MANUAL

C O M A L - 8 0 p å C O M E T - c o m p u t e r e n.

En lille håndbog for den, der ligesom jeg ikke kan huske det hele.

Jens Peder Pedersen, Brundlundskolen, Aabenraa.

Opstart af COMET med diskette og skærm:

1. Tænd skærmen (øverste knap) › rødt lys.
2. Tænd computeren (rød knap POWER) › rødt lys.
3. Tryk på RESET-knappen på computeren › tekst på skærm.
4. Sæt en diskette indeholdende CP/M og COMAL-80 i drive A.
5. Tast 1 (et ettal) › der skrives A› på skærmen.
6. Tast COMAL-80 og tryk derefter på RETURN-knappen på tastaturet. › der stilles et spørgsmål.
7. Svar på spørgsmålet.
8. Skru lysstyrken så langt ned som muligt ved at dreje på BRIGHT-knappen på skærmen.
9. Indstil CONT-knappen på skærmen, således, at teksten kun lige kan anes.
10. Indstil nu på BRIGHT-knappen til den ønskede lysstyrke er opnået. › Held og lykke.

Denne manual er skrevet af Jens P. Pedersen, Brundlundskolen, Aabenraa ved hjælp af et tekstbehandlingsprogram, der på Brundlundskolen er udviklet til brug på COMET MPS-3000 med en diskettestation.

I håbet om at den kan være til gavn.

Jens P. Pedersen.

DIREKTE KOMMANDOER, SOM UDFØRES ØJEBLIKKELIGT.

A) ... uden aktivering af diskette:

AUTO efterfulgt af tryk på RETURN. Bruges under orogrammering og bevirker at datamaten automatisk skriver et nyt linienummer, når der trykkes på RETURN.
Kommandoen kan bruges på tre måder:
1) Når der blot skrives AUTO, begyndes med linienummeret 0010 og de efterfølgende numre ooskrives automatisk med 10.
2) Når der skrives AUTO t, hvor t er et naturligt tal under 10000, begyndes der med linienummeret t og de efterfølgende numre ooskrives automatisk med 10.
3) Når der skrives AUTO t,T hvor både t og T er naturlige tal, begyndes der med linienummeret t og de efterfølgende numre ooskrives automatisk med T.

CLEAR efterfulgt af tryk på RETURN giver ren skærm med "blinkeren" placeret i øverste venstre hjørne.

DEL bruges til sletning af orogramlinier. Kommandoen kan bruges på fire måder:
1) Når der skrives DEL t hvor t er et eksisterende linienummer, slettes den oågældende programlinie.
2) Når der skrives DEL t, slettes alle programlinier med numre fra t og oefter.
3) Når der skrives DEL ,t slettes alle programlinier med numre fra 1 oo til t.
4) Når der skrives DEL t,T slettes alle programlinier med numre fra t oo til T.

EDIT Bruges, når man vil rette i nogle orogramlinier. Når kommandoen er anvendt skrives de(n) udoegede orogramlinie(r) på skærmen og "blinkeren" er da placeret i linien, således at man v.hj.a. oiletasterne kan bevæge sig frem og tilbage i linien og udføre rettelser i den.
Efter at rettelserne er udført trykkes på RETURN. Trykker man i stedet på ESC, stoppes retteproceduren og datamaten "glemmer" ca de ændringer som man evt. har foretaget i den sidste linie, man var i.
Kommandoen bruges på tre måder:
1) EDIT t hvor t er et eksisterende linienummer, kaldes kun denne linie frem til retning.
2) EDIT t, kalder alle linier fra t og oefter frem til retning, dog kun en ad gangen. Man kommer videre fra den ene linie til den anden ved at trykke på RETURN.
3) EDIT uden talangivelse kalder hele orogrammet frem til retnign, således at der begyndes med det laveste linienummer.

ESC et tryk på denne KNAP vil som regel stoppe computeren (bruges til at "flygte" ud af en uønsket situation).

LIST bruges, når man ønsker at få en udskrift af (dele af) et program. Kommandoen kan bruges på flere måder:

1) Når der blot skrives LIST fås en udskrift af hele programmet (på skærmen).
2) Når der skrives LIST t hvor t er et linienummer, fås kun en udskrift af den angivne linie (på skærmen).
3) Når der skrives LIST t, hvor t er et linienummer, fås en udskrift af programmet fra den angivne linie og fremefter (på skærmen).
4) Når der skrives LIST ,t hvor t er et linienummer, fås en udskrift af programmet indtil og med den angivne linie (på skærmen).
5) Når der skrives LIST t,T hvor t og T er to linienumre, fås en udskrift af programmet fra og med linie t til og med linie T (på skærmen).

Bemærk at LIST-kommandoen også bruges i forbindelse med skrivning på diskette (se side 3).

LIST LP: I alle fem ovennævnte tilfælde under LIST kan man efter kommandoen (efter linienummer-angivelsen) tilføje LP: hvilket bevirker at udskriften udføres på printeren i stedet for på skærmen.

NEW efterfulgt af tryk på RETURN sletter det program, der i øjeblikket er gemt i arbejdslageret.

PRINT efterfulgt af et variabelnavn eller et "regneudtryk" og afsluttet med tryk på RETURN bevirker at computeren på skærmen skriver variablens aktuelle værdi eller regneudtrykkets resultat, såfremt de nødvendige oplysninger er gemt i arbejdslageret.
Om brugen af PRINT i programmer - se side 8 og 9.

RENUM bruges til at om-nummerere programmets linier. Kommandoen kan bruges på følgende måder:

1) Når der blot skrives RENUM omnummereres alle programlinier, således at første linie får nummeret 0010 og afstanden mellem numrene bliver 10.
2) Når der skrives RENUM t hvor t er et naturligt tal, får første linie nummeret t og nummerafstanden mellem de efterfølgende linier bliver 10.
3) Når der skrives RENUM t,T får første programlinie nummeret t og afstanden mellem linienumrene bliver T.
4) Når der skrives RENUM s:S,t,T hvor s og S er nogle allerede eksisterende linienumre (s < S) vil de programlinier, der har numre fra og med s til og med S få tildelt nye numre, således at den første af disse programlinier får nummeret t og nummerafstanden bliver T. Man kan ikke komme til at slette nogle allerede eksisterende linier ved denne form for renummerering, da computeren protesterer, hvis de nye numre, som vil blive dannet, allerede er optaget i forvejen.

RUN efterfulgt af tryk på RETURN får computeren til at udføre det program, der er gemt i arbejdslageret.

SIZE efterfulgt af tryk på RETURN giver oplysning om, hvor meget lagerplads, der foreløbig er brugt til program og variable. Desuden oplyses, hvor meget plads, der er tilbage i arbejdslageret.

B) ... med aktivering af diskette:

ADVARSEL: Nedenstående kommandoer bør kun bruges, når der er placeret en diskette i drivet. I modsat fald er der stor risiko for at datamaten blokeres på en sådan måde, at man er nødt til at trykke på RESET for at komme i gang igen. Dette medfører at hele arbejdslageret slettes!
Noget tilsvarende kan evt. ske, hvis man bruger en af kommandoerne LIST, SAVE eller DELETE uden først at have brugt kommandoen INIT.

Altså: hver gang der er sat en ny diskette i drivet, bør man kommandere INIT.

CAT efterfulgt af tryk på RETURN giver en oversigt over de filer, som disketten indeholder. Dog vises opstartfilen for CP/M aldrig i denne oversigt.

DELETE efterfulgt af et filnavn med typeangivelse sletter den angivne fil fra disketten, hvis den ikke er skrivebeskyttet.

ENTER efterfulgt af et filnavn uden typeangivelse henter det angivne program fra disketten, hvis filens type er .CML
Bemærk, at datamatens arbejdslager ikke først slettes, men blot overskrives!

INIT efterfulgt af RETURN bevirker, at diskettens magnetiske baggrund undersøges. DENNE KOMMANDO BØR BRUGES HVER GANG EN NY DISKETTE SÆTTES I DRIVET.

LIST efterfulgt af filnavn uden typeangivelse bevirker, at arbejdslagerets programlinier gemmes i en fil med det angivne filnavn og af typen .CML såfremt disketten ikke er skrivebeskyttet.

LOAD efterfulgt af et filnavn uden typeangivelse henter det angivne program fra disketten, hvis typebetegnelsen er .CSB
Samtidig slettes alt det, der i forvejen er gemt i arbejdslageret. LOAD virker altså samtidig som kommandoen NEW.

RENAME efterfulgt af et eksisterende filnavn med typeangivelse efterfulgt af komma efterfulgt af et ikke-eksisterende filnavn med typeangivelse bevirker, at den eksisterende fil tildeles et nyt navn, såfremt disketten ikke er skrivebeskyttet.

SAVE efterfulgt af et filnavn uden typeangivelse sørger for at arbejdslagerets programlinier gemmes i en fil med det angivne navn og med typebetegnelsen .CSB såfremt disketten ikke er skrivebeskyttet.

A R B E J D S P R O G R A M M E R .

Når computeren skal gøre mere end en eneste ting ad gangen (eller lige efter hinanden) kan vi ikke blot bruge de direkte kommandoer, men må opstille et arbejdsprogram, som den skal følge. I COMAL består et program af en række ordrer, som skal udføres efter hinanden. Hver ordre er forsynet med et nummer (linienummer) og ordrerne udføres i rækkefølge efter numrenes størrelse.

Man kan få computeren til at gøre en ganske bestemt række ting, såsom at skrive noget på skærmen, printeren eller disketten, hente oplysninger fra disketten eller bede os om at give nogle oplysninger midt i det hele, og behandle disse oplysninger sådan som vi bestemmer i programmet.

Men man kan også i programmet bestemme, at computeren skal gøre det ene ELLER det andet ELLER tredie o.s.v., således at den selv skal bestemme hvad den skal gøre af disse ting. Hvad den så gør, afhænger af de ting, som vi har bestemt i programmet og af de ting, den indtil da har fundet ud af ved at behandle oplysninger.

Programmerne kan vi skrive til den eller vi kan bede den om at læse et program på disketten.
Afviklingen af et program følger efter kommandoen RUN.

Et program, der er ved at blive afviklet, kan måske stoppes ved et tryk på ESC-knappen eller ved at vi giver en bestemt oplysning undervejs.
Programmet stopper også, når computeren i programmet kommer til ordren STOP eller END.
Ligeledes stopper programmet som regel, hvis der er alvorlige fejl i det.
Og naturligvis stopper programmet, når computeren er nået til vejs ende i det.

Programmer, som man har lavet, kan gemmes på en diskette til senere brug, så man slipper for at skulle taste det ind igen. Det gøres ved kommandoen SAVE eller LIST (se side 3), og programmerne kan da hentes frem igen ved kommandoen LOAD eller ENTER (se side 3).

De programmer, som man arbejder med er gemt i arbejdslageret. Vil man se dem, kan man (mens programmet ikke er under udførelse!) få en udskrift af dem på skærmen eller printeren v.hj.a kommandoen LIST (se side 2).

Skal computeren "glemme" et program (=rense arbejdslageret) kommanderes NEW (se side 2) eller QUIT. Sidstnævnte kommando får også computeren til at "glemme" COMAL.

LOAD-kommandoen henter som overfor nævnt et tidligere skrevet program frem fra disketten. Ved udførelsen af denne kommando bliver arbejdslageret også renset ud. NB: dette gælder ikke ved ENTER-kommandoen.

Endelig kan man være så uheldig at computeren "blokeres" p.gr.a. fejl i programmet eller fejlbetjening. I så tilfælde risikerer man også at indholdet i arbejdslageret "glemmes".

V A R I A B L E .

Arbejdslageret (den plads, der er tilbage i hukommelsen, når COMAL er indlæst) bruges til to ting:
1) at huske programlinierne
2) at huske oplysninger, der bruges mens programmet udføres (variable)

Computeren husker kun de oplysninger, som vi beder den om at huske. Det gør vi ved at oprette variable og tildele dem værdier (indhold).

Når vi opretter en variabel, reserverer vi plads i arbejdslageret og ved pladsen "noterer" computeren navnet på den variabel, som vi ønsker oprettet.
Når vi tildeler en variabel en værdi, "noteres" denne værdi i arbejdslageret på variablens plads. Og her bliver værdien stående indtil vi tildeler variablen en ny værdi eller indtil arbejdslageret renses ud.

I COMAL80 kan vi arbejde med seks forskellige typer af variable:
1) simple heltalsvariable
2) simple reelle talvariable
3) simple tekstvariable
4) indicerede heltalsvariable
5) indicerede reelle talvariable
6) indicerede tekstvariable

skal dimensioneres før brug !

En TEKSTVARIABEL indeholder en tekst omgivet af anførselstegn. Tekstvariables navne skal afsluttes med $.
I arbejdslageret fylder de så meget, som de er dimensionerede til.
Dimensioneringen foregår ved ordren: DIM xxx$ OF t
hvor xxx$ er variablens navn og hvor t er et naturligt tal, som angiver hvor mange tegn teksten maksimalt må indeholde.

En HELTALSVARIABEL indeholder et helt tal mellem -32767 og +32767.
Heltalsvariables navne skal afsluttes med #.
I arbejdslageret fylder de simple heltalsvariable hver 2 bytes, mens de indicerede heltalsvariable fylder så meget som de er dimensionerede til gange 2 bytes.

En REEL TALVARIABEL indeholder et reelt positivt tal mellem
0.00000000000000000000000000000000029387359 og
170141160000000000000000000000000000000 eller et tilsvarende negativt tal. Da computeren ikke kan notere tallene med så stor præcision, hedder tallene i stedet 2.9387359 E-39 og 1.7014116 E38.
En reel talvariabel fylder i arbejdslageret 4 bytes, mens indicerede reelle talvariable fylder så meget som de er dimensionerede til gange 4 bytes.

En VARIABELS NAVN må i de nyere versioner af COMAL fylde maksimalt 80 tegn. Første tegn skal være et bogstav, mens resten skal være bogstaver eller tal.

I N D I C E R E D E V A R I A B L E .

I mange programmer er det praktisk at have flere variable, som bærer samme navn, men dog indeholder forskellige værdier. Da bruges indicerede variable.

Mens en simpel variabel kan sammenlignes med en hylde, der er reserveret til en bestemt ting, kan en indiceret variabel sammenlignes med et helt reolsystem, der er reserveret til en række ting, der er lidt forskellige, men dog af samme slags og hvor hylderne er nummererede.

Man skelner mellem enkeltindicerede variable svarende til en enkelt reolsektion med et antal hylder, dobbeltindicerede variable svarende til et reolsystem med et antal sektioner der hver har et antal hylder, og flerdobbelt-indicerede variable svarende til et helt bibliotek.

EKS. PÅ BRUG AF ENKELTINDICEREDE VARIABLE:

En række priser for en række varer kan gemmmes i en variabel ved navn PRIS. Den første pris gemmes i PRIS(1), den anden i PRIS(2), o.s.v. Ligeledes kan varernes navne gemmes i en variabel ved navn VARE$. Det første navn gemmes i VARE$(1), det andet i VARE$(2), o.s.v.

DIMENSIONERING AF INDICEREDE VARIABLE:

Enkeltindicerede talvariable dimensioneres således: DIM PRIS(t)
eller: DIM PRIS#(t) hvor t er antallet af priser, som vi vil reservere plads til i arbejdslageret.

Dobbeltindicerede talvariable dimensioneres ved DIM PRIS(t,u) eller DIM PRIS#(t,u) hvor t og u er de to maksimale indeks. Tilsvarende for flerdobbelt indicerede variable.

Samtlige variable kan da ved blot en enkelt kommando tildeles en fælles begyndelsesværdi ved f.eks.: MAT PRIS:=0

Indicerede tekstvariable dimensioneres tilsvarende ved
DIM VARE$(t) OF m eller DIM VARE$(t,u) OF m eller o.s.v., hvor t og u er de maksimale indeks, mens m er den maksimale længde af teksterne, som skal gemmes i variablen.

Også de enkelte variable i den indicerede tekstvariabel kan tildeles en fælles begyndelsesværdi ved f.eks.: MAT VARE$:=""

Man må være opmærksom på at indicerede variable kan optage megen plads i arbejdslageret. Eks.:

DIM PRIS(t)	afsætter 4*t bytes
DIM PRIS#(t)	afsætter 2*t bytes
DIM PRIS(t,u)	afsætter 4*t*u bytes
DIM PRIS#(t,u)	afsætter 2*t*u bytes
DIM VARE$(t) OF m	afsætter t*m bytes
DIM VARE$(t,u) OF m	afsætter t*u*m bytes
o.s.v.	

V A R I A B L E N F Å R V Æ R D I .

De simple talvariable oprettes ved at man tildeler dem en værdi. De indicerede talvariable derimod, og tekstvariable, skal først dimensioneres, inden de kan tildeles værdier.
Værditildeling kan ske på flere forskellige måder:

1) direkte tildeling (evt. under programafviklingen)
2) tildeling fra tastatur under programafviklingen
3) værdien læses i en data-linie under programafviklingen
4) værdien læses på disketten under programafviklingen

Når der i det følgende skrives en kommando indeholdende ordet variabel, menes der naturligvis en variabels navn!

DIREKTE TILDELING:

```
Eks.: reeltalvariabel := 13.158
      heltalsvariabel# := 13
      tekstvariabel$ := "13 agurker"
```

I disse eksempler tildeles de tre variable præcis de værdier, der står efter lighedstegnene.

```
Eks.: reeltalvariabel :+ 14.842
      heltalsvariabel# :+ 15
      tekstvariabel$ :+ " er rådnet"
```

Her forhøjes talvariablenes værdier med det tal, der står efter plustegnet, mens tekstvariablens værdi forlænges, så den herefter indeholder teksten "13 agurker er rådnet".

Ved talvariable kan man anvende tegnet :- på samme måde som :+
Efter :=, :+ eller :- kan der stå et beregningsudtryk.
I samme programlinie kan man tildele flere variable en værdi. Eks.:

```
     reeltal := 14.15 ;heltal# :+ 13 ;tekst$ := "Alt OK"
```

Tildelingerne adskilles ved semikolon.

TILDELING FRA TASTATUR: hertil bruges kommandoen INPUT.

```
Eks.: INPUT variabel
```

Denne linie i et program vil midlertidigt standse programafviklingen, mens computeren venter på et svar fra tastaturet. På skærmen vil der være et spørgsmålstegn.

```
Eks.: INPUT "Skriv prisen: ":variabel
```

Denne linie fungerer som linien ovenfor med den forskel, at der på skærmen nu noteres den tekst, der står i INPUT-sætningen i stedet for spørgsmålstegnet.
Ved hjælp af INPUT-kommandoen kan man sagtens samtidig bede om flere svar fra tastaturet. Eks.:

```
INPUT  "Angiv antal, vare og stykpris: ":var1,var$,var2
```

Efter hvert svar fra tastaturet skal der da trykkes på RETURN.

LÆSNING I DATALINIE: kræver at de relevante oplysninger i form af tal og/eller tekster er noteret i en DATA-sætning i en programlinie. Eks.:

```
DATA 3, 17, "Sørensen", "Åbenrå d."
```

Når en variabel skal tildeles en værdi fra en sådan datalinie skrives i programmet: READ variabel
Variablen tildeles da den første, som endnu ikke er brugt, af de værdier, som står i datalinien.
Næste gang en READ-sætning mødes, er det den næste værdi fra datalinien der bliver brugt.
Man kan tildele flere variable en værdi på en gang:

```
READ variabel1, variabel2, variabel3, o.s.v.
```

Ønsker man, at den næste READ-sætning skal læse forfra i datalinien, klares det med kommandoen RESTORE, der bevirker, at computeren "glemmer", at der allerede er brugt nogle af oplysningerne i datalinien.

LÆSNING PÅ DISKETTE behandles senere.

S K R I V N I N G .

At få computeren til at skrive noget er meget brugt. Der skal måske skrives noget til brugeren, og det foregår på skærmen eller printeren. Skal computeren imidlertid skrive noget til sig selv, fordi den ikke har plads nok i arbejdslageret, eller fordi den skal slukkes, inden den (en anden gang) skal bruge oplysningerne igen, bruges disketter, magnetbånd, pladelagre eller lignende til at skrive på.

Når der skrives på skærm eller printer, skrives der til mennesker. Derfor er det meget vigtigt, at det, der skrives, skrives på en ordentlig måde, således at læserne kan få gavn af det, og ikke blot bliver forvirrede.
Man må her være opmærksom på, at printeren kun kan skrive fra oven og ned, fra venstre mod højre. På skærmen derimod kan man springe rundt og skrive snart her, snart der.

Når noget skal skrives, bruges ordren PRINT. Efter ordren fortælles, hvad der skal skrives. Det kan være en tekststreng, en variabel, et udtryk indeholdende en variabel, eller en blanding af disse ting. Eks.:
```
PRINT "Navn: ";navn$;" ";adresse$;" ";postnr#;by$;"      Hold A"
```

Semikolon, der bruges som adskillelsestegn, bevirker, at det næste, der skal skrives, skrives umiddelbart efter det foregående (uden mellemrum, med mindre det foregående var et tal).
Bruges komma i stedet for semikolon, skal man være opmærksom på, at systemvariablen ZONE kan forpurre resultatet (se nedenfor).

På skærmen vil der altid blive skrevet fra "blinkerens" position, men "blinkeren" kan dirigeres rundt på skærmen til den ønskede position.
Eks.: CURSOR 55,12
Denne kommando placerer "blinkeren" på plads nummer 55 fra venstre i linie nummer 12 fra oven.
CURSOR-kommandoen virker kun på skærmen!

Vil man udskrive en hel række mellemrum, kan man bruge kommandoen:
PRINT " " men denne kommando kan erstattes af
PRINT SPC$(t), hvor t angiver, hvor mange blanktegn, der skal skrives.
Eks.: PRINT navn$,spc$(15),adresse$

Der findes en række specielle print-ordrer:
```
PRINT CHR$(7)    "blinkeren" flyttes en plads til højre
PRINT CHR$(8)    "blinkeren" flyttes en plads til venstre
PRINT CHR$(9)    som CHR$(7)
PRINT CHR$(10)   "blinkeren" flyttes en linie ned
PRINT CHR$(11)   "blinkeren" flyttes en linie op
PRINT CHR$(12)   som CHR$(7)
PRINT CHR$(13)   linieskift
PRINT CHR$(14)   som CHR$(7)
PRINT CHR$(29)   lader "blinkeren" stå, men sletter resten af skærmsiden
PRINT CHR$(30)   fører "blinkeren" til første linie første plads
PRINT CHR$(31)   lader "blinkeren" stå, men sletter resten af linien
PRINT CHR$(t)    hvor t er et tal i området 0-255, men dog forskellig
```
fra ovenstående tal, udskriver et skrivetegn, der svarer til tallet t.

NB: Mange af disse specielle PRINT CHR$() -ordrer, hvor tallet er angivet, virker naturligvis kun på skærmen.

ZONE er en speciel variabel, der bruges til at inddele skærmen i skrivezoner. I nogle versioner af COMAL bruges TAB i stedet for ZONE.

SELECT OUTPUT "LP:" bevirker at alle efterfølgende print-kommandoer udføres på printeren, indtil kommandoen SELECT OUTPUT "DS:" mødes.

P R I N T T A B / P R I N T U S I N G .

Disse to kommandoer er meget velegnede, når det gælder om at få en god, ordnet udskrift - og det gør det jo altid! Kommandoerne virker både på skærm og printer.

Til almindelig tabulering bruges PRINT TAB(t), hvor t angiver, hvilken plads i den aktuelle linie, "blinkeren" skal placere sig i, inden der skrives videre. Eks.:

```
PRINT navn$; TAB(35); adresse$
```

Taludtrykket i parentesen skal være så stort, at "blinkeren" enten bliver stående eller dirigeres mod højre.

Kommandoen PRINT USING er lidt mere besværlig at forklare, men er dog en særdeles hjælpsom facilitet. Kommandoen ser således ud:

```
PRINT USING "streng-udtryk": variabeludtryk
```

Eksempler:

```
PRINT USING "Aabenraa d. ##/##-##": dd,mm,åå
PRINT USING "### stk. ############# til i alt kr. #####.##": antal,vare,
antal*enhedspris
```

Den tekst og de mellemrum og tegn (undtagen #), der findes i strengudtrykket, bliver udskrevet sådan som de står i strengudtrykket. Et # eller flere #'er efter hinanden markerer, at her skal computeren indsætte værdien af det/de variabeludtryk, der står efter kolon.
Hvis der mellem en række #'er uden mellemrum er sat et punktum opfattes det at computeren som et komma i et tal. Således opfattes ###.## som en pladsmarkering til et tal, der højst er 999.99 Hvis tallet er mindre end 100 bliver første # erstattet med et blanktegn.

NÅR VARIABELUDTRYKKETS VÆRDI ER ET TAL placeres det højrestillet på den plads, der er lavet til det i strengudtrykket. Hvis tallets værdi er for stor i forhold til den tilrettede plads, udskrives tallet ikke. Har det derimod "kun" for mange decimaler i forhold til den tilrettede plads, afrundes tallet inden det skrives.
Et + foran #'erne bevirker, at der altid skrives et fortegn foran tallet, hvad enten det er positivt eller negativt.
Et - foran #'erne bevirker, at kun negative tal skrives med fortegn.

NÅR VARIABELUDTRYKKETS VÆRDI ER EN TEKSTSTRENG placeres det venstrestillet. Er tekststrengen for lang, skrives det, der er plads til, mens resten smides væk. Er den omvendt for kort, suppleres der med blanktegn efter strengen.

En PRINT USING -kommando affsluttes, når alle variabeludtryks værdier er udskrevet. Er der flere variabeludtryk end pladser, begyndes der forfra i format-tekststrengen.

Hvis et streng-udtryk i en PRINT USING -kommando er langt og skal bruges flere gaange i et program, kan det betale sig at definere det først ved:

```
DIM FORM$ OF t   hvor t er længden
```

hvorefter det kan fyldes med indhold ved:

```
FORM$:="### stk. ################ til i alt kr. #####.##"
```

Senere kan det så bruges:

```
PRINT USING FORM$: variabel1, tekstvar$,variabel2
```

og eventuelt genbruges mange gange!

S E L V V A L G .

Man kommer ofte ud for at have brug for, at computeren selv kan tage stilling til om den skal udføre det ene eller det andet eller tredie eller .... af en række forskellige programafsnit.

Hvis man kan formulere en variabel, hvis værdi kan afgøre, hvilke af programafsnittene, der skal udføres, kan man i programmet diktere computeren, at den selv skal finde og udføre det relevante programafsnit. Det sker ved:

```
CASE variabel OF
WHEN
     programafsnit 1
WHEN
     programafsnit 2
WHEN
     programafsnit 3
WHEN
  :
  :
OTHERWISE
     alternativt program-
     afsnit
ENDCASE
```

Efter hvert WHEN skrives den/de værdi(er), som variablen må/skal antage, for at det efterfølgende programafsnit udføres.
For en ordens skyld må nævnes at kun et af disse programafsnit kan bringes til udførelse, når CASE-kommandoen udføres.
Det alternative programafsnit kan udelades, hvis det ønskes. I så tilfælde kan også OTHERWISE udelades, MEN KUN hvis variablen ikke kan antage andre værdier end de, der er nævnt efter WHEN'erne.

En anden måde, hvorpå computeren kan programmeres til selvvalg, bygger på IF-kommandoen. Det kræver at man kan formulere nogle betingelser, der skal være opfyldt for at programafsnittene udføres:

```
IF betingelse
     programafsnit
ENDIF
```

```
IF betingelse
     programafsnit
ELSE
     alternativt
     programafsnit
ENDIF
```

```
IF betingelse 1
     programafsnit 1
ELIF betingelse 2
     programafsnit 2
ELIF betingelse 3
  :
  :
ELSE
     alternativt
     programafsnit
ENDIF
```

Som det ses, findes IF-kommandoen i flere udgaver, afhængigt af, hvor mange alternative programafsnit man ønsker, at computeren skal vælge imellem.
Betingelserne skal være ægte udsagn, således at der ikke kan herske tvivl om betingelsernes sandhedsværdi, når IF-kommandoen udføres. Er der flere programafsnit at vælge imellem, udføres højst et af dem. Computeren undersøger betingelsernes sandhedsværdier fra oven. Findes en sand betingelse, udføres det tilhørende programafsnit, hvorefter IF-kommandoen er overstået. Findes ingen sande betingelser, udføres det alternative programafsnit, hvis et sådant er noteret i programmmet.

Inde i de skitserede programafsnit, som computeren skal vælge imellem, kan der igen være nye CASE-kommandoer eller nye IF-kommandoer. De to selvvalgsformer kan altså optræde inde i hinandens programafsnit, ligesom der kan være flere CASE-kommandoer i CASE-kommandoens programafsnit og flere IF-kommandoer inde i IF-kommandoens programafsnit.

Det er blandt andet disse to kommandoer (sammen med gentagelseskommandoerne) der sætter computeren i stand til at behandle selv meget komplicerede opgaver ret hurtigt.

O P E R A T O R E R .

Når man skal operere med sandhedsværdier eller udtryksberegninger, må man nødvendigvis bruge nogle operatorer. Der er fire typer:

1) regne-operatorer:

^	potensopløftning, eks.: x^3 (x i tredie)
/	divisionstegn
*	multiplikationstegn
DIV	heltalsdivision, virker som / men runder altid ned til nærmeste hele tal
MOD	modulus, angiver "resten" ved division
+	plus
-	minus

2) streng-operatorer:

&	som + nedenfor
+	forlænger den tekststreng, der står til venstre for + med den tekststreng, der står til højre for +
IN	undersøger om tekststrengen, der står til venstre for IN er indeholdt i tekststrengen til højre for IN. Resultatet er enten 1 (=TRUE) eller 0 (=FALSE).

3) logiske operatorer:

AND	anbragt mellem to udsagn/logiske udtryk, svarer til det matematiske tegn "både og", der i matematikken skrives: ü
OR	anbragt mellem to udsagn/logiske udtryk, svarer til det matematiske tegn "eller", som skrives: v
NOT	som - nedenfor
-	negationstegn, må optræde foran udsagn/logiske udtryk, og bevirker skift til modsatte sandshedsværdi.

Ved disse logiske operatorer bliver resultatet altid 1 (=TRUE) eller 0 (=FALSE)

4) relations-operatorer:

<	mindre end (hvis det er anbragt mellem to tal eller beregningsudtryk)
	kommer før (hvis det er anbragt mellem to tekststrenge)
<=	mindre end eller lig med
	kommer før eller er lig med
=	er lig med
>	større end (hvis det er anbragt mellem to tal eller beregningsudtryk)
	kommer efter (hvis det er anbragt mellem to tekststrenge)
>=	større end eller lig med
	kommer efter eller er lig med
<>	er forskellig fra

Som det fremgår kan disse relationstegn skrives mellem to tal/beregningsudtryk eller mellem to tekststrenge. I alle tilfælde bliver resultatet enten 1 (=TRUE) eller 0 (=FALSE).

Når relationsoperatorerne bruges ved tekststrenge må man erindre, at alle taster på tastaturet faktisk repræsenterer et tegn. Når computeren sorterer efter "alfabetisk" orden, sorterer den i virkeligheden efter størrelsen af de talkoder (ASCII-værdier), som tegnene er repræsenteret af i computerens hukommelse.

Den "alfabetiske" orden er nogenlunde følgende:

7 semigrafiktegn fra venstre mod højre uden SHIFT
pil mod venstre (=BS), TAB, øvrige pile
øvrige semigrafiktegn uden SHIFT, semigrafiktegn med SHIFT, mellemrum
! " # $ % ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
store bogstaver i alfabetisk orden, Æ Ø Å
små bogstaver i alfabetisk orden, æ ø å

O M A T C Y K L E (L Ø K K E R) .

Som tidligere omtalt, skal computeren have et program at køre efter. Et program, hvor den kan gå frem fra ordre til ordre, indtil programmet er slut. Undervejs kan den tage stilling til en række betingelser, der afgør, om den skal udføre visse programafsnit eller blot overspringe dem. Ligeledes kan computeren bringes til at køre i ring i programmet, hvorved en række ordrer gentages flere gange, eventuelt uendeligt mange gange (indtil den slukkes eller standses). Sådanne ringafsnit i et program kaldes løkker. Der er fire typer af løkker:

```
REPEAT
   programafsnit
UNTIL betingelse
```

Programafsnittet udføres en gang, hvorefter det gentages indtil betingelsen er opfyldt. Hvis computeren skal kunne slippe ud af løkken igen, skal betingelsen indeholde en variabel, der påvirkes i programafsnittet, og som på et eller andet tidspunkt vil gøre betingelsen sand. I modsat fald bliver computeren ved med at gentage programafsnittet i en uendelighed, indtil den eventuelt slukkes eller standses ved et tryk på ESC.

```
WHILE betingelse DO
   programafsnit
ENDWHILE
```

Programafsnittet udføres kun, hvis betingelsen er sand, og det gentages herefter, indtil betingelsen bliver falsk. Ligesom REPEAT-løkken skal betingelsen her normalt indeholde en variabel, som påvirke i programafsnittet.

```
LOOP
   programafsnit
ENDLOOP
```

Dette er principielt en uendelig løkke, som computeren ikke uden videre slipper ud af igen. Dog kan man i programmet indføje en mulighed for at der kan springes ud af løkken. Det gøres ved:

```
IF betingelse THEN EXIT
```

I så fald må betingelsen nødvendigvis indeholde en variabel, som påvirkes i det øvrige programafsnit, for at computeren kan få mulighed for at udføre EXIT-kommandoen. Når EXIT udføres, fortsætter computeren i næste programlinie efter ENDLOOP.
De tre ovenanførte løkker ligner alle hinanden derved, at de skal indeholde en betingelse, som enten skal være sand (ved REPEAT og LOOP) eller falsk (ved WHILE) for at computeren kan hoppe ud af løkken.

Den fjerde løkke-type er en løkke, som altid skal udføres et bestemt antal gange. Hvor mange gange, bestemmes ved programmeringen.

```
FOR tællevariabel=begyndelsesværdi TO slutværdi DO
   programafsnit
NEXT tælleværiabel
```

Computeren forhøjer tællevariablen med 1 hver gang løkken er gennemløbet. Når slutværdien nås, gennemløbes løkken for sidste gang. Antallet af gennemløb kan således fastlægges helt præcist. Hvis slutværdien fastlægges på en sådan måde, at den er uopnåelig, er løkken uendelig!
Ved FOR-løkken kan der også tælles baglæns (nedtælling):

```
FOR tællevariabel=begyndelsesværdi DOWNTO slutværdi DO
   programafsnit
NEXT tællevariabel
```

FOR-løkken kan ofte med fordel bruges ved brug af indicerede variable, idet man da kan lade variablens indeks være identisk med løkkens tællevariabel. I den forbindelse kan man få brug for, at tællevariablens værdi ikke kun kan ændres med +1 eller -1 for hvert gennemløb. I så fald må løkken udstyres med en hopværdi, der afgør, hvorledes tællerens værdi skal ændres efter hvert gennemløb:

```
FOR tællevariabel=begyndelsesværdi TO slutværdi STEP hopværdi
   programafsnit
NEXT tællevariabel
```

Udover nedennævnte standardfunktioner, kan brugeren selv definere en hel række andre funktioner. Dette beskrives dog ikke her.

Forkortelser: xr = reelt tal eller reel talvariabel (regneudtryk)
xh = helt tal eller heltalsvariabel (regneudtryk)
x = en størrelse af typen xr eller xh
t$ = tekst eller tekstvariabel (tekstvariabeludtryk)

ABS(x) angiver den absolutte værdi af x.

ATN(x) angiver arctangens til x, hvor x ikke måles i grader, men i radianer. 1 grad = pi/180 radianer.

BSTR$(x) angiver tallet x som en tekst, der udtrykker tallet x på binær form med 8 cifre. x afrundes først til et helt tal. x skal ligge i intervallet fra 0-255.

BVAL(t$) angiver det hele tal, der i t$ er angivet på binær form med 8 cifre. Modsat til BSTR$(x).

CHR$(x) angiver det tegn, som har ASCII-koden x. x afrundes først til et helt tal. x skal ligge i intervallet 0-255. Modsat til ORD(t$).

COS(x) angiver cosinus til x, hvor x ikke måles i grader.Se ATN(x)

ERRTEXT$(x) angiver den fejltekst, der svarer til fejlmelding nr x. Virker kun,når fejlmeldinger ikke er frakoblet ved opstart.

EXP(x) angiver resultatet af eksponentialfunktionen anvendt på x. Modsat til LOG(x).

FRAC(xr) angiver decimaldelen i tallet xr.Svarer til ABS(xr-INT(xr))

INT(xr) angiver heltalsdelen af xr. d.v.s. det tal, som fremkommer, når decimaldelen bortkastes. Resultatet heraf er et helt tal, MEN det opfattes stadig som et reelt tal af computeren - sammenlign med TRUNC(xr).

IVAL(t$) angiver det hele tal,der er noteret som tekst i t$. Sammenlign med VAL(t$).

LEN(t$) angiver længden af tekststrengen t$.

LOG(x) angiver den naturlige logaritme til x. Modsat til EXP(x).

ORD(t$) angiver ASCII-koden for første tegn i teksten t$.

POS(t1$,t2$) angiver startpositionen for den første forekomst af t1$ i t2$ (regnet fra venstre mod højre). Hvis t1$ ikke er med i t2$ bliver resultatet 0. Hvis t1$="" bliver resultatet 1.

RND angiver et tilfældigt reelt tal i intervallet 0-1.

RND(x,y) angiver et tilfældigt helt tal i intervallet x-y. x og y afrundes først af computeren til hele tal.

ROUND(xr) angiver det hele tal, der fremkommer ved sædvanlig afrunding af xr. Resultatet opfattes af computeren som et helt tal. Sammenlig med INT og TRUNC.

SQR(x) angiver kvadratroden af x.

SGN(x) angiver om x er positiv, negativ eller nul. Resultatet bliver et helt tal: -1 (for negativ), 0 (for nul) eller 1 (for positiv).

SIN(x) angiver sinus til x, hvor x måles i radianer. Se ATN(x).

SPC$(x) angiver en tekst bestående af x mellemrum (blanktegn). x afrundes først af computeren til et helt tal.

STR$(x) angiver tallet x skrevet som en tekst,der netop (kun) indeholder tallet x.

TAN(x) angiver tangens til x, hvor x måles i radianer. Se ATN(x).

TRUNC(xr) angiver det hele tal,som fås ved bortkastning af decimalerne. Resultatet opfattes af computeren som et helt tal.

VAL(t$) angiver det reelle tal, der i t$ er noteret som en tekst. Modsat til STR$(x).

O M P R O G R A M M E R I N G S K U N S T .

Når man skal programmere en datamat, må man nødvendigvis bruge et programmeringssprog. Sproget skal naturligvis kunne bruges på datamaten, men det skal også kendes af programmøren.

Er disse betingelser opfyldt, kan programmøren lave mange programmer - måske endda nogle, der virker efter hensigten. Hvis programmet imidlertid ikke virker efter hensigten, kan programmøren få store kvaler. Der må jo da være nogle fejl i programmet! - eller hvad? Hvad er det nu den variabel betyder? Hvad skal computeren egentlig lave i dette afsnit af programmet? - Jeg må nok hellere spørge en person, der har lidt mere forstand på disse ting, end jeg selv har.

Men, ak, den dygtigere kan måske heller ikke klare det, thi der er umuligt at få overblik over programmet og det, der sker i det!

Det er ingen sag at lave små programmer og få dem til at fungere. Men der er kunst i at få indviklede programmer til at være OVERSKUELIGE !!

REGEL 1: Giv alle variable sådanne navne, at de forklarer sig selv.

REGEL 2: Inddel programmet i små, overskuelige afsnit (procedurer) og giv dem navne, således at de forklarer sig selv.

REGEL 3: Skriv kommentarer om nødvendigt v. hj.a. REM (//).

REGEL 4: Lad være med at bruge kommandoen GOTO, da kun en dygtig programmør kan anvende den fornuftigt.

Det må bemærkes, at et program ofte kan opdeles i flere forskellige selvstændige programmer, som man kan skifte imellem v.hj.a. CHAIN.

D E T Y D R E L A G E R (F I L E R).

Computeren har et indre lager (hukommelsen), hvori den noterer sig de programlinier, den skal arbejde efter, og de oplysninger (variable), som den i den forbindelse har brug for. Fordelen ved at bruge det indre lager er, at det går meget hurtigt, når oplysningerne "noteres" eller "læses". Ulempen er, at dette lager slettes, når strømmen afbrydes, f. eks. når computeren slukkes eller reset'es. Også når man bruger i COMAL-80 bruger en af kommandoerne CHAIN, LOAD, NEW eller RUN kan alle oplysningerne blive slettet.
Endelig er der den hage ved det indre lager, at det kan være for lille til at rumme alle de oplysninger, som computeren skal bruge ved de forskellige programkørsler.

Disse problemer løser man ved at bruge det ydre lager, der i princippet kan være hvad som helst (blot det er tilpasset computeren). I praksis er det for tiden disketter, vi arbejder med.
Brugen af det ydre lager giver imidlertid andre ulemper: det er meget langsommere.

Computeren opfatter det ydre lager som en uendelig stor hal fyldt med skuffer, hvori der kan gemmes oplysninger. Skuffernes størrelse bestemmes af det, der "noteres" i dem. - En sådan skuffe kaldes en FIL.

Nogle skuffer er lange og smalle, så oplysningerne må "noteres" i en lang række efter hinanden i dem. Følgelig kan de også kun "læses" i samme rækkefølge, fra begyndelsen til enden, og man kan ikke både "læse" og "skrive" i dem i den samme arbejdsgang. - Sådanne filer kaldes SEKVENTIELLE filer.

Andre skuffer kan, når de bruges, inddeles i en række små rum, der er nummererede og alle har samme størrelse, men størrelsen bestemmes fra computeren, når den åbner skuffen for at "notere" i den. De enkelte rum kaldes POSTER. Når en sådan fil er åben, kan computeren i samme arbejdsgang både "læse" og "skrive" i en hvilken som helst post i den. Der er altså DIREKTE TILGANG til disse filer, der normalt kaldes for RANDOM-filer. Deres typebetegnelse er altid .RAN

For alle filer gælder, at computeren hverken kan "læse" eller "skrive" i dem, med mindre de er åbne. Når en RANDOM-fil er åben, kan der både "læses" og "skrives" i den. En SEKVENTIEL fil derimod skal enten være åben for at computeren kan "skrive" eller den skal være åben for at computeren skal "læse".
På samme måde gælder det, at computeren sagtens kan "skrive" noget nyt i en RANDOM-fil, hvor der i forvejen er "noteret" noget (det gælder også i en post, der i forvejen er optaget), hvorimod der ikke kan ændres overhovedet på det, der en gang er "noteret" i en SEKVENTIEL fil. Hvis der altså skal ændres på indholdet i en SEKVENTIEL fil, skal den først slettes totalt (mens den er lukket!), hvorefter den skal skrives helt om fra begyndelsen til enden.

Filer bruges både til at gemme resultater fra en programkørsel for at de senere kan hentes frem igen (evt. af et andet program) og til at gemme selve programmerne i.

R A N D O M - f i l e r .

En random-fil er en fil, hvis hukommelsesområde er delt op nummererede felter, der kaldes poster. Alle poster i samme fil er lige store, og størrelsen blev bestemt, da filen blev åbnet første gang.
Åbning af en random-fil sker f.eks. ved:

```
OPEN FILE 5, "Navne", RANDOM, 35
```

Filen "Navne.RAN" er herefter åben for læsning/skrivning. Dens postlængde er sat til 35, dvs. at de variable, der skal gemmes i dens poster maksimalt må optage 35 bytes i alt. Tallet 35 er bestemt af programmøren, idet der skal bemærkes, at:
reelle tal fylder 4 bytes - hele tal fylder 2 bytes - tekster fylder det de er dimensionerede til + 2 bytes.

Efter at filen er åbnet, behøver vi ikke længere fortælle computeren, hvad filen hedder, når der skal skrives/læses i den. Det er nok at angive filens nummer (her: 5). Nummeret bestemmes af programmøren, men det skal være et helt tal mellem 0 og 9. At der er flere tal at vælge imellem betyder, at vi kan have flere forskellige filer åbnet samtidig.

Når man vil læse/skrive i en post (f.eks. nr. 17) klares det ved:

```
READ FILE 5, 17: variable        (læsning)
WRITE FILE 5, 17: variable       (skrivning)
```

"variable" står for en række (mindst en) variable adskilt med komma.

Efter endt brug lukkes filen ved: CLOSE FILE 5

D A T A - f i l e r .

Sekventielle filer er mere simple at bruge, da man ikke skal tage hensyn til postnumre og postlængde. Til gengæld kan de ikke samtidig være åbne for læsning og skrivning, og man kan ikke rette i dem uden først af slette dem og derefter på ny skrive dem forfra.

Filerne åbnes ved:

```
        OPEN FILE 5, "Navne.DAT", WRITE        (for skrivning)
eller   OPEN FILE 5, "Navne.DAT", READ         (for læsning)
```

Bemærk, at typebetegnelsen .DAT angives som en del af navnet. Fidusen er, at man kan angive en anden speciel eller personlig typebetegnelse i stedet for .DAT - f.eks. kunne man kalde filen "Navne.JAN" - og da kan man altid se i en kataloguds kriftt, at her er en speciel fil. Hvis man ingen typebetegnelse angiver, sættes den automatisk til .DAT

Læsning/skrivning foregår ved:

```
        READ FILE 5: variable
og      WRITE FILE 5: variable
```

Efter endt brug lukkes filen ved: CLOSE FILE 5

Det må bemærkes, at hvis man kun skriver CLOSE, vil computeren lukke alle de filer, der er åbne.

Ved den her nævnte skrive/læse-måde vil computeren automatisk forstå, at alle oplysninger skal lagres på binær form. Ønsker man i stedet at lagre oplysningerne på hexadecimal form, bruges LIST i stedet for WRITE og ENTER i stedet for READ. - Den binære form er pladsbesparende. Til gengæld vil den hexadecimale form lettere kunne læses af andre computere.

```
9796 //.....................................................................
9798 //
9800 PROC SORTERTAST1
9802   IF T#)31 AND T#(127 THEN
9804     IF T#(65 OR T#=94 OR T#=95 OR T#=96 OR T#=126 THEN
9806       IF T#)47 AND T#(58 THEN
9808         TEKST$:="tallet:"
9810       ELSE
9812         TEKST$:="skrivetegnet:"
9814       ENDIF
9816     ELSE
9818       TEKST$:="bogstavet:"
9820     ENDIF
9822     TAST$:=CHR$(T#)
9824   ELIF (T#)0 AND T#(7) OR (T#)14 AND T#(29 AND T#()27) THEN
9826     TEKST$:="grafiktegnet:"; TAST$:=CHR$(T#)
9828   ELSE
9830     TEKST$:="en speciel funktion"; TAST$:=SPC$(1)
9832   ENDIF
9834 ENDPROC SORTERTAST1
9836 //
9838 PROC SORTERTAST2
9840   CASE T# OF
9842   WHEN 0, 7, 8, 9, 10, 11, 12, 13, 14, 27, 30, 31, 127
9844     TEKST$:="en speciel funktion"; TAST$:=SPC$(1)
9846   WHEN 1, 2, 3, 4, 5, 6, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28
9848     TEKST$:="grafiktegnet:"; TAST$:=CHR$(T#)
9850   WHEN 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47
9852     TEKST$:="skrivetegnet"; TAST$:=CHR$(T#)
9854   WHEN 58, 59, 60, 61, 62, 63, 64, 94, 95, 96, 126
9856     TEKST$:="skrivetegnet:"; TAST$:=CHR$(T#)
9858   WHEN 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
9860     TEKST$:="tallet:"; TAST$:=CHR$(T#)
9862   OTHERWISE
9864     TEKST$:="bogstavet:"; TAST$:=CHR$(T#)
9866   ENDCASE
9868 ENDPROC SORTERTAST2
9870 //
9872 PROC SORTERTAST3
9874   CASE T# OF
9876   WHEN 0, 7, 8, 9, 10, 11, 12, 13, 14, 27, 29, 30, 31, 127
9878     TEKST$:="en speciel funktion"; TAST$:=SPC$(1)
9880   WHEN 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
9882     TEKST$:="tallet:"; TAST$:=CHR$(T#)
9884   OTHERWISE
9886     IF T#(29 THEN
9888       TEKST$:="grafiktegnet:"
9890     ELIF (T#)64 AND T#(94) OR (T#)96 AND T#(126) THEN
9892       TEKST$:="bogstavet:"
9894     ELSE
9896       TEKST$:="skrivetegnet:"
9898     ENDIF
9900     TAST$:=CHR$(T#)
9902   ENDCASE
9904 ENDPROC SORTERTAST3
9906 //
```

```
0010 EXEC SKRIVE
0020 //
0030 //
9000 //-----------------------------------------------------------------------------
9010 // PROCEDURER TIL BRUG I EDITERING (v. JPP, Brundlundskolen).
9020 //
9030 PROC LÆSTAST
9040   TRAP ESC-
9050   POKE 256, 255
9060   REPEAT
9070     T#:=PEEK(256)
9080     IF ESC THEN T#:=27
9090     TRAP ESC-
9100   UNTIL T#<>255
9110   TRAP ESC+ // udelades hvis ESC-knappen skal annuleres totalt.
9120 ENDPROC LÆSTAST
9130 //
9140 PROC SKRIVE
9150   OP#:=1; NED#:=-1; VM#:=8; HM#:=77; LA#:=1; SLUT#:=0; BRED#:=HM#-VM#+1
9160   X#:=VM#; LINIE#:=1; Y#:=1; NR#:=X#-VM#+1
9170   CLEAR
9180   PRINT TAB(7);""16""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""
       5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5"";
9190   PRINT ""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5""5"
       "5""5""5""5""5""5""17""
9200   FOR I#:=1 TO 20 DO
9210     PRINT I#;TAB(7);""4"";TAB(78);""4""
9220   NEXT I#
9230   PRINT ""5""5""5""5""5""5"0"5""5""5""5""5""5""5""5"1"5""5""5""5""5""5""
       5""5""5"2"5""5""5""5""5""5""5""5"3"5""5""5""5""5""5""5""5"4";
9240   PRINT ""5""5""5""5""5""5""5""5""5"5"5""5""5""5""5""5""5""5""5"6"5""5""5""
       5""5""5""5""5""5"70"5""5""
9250   REPEAT
9260     CURSOR X#, Y#+1
9270     EXEC LÆSTAST
9280     IF T#>31 AND T#<127 THEN // T# indeholder ASCII-koden for skrivetegn.
9290       PRINT CHR$(T#);
9300       X#:+1; NR#:+1
9310     ELSE
9320       CASE T# OF
9330       WHEN 0
9340         SLUT#:=1
9350       WHEN 1, 2, 3, 4, 5, 6, 7, 14, 15, 16, 17, 18
9360       WHEN 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30
9370       WHEN 8 // BS eller venstrepil
9380         X#:-1; NR#:-1
9390       WHEN 9 // TAB
9400       WHEN 10 // pil ned
9410         Y#:+LA#; LINIE#:+LA#; NR#:+LA#*BRED#
9420       WHEN 11 // pil op
9430         Y#:-1; LINIE#:-LA#; NR#:-LA#*BRED#
9440       WHEN 12 // højrepil
9450         X#:+1; NR#:+1
9460       WHEN 13 // RETURN
9470         NR#:+(HM#-X#+1); X#:=VM#; Y#:+LA#; LINIE#:+LA#; NR#:+(LA#-1)*BRED#
9480       WHEN 27, 31, 127
9490       OTHERWISE
9500       ENDCASE
9510     ENDIF
9520     IF X#<VM# THEN X#:=HM#; Y#:-LA#; LINIE#:-LA#; NR#:-(LA#-1)*BRED#
9530     IF X#>HM# THEN X#:=VM#; Y#:+LA#; LINIE#:+(LA#-1)*BRED#
9540     IF Y#<1 THEN EXEC BILLEDRUL(NED#)
9550     IF Y#>20 THEN EXEC BILLEDRUL(OP#)
9560   UNTIL SLUT#
9570 ENDPROC SKRIVE
9580 //
9590 PROC BILLEDRUL(I#)
9600   //.......................stub
9610   NR#:-I#*LA#*BRED#; Y#:-I#*LA#; LINIE#:-I#*LA# // skal senere slettes.
9620 ENDPROC BILLEDRUL
```