y er konstanter eller udtryk.

(ii) Programmet: kan erstattes af:

```
LET I=x                          FOR I=x TO y STEP q
WHILE I<=y DO                      [----A-----]
  [----A-----]                     NEXT I
  LET I=I+q
ENDWHILE
```

hvor x, y og q er konstanter eller udtryk og q er positiv.

(iii) Programmet: kan erstattes af:

```
LET I=x                          FOR I=x TO y STEP -q
WHILE I>=y DO                      [----A-----]
  [----A-----]                     NEXT I
  LET I=I-q
ENDWHILE
```

hvor x, y og q er konstanter eller udtryk og q er positiv.

I alle tre tilfælde er A det programafsnit, som står i løkken. Den variable, som styrer udførelsen af løkken, er en simpel variabel og er for nemheds skyld betegnet ved I i alle tre tilfælde, men kan naturligvis angives ved et hvilket som helst navn, der er tilladt for simple

variable. Det skal bemærkes, at hvis x, y og q er udtryk, bliver disse udtryk udregnet i FOR-sætningen og ændrer sig ikke under udførelsen af løkken. En WHILE-løkke er ikke underkastet sådanne restriktioner, og man må altså betragte FOR...NEXT-løkker som særdeles nyttige specialtilfælde af WHILE-løkker.

Eksempel 15.1.
Vi vil omskrive programmerne fra eksempel 13.3 og 14.1, idet vi anvender FOR...NEXT sætninger som erstatning for WHILE...ENDWHILE:

```
10 DIM ANTAL(100), PRIS(100)
20 VAERDI=0
30 FOR NR=1 TO 100
40   INPUT "STYKTAL OG STYKPRIS: ",ANTAL(NR),PRIS(NR)
50   LET VAERDI=VAERDI+ANTAL(NR)*PRIS(NR)
60 NEXT NR
70 PRINT "VARELAGERETS VAERDI: ";VAERDI;"KR."
```

```
10 DIM ANTAL(5,20),PRIS(5,20)
20 LET VAERDI=0
30 FOR GR=1 TO 5
40   FOR NR=1 TO 20
50     INPUT "STYKTAL OG STYKPRIS: ",ANTAL(GR,NR),PRIS(GR,NR)
60     LET VAERDI=VAERDI+ANTAL(GR,NR)*PRIS(GR,NR)
70   NEXT NR
80 NEXT GR
90 PRINT "VARELAGERETS VAERDI: ";VAERDI;"KR."
```

Læseren opfordres til at foretage en detaljeret sammenligning mellem disse programmer og de oprindeligt anførte.
□

I det andet program i eksempel 15.1 er anvendt to FOR... NEXT-løkker inden i hinanden. I COMAL II kan man have indtil fire sådanne løkker inden i hinanden uafhængig af, om der ind imellem er benyttet andre af COMAL's styresætninger.

16. Indbyggede funktioner.

I COMAL II findes en række indbyggede funktioner, som man uden videre kan anvende i sine programmer. Herunder er disse funktioner angivet, og det bemærkes at argumentet

x i alle tilfælde kan være en konstant, en variabel eller et udtryk, hvis værdi kun begrænses af den pågældende funktions definitionsmængde:

INT(x): heldelen af x (sædvanlig notation: [x]).
ABS(x): den numeriske (absolutte) værdi af x.
SQR(x): kvadratroden af x.
LOG(x): den naturlige logaritme af x.
SIN(x): sinus af x, hvor x er i radianer.
COS(x): cosinus af x, hvor x er i radianer.
TAN(x): tangens af x, hvor x er i radianer.
ATN(x): arctangens af x (princ. værdi i radianer).
EXP(x): eksponentialfunktionen: e^x.
SGN(x): antager værdierne -1, 0 eller 1, når x er hhv. negativ, nul eller positiv.

Herudover findes nogle specielle funktioner, som vil blive omtalt i anden sammenhæng.
De indbyggede funktioner kan anvendes overalt, hvor konstanter, variable eller udtryk kan anvendes.

Eksempel 16.1.
Som eksempel på anvendelsen af en indbygget funktion vil vi anføre følgende lille program, der med to hele tal som inddata afgør, om det ene af tallene går op i det andet:

```
10 INPUT "INDTAST TALLENE A OG B: ",A,B
20 IF INT(B/A)*A=B THEN DO
30   PRINT A;"GAAR OP I";B
40 ELSE DO
50   PRINT A;"GAAR IKKE OP I";B
60 ENDIF
```

Bemærk, at argumentet for funktionen INT er et udtryk (nemlig B/A). Det overlades til læseren at overbevise sig om, at det åben udsagn INT(B/A)*A=B er ensbetydende med det åben udsagn "A går op i B".
□

17. Funktioner, som defineres af brugeren.

Som nævnt ovenfor, udregner funktionen LOG(x) den natur-

lige logaritme af x, hvor x er et positivt tal. Hvis man i et program har brug for titalslogaritmen af x, kan man selv definere denne funktion i en DEF-sætning:

```
DEF FNL(X)=LOG(X)/LOG(10)
```

Sætningen består af ordet DEF (definér) efterfulgt af navnet på den funktion, man ønsker at definere. Dette navn består altid af bogstaverne FN efterfulgt af ét bogstav, som brugeren selv kan vælge i det engelske alfabet (A - Z). Derpå følger som sædvanligt argument-parentesen, et lighedstegn og det udtryk, funktionen er bestemt ved. Den funktion (titalslogaritmen), som er defineret i linje 10, har altså navnet FNL, og under dette navn kan den derefter benyttes i resten af programmet. Således vil eksempelvis sætningerne:

```
LET X=FNL(1000); Y=FNL(X/3)
PRINT X,Y
```

bevirke, at tallene 3 og 0 udskrives.

Eksempel 17.1.
Ved konkurrenceskydning anvendes lerduer, som kastes op i luften af en kastemaskine. Lerduerne beskriver en kurve, som med god tilnærmelse svarer til grafen for funktionen

$$y = x \cdot \tan(w) - \frac{9.81 \cdot x^2}{2 \cdot v_0^2 \cdot \cos^2(w)} \cdot (1 + k \cdot x)$$

hvor v_0 er den fart (m/s), med hvilken lerduen forlader kastemaskinen, w er vinklen (grader), som udkastets retning danner med det vandrette plan, og k er en konstant, der afhænger af lerduens form og overfladebeskaffenhed. Vi vil skrive et program, der med v_0, w og k som inddata beregner og udskriver lerduens højde y (meter) over jordoverfladen for x = 1, 2, 3, ..., x_s (meter), hvor x_s er den første x-værdi, der giver en negativ y-værdi. Af disse tal kan vi dels få et billede af lerduens banekurve og dels få at vide, omtrent hvor lerduen har ramt jorden (nemlig indenfor afstanden x_s - 1 til x_s meter fra kaste-

maskinen). Programmet kan se således ud:

```
10 INPUT "FART(M/S), VINKEL(DEG.) OG LUFTMODS.KOEFF:",V0,W,K
15 LET W=W*3.14159/180
20 DEF FNY(X)=X*TAN(W)-9.81*X*X*(1+K*X)/2/V0/V0/COS(W)↑2
25 LET X=0
30 REPEAT
35   LET X=X+1
40   PRINT X, FNY(X)
45 UNTIL FNY(X)<=0
```

Bemærk omskrivningen i linje 15 af vinklen fra grader til radianer. Denne omskrivning er nødvendig, da argumentet for COS i linje 20 skal være i radianer.
Det kan nævnes, at typiske værdier for inddata er: $v_0 = 9$, $w = 45$ og $k = 0.01$.

□

18. REM- og STOP-sætninger.

Et COMAL-program er skrevet til en datamat, for at få den til at udføre de processer, der er beskrevet i programmet. Det er imidlertid også meget vigtigt, at andre mennesker end den eller de, der har skrevet programmet, kan læse og forstå det. Måske ønsker andre at bruge programmet til at løse et problem, som ligner det, hvortil programmet egentligt er anvendt. Man behøver da blot ændre nogle detaljer i det oprindelige program, og spares derved for en mængde overvejelser og en masse skrivearbejde. For at gøre det lettere for andre at læse ens program, bør man udstyre det med passende vejledende tekster og kommentarer. Vi har allerede set eksempler på vejledende tekster, fx. indeholder INPUT-sætningen ovenfor en operatørvejledning, der imidlertid også virker oplysende for den, som blot læser programmet. På tilsvarende måde virker de fleste tekster i PRINT-sætninger. Fælles for vejledende tekster er, at de anvendes ved kommunikationen system-menneske under udførelsen af programmet. I modsætning hertil er kommentarer sådanne tekster, som ikke kommer til syne under eller påvirker udførelsen af programmet. I COMAL II kan man skrive kommentarer efter COMAL-ordene REPEAT, ELSE, ENDIF og ENDWHILE. Endvidere findes der

```
70
RENUMBER
* PUNCH
0010 REM PROGRAM, SOM ORDNER TALPAR MED DET MINDSTE FORST
0020 INPUT "ANTAL TALPAR: ",ANTAL
0030 PRINT "SKRIV '60 DATA' OG EN TILFAELDIG RAEKKE TALPAR."
0040 PRINT "SKRIV CON, NAAR DU ER FAERDIG MED DINE TALPAR."
0050 STOP
0060 DATA 67,56,78,56,34,23,89,67,45
0070 FOR T=1 TO ANTAL
0080   READ T1,T2
0090   IF T1>T2 THEN DO
0100     LET GEM=T1; T1=T2; T2=GEM
0110     PRINT T1,T2
0120   ELSE DO
0130     PRINT T1,T2
0140   ENDIF
0150 NEXT T
0160 STOP
```

Ordning af talpar

77-02-23

```
RUN
ANTAL TALPAR: 7
SKRIV '60 DATA' OG EN TILFAELDIG RAEKKE TALPAR.
SKRIV CON, NAAR DU ER FAERDIG MED DINE TALPAR.

STOP AT 0050

* 60 DATA 4,5,56,34,67,23,89,45,67,45,23,12,78
CON
 4                5
 34               56
 23               67
 45               89
 45               67
 12               23

ERROR 15 AT 0080 - IKKE FLERE DATA TIL READ

*
```

i COMAL II en særlig sætning: REM-sætningen (REM: remark), som udelukkende anvendes ved skrivning af kommentarer. Følgende eksempel viser en typisk REM-sætning:

```
100 REM   //UDSKRIVNING AF RESULTATER//
```

Sætningen begynder med ordet REM, og efter dette følger en eller anden række af tegn, som i almindelighed indeholder oplysninger om det programafsnit i hvilket sætningen forekommer, men iøvrigt kan være en hvilken som helst sammenstilling af tegn, som er til rådighed for det pågældende system. Således har de to skråstreger (//)i eksemplet ovenfor kun til formål at fremhæve den egentlige kommentar, nemlig at der nu følger en udskrivning af resultater. Grunden til, at man kan skrive hvad som helst efter ordet REM, er den, at REM-sætninger overhovedet ikke bliver udført af systemet under kørslen af programmet, men blot bliver gemt tilside, indtil brugeren ved en LIST-kommando forlanger en udskrift af programmet. På tilsvarende måde vil enhver tegnfølge, som skrives efter et af ordene: REPEAT, ELSE, ENDIF og ENDWHILE, være uden indflydelse på programudførelsen og kun komme til syne ved en listning af programmet.

Eksempel 18.1.

Vi vil udstyre programmet fra eksempel 11.1 med kommentarer:

```
 10 INPUT "ANTAL KOPIER: ",ANTAL
 20 REM   //SAEDVANLIGT ANTAL KOPIER//
 30 IF ANTAL>=6 AND ANTAL<=25 THEN DO
 40   AFGIFT=180+(ANTAL-5)*15
 50 ELSE  //MEGET STORT ELLER MEGET LILLE ANTAL KOPIER//
 60   IF ANTAL<6 THEN DO
 70     AFGIFT=36*ANTAL
 90   ELSE   //MERE END 25 KOPIER//
100     AFGIFT=480+(ANTAL-25)*6
110   ENDIF   //SLUT FAA ELLER MANGE//
120 ENDIF   //SLUT SAEDVANLIGT ANTAL//
130 PRINT "AFGIFTEN FOR";ANTAL;"KOPIER ER: ";AFGIFT/100;"KR."
```

Læg mærke til, at kommentarerne er "menneskelige" i den forstand, at de hentyder til problemet således som det

fremtræder for brugerne af kopimaskinen. Kommentarer af typen

```
20 REM   //FRA 5 TIL 25 KOPIER//
```

føjer ikke noget væsentligt til det, der i forvejen står i programmets linje 30.

□

I nogle tilfælde kan man være interesseret i at standse programudførelsen et eller flere steder i programmet under nærmere angivne omstændigheder. Hertil kan man bruge en sætning, som kun består af ordet STOP.

Eksempel 18.2.
Vi vil igen betragte "ægpakkeren" fra eksempel 12.3. Det anførte program med REPEAT...UNTIL-løkken virker som nævnt ikke under alle forhold. Ved anvendelse af en STOP-sætning kan programmet imidlertid bringes til at virke tilfredsstillende. Vi skriver således:

```
 10 INPUT "ANTAL AEG: ",A
 20 IF A<6 THEN DO
 30   PRINT "DER ER IKKE AEG NOK TIL EN BAKKE."
 40   STOP
 50 ENDIF   //SLUT, HVIS DER IKKE ER AEG NOK.//
 60 LET B=0
 70 REPEAT   //SAA PAKKES DER//
 80   LET A=A-6; B=B+1
 90 UNTIL A<6
100 PRINT "DER BLEV PAKKET: ";B;"BAKKER,"
110 PRINT "OG DER ER";A;"AEG TILBAGE I LØS VAEGT."
```

Det programafsnit, som er beskrevet i linje 20 - 50, virker som en "beskyttelse" af den ellers åbne REPEAT... UNTIL-løkke, idet programudførelsen vil standse i linje 40 efter at der er udskrevet en vejledende tekst til brugeren.

□

Når programudførelsen standses af en STOP-sætning, skriver systemet meldingen: STOP AT nn, hvor nn er STOP-sætningens linjenummer.

Hvis man ønsker, at programudførelsen skal fortsætte efter at den er blevet standset af en STOP-sætning, skriver man uden linjenummer ordet CON (continue) fra tastaturet og trykker derpå på RETURN-tasten. Systemet fortsætter da med at udføre sætningen umiddelbart efter den STOP-sætning, der har afbrudt udførelsen.

RESUMÉ.

I dette kapitel har vi hørt om:

Lineære lister, enkelt- og dobbelt indicerede variable, vektorer, matricer, dimension af en vektor, dimension af en matrix, indekser, kommentarer, DIM-sætninger, FOR...NEXT-sætninger, DEF-sætninger, REM-sætninger, STOP-sætningen, de indbyggede funktioner: INT, ABS, SQR, LOG, SIN, COS, TAN, ATN, EXP, SGN samt kommandoen CON.

DIM-sætninger er opbygget således:

DIM $navn_1(dim)$, $navn_2(dim)$, ..., $navn_q(dim)$,

hvor $navn_1$, $navn_2$, ..., $navn_q$ er navne på vektorer eller matricer (sædvanlige variabelnavne), og dim er dimensionen af den pågældende vektor eller matrix (dim kan altså bestå af én eller to komponenter, hvor hver enkelt komponent er en konstant, en variabel eller et udtryk).

FOR...NEXT-sætninger konstituerer en løkke-struktur, der er opbygget således:

```
FOR var=xx TO yy STEP qq
  [------A------]
NEXT var
```

hvor var er en variabel, mens xx, yy og qq er konstanter, variable eller udtryk. De værdier, som er tildelt eller tildeles xx, yy og qq i FOR-sætningen, fastholdes under udførelsen af FOR...NEXT-løkken.
Med denne modifikation er den anførte FOR...NEXT-løkke ækvivalent med én af følgende to WHILE-løkker:

```
LET I=xx                    LET I=xx
WHILE I<=yy DO              WHILE I>=yy DO
  [----A-----]                [----A-----]
  LET I=I+qq                  LET I=I+qq
ENDWHILE                    ENDWHILE
```

Hvor det første alternativ gælder, når qq er positiv, mens det andet gælder, når qq er negativ. Hvis intet STEP er anført, sættes qq automatisk til 1.

DEF-sætninger er opbygget således:

DEF FNb(var)=xx

hvor b er et enkelt bogstav (A-Z), var en variabel, som i denne sammenhæng kun må angives ved ét bogstav, og xx et udtryk, som skal indeholde den variable var. Når funktionen anvendes i programmet, kan der på var's plads indsættes en konstant, en variabel eller et udtryk. Funktioner kan anvendes overalt, hvor konstanter, variable eller udtryk kan anvendes.

REM-sætninger er opbygget således:

REM <tegnfølge>

hvor <tegnfølge> er en vilkårlig følge af tegn fra det til systemet hørende tegnsæt. REM-sætninger er kun aktive ved listning af programmet. Tegnsættets længde bestemmes af den for systemet fastlagte linjelængde.

STOP-sætningen består kun af ordet STOP:

STOP

og bevirker, at udførelsen af programmet standser ved den pågældende sætning. Systemet afgiver meldingen: STOP AT nn, hvor nn er STOP-sætningens linjenummer.

Kommandoen CON bevirker, at programudførelsen fortsættes, efter at den har været standset af en STOP-sætning. Udførelsen fortsætter med linjen umiddelbart efter STOP-sætningen.

KAP. IV.
TEGNFØLGER OG STRENGVARIABLE.

19. Tegnfølger.

Hidtil har vi kun beskæftiget os med processer, der udføres med tal. Vi har skrevet programmer, der behandler numeriske data. I disse programmer har vi dog ofte indføjet vejledende tekster, kommentarer eller tekster, som udskrives sammen med regneresultater.

Eksempel 19.1.
I programmet

```
10 REM   //PROGRAM TIL BEREGNING AF KILDESKAT//
20 INPUT "BRUTTOLØN, FRADRAG OG TRAEKPROCENT: ",BRLØN,FRDR,TRKPCT
30 PRINT "AT BETALE I KILDESKAT:"
40 PRINT
50 PRINT (BRLØN-FRDR)*TRKPCT/100;"KR."
```

finder vi såvel en kommentar (linje 10) som en vejledende tekst (linje 20) og tekster til udskrivning sammen med resultatet (linje 30 og 50).
□

Vi betragter linje 20 i programmet ovenfor. I denne linje finder vi teksten:

```
BRUTTOLØN, FRADRAG OG TRAEKPROCENT:
```

idet vi bemærker, at tegnet " ikke hører med til selve teksten, men blot tjener til at afgrænse denne. Nævnte tekst er sammensat af bogstaver, specialtegn (,:) og mellemrum (som naturligvis også er et tegn; det indtastes fra den største taste på tastaturet!). En sådan sammenstilling af tegn fra et bestemt tegnsæt (i praksis det, der er angivet på tastaturet) vil vi helt generelt kalde en tegnfølge.

Eksempel 19.2.
"BRAGESNAK I ØLGOD", "L5IR?&%:Ñ□" og "JEG HEDDER ANTON, ER 5 ÅR OG FORKØLET."

er tre eksempler på tegnfølger. Bemærk, at den midterste følge ikke er dannet af tegnsættet på et sædvanligt fjernskrivertastatur.

□

Det kan undertiden være en fordel at have en tegnfølge uden indhold til sin rådighed. Denne følge angives ved

""

og kaldes den tomme følge. Den tomme følge må ikke forveksles med følgen: " ", som indeholder ét tegn, nemlig mellemrummet (blanktegnet).
Som allerede nævnt skal tegnfølger i almindelighed afgrænses med anførselstegn ("i gåseøjne"). Grunden hertil er, at indholdet af følgen ikke må forveksles med variabelnavne, udtryk, konstanter eller COMAL-ord. Undtaget fra denne regel er kun de tegnfølger, der optræder som kommentarer (dvs. efter ordene: REM, ELSE, ENDIF, REPEAT, ENDWHILE, ENDCASES, END og ENDPROC (RETURN)).
En tegnfølges længde er det antal tegn, der indgår i følgen. Den tomme følge har længden 0.

Eksempel 19.3.
Tegnfølgen i linjen

```
20 INPUT "BRUTTOLØN, FRADRAG OG TRAEKPROCENT: ",BRLØN,FRDR,TRKPCT
```

har længden 36 (tæl efter!), og tegnfølgen i linjen

```
50 PRINT "              MED VENLIG HILSEN"
```

har længden 29 (brug linjen ovenover som tællelinje).

□

20. Strengvariable. Lagring af tegnfølger.

Når vi ønsker at anvende et tal i et program, kan vi enten angive det direkte som en konstant i programmet eller ved hjælp af en variabel, som viser hen til det sted i lageret, hvor tallet er gemt.
På tilsvarende måde kan en tegnfølge angives direkte

eller ved hjælp af en variabel, som henviser til følgen. Indtil nu har vi udelukkende anvendt den første metode og angivet eventuelle tegnfølger direkte i programteksten. I det følgende vil vi se, hvorledes en tegnfølge kan angives ved hjælp af en såkaldt strengvariabel, men for at gøre det lettere at forstå dette begreb, vil vi først antyde, hvorledes en tegnfølge gemmes i datamatens lager. Vi tænker os lageret opdelt i en mængde små enheder (oktetter), hvor hver enhed kan rumme netop ét tegn. Vi må også tænke os, at systemet hele tiden har nøje rede på, hvad der ligger i lageret, og hvor det ligger. Vi kan altså forestille os, at tegnfølgen

ANDERS HANSEN, ULDGADE 5

ligger lagret således:

Til dette "lagerbillede" bemærker vi for det første, at de to visere skal antyde, at systemet har oplysninger om, hvor tegnfølgen begynder, og hvor den slutter. Endvidere angiver tegnet Δ, at koden for mellemrum ligger lagret på det pågældende sted. Tegnfølgen slutter med cifret 5, og i denne sammenhæng vil dette tal blive behandlet som et tegn på lige fod med bogstaver, specialtegn og mellemrum. Når et tal er lagret på denne måde, kan man ikke benytte det i beregninger efter de regler, som gælder for numeriske konstanter.
Man kan nu ved hjælp af en særlig type af variable, nemlig de ovenfor nævnte strengvariable, henvise til sådanne dele af lageret, som indeholder tegnfølger. En strengvariabel angives ved et sædvanligt variabelnavn efterfulgt af tegnet $. Således er NAVN$, ORDRE$, ADRESSE$ og STRENG$ eksempler på strengvariable i COMAL.
På grund af den særlige måde, hvorpå tegnfølger lagres, skal strengvariable altid dimensioneres. Hertil anvender man DIM-sætninger (jvf. side 33), og strengvariable kan

dimensioneres i samme sætning som evt. vektorer eller matricer. Ved dimensioneringen af en strengvariabel angiver man det maksimale antal tegn, man i givet fald ønsker at gemme i det lagerafsnit, den variable henviser til.

Eksempel 20.1.
Sætningen

```
DIM LISTE$(200), ADR$(600), TABEL(5,20)
```

vil, når den udføres, bevirke at der afsættes plads til 6×21 tal i matricen TABEL, og at der afsættes to felter i lageret med plads til hhv. 200 og 600 tegn. I resten af programmet kan man henvise til disse to felter ved brug af navnene LISTE$ og ADR$.
□

21. Processer med tegnfølger.

Når man ønsker at skrive programmer for processer med tegnfølger, anvender man de samme COMAL-ord, som bruges i forbindelse med behandling af numeriske data. Der er dog på mange måder en afgørende forskel på behandlingen af tegnfølger og behandlingen af numeriske data. I det følgende vil vi i en række eksempler søge at få indblik i, hvorledes man kan programmere processer med tegnfølger.

Eksempel 21.1.
I et program skal man flere steder bruge teksten

```
UDREGNINGERNE GIVER I DETTE TILFAELDE:
```

Man kan naturligvis anføre denne tekst de steder i programmet, hvor man ønsker den udskrevet, men ved hjælp af en strengvariabel kan man gøre det på en mere praktisk måde ved fx. at indlede programmet med følgende:

```
10 DIM TEKST$(40)
20 LET TEKST$="UDREGNINGERNE GIVER I DETTE TILFAELDE: "
```

Når linje 20 er udført vil den variable TEKST$ vise hen til et afsnit af lageret, hvor følgen "UDREGNINGERNE GIVER I DETTE TILFÆLDE: " ligger gemt. Dette kan vi benytte senere i programmet i evt. PRINT-sætninger, idet vi fx. kan skrive:

```
110 PRINT TEKST$; RESULTAT
```

Når denne sætning udføres, har den numeriske variabel RESULTAT i mellemtiden fået tildelt en værdi, og systemet vil nu udskrive den tegnfølge, vi lagrede i linje 20, sammen med værdien af RESULTAT. Tænker vi os, at RESULTAT har værdien 55.45, får vi altså udskriften:

```
UDREGNINGERNE GIVER I DETTE TILFÆLDE:  55.45
```

□

Eksemplet viser, hvorledes en strengvariabel kan benyttes i en LET- og en PRINT-sætning. I eksemplets linje 20 knyttes den variable TEKST$ til den tegnfølge, der er anført i linjen. Man siger også, at TEKST$ har fået tildelt den pågældende følge som værdi, eller at følgen er blevet indlæst i den variable TEKST$.

Eksempel 21.2.
Strengvariable kan også tildeles værdier ved hjælp af DATA-READ-sætninger (jvf. §6, p.7 ff.), og man kan have strengvariable og numeriske variable mellem hinanden i READ-sætningen, blot man sørger for, at de i DATA-sætningen anførte data er af samme type som de variable, de skal indlæses i. Man må naturligvis ikke forsøge at læse en følge ind i en numerisk variabel eller en numerisk konstant ind i en strengvariabel (jvf. bemærkningen på side 49 om cifre i tegnfølger).

Eksempel 21.2.
Når linjerne

```
50 DATA "OLE KØRER",3,"OMGANGE PAA",25,"MINUTTER"
60 READ NAVN$,TID,TUR$,ANTAL,ENHED$
```

er udført, vil sætningen

```
PRINT NAVN$;ANTAL;TUR$;TID;ENHED$
```

give udskriften:

```
OLE KØRER 25 OMGANGE PAA 3 MINUTTER
```

□

Man kan også indlæse tegnfølger i strengvariable i INPUT-sætninger, og i PRINT-sætninger kan man blande strengvariable med direkte anførte tegnfølger. På den måde kan man fx. indsætte forskellige navne i en iøvrigt fast formular.

Eksempel 21.3.
Nedenstående program kan anvendes ved juletid:

```
10 DIM AFS$(20),MODT$(20)
20 INPUT "AFSENDER OG MODTAGER: ",AFS$,MODT$
30 PRINT
40 PRINT "KAERE ";MODT$
50 PRINT "DU ØNSKES EN RIGTIG GLAEDELIG JUL"
60 PRINT "SAMT ET GODT NYTAAR AF"
70 PRINT "          DIN ";AFS$
```

I linje 20 skal man indtaste navnet på afsenderen og modtageren af meddelelsen, fx. TANTE IDA og JULIUS. Programmet vil da give følgende udskrift:

```
KAERE JULIUS
DU ØNSKES EN RIGTIG GLAEDELIG JUL
SAMT ET GODT NYTAAR AF
          DIN TANTE IDA
```

Læg mærke til de i teksten anbragte mellemrum. I linje 40 står der således: "KAERE "; der er altså anbragt et mellemrum efter E'et. Hvis man ikke skriver således, vil tegnet ; bevirke, at de to tekster sammenskrives. Således ville

```
40 PRINT "KAERE";MODT$
```

give udskriften

KAEREJULIUS

og det ville måske ikke bringe modtageren i den rette stemning.

□

22. Mere om processer med tegnfølger.

I det foregående afsnit har vi set eksempler på, hvorledes man kan have nytte af strengvariable. COMAL giver imidlertid muligheder for en mere avanceret anvendelse af tegnfølger. For at få indblik i disse muligheder vil vi gennemgå et eksempel.

Eksempel 22.1.

Vi ønsker at skrive et program, som kan hjælpe os med at føre kartotek over nogle varekoder. Hver vare angives ved tre bogstaver og to cifre, fx. SKF45, FDB18 og HKH56. Varekoderne skal kunne indtastes fra et tastatur og derefter lagres i datamatens lager. Kartoteket skal organiseres således, at man til enhver tid kan få en oversigt over, hvad der står i det, kan få at vide, om en bestemt kode står i det eller ej, kan indsætte nye koder, slette allerede eksisterende og bytte om på to koder i kartoteket. Idet vi regner med, at vi aldrig vil få mere end 100 koder i kartoteket, skal vi afsætte plads i lageret til ialt 5·100 = 500 tegn. Vi dimensionerer derfor en strengvariabel KARTOTEK$ til 500. Som hjælpevariabel (buffer) for indtastningen dimensionerer vi endvidere en strengvariabel KODE$ til 5 tegn. De to variable dimensioneres i

```
10 DIM KARTOTEK$(500),KODE$(5)
```

Vi skriver derpå den del af programmet, som skal administrere indtastningen:

```
20 INPUT "VAREKODE: ",KODE$
30 WHILE KODE$<>"SLUT" THEN DO
40   KARTOTEK$=KARTOTEK$,KODE$
50   INPUT "NAESTE VAREKODE: ",KODE$
60 ENDWHILE
```

Når sætning 10 er udført, er der afsat to lagerafsnit i datamatens lager, til hvilke man kan henvise med navnene hhv. KARTOTEK$ og KODE$. Vi ved ikke noget om, hvad der i øjeblikket er gemt i disse afsnit, men som regel vil det være affald fra tidligere processer. Når sætning 20 udføres, vil systemet først skrive

```
VAREKODE:
```

ud på terminalen og derpå vente på, at operatøren skal indtaste en varekode. Lad os tænke os, at koden SKF57 indtastes. Disse fem tegn vil da blive lagret i det afsnit, som den variable KODE$ viser hen til. Vi vil illustrere dette således:

I sætning 30 finder vi det Boolske udtryk: KODE$<>"SLUT". Dette udtryk vil have værdien *sand*, hvis tegnfølgen i KODE$ er forskellig fra "SLUT", og værdien *falsk*, netop når indholdet i KODE$ er lig med "SLUT". Da det ord, operatøren har indtastet, ikke er SLUT, udføres linjerne 40 og 50. Linje 40 indeholder tildelingen:

```
LET KARTOTEK$=KARTOTEK$,KODE$
```

som bevirker, at den tegnfølge, der står i KODE$, bliver kædet sammen med den følge, som står i KARTOTEK$, hvorpå resultatet af sammenkædningen læses ind i KARTOTEK$. Sammenkædningstegnet (sammenkædningsoperatoren) er tegnet komma (,), som ikke selv bliver læst med ind under operationen. Da der ikke hidtil har stået noget, som vedrører dette program, i KARTOTEK$, vil resultatet blive, at følgen "SKF57" indlæses i denne variabel som dens fem første tegn:

Samtidigt opretter systemet en viser (pointer), som skal holde rede på, hvor det hidtil sidste tegn er indlæst. Når linje 50 udføres, skriver systemet først:

NAESTE VAREKODE:

på terminalen og går i ventetilstand. Vi tænker os, at operatøren indtaster koden FDB89. Disse fem tegn vil blive gemt i KODE$ med det resultat, at den kode, som stod der i forvejen, bliver overskrevet. Status for KODE$ er altså nu:

KODE$ → | F | D | B | 8 | 9 |

Efter linje 60 med ENDWHILE udføres linje 30 atter, og da KODE$ stadig er forskellig fra "SLUT", udføres linje 40 og 50 endnu engang. I linje 40 kædes indholdet af KARTOTEK$ sammen med indholdet af KODE$, og resultatet indlæses i KARTOTEK$. Vi kan illustrere det således:

KARTOTEK$ → | S | K | F | 5 | 7 | F | D | B | 8 | 9 | ...

Således fortsættes indlæsningen, og kode efter kode bliver hægtet på KARTOTEK$, indtil operatøren indtaster ordet SLUT. Så bliver udsagnet i linje 30 falsk, og indlæsningen standser. Læg mærke til, at ordet SLUT ikke bliver lagret i KARTOTEK$, da linjerne 40 og 50 som bekendt (jvf. p. 29) ikke bliver udført, når det Boolske udtryk i WHILE-sætningen er falsk.

Vi vil dernæst se på, hvorledes vi kan få udskrevet en liste over de koder, som findes i KARTOTEK$. For en ordens og oversigts skyld ønsker vi ikke at få denne liste skrevet ud som én lang, sammenhængende tegnfølge, men vil have de enkelte varekoder udskrevet hvor for sig. Vi skal derfor have koderne pillet ud én for én af strengen KARTOTEK$. Dette gør vi ved hjælp af nogle ekstra visere, som vi kan få systemet til at sætte på tegnfølgen. Vi tænker os, at vi gerne vil have fat i den kode, der står anført som den tredje i KARTOTEK$. Da hver kode er på fem tegn, må den tredje kode indbefatte tegnene fra og med nummer 11 til og med nummer 15. Vi kan afbilde situationen således:

KARTOTEK$ → | S | K | F | 5 | 7 | F | D | B | 8 | 9 | H | K | H | 1 | 5 | J | A | P | 2 | 9 | ...

11 15

På tilsvarende måde kan vi regne ud, at kode nummer I må indbefatte tegnene fra og med nummer 5*I-4 til og med 5*I (regn efter!). Vi kan altså let beregne de to visere, som udpeger den kode, vi er interesseret i. Kalder vi de to visere, som udpeger første og sidste tegn i den I'te kode, for hhv. P1 og P2, udføres udregningen lettest ved, at vi først udregner P2 og derefter P1:

```
LET P2=5*I; P1=P2-4
```

Helt generelt har vi følgende billede:

idet BSA92 er kode nummer I. I programmet angives det udpegede felt ved

```
KARTOTEK$(P1,P2)
```

der kan læses således: Den tegnfølge, der står i KARTOTEK$ fra og med plads nummer P1 til og med plads nummer P2. Inden vi kan udskrive fortegnelsen, må vi også vide, hvor den sidste varekode står. Hertil benytter vi den viser, som systemet sætter på strengens sidste tegn. I COMAL findes en funktion, som gør det muligt for os at få at vide, hvor denne viser står. Funktionen hedder LEN (length), og dens definitionsmængde er mængden af alle strengvariable. Hvis det hidtil sidste tegn i KARTOTEK$ fx. står på plads nummer 115, har vi at

```
LEN(KARTOTEK$)
```

har værdien 115, og vi kan bl.a. regne ud, at der må være indlæst ialt 115/5 = 23 varekoder i kartoteket. Nu kan vi skrive det programafsnit, i hvilket indholdet af kartoteket udskrives:

```
100 FOR I=1 TO LEN(KARTOTEK$)/5
110   LET P2=5*I; P1=P2-4
120   PRINT KARTOTEK$(P1,P2)
130 NEXT I
```

Udskrivningen kan også klares af følgende program, hvor udregningen af viserne måske er lidt simplere:

```
100 P1=1; P2=5
110 WHILE P1<=LEN(KARTOTEK$) DO
120   PRINT KARTOTEK$(P1,P2)
130   P1=P1+5; P2=P2+5
140 ENDWHILE
```

Udskriften vil for begge programmers vedkommende få følgende format:

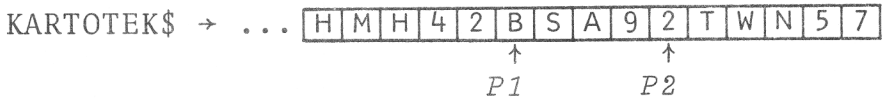
```
SKF57
FDB89
HKH15
JAP29
...
```

Ved hjælp af de visere, vi har indført i det foregående, kan vi let skrive et programafsnit, som kan bruges til at søge strengen igennem og se efter, om en nærmere angivet varekode findes i kartoteket. Vi begynder at søge fra en ende af, idet vi blot piller koderne ud én for én og sammenligner med den kode, vi har angivet, indtil vi finder den i kartoteket, eller vi har bladet hele kartoteket igennem og konstaterer, at den ikke er der:

```
200 INPUT "DEN SØGTE KODE: ",KODE$
210 LET I=0
220 REPEAT    //SØG EFTER KODEN//
230   LET I=I+1; P2=5*I; P1=P2-4
240 UNTIL KODE$=KARTOTEK$(P1,P2) OR P1>LEN(KARTOTEK$)
250 IF KODE$=KARTOTEK$(P1,P2) THEN DO
260   PRINT "DEN SØGTE KODE FINDES PÅ LISTEN SOM NR.";I
270 ELSE
280   PRINT "DEN SØGTE KODE FINDES IKKE PÅ LISTEN."
290 ENDIF
```

I linje 200 indtastes den søgte varekode, og efter at I er tildelt værdien 0 i linje 210, fortsættes udførelsen med REPEAT-løkken (220-240) indtil den indtastede kode er fundet, eller vi når ud over det sidste tegn i strengen KARTOTEK$ (P1>LEN(KARTOTEK$)). Hvis koden er fundet, udføres PRINT-sætningen i linje 260, og ellers udføres PRINT-sætningen i linje 280. Det kan bemærkes, at vi ikke blot får at vide, om koden findes, men også hvilket nummer i kartoteket, den i givet fald har.

Hvis vi ønsker at slette en varekode fra kartoteket, benytter vi først det netop opstillede program til at finde koden. Ved denne fremgangsmåde får vi, som nævnt ovenfor, at vide hvor den pågældende kode står, idet tælleren I ved udløbet af løkken vil indeholde kodens nummer i rækken af øvrige koder. Vi tænker os, at vi har fundet den kode, der skal slettes, og afbilder situationen således:



hvor P2=5*I og P1=P2-4. Den udpegede kode kan nu ganske enkelt slettes ved, at vi lader den efterfølgende del af strengen rykke 5 pladser til venstre. Det nye indhold i KARTOTEK$ kan altså bestemmes således: Den delfølge af KARTOTEK$, som indbefatter tegnene fra og med tegn nummer P1 og følgen ud, skal erstattes af den delfølge, der indeholder tegnene fra og med nummer P2+1 (se på tegningen ovenfor!) til og med det sidste tegn i KARTOTEK$. Disse to delfølger kan i COMAL angives ved hhv.

```
KARTOTEK$(P1) og KARTOTEK$(P2+1)
```

idet fx. KARTOTEK$(P1) betyder: Den tegnfølge, som står i KARTOTEK$ fra og med tegn nummer P1 og tegnfølgen ud. For at slette den angivne kode, behøver vi altså blot én linje i programmet:

```
350 LET KARTOTEK$(P1)=KARTOTEK$(P2+1)
```

Det skal også være muligt at bytte om på to koder, som står i kartoteket. Vi kan tænke os, at varekoden, der står som nummer I, skal bytte plads med varekoden, der står som nummer J. Efter udregning af de fire visere, der skal udpege de to koder, benytter vi i princippet den samme algoritme, som er beskrevet i eksempel 11.2, side 21:

```
400 LET P2=5*I; P1=P2-4; P4=5*J; P3=P4-4
410 LET KODE$=KARTOTEK$(P1,P2)
420 LET KARTOTEK$(P1,P2)=KARTOTEK$(P3,P4)
430 LET KARTOTEK$(P3,P4)=KODE$
```

Dersom vi ønsker at indtaste nye varekoder, vender vi blot tilbage til det program, vi anvendte ved indtastningen af kartoteket fra begyndelsen af (p. 53).

Vi har nu opstillet en række enkeltprogrammer, som hver for sig løser én af de opgaver, vi stillede os i begyndelsen af eksemplet. Vi har imidlertid ikke skrevet ét sammenhængende program, som i sin helhed kan hjælpe os ved administrationen af kartoteket. Dette vil vi udskyde til næste kapitel, hvor vi indfører en styrestruktur, som i særlig grad egner sig til at holde rede på en række programmer, der hver for sig udfører specielle dele af en større proces.

□

I eksemplet har vi vist en række af de processer, vi kan udføre med tegnfølger i COMAL. Vi vil ganske kort resumere:

Tegnfølger kan sammenkædes. Således vil sætningen

```
LET NAVN$=FORNAVN$,EFTNAVN$
```

resultere i, at tegnfølgen i FORNAVN$ kædes sammen med tegnfølgen i EFTNAVN$, og at resultatet af sammenkædningen indlæses i NAVN$. Endvidere gælder følgende for en forelagt strengvariabel NAVN$:

NAVN$	angiver hele følgen i NAVN$
NAVN$(8)	angiver den del af følgen i NAVN$, som indbefatter tegnene fra og med nummer 8 i følgen til og med det sidste tegn i denne.
NAVN$(I)	hvor I er et naturligt tal, angiver den del af følgen i NAVN$, som begynder med det I'te tegn i følgen og indbefatter resten af denne.
NAVN$(3,7)	angiver den del af tegnfølgen i NAVN$, som indbefatter tegnene fra og med nummer 3 til og med nummer 7.
NAVN$(I,J)	hvor I og J er naturlige tal, og $I \leq J$, an-

giver den del af følgen i NAVN$, som indbefatter tegnene fra og med nummer I til og med nummer J.

NAVN$(7,7) angiver tegn nummer 7 i følgen, som er tildelt NAVN$.

Eksempel 22.2.

Vi ønsker at skrive et program ved hjælp af hvilket vi kan sortere en liste med personnavne, således at navnene kommer til at stå i alfabetisk orden. Navnene skal indtastes og lagres således: <efternavn><kaldenavn>, altså fx.

```
JESPERSON ANDERS
```

eller - hvis kaldenavnet er dobbelt -

```
LUND HANS HENRIK
```

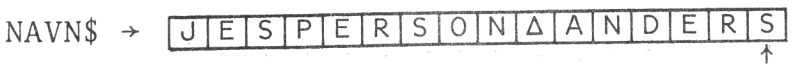
Vi vil gemme navnene i en strengvariabel LISTE$, og som hjælpevariabel (buffer) ved indtastningen vil vi bruge en strengvariabel NAVN$. Vi regner med, at intet personnavn er på mere end 30 bogstaver, og at vi ikke skal sortere mere end højst 50 navne. Vi kan derfor dimensionere vore variable således:

```
DIM NAVN$(30), LISTE$(1500)
```

Vi forsøger at bruge programmet fra eksempel 22.1 (p. 53) som model for indlæsningsdelen:

```
20 INPUT "FØRSTE NAVN: ",NAVN$
30 WHILE NAVN$<>"SLUT" DO
40   LISTE$=LISTE$,NAVN$
50   INPUT "NYT NAVN: ",NAVN$
60 ENDWHILE
```

Dette program er tidligere beskrevet i princippet, og vi tænker os, at det første navn, der indtastes er JESPERSON ANDERS. Status for NAVN$ vil efter indtastningen være:



hvor viseren og prikkerne antyder, at NAVN$ ikke er

fyldt op. Dette vil i almindelighed være tilfældet, idet det er yderst sjældent, at et kaldenavn og et efternavn tilsammen er på 30 tegn. Når det indtastede navn derefter hægtes på følgen i LISTE$, fylder det stadigvæk mindre end 30 tegn (in casu: 16), og efter yderligere nogle indtastninger af navne, bliver det svært at holde rede på, hvor ét navn ophører og et nyt begynder. Hvis vi senere skal have navnene i LISTE$ læst ud, bliver vi nødt til enten at sætte et særligt skilletegn mellem de enkelte navne eller sørge for at fylde NAVN$ op med tegn, så følgen i den kommer til at omfatte netop 30 tegn. I dette eksempel vil vi vælge den sidste udvej, og vi vil fylde op med mellemrum ved hjælp af en såkaldt fejekost (broom: heksekost). Indlæsningprogrammet kommer til at se således ud:

```
10 DIM NAVN$(30), FEJEKOST$(30), LISTE$(1500)
20 FEJEKOST$="                              "
30 INPUT "FØRSTE NAVN: ",NAVN$
40 WHILE NAVN$<>"SLUT" DO
50   NAVN$=NAVN$,FEJEKOST$; LISTE$=LISTE$,NAVN$
60   INPUT "NYT NAVN: ",NAVN$
70 ENDWHILE
```

Når linje 20 er udført, vil FEJEKOST$ indeholde 30 mellemrum (tæl efter!). I linje 30 indtastes navnet ganske som før, og i linje 40 ser systemet efter, om NAVN$ er forskellig fra "SLUT". Lad os tænke os, at det indtastede navn er OLSEN OLE. Status for NAVN$ vil da være:

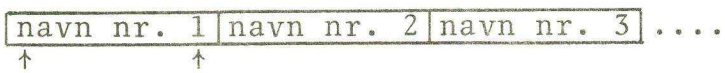
```
NAVN$ → OLSENΔOLE...
          ↑
```

idet vi fra nu af udelader de skillestreger, vi hidtil har benyttet ved lagerbilleder. Den første tildeling i linje 50 bevirker, at de tiloversblevne pladser i NAVN$ fyldes op med mellemrum (NAVN$ "fejes ren"). Efter at den nævnte tildeling er udført, vil status for NAVN$ altså være

```
NAVN$ → OLSENΔOLEΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ
                                    ↑
                                    30
```

Når den anden tildeling i linje 50 derpå udføres, bliver tegnene i NAVN$ læst ind på de første 30 pladser i LISTE$, og det næste navn, som indtastes, bliver først fejet ren og derpå indlæst på de næste 30 pladser i LISTE$. På den måde fortsætter vi, indtil der ikke er flere navne, som skal med på listen. Så indtaster vi ordet SLUT, og programmet går videre med eventuelle sætninger efter linje 70. Vi har ved at benytte denne fremgangsmåde fået en simpel opdeling af LISTE$ af samme slags, som den vi benyttede i forbindelse med KARTOTEK$ i eksempel 22.1. Der ganske vist gået en del plads til spilde i lageret, men det må man acceptere, hvis man ønsker let adgang til de lagrede oplysninger.

Navnene i LISTE$ skal derpå sorteres, så de kommer til at stå i alfabetisk orden. Vi benytter følgende fremgangsmåde ved denne sortering: Først ser vi på det første navn på listen:



Idet vi husker, at hvert navnefelt fylder 30 tegn. Derpå henter vi det andet navn frem, og sammenligner det med det første navn. Hvis navn nr. 2 kommer før navn nr. 1 i alfabetisk orden, lader vi de to navne bytte plads, ellers lader vi navn nr. 2 stå og går videre med navn nr. 3. Også dette sammenlignes med navn nr. 1, og hvis navn nr. 3 kommer før navn nr. 1 i alfabetisk orden, skal de to navne bytte plads, ellers lader vi navn nr. 3 stå og fortsætter med navn nr. 4. På den måde fortsætter vi, indtil vi har været hele listen igennem. Man vil indse (gør det!), at efter denne gennemgang af listen vil det navn, der nu står på første plads, også være det første i alfabetisk orden. Dernæst ser vi på navn nr. 2 på listen og sammenligner som før dette navn med de efterfølgende navne ét for ét. Hvis vi finder et navn, som kommer før navn nr. 2 i alfabetisk orden, bytter vi de to navne om. Når denne runde er tilende, vil det navn, som er det næst-

```
* 290 PRINT T;TAB(5);LISTE$(I1,I2)
285
RUN
KLASSENS NAVN: 345
NYT NAVN: 3456
NYT NAVN: 2134
NYT NAVN: 1237
NYT NAVN: 9876
NYT NAVN: 4560
NYT NAVN: SLUT
 KLASSE        345

 1    1237
 2    2134
 3    3456
 4    4560
 5    9876

END AT 0310

* PUNCH
0010 DIM NAVN$(30),FEJEKOST$(30),LISTE$(1500)
0020 DIM KLAS$(20)
0030 LET T=-1
0040 LET FEJEKOST$="                              "
0050 INPUT "KLASSENS NAVN: ",KLAS$
0060 WHILE NAVN$<>"SLUT" THEN DO
0070   LET NAVN$=NAVN$,FEJEKOST$; LISTE$=LISTE$,NAVN$
0080   INPUT "NYT NAVN: ",NAVN$
0090 ENDWHILE
0100 PRINT "<14> KLASSE",KLAS$
0110 PRINT
0120 LET ANTAL=LEN(LISTE$)/30
0130 FOR I=1 TO ANTAL
0140   REM //SE PAA DET I'TE NAVN//
0150   LET I2=30*I; I1=I2-29
0160   FOR J=I+1 TO ANTAL
0170     REM //SAMMENLIGN MED DET J'TE NAVN//
0180     LET J2=30*J; J1=J2-29
0190     REM //HVIS NAVN NR. J KOMMER FØR NAVN NR. I//
0200     IF LISTE$(J1,J2)<LISTE$(I1,I2) THEN DO
0210       REM //SAA BYT DE TO NAVNE//
0220       LET NAVN$=LISTE$(J1,J2)
0230       LET LISTE$(J1,J2)=LISTE$(I1,I2)
0240       LET LISTE$(I1,I2)=NAVN$
0250     ENDIF //SLUT EVT. OMBYTNING//
0260   NEXT J
0270   LET T=T+1
0280   IF T<>0 THEN DO
0290     PRINT T;TAB(5);LISTE$(I1,I2)
0300   ENDIF
0310 NEXT I
```

77-02-24

30

(-1)

Virker kun 1 gang

```
END AT 0100

*
  30 PRINT"<14>BEDES SKRIVE SIG PÅ LISTEN,"
LRUN

END AT 0100

* NEW
* 10 DIM A(5,4
ERROR 02 - SAETNINGEN ER IKKE RIGTIGT OPBYGGET

10 DIM A(5,4)
FOR
* 20 FOR I=1 TO 5
30 FOR J=1 TO 4
40 READ A(I,J)
50 NEXT I
60 NEXT J
70 MAT PRINT A
5 DATA 3,4,7,6,8,6,7,8,,,3,5,6,7,,,5,2,4,5,34,23,56,67,,3,4
ERROR 02 - SAETNINGEN ER IKKE RIGTIGT OPBYGGET

5 DATA 34,56,34,56,78,23,7,89,6,5,3,4,6,7,8,9,3,4,5,6,7,8
LIST
0005 DATA 34,56,34,56,78,23,7,89,6,5,3,4,6,7,8,9,3,4,5,6,7,8
0010 DIM A(5,4)
0020 FOR I=1 TO 5
0030   FOR J=1 TO 4
0040     READ A(I,J)
0050   NEXT I
0060 NEXT J
0070 MAT PRINT A

RUN

ERROR 22 AT 0060 - NEXT UDEN FOR

* 70 MAT PRINT A;
```

```
RUN

ERROR 22 AT 0060 - NEXT UDEN FOR

* NEW
* 10 DATA 34,5,6,7,8,9,,,4,4,5,6,7,3,2,4,6,7,899,7,3,4,5,7,6,8
20DIM A(5,4)
30 FOR I=1 TO 5
40 FOR J = 1 TO 4
50 READ A(I,J)
60 NEXT J
70 NEXT I
80 MAT PRINT A;
RUN

 34   5   6   7
 8   9   4   4
 5   6   7   3
 2   4   6   7
 8   7   3   4

END AT 0080

* 20 DIMA(4,5
ERROR 02 - SAETNINGEN ER IKKE RIGTIGT OPBYGGET

20 DIM A(4,5)
RUN

ERROR 31 AT 0050 - INDEKS OVERSKRIDER DIMENSION

* 50 READ A(J,I)
RUN

 34   8   5   2   8
 5   9   6   4   7
 6   4   7   6   3
 7   4   3   7   4

END AT 0080
```

```
LIST
0010 DIM A(4,4)
0020 FOR I=0 TO 4
0030   FOR J=0 TO 4
0040     READ A(I,J)
0050   NEXT J
0060 NEXT I
0070 MAT PRINT A;
0080 DATA 1,4,7,4,9,3,6,7,8,4,23,7,4,2,8,4,7,9,2,1,7,9,3,5,6
0090 IF A(2,3)=2 THEN DO
0100   IF A(3,2)=9 THEN DO
0110     PRINT A(1,1)
0115     PRINT
0120   ENDIF
0125   PRINT
0130 ENDIF
0140 IF A(1,1)=6 THEN DO
0150   IF A(3,4)=1 THEN DO
0160     IF A(4,4)=6 THEN DO
0170       PRINT "A(4,1) = ",A(4,1),"A(3,3) = ",A(3,3),
0172     ENDIF
0174   ENDIF
0176 ENDIF
0180 PRINT

RUN

 6   7   8   4
 7   4   2   8
 7   9   2   1
 9   3   5   6
 6

A(4,1) =          9            A(3,3) =          2

END AT 0180

*
```

```
* NEW
* ENTER "$PTR
* RUN
KLASSENS NAVN: 56
NYT NAVN: 34
NYT NAVN: 45
NYT NAVN: 23
NYT NAVN: 6
NYT NAVN: 8
NYT NAVN: SLUT
 KLASSE        56

 Ø
 1 23
 2 34
 3 45
 4 6
 5 8

END AT Ø36Ø

* RUN
KLASSENS NAVN: 8B
NYT NAVN: PETER
NYT NAVN: JORGEN
NYT NAVN: JENS
NYT NAVN: AAGE
NYT NAVN: ANE
NYT NAVN: SLUT
 KLASSE        8B

 Ø
 1 AAGE
 2 ANE
 3 JENS
 4 JORGEN
 5 PETER

END AT Ø36Ø

* 351 LET T = T+1
352 IF T<>Ø THEN DO
355 ENDIF
RUN
KLASSENS NAVN: 45
NYT NAVN: PETER
NYT NAVN: SOREN
```

```
NYT NAVN: JENS
NYT NAVN: HANS
NYT NAVN: ANE
NYT NAVN: AAGE
NYT NAVN: SLUT
 KLASSE        45

 1 AAGE
 2 ANE
 3 HANS
 4 JENS
 5 PETER
 6 SOREN

END AT 0360

* RENUMBER
* PUNCH
0010 DIM NAVN$(30),FEJEKOST$(30),LISTE$(1500)
0020 DIM KLAS$(20)
0030 LET T=-1                                    30
0040 LET FEJEKOST$="                                "
0050 INPUT "KLASSENS NAVN: ",KLAS$
0060 WHILE NAVN$<>"SLUT" THEN DO
0070   LET NAVN$=NAVN$,FEJEKOST$; LISTE$=LISTE$,NAVN$
0080   INPUT "NYT NAVN: ",NAVN$
0090 ENDWHILE
0100 PRINT "<14> KLASSE",KLAS$
0110 PRINT
0120 LET ANTAL=LEN(LISTE$)/30
0130 FOR I=1 TO ANTAL
0140   REM //SE PAA DET I'TE NAVN//
0150   LET I2=30*I; I1=I2-29
0160   FOR J=I+1 TO ANTAL
0170     REM //SAMMENLIGN MED DET J'TE NAVN//
0180     LET J2=30*J; J1=J2-29
0190     REM //HVIS NAVN NR. J KOMMER FØR NAVN NR. I//
0200     IF LISTE$(J1,J2)<LISTE$(I1,I2) THEN DO
0210       REM //SAA BYT DE TO NAVNE//
0220       LET NAVN$=LISTE$(J1,J2)
0230       LET LISTE$(J1,J2)=LISTE$(I1,I2)
0240       LET LISTE$(I1,I2)=NAVN$
0250     ENDIF //SLUT EVT. OMBYTNING//
0260   NEXT J
0270   LET T=T+1
0280   IF T<>0 THEN DO
0290     PRINT T;LISTE$(I1,I2)
0300   ENDIF
0310 NEXT I
```

```
RUN

END AT 0010

* ENTER "$PTR
* RUN
KLASSENS NAVN: 1977
NYT NAVN: 5876
NYT NAVN: 4567
NYT NAVN: 9876
NYT NAVN: 4321
NYT NAVN: 2341
NYT NAVN: 0234
NYT NAVN: 0067
NYT NAVN: 0009
NYT NAVN: 0008
NYT NAVN: 0007
NYT NAVN: SLUT
 KLASSE        1977

 1 0007
 2 0008
 3 0009
 4 0067
 5 0234
 6 2341
 7 4321
 8 4567
 9 5876
 10 9876

END AT 0310

* RUN
KLASSENS NAVN: 1977 8B
NYT NAVN: PETER
NYT NAVN: JORGEN
NYT NAVN: JENS
NYT NAVN: ARNE
NYT NAVN: AAGE
NYT NAVN: SLUT
 KLASSE        1977 8B

 1 AAGE
 2 ARNE
 3 JENS
 4 JORGEN
 5 PETER

END AT 0310
```

Fejlkast 30

Virker hver gang

første i alfabetisk orden, stå på plads nr. 2. Sådan fortsætter vi, indtil hele navnelisten er sorteret. For at illustrere fremgangsmåden viser vi herunder, hvorledes den virker på en liste bestående af 5 fornavne. Læseren kan evt. skrive navnene op på 5 stykker papir, lægge dem i række på bordet og udføre en konkret sortering af dem efter den beskrevne metode:

0.	KURT	TAGE	JENS	KARL	PETER
1.	JENS	TAGE	KURT	KARL	PETER
2.	JENS	KARL	TAGE	KURT	PETER
3.	JENS	KARL	KURT	TAGE	PETER
4.	JENS	KARL	KURT	PETER	TAGE

Den programdel, som skal udføre sorteringen, kan se således ud:

```
200 ANTAL=LEN(LISTE$)/30
210 FOR I=1 TO ANTAL-1
220   REM   //SE PAA DET I'TE NAVN//
230   I2=30*I; I1=I2-29

250   FOR J=I+1 TO ANTAL
260     REM   //SAMMENLIGN MED DET J'TE NAVN//
270     J2=30*J; J1=J2-29
280     REM   //HVIS NAVN NR.J KOMMER FØR NAVN NR.I//
290     IF LISTE$(J1,J2)<LISTE$(I1,I2) THEN DO
300       REM   //SAA BYT DE TO NAVNE//
310       NAVN$=LISTE$(J1,J2)
320       LISTE$(J1,J2)=LISTE$(I1,I2)
330       LISTE$(I1,I2)=NAVN$
340     ENDIF   //SLUT EVT. OMBYTNING//
350   NEXT J
360 NEXT I
```

I linje 290 finder vi det Boolske udtryk

```
LISTE$(J1,J2)<LISTE$(I1,I2)
```

der har værdien *sand*, hvis og kun hvis tegnfølgen i LISTE$(J1,J2) kommer før tegnfølgen i LISTE$(I1,I2) i sædvanlig alfabetisk orden (leksikografisk orden).

□

I processer med tegnfølger kan vi benytte de Boolske

relationsoperatorer (jvf. p. 17), idet disse virker med hensyn til sædvanlig alfabetisk ordning (leksikografisk ordning). I denne sammenhæng kan tegnene passende fortolkes således:

=	er lig med
<	kommer før end
>	følger efter
<=	kommer før end eller er lig med
>=	følger efter eller er lig med
<>	er forskellig fra

Eksempel 22.3.
Efter at sætningen

```
FORNAVN$="OLE"; EFTNAVN$="OLESEN"
```

er udført, vil

FORNAVN$<EFTNAVN$	have værdien *sand*
FORNAVN$=EFTNAVN$	have værdien *falsk*
FORNAVN$=EFTNAVN$(1,3)	have værdien *sand*
"OLSEN">EFTNAVN$	have værdien *sand*
"OLSEN">FORNAVN$	have værdien *sand*
"OLSEN"<>EFTNAVN$	have værdien *sand*

□

RESUMÉ.

I dette kapitel har vi hørt om:

Tegnfølger, strengvariable, en følges længde, funktionen LEN, dimension af en strengvariabel, sammenkædning af tegnfølger, delfølger, fejekoste og relationsoperatorer i forbindelse med leksikografisk ordning.

KAP. V.
MULTIFORGRENINGER OG UNDERPROGRAMMER.

23. Multiforgreninger.

Hidtil har vi kun skrevet korte programmer, som hver for sig tjener til at udføre ret simpel, usammensatte processer. I dette kapitel beskriver vi nogle COMAL-sætninger, som kan anvendes, når man ønsker at skrive mere omfattende programmer, og som gør det muligt for os at bevare overblikket over sådanne programmer, selv om de måske bliver på flere sider.

Eksempel 23.1.
I eksempel 22.1 (p. 53 - 59) fik vi skrevet en række programmer, som hver for sig udfører nogle processer i forbindelse med et kartotek over varekoder. Vi vil se, hvorledes vi stiller disse programdele sammen til en helhed. Som nævnt i begyndelsen af eksempel 22.1 skal vi kunne: 1) indtaste koder (enkeltvis eller flere i række), 2) få udskrevet en oversigt over koderne i kartoteket, 3) søge efter en given kode, 4) slette en nærmere angivet kode og 5) bytte om på to koder i kartoteket. Vi vil indrette det således, at operatøren ved en simpel kommando kan bestemme, hvilken af disse fem operationer, der skal udføres.
Inden vi fortsætter gennemgangen, beder vi læseren finde bilaget til dette eksempel (findes bagest i heftet). Nævnte bilag indeholder et program, som opfylder de krav, vi har anført ovenfor.
I INPUT-sætningen i linje 10 skal et af tallene 1, 2, 3, 4 eller 5 tildeles den variable JOBKODE. Vi kan tænke os, at vi indtaster talværdien 1. Systemet udfører derpå linje 20, som indeholder sætningen:

```
SELECT JOBKODE THEN DO
```

som skal ses i sammenhæng med sætningerne i linierne 30, 100, 160, 280 og 400:

CASE 1, CASE 2, CASE 3, CASE 4 og CASE 5

samt sætningen i linje 470

ENDCASES

SELECT-sætningen har følgende virkning: Hvis JOBKODE har værdien 1, vil systemet udføre (THEN DO) den del af programmet, som er anført mellem CASE 1 og CASE 2. Hvis JOBKODE har værdien 2, vil systemet udføre den del af programmet, som står mellem CASE 2 og CASE 3 og så fremdeles. Når det tilfælde (CASE), som vælgeren (selectoren) peger på, er udført, fortsætter udførelsen af programmet med den sætning, der står efter ENDCASES, hvilket i dette tilfælde vil betyde, at udførelsen standser. Vi har indtastet værdien 1 til JOBKODE, og vi får derfor udført den programdel, som administrerer indtastningen af varekoder.
Vi fortsætter behandlingen af eksemplet nedenfor, men giver først en formel beskrivelse af SELECT-strukturen.
□

Ved hjælp af sætningerne SELECT, CASE og ENDCASES kan vi udpege ét bestemt program af en række sideordnede programmer. Vi har altså med en styrestruktur at gøre, og man taler i dette tilfælde om en multiforgrening. Strukturen er opbygget således:

SELECT udtryk THEN DO

[A]

CASE <liste af hele tal>$_1$

[A_1]

CASE <liste af hele tal>$_2$

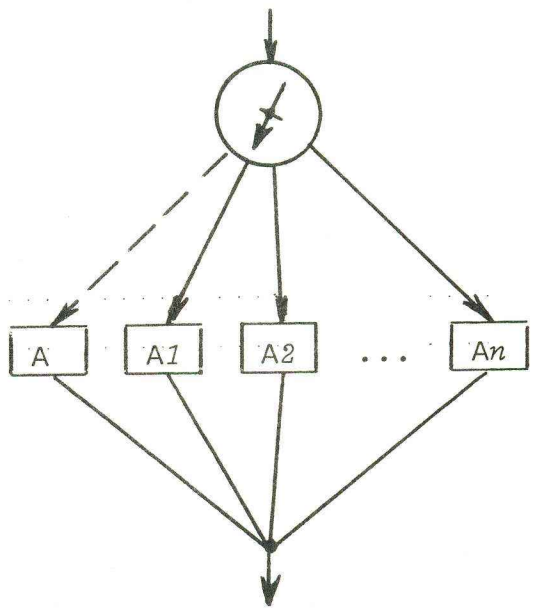
[A_2]

...

CASE <liste af hele tal>$_3$

[A_n]

ENDCASES



Styresætningerne virker således: udtryk er en variabel eller en formel, som kan antage heltallige værdier. Hvis den værdi, der tildeles udtryk findes i en liste efter et CASE, udføres den programdel A_i, som er anført mellem det udpegede CASE og det efterfølgende CASE, eller - hvis det udpegede CASE er det sidste i rækken - mellem det udpegede CASE og ENDCASES. Derefter fortsættes udførelsen af programmet med sætningen efter ENDCASES. Der kan indsættes et delprogram A mellem SELECT-sætningen og den første CASE-sætning. Hvis værdien af udtryk ikke er anført i nogen liste efter et CASE, vil delprogrammet A blive udført og derpå sætningen efter ENDCASES. Hvis delprogrammet A ikke findes, og der ikke udpeges noget CASE, fortsættes uden videre med sætningen efter ENDCASES. Der er ingen begrænsninger i antallet af CASEs efter SELECT, og man kan have så mange SELECT.. ENDCASES inden i hinanden, som man ønsker.
Ved listning af programmet sker automatisk indrykning af alle teksterne til A, A_1, A_2, ..., A_n.

Eksempel 23.2.
Dette eksempel er en fortsættelse af eksempel 23.1. Til nærværende eksempel hører dog et andet bilag, som vi beder læseren have ved hånden under den følgende gennemgang. I det program, der er skrevet til eksempel 23.1, skal vi søge efter en bestemt kode i to tilfælde, nemlig når vi ønsker at se efter, hvor den står - hvis den står nogensteder - og når vi ønsker at slette den (CASE 3 og CASE 4). Vi bliver altså nødt til at skrive en bestemt programdel to gange (lin. 200 - 220 og lin. 320 - 340). Det er ikke særlig praktisk, og i nogle programmer kan man komme ud for, at en bestemt proces skal udføres i mange forskellige sammenhæng, og det ville da være en fordel, hvis man kan skrive det program, som udfører den pågældende proces, én gang og derefter henvise til det, hver gang man har brug for det. Problemet ligner det, vi løser ved at indføre variable, når vi ønsker at kunne benytte en bestemt størrelse i forskellige udtryk.

Et program, som kan bruges på denne måde, kaldes også et underprogram eller en procedure, og i dette eksempel benytter vi to sådanne underprogrammer. Under CASE 3, i hvilket en nærmere angivet kode søges, finder man sætningen (linje 310):

```
EXEC SPORHUND
```

(EXEC: execute), som bevirker, at systemet søger efter en sætning, der skal se således ud:

```
PROC SPORHUND
```

(PROC: procedure). Denne sætning findes i linje 550, og systemet vil nu udføre det program, som står anført i linjerne 570 - 660, og som slutter med ordet:

```
ENDPROC
```

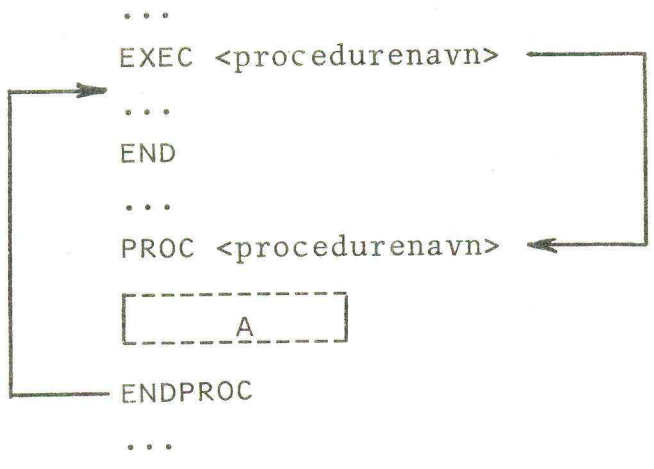
Dette underprogram indeholder netop den søgning, vi før havde stående i selve hovedprogrammet, og vi kan se, at når det er udført, vil den variable FUNDET have værdien 1, hvis koden er fundet, og værdien 0, hvis koden ikke er fundet. Endvidere vil den variable KODESTED indeholde kodens nummer, hvis den står i kartoteket. Når systemet når til linjen med ordet ENDPROC, vil udførelsen af programmet fortsætte med sætningen, som kommer lige efter den linje, hvorfra underprogrammet blev kaldt. Da sporhunden blev kaldt fra linje 310, vil udførelsen altså fortsætte med linje 330, og det er let at se, at resultatet af søgningen udskrives i de følgende linjer.
Under CASE 4, i hvilket en kode ønskes slettet, kaldes sporhunden igen i linje 410, og det tilhørende underprogram udføres. Resultatet af eftersøgningen bruges i linje 420, hvorefter programmet går videre som det oprindelige.
Samtidig med, at vi har indført underprogrammer i eksemplet, har vi også på andre måder ændret programmet, så det bliver mere anvendeligt. Vi har således lagt hele hovedprogrammet ind i en REPEAT..UNTIL-løkke (linje 30 - 510), så operatøren ikke skal starte forfra, hver gang en af operationerne ønskes udført. Sætningen UNTIL 0 virker således,

at løkken bliver udført, indtil operatøren taster ESC. Den primitive ombytning, som findes i det første udkast til programmet, er erstattet af en sortering, som opstiller koderne i alfabetisk orden. Også her benyttes et underprogram: ALFASORT, som kaldes af sætningen i linje 490.
Endelig kan det bemærkes, at hovedprogrammet afsluttes med linje 520, som indeholder ordet

```
END
```

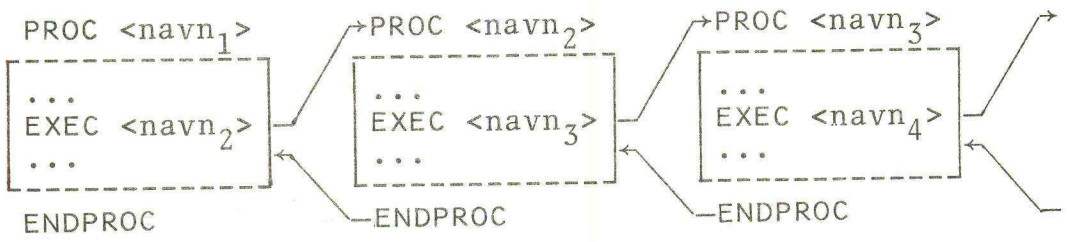
der virker som STOP (jvf. p. 44), men som regel bliver brugt, når det danner afslutningen på et program, og bl.a. tjener til at adskille dette fra underprogrammerne.
□

Ved hjælp af ordene EXEC, PROC og ENDPROC kan man afgrænse og kalde underprogrammer (procedurer) for et program. Virkningen af ordene kan illustreres således:



og kan beskrives således: Når systemet udfører EXEC-sætningen, vil det søge efter en sætning, som begynder med ordet PROC efterfulgt af det navn, som er anført efter ordet EXEC. Hvis det finder en sådan sætning, udfører det den programdel A, som er beskrevet i de følgende linjer. Underprogrammet A efterfølges af en linje med ordet ENDPROC, som bevirker, at udførelsen fortsættes med linjen efter den EXEC-sætning, som har kaldt underprogrammet.

Hvis systemet ikke finder den til EXEC-sætningen hørende PROC-sætning, afgives en fejlmelding. En manglende ENDPROC sætning vil også resultere i en fejlmelding.
Fra ét underprogram kan man kalde et andet underprogram, som igen kan kalde et tredje underprogram, osv. indtil en dybde af 6 underprogrammer. Strukturen af sådanne kald kan illustreres således:



Det skal bemærkes, at en procedure ikke må kalde sig selv, hverken direkte eller indirekte. Hvis fx. <navn$_4$> er lig med <navn$_1$>, vil man let risikere at nedkalde en fejlmelding over udførelsen af programmet.

Eksempel 23.3.
Vi foretager nu en sidste justering af programmet, som styrer varekartoteket (se bilag til nærværende eksempel). For at gøre kørslen så sikker som mulig, indføjer vi nogle ekstra detaljer, som skal bevirke, at programmet får en vis operatørkontrol. Vi forsøger at forudse nogle almindelige operatørfejl, og at få programmet til at afværge disse. Først udnytter vi muligheden af at indsætte en ekstra programdel mellem SELECT-sætningen (linje 50) og den første CASE-sætning (linje 70). Ifølge definitionen af SELECT..CASE (jvf. p. 66), vil denne programdel (linje 60) blive udført, hvis værdien af JOBKODE ikke udpeger noget af de anførte CASEs; men det betyder netop, at operatøren har indtastet en illegal jobkode, hvilket derpå bliver meddelt vedkommende.
Videre kan man forestille sig, at operatøren staver forkert, og dermed vil komme til at indtaste en forvrøvlet varekode. En sådan varekode bør ikke læses ind i kartoteket, og følgelig anbringer vi en analyse af den indtastede

kode i et særligt underprogram, der har navnet: ANALYSE. Underprogrammet kaldes af sætningen EXEC ANALYSE, og af bilaget fremgår, at det tilhørende PROC ANALYSE ganske enkelt undersøger, om koden er opbygget som den skal, altså om den består af tre bogstaver efterfulgt af to cifre. Hvis det er tilfældet, får den prædikatet OK (OK=1). Denne værdi anvendes i hovedprogrammet til at afgøre, om koden kan indlemmes i kartoteket, eller om der skal afgives en fejlmelding (lin. 120 - 160).
Vi benytter dette eksempel til at vise endnu en detalje i COMAL. I linje 850 finder man følgende sætning:

```
IF KODE$(I,I)<"A" OR "Z"<KODE$(I,I) THEN LET OK=0
```

Sætningen, hvis indhold kort kan beskrives således: hvis kodens I'te tegn kommer før "A" eller efter "Z" (altså ikke er noget bogstav), så sæt OK lig med 0 (og angiv dermed, at koden er illegal), består, som man vil se, af to sætninger efter hinanden: en IF-sætning efterfulgt af en LET-sætning. I COMAL er sådanne sammensatte sætninger (compound statements) tilladt, og det kan i mange tilfælde anbefales at anvende dem, idet det kan give et bedre overblik over programmet, og gøre det lettere at læse. Linje 600 indeholder også et eksempel på en sammensat sætning.
□

24. Kommandoen RUN <linjenummer>.

Som nævnt i eksempel 23.2 (p. 68 n.), er det første programudkast besværligt at anvende, idet det skal genstartes efter hver operation. Faktisk forholder det sig værre, end man måske aner, idet programmet er aldeles uanvendeligt, hvis man forsøger at genstarte det ved hjælp af kommandoen RUN (jvf. p. 13). Når kommandoen RUN afgives, vil systemet nulstille alle variable og sætte alle tegnfølger lig med den tomme følge. Hvis operatøren således med megen møje og besvær har indtastet et større udvalg af varekoder under CASE 1, vil det næste RUN, der bliver

afgivet, uden omsvøb eller advarsler slette hele det indlæste kartotek, hvorefter de øvrige programafsnit ikke er til megen nytte. Man har mulighed for at benytte en modificeret RUN-kommando, nemlig RUN <linjenummer>, som vil bevirke, at programmet bliver udført fra og med den linje, hvis nummer står anført i <linjenummer>, og at samtlige variable - numeriske som strenge - bevarer den status, de havde, da programmet standsede sidste gang. Hvis operatøren således skriver RUN 10 i dette tilfælde, vil udførelsen af programmet fortsætte med linje 10, som om intet var hændt, siden udførelsen standsede i linje 470. I bedste fald må dette dog karakteriseres som en besværlig løsning, og erfaringen siger, at man på et eller andet tidspunkt af vanvare kommer til at skrive RUN og dermed spolerer den datamængde, man møjsommeligt har opbygget. En konstruktion, som den, der er vist i næste udkast (bilaget til eksempel 23.2), må være regelen, før et program kan siges at være færdigt. Under opbygning og afprøvning af et program kan det dog være nyttigt at have kommandoen RUN <linjenummer> til sin rådighed. Endnu en advarsel: det kan ikke nytte, at man i det nævnte eksempel starter programmet med RUN 5. I så fald udføres dimensioneringerne i linje 5 med det resultat, at såvel KARTOTEK$ som KODE$ sættes lig med den tomme følge. RUN <linjenummer> skal benyttes med megen omtanke.

25. Pseudo-Boolske variable.

I programmet til eksempel 23.3 finder man i linje 120 sætningen

```
IF OK THEN DO
```

Denne sætning bliver fortolket således af systemet: Hvis den variable OK har en værdi, som er forskellig fra 0, regnes den for et udsagn med værdien *sand*, og hvis OK har værdien 0, regnes den for et udsagn med værdien *falsk*. Herefter skulle det være klart, at sætningen

virker efter sin hensigt (se efter!)
I almindelighed gælder, at en konstant, en variabel eller et udtryk, som optræder på udsagnets plads i hovedsætningen for en styrestruktur (IF-, WHILE- eller UNTIL-sætningen), vil blive fortolket som et *sandt* udsagn, hvis konstanten, den variable eller udtrykket har en værdi, som er forskellig fra 0, og som et *falsk* udsagn, hvis værdien er lig med 0.
En variabel, der anvendes på denne måde, kaldes også en (pseudo-) Boolsk variabel.

Eksempel 25.1.
I programmet til eksempel 23.3 kan man skrive sætningen i linje 420 således:

```
IF FUNDET THEN DO
```

uden at dette vil medføre nogen ændring i programmets virkemåde. Den variable FUNDET bliver derved anvendt som en Boolsk variabel.
□

RESUMÉ.

I dette kapitel har vi hørt om:

Multiforgreninger, underprogrammer (procedurer), pseudo-Boolske variable, sammensatte sætninger, SELECT-sætninger, CASE-sætninger, ENDCASE-sætningen, EXEC-sætninger, PROC-sætninger, ENDPROC-sætningen og END-sætningen samt kommandoen RUN <linjenummer>.

Sætningerne: SELECT, CASE og ENDCASES er beskrevet på side 66, sætningerne EXEC, PROC og ENDPROC er beskrevet på side 69 og END-sætningen er beskrevet på side 69.

Kommandoen RUN <linjenummer> virker som RUN, dog således at udførelsen af programmet begynder med den linje, som er angivet ved <linjenummer>, og kommandoen giver ikke anledning til en ændret status for de variable i programmet.

KAP. VI.
FORMATSTYRING I COMAL.

26. PRINT USING-sætninger.

I PRINT-sætninger har vi mulighed for at udskrive tal og/eller tekster under udførelsen af et program. I det foregående (p. 10 ff. og p. 50 ff.) har vi set, hvorledes man på mange måder kan redigere de tekster, man ønsker udskrevet.

Eksempel 26.1.
Når programmet

```
10 LET PRIS=56.5
20 PRINT "UNDER UDSALGET ER PRISEN KUN";PRIS;"KR"
```

udføres, får man udskriften:

```
UNDER UDSALGET ER PRISEN KUN 56.5 KR
```

□

Som eksemplet antyder, har vi mange muligheder for at bestemme, hvorledes teksten i udskriften skal se ud, men vi har hidtil måttet finde os i, at systemet bestemmer, hvorledes udskriften af talværdier kommer til at se ud. I eksemplet ovenfor vil man næppe finde udskriften af prisangivelsen helt tilfredsstillende. Det sædvanlige i denne sammenhæng må være:

```
UNDER UDSALGET ER PRISEN KUN 56.50 KR
```

Et forsøg på at løse problemet ved at skrive programmet om, så der i linje 10 indtastes sætningen

```
LET PRIS=56.50
```

er dømt til at mislykkes, idet systemet uden videre sletter det "overflødige" nul efter femtallet.
Mange problemer af denne art kan løses ved, at man anvender PRINT USING-sætninger i stedet for PRINT-sætninger. Ordet USING ("idet man bruger" eller "under anvendelse af"), bevirker, at man kan angive helt nøje hvilken form eller hvilket format, de numeriske størrelser i

en udskrift skal have. Det netop omtalte problem kan løses ved at man skriver sætningen

```
PRINT USING "UNDER UDSALGET ER PRISEN KUN ##.## KR",PRIS
```

i stedet for den oprindeligt anførte PRINT-sætning.
Når denne sætning udføres, bliver bogstaverne i følgen udskrevet nøjagtigt som de står i forbilledet (formatet), og på det sted, hvor delfølgen: ##.## er anbragt, vil systemet skrive tallet, som er tildelt den variable PRIS. Hvis PRIS er tildelt værdien 38.3, får vi udskriften:

```
UNDER UDSALGET ER PRISEN KUN 38.30 KR
```

En delfølge af formen: ##.## vil vi også kalde et formatfelt, og man kan tænke på det som en "støbeform", systemet bruger, når det skal udskrive det angivne datum.
Man kan have flere formatfelter i en PRINT USING-sætning; således vil fx. sætningen

```
PRINT USING "VI SAELGER ### DYR FOR #.## PR STK",ANTAL,PRIS
```

bevirke, at den talværdi, der er indlæst i ANTAL udskrives i det første formatfelt, mens den talværdi, der er indlæst i PRIS, udskrives i det andet formatfelt. Tænker vi os at ANTAL=35 og PRIS=8.8, bevirker sætningen, at vi får denne udskrift:

```
VI SAELGER  35 DYR FOR 8.80 PR STK
```

Man vil bemærke, at det første formatmærke i det første formatfelt bliver erstattet af et mellemrum, da det tilsvarende datum kun indeholder to cifre. Derimod bliver eventuelle ledige pladser efter decimalpunktummet fyldt op med nuller. Læg også mærke til, at der ikke er sat noget punktum efter forkortelserne PR og STK i tegnfølgen. Sådanne punktummer kan af systemet forveksles med decimalpunktummer og vil da give anledning til en forkert udskrift.
Som antydet i det foregående findes der en række muligheder for anvendelse af formatfelter i PRINT USING-sætninger. Disse muligheder belyser vi i det følgende eksempel.

Eksempel 26.2.

I det følgende viser vi nogle af de mest anvendte typer for skrivefelter. For hver type er angivet nogle data og de udskrifter, de pågældende data giver i det betragtede formatfelt.

1) #####

datum	udskrift	kommentar
25	25	formatmærker til højre i feltet erstattes af mellemrum.
-30	30	fortegn og andre tegn, som ikke er cifre, bliver overset.
1.92	2	kun hele tal bliver udskrevet; rationale tal bliver forhøjet.
568498	*****	angiver, at det anførte datum er for stort for feltet.

2) ####.##

20	20.00	ledige pladser efter punktum bliver fyldt op med nuller.
29.347	29.35	decimalbrøker forhøjes; hvis heldelen er 0, sættes 0 foran decimalpunktummet.
0.079	0.08	
789012.34	*******	hvis datum har for mange betydende cifre i heldelen, udskrives en række asterikser (som også vist ovenfor).

Eksemplet fortsættes.

3) -##.##

datum	udskrift	kommentar
20.5	20.50	
-20.5	-20.50	fortegnet for tallet udskrives, når tallet er negativt.
-.8	- 0.80	der sættes mellemrum mellem fortegn og det først skrevne ciffer.

4) +###.##

18.3	+ 18.30	der sættes altid fortegn.
.4	+ 0.40	

5) --####

103	0103	der udskrives foranstillede nuller til at udfylde formatet.
-5	-0005	

6) --##.##

2	02.00
-2	-02.00

7) ##.##↑↑↑↑

170.35	17.03E+01	tallet angives på flydende form. Der skal angives fire opadpile i formatfeltet, hvis denne udskriftform ønskes.
175678	17.56E+04	
.175678	17.56E-02	

Eksemplet fortsættes.

8) -###.##↑↑↑↑

datum	udskrift	kommentar
.8	800.00E-03	
-90567893	-905.67E+05	

□

I det netop afsluttede eksempel er vist, hvorledes de mest anvendte formatfelter virker. Herudover findes nogle mere specielle, som kun sjældent anvendes. En beskrivelse af disse kan læseren finde i EXTENDED BASIC USER'S MANUAL (093-000065-04) p. 3-31 ff.

Hvis man i samme linje skal skrive flere variable i samme format, kan ét formatfelt benyttes flere gange. Fænomenet lader sig lettest forstå gennem et par eksempler.

Eksempel 26.3.
En vektor OBS indeholder fire komponenter: 5.678, 3.3, 5.98, 4.3456. Vektoren ønskes udskrevet i formatet: OBS(0)=5.68. Udskriften kan styres af følgende sætning:

```
PRINT USING "OBS(#)=#.## ",0,OBS(0),1,OBS(1),2,OBS(2),3,OBS(3)
```

De to formatfelter bliver først benyttet til udskrift af 0 og indholdet af OBS(0), dernæst til udskrift af 1 og indholdet af OBS(1) osv. til linjen er skrevet færdig. Resultatet af udskriften bliver (jvf. reglerne ovenfor):

```
OBS(0)=5.68  OBS(1)=3.30  OBS(2)=5.98  OBS(3)=4.35
```

Man kan frembringe den samme udskrift ved at bruge følgende program:

```
100 FOR I=0 TO 3
110   PRINT USING "OBS(#)=#.##  ",I,OBS(I);
120 NEXT I
```

idet semikolon i slutningen af en PRINT USING-sætning har samme virkning som i slutningen af en PRINT-sætning (jvf. side 13).

□

27. Formatstyring med strengvariable.

Et skriveformat kan indlæses i en strengvariabel, som derefter kan indtage formatets plads i en PRINT USING-sætning (jvf. brugen af strengvariable i PRINT-sætninger p. 50 ff.).

Eksempel 27.1.
I et program skal vi flere steder bruge en udskrift af formen:

```
SALGSPRIS INCL MOMS ER: 107.75 KR
```

Dette skriveformat læser vi ind i en strengvariabel i begyndelsen af programmet:

```
10 DIM FORMAT$(80)
20 LET FORMAT$="SALGSPRIS INCL MOMS ER: ####.## KR"
```

Hver gang vi har brug for den nævnte udskrift, benytter vi sætningen

```
PRINT USING FORMAT$,SALGSPR
```

hvor den variable SALGSPR indeholder det aktuelle beløb. Når den sidstnævnte sætning udføres, benytter systemet strengen i FORMAT$ som skriveformat, og vi får den ønskede opstilling af udskriften.
□

Vi vil afslutte gennemgangen med et par mere omfattende eksempler.

Eksempel 27.2.
Til dette eksempel hører et bilag, som findes bagest i heftet. Programmet, som er vist på bilaget, kan indfange et observationssæt og lagre dette i en matriks, som kaldes OBS. Derpå udregner programmet summen af hver række (SUMREKKE) og af hver søjle (SUMSØJLE) samt totalsummen af alle observationerne (SUM). Til slut udskrives en tabel med de nævnte størrelser. Vi vil især interessere os for udskrifterne i programmet. I linje 20 og 40 indføres en strengvariabel TABEL$,

som tildeles tegnfølgen: " ###.## " (bemærk mellemrummene). Denne tegnfølge bruges som skriveformat i linje 240 og i linje 300. I begge tilfælde anvendes skriveformatet flere gange i samme linje (semikolon!). I linje 70 finder man et eksempel på en operatørvejledning. Linjerne 70 og 80 vil bevirke, at systemet udskriver en tekst af formen:

OBS(3; 5):

og derpå går i venteposition på grund af INPUT-sætningen. Det kan bemærkes, at man ikke må sætte komma mellem de to formatfelter i parentesen (se næste eksempel). Der er altså tale om en dynamisk vejledning, idet udskriften ændrer sig under indtastningen, så operatøren hele tiden ved, hvor langt indtastningen er fremme. Sætningerne i linje 260 og linje 320 overlades til den årvågne læsers selvstudium.
Linjerne 180 til 200 viser en lille, men nyttig teknisk detalje. Når man er færdig med indtastningen (som naturligvis sker fra terminalen), ønsker man måske at få udskriften på linjeskriveren. I så fald skal man blot taste RUNL 220 (210+10). Grunden til, at det linjenummer, ved hvilket systemet skal genstarte udførelsen af programmet, er angivet ved <stoplinje+10>, er naturligvis, at man ved eventuelle senere rettelser i programmet ikke kan regne med, at udskriften altid starter i linje 220. Systemet vil dog altid skrive meldingen

STOP AT nn

hvor nn er nummeret på den linje, der indeholder STOP-ordren, og operatøren kan da regne ud, at programmet skal startes i linjen med nummeret nn+10 (programeditering afsluttes naturligvis med et RENUMBER).
□

Eksempel 27.3.
Programmet, som er bilagt dette eksempel, viser hvorledes man kan styre udskriften af et mere omfattende materiale.

Programmet virker som underprogram for et større program, der indfanger data og beregner de i underprogrammet benyttede uddata. Ved som vist at arbejde med en række sideordnede strengvariable, som hver for sig indeholder skriveformatet til en linje, kan man allerede under indtastningen af programmet få et godt overblik over det færdige resultat. Man spares derved for at tælle positioner, mellemrum mv.
Det viste program indeholder i øvrigt kun én ny detalje. I linje 5050 står der bl.a.

```
... LØNPERIODE ##,##,## - ##,##,##
```

Det viste formatfelt bevirker, som man vil se af den viste udskriftprøve, at det tal, der udlæses i feltet, splittes op af kommaer, hvilket i dette tilfælde er meget hensigtsmæssigt. Kommaet har altså i denne sammenhæng en helt speciel betydning og må derfor bl.a. ikke anvendes i sætninger som den i forrige eksempel viste (linje 70):

```
PRINT USING "OBS(##;##):  ",RK,NR;
```

Hvis man i en sådan sætning skriver: OBS(##,##), vil systemet opfatte formatfeltet i parentesen som skrivefelt for ét tal, der blot skal opsplittes under udskriften.
□

28. TAB-funktionen.

I COMAL II findes endnu en mulighed for styring af udskrifter, idet man kan benytte den indbyggede funktion: TAB, hvis format er følgende:

```
TAB(<udtr>)
```

hvor <udtr> er en konstant, en variabel eller et aritmetisk udtryk. TAB-funktionen anvendes i PRINT-sætninger og sætter skrivepositionen for det første tegn, som følger efter TAB(<udtr>), medmindre denne skriveposition er blevet overhalet af eventuelle forudgående udskrifter. Skrivepositionen udregnes som heldelen af vær-

dien af <udtr>. TAB(<udtr>) skal altid efterfølges af tegnet semikolon (;).

Eksempel 28.1.

Følgende program

```
10 DATA 5,-7,9,-11
20 READ TAL1,TAL2,TAL3,TAL4
30 PRINT TAB(5);TAL1;TAB(10);TAL2;TAB(15);TAL3;TAB(20);TAL4
```

giver os følgende udskrift, som vi analyserer ved at anbringe tabulatormærker over den:

```
0    5   10   11   15   20   ←position
     ↓   ↓    ↓    ↓    ↓
     5   -7   9    -11       ←udskrift
```

□

Eksempel 28.2.

Vi forsøger at omskrive programmet fra eksempel 28.1:

```
10 DIM TAL(4)
20 DATA 5,-7,9,-11
30 FOR I=0 TO 3
40   READ TAL(I)
50   PRINT TAB(5);TAL(I);
60 NEXT I
```

Dette program giver imidlertid følgende udskrift:

```
0    5    10   15   20        ←position
     ↓ ↓  ↓ ↓
     5 -7 9 -11              ←udskrift
```

TAB-funktionen virker altså kun på udskriften af det første tal; derefter virker semikolon på sædvanlig måde (jvf. p. 11 ff.). Hvis vi ændrer sætningen i linje 50 til

```
PRINT TAB(5*(I+1));TAL(I);
```

får vi samme udskriftformat som i eksempel 28.1, idet argumentet for TAB-funktionen denne gang opdateres ved hvert gennemløb af løkken og således ikke overhales af den forudgående udskrift.

□

Eksempel 28.3.

Man kan ved hjælp af TAB-funktionen få systemet til at tegne forskellige figurer. Således vil programmet

```
10 FOR LINJE=1 TO 9
20   IF LINJE<4 OR 6>LINJE THEN DO
30     PRINT TAB(5);"**********"
40   ELSE
50     PRINT TAB(5);"***";TAB(12);"***"
60   ENDIF
70 NEXT LINJE
```

give følgende udskrift:

```
**********
**********
**********
***    ***
***    ***
***    ***
**********
**********
**********
```

□

Eksempel 28.4.

Ved at anvende TAB-funktionen kan man fremstille skitser af grafer for forelagte funktioner, når blot disse funktioner er tilstrækkeligt simple. På bilaget til dette eksempel er vist et program, som "tegner" en skitse af grafen for funktionen

$$f(x) = 12x^3 + 24x^2 + 12.$$

Som man vil se, giver tegningen et udmærket indtryk af grafens forløb i hovedtræk. På tegningen er antydet en abcisseakse, og med fornøden teknisk indsats og tålmodighed kan man også få tegnet en ordinatakse.

□

RESUMÉ.

I dette kapitel har vi hørt om:

Formatstyring, formatfelter, formatmærker, PRINT USING-sætninger, TAB-funktionen og kommandoen LRUN.

En PRINT USING-sætning er opbygget således:

PRINT USING <skriveformat>,<liste af udtryk>

hvor <skriveformat> er en tegnfølge, som enten kan anføres direkte eller ved hjælp af en strengvariabel. Skriveformatet indeholder formatfelter, som efter nærmere angivne regler styrer udskriften af de talværdier, der er tildelt udtrykkene i <liste af udtryk>.
I en PRINT USING-sætning er de sædvanlige konventioner for PRINT-sætninger (TAB-funktionen, kommaer og semikolonner) ophævet. Et komma eller et semikolon, som afslutter en PRINT USING-sætning, har dog samme virkning som i en sædvanlig PRINT-sætning (jvf. p. 11 ff.).

TAB-funktionen er defineret på mængden af aritmetiske udtryk i COMAL II. Virkningen er funktionen er en tabulering til den position, der udregnes ved afrunding af værdien af udtrykket, som er argument for TAB. Funktionen optræder altid i PRINT-sætninger og skal efterfølges af tegnet semikolon.

Hvis man råder over en linjeskriver i sin konfiguration, kan man benytte kommandoen LRUN, der virker som RUN, bortset fra, at alle PRINT-sætninger bliver udført på linjeskriver i stedet for på terminalen.

På tilsvarende måde anvendes kommandoen CONL, der virker som CON (side 45), bortset fra, at alle PRINT sætninger bliver udført på linjeskriveren, når CONL er afgivet.

KAP. VII.
RND-FUNKTIONEN. SIMULATORER.

29. RND-funktionen.

Datamater benyttes ofte til at efterligne eller simulere processer, hvis forløb på et eller flere punkter bestemmes af tilfældigheder. En forudsætning for, at dette kan lade sig gøre, er naturligvis, at processen kan gives én eller anden talmæssig iklædning (numerisk repræsentation). I det system, under hvilket COMAL-fortolkeren virker, findes indbygget en funktion til sådanne formål. Funktionen, som betegnes RND (forkortelse for random: tilfældig), antager (pseudo-) tilfældigt værdier i det åbne interval fra 0 til 1. Funktionen har ikke nogen egentlig definitionsmængde, men skal dog af formelle grunde udstyres med et argument, fx. tallet 0.

Eksempel 29.1.
Programmet herunder

```
10 FOR NR=1 TO 12
20   PRINT RND(0);
30 NEXT NR
```

gav følgende udskrift

```
 .21132          .14464          .852625         .927054
 .162866         .433095         .563933         .20965
 .727678        5.35766E-2       .576015        1.17701E-2
```

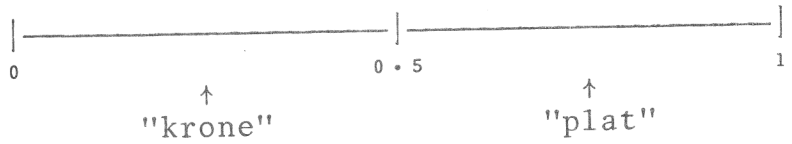
□

Vi skal dernæst se, hvorledes man kan bruge RND-funktionen til at efterligne et simpelt spil.

Eksempel 29.2.
Vi vil skrive et program, som kan simulere kast med en mønt, og vi ønsker at tælle op, hvor mange gange vi får udfaldet "krone" ved en serie af sådanne kast. Programmet kan fx. se således ud:

```
10 INPUT "HVOR MANGE KAST ØNSKES UDFØRT? ",ANTAL
20 ANTALKR=0
30 FOR KAST=1 TO ANTAL
40   IF RND(0)<.5 THEN ANTALKR=ANTALKR+1
50 NEXT KAST
60 PRINT ANTALKR
```

Vi forudsætter som sædvanlig, at sandsynligheden for "plat" og "krone" er den samme, nemlig 0.5, og vi fortolker derfor et tilfældigt tal (RND(0)), som er mindre end 0.5, som udfaldet "krone", mens et tilfældigt tal, som er større end eller lig med 0.5, fortolkes som udfaldet "plat".



□

Den metode, som er anvendt i eksempel 29.2, vil vi kalde intervalmetoden, og vi skal senere se flere eksempler på denne. I det næste eksempel vil vi vise omskrivningsmetoden:

Eksempel 29.3.
Vi vil nu skrive et program, som kan efterligne kast med en terning. Udfaldet af et terningkast kan som bekendt beskrives ved et af tallene 1, 2, 3, 4, 5 og 6, og ved at benytte intervalmetoden, kan vi fortolke et tilfældigt tal i intervallet fra 0 til $\frac{1}{6}$ som udfaldet 1, et tilfældigt tal i intervallet fra $\frac{1}{6}$ til $\frac{2}{6}$ som udfaldet 2, osv. Vi kan imidlertid også foretage en omskrivning (transformation) af RND-funktionen, som gør det muligt for os at få det frembragte øjental angivet direkte, og det vil vi benytte i dette tilfælde. Vi sætter

```
UDFALD=INT(6*RND(0)+1)
```

og har

$$0 < \text{RND}(0) \leq \tfrac{1}{6} \Leftrightarrow 0 < 6*\text{RND}(0) \leq 1 \Leftrightarrow$$

$$1 < 6*\text{RND}(0)+1 \leq 2 \Leftrightarrow \text{INT}(6*\text{RND}(0)+1) = 1$$

og

$\frac{1}{6} < RND(0) \leq \frac{2}{6} \Leftrightarrow 1 < 6*RND(0) \leq 2 \Leftrightarrow$

$2 < 6*RND(0)+1 \leq 3 \Leftrightarrow INT(6*RND(0)+1) = 2$

osv.

Den variable UDFALD vil altså tilfældigt antage en af værdierne 1, 2, 3, 4, 5 eller 6, og vi får netop efterlignet et kast med en terning.
Programmet kan se således ud:

```
10 INPUT "HVOR MANGE KAST ØNSKES? ",ANTAL
20 SEKSERE=0
30 FOR KAST=1 TO ANTAL
40   UDFALD=INT(6*RND(0)+1)
50   IF UDFALD=6 THEN SEKSERE=SEKSERE+1
60 NEXT KAST
70 PRINT "DET BLEV TIL";SEKSERE;"SEKSERE I";ANTAL;"KAST."
```

□

30. Simulering af systemer.

I de senere år er man i stigende grad begyndt at anvende datamater til simulering af indviklede systemer i samfundet. Sådanne store simulatorer kan ofte give værdifulde oplysninger om, hvorledes et planlagt system vil virke i givne situationer, og disse oplysninger kan måske give anledning til, at man ændrer planerne for systemet og derved undgår store fejlinvesteringer i den fysiske opbygning af det. COMAL II er ikke særligt velegnet til at beskrive sådanne store systemer; dertil har sproget for få faciliteter; men man kan skrive programmer i COMAL II til efterligning af mindre systemer og på den måde give et indtryk af, hvorledes de mere omfattende simulatorer virker, og hvilke principper der ligger bag opbygningen af dem.
I det følgende eksempel vil vi beskrive et system, som er tilpas simpelt til, at vi kan lade et COMAL-program simulere det, men som dog er stort nok til i givet fald at volde planlæggere hovedbrud.

Eksempel 30.1.
Til eksemplet hører et bilag, som vi referer til under

gennemgangen (B9 - B12).
Det system, vi vil beskrive, udfolder sig omkring tre kasser i et supermarked. I beskrivelsens første fase deler vi systemet op i følgende delsystemer: Kasse nr. 1, kasse nr. 2 og kasse nr. 3. Til hver af disse kasser hører en kø af kunder og en ekspedient. Endvidere betragter vi det omgivende supermarked som et delsystem, hvis eneste opgave i vor beskrivelse er at producere nye kunder til køerne ved kasseapparaterne (synspunktet er formentlig også relevant fra et rent erhvervsmæssigt synspunkt).
En kasse kan befinde sig i tre tilstande: den kan være åben eller lukket, og den kan befinde sig i en slags mellemtilstand, som vi beskriver ved ordet: lukkes, og som indtræder, når ekspedienten står i begreb med at lukke kassen og derfor lukker for tilgangen af nye kunder til køen, men dog færdigekspederer de kunder, der er kommet ind i køen på dette tidspunkt.
Lad os derpå beskrive de muligheder, ekspedienterne har i nærværende system. Når en kasse er åben, kan en ekspedient være i færd med at betjene en kunde eller - hvis der ikke er nogen kunder i køen - være ved at udføre andet arbejde (sætte tomme returflasker i kasser, åbne cigaretkartoner, fylde veksleautomaten op, osv.). Endvidere kan ekspedienten, som nævnt ovenfor, sætte kassen i tilstanden lukkes, og - når køen er afviklet - sætte kassen i tilstanden lukket. Dette handlingsforløb bliver aktuelt, når ekspedienten vil holde frokostpause. Denne pauses længde er fastlagt på forhånd ved overenskomst mellem supermarkedet og ekspedienterne, men begynder naturligvis på forskellige tidspunkter ved de forskellige kasser. Endvidere er det en aftale, at en kasse ikke bliver lukket, før en eventuelt eksisterende kø er afviklet (jvf. beskrivelsen af kassernes tilstande). Tidspunktet for frokostpausens begyndelse er altså ikke helt fast, og derfor må pausens sluttidspunkt udregnes i hvert enkelt tilfælde ud fra det aktuelle begyndel-

tidspunkt og pausens varighed.
Da ikke alle ekspedienter er lige hurtige, er der for hver enkelt ekspedient givet en betjeningsfaktor, som bruges ved udregningen af betjeningstiden (se nedenfor). En kø opdeles i kunder, og en kunde karakteriseres ved det antal vareenheder, vedkommende bringer med sig ind i systemet. På grundlag af denne varemængde og ekspedientens betjeningsfaktor udregnes den tid (betjeningstiden eller kundetiden), det vil tage at ekspedere kunden. Som nævnt ovenfor produceres nye kunder af det omgivende system, og en eventuel ny kunde vil inspicere køerne for at finde ud af, hvilken af dem, der må formodes at blive afviklet hurtigst. Kundens vurdering af en kø bestemmes ikke blot af, hvormange der i forvejen står i køen, men først og fremmest af den varemængde disse bringer med sig. En kunde formodes således at ville foretrække en kø med fem kunder der tilsammen har få varer, frem for en kø med to kunder, der har fuldt læssede kurve. En ny kunde vil naturligvis ikke stille sig op ved en kasse, der er lukket, eller som står i tilstanden lukkes.

Programmet, som skal simulere dette system, er delt op i et hovedprogram (10 - 300) og nogle underprogrammer. I hovedprogrammet indlæses parametrene for simulationen: LUKKETID, PAUSETID, BEGPAUSE(KASSE), BETJFAK(KASSE) og den tidsperiode med hvilken der skal aflægges rapport (PER). De enkelte variables betydning fremgår klart af teksten til programmet.
Derpå åbnes kasserne, og simulatoren startes (210).
Problematikken ved denne simulation er af en særlig art, idet der er tale om flere processer, som i virkeligheden foregår samtidig. Der ekspederes - eller der kan i det mindste ekspederes - ved de tre køer samtidig, og mens der ekspederes, ankommer nye kunder fra butikkens indre og tager plads i en af køerne. Vi har altså at gøre med en række parallelle processer, som skal simuleres i et system, der kun kan arbejde lineært (sekventielt), altså

ved at udføre de enkelte trin i processen efter hinanden i en ganske bestemt rækkefølge. Heldigvis har vi med tænkte objekter at gøre, og vi udnytter disses uendelige tålmodighed til at efterligne systemet på følgende måde: Forløbet deles op i adskilte delforløb, som hver tænkes at vare en tidsenhed. I hvert delforløb foretager vi os følgende: Vi inspicerer tilstandende ved den første kasse og foretager de justeringer, som er nødvendige for spillets forløb. Det samme gør vi derefter ved den anden og den tredje kasse. Når én kasse med kø og ekspedient er inspiceret, stiller vi den altså simpelthen i bero og går videre med næste delsystem. Når vi er færdige med at inspicere og justere kasserne, ser vi efter, om det er aktuelt med en ny kunde og placerer i givet fald denne i en kø efter de regler, som er nævnt ovenfor. Derpå stiller vi uret én tidsenhed frem og gennemfører en ny runde til de forskellige delsystemer. Således bliver vi ved, indtil det er lukketid. Når butikken er lukket, vil køerne naturligvis ret hurtigt tømmes, så kasserne kan lukkes, men denne del af processen interesserer os ikke i denne sammenhæng.
Den således beskrevne fremgangsmåde er programmeret i linjerne 210 - 280.

Vi ser dernæst på underprogrammerne. Den første hedder KASSE (linje 310 - 530) og beskriver tilstande og aktiviteter i forbindelse med den enkelte kasse. Den næste har titlen EKSPEDER og beskriver en ekspedition og den eventuelt påfølgende oprykning af køen. I underprogrammet NYKUNDE anvender vi RND-funktionen til at afgøre, om der skal komme en ny kunde ind fra supermarkedet eller ej. Som man vil se, regner vi med, at sandsynligheden herfor er 0.13, og vi anvender intervalmetoden ved fortolkningen af funktionsværdien (linje 700). I linje 720 bruger vi atter RND-funktionen; denne gang til at afgøre, hvor mange varer, kunden medbringer. Ved at anvende omskrivningsmetoden, får vi - tilfældigt - antallet af varer sat til et helt tal i det lukkede interval mellem

1 og 20. I underprogrammet KASSEVLG afgøres det, i hvilken kø en eventuel ny kunde vil stille sig, når man forudsætter, at vedkommende vil søge at nå frem til en kasse så hurtigt som muligt. Underprogrammet RAPPORT udskriver meldinger om tilstandene i simulatoren med de tidsintervaller, vi har fastlagt i hovedprogrammet (ved den variable PER).

Vi giver til slut en mere detaljeret gennemgang af underprogrammerne og den måde, på hvilken tilstande og aktiviteter er repræsenteret i dem. Først giver vi en uformel beskrivelse af algoritmen. Denne beskrivelse danner en overgangsform mellem den sproglige beskrivelse af systemet, vi gav ovenfor, og det helt formelle program, som bruges ved udførelsen af processerne. Fremgangsmåden er iøvrigt blevet anvendt ved opstillingen af programmet, og det kan anbefales at benytte den, når man skal skrive programmer, som omfatter mere end nogle få sætninger.

```
proc kasse
hvis kassen betjenes så
  hvis kunde ved kassen så
    ekspedér kunden
  ellers  //altså ingen kunde ved kassen//
    udfør andet arbejde
  slut
ellers  //ekspedienten er ved andet arbejde//
  hvis kunde venter så gør klar til at betjene kunden
slut
hvis frokosttid eller senere og eksp. ikke har spist så
  klar til at lukke
  hvis ingen kunder så
    luk kassen; se på klokken og regn ud, hvornår pausen er slut
    markér spisningen
  slut
slut
hvis kassen er lukket så
  hvis senere end slutpause så luk kassen op
slut
slutproc kasse
```

Tilstanden: *kassen betjenes* repræsenteres i programmet ved den Boolske vektor BETJ, som har tre komponenter, én for hver kasse. Hvis fx. KASSE er lig med 2 og BETJ(KASSE) har værdien 1, betyder det, at kasse 2 bliver betjent. Hvis ekspedienten ved en given kasse er beskæftiget med andet arbejde, har BETJ(KASSE) værdien 0 (linje 320 og linje 360).
Tilstanden: *kunde ved kassen* repræsenteres ved matricen KUNDE. I denne matrix er indlæses oplysningerne om de kunder, som står i køerne. Således vil KUNDE(2,5) indeholde det tal, som angiver den varemængde kunde nr. 5 i køen ved kasse nr. 2 bringer med sig. En kø vil altså være tom, hvis KUNDE(KASSE,1) er lig med nul (der er ikke nogen varemængde på første plads i køen ved KASSE).
Kassens forskellige tilstande: *kassen er åben*, *kassen er lukket* og *klar til at lukke* indeholdes i en tilstandsvektor STATUS, som har tre komponenter, én for hver kasse. Hver enkelt komponent i STATUS kan antage tre værdier: AABEN (=1), LUKKET (=0) og LUKKES (=-1). Den verbale udtryksform i programmet er på dette punkt så klar, at yderligere forklaringer turde være overflødige.
Oplysninger om, hvorvidt ekspedienten har spist eller ej, er lagret i den Boolske vektor SPIST, som ligeledes har tre komponenter, én for hver kasse.

Vi vil derpå beskrive underprogrammet EKSPEDER.

proc ekspedér
udfør ekspeditionen
hvis kunden gøres færdig så
  lad kunden passere tælleren; læg hans indkøb til dagens øvrige; registrér at kunden forlader køen;
  ryk køen op og se efter hvormange varer evt. næste kunde har med sig
slut
slutproc ekspedér

Aktiviteten: *udfør ekspeditionen* repræsenteres ved tildelingen: KUNDETID(KASSE)=KUNDETID(KASSE)-1, idet vektoren

KUNDETID til ethvert tidspunkt indeholder oplysninger om, hvor langt ekspeditionen af den forreste kunde i hver enkelt kø er skredet frem. For en given kasse, udregnes kundetiden på det tidspunkt, kunden træder ind på køens forreste plads. Denne beregning sker i linje 640 (ved oprykning af køen) eller i linje 750 (nemlig når en ny kunde går direkte ind på køens forreste plads). Oprykningen af køen er beskrevet i linjerne 600 - 620 og man vil bemærke vektoren MAXNR, som til ethvert tidspunkt indeholder nummeret på den sidste kunde i den enkelte kø. Hvis MAXNR(KASSE) for en given kasse har værdien 0, betyder det altså, at køen er tom.

Underprogrammet NYKUNDE er opbygget således:

```
proc nykunde
hvis nykunde fra supermarkedet så
  den nye kunde passerer en tæller;
  og bringer en vis varemængde med sig;
  den nye kunde vælger kø, og
  hvis den valgte kø er tom så
    ekspeditionen af den nye kunde begynder straks
  slut
  kunden bliver under alle omstændigheder registreret
  som foreløbig sidste i køen; og den varemængde han
  bringer med sig lægges til de øvrige i køen
slut
slutproc nykunde
```

Som før nævnt lader vi RND-funktionen være afgørende for, om en ny kunde dukker op eller ej (linje 690) og den varemængde, kunden i givet fald bringer med sig, bestemmes ligeledes ved hjælp af RND-funktionen (linje 710).
I programmet optræder atter vektoren SUMVARER, hvis komponent for den valgte kasse nu øges med det antal varer, den nye kunde bringer med sig ind i køen.
De øvrige repræsentationer i programmet er nævnt i det foregående. Vi mangler nu blot at beskrive underprogrammet KASSEVLG, som udfører processen: *den nye kunde vælger kø.*

```
proc kassevlg
kunden sorterer køerne ved de enkelte kasser efter den
samlede varemængde i disse køer;
kassen, ved hvilken den mindste varemængde findes, kaldes
kasse1;
kassen, ved hvilken den næstmindste varemængde findes,
kaldes kasse2;
og kassen, ved hvilken den største varemængde findes,
kaldes kasse3.
hvis kasse1 er åben så
  kunden vælger køen ved kasse1
ellers  //kasse1 er altså ikke åben//
  hvis kasse2 er åben//
    kunden vælger køen ved kasse2
  ellers  //kasse2 er heller ikke åben//
    kunden vælger køen ved kasse3
  slut
slut
slutproc kassevlg
```

Den nævnte sortering af kasserne efter varemængde er programmeret i linjerne 840 - 940. Derefter følger programmet fuldstændig ovenstående beskrivelse, idet aktiviteten: *kunden vælger køen ved kassex* udføres ved tildelingen KASSE=KASSEX, hvor X er 1, 2 eller 3.

Underprogrammet RAPPORT er meget simpelt, og der indgår ikke nye repræsentationer i det. Bemærk dog brugen af variabelnavne efter CASE.

På side B11 og B12 er vist udskrifter af aktuelle parametre for en kørsel, og dele af de udskrevne rapporter er også vist. Rapporterne tyder på, at frokostpauserne ikke er alt for hensigtsmæssigt placeret i forhold til hinanden. Endvidere set det, at det næppe er klogt at anbringe en forholdvis langsom ekspedient (betj.fak.=1.7) ved kasse 1 (10 ventende kunder kl. 880).

Simulatoren er behæftet med en række åbenlyse mangler. Således vil tilgangen af kunder ikke ske med samme sand-

sandsynlighed hele dagen. RND-funktionen i linje 690 bør givetvis modificeres med en eller anden funktion, som kan gengive den rytme, der normalt findes i tilgangen af kunder til en butik. Endvidere bør den tilstand i hvilken kasserne befinder sig (åben, lukket eller lukkes) kunne styres af andre faktorer end frokostpausen, bl.a. af tilstrømningen af kunder.
Der venter således en række opgaver på læseren, og disse opgaver er ikke af triviel art.
□

RESUMÉ.

I dette kapitel har vi hørt om:

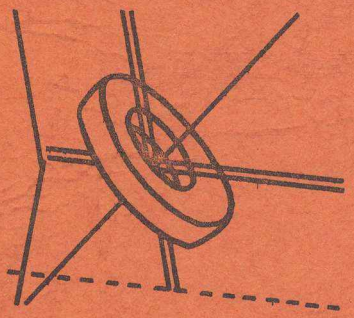
RND-funktionen, spil og simulering af systemer.

RND-funktionens værdimængde er (pseudo-) tilfældige tal i det åbne interval fra 0 til 1. RND-funktionen har ikke nogen egentlig definitionsmængde, men skal af formelle grunde forsynes med et argument. Som argument anvendes ofte tallet 0 eller 1.

INDEX.

-oOo-

another implementation
from:



DATO, *Tønder.*