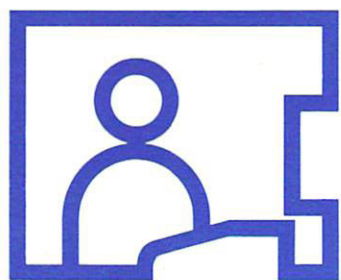


# COMAL 80



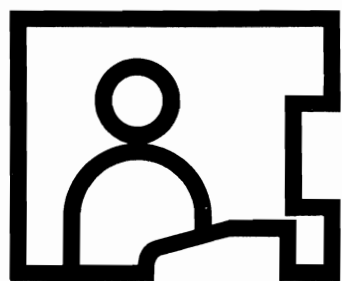






**COMAL**

for Commodore 64



**COMAL**

for Commodore 64



©English edition copyright, 1985  
by Frank Bason, Leo Højsholt-Poulsen  
and Commodore Data A/S, Horsens, Denmark  
all rights reserved

©Danish edition, 1985  
by Leo Højsholt-Poulsen, Frank Bason  
and Commodore Data A/S, Horsens, Denmark.  
Konsulenter: UniComal ApS, Jels, Rødding  
Redaktion: Commodore Data A/S, Horsens  
Omslag og illustrationer: Fejltræk, Chr. VIII Vej 14.1,  
8600 Silkeborg  
Sats: Werks Fotosats, Århus  
Tryk: Werks Offset, Århus

## ANSVAR

Hverken Commodore Data A/S eller dette firmas forhandlere påtager sig noget ansvar, direkte eller underforstået, vedrørende det heri omtalte programmeringssprogs kvalitet, ydeevne, værdi eller anvendbarhed til specielle opgaver. Programmet sælges "som beset". Risiko med hensyn til kvalitet og ydeevne tages af køberen. Såfremt programmet skulle udvise defekter efter anskaffelsen, må køberen (og ikke fremstilleren af programmet, Commodore Data A/S, dets distributører og forhandlere) påtage sig alle nødvendige reparations og service-omkostninger, i forbindelse med enhver tilfældig skade forårsaget af disse defekter i programmet.

## COPYRIGHT

Computersproget COMAL for Commodore 64 er omfattet af copyright

©Commodore Data A/S og UniComal ApS 1984,1985

Den dertil hørende dansk manual er omfattet af copyright 1985 Leo Højsholt-Poulsen, Frank Bason og Commodore Data A/S, Danmark. Ingen del af systemet, programkapslen eller denne manual må reproducere, lagres i et genfindingsystem eller på nogen måde transmitteres elektronisk, mekanisk, fotokopieres eller på anden måde optages, uden forud indhentet tilladelse fra indehaveren af copyright. Det er forbudt at lave kopier af COMAL programkapslen, hvorimod Demonstrationsdisketten eller båndet godt må kopieres.

## HJÆLP

Hvis De har nogle kommentarer vedrørende denne COMAL manual eller selve programmeringssproget, kontakt da venligst Deres forhandler. Commodore Data A/S har bestræbt sig på at gøre indholdet af manualen korrekt og at selve programmeringssproget virker efter hensigten. Fejl opdaget af brugere rettes så hurtigt som muligt. Deres evt. hjælp i så henseende vil blive værdsat meget af andre brugere.

# INDHOLDSFORTEGNELSE

<b>FORORD</b>	9
COMAL 80	9
COMAL's oprindelse	10
COMAL og Commodore	10
Hvordan udnyttes denne vejledning?	11
 <b>Kapitel 1 OPSTILLING AF COMPUTEREN</b>	13
Computeren og tilbehør	13
Montering af COMAL-kapslen	15
Tilslutning af TV eller monitor	16
Lidt om tastaturet	18
Tal dansk med din computer	19
Hent demonstrationsprogrammet	19
Brug af Datassette	19
Brug af diskettestation	21
Kør demonstrationsprogrammet	21
Resumé	22
 <b>Kapitel 2 LAD OS SÅ KOMME I GANG!</b>	23
Hvorfor lære at programmere?	23
Direkte udførelse af COMAL kommandoer	26
Et hurtigt kig på turtle grafik	28
Hvad er et program?	31
Gentagelse af instruktioner	35
COMAL procedurer	37
Lagring af programmer og procedurer	40
Brug af en Datassette enhed	40
Brug af en diskettestation	42
Repetition	42
 <b>Kapitel 3 PROGRAMMERING MED COMAL</b>	45
Få gode programmeringsvaner	45
Den første beregning	46
INPUT sætningen	48
Cirkler	50
Procedurer I	52
COMAL og tekster	55

Forgreninger: betingede udførelser .....	58
CASE-strukturen .....	61
Gentagelser: løkker .....	64
Tabeller: indicerede variabler .....	69
Teksttabeller .....	71
Procedurer II .....	74
Lokale og globale navne .....	75
Funktioner .....	78
Strengfunktioner .....	82
Lukkede procedurer .....	83
Filbehandling .....	87
Fejlhåndtering .....	90

<b>Kapitel 4 COMAL OVERSIGHT</b> .....	95
Kommandoer, som benyttes før og under indtastning af et program: .....	95
NEW-AUTO-RENUM	
Kommandoer, som benyttes til redigering i program: .....	96
EDIT-FIND-CHANGE-DEL-SCAN	
Andre kommandoer .....	98
SETEXEC	
Kommandoer, som benyttes til oversigt over disketteindhold ...	99
SIZE-CAT-DIR	
LIST-ENTER-MERGE-DISPLAY	
SAVE-LOAD	
RUN-CHAIN-CON	
STATUS-STATUS\$	
VERIFY	
COPY-DELETE-RENAME-PASS	
SELECT INPUT SELECT OUTPUT	
Kommandoer i forbindelse med systemopstart .....	106
BASIC-SYS til COMAL	
Kommandoer og sætninger vedrørende brug af programpakker i maskinkode .....	106
USE-LINK-DISCARD	
Sætninger, som benyttes under indlæsning og udskrift .....	107
INPUT-INPUT AT-KEY\$	
PRINT-PRINT AT-PRINT USING-TAB-ZONE	
PAGE-CURSOR	
READ-DATA-RESTORE-Etikette:-EOD	
Kommunikation med filer .....	112
MOUNT-CREATE	
OPEN FILE/OPEN-READ-WRITE-APPEND-RANDOM	
PRINT FILE-INPUT FILE	
WRITE FILE-READ FILE	
CLOSE FILE/CLOSE	
EOF	

UNIT-UNIT\$	
Sætningsstrukturer .....	117
Betingelsessætninger .....	117
IF-THEN-ELIF-ELSE-ENDIF	
CASE-OF-WHEN-OTHERWISE-ENDCASE	
Gentagelsessætninger .....	120
REPEAT-UNTIL	
WHILE-DO-ENDWHILE	
FOR-TO-STEP-DO-ENDFOR	
LOOP-EXIT-EXIT WHEN-ENDLOOP	
Fejlhåndtering .....	123
TRAP-HANDLER-ENDTRAP	
ERR-ERRFILE-ERRTEXT\$	
REPORT	
GOTO-<Etikette:> .....	126
Procedurer .....	126
PROC-ENDPROC	
REF-CLOSED-IMPORT	
EXTERNAL-MAIN	
Funktioner .....	131
FUNC-ENDFUNC-RETURN	
Andre funktioner .....	133
ABS-INT-SGN-SQR-PI	
COS-SIN-TAN-ATN	
LOG-EXP	
CHR\$-STR\$-SPC\$	
ORD-VAL-LEN	
TRUE-FALSE	
TIME	
RANDOMIZE-RND	
ESC-TRAP ESC	
Operatorer .....	139
DIV-MOD	
Logiske operatorer .....	140
NOT-AND-AND THEN-OR-OR ELSE-IN	
BITAND-BITOR-BITXOR	
Andre ordrer .....	144
//	
TRACE	
DIM	
PEEK-POKE	
SYS	
NULL	
STOP-END	

<b>Kapitel 5 COMAL PAKKER</b> .....	149
Hvad er en pakke? .....	149

Oversigt over pakker .....	149
Pakken ENGLISH .....	150
Pakken DANSK .....	150
Grafik med COMAL .....	150
Grafik oversigt .....	153
Uddybning .....	154
Sprites .....	167
Spriten forstørres .....	169
To sprites stødder sammen .....	169
Lagr en tegning .....	170
Sprites sammen med anden grafik .....	170
Sprite tegnefilm .....	171
En sprite i flere farver .....	173
Sprite oversigt .....	175
Lyd og musik .....	184
Uddybning .....	191
Pakker til brug sammen med kontrolportene .....	197
Paddles .....	197
Joysticks .....	199
Lyspen .....	201
Oversigt over pakken LIGHTPEN (lyspen) .....	204
Pakken SYSTEM .....	206
Pakken FONT .....	213
Uddybning .....	216

<b>Kapitel 6 COMAL FILER</b> .....	219
Hvad er en fil? .....	219
Gem og hent programmer og procedurer .....	220
Sekventielle filer .....	223
Filer med direkte tilgang et varekartotek .....	232
Flytning af en sekventiel fil rub og stub .....	236
Filtyper .....	237
Filer og skærm, tastatur og diskettstation .....	238
Datasetten og filer .....	239
Brug af 1520 Printer-Plotter .....	239
Resumé .....	240

<b>Kapitel 7 YDRE ENHEDER</b> .....	243
Indledning .....	243
RS-232C forbindelsesled .....	243
IEEE kapsler .....	247
Parallelstikket .....	247
Filoverførsel mellem computere .....	252
Kabelforbindelsen .....	253
Overførsel af programfiler .....	253

Send programmer fra C64 til Partner .....	254
Send programmer fra Partner til C64 .....	255
Overførsel af sekventielle ASCII filer .....	255
Kontrolportene .....	258
Resumé .....	261

<b>Kapitel 8 COMAL OG MASKINSPROG</b> .....	263
Hvad er maskinsprog? .....	263
Moduler .....	264
Pakker .....	265
Procedurer og funktioner .....	265
Signaler .....	266
Hvordan er hukommelsen organiseret? .....	266
Lagerstyring .....	268
Opbygning af moduler .....	269
Parameteroverførsel .....	272
Hvor kan modulets variabler anbringes? .....	275
Signalrutiner .....	275
Fejlgivning .....	277
Pakkeeksempel .....	278

<b>Appendiks A COMMODEORE 64 TEGNKODER</b> .....	285
--	-----

<b>Appendiks B FARVER</b> .....	289
---------------------------------	-----

<b>Appendiks C TALBEHANDLING MED COMAL</b> .....	291
--	-----

<b>Appendiks D TASTATUR OG SKÆRMREDIGERING</b> .....	295
--	-----

<b>Appendiks E TEKSTBEHANDLING MED COMAL</b> .....	299
--	-----

<b>Appendiks F COMAL FEJLNUMRE OG FEJLTEKSTER</b> .....	303
---	-----

<b>Appendiks G BRUGERKOMMENTARER OG RETTELSE</b> ..	317
---	-----

<b>Appendiks H COMAL PROGRAMEKSEMPLER</b> .....	319
---	-----

<b>Stikordsregister</b> .....	335
-------------------------------	-----





# FORORD

## COMAL 80

COMAL 80 er computersproget for alle, der gerne hurtigt vil lære at programmere og for alle, der gerne vil skrive større og bedre computerprogrammer eller undervise i, hvordan man skriver programmer. Det indeholder de fleste af de gode egenskaber fra både Basic, Logo og Pascal. Blandt de mange muligheder, COMAL 80 rummer, er:

- \* COMAL (COMMon Algorithmic Language) er en udvidelse af Basic med mange af Pascals stærkeste instruktioner.
- \* COMAL 80 er nemt at gå til, som Basic og Logo.
- \* COMAL 80 har de samme brugervenlige omgivelser som Basic, og mange COMAL 80 ordrer vil være bekendte for den tidligere Basic programmør.
- \* COMAL 80 til Commodore 64 rummer den ligefremme turtle grafik, som lanceredes i Logo sproget.
- \* COMAL 80 yder vejledende hjælp ved fejlindtastninger og indeholder strukturer til håndtering af fejl, som måtte opstå under udførelse af et program
- \* COMAL 80 indbyder til struktureret programmering med gentagelsessætninger som:

**REPEAT - UNTIL**  
**WHILE - DO - ENDWHILE**

sætninger til styring af betingelser og forgreninger som

**IF - THEN - ELIF - ELSE - ENDIF**  
**CASE - OF - WHEN - OTHERWISE - ENDCASE**

og værdifulde byggestene som **procedurer** og **funktioner**.

- \* COMAL 80 til Commodore 64 hjælper brugeren med at udnytte mange af de specielle egenskaber, som har gjort C64 til klassens mest populære mikrocomputer:

farvegrafik  
sprites  
lyd  
joystick

paddles  
lyspen  
og meget mere...

## COMALS OPRINDELSE

Grundlaget for COMAL skabtes i 1974 af Børge Christensen, som er lektor på Tønder Statsseminarium. Han syntes, at de studerende programmerede forfærdelig dårligt. Programmerne var vanskelige at læse. Det var svært at overskue programmeringsfejlene og et problem at vedligeholde programmerne. Børge Christensen rådførte sig med kolleger på Datalogisk Institut, Aarhus Universitet. Her anbefalede lektor Benedict Løfstedt ham at læse bogen **Systematic Programming** af Niklaus Wirth.

Der er sikkert mange, der genkender Niklaus Wirth (fra Schweiz' Statslige Teknologiske Institut i Zurich) som faderen til programmeringssproget Pascal. Inspireret af Wirth's klare formulering af principperne for struktureret programmering tog Børge Christensen igen kontakt med Benedict Løfstedt.

De var enige om, at Basic-sproget var bekvemt for brugeren under indtastning af et nyt program, idet Basic systemet gav megen hjælp. Sproget var nemt tilgængeligt for nybegynderen. Direkte udførelse af kommandoer fra tastaturet var muligt. Man behøvede hverken at definere variabelnavne forlods eller at oversætte programmet, før det kunne udføres. Disse egenskaber er gode i en undervisningssituation. På den anden side savnedes i Basic muligheden for at benytte lange, beskrivende variabelnavne, og sproget manglede helt Pascals elegante syntaks. Hvis Basic kunne udvides med disse egenskaber, ville det gøre det nemmere at skrive klarere, velstrukturerede programmer.

Sammen med Børge Christensen's studenter gik de i gang med at formulere, hvad man måtte kræve af et nyt COMAL programmeringssprog. Op gennem 70'erne kom andre til, og sproget modnedes med mikrocomputernes indførsel. I 1978 definerede man ved en standardiseringsaftale syntaksen for Comal 80 kernen. Fordi COMAL nu kommer tættest på at være det ideelle sprog for hjemme- og skolecomputere, er det valgt som undervisningssprog i Danmark, Sverige og Irland.

Denne dokumentation beskriver COMAL 80, version 2.0. udviklet af det danske firma UniComal ApS. Udover COMAL 80 kernen indeholder den ordrer, som er målrettede til udnyttelse af Commodore 64 computerens mange særlige egenskaber.

## COMAL OG COMMODORE

Gennem deres tillid til produktet har Commodores danske datterselskab spillet en fremtrædende rolle i forbindelse med COMAL sprogets voksende popularitet på verdensbasis. Den COMAL 80,

som du nu er ejer af til din Commodore 64, er udviklet af Jens Erik Jensen, Mogens Kjær, Helge Lassen og Lars Laursen fra firmaet UniComal ApS.

Når du har arbejdet dig gennem denne vejledning og selv skrevet dine første programmer, håber vi, at du vil være enige med os i, at COMAL er **sproget**.

## HVORDAN UDNYTTES DENNE VEJLEDNING

Når du gennemlæser Indholdsfortegnelsen, vil du se, at denne vejledning begynder med et kapitel om, hvordan computeren klargøres, og hvordan COMAL kapslen sættes i. Derefter følger en letlæselig, trinvis indføring i COMAL for begyndere. I **kapitel 3** regner vi med, at den første usikkerhed, som alle føler ved starten på et nyt emne, har fortaget sig.

**Kapitel 3** er et decideret undervisningskapitel med gennemgang af de mest almindelige COMAL vendinger. Her præsenteres du for værktøj til seriøs programmering, og du kan begynde at skrive dine egne COMAL programmer.

Enhver programmør har brug for et godt sted, hvor hun/han kan finde oplysning om den præcise betydning af det anvendte sprogs vigtigste faciliteter. Vi har forsøgt at give en sådan systematisk oversigt - med eksempler - i **kapitel 4**: COMAL oversigt. For de læsere, som ønsker endnu fyldigere kendskab til den præcise COMAL 80 syntaks, kan vi anbefale at anskaffe opslagsværket:

Len Lindsay, COMAL HANDBOOK, 1984  
Reston Publishing, 11480 Sunset Hills Road  
A Prentice-Hall Company  
Reston, VA 22090 USA (703) 437-8900

på dansk:

Len Lindsay, COMAL HÅNDBOGEN, 1985  
teknisk forlag A/S  
Skelbækgade 4, 1717 København V

En af de egenskaber, som gør COMAL til Commodore 64 uhyre nyttig, er pakke-begrebet. For at sikre sig stor udbredelse af COMAL sproget til mange forskellige computerfabrikater, er man som tidligere nævnt enedes om en standardiseret kerne. COMAL kapslen indeholder imidlertid en udvidelse af kernen med et bibliotek af 'pakker' med specielle procedurer og funktioner, som sætter programmøren i stand til at udnytte de særlige egenskaber ved en Commodore 64. Man kan f.eks. aktivere en pakke til brug ved grafiske tegninger med høj opløsning. Turtle grafik er også indbygget. Ydre enheder som joystick og lyspen udnyttes bekvemt ved aktivering af

en aktuel pakke. En anden pakke indeholder procedurer til omkonfigurering af computerens tastatur, skærmens farvesammensætning m.m. Hvordan man udnytter disse pakker, behandles i **kapitel 5**.

**Kapitel 6** giver yderligere information om fil-håndtering i COMAL. Dette kapitel vil være specielt nyttigt for de mange brugere, som påtænker at udnytte COMAL 80 til at skrive programmer til regnskabsføring, fakturering, kartotek og andre programmer, der kræver fil-håndtering. Her har man brug for COMALs avancerede kommunikation med en Commodore diskettestation.

I **kapitel 7** behandles tilslutninger af ydre enheder til Commodore 64. Det omfatter brugen af kontrol portene samt brugen af RS232 og parallel grænseflade samt tilslutningen af andre kapsler (f.eks. IEEE kapslen). Dette sidste emne kan have særlig interesse for de brugere, som får lyst til at udvikle deres egne nøglefærdige systemer, baseret på Commodore 64.

De af jer med 16 fingre kunne måske få lyst til at krybe ind i COMAL og lære om, hvordan man fra COMAL udnytter Commodore 64 lageret og skriver sine egne maskinkodeprogrammer. Læs **kapitel 8** for at lære mere om, hvordan det gøres.

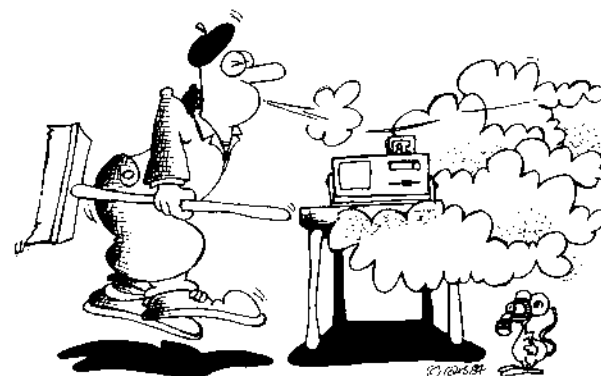
Denne vejledning afsluttes med en samling appendices med yderligere oplysninger. De drejer sig bl.a. om Commodore ASCII tegn værdier, farver og farvekoder, tal- og tekstbehandling med COMAL, brugen af tastatur og COMAL skærmredigering, samt fejlmeddelelser. Der er også et appendiks med nyttige programmer og procedurer, plus brugervejledning til nogle af programmerne på demonstrationsdisketten/kassetten.

Sidst, men ikke mindre vigtigt, er der et fyldigt stikordsregister. Det kan være en god hjælp, når du har brug for specifikke oplysninger.

Leo Højsholt-Poulsen & Frank Bason  
Silkeborg, september 1985

## Kapitel 1 -

# OPSTILLING AF COMPUTEREN



### COMPUTEREN OG TILBEHØR

Følgende udstyr er nødvendigt for at kunne udnytte kapslen med COMAL 80 sproget:

- \* En Commodore 64 (eller den transportable SX-64)
- \* En kapsel med COMAL 80 sproget
- \* Et fjernsyn eller en monitor (farve eller sort/hvid)

Man kan nemt udføre COMAL programmer på Commodore 64 uden et ydre lager i form af diskette eller kassettebånd. Større programmer vil man imidlertid gerne være i stand til at gemme til senere brug. Til det benyttes:

- \* En Commodore Datassette båndoptager, eller
- \* En Commodore 1541 diskettestation.

Det kan også være nyttigt, men ikke nødvendigt, med:

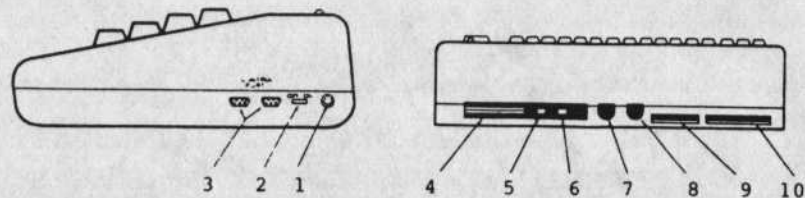
- \* En Commodore MPS 801/MPS 802 printer eller lignende
- \* En ekstra Commodore 1541 diskettestation

Ved større programmer er en printer god at have. Til seriøs program-



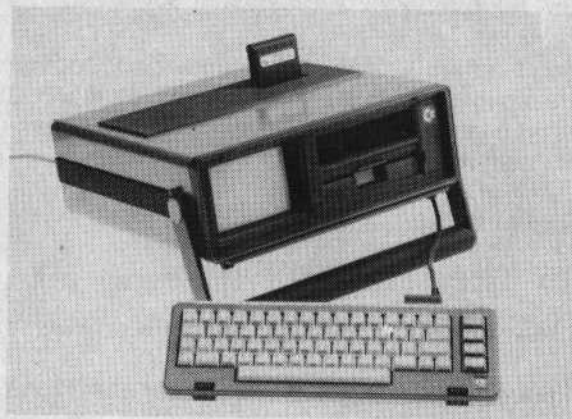
mering har man brug for udskrift af programmer og data. Det er ikke absolut nødvendigt med en ekstra diskettestation, men hvis man påtænker at bruge sin computer intensivt, kan den være praktisk til nemmere kopiering af programmer og filer.

Figur 1.1 viser stikforbindelserne på bagsiden og højresiden af Commodore 64. Brug diagrammet til at se, hvor udstyret skal tilsluttes.



**Figur 1.1:** Tilbehør og ydre udstyr tilsluttes Commodore 64 via forbindelserne på bagsiden og højresiden af computeren. (1) tilslutning for spændingsforsyningen, (2) tænd/sluk, (3) kontrolporte, (4) kapsel port, (5) TV finjustering, (6) UHF (TV) udgang, (7) monitor tilslutning, (8) serielporsten, (9) Datassette tilslutning, (10) brugerporten (I/O port).

COMAL kapslen kan også bruges på SX-64, som er den transportable version af Commodore 64 computeren. SX-64 er vist på Figur 1.2. I denne version er indbygget både en diskettestation og en farvemonitor. Hvis du har en COMAL kapsel og en SX-64, kan du springe frem til afsnittet om montering af COMAL kapslen.



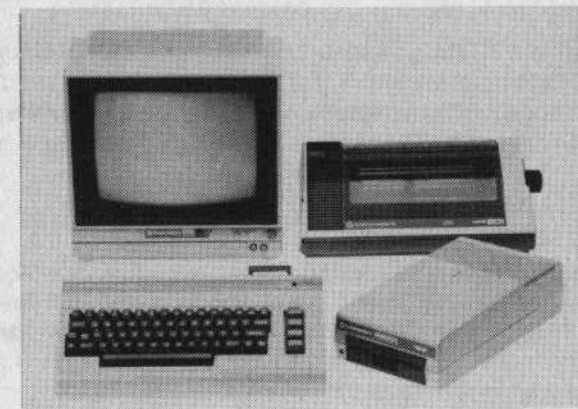
**Figur 1.2:** Den transportable Commodore SX-64 computer er kompatibel med Commodore 64. Den har sin egen indbyggede monitor og diskettestation men ingen Datassette grænseflade.

Hvis du har en 1541 diskettestation, så forbind den til Commodore 64 via det 6-benede stik på bagpanelet. Samme stik (serielporsten) kan benyttes til en printer.

Hvis du har både en printer og en diskettestation, forbindes printereren til det andet stik på bagsiden af diskettestationen. Det er ligegyldigt, hvilken af de to stik på diskettestationens bagside du forbinder til computeren, og hvilken du forbinder med kablet til printereren.

Hvis du har en Datassette, d.v.s en Commodore kassettebåndoptager, forbindes den til computeren via det 12 polede kantstik (lige ved siden af det bredere parallelstik, se figur 1.1). En almindelig kassettebåndoptager kan ikke bruges.

Der findes uddybende oplysninger om brugen af disse enheder i vejledningen til din Commodore 64, og vejledningerne til diskettestation, datassette og printer. En typisk opstilling er vist på figur 1.3. Systemet omfatter en enkelt diskettestation, en matrixprinter og et transportabelt TV som skærm.



**Figur 1.3:** En god opstilling omfatter en Commodore 64 computer med tilsluttet farvefjernsyn eller farvemonitor, samt diskettestation og printer.

Lad være med at tænde for noget som helst endnu. Du skal først have monteret COMAL kapslen!

### MONTERING AF COMAL KAPSLER

Kapslen med COMAL 80 sproget er vist på figur 1.4. Den skal monteres i det særlige modulstik til kapsler på bagsiden af Commodore 64 computeren. Hvis du har en SX-64, findes modulstikket til højre på computerens overside.



**Figur 1.4:** COMAL 80 kapslen forøger Commodore 64's styrke uden at optage yderligere lagerplads. Kapslen passer ind i stikket på bagsiden af Commodore 64 (På SX-64 er kapselstikket anbragt på oversiden).

Kig nøje på din COMAL kapsel. På kapslens FORSIDE er der en FORDYBNING med en mærkat. Denne forside skal vende OPAD, når du anbringer COMAL kapslen i stikket på bagpanelet af Commodore 64 (og vende FREMAD, når kapslen anbringes i stikket på SX-64).

---

ADVARSEL: Sæt aldrig kapslen i din Commodore 64 eller SX-64 (og fjern den aldrig), mens computeren og tilsluttet udstyr er tændt. Sluk for strømmen til computeren, mens du sætter kapslen i eller fjerner den.

---

Når du har forsikret dig om, at kapslens kantstik står lige ud for de rigtige forbindelser i computerens stik, skubbes kapslen helt i med en blidt vrikkende bevægelse.

#### TILSLUTNING AF TV ELLER MONITOR

På Commodore 64 er der to udgange til billede og lyd:

- \* Udgang til farvemonitor (lyd, lys og sammensat video)
- \* Udgang med moduleret signal til farve TV (UHF kanal 36)

Udgangsstikkene er vist på figur 1.1.

Da SX-64 har sin egen indbyggede farvemonitor, omhandler det følgende kun Commodore 64. Hvis du anvender en SX-64, kan du

springe frem til afsnittet om, hvordan man kører demonstrationsprogrammet.

En farvemonitor giver det skarpeste billede fra din Commodore 64. Hvis du har en Commodore monitor, skal den ene ende af kablet sluttes til det 8-polede DIN stik på computerens bagside. Slut stikkene i den anden ende af kablet til de 3 runde stik på monitorens bagside. Hvis du bruger en anden type monitor, vil forhandleren kunne hjælpe dig med at finde et egnet kabel.

Der følger et tilslutningskabel til TV med ved købet af en Commodore 64. Hvis du benytter et farvefjernsyn eller et sort/hvidt fjernsyn, forbindes den ene ende af kablet til fjernsynets almindelige antenneindgang og den anden ende sluttes til TV-stikket på computerens bagside. Kablet kan kun anbringes på en måde.

Du mangler nu kun at slutte strømforsyningen til apparaturet. Computeren forsynes med strøm fra den særlige omformer, som omsætter netspændingen til en lav jævnspænding. Kablet fra strømforsyningen tilsluttes på højre side af Commodore 64.

Når alle forbindelser er eftersat, og beskyttelsen fra forsendelsen er fjernet fra en eventuel diskteststation og printer, kan du tænde for strømmen til apparaturet. Tænd selve computeren sidst!

Indstil fjernsynets UHF kanalvælger, så billedet er skarpest. (Der er også en finindstillings skrue ved siden af antenneudgangen på computerens bagpanel. Hvis det lykkes at få et godt billede ved hjælp af indstillingen på TV-apparatet, er det unødvendigt at indstille med denne skrue.)

Hvis alt er forløbet som planlagt, skulle skærbilledet vise:

\$\$\$ Commodore-64 COMAL 80 rev 2.01 \$\$\$  
(C) 1984 by UniComal & Commodore  
30714 bytes free

med markøren blinkende 3 linier under teksten. Hvis der er skruet op for lyden på fjernsynet eller højttaleren, vil man også have hørt systemets COMAL-klokke, som fortæller, at COMAL er klar til at gå i gang. Til at begynde med kan man, hvis man har en farveskærm, prøve at trykke på <v> mens <CTRL>-tasten holdes nede, for at ændre skærmens farver til en behagelig blå med hvid markør og tekst. Hvis du bruger en sort/hvid skærm, så prøv <w>, mens <CTRL>-tasten holdes nede, for at få en grå baggrund med sort tekst.

Hvis beskeden ikke viser sig, kan det have mange forskellige årsager. Prøv først at slukke for strømforsyningen til computeren og vrik lidt med COMAL kapslen for måske at få den helt på plads. Hvis det stadigvæk ikke lykkes at fremtvinge det ønskede skærbillede, når computeren igen tændes, kan du prøve at gennemgå opstillingspro-

ceduren endnu en gang. Yderligere hjælp findes i Commodore 64 Brugervejledning eller hos din Commodore forhandler.

### LIDT OM TASTATURET

Hvis du ikke er bekendt med Commodore 64 tastaturet, så skriv blot et eller andet for at vænne dig til det.

Prøv også at skrive, mens <SHIFT> tasten holdes nede, f.eks. ved brug af <SHIFT LOCK> tasten.

Hvis du laver en indtastningsfejl, sørg så for, at <SHIFT LOCK> er slået fra og tryk på "indsæt eller slet" tasten <INST/DEL> øverst til højre på tastaturet. Herved slettes tegnet umiddelbart til venstre for markøren.

Du kan også flytte markøren rundt på skærmen ved hjælp af tastene med markør-pilene <CRSR> nederst til højre på tastaturet. Retningen øverst på tasten fremkommer ved at trykke på markør-tasten, mens <SHIFT> holdes nede.

Anbring markøren over det andet bogstav i et eller andet ord på skærmen. Tast nu <INST/DEL>, mens <SHIFT> tasten holdes nede. Bogstaverne vil herved skubbes fra hinanden, og der vil fremkomme et mellemrum på markørens plads. Herved skabes plads til at indsætte manglende tegn.

Afprøv også <CLR/HOME> tasten og <CLR/HOME> samtidig med <SHIFT> for at se, hvordan den virker.

Prøv at trykke på Commodore tasten <C=> (nederst til venstre på tastaturet), mens <SHIFT> holdes nede. (Forkortet skrivemåde: <SHIFT-C=>). Gør det flere gange. Derved skifter man frem og tilbage mellem de to tegnsætmuligheder: små/store bogstaver og store bogstaver/semigrafiske tegn. Før du går videre til næste emne, bør du sikre dig, at tegnsættet er i tilstand **små/store bogstaver**.

Kontrol:	Tryk på tasterne	<A>	<S>	<D>
	Computeren skriver	a	s	d
	Tryk igen på de			
	samme taster, mens			
	<SHIFT> holdes nede			
	Computeren skriver	A	S	D

Hvis computeren ikke har reageret korrekt, tryk da på <SHIFT-C=> for at skifte tegntilstand.

Hermed er det nok om tastatur for denne gang. Det er tilstrækkelig baggrund for at gå videre i denne COMAL begyndervejledning. Du vil komme til at lære mere om andre muligheder hen ad vejen. En fuldstændig beskrivelse af tastaturet og COMAL skærmredigering findes i appendiks D.

### TAL DANSK MED DIN COMPUTER

Selv om computerprogrammer skrives med brug af engelske nøgleord, kan du få computeren til at tale dansk til dig. Skriv:

**use dansk**

efterfulgt af et tryk på <RETURN> tasten.

Herefter vil alle COMAL fejlmeddelelser fra systemet være på dansk.

### HENT DEMONSTRATIONSPROGRAMMET

Blandt computerfolket benyttes vendingen "at **køre** et program". Det betyder, at programmet bringes til udførelse. Denne vending er så indgroet, at vi fremover vil gøre brug af den.

Hvis du har en Datasette eller en diskettstation, som er korrekt forbundet til computeren, er du nu klar til at køre programmer. Gå frem efter det afsnit, som passer på din situation:

### Brug af Datasette:

Sæt kassettebåndet, som kan fås med ved købet af COMAL kapslen, i båndoptageren og skriv:

**load "cs:demoprogram"**

og tryk på <RETURN> tasten.

Bemærk, at hvis man regner med udelukkende at benytte Datasette båndenheden, kan den gøres til systemets foretrukne enhed med kommandoen

**unit "cs:"** efterfulgt af <RETURN>

I dette tilfælde er det ikke nødvendigt med **cs:** foran programnavnet. Det vil være tilstrækkeligt med:

**load "demoprogram"**

---

Bemærk, at ordet RETURN i spidse parenteser <> betyder, at man skal trykke på den pågældende taste. Man skal ikke stave til ordet med tegn fra tastaturet.

---



Commodore 64 svarer med

**press play on tape**

Vær sikker på, at båndet er spolet tilbage til begyndelsen og start båndet ved at trykke på <PLAY> på båndoptageren.

Computeren svarer

**ok**

**searching for demoprogram**

hvorefter skærmen hurtigt bliver blank. Efter nogle få sekunders forløb vil programmet være fundet, og beskeden

**found demoprogram**

vises på skærmen. Efter en kort pause startes indlæsningen af programmet, som tager omkring et minut. Under indlæsningen vil skærmen være blank, men blinke nogle gange. (Commodore tangenten <C=> kan bruges til at afbryde indlæsningen.) Når programmet er færdigindlæst, vender det normale skærm billede tilbage, og markøren blinker for at tilkendegive, at en kopi af programmet befinder sig i computerens arbejdslager. Klar til at bringes til udførelse.

Hvis der er besvær med at indlæse demonstrationsprogrammet, kan man f.eks. prøve følgende:

- \* Sluk for strømforsyningen til computeren og datassetten, og se igen efter, om båndoptageren er forbundet rigtigt. Er kablet helt og sat helt ind i stikkene?
- \* Har du taget det rigtige kassettebånd og spolet det helt tilbage (al båndet skal være på spolen til venstre).
- \* Når du har checket ovenstående, tændes igen for strømforsyningen. Gentag derefter indlæsningsproceduren.
- \* Er der stadigvæk problemer, må du søge assistance hos din forhandler.

Diskette indhold er beskyttet, når indhakket er dækket til.

Skrive-  
beskyttelses-  
indhak



**Figur 1.5:** Behandl disketten varsomt. Lågen til diskettestationen åbnes med et lille tryk. Sæt disketten ind i sprækken som vist, og skub den langsomt hele vejen ind. Når disketten er på plads, lukkes lågen i, til den giver et klik.

### Brug af diskettestation

Sæt den Demonstrationsdiskette, som fulgte med ved købet af COMAL kapslen, i diskettestationen. Diskettens **etikette** skal vende **opad** og ud **mod** dig, når disketten skubbes ind (se figur 1.5).

Skriv:

**load "demoprogram"**

og tryk på <RETURN> tasten. Herved vil en kopi af programmet overføres fra disketten til computerens arbejdslager. Aktivitetslampe på stationen vil lyse, mens programmet overføres. Det tager nogle få sekunder.

### KØR DEMONSTRATIONSPROGRAMMET

Hvad du end har gjort for at hente demonstrationsprogrammet ind i computerens arbejdslager, kan du nu taste **run** og trykke på <RETURN>. Sæt dig så tilbage i stolen, slap af og nyd demonstrationen. Sørg for, at der er skruet lidt op for lyden. Hvis farverne er dårlige, benyttes lejligheden til en finjustering af displayet.

- \* Programmet stoppes ved tryk på <RUN/STOP> tasten.
- \* Tag Demonstrationsdisketten ud og opbevar den på et sikkert sted.

Hvis du følger vejledningen i de følgende kapitler, vil du sikkert snart være i stand til selv at kombinere styrken fra COMAL 80 med Commodore 64'eres mange muligheder - f.eks. højopløsningsgrafik, sprites og lyd.

### KLARGØRING AF EN NY DISKETTE

Før vi fortsætter med den indledende gennemgang i kapitel 2, gøres

en ny diskette klar til at gemme programmer på. Det kaldes at **formattere** disketten.

Datassettebrugere behøver ikke formattere kassettebånd. Det er ikke nødvendigt.

- \* Tag en ubrugt diskette (eller én, som må slettes). Vær sikker på, at **skrivebeskyttelses indhakket** til højre for mærkaten ikke er dækket (hvis det firkantede indhak dækkes med et stykke tape eller lignende, er disketten beskyttet mod overskrivning).
- \* Sæt disketten i diskteststationen (figur 1.5) og luk lågen med et klik.
- \* Skriv nu:  
`pass "n0:<diskettenavn>,<xx>"`

Når du trykker på <RETURN> vil disksystemet arbejde på at formatere disketten i omkring 2 minutter. Aktivitetslampen på diskteststationen slukkes, når formatteringen er slut. Disketten er nu klar til at gemme programmer og datafiler.

Nogle få forklarende bemærkninger om kommandoen **PASS**: den fortæller COMALsystemet, at den efterfølgende tekst skal videregendes til diskteststationens operativsystem. Bogstavet "n" står for en **ny** diskette, og 0 betyder, at processen skal foregå på den første diskteststation (hvis man har mere end en). Vælg selv diskettenavn (op til 16 tegn langt). Dette navn vil være overskrift, når du forlanger et katalog over diskettens indhold (mere om dette i kapitel 2). De to tegn efter kommaet **xx** er to vilkårlige tegn, som skal tjene til at identificere disketten.

F.eks.: `pass "n0:COMAL diskette,c1"`

## RESUMÉ

Udstyret skulle nu være stillet op og klar til brug. COMAL kapslen er monteret, apparaturet tændt, og du har vænnet dig til tastaturet. Hvis du har en datassette eller en diskteststation, har du også prøvet at indlæse et demonstrationsprogram og kørt det for at efterprøve dit system.

Programmet har forhåbentlig givet dig en forsmag på de muligheder, der er i COMAL. Her til sidst har du måske formatteret en ny diskette til at gemme de programmer på, som du skriver under gennemgangen af de følgende kapitler.

## Kapitel 2 -

# LAD OS SÅ KOMME I GANG!



## HVORFOR LÆRE AT PROGRAMMERE?

Computeren er et redskab til håndtering af informationer. Hvis computeren programmeres korrekt, kan den foretage beregninger, bearbejde tekster, sortere oplysninger, opsamle data, styre maskiner, skabe billeder, lave lyde og meget mere. Computerens hjerte er den nu velkendte komponent *mikroprocessoren*. Hvis mikroprocessoren forbindes med tilstrækkeligt arbejdslager og et middel til ind- og udlæsning af informationer, har vi en *mikrocomputer*.

Den elementære informationsenhed, som din Commodore 64 arbejder med kaldes en *byte* eller *oktet*. Den består af otte *binære cifre* (*bits*) som kan være enten 0 eller 1. De binære cifre repræsenteres i computeren ved spændinger på 0 eller 5 volt. De grundlæggende operationer, som mikrocomputeren udfører på de enkelte oktetter (*bytes*), er meget enkle. Blot det at lægge to tal sammen, f.eks. 2543 og 9320, kræver udførelsen af hundreder af simple operationer.

Alligevel gennemføres beregningen på nogle tusindte dele af et sekund, fordi hver operation kun tager en milliontedel af et sekund!

Når man programmerer en computer, er det muligt (men bestemt ikke nødvendigt!) at arbejde med de binære tal (to-tals systemet), som processoren bruger. Commodore 64 bruger en **6510 chip** som er i familie med den populære **6502** mikroprocessor. Man kan bruge *assemblersprog*, hvis man ønsker at programmere den direkte.

For at gøre livet lettere har programmører udviklet *højniveausprog*, som gør det muligt at bruge meget enkle ordrer til udførelse af en række grundlæggende processor operationer. En sætning som:

**print 2543 + 9320**

er et eksempel på en ordre i højniveausprog.

Denne sætning kan man forestille sig som en "*procedure*" med to startoplysninger. Sætningen udvirker, at de to tal bliver lagt sammen og resultatet udskrives f.eks. på en skærm.

Et ideelt datamatsprog gør det muligt for programmøren at gruppere flere instruktioner sammen, så de udretter mere komplekse opgaver, og at give dem et fælles navn. Det ville f.eks. være bekvemt at have en ordre som

**rente(12535,8)**

der kunne beregne det afkast, som en investering på 12535 kr. ville give over en otteårig periode. Mens alle brugere af et computersprog vil have brug for at kunne lægge tal sammen, vil ikke alle have brug for netop denne procedure. Så det ideelle sprog må indeholde et stort antal nyttige standardprocedurer og samtidigt gøre det let for programmøren at konstruere sine egne specielle procedurer.

COMAL er et sådant sprog. Det er et *procedureorienteret* sprog, der indeholder mange klare og nyttige grundlæggende ordrer, så man kan skabe sine egne procedurer. Disse procedurer er måske igen så anvendelige, at de selv kan indgå igen i andre programmer eller i andre procedurer, som udretter større opgaver. COMAL's hele opbygning gør det let og bekvemt at foretage sådanne operationer. Når brugeren har lært at beherske COMAL sproget, vil det være et effektivt værktøj til løsning af en lang række problemer.

Det er en oplevelse at lære at bruge et effektivt programmeringsprog. Oplevelser kan være sjove og spændende, men ingen ægte oplevelser er uden udfordringer og faldgruber. Evnen til at skrive programmer udvikles kun gennem øvelse, udholdenhed, nysgerrighed og tålmodighed. Når man først er begyndt på oplevelsen, må man være forberedt på, at der også kommer perioder med prøvelser, før man bliver en habil programmør.

Programmering er ikke kun til løsning af seriøse professionelle opgaver. Det kan også være morsomt! Spørg blot den programmør, der har siddet oppe til langt ud på natten for at få et spilleprogram til at virke. Den tilfredsstillelse, det er at få et program til at blive levende efter omhyggeligt at have bygget det op af dets bestanddele, ligner andre meget kreative aktiviteter. (Men spørg ikke program-

møren, før han/hun har fundet den sidste fejl og fået programmet til at virke!)

Programmering kan bruges til så mange formål, at en udtømmende liste er umulig. Her er blot nogle få; mange flere kan naturligvis tilføjes. Korrekt programmeret kan computeren:

- \* spille et spil med grafik, som børn kan lære af
- \* bistå ved musikundervisning ved at vise og spille noder
- \* opstille et regnskab og sammenligne det med familiens budget
- \* føre salgsstatistik for et mindre firma
- \* registrere og vise vejrdata
- \* foretage målinger i et laboratorium eller ved et samleband
- \* udfylde selvangivelsen og skrive den ud
- \* hjælpe til ved design og administration af et byggeprojekt
- \* beregne varmetab fra en bygning
- \* være et motiverende værktøj i undervisningen

En stor del af den programmering til mikrocomputere, der finder sted i dag, drejer sig om *spil*. Lige siden computerers fremkomst har programmører holdt af at "lege". Dengang da computertid kostede et betragteligt beløb, var det en luksus, få kunne tillade sig. I dag koster tid ved mikrocomputere kun nogle ører om dagen, så spilprogrammer er myldret frem som aldrig før. Hvis du gerne vil spille computerspil på din computer eller selv skrive nogle, så velkommen til COMAL. Det er et hurtigt sprog til styring af computerens farvegrafik, sprites og lydeffekter. Mulighederne for at lave spilprogrammer er endeløse.

*Simulationer* er måske de mest fascinerende af de mange mulige programtyper. Man kan blive pilot i et kampfly fra Første Verdenskrig, skarpt forfulgt af fjendtlige fly. Lav om på programmets parametre, og man er nu ved rorpinden i en 747 jumbojet, der lægger an til landing i Paris, London eller New York. Eller simuler Charles Lindbergs fly, Spirit of Saint Louis, på den første non-stop flyvetur fra New York til Paris. Selv en tur med rumfærger eller Concorde kan simuleres ved brug af en mikrocomputer. Med farvegrafik og et *joy-stick* bliver sådanne simulationer forbavsende realistiske.

Men simulationer er meget mere end blot spil. De kan være effektive redskaber til indlæring - både for studerende og for professionelle. Med simulationsprogrammer kan man bl.a. undersøge:



- \* en virksomheds økonomiske dispositioner og status
- \* hvordan et solfangersystem fungerer
- \* hvordan en atomreaktor fungerer
- \* ladede atompartiklers bevægelser i et elektrisk og magnetisk felt
- \* en satellits kredsløb om jorden
- \* eller en rakets bane

Igen gælder det, at mulighederne er næsten ubegrænsede for den, som giver sig i kast med at lære at programmere. Mulighederne er faktisk kun begrænset af brugerens fantasi og evne til at lære at bruge værktøjet: Commodore 64 og programmeringssproget COMAL. Lad os nu lære at bruge dette værktøj!

### DIREKTE UDFØRELSE AF COMAL KOMMANDOER

Computeren skal nu have COMAL kapslen installeret, og computeren skal tændes. Når man gør det, bør følgende melding komme frem på skærmen:

```
$$$ Commodore-64 COMAL 80 rev 2.01 $$$
(C) 1984 by UniComal & Commodore
30714 bytes free.
```

Når denne melding står på skærmen, kan man gå i gang .....

---

HVIS DU TASTER FORKERT: Du kan slette tegnet lige til venstre for markøren ved at trykke på <INST/DEL> tasten øverst til højre på tastaturet. (Tasten <SHIFT/LOCK> må ikke være nede, når man gør det!) Appendiks D indeholder en komplet beskrivelse af brugen af tastaturet og en oversigt over COMAL's mange redigeringsfaciliteter. Se specielt <CTRL-A>.

---

Den letteste måde at begynde at lære COMAL på er at indtaste nogle direkte kommandoer fra tastaturet. Prøv at skrive:

```
print "goddag"
```

Efter et tryk på <RETURN> skulle ordet **goddag** nu stå skrevet på skærmens næste linie.

---

Det er vigtigt at forstå, at computeren først *udfører* kommandoen, når der trykkes på <RETURN>. RETURN er egentlig en ordre til computeren om at behandle den pågældende linie.

---

Læg mærke til, at ordrer kan indtastes enten med store eller med små bogstaver. (Man skifter frem og tilbage på skærmen mellem store bogstaver/grafik og små bogstaver/store bogstaver ved at trykke på <C=>, mens <SHIFT> holdes nede.)

---

I denne vejledning antager vi, medmindre andet skrives, at tilstanden **små bogstaver/store bogstaver** er blevet valgt.

---

Man kan også udføre beregninger ved brug af ordrer fra tastaturet. Prøv følgende kommando, men vær omhyggelig med **ikke** at trykke på <SHIFT> tasten, mens der trykkes på + tegnet:

```
print 217+305
```

Efter at have tastet <RETURN> vil du se, at kommandoen PRINT bevirker, at computeren skriver tallet 522 på næste linie.

Man kan også blande tekst og tal sammen i en PRINT kommando som i følgende eksempel:

```
print "sum =",217+305
```

Efter et tryk på <RETURN> er kommandoen indlæst, og computeren vil skrive:

```
sum =522
```

Læg mærke til, at hvis der ikke gives anden instruktion, vil teksten og tallet ikke blive adskilt af mellemrum, ved udskriften. Det kan ændres ved brug af et **semikolon** ;. Hvis et semikolon anvendes til adskillelse, vil et mellemrum blive trykt til højre for hvert tekstafsnit eller tal.

Man kan også bestemme placeringen af tekst og tal på skærmen ved brug af ZONE kommandoen. Tast:

```
zone 10
```

Vi vil nu gentage kommandoen fra tidligere. Prøv dette arbejdsbe-

sparende kneb: Tryk på <SHIFT> tasten og tryk samtidig markør op-ned tasten (<CRSR> lige under <RETURN>). Flyt markøren opad, indtil den blinker på linien:

```
print "sum =",217+305
```

Slip <SHIFT> tasten og tryk <RETURN> igen. Kommandoen udføres igen, men denne gang vil der være 10 tegn mellem tekstens begyndelse og tallets begyndelse! ZONE ordren bruges til at specificere bredden af de søjler der trykkes ud i, når tekst og tal er adskilt med kommaer. Når der tændes for computeren, er **zone** sat til 0.

Det vil være hensigtsmæssigt at lave nogle øvelser med ZONE og PRINT kommandoerne, før der arbejdes videre med denne vejledning. Det gøres let ved at bruge markør tasterne til at køre op og ned på skærmen. Bemærk, at man ikke behøver at være i slutningen af linien på skærmen, når der tastes <RETURN> for at få kommandoen udført. Læg også mærke til, at hvis der er overflødig tekst på samme linie, fortolkes det sammen med den kommando, der ønskes udført, og resultatet bliver en fejlmelding. Man kan enten slette den overflødige tekst (<CTRL-K> vil slette resten af linien fra markørpositionen), eller man kan skrive kommandoen på en ledig linie for at undgå denne type fejl. Man kan også helt slette skærmen ved at udføre kommandoen PAGE eller ved at holde <SHIFT> tasten nede, samtidig med at der trykkes på <CLEAR/HOME> tasten.

COMAL har mange andre faciliteter til håndtering af tekst og tal. Vi vil gå meget mere i dybden med dem senere. Inden vi fortsætter med at skrive programmer, vil vi kort undersøge, hvordan man bruger skærmen til højopløsningsgrafik.

## ET HURTIGT KIK PÅ TURTLE GRAFIK

Commodore 64 computeren er omtrent parat til at udføre *turtle grafik*, så snart der er sat strøm på. Tryk blot på <f3> for at udvide COMAL med ordrene i *turtle grafikpakken*. Efter et tryk på <f3> vil ordene **USE turtle** komme frem på skærmen. Dernæst vil skærmens udseende ændres. En lille pilespidse vil komme til syne midt på skærmen, og ordene **USE turtle** vil nu være øverst på skærmen med markøren blinkende på næste linie. Nu har vi en *opdelt skærm* med fire linier synlig tekst øverst. Et tryk på <f1> vil bringe brugeren tilbage til *tekstskærmen*. Hvis vi trykker <f5> ned, ses *grafikskærmen*. Hele skærmen kan bruges til grafik, men man vil ikke kunne se kommandoerne, efterhånden som de tastes ind. Tryk nu igen <f3> for at komme tilbage til den delte skærm.

---

Læg mærke til, at COMAL sproget ved hjælp af USE ordren er blevet udvidet med et ekstra ordresæt, kaldet en *pakke*. Som vi skal se senere, er der mange andre pakker til rådighed i COMAL kapslen. Meget mere om pakker i kapitel 5!

---

For at fjerne de COMAL udvidelser, der blev aktiveret med USE ordren, skal man taste:

```
discard <RETURN>
```

---

Det vil fjerne ALLE pakker fra computerens aktuelle arbejdsområde. (Man kan ikke fjerne udvalgte pakker enkeltvis.) Ved at skrive **new** sletter man sit program og sætter samtidigt alle pakker ud af funktion.

---

Lad os se på, hvordan *pilespidsen*, som også kaldes *grafikmarkøren*, *pennen* eller *skildpadden*, kan flyttes rundt på skærmen og tegne. Vi vil i første omgang bruge direkte kommandoer, men senere i vejledningen vil vi skrive et helt program.

Turtle grafik kommandoer er så ligefremme, at man kan lære at bruge dem ved at prøve sig frem. Prøv at skrive:

```
forward(50) <RETURN>
right(90) <RETURN>
```

Tast de samme kommandoer igen. Herefter bør der være et halvfærdigt kvadrat på skærmen. Brug kommandoerne igen til at gøre kvadratet færdigt. Pilespidsen bør slutte med at pege opad igen.

Prøv nu følgende kommandoer og husk at trykke <RETURN> efter hver. Læg mærke til, hvad der sker med pilespidsen og tegningen:

```
penup
back(50)
pendown
forward(50)
```

Læg også mærke til, at hvis pilespidsen bringes for langt ud, vil den komme til syne på skærbilledets modsatte side.

Indtast **clearscreen** for at ryde skærmen. Bring pilespidsen tilbage til skærmens midte ved at taste **home**.

Prøv nu:

```
left(90)
forward(50)
setheading(45)
forward(70)
```

Hvad udretter hver kommando? Tast **clearscreen** (eller blot **cs**) for at slette skærmen. Lav selv nogle øvelser for at lære, hvordan pilespidsen flyttes rundt på skærmen. Det vil være værd at prøve følgende:

```
for side = 1 to 4 do forward(50);left(90)
```

Dette eksempel illustrerer et enestående træk ved COMAL: en *følge* af instruktioner adskilt med semikolon kan ofte udføres direkte fra tastaturet!

Som illustration af COMALs aktive hjælp under indtastning (hvis du ikke er stødt på den før), kan du forsøge at lave fejl i indtastningen af forrige kommando:

```
skriv: for      og tryk <RETURN>
Bemærk computerens svar.
skriv: for =    og tryk <RETURN>
Bemærk svaret.
skriv: for i=   og tryk <RETURN>
Bemærk svar. osv.
```

En anden hjælp som COMAL giver er, at fejlmeldinger ryddes fra skærmen, når du har rettet fejlen og tastet <RETURN>.

Læg mærke til, hvad disse kommandoer udretter:

```
hideturtle
showturtle
```

Hvis man har farveskærm til rådighed, kan man eksperimentere med følgende:

```
background(<tal>)
pencolor(<tal>)
```

hvor **tal** er en farvekode. Appendiks B indeholder en liste over *farvekoder*. Eksempelvis gør `pencolor(7)` pilespiden gul.

Følgende tabel viser turtle grafik ordrene med en forkortelse for hver, samt en kort beskrivelse. Med ordren **USE turtle** får man rådighed over alle disse ordrer såvel som alle øvrige ordrer i *grafikpakken*.

TURTLE ORDRE	KORT FORM	BESKRIVELSE
back(L)	bk(L)	flyt L enheder baglæns
forward(L)	fd(L)	flyt L enheder fremad
background(C)	bg(C)	sæt baggrundsfarve til C
clearscreen	cs	slet grafikskærmen
hideturtle	ht	skjul pilespiden
showturtle	st	vis pilespiden
pencolor(C)	pc(C)	sæt tegnefarven til C
pendown	pd	sænk pennen
penup	pu	løft pennen
left(D)	lt(D)	drej D grader til venstre
right(D)	rt(D)	drej D grader til højre
setheading(H)	seth(H)	pilespiden peger i retning H; H=0 er op, 90 er til højre osv.

Læg nøje mærke til disse ordrer. De vil blive brugt i de følgende programeksempler.

### HVAD ER ET PROGRAM?

For at en maskine eller en computer kan udrette en opgave, skal den "have at vide", hvordan den skal gøre det. Modsat et menneske, som baserer sine handlinger på færdigheder og erfaringer, skal maskinen modtage meget præcise instruktioner. Brugeren må ikke tage noget for givet. I praksis betyder det, at man skal skrive en liste af ordrer, som hver for sig tolkes af computeren, og som præcist beskriver, hvad der skal udføres.

Det ville være et meget møjsommeligt arbejde, hvis vi hver gang skulle give detaljer, f.eks. om addition af to tal. Det er naturligvis ikke nødvendigt. Når først computeren er blevet programmeret til forstå kommandoen **PRINT x + y**, hvor **x** og **y** er et hvilket som helst talpar, kan den lægge alle andre talpar sammen (inden for meget vide grænser - se appendiks C). Det samme gælder andre operationer, vi forventer computeren skal kunne udrette. Nogle af de mest anvendte er:

- \* addition, subtraktion, multiplikation og division af tal
- \* udskrivning af tal og tekst
- \* tegning af en linie fra et punkt til et andet
- \* foretage et valg mellem to muligheder

- \* gentage operationer et vist antal gange
- \* udvælge forskellige opgaver, når visse betingelser er opfyldte.

Disse operationer er defineret i et *computersprog* som er relativt let at bruge. COMAL er enestående, fordi det er et særlig klart, effektivt og fleksibelt sprog.

Lad os prøve at skrive et COMAL program for at illustrere nogle af disse ideer.

Lad os antage, at vi ønsker at tegne et kvadrat på computerens skærm. Selv uden noget forudgående kendskab til programmering kan vi i dagligdags sprog skrive en liste over de opgaver, der skal udrettes:

- \* Gør computeren parat til brug af skærmen til grafik.
- \* Beskriv hvor langt pilespiden skal flyttes og drejes for at tegne en kvadratside.
- \* Gentag ovenstående trin fire gange for at fuldende kvadratet.

Hvis vi pensler det lidt mere ud, kan vi udtrykke det ved at skrive følgende instruktioner. Det er vores mening at tegne et kvadrat på 75 "enheder" på hver side, startende midt på skærmen. Vi vil have kvadratets sider parallelle med skærmens sider:

- \* Vælg turtle grafiktilstanden.
- \* Flyt pennen 75 enheder frem og drej 90 grader til højre.
- \* Flyt igen 75 enheder frem og drej 90 grader til højre.
- \* Flyt 75 frem, og drej 90 til højre.
- \* Flyt 75 frem;drej 90 til højre.

Når alt dette er udrettet, bør vi have et kvadrat på skærmen med pennen tilbage på sin udgangsposition. Det er sædvanligvis god programmeringspraksis at lade pennen slutte i den samme *tilstand*, når instruktionssekvensen slutter, som da den begyndte. Det råd er specielt vigtigt at følge, når man begynder at skrive COMAL procedurer. Det gør det lettere, når man ønsker at bygge et program op ved brug af moduler eller byggeblokke, som skal arbejde sammen om at udføre en opgave.

Lad os se på, hvordan et egentligt COMAL program vil se ud. Læg mærke til, at det måske ikke umiddelbart er indlysende, hvorfor man gør visse ting. De fleste af disse mysterier vil blive opklaret, efterhånden som vi arbejder os gennem de mere indgående forklaringer!

Vær først sikker på, at der bruges *delt skærm* (tryk <f1> og derefter <f3>). Vær også sikker på, at der ikke ved et tilfælde er skabt

COMAL programlinier i computerens arbejdslager (tast **new** <RETURN>). Det vil nok være en god ide at rense skærmen og flytte markøren til øverste venstre side af skærmen. Tryk samtidig på <SHIFT> tasten og <CLR/HOME> for at gøre dette på én gang.

---

Hvis der er problemer med at få computeren i teksttilstanden med skærmen rensed, er der en helt sikker metode til at få styr på tingene. Tryk på <RUN/STOP> tasten og hold den nede, mens der trykkes på <RESTORE> tasten. Dette vil starte tingene op igen forfra, uden at programmet går tabt. Man kan selvfølgelig altid slukke for computeren, vente nogle sekunder og så tænde igen. Så bør man være tilbage i COMAL med det indledende budskab på skærmen, parat til at starte, men denne løsning vil slette programmet i arbejdslageret.

---

Når man indtaster et program, skal de forberedte instruktioner ikke udføres straks. Det sikres ved at anbringe et *linienummer* foran hver ordre. Linienummeret gør, at COMAL systemet opfatter ordren som en *programsætning* og ikke som en kommando, der skal udføres med det samme. Programlinierne lagres i arbejdslageret og udføres først, når der gives en **run** kommando. Det vil efterhånden blive klart, at linienumre ikke er vigtige i COMAL, undtagen som en hjælp, når man indlæser eller redigerer i et program. Linienumrene er bestemmende for, i hvilken rækkefølge programsætninger udføres, men faktisk kan man helt ignorere linienumre, når programmet er færdigskrevet, og linierne anbragt i korrekt rækkefølge.

Hvis man ønsker automatisk linienummerering, skal der blot trykkes <f4> (det opnås ved at holde <SHIFT> nede og trykke på <f3>). COMAL svarer med AUTO. Tryk <RETURN>, og den automatiske linienummerering vil være sat i gang.

Datamaten bør nu være parat til at modtage en instruktion på linie 10.

Det er oftest hensigtsmæssigt at nummerere instruktionerne med intervaller på 10, så der bliver plads til at indsætte instruktioner, der ønskes indføjet senere. I denne forbindelse er RENUM ordren meget nyttig (fås også ved at taste <f1>). RENUM sørger for omnummering af linienumre. Se kapitel 4.

---

For at slippe af med den automatiske nummerering eller for at ændre den, skal man blot taste <RUN/STOP> i stedet for at indlæse en ny linie. Hvis man så indtaster **auto** eller trykker <f4> igen, vil man være tilbage i den automatiske nummerering efter den linie, man forlod. Hvis man skriver **auto,5** <RETURN>, vil nummerin-

tervallet blive 5, og nummereringen fortsætter fra, hvor man var nået til. Hvis man skriver **auto 100,5**, vil nummereringen starte ved linie 100 med et interval på 5.

Hvis vi genkalder listen over ønskede handlinger, udtrykt på dansk, kan vi starte med de COMAL ordre, som skal bruges til at forberede skærmen til turtle grafik:

#### 0010 use turtle

Tryk <RETURN> efter hver ordrelinie (selv om det ofte er muligt at skrive flere ordre på samme linie, adskilt af ;, er det sædvanligvis klogt kun at skrive én ordre på hver linie). Efterhånden som programlinier indlæses, udskriver COMAL systemet selv næste linienummer og er parat til næste instruktion. Indtast som i følgende eksempel, så vi kan fortsætte programmet. Brug markørtasterne og <INST/DEL> efter behov til rettelse af indtastningsfejl. Forkortelser kan benyttes efter eget valg.

```
0020 splitscreen
0030 forward(75)
0040 right(90)
0050 forward(75)
0060 right(90)
0070 forward(75)
0080 right(90)
0090 forward(75)
0100 right(90)
0110 while key$=chr$(0) do null
```

Efter de tidligere øvelser med pilespiden skulle disse kommandoer være lette nok at forstå, måske undtagen ordren i linie 110. Vi vil gerne holde grafikskærmen synlig, efter vi har tegnet kvadratet. Når et COMAL program slutter, overgår kontrollen automatisk til tekstskærmen. Linie 110 fastfryser imidlertid grafikskærmen, indtil man trykker en eller anden taste. Denne handling vil gøre **KEY\$** forskellig fra (<>) **CHR\$(0)**, som står for det tomme tegn (intet-tegn). Indtil da vil computeren gøre **null** (intet). Når programmet fortsætter forbi denne linie, er der ikke flere instruktioner, så programmet vil stoppe.

Prøv at starte programmet. Tryk først <RUN/STOP> for at komme ud af AUTO tilstanden. Tast så **run**. Når der så tages <RETURN>, vil programmet blive udført trin for trin. Denne proces kaldes **udførelse** af et program.

Man kan spare lidt arbejde ved at taste <f7> i stedet for at skrive **run** samt <RETURN>.

Tryk <f1> for at vende tilbage til tekstskærmen. Lav ændringer i programmet for at se, hvad der sker. Forsøg med forskellige længder og vinkler for at lave andre figurer. Herefter vil vi fortsætte med nogle flere COMAL ordre.

Bemærk, at et tryk på <f3> tasten aktiverer grafiktilstanden. Dermed sætter det samtidigt sin egen funktion ud af kraft. Et nyt tryk på <f3> efter f.eks. et programstop med efterfølgende ændring af programmet, opstarter ikke turtle-pakken forfra. Tryk først på <CTRL-u> for at genaktivere funktionstastens virkning.

### GENTAGELSE AF INSTRUKTIONER

Efter at have eksperimenteret med programeksemplet - og måske med femkanter og ottekanter - får man den tanke, om det var muligt at gentage et givet sæt instruktioner, som skal udføres flere gange. Det er også muligt. Processen kaldes *gentagelse* og er et af de mest almindelige begreber inden for programmering.

Der findes en lettere måde at tegne et kvadrat på. Slet arbejdslageret (brug **new** <RETURN>) og prøv følgende program. (Hvis **AUTO** benyttes, udskrives nye linienumre med foranstillede 0'er. Det er ikke nødvendigt, hvis man vælger selv at indtaste linienumrene):

```
0010 // program: KVADRAT
0020 // af: <dit navn>
0030 use turtle
0040 splitscreen
0050 for side:=1 to 4 do
0060 forward(75)
0070 right(90)
0080 endfor
0090 while key$=chr$(0) do null
0100 end // programmet er slut
```

Tryk <RUN/STOP> for at standse den automatiske linienummerering, og skriv så **llst** for at få en udskrift af programmet. Den bør se sådan ud:

```
0010 // program: KVADRAT
0020 // af: <dit navn>
0030 USE turtle
0040 splitscreen
0050 FOR side:=1 TO 4 DO
0060 forward(75)
0070 right(90)
0080 ENDFOR
0090 WHILE KEY$=CHR$(0) DO NULL
0100 END // programmet slut
```



Som det ses, er det muligt at indføje kommentarer i programmet, blot kommentarlinierne indledes med //. Sådanne sætninger bliver ikke udført, men vil være indeholdt i udskriften. De kan også anvendes efter COMAL sætninger i en programlinje som i linie 100. Læg mærke til, at programmet indrykker linie 60 og 70 i udskriften for at gøre programstrukturen klarere. Konstruktionen FOR-ENDFOR (50-80) bevirker, at linie 60-70 gentages fire gange.

NØGLEORD er fremhævet i den anden udskrift. (Nøgleord er ord, som COMAL systemet på forhånd kender.) Efterse programstrukturen ved hjælp af ordren SCAN. SCAN kan f.eks. udføres ved tryk på funktionstasten <f8>. Foretag derefter en ny udskrift af programmet ved hjælp af LIST kommandoen. Bemærk da, at variabelnavnet **side** er inkluderet efter ENDFOR i linie 80. Ved at gennemse programmet vil man i øvrigt ved den automatiske indryknings hjælp opdage eventuelle fejl i programstrukturen.

---

Vi har nu set, hvordan COMAL redigerer programmet for at gøre udskriften mere overskuelig. Fra nu af i denne vejledning vil vi vise programmerne i deres endelige, redigerede form. Det vil dog nok være lettest for brugeren at blive ved med at skrive programmerne med små bogstaver. Lad COMAL udføre det ekstra arbejde med at lave en pæn udskrift!

---

Prøv at bringe programmet **kvadrat** til udførelse. Tryk en tilfældig taste for at stoppe programmet og tryk så <f1> for at komme tilbage til den fulde tekstskeerm. Lad os nu foretage nogle ændringer for at se, hvad der sker. Vi kan f.eks. ændre programmet, så det tegner en sekskant eller en ottekant. Hvor en sætning skal gentages mange gange, bliver FOR-ENDFOR konstruktionen specielt nyttig. Prøv at tilpasse programmet, så pilespiden tegner en figur, der er tæt ved at være en cirkel.

---

Måske har du bemærket, at pilespiden skal dreje i alt 360 grader for at fuldende en polygon og ende med at pege i samme retning som ved starten. (Kendere af Logo vil måske genkende dette princip som Den Totale Skildpaddetur.) Så for at tegne en regulær polygon med **antal** sider, skal pilespiden dreje **360/antal** grader ved hver vinkelspids.

---

Det er naturligvis muligt at tilpasse dette program, så det tegner en polygon med et vilkårligt antal sider. For at gøre det skal vi specificere det ønskede antal sider og sidelængden ved hjælp af INPUT sætninger. Slet arbejdslageret (**new** <RETURN>), og prøv at indlæse følgende program:

```
0010 // program: polygon
0020 // af: <dit navn>
0030 PAGE // slet tekst
0040 INPUT "Hvor mange sider? ": antal
0050 INPUT "Sidestykke?": lang
0060 USE turtle
0070 splitscreen
0080 FOR side:=1 TO antal DO
0090   forward(lang)
0100   right(360/antal)
0110 ENDFOR side
0120 WHILE KEYS:=CHRS(0) DO NULL
0130 END "programmet er slut"
```

Læg mærke til, at programmet her er vist, som det ville blive udskrevet. Man kan indtaste programmet med små bogstaver og uden indrykninger og derpå skrive **list** for at få ovenstående programstruktur at se. Bring programmet til udførelse for at blive forvisset om, at det gør, hvad det skal. Bemærk specielt, at teksten i anførselstegn efter END udskrives på skærmen som sluttekst.

---

Det vil vise sig, at brugen af linienumre i COMAL er ret så forskellig fra deres brug i andre linieorienterede sprog, f.eks. Basic. I denne henseende ligner COMAL meget mere Pascal. Brug RENUM kommandoen hyppigt for at få "ryddet op" i programmet (<f1> vil udføre denne kommando, hvis man ikke er i grafiktilstanden). COMAL ordre henviser sjældent til et linienummer, og derfor behøver man ikke interessere sig så meget for dem. I almindelighed er det en god idé at gruppere sine programinstruktioner i tre afsnit:

#### **begyndelsen**

programnavn, dato, bemærkninger, dimensionering af variabler, opstart af pakker osv.

#### **midten**

hovedprogrammet, bestående overvejende af procedurekald

#### **slutningen**

samlingen af procedurer, som kaldes af hovedprogrammet

---

## **COMAL PROCEDURER**

Procedurer er moduler eller byggeblokke, som man selv kan skabe for at lette programmeringen. Der er en linie i programmet **polygon**, som egner sig til at blive lavet om til en procedure. Programmet bliver lettere at læse og lettere at forstå, hvis man benytter sig af en procedure. Denne teknik bliver meget vigtig, efterhånden som man begynder at skrive længere programmer.

Se engang på programmet og læg især mærke til sætningen linie 120:

```
0120 WHILE KEYS=CHRS(0) DO NULL
```

her brugt i programmet **kvadrat** for at holde grafiskskærmen synlig, indtil én eller anden taste anslås. Linien kunne omdannes til en procedure, så hovedprogrammet fremtræder mere overskueligt:

```
0140
0150 PROC afvent'taste
0160 WHILE KEYS=CHRS(0) DO NULL
0170 ENDPROC afvent'taste
0180
```

Bemærk her, at vi har kaldt proceduren **afvent'taste**. *Apostroffen* 'er nødvendig for at sammenkæde de to ord, der beskriver proceduren, til ét sammenhængende ord uden mellemrum (blanktegn). Hvis man ikke gør det, vil COMAL kun fortolke bogstaverne før det første blanktegn som procedurenavnet, og resultatet vil være en fejlmelding, når COMAL kontrollerer procedures struktur.

Føj denne procedure ind i programmet og erstat linie 120 med:

```
0120 afvent'taste
```

Bring nu kommandoen LIST til udførelse (det gøres nemmest med <f6><RETURN>). Læg mærke til følgende træk ved programstrukturen, som den nu fremtræder:

- \* LIST kommandoen rykker instruktionerne i proceduren ind, hvorved proceduren skiller sig ud fra det øvrige program, som nu er blevet lettere at læse.
- \* Proceduren skal afsluttes med ENDPROC. Hvis der gives en SCAN kommando, sætter COMAL selv **procedurenavnet** ind i ENDPROC instruktionen, hvis programmøren ikke allerede har gjort det.
- \* **De tomme sætninger** i linie 140 og 180 er ikke nødvendige. De er føjet til for at få proceduren til at fremtræde klart adskilt fra hovedprogrammet, når programmet udskrives.  
Programmet **polygon** kunne forbedres yderligere ved at lave en procedure af de sætninger, som faktisk tegner polygonen.  
Polygonproceduren kunne skrives på denne måde:

```
1200 proc polygon(antal,lang)
1210 for side:=1 to antal
1220 forward(lang)
1230 right(360/antal)
1240 endfor
1250 endproc
1260
```

Når der er givet kommandoerne SCAN og LIST, skulle proceduren se sådan ud:

```
1200 PROC polygon(antal,lang)
1210 FOR side:=1 TO antal DO
1220     forward(lang)
1230     right(360/antal)
1240 ENDFOR side
1250 ENDPROC polygon
1260
```

Der er nogle ting, man bør lægge mærke til ved ovenstående:

- \* På printer indrykkes linier to tegn for at fremhæve programstrukturen. På billedskærmen foretages der af pladshensyn kun én indrykning.
- \* **Procedurenavnet** følges af to navne i en parentes (**antal,lang**), hvilket antyder, at proceduren vil kræve at kende værdien af **antal** sider og at vide, hvor **lang** siden er. En procedure behøver ikke have en variabeliste efter navnet; men den kan have det, som vist her.

Igen skal vi **kalde** proceduren for at bringe den til udførelse. Det oprindelige program skal ændres, så det ser sådan ud, når der er foretaget omnummerering (med RENUM) og kommandoen LIST er udført:

```
0010 // program: polygon
0020 // af <dit navn>
0030 PAGE
0040 USE turtle
0050 splitscreen
0060 INPUT "Hvor mange sider? ": antal
0070 INPUT "Sidedestykke? ": lang
0080 // HOVEDPROGRAM
0090 polygon(antal,lang)
0100 afvent'taste
0110 END // HOVEDPROGRAM slut
0120
0130 PROC afvent'taste
0140 WHILE KEYS=CHRS(0) DO NULL
0150 ENDPROC afvent'taste
```

```

0160
0170 PROC polygon(antal,lang)
0180   FOR side:=1 TO antal DO
0190     forward(lang)
0200     right(360/antal)
0210   ENDFOR side
0220 ENDPROC polygon
0230

```

Som allerede nævnt efterses et program, før det bringes til udførelse, ved at benytte kommandoen SCAN (kan enten skrives, eller man kan benytte <f8>). Når man gør det, vil COMAL gennemse programmet og "lære" de definerede procedurer. Hvis man derefter skriver et defineret procedur navn som kommando, vil det udføres. Dette gør det muligt at efterse procedurerne én efter én, hvad der er en stor fordel, når man foretager fejlsøgning i sit program!

Endnu nogle kommentarer er på sin plads her: Vi har brugt den generelle struktur, beskrevet tidligere, med en tydelig *begyndelse*, en *midte* og en *slutning* på programmet. Data indlæses i linie 60 og 70; hovedprogrammet er blot nogle få linier (80-110) og procedurerne er placeret sidst i programmet.

I linie 90 kaldes proceduren **polygon**. I parentes følger de to variable, som proceduren behøver for at tegne polygonen. Navnene i kaldet behøver ikke være de samme som variabelnavnene i proceduren, men det er vigtigt, at de står i samme rækkefølge.

Også lige en bemærkning om linie:

```
0110 END // HOVEDPROGRAM slut
```

Denne linie er ikke nødvendig for at stoppe programmet. Et COMAL program vil standse, når der ikke er flere linier at udføre i hovedprogrammet. Linien er medtaget her for at gøre hovedprogrammets struktur tydeligere. Det er mest af alt et spørgsmål om stil. Efterhånden som man får mere programmerings erfaring, udvikler man stærke meninger om programmeringsstil!

### LAGRING AF PROGRAMMER OG PROCEDURER

Nu da vi er begyndt at skrive programmer, der kan bruges igen senere, vil det være ønskeligt at gemme programmerne. Følg venligst de instruktioner, der er relevante i dit tilfælde:

#### Brug af en Datassette enhed

For at lagre programmet **polygon** på kassettebånd, skal man gøre som følger:

- \* Anbring kassettebåndet i båndenheden og kontroller, at det er spolet helt tilbage til begyndelsen.

---

**ADVARSEL:** Hvis båndet starter med et stykke **tomt bånd** uden magnetisk belægning, skal båndet spilles lidt frem. Ellers risikerer man, at første del af programmet ikke gemmes.

---

- \* Indtast følgende kommando på tastaturet:

```
save "cs:polygon" <RETURN>
```

- \* På skærmen vil meldingen **Press record & play on Tape** komme til syne.
- \* Tryk **RECORD** og **PLAY** på båndenheden. Det tager kun omkring 15 sekunder at lagre et kort program som **polygon**.
- \* Mens programmet lagres, vil skærmen være blank.
- \* Når programmet er lagret, vil meldingen:

```
program saved
```

komme frem.

- \* Det kan meget kraftigt anbefales, at man gentager denne proces og laver endnu en kopi: **sikkerhedskopien**. Det vil nok være mest bekvemt at gøre det bag på båndet, hvis der er tale om databånd på 10 eller 15 minutter. På længere bånd er det nok lettest at gemme kopien lige efter den første optagelse for at undgå tidrøvende tilbagespilning.
- \* Sæt nu en **etikette** på båndet, så du ved, hvad du har! Det tager nogle få ekstra minutter nu, men det kan spare en masse besvær senere!

---

De fleste erfarne programmører gemmer deres programfiler ca. hvert kvarter, mens de arbejder. Det er klogt at gemme programmet, så snart der er skrevet mere, end man vil bryde sig om at miste i tilfælde af strømsvigt eller andet uheld. Det er endvidere klogt at gemme to arbejdskopier: den **nuværende kopi** og den **foregående kopi**. På en båndenhed gøres det ved at skifte side, hver gang der lagres på et kort bånd. Hvis man følger dette råd, indlæses den foregående kopi blot, hvis der sker et uheld, mens man arbejder på den nuværende. Når hele programmet er færdigt og **fejlrrettet**, vil det være hensigtsmæssigt at have mindst to kopier af den endelige udgave: en **original udgave** og en **sikkerhedskopi**.

---

## Brug af en diskettestation

Nu bliver der brug for *lagerdisketten*, som forberedtes tidligere. Hvis du ikke har gjort det endnu, så følg vejledningen i sidste del af kapitel 1. Dernæst gør man som følger:

- \* Indsæt *lagerdisketten* i *diskettestationen*.
- \* Indtast nu følgende på tastaturet:

**save "polygon" <RETURN>**

- \* Aktivitetslampen tændes, og diskettestationens motor vil kunne høres i nogle få sekunder, mens en kopi af programmet bliver lagret på disketten. Man er frit stillet med hensyn til valg af navn, op til 16 tegn langt. Naturligvis er det klogt at vælge navne, som er beskrivende og som gør det let at finde programmet igen. Det er endvidere altid klogt at medtage filnavnet i en af de første programlinier i en kommentarsætning.
- \* For at se om programmet nu også er gemt, tastes **dir** (eller **cat**) og trykkes **<RETURN>**. Det gør, at der vises et katalog over diskettens indhold, og hvor mange blokke hvert program fylder (1 blok = 256 oktetter), og hvor mange ledige blokke. (XXX blocks free.).
- \* Der bør altid laves en **sikkerhedskopi** af alle vigtige programmer på en anden diskette ... blot for det tilfælde...! Og mens du er ved at udvikle programmet, bør der laves en kopi ca. hvert kvarter for at undgå spildt arbejde i tilfælde af strømsvigt eller andet uheld! Det er bedst altid at have to aktuelle kopier, blot for en sikkerheds skyld.
- \* Vær omhyggelig med at sætte **etikette på disketterne** (gør det straks!). Så bliver det lettere at finde frem til programmerne igen. Når man først begynder at skrive mange programmer, vil disketterne formere sig som mus!

Det er også muligt at lagre procedurerne individuelt. Det gøres ved at anvende en udgave af LIST kommandoen. Den beskrives i forbindelse med diskussionen af mere avanceret filbehandling i kapitel 6.

## REPETITION

I dette kapitel har vi præsenteret oplysninger om, hvordan man skal:

- \* udstede direkte ordrer fra tastaturet (kommandoer)
- \* rette indtastningsfejl og styre markøren rundt
- \* bruge turtle grafik

- \* skrive simple programmer ved brug af procedurer
- \* bruge automatisk linienummerering
- \* bruge en Datasette båndenhed eller diskettestation
- \* direkte udførelse kontra programmeret udførelse
- \* Den Totale Skildpaddetur
- \* trykning af tekst og tal på tekstskræmen
- \* kald af procedurer
- \* brug af procedurer med variabler
- \* brug af gentagelse

Følgende COMAL kommandoer og nøgleord er blevet præsenteret i dette kapitel:

- \* PRINT <tekst eller tal>
- \* ZONE <afstand>
- \* forward(<trin>)
- \* back(<trin>)
- \* right(<grader>)
- \* left(<grader>)
- \* penup
- \* pendown
- \* USE <pakke>
- \* clearscreen
- \* splitscreen
- \* showturtle
- \* hideturtle
- \* pencolor(<farve>)
- \* background(<farve>)
- \* setheading(<grader>)
- \* WHILE - DO gentagelser
- \* KEY\$ - (aflæser tastatur bufferen)
- \* CHR\$(0)

- \* AUTO - (for automatisk linienummerering)
- \* RUN - (bringer et program til udførelse)
- \* END - (markerer afslutningen på et program)
- \* // - (til indsætning af kommentarer i programmet)
- \* FOR - DO - ENDFOR gentagelseskonstruktion
- \* INPUT "<input stikord>": <variabel>
- \* NULL - en sætning, som intet udretter!

Hvis du nu har gennemarbejdet dette kapitel, skulle du være parat til de mere avancerede beskrivelser af COMAL programmering, som følger i næste kapitel. Det kan være en hjælp at holde sig for øje, at programmering i virkeligheden kan reduceres til tre grundlæggende elementer:

- \* **Handlingsblokke** er grupper af instruktioner, som indlæser data, udfører beregninger, laver en tegning, udlæser data eller udfører en anden proces i programmet.
- \* **Løkkeblokke** er grupper af instruktioner, som gentages et antal gange. Instruktionerne FOR - DO - ENDFOR og WHILE - DO - ENDWHILE er eksempler på løkkeblokke, men der er endnu flere til rådighed i COMAL.
- \* **Forgreningsblokke** er instruktionsserier, som indeholder beslutninger om, hvilke ordrer der nu skal udføres. Der vil være mere at lære om denne type instruktioner i næste kapitel.

## Kapitel 3 -

# PROGRAMMERING MED COMAL



Dette kapitel er tænkt som en første indføring i, hvordan programmer kan skrives i COMAL. COMAL begreberne indføres skridt for skridt uden dog at gå helt i dybden. Der er givet eksempler med hvert nyt begreb. Vi vil gennemgå nogle enkle programmer og forklare, hvordan de virker.

Eksemplerne er forsøgt valgt sådan, at de foruden COMAL-emnerne også viser noget om Commodore 64'ens muligheder eller er en uddybning af tidligere indførte COMAL-sætninger. Der er valgt stigende sværhedsgrad i eksemplerne.

Algoritmebegrebet er omtalt ved de sidste eksempler, som er valgt for at vise COMAL's styrke ved mere avanceret, struktureret programmering.

Det er ikke meningen, at man skal stille sig tilfreds efter at have prøvet eksemplerne og øvelserne, for de er langt fra udtømmende. Læs andre bøger om COMAL og prøv at forstå og forbedre de programmer, som du får adgang til. Fortsæt med at programmere. Nyttige programmer kan være en stor hjælp i undervisning, i erhvervslivet og dagligdagens øvrige gøremål.

### FÅ GODE PROGRAMMERINGSVANER

Enhver, der programmerer meget, får efterhånden sin egen pro-



grammeringsstil. I starten kan det dog være rart med nogle gode råd. For at komme i gang med at løse en programmeringsopgave kan følgende forslag måske være en hjælp:

- \* Tast **new** for at slette tidligere programmer i arbejdsområdet.
- \* Tast f.eks. **auto 100** for automatisk linienummerering.
- \* Gå i gang med at skrive hovedprogrammet som en række kald af procedurer, der skal udføres, for at opnå programmets mål. COMAL's smidige redigeringsfaciliteter gør det enkelt at rette og ændre undervejs.
- \* Når hovedprogrammets struktur er klar, gå så i gang med at skrive hver procedure for sig. Del evt. større opgaver i andre, mindre procedurer.
- \* Lav hyppigt en LISTning af programmet for at se, hvordan det ser ud. Det er ikke altid som forventet!
- \* Lav et SCAN af programmet for at kontrollere korrekt opbygning.
- \* Efter LIST og SCAN rettes eventuelle fejl ved hjælp af de specielle redigeringsordrer. Se f.eks. appendiks D for informationer herom. For en sikkerheds skyld, husk altid at lave *sikkerhedskopier* af dit program med jævne mellemrum.
- \* Når programmet forekommer fejlfrit, bringes det til udførelse med ordren RUN. Programmet kan i de fleste tilfælde standses med et eller flere tryk på <RUN/STOP>, hvis det ikke skulle standse af sig selv. Samtidig tryk på <RUN/STOP> og <RESTORE> (svarende til en genstart af COMAL) kan også bruges, hvis <RUN/STOP> alene ikke standser programmet.
- \* Når programmet er køreklart, bør programmet gemmes på diskette eller kassettebånd til senere brug. F.eks. ordren **save**
- \* "<programnavn>" til diskette eller **save "cs:<programnavn>"** til kassette.

---

Programmerne vises, som de ser ud efter at være struktur kontrollerede med ordren SCAN. Under indtastningen er det unødvendigt at skrive med store bogstaver og ekstra mellemrum til indrykning. Det søger COMAL systemet selv for. Alle programmerne fra kapitel 3 medfølger på demonstrationsdisketten eller kassetten.

---

## DEN FØRSTE BEREKNING

Det første eksempel viser noget om, hvordan computeren behandler tal:

## Program 1:

```
new
auto 100
0100 // gennemsnit udregnes
0110 tala:=7
0120 talb:=15
0130 gennemsnit:=(tala+talb)/2
0140 PRINT "Gennemsnittet af tallene"
0150 PRINT tala;"og";talb
0160 PRINT "er";gennemsnit
0170 END
```

Undersøg ved hjælp af LIST og SCAN, om programmet er korrekt indtastet. Ret eventuelle fejl.

Skriv **run** og tryk på <RETURN>-tasten.

Computeren vil da på de næste linier svare med:

**Gennemsnittet af tallene  
7 og 15 er 11**

## Bemærk om program 1:

De to // streger i linie 100 angiver, at linien er en kommentarlinie, som systemet ikke skal regne på.

Computere "husker" tal og andre størrelser ved hjælp af *variabler*. En variabel er et navn, der f.eks. kan indeholde en talværdi. Program 1 indeholder 3 variabler: **tala**, **talb** og **gennemsnit**.

I linie 110 *tildeles* variablen **tala** værdien 7, og i linie 120 tildes **talb** værdien 15. Ved hjælp af det særlige lighedstegn :=, der kaldes et *tildelingstegn*, forsynes variabler altså med en værdi. := kaldes også for et *dynamisk lighedstegn*.

---

Hvis man under indtastning blot angiver := som et sædvanligt lighedstegn, vil systemet selv omdanne det til et tildelingstegn.

---

Et variabelnavn skal altid begynde med et bogstav og bestå af i alt højst 80 tegn (d.v.s. bogstaver, tal eller bestemte specialtegn). Et navn kan afsluttes med #, \$ eller (), hvorved det får en særlig betydning. Se senere. Inden for et givet virkefelt vil **a**, **a#**, **a\$** og **a()** dog altid anses for at være samme navn. Da de er af forskellig type, vil det resultere i en fejlmeddelelse.

I linie 130 udregnes *udtrykket* **(tala+talb)/2** (som betyder: læg **tala** og **talb** sammen, og divider derefter med 2), hvorefter variablen **gennemsnit** tilskrives værdien af det udregnede udtryk.

**NB:** Rækkefølgen er vigtig: Udtrykket på højre side af tildelingstegnet udregnes altid først, hvorefter navnet på venstre side tildeles den udregnede værdi.

At bytte om på højre og venstre side vil resultere i en fejlmeddelelse.

I linierne 140 til 160 udskrives resultatet ved hjælp af PRINT sætninger. Som det ses, er det nemt at få både tekst og tal udskrevet på skærmen.

I linie 140 udskrives teksten mellem de to anførselstegn.

I linie 150 udskrives først **værdien af tala**. Derefter følger teksten **og**, og til sidst **værdien af talb**. Bemærk semicolon (;) mellem tallene og teksten. Det er nødvendigt som adskillestegn i programlinien og bevirker udskrift af et mellemrum.

I linie 160 skrives først teksten **er**, efterfulgt af **værdien af gennemsnit**.

### Bemærk:

- \* Udskriften starter på en ny linie ved hver PRINT sætning.
- \* Det er ikke variabelens navn, som udskrives, men dens værdi.

I linie 170 afsluttes programmet med sætningen END.

### Øvelser:

1. Ret i programmet, så **tala** får værdien 5.
2. Forsøg med andre værdier til **tala** og **talb**.
3. Føj en ny linie til programmet:

**0105 page**

Hvilken effekt har denne ordre?

4. Sæt et semicolon (;) som afslutning på hver af linierne 140 - 160. RUN programmet, og bemærk at ; forhindrer lineskift.
5. Prøv at lave et program, som beregner gennemsnittet af tre tal. Sørg også for, at udskriften er korrekt.

### INPUT SÆTNINGEN

I forrige eksempel så vi et program, hvori computeren regnede med tal og skrev resultatet ud på skærmen. For at udregne gennemsnittet af to tal måtte vi for hver gang ændre i to programlinier.

Disse linier ændres nu én gang for alle, så programmet kan udregne gennemsnittet af to vilkårlige tal.

### Program 2:

Hvis du har en diskette eller et kassettebånd, hvor **Program 1** er gemt, kan det hentes ind i arbejdslageret med ordren **load "program 1"**. Derefter kan de to linier ændres.

Ellers skrives **Program 2** således:

```
new
auto 100
0100 // gennemsnit udregnes
0110 INPUT "Skriv 1. tal ": tala
0120 INPUT "Skriv 2. tal ": talb
0130 gennemsnit:=(tala+talb)/2
0140 PRINT "Gennemsnittet af tallene"
0150 PRINT tala;"og";talb
0160 PRINT "er";gennemsnit
0170 END
```

Når programmet er rettet for eventuelle indtastningsfejl, køres det med kommandoen RUN.

Med sætningen INPUT kan programvariablerne tildeles en værdi, mens programmet udføres.

Det er altså muligt ikke blot at udskrive variabelværdier fra et program, men også indlæse værdier til et program.

### Bemærk:

- \* Programudførelsen standses i en INPUT sætning, indtil man har reageret på den. I **Program 2** kræves det f.eks., at man indtaster et tal som svar, efterfulgt af <RETURN>.
- \* INPUT sætningens tekst afsluttes altid af et kolon (:) før variabelen. Alle andre tegn vil resultere i fejlmeddelelser.

### Øvelser:

1. Føj en linie med PAGE ordren til programmet, så skærbilledet fremtræder pænere.
2. Det er også muligt at lade udskriften foregå til en printer, hvis en sådan er tilsluttet computeren.

Tilføj f.eks. linierne

```
135 SELECT OUTPUT "lp:"
165 SELECT OUTPUT "ds:"
```

Kør programmet igen og se, hvad der sker.

Linie 135 dirigerer udskriften til printeren, og linie 165 bringer den tilbage til dataskærmen.

3. Skriv et program, som beregner gennemsnittet af 3 tal. Tallene skal indlæses i INPUT sætninger.

### CIRKLER

Udskrift fra et program kan også være i form af en tegning. Næste program tegner cirkler.

### Program 3:

```

auto 100
0100 // cirkler tegnes
0110 PAGE
0120 INPUT "Skriv 1. radius ": radiusa
0130 INPUT "Skriv 2. radius ": radiusb
0140 sumradius:=radiusa+radiusb
0150
0160 USE graphics
0170 graphicscreen(1)
0180 circle(160,100,radiusa)
0190 circle(160,100,radiusb)
0200 circle(160,100,sumradius)
0210
0220 WHILE KEYS=CHR$(0) DO NULL
0230 END

```

Efterse det indtastede program og kør det.

Programmet består af en indlæsnings- og beregningsdel adskilt fra en udskrivningsdel af den **tomme linie** 150. Sådanne tomme linier kan indsættes på passende steder i et program for at fremme overskueligheden.

Linie 160 og 170 er nødvendige for at sætte computeren i stand til at tegne.

Ved hjælp af linierne 180-200 tegnes tre cirkler, der alle har centrum i skærmpunktet (160,100), dvs. ca. midt på skærmen.

Radius i de tre cirkler fremgår af linierne 120-140. Hvis radius bliver større end 99 vil cirklen tegnes ud over skærmområdet.

Sætningen i linie 220 er beskrevet i kapitel 2. Den har til formål at opretholde billedet på grafikskærmen, indtil brugeren trykker på en vilkårlig taste.

Funktionen KEY\$ er en anden måde at læse værdier ind i et program under udførelse. Den vender vi tilbage til senere.

### Bemærk:

Det kan være, at "cirklen" mere ligner en ægformig kurve på skærmen. Det skyldes, at skærbilledets højde/bredde forhold er stillet forkert. Hvis TV'et/monitoren er forsynet med en manuel højdejustering, kan man nu indstille den, så cirkler virkeligt fremtræder som cirkler.

### Øvelser:

1. Ret i programmet, så den tredje cirkel tegnes med en radius, som er *forskellen* mellem de to indtastede radier. Det er sjusk, hvis ikke også navnet **sumradius** ændres!
2. Forsøg med andre regneoperationer i udtrykket i linie 140. Se appendiks C.
3. Flyt cirklernes centre.
4. Tilføj linier, så der tegnes flere cirkler med andre radier og andre centre.
5. Centrum kan også indlæses i en INPUT sætning.

Tilføj f.eks. linien

```
135 INPUT "Centrum: X,Y =": xc,yc
```

Ret linie 180-200 til:

```

180 circle(xc,yc,radiusa)
190 circle(xc,yc,radiusb)
200 circle(xc,yc,sumradius)

```

Kør programmet.

Bemærk, at man i den nye INPUT-sætning skal svare med to værdier, adskilt af et (,).

6. Cirklerne kan udfyldes med farver. Dertil benyttes proceduren **fill(x,y)**, hvor (x,y) skal være koordinaterne for et punkt inden i den lukkede figur, som ønskes farvet.

Hvis f.eks. **Program 3** udvides med linierne

```

202 pencolor(2)
204 fill(160,100)

```

vil den inderste cirkel farves rød. Prøv!

7. Prøv at farvelægge andre områder på skærmen ved at ændre på koordinaterne i linie 204.  
Udskift f.eks. linie 204 til

**204 fill(0,0)**

Hvad sker der?

8. Forsøg nu selv at farvelægge andre områder på skærmen. Ved at ændre tallet i proceduren **pencolor** linie 202 opnås andre farver. Se farvekodetabellen i appendiks B.

### PROCEDURER 1

Når man skal lave større COMAL programmer, bør man udnytte begrebet *procedurer*:

En procedure er et underprogram, som kan kaldes fra selve hovedprogrammet. Det illustreres bedst ved nogle eksempler.

### Program 4:

```
new
auto 100
0100 //tyldte cirkler og firkanter
0110 grafikopstart
0120 tegn'firkant(10,10,300,180,brun)
0130 tegn'cirkel(160,100,70,gul)
0140 tegn'firkant(100,50,50,50,lilla)
0150 tegn'cirkel(125,75,20,orange)
0160
0170 WHILE KEYS=CHRS(0) DO NULL
0180 END
0190
0200
0210 PROC grafikopstart
0220   USE graphics
0230   graphicscreen(1)
0240   brun:=8
0250   gul:=7
0260   lilla:=4
0270   orange:=10
0280 ENDPROC grafikopstart
0290
0300 PROC tegn'firkant(xmin,ymin,xside,yside,farve)
0310   pencolor(farve)
0320   moveto(xmin,ymin)
0330   draw(xside,0)
0340   draw(0,yside)
0350   draw(-xside,0)
0360   draw(0,-yside)
0370   xpunkt:=xmin+.5*xside
0380   ypunkt:=ymin+.5*yside
0390   paint(xpunkt,ypunkt)
0400 ENDPROC tegn'firkant
```

```
0410
0420 PROC tegn'cirkel(xcentrum,ycentrum,radius,farve)
0430   pencolor(farve)
0440   circle(xcentrum,ycentrum,radius)
0450   paint(xcentrum,ycentrum)
0460 ENDPROC tegn'cirkel
```

Kør programmet efter at have rettet eventuelle indtastningsfejl. Bag-efter kan vi se på, hvorfor programmet gør, som det gør.

Program 4 består af:

Hovedprogram	(linie 100-180)
Tre procedurer:	
grafikopstart	(linie 210-280)
tegn'firkant	(linie 300-400)
tegn'cirkel	(linie 420-460)

En procedure kaldes ved et navn, samt måske en parentes med en liste af navne eller tal (parametre), hvis værdier skal overføres til proceduren.

Selve proceduren opbygges som

```
PROC <navn>(<a>,<b>,<c>,...)
<sætning 1>
<sætning 2>
...
...
...
ENDPROC <navn>
```

---

Husk, at de kantede parenteser < > om et ord betyder, at ordet og de kantede parenteser kan udskiftes med navne eller sætninger efter brugerens valg: F.eks. <navn> kunne tænkes erstattet med navnet **grafikopstart**, **udskrift** eller noget andet, som beskriver procedurens formål. Betegnelsen <sætning nr> står for en lovlig COMAL sætning.

---

Hovedprogrammet består af en kommentarlinie efterfulgt af 5 linier, der alle kalder en procedure.

I linie 110 kalder hovedprogrammet blot proceduren med navnet **grafikopstart**, og computeren går nu over og udfører de sætninger, der står i denne procedure.

Når computeren har udført sætningerne i proceduren, vender den tilbage til hovedprogrammet og går videre med næste linie.

I linie 120 kaldes så proceduren med navnet **tegn'firkant**. Her kaldes den ikke kun med sit navn, men også med en parentes, der indeholder en række tal. Tallene adskilles med (,).

Der skal være **præcist** lige så mange tal i kaldet, som der er variabelnavne i procedurens egen parentes:

```
tegn'firkant(10 ,10 ,300 ,180 ,brun)
PROC tegn'firkant(xmin,ymin,xside,yside,farve)
```

### Bemærk:

- \* Variablen **brun** har talværdien 8. Den værdi fik den tildelt i proceduren **grafikopstart**.
- \* Ved kaldet af **tegn'firkant** tildeles procedurens variabler de tilsvarende værdier:

```
xmin:=10
ymin:=10
xside:=300
yside:=180
farve:=brun  (:=8)
```

- \* Nu går computeren i gang med at udføre sætningerne i **tegn'firkant**, idet den kender variabelernes værdi.
- \* Procedurene **tegn'firkant** og **tegn'cirkel** består af en række grafikordrer. Benyt stikordsregistret bag i denne bog til at finde frem til sætningernes betydning.
- \* Endvidere beregner **tegn'firkant** i linie 370 og 380 et punkt midt i firkanten.
- \* Når computeren er færdig med proceduren **tegn'firkant**, hopper den tilbage til hovedprogrammets næste linie.
- \* I linie 130 kaldes proceduren **tegn'cirkel**, som derefter udføres.
- \* I linie 140 og 150 kaldes procedurene igen, men denne gang med andre talværdier.
- \* En procedure kan kaldes mange gange, og med forskellige værdier i parentesen. Det er netop en af fordelene ved en procedure.

### Øvelser:

1. Forsøg at flytte rundt på cirklerne og firkanterne. Det gøres ved at ændre i parentesernes to første tal, som står for koordinaterne

for henholdsvis cirkelns centrum og firkantens nederste, venstre hjørne.

Flyt f.eks. den sidste firkant og cirkel ind midt på skærmen:

```
140 tegn'firkant(135,75,50,50,lilla)
150 tegn'cirkel(160,100,20,brun)
```

2. Firkanternes sidelængder kan også ændres. Og cirkelernes radier.
3. Tilføj andre farver. Se farvekode i appendiks B.
4. Andre cirkler og firkanter kan tegnes ved at tilføje nye programlinier med procedurekald i hovedprogrammet. Prøv.
5. Prøv selv at lave en procedure, der tegner en trekant og fylder den ud med en farve. Lav en programlinje som kalder den nye procedure.

### COMAL OG TEKSTER

Det næste eksempel er også opbygget om et hovedprogram, der kalder to procedurer:

```
Hovedprogram          (100 - 160)
Proceduren indtastning (190 - 260)
Proceduren udskrift    (280 - 460)
```

Før vi går i gang med at indtaste, afprøve og studere det i detaljer, skal vi have indført begrebet *streng*.

En **strengkonstant** er en tekst, indrammet i anførselstegn. F.eks. "John", "dejligt vejr" og "han har 7 segl".

Hidtil har alle de variabler, vi har beskæftiget os med, været talvariabler.

Det er også muligt at definere variabler som kan bruges til at lagre rækker af bogstaver, tal og andre tegn. Sådanne variabler kaldes *strengvariabler*.

*Strengnavne* kendes på, at de altid skal afsluttes med et dollartegn (\$). Eksempler på strengnavne er:

```
navn$, by$, land$
```

Når en streng skal tildeles en værdi, må man først oplyse computeren om, hvor mange tegn, den skal gøre plads til. Det sker i en *erklæringssætning*, også kaldet en *dimensioneringssætning*. En erklæringssætning kan dog udelades. Se appendiks C.)



## Eksempler:

```
DIM navn$ OF 20      (plads til 20 bogstaver)
DIM by$ OF 25        (plads til 25 bogstaver)
DIM land$ OF 40      (plads til 40 bogstaver)
```

Nu kan strengvariablerne tildeles tekstværdier (strengkonstanter):

```
navn$:="Esbern Snare"
by$:="Silkeborg"
land$:="Danmark"
```

## Bemærk:

- \* Tekst skal altid omsluttet af anførselstegn " .
- \* Man behøver ikke gøre teksten så lang, som der er afsat plads til i erklæringssætningen.
- \* En tekstvariabel kan rumme store og små bogstaver, mellemrum, tal, eller specialtegn som (.,/<>?!"#\$\$%' + - ; =). På Commodore 64 kan den også indeholde grafiske symboler. Et bogstav, tal, specialtegn osv. benævnes med fællesbetegnelsen *tegn*. Når vi således taler om plads til 25 bogstaver, mener vi: plads til 25 tegn. F.eks. er variabelen **land\$** ovenfor tildelt 7 tegn i sin streng.

I det følgende eksempel (**program 5**) skal vi altså både indøve brugen af procedurer og lære mere om strenge og strengvariabler:

Desuden skal vi prøve at benytte computerens semigrafiske tegn, der står afbildet på forsiden af tastaturet. Se i appendiks D om tastaturet, hvordan disse tegn tages.

Læg især mærke til følgende i proceduren **udskrift** under indtastning af programmet:

- \* Linie 310: 2 mellemrum og 36 <C= o> tegn.
- \* Linie 320: 2 mellemrum, 1 <C= j>, 34 mellemrum og 1 <C= l> tegn.
- \* Linie 400: 2 mellemrum og 36 <C= u> tegn.

(NB: <C= o> betyder: hold Commodore-tasten nede, mens der trykkes på o-tasten.)

## Program 5:

```
new
auto 100
0100 // indtastning og udskrift af tekst
```

```
0110 DIM tils OF 25
0120 DIM fra$ OF 25
0130 DIM tekst$ OF 30
0140 indtastning
0150 udskrift
0160 END
0170
0180
0190 PROC indtastning
0200 PAGE
0210 PRINT "Skriv en besked:"
0220 INPUT "Brevet er til ": tils
0230 INPUT "Brevet er fra ": fra$
0240 PRINT "Beskeden kan fylde en linie."
0250 INPUT "Start her:": tekst$
0260 ENDPROC indtastning
0270
0280 PROC udskrift
0290 PAGE
0300 PRINT
0310 PRINT "-----"
0320 PRINT " | "
0330 PRINT " | "
0340 PRINT " | "
0350 PRINT " | "
0360 PRINT " | "
0370 PRINT " | "
0380 PRINT " | "
0390 PRINT " | "
0400 PRINT "-----"
0410 PRINT AT 4,6: "Til ";tils
0420 PRINT AT 6,6: tekst$
0430 PRINT AT 8,6: "Venlig hilsen"
0440 PRINT AT 9,6: fra$
0450 CURSOR 20,1
0460 ENDPROC udskrift
```

I hovedprogrammet erklæres først variablerne **tils**, **fra\$** og **tekst\$**. Dernæst kaldes proceduren **indtastning**, som sørger for, at variablerne fyldes med tekst.

Når indtastningsproceduren er færdig, vender computeren tilbage til hovedprogrammet. I næste linie dirigeres udførelsen til proceduren **udskrift**, som udskriver beskeden i en pæn ramme.

## Bemærk:

- \* En ny **PRINT**-version, nemlig

**PRINT AT <linie>,<kolonne>.**

Eks. i linie 440, hvor **fra\$**-teksten skal begynde på linie 9, kolonne 6.

Denne skrivemåde giver mulighed for at placere tekst (eller tal) et vilkårligt sted på skærmen.

- \* Linie 450: **CURSOR 20,1**

**CURSOR** <linie>, <kolonne> placerer markøren (eng. cursor) et givet sted på skærmen, men skriver ingen besked.

- \* Se i øvrigt **INPUT AT**, som benyttes i Program 10.

### Øvelser:

1. Kør programmet nogle gange med forskellige meddelelser for at få et indtryk af virkemåden.
2. Hvis en printer er tilkoblet, kan man opnå en kopi af tekstskræmen på papir ved hjælp af kommandoen <CTRL P>:  
Når programmet er kørt færdigt, og teksten står som ønsket på skærmen, trykkes på **P**, mens <CTRL>-tasten holdes nede.
3. Skriv et nyt program, som indlæser nogle strenge og skriver dem ud forskellige steder på dataskærmen.

Her er en kort repetition af det foregående om strenge og tekst:

1. En computer kan arbejde med såvel tal som ord. Det gøres ved hjælp af talvariabler med almindelige navne - og tekstvariabler med navne efterfulgt af \$.
2. Variabler kan tildeles en værdi bl.a.
  - \* i tildelingssætninger :=
  - \* i parenteser i procedurekald
  - \* i INPUT-sætningen
3. Tekst udskrives på dataskærmen ved hjælp af PRINT sætninger. (Det kan også gøres på andre måder. F.eks. i tekstdelen til en INPUT sætning, som vi har set).
4. Tegninger kan laves på skærmen ved hjælp af grafikordrer fra grafikpakken. (**use graphics** eller **use turtle**), eller ved hjælp af det semigrafiske tegnsæt, som står foran på tastaturet.
5. Hvis et program er blot af en vis størrelse, bør det opbygges af *procedurer*. En procedure er et underprogram, som kan benyttes gentagne gange fra et hovedprogram eller fra en anden procedure. Se senere mere om brugen af procedurer.

### FORGRENINGER: BETINGEDE UDFØRELSE

Computeren kan også skelne mellem udtryk, der er **sandt** eller **falsk**, (eng. **true** eller **false**). Sådanne udtryk kaldes *logiske udtryk*. Nogle eksempler:

**7=2** er et logisk udtryk, som både vi og computeren vil opfatte som falsk (false).

**23<54** er et sandt (true) logisk udtryk.

Om det logiske udtryk **tal>10** er sandt eller falsk kan ikke afgøres, før vi kender værdien af **tal**.

COMAL indeholder de to logiske konstanter TRUE og FALSE, som har talværdierne hhv. 1 og 0.

Her følger eksempler på, hvordan computeren kan bringes til at udføre forskellige sætninger, afhængigt af om et logisk udtryk er sandt eller falsk.

### Program 6:

```
new
auto 100
0100 // find maksimum
0110 PAGE
0120 PRINT "To tals maksimum:"
0130 PRINT
0140 INPUT "Skriv 1. tal ": a
0150 INPUT "Skriv 2. tal ": b
0160
0170 maksimum:=a
0180 IF maksimum<b THEN maksimum:=b
0190
0200 PRINT
0210 PRINT "Maksimum er ";maksimum
0220 END
```

Det nye er i linie 180: **IF - THEN**

Det er et eksempel på en *forgrening*, også kaldet en *betinget udførelse*, som i dette tilfælde betyder:

"HVIS variabelen **maksimum** er mindre end variabelen **b**, SÅ sættes **maksimum** lig med **b**".

Computeren vurderer det logiske udtryk **maksimum<b**.

Hvis det er sandt, gør den det, der står efter ordren THEN. Det udtrykker man ofte som: *betingelsen* mellem IF og THEN skal være opfyldt.

Computeren går blot videre til næste linie, hvis betingelsen ikke er opfyldt.

Der kan imidlertid tit være tilfælde, hvor det er ønskeligt, at nogle sætninger udføres, hvis betingelsen er opfyldt, mens andre udføres.

hvis betingelsen ikke er opfyldt. Det klares i COMAL med en ny struktur:

#### IF - THEN - ELSE - ENDIF:

IF <betingelse> THEN

<sætning 1>

<sætning 2>

...

ELSE

<sætning a>

<sætning b>

...

ENDIF

Linie 170 - 180 i program 6 kunne derfor også skrives ved hjælp af denne IF - version:

```
170 IF a<b THEN
172   maksimum:=b
174 ELSE
176   maksimum:=a
180 ENDIF
```

#### Program 7:

```
new
auto 100
0100 // forkert eller rigtigt
0110 DIM tekst$ OF 10
0120 PAGE
0130 PRINT "Find mit tal. 1, 2 eller 3"
0140 INPUT "Kom med et bud ": svar'tal
0150
0160 RANDOMIZE
0170 mit'tal:=RND(1,3)
0180
0190 IF svar'tal=mit'tal THEN
0200   tekst$="RIGTIGT"
0210 ELSE
0220   tekst$="FORKERT"
0230 ENDIF
0240
0250 PRINT
0260 PRINT "Mit tal var ";mit'tal
0270 PRINT "Buddet var ";svar'tal
0280 PRINT
0290 PRINT "Det betyder, at buddet var ";tekst$
0300 END
```

#### Bemærk i dette program:

- \* Linie 190-230: IF - THEN - ELSE - ENDIF strukturen, hvis virkemåde allerede er beskrevet.
- \* Linie 160-170: Computeren er i stand til at frembringe et tilfældigt tal ved hjælp af ordrene RANDOMIZE og RND:

RANDOMIZE gør, at computeren stiller sig et "tilfældigt" sted i en "tilfældigtal"-række.

I **mit'tal:=RND(1,3)** tildeles **mit'tal** en **tilfældig** (eng. RaNDom) værdi mellem 1 og 3.

Værdiområdet kan ændres. F.eks. vil RND(-10,10) tilfældigt frembringe et af tallene -10, -9, -8,...,0,..., 8, 9, 10.

#### Øvelser:

1. Forsøg med andre talområder i RND.
2. Prøv at ændre sætningen RANDOMIZE til RANDOMIZE 1, og kørs programmet flere gange. Hvad sker der?

#### CASE-STRUKTUREN

Hvis man skal skelne mellem mange betingelser på én gang, er *CASE-strukturen* fordelagtig. Den opbygges efter princippet:

**CASE** <variabel> **OF**

**WHEN** <1.værdi>

<sætning 1a>

<sætning 1b>

...

...

**WHEN** <2.værdi>

<sætning 2a>

<sætning 2b>

...

...

(flere **WHEN**-værdier)

...

...

**OTHERWISE**

<sætning a>

<sætning b>

...

...

**ENDCASE**

Hvis f.eks. <variabel> er lig <2.værdi>, springer computeren ned og udfører sætningerne i det tilhørende afsnit: <sætning 2a> - <sætning 2b> osv. Derefter hopper den ud og fortsætter med linien efter ENDCASE.

Hvis <variabel> ikke er lig nogle af de anførte WHEN værdier, springer computeren ned og udfører sætningerne i OTHERWISE afsnittet. OTHERWISE med tilhørende sætninger kan udelades.

Denne struktur er benyttet i det følgende eksempel, hvor man kan vælge mellem nogle forskellige opgaver om regne-operationer.

Hver opgave er anført i en procedure. Et svar på en opgave vurderes i proceduren **resultat**, som derfor kaldes fra hver opgave-procedure:

**Hovedprogram:**    **opgave1 - resultat**  
                       **opgave2 - resultat**  
                       **opgave3 - resultat**  
                       **opgave4 - resultat**

### Program 8:

```
new
auto 100
0100 // Regneopgaver
0110 PAGE
0120 PRINT "Tag en opgave:"
0130 PRINT
0140 INPUT "Hvilket nummer (1 - 4) ": nummer
0150
0160 CASE nummer OF
0170 WHEN 1
0180   opgave1
0190 WHEN 2
0200   opgave2
0210 WHEN 3
0220   opgave3
0230 WHEN 4
0240   opgave4
0250 OTHERWISE
0260   PRINT "Der er valgt et ukorrekt nummer"
0270 ENDCASE
0280
0290 END
0300
0310
0320 PROC opgave1
0330   PRINT
0340   INPUT "INT(7.3+3.2 DIV 2) = ": svar
0350   facit:=INT(7.3+3.2 DIV 2)
0360   resultat(facit,svar)
0370 ENDPROC opgave1
0380
```

```
0390 PROC opgave2
0400   PRINT
0410   INPUT "3-30/2+12 = ": svar
0420   facit:=3-30/2+12
0430   resultat(facit,svar)
0440 ENDPROC opgave2
0450
0460 PROC opgave3
0470   PRINT
0480   INPUT "4.25+2.5/5*2 = ": svar
0490   facit:=4.25+2.5/5*2
0500   resultat(facit,svar)
0510 ENDPROC opgave3
0520
0530 PROC opgave4
0540   PRINT
0550   INPUT "34 MOD 10-2*5 = ": svar
0560   facit:=34 MOD 10-2*5
0570   resultat(facit,svar)
0580 ENDPROC opgave4
0590
0600 PROC resultat(facit,svar)
0610   PRINT
0620   PRINT "Svaret er: ";svar
0630   PRINT "Facit er: ";facit
0640   PRINT
0650   IF svar=facit THEN
0660     PRINT "Svaret er helt korrekt"
0670   ELSE
0680     PRINT "Bedre held en anden gang"
0690     PRINT "Se i Appendiks C om regneoperationer"
0700   ENDIF
0710
0720 ENDPROC resultat
```

### Bemærk:

En procedure kan også kaldes fra en anden procedure, ikke blot fra hovedprogrammet. Eks. **resultat** kaldes fra **opgave**-procedurerne.

### Øvelser:

1. Prøv at besvare opgaverne. Se appendiks C om regneoperationer.
2. Lav en ny opgave 5:  
     Skriv en procedure **opgave5**  
     Tilføj den nye WHEN værdi i CASE strukturen  
     Husk også at ændre i INPUT sætningen
3. Lav et program, som udskriver 4 forskellige meddelelser. Meddelelsen skal afhænge af værdien af en indlæst variabel.

## GENTAGELSER OG LØKKER

*Gentagelser* er én af de grundlæggende byggeblokke i programmering. Computeren er velegnet til at gentage operationer mange gange. Der findes i COMAL flere forskellige sætninger, som forårsager gentagelser. Disse sætninger benævnes med et fællesord: *løkker*.

Det første eksempel viser, hvordan computeren kan bringes til at udføre nogle ordrer et bestemt antal gange:

**Udfør** <disse sætninger> **100 gange.**

Det sker i en FOR - ENDFOR løkke:

```
FOR <nr>:=<start> TO <slut> DO
```

```
  <sætning a>
```

```
  <sætning b>
```

```
...
```

```
...
```

```
ENDFOR <nr>
```

Sætning a, b m.m. gentages (<slut>-<start>+1) gange, hvis <slut> og <start> er hele tal:

første gang er <nr> lig <start>

anden gang er <nr> lig <start>+1

trede gang er <nr> lig <start>+2

sidste gang er <nr> lig <slut>

### Program 9:

```
new
```

```
auto 100
```

```
0100 // test af RND
```

```
0110 USE graphics
```

```
0120 graphicscreen(0)
```

```
0130 wrap
```

```
0140 window(0,1000,-10,10)
```

```
0150 moveto(1000,0); drawto(0,0)
```

```
0160
```

```
0170 FOR i:=0 TO 1000 DO
```

```
0180   tal:=RND(-10,10)
```

```
0190   moveto(i,0); draw(0,tal)
```

```
0200 ENDFOR i
```

```
0210
```

```
0220 WHILE KEYS=CHRS(0) DO NULL
```

```
0230 END
```

Programmet illustrerer, hvor tilfældige de RND frembragte tal fra computeren er.

Linie 170-200: FOR - ENDFOR sætningen

Løkken gentages 1001 gange.

Sætningen kan udbygges med en STEP parameter:

**FOR** <nr>:=<start> **TO** <slut> **STEP** <trin> **DO**

hvor STEP bevirker, at <nr> har værdierne: <start>, <start+trin>, <start+2\*trin> osv. Der slutes, når <nr> passerer <slut>.

Hvis STEP parametren er udeladt, (hvad vi hidtil har gjort), sættes STEP automatisk til 1.

---

I COMAL kan man benytte heltals-variabler. De kendetegnes ved et #-tegn efter navnet. Heltal fylder mindre i lageret, men vigtigst er det, at programudførelse i mange tilfælde kan foregå hurtigere, f.eks. i FOR - ENDFOR løkker.

---

Foruden de grafiksætninger, vi allerede har stiftet bekendskab med, indeholder programmet nogle nye, hvis betydning fremgår af afsnittet om grafik.

Endelig er det nu tiden at studere sætningen i linie 220. Her er nemlig også et eksempel på en gentagelse:

I WHILE - DO sætningen aflæser computeren tastaturet igen og igen, indtil en taste trykkes ned.

Nøgleordet KEY\$ er en funktion, som angiver det sidst indtastede tegn, hvis man har trykket på tastaturet. Er intet tastet, er KEY\$ lig CHRS(0), indtil en taste trykkes ned.

```
MENS <intet tegn tastet> GØR <intet>
```

```
WHILE KEY$=CHR(0) DO NULL
```

Den mest udbredte anvendelse af WHILE sætningen er dog i en version, som strækker sig over flere linier:

**WHILE** <betingelse> **DO**

```
  <sætning a>
```

```
  <sætning b>
```

```
...
```

```
...
```

**ENDWHILE**

Hvis betingelsen mellem WHILE og DO er opfyldt, går computeren i gang med sætning a, b,... Disse sætninger eksekveres forfra, indtil der sker et eller andet i sætningerne, som gør, at betingelsen ikke er opfyldt. Da vil programudførelsen springe fra WHILE linien til linien efter ENDWHILE.



Se under WHILE i stikordsregistret for at finde nærmere redegørelse for mulighederne.

En hyppigt anvendt løkkestruktur er REPEAT - UNTIL:

#### REPEAT

<sætning a>  
<sætning b>

...  
...

UNTIL <betingelse>

Sætningslisten gentages indtil <betingelse> er opfyldt.

I det næste eksempel indgår denne løkke som bestemmende for, hvor længe brugeren skal blive ved med at gætte på bogstaver i et "hemmeligt" ord. Eksemplet viser også, hvordan man i COMAL arbejder med strenge.

#### Programstruktur:

Hovedprogram - bestem'ord  
- nyt'bogstav

#### Program 10:

```
new
auto 100
0100 // ordbestemmelse
0110 PAGE
0120 bestem'ord
0130 nummer:=0
0140
0150 REPEAT
0160 nummer:=nummer+1
0170 nyt'bogstav
0180 UNTIL svar$=husk$
0190
0200 PRINT AT 20,5: "Hermed slut"
0210 PRINT AT 21,5: nummer;"bogstaver er brugt"
0220 END
0230
0240
0250 PROC bestem'ord
0260 DIM navn$ OF 20, bogstav$ OF 1
0270 DIM brugt$ OF 200
0280 INPUT "Nyt ord: ": navn$
0290 antal:=LEN(navn$)
0300 DIM svar$ OF antal, husk$ OF antal
0310 svar$="-----"
0320 husk$=navn$
0330 brugt$=""
0340 PAGE
```

```
0350 PRINT "BESTEM DETTE ORD MED";antal;"BOGSTAVER"
0360 PRINT AT 8,5: "Ord: ";svar$
0370 ENDPROC bestem'ord
0380
0390 PROC nyt'bogstav
0400 INPUT AT 10,5,1: "Nyt bogstav ": bogstav$
0410 brugt$=brugt$+bogstav$
0420
0430 position:=bogstav IN navn$
0440 IF position>0 AND position<=antal THEN
0450 svar$(position):=bogstav$
0460 navn$(position):="#"
0470 ENDIF
0480
0490 PRINT AT 10,17: " "
0500 PRINT AT 8,5: "Ord: ";svar$
0510 PRINT AT 12,1: brugt$
0520 ENDPROC nyt'bogstav
```

Linie 150-180: REPEAT - UNTIL løkken

Når brugeren har det svar, som computeren husker på, fortsætter programmet i linie 190.

#### Bemærk:

- \* Linie 160: variabelen **nummer** står på begge sider af tildelingstegnet. Det er lovligt (og meget benyttet). Husk på, hvordan et tildelingstegn (:=) virker. Først udregnes udtrykket på højre side af tegnet. Variablen på venstre side tildeles derefter denne udregnede værdi.
- \* Linie 400: **input at 10,5,1** betyder, at INPUT sætningen skal starte på linie 10, kolonne 5, og der skal være plads til 1 tegn i svarfeltet. Forsøg at skrive flere for at se, hvordan det virker. Prøv at ændre 1 til f.eks. 3, og afprøv ændringen.
- \* I linie 440 begynder en IF - ENDIF forgrening, som strækker sig over flere linier med afslutning i linie 470.
- \* Linie 440: **AND** er et eksempel på en *logisk operator*. Den kræver, at begge delbetingelser i IF - THEN sætningen skal være opfyldte.

#### Bemærk specielt om strenge:

- \* Linie 290: **LEN** funktionen angiver hvor mange tegn, der indgår i ordet. Hermed defineres ordets **længde**.
- \* Linie 300: Det er muligt at benytte variabler i DIM sætninger.
- \* Linie 410: Man kan 'lægge ord sammen' med + tegnet. Det kaldes at *sammenkæde* strenge.

Eksempel: "hav"+"kat" giver ordet "havkat"

- \* Linie 430: **IN** er en logisk operator, som virker på strenge. Den angiver første position af søgestrengens første tegn.

Eksempler: "**lav**" **IN** "**madlavning**" giver talværdien **4**.

"**a**" **IN** "**madlavning**" giver talværdien **2**

Hvis søgestrengen ikke findes i den givne tekststreng, bliver værdien **0** (nul).

Eksempler:

"**peber**" **IN** "**madlavning**" giver værdien **0**

"**adle**" **IN** "**madlavning**" giver værdien **0**.

- \* Linie 450-460: Man kan udvælge bestemte delstrenge i en tekst ved strengens position i teksten.

Eksempel:

Lad **tekst\$:=madlavning**

**tekst\$(3)** er bogstavet "**d**"

**tekst\$(4:10)** er strengen "**lavning**"

- \* I linie 460 erstattes det fundne bogstav med et andet udvalgt tegn, som absolut ikke indgår i noget ord. Det sker for at tillade det samme bogstav at indgå flere steder i det ord, som skal gættes. I tilfældet her indsættes tegnet **#**.

### Øvelser:

1. Programmet kan hurtigt forenkles til at indsætte det gættede bogstav alle steder, hvor det indgår i det hemmelige ord. Foretag følgende ændringer:

**430 FOR position: TO antal DO**

**440 IF bogstav\$=navn\$ (position:position) THEN**

**475 ENDFOR position**

Afprøv dit nye program.

2. Med ovenstående ændringer vil programmet virke på den beskrevne måde. Nu er variabelen **husk\$** og linie 460 imidlertid overflødige. Kan du få programmet omskrevet med disse forenklinger?
3. Hvor skal programmet ændres for at kunne benytte hemmelige ord på mere end 20 bogstaver? Prøv.

## TABELLER: INDICEREDE VARIABLER

Når man f.eks. skal arbejde med mange tal, vil det blive for omstændeligt at indlæse og give dem forskellige navne. Det kan nemt blive op til 100 eller flere variabelnavne. Betragt f.eks. løsning af følgende opgaver:

- \* Udregning af gennemsnittet af 50 tal
- \* Bestemmelse af maksimum og minimum af 80 tal
- \* Sorter 40 forskellige tal

Det klares i COMAL ved at erklære en *tabel* i en dimensionerings-sætning, som f.eks. følgende:

**DIM x(50)**

Dvs., der oprettes plads til 50 tal, som alle har samme navn **x**, men forskelligt nummer:

**x(1), x(2), x(3),..., x(49), x(50)**

Sådanne variable kaldes *nummererede variable* eller *indicerede variable*, fordi de nummereres med et indeks.

Man kunne (men det gøres sjældent) give alle de indicerede variable en værdi ved hjælp af 50 tildelingssætninger:

**x(1):=23**

**x(2):=71**

**x(3):=-12.45**

.

.

**x(49):=6**

**x(50):=0.852**

I det næste programeksempel skal vi arbejde med indicerede variable, som tildeles en værdi i en **INPUT** sætning.

Programmet tegner en sammenhængende linie gennem nogle indtastede punkter.

Programmet består af  
en indlæsningsdel  
(linie 110-220)  
en tegnedel  
(linie 270-300)

**Program 11:**

```

new
auto 100

0100 // punkttegning
0110 DIM x(50), y(50)
0120 PAGE
0130 PRINT "En linie tegnes gennem punkterne"
0140 PRINT
0150 REPEAT
0160   INPUT "Antal punktpar : ": antal
0170 UNTIL antal >= 2 AND antal <= 50
0180 PRINT
0190 FOR nr:=1 TO antal DO
0200   PRINT "Indtast x(",nr,"),y(",nr,"):";
0210   INPUT "": x(nr),y(nr)
0220 ENDFOR nr
0230 PRINT
0240 PRINT "Tryk en taste, og figuren tegnes"
0250 WHILE KEYS=CHRS(0) DO NULL
0260
0270 USE graphics
0280 graphicscreen(0)
0290 moveto(x(1),y(1))
0300 FOR nr:=2 TO antal DO drawto(x(nr),y(nr))
0310 WHILE KEYS=CHRS(0) DO NULL
0320 END

```

**Bemærk:**

- \* Linie 110: Der gøres plads til 50 punkters x- og y-koordinater.
- \* Linie 160: Programmet spørger i en INPUT sætning om, hvor mange punktpar, der ønskes indlæst. INPUT sætningen omslutes af en REPEAT - UNTIL løkke, som skal sørge for, at der mindst svares 2.
- \* Linie 190-220: Punktparrene x(1),y(1) x(2),y(2)... x(antal),y(antal) indlæses i en FOR - ENDFOR løkke.
- \* I linie 270-300 tegnes figuren ved hjælp af grafiksætninger.

**Øvelser:**

1. Prøv programmet på nogle punkter.
2. Tilføj en linie, så der sættes en lille cirkel om hvert punktpar. F.eks. `circle(x(nr),y(nr),3)`.
3. Skriv et program, som beregner gennemsnittet af et vilkårligt antal værdier.

Indlæs antal

Indlæs talværdierne i en tabel

Beregn summen af tallene  
 Gennemsnittet := Summen/antal

4. De tabeller, vi har beskæftiget os med, har haft et enkelt indeks. Derfor benævnes de *en-dimensionale*-tabeller. I COMAL kan en tabel have flere dimensioner, f.eks.:

**DIM reol (3,4)**

Variablen **reol** er en to-dimensional tabel. Man kan opfatte det som en reol med 3 hylder, hver med plads til 4 ting:

56	17	-3	72
89	0.5	14	94
8	-6	78	66

Med ovenstående værdier gælder f.eks.:

**reol(2,3)=14** og **reol(3,1)=8**

Prøv at ændre i **Program 11**, så de en-dimensionale tabeller **x()** og **y()** erstattes af en to-dimensional tabel **punkt(,)**:

Linie 110: **DIM punkt(50,2)**

Indføj selv ændringerne i linie 210 og 290-300.

**TEKSTABELLER**

Ikke blot kan vi erklære tabeller indeholdende tal. Vi kan også erklære tabeller, som indeholder strenge:

**DIM besked\$(8) OF 20**

Der sættes plads af til 8 **besked\$**'er, hver af 20 tegns længde:

**besked\$(1):="Glem ikke solen"**

**besked\$(8):="Hurra. Hurra"**

Ligesom ved tal-tabeller kan tekst-tabeller have flere dimensioner.

Det næste program er et eksempel på anvendelse af en 2-dimensional teksttabel.

Tabellen erklæres i linie 130:

**DIM person\$(50,4) OF 30**

Det udnyttes som en fortegnelse af op til 50 personer, hver arkiveret med 4 oplysninger:

```

person$(nr,1):="<navn>"
person$(nr,2):="<gade>"
person$(nr,3):="<by>"
person$(nr,4):="<Tlf. nummer>"

```

I programmet skal vi også stifte bekendtskab med endnu en måde at indlæse variabelværdier på: En DATA sætning.

Oplysninger kan lagres i DATA sætninger, som igen læses med READ sætninger.

Udføres f.eks. sætningerne:

```

READ tal,vare$,x,points
DATA 17,"dukke",-346,10

```

virker det som fire tildelinger:

```

tal:=17
vare$:="dukke"
x:=-346
points:=10

```

Bemærk her, at tal og strenge kan blandes.

Programmet består af:

Linie 120-250:	dimensionering og datatildeling
Linie 270-350:	udskrift af oplysninger, som passer til søge- teksten
Linie 380-500:	DATA sætninger

#### Program 12:

```

new
auto 100

```

```

0100 // kartotek
0110 PAGE
0120 antal:=50; nr:=0
0130 DIM person$(antal,4) OF 30, tekst$ OF 30
0140 DIM fundet(antal)
0150 REPEAT
0160   nr:=1
0170   FOR oplysning:=1 TO 4 DO READ person$(nr,oplysning)
0180   UNTIL EOD
0190   antal:=nr
0200
0210 INPUT "Led efter: ": tekst$
0220 FOR nr:=1 TO antal DO
0230   oplysning:=0

```

```

0240 REPEAT
0250   oplysning:=1
0260   fundet(nr):=tekst$ IN person$(nr,oplysning)
0270   UNTIL fundet(nr)>0 OR oplysning=4
0280 ENDFOR nr
0290
0300 PRINT
0310 PRINT "Personer, der passer til tekst:"
0320 PRINT
0330 FOR nr:=1 TO antal DO
0340   IF fundet(nr)>0 THEN
0350     FOR oplysning:=1 TO 4 DO PRINT person$(nr,oplysning)
0360     PRINT
0370   ENDIF
0380 ENDFOR nr
0390 END
0400
0410 DATA "Peter Hansen","Lindebakken 13"
0420 DATA "Silkeborg","06-841723"
0430 DATA "Commodore Data a-s","Bjerrevej 67"
0440 DATA "8700 Horsens","05-641155"
0450 DATA "Jan Mogensen","Skovgade 4"
0460 DATA "1717 København V","01-456701"
0470 DATA "Knud Børge Jensen","Sneglevej 12 D"
0480 DATA "2820 Gentofte","hemmeligt"
0490 DATA "Silkeborg Amtsgymnasium","Oslovej 10"
0500 DATA "8600 Silkeborg","06-810800"

```

#### Bemærk:

- \* Linie 160. En COMAL mulighed: `nr:=nr+1` svarer til `nr:=nr+1`.
- \* READ sætningerne behøver ikke stå sammen med DATA sætningerne. Programmets første READ ordre starter med indlæsning af den første dataværdi, ligegyldigt hvor den er placeret. (Det kan ændres. Benyt indholdsfortegnelsen til at finde referenceafsnit om READ og DATA.)
- \* I linie 180 benyttes funktionen EOD til at afslutte værdiindlæsningen. Dens værdi er 0 (d.v.s. falsk), indtil sidste dataværdi er indlæst. Da sættes den af systemet til 1 (d.v.s. sand). Derfor er UNTIL betingelsen opfyldt, og computeren fortsætter i linie 190.

#### Øvelser:

1. Afprøv programmet. Forsøg at forstå, hvordan det virker. Prøv at besvare **Led efter:** med blot <RETURN>. Tilføj nye DATA sætninger.
2. Udskift værdierne i DATA sætningerne til eget brug. Foruden kartotek over personer kan det f.eks. benyttes som vareoversigt. Udskiftes variabelnavnet **person\$** til **vare\$** kunne man forestille sig kartoteket som en beskrivelse af varenumre f.eks.:

```
vare$(nr,1):="lokale"
vare$(nr,2):="reol"
vare$(nr,3):="hylde"
vare$(nr,4):="genstand"
```

3. Føj en linie til programmet, som udskriver personens (varens) kartoteksnummer sammen med de øvrige oplysninger.
4. Tilføj endnu en oplysning om hver person i kartoteket:

```
DIM person$(antal,5) hvor f.eks.
person$(nr,5):="<andet>"
Eksempel:person$(3,5):="skylder mig en 10'er"
```

## PROCEDURER II

I afsnittet *PROCEDURER I* stiftede vi bekendtskab med to forskellige måder at bruge procedurer på:

### UDEN overførsel af parametre

```
//Hovedprogram
<sætninger>
...
  navn
  <sætninger>
...
END
//
PROC navn
  <sætninger>
...
ENDPROC navn
```

### MED overførsel af parametre

```
//Hovedprogram
<sætninger>
...
  navn(4,"Christina")
  <sætninger>
...
END
//
PROC navn(tal,tekst$)
  <sætninger>
...
ENDPROC navn
```

Hvis der er parameteroverførsel i parentes, skal antal og type passe sammen:

```
      navn (4, "John",nummer, x(), logo$)
PROC      navn (tal, tekst$, start,      nr(), streng$)
```

Man siger, at antallet og typen af de *aktuelle* parametre i procedurekaldet skal svare til antallet og typen af de *formelle* parametre i procedurens parentes.

**4,"John",nummer,x(),logo\$** er de aktuelle parametre  
**tal,tekst\$,start,nr(),streng\$** er de formelle parametre

Blot parametrene passer sammen, behøver de ikke have samme navn.

Vi har understreget, at procedurer bør udnyttes til programopbygning, fordi...

- \* Procedurer kan bruges igen og igen forskellige steder i programmet.
- \* Programmet bliver mere overskueligt og logisk forståeligt ved at blive opdelt i procedurer med velvalgte navne.
- \* Procedurer kan gemmes i et procedurebibliotek på diskette/kassette til senere brug i andre sammenhænge.

Der er mange måder at benytte procedurer på. De følgende afsnit er en indledende indføring i, hvordan andre procedure- og funktionsformer kan se ud:

- \* hvordan ligner de hinanden
- \* hvordan er de forskellige
- \* eksempler på deres brug

### LOKALE OG GLOBALE NAVNE

I COMAL skelner man mellem *globale* og *lokale* navne. Et lokalt variabelnavn er i modsætning til et globalt kun defineret og kendt i en begrænset del af programmet. For eksempel:

```
FOR nr:=2 TO antal DO
  <sætninger>
...
ENDFOR nr
```

Variabelnavnet **nr** er lokalt i FOR - ENDFOR løkken. Det er ikke defineret udenfor.

I forbindelse med procedurer taler man også om lokale navne, der kun er kendte i proceduren, og globale navne, som er kendte i hele programmet.

I det foregående procedure-resumé kan alle navne i *Eksempel 1* være globale.

I *Eksempel 2* er navnene **tal** og **tekst** lokale. Det gælder generelt for alle formelle parametre i en procedures parentes, at de er lokale. Desuden kan proceduren indeholde andre globale og lokale variable.

Fordelen ved lokale navne er, at de ikke griber forstyrrende ind i andre dele af programmet og omvendt.

Studér næste eksempel med globale og lokale navne. Bemærk værdien af de udskrevne variable.

### Program 13:

```
new
auto 100

0100 //lokale variable
0110 a:=1;b:=1
0120 PRINT a;b
0130 lokal'global(4)
0140 PRINT a;b
0150 END
0160
0170 PROC lokal'global(a)
0180 PRINT a;b
0190 ENDPROC lokal'global
```

I parameteroverførslerne indtil nu har der været tale om én-vejs "ind"-førsel fra hovedprogram TIL procedure. For at tillade "ud"-førsel af lokale variabelværdier FRA proceduren må parametrene erklæres med en REF ordre. Næste procedureeksempel viser hvordan.

### Program 14:

```
new
auto 100

0100 PROC minmax(a,b,REF min,REF max)
0110 // minimum og maksimum findes
0120 IF a<b THEN
0130 min:=a; max:=b
0140 ELSE
0170 min:=b; max:=a
0180 ENDIF
0190 ENDPROC minmax
```

Et hovedprogram, som udnytter denne procedure kan f.eks. se således ud:

```
0010 //Hovedprogram
0020 t:=23
0030 s:=-41
0040 minmax(t-s,t+s,minimum,maksimum)
0050 PAGE
0060 PRINT "t-s=";t-s;"og t+s=";t+s
0070 PRINT "Minimum, maksimum::minimum;maksimum
0080 END
```

### Øvelser:

1. Navnene er uden betydning. Udskift variabelnavnene **minimum** og **maksimum** med henholdsvis **a** og **b**. Bemærk, at det ikke giver nogen forskel i resultatet. En udskiftning sker nemmest med kommandoen **CHANGE**. F.eks. **change "minimum","a"**. Tryk **<RETURN>** for hver ændring. Tryk **<n>**, hvis ingen ændring ønskes.
2. Efter at en indtastet procedure er kontrolleret med **SCAN** kommandoen kan den benyttes som en direkte ordre.

Tast f.eks. direkte på tastaturet:

```
scan
minmax(12/7,7/12,x,y)
print x;y
```

Prøv med andre værdier og andre procedurer som direkte ordrer.

3. Foretag nedenstående ændring og kørs programmet:

```
100 PROC minmax(REF a,REF b)
185 a:=min;b:=max
og
40 minmax(t,s)
60 slettes ved at skrive:del 60 <RETURN>
70 print "Minimum, maksimum:";t;s
```

Bemærk, at variable **t** og **s** ændrer værdi i proceduren.

Nu kan proceduren imidlertid ikke længere benyttes på formen **minmax(67,78)** med konstanter i kaldet, men godt på formen **minmax(x,y)**, hvis variable **x** og **y** er tildelt værdier i forvejen:

```
scan
x=1236;y=251
dit=(x+y)/x;dat=(x-y)/y
minmax(dit,dat)
print "Minimum, maximum:";dit;dat
```

Forsøg med såvel den lovlige som den ulovlige version.

En særlig elegant egenskab ved procedurer er, at de kan kalde hinanden. En procedure kan sågar kalde sig selv. Man siger da, at proceduren er *rekursiv*.

Næste program viser et grafisk eksempel på, hvordan. Se programmer på demonstrationsdisketten eller kassetten for andre eksempler.

### Program 15:

```
new
auto 100

0100 // koncentriske fyldte cirkler
0110 USE graphics
0120 graphicscreen(1)
0130
0140 tegn'cirkel(160,100,100,2)
0150
0160 WHILE KEYS=CHRS(0) DO NULL
0170 END
0180
0190 PROC tegn'cirkel(xc,yc,r,farve)
0200   pencolor(farve)
0210   circle(xc,yc,r)
0220   paint(xc,yc)
0230
0240 IF r>10 THEN tegn'cirkel(xc,yc,r-10,farve+1)
0250
0260 ENDPROC tegn'cirkel
```

I linie 240 kalder proceduren **tegn'cirkel** sig selv, indtil **r** bliver for lille.

### FUNKTIONER

COMAL's indbyggede standardfunktioner kan benyttes ved udregninger. Vi har allerede benyttet standardfunktioner som f.eks. PI, RND, INT, LEN. Se i kapitel 4 om andre standardfunktioner.

På tilsvarende måde som man definerer procedurer ved

### PROC - ENDPROC

kan man i COMAL definere sine egne funktioner ved

### FUNC - ENDFUNC

Procedurer og funktioner har mange egenskaber og anvendelser tilfælles. Næste program viser, hvordan funktioner kan defineres og

f.eks. udnyttes til at finde rødder i en vilkårlig ligning. I programmet benyttes også nogle af standardfunktionerne.

Oversigt:

Hovedprogram (linie 100-350)

funktion **afrund** (linie 380-400)

funktion **f** (linie 420-440)

hvor funktionerne opbygges efter strukturen

```
FUNC <navn>(<tal>)
  <sætning a>
  <sætning b>
  ...
  ...
  RETURN <udregnet'udtryk>
  ...
ENDFUNC <navn>
```

Forståelsen af teorien bag den anvendte beregningsmetode kræver matematiske forkundskaber. De er imidlertid ikke nødvendige for at benytte programmet og forstå programsætningerne.

Inden for datalogien benytter man ordet *algoritme* om en regneforskrift eller beregningsmetode. Det hører også til god programmering at kunne beskrive den algoritme, programmet bygger på. Beskrivelsen kan være mere eller mindre detaljeret afhængigt af, hvem der skal læse den, men mindstekravet er, at man selv skal kunne forstå programmets opbygning, når der skal rettes i det senere. Der findes mange eksempler på stor ressourcespild, både i offentlige og i private virksomheder, som skyldes dårlig dokumentation af programmer.

### Programbeskrivelse:

1. Rodsøgning ved *midtpunktsmetoden*.
2. Programmet finder en løsning til ligningen **f(x)**, hvor **f** er funktionsudtrykket for en sammenhængende funktion.
3. Brugeren skal være i stand til at komme med startgæt på to tal **a** og **b**, således at **f(a)** og **f(b)** har forskelligt fortegn. Se nedenstående figur. Ellers går programmet ikke videre, men spørger igen om startgæt.
4. Midtpunktet mellem **a** og **b** findes, og funktionsværdien udregnes for denne værdi.
5. Hvis funktionsværdien er tilstrækkelig lille (det vil sige så tæt på



0, at man er tilfreds med regnenøjagtigheden), standser programmet efter at have udskrevet den fundne rod.

6. Ellers fortsætter programmet med at sammenligne fortegn:

Hvis funktionsværdien i midtpunktet har samme fortegn som funktionsværdien i **a** som på figuren, ligger den søgte rod ikke mellem midtpunktet og **a**, men mellem midtpunktet og **b**. Derfor sættes midtpunktet lig det nye **a**-endepunkt i den fortsatte søgning.

Hvis derimod funktionsværdien i **a** og funktionsværdien i midtpunktet har forskelligt fortegn, må der være en rod mellem **a** og midtpunktet. Midtpunktet sættes derfor lig det nye **b**-endepunkt.

7. Derefter starter programmet igen forfra med punkt 4.  
8. Således fortsættes med at snævre området omkring roden ind, indtil en rod er fundet inden for den ønskede regnenøjagtighed, eller brugeren har trykket på <STOP>.

#### Program 16:

```
new
auto 100

0100 // rod i ligningen f(x)=0
0110 PAGE
0120 lille:=1e-04
0130 REPEAT
0140 INPUT "Endepunktsværdier A,B: ": a,b
0150 UNTIL SGN(f(a))=-SGN(f(b))
0160
0170 LOOP
0180 fortegn'a:=SGN(f(a))
0190 fortegn'b:=SGN(f(b))
0200 xmidt:=(a+b)/2
0210 ymidt:=f(xmidt)
0220 IF ABS(ymidt)<lille THEN
0230 PRINT
0240 PRINT "Jeg har fundet en rod =";afrund(xmidt)
0250 STOP
0260 ELSE
0270 PRINT " ";
0280 IF SGN(ymidt)=fortegn'a THEN
0290 a:=xmidt
0300 ELSE
0310 b:=xmidt
0320 ENDIF
0330 ENDIF
0340 ENDLOOP
0350 END
0360
0370
```

```
0380 FUNC afrund(tal)
0390 RETURN INT(tal*10000+.5)/10000
0400 ENDFUNC afrund
0410
0420 FUNC f(x)
0430 RETURN 3*x*x+2*x-5
0440 ENDFUNC f
```

Selve funktionen **f(x)** defineres i funktionen **FUNC f(x)**. Her er den defineret ved udtrykket  $3*x*x+2*x-5$ , som betyder, at programmet finder en rod i ligningen

$$3*x*x+2*x-5 = 0$$

Vilkårlige regneforskrifter kan (og bør) indsættes i stedet for som afprøvning af programmet.

#### Bemærk:

- \* En ny COMAL løkke: **LOOP - ENDLOOP** (vedvarende gentagelse).
- \* Forståelsen fremmes ved beskrivende navne.
- \* Standardfunktionen **SGN(<udtryk>)**  
 $SGN(<udtryk>) = 1$ , hvis  $<udtryk>$  er større end 0  
 $SGN(<udtryk>) = 0$ , hvis  $<udtryk>$  er lig 0  
 $SGN(<udtryk>) = -1$ , hvis  $<udtryk>$  er mindre end 0
- \* Standardfunktionen **ABS(<udtryk>)** leverer udtrykkets numeriske værdi. Eks. **ABS(-2)** er lig 2
- \* Funktionen **afrund** afrunder udtrykket for **tal** til 4 decimaler.

Eks. **afrund(3.141593)** er lig **3.1416**

Skal der være 3 decimaler, erstattes de 10000 med 1000, osv.

Der skal helst være overensstemmelse mellem den krævede regnenøjagtighed bestemt ved variablen **lille** og afrundingsnøjagtigheden bestemt ved f.eks. **10000**. Der må som mindstekrav ikke afleveres decimaler ud over **lille**'s nøjagtighed.

#### Øvelser:

1. Kør programmet med forskellige funktionsudtryk for **f(x)**.  
Test programmet med funktioner med velkendte rødder f.eks.  $2x-6$ .

Benyt programmet til at løse ligninger, som ikke kan løses ved almindelige regnemetoder:

Ligningen  $\text{EXP}(x) = x + 7$  er et eksempel på en sådan "transcendent" ligning. Den kan løses med programmet ved at definere funktionen  $f$  som  $\text{EXP}(x) - x - 7$ .

2. Funktioner kan også benyttes som direkte kommando efter at være SCANNede. Prøv f.eks.:

```
scan
print afrund(2.71828183)
```

3. Lav en talfunktion **FUNC gennemsnit(a,b)**, som returnerer gennemsnittet af **a** og **b**. Afprøv den som direkte kommando.
4. Lav en talfunktion **FUNC vokantal(tekst\$)**, som tæller antallet af vokaler i en given streng. Afprøv den som direkte kommando. **Program 17** kan måske inspirere.

### STRENGFUNKTIONER

Funktioner kan benyttes til andet end beregninger af udviklede matematiske udtryk, selv om de der gør stor nytte.

Funktionerne, vi lige har set, er talfunktioner. I COMAL arbejder man også med *strengfunktioner*. En strengfunktion leverer, som det fremgår af navnet, en streng som resultat. En strengfunktions navn skal lige som en strengvariabels navn afsluttes med et \$-tegn.

**KEYS** er et eksempel på en indbygget standard-strengfunktion. Af andre kan nævnes **STRS(327)**, som laver talkonstanten 327 om til strengkonstanten "327".

Det følgende program viser et eksempel på, hvordan man selv laver en strengfunktion. Det består af et (kort) hovedprogram, samt funktionen **opdel\$**, som opdeler en given tekst i vokaler og konsonanter.

#### Program 17:

```
new
auto 100
0100 PRINT opdel$("COMAL strengfunktioner")
0110 END
0120
0130
0140 FUNC opdel$(a$)
0150 //konsonanter efter vokaler
0160 lang:=LEN(a$)
0170 FOR i:=1 TO lang DO
0180 IF a$(i) IN "aeiouyæøåAEIOUYÆØÅ" THEN
0190 a$=a$(i)+a$(:i-1)+a$(i+1:)
0200 ENDIF
0210 ENDFOR i
0220 RETURN a$
0230 ENDFUNC opdel$
```

Eksempel til afprøvning:

Hvis **a\$="firetal"** og **i:=2**; så vil linie 190 fungere som **a\$="i" + "f" + "retal"**

### Bemærk:

- \* Vokalerne opstilles i omvendt rækkefølge.
- \* COMAL fortolkeren kan forstå et udtryk som f.eks. **a\$(7:6)**. Det er aktuelt i linie 190, når **i:=lang**. (Men **a\$(8:6)** er udefineret).
- \* **a\$(i+1:)** betyder: **a\$** fra det (i+1)'te tegn til og med sidste tegn.
- \* **a\$(i:1)** betyder: **a\$** fra det første tegn til og med det (i-1)'te tegn.

### Øvelser:

1. Afprøv programmet for at se om det virker efter hensigten. Find selv på andre strenge. Til en afprøvning hører også specialtilfælde som f.eks. "a", "iiiiiiiieeeee", "qwrtp" og den tomme streng.
2. Lav selv en strengfunktion, som ombytter bogstavernes rækkefølge i en vilkårlig streng. Afprøv den.
3. Efter at være SCANNet kan en strengfunktion, ligesom en talfunktion, benyttes som direkte kommando. Eks.:

```
scan
print opdel$("Saftevand og Ispinde")
```

4. Lav en strengfunktion **FUNC udfyld\$(antal,bogstav\$)**, som udskriver **antal** ens **bogstav\$**'er. Afprøv den, f.eks. som kommando: **print udfyld\$(30,"\*")**.

### LUKKEDE PROCEDURER

Hvis man vil være helt sikker på at undgå navnekonflikt mellem variabelnavne i procedurer og hovedprogrammer, kan man LUKKE sin procedure eller funktion af for omverdenen. Derved bliver alle dens variabelnavne lokale. Kun værdier, som er angivet i parenteser efter procedurenavnet, slipper ind eller ud.

Det gøres med ordren **CLOSED**. For eksempel:

```
PROC navn(tal,tekst$) CLOSED
```

Det kan være særdeles nyttigt at være i stand til at lukke en procedure. Særligt når det drejer sig om en generel procedure, der gemmes i et procedurebibliotek, og som tænkes brugt i mange forskellige sammenhænge. Da kan man ikke gå og huske på, hvilke variabelnavne, der er benyttet i den.

Næste program viser et eksempel på en generel procedure, der sorterer en tilfældig talrække, så tallene stilles i voksende rækkefølge.

F.eks. 4, 3, 7, -1 bliver -1, 3, 4, 7

Sorteringsmetoden hedder *boblesortering*.

Der findes adskillige algoritmer til sortering. På demonstrationsdisketten- og kassetten er f.eks. medtaget **quick-sort**, som er hurtig og effektiv.

Boblesortering er ikke den mest effektive sorteringsmetode, men til gengæld sjov og enkel at forstå:

Man betragter tallene parvis ("man sætter en lille boble omkring"). Hvis det første tal er større end det næste, ombyttes de. Derefter betragtes det andet tal. Hvis det er større end det tredje, ombyttes de osv. Boblen glider hen ad talrækken. Boblen starter derefter forfra på rækken, og glider igen henad, mens tallene eventuelt ombyttes. Således fortsættes, til der ikke er flere tal at ombytte. En kort illustration følger:

1. gennemløb:

```
4 3 7 -1 ombyttes til 3 4 7 -1
3 4 7 -1 ingen ombytning
3 4 7 -1 ombyttes til 3 4 -1 7
```

2. Gennemløb:

```
3 4 -1 7 ingen ombytning
3 4 -1 7 ombyttes til 3 -1 4 7
3 -1 4 7 ingen ombytning
```

3. gennemløb:

```
3 -1 4 7 ombyttes til -1 3 4 7
-1 3 4 7 ingen ombytning
-1 3 4 7 ingen ombytning
```

Dernæst vil næste gennemløb være uden ombytninger.

Hovedprogram	(linie 100-290)
procedure <b>udskrift</b>	(linie 320-370)
procedure <b>ombyt</b>	(linie 390-420)
procedure <b>boblesortering</b>	(linie 440-610)

Alle procedurer er lukkede.

## Program 18:

new

auto 100

```
0100 DATA 2,4,78,45,23,-2,56,45,199,43
0110 DATA 3,0,100,34,-19,34,67,88,4,10
0120
0130 //data hentes ind
0140 DIM plads(100)
0150 nr:=0
0160 WHILE NOT EOD
0170 nr:=+1
0180 READ plads(nr)
0190 ENDWHILE
0200
0210 PAGE
0220 PRINT "Usorterede tal:"
0230 udskrift(nr,plads())
0240 PRINT
0250 PRINT
0260 boblesortering(nr,plads())
0270 PRINT "Sorterede tal :"

```

I boblesortering linie 460 benyttes sætningen **IMPORT**. Den kan be-

nyttes til at gøre variabler eller procedurer tilgængelige inde i en ellers lukket procedure. Her gøres procedurenavnet **ombyt** kendt i proceduren **boblesortering**.

I hovedprogrammet linie 340 benyttes ordren **ZONE 8** til at opdele udskriften i kolonner. Udskrift af en række tal, adskilt med komma (,) i PRINT-sætningen, vil foregå i kolonner med 8 tegn mellem tallenes begyndelse.

### Bemærk:

- \* DATA-sætningerne står først i hovedprogrammet. Så er de nemme at finde ved udskiftning af værdier.

### Øvelser:

1. Afprøv programmet med de viste talværdier. Forsøg med egne talværdier. Også specieltfælde som f.eks. **DATA 2** eller **DATA 3,3,3,3,3** bør indsættes som datasætninger.
2. Denne øvelse drejer sig om *Eksterne procedurer*: Hvis en disktestation er til rådighed, kan procedurer gemmes på diskette for sig selv. Senere kan de hentes ind i et hvilket som helst program, benyttes af programmet og gemmes igen efter brug.

Man siger, at en procedure er *ekstern*, når den er til rådighed uden for arbejdslageret, f.eks. på diskette.

Der er to betingelser som eksterne procedurer skal opfylde:

- a. De skal være lukkede: **CLOSED**.
- b. De må ikke indeholde **IMPORT** sætninger.

Slet alle programlinier på nær proceduren **udskrift**. Gem så proceduren **udskrift** på diskette. F.eks. som

```
save "ext.taludskrift"
```

**ext.** er tilføjet for at skelne den eksterne procedure fra andre gemte programmer.

Hent igen **program 18** ind fra diskette. Slet proceduren **udskrift** fra **program 18** og tilføj dernæst en linie med erklæring om, at programmet vil bruge en ekstern procedure:

```
300 PROC udskrift(antal,plads()) EXTERNAL "ext.taludskrift"
```

Kør nu programmet, og bemærk, at den eksterne procedure hentes ind fra diskette to gange under programudførelsen.

Metoden med eksterne procedurer sparer lagerplads, men er forholdsvis langsom og bør kun benyttes ved store procedurer, som kaldes enkelte gange fra et program.

3. Lav et program, som sorterer ord i alfabetisk rækkefølge.

Det kræver kun ganske få rettelser i **program 18**. Disse rettelser kan foretages på flere forskellige måder, men prøv med **CHANGE** kommandoen:

```
Først: 140 DIM ts(100) OF 20
```

```
510 IF ts(nr#)>ts(nr#+1) THEN
```

der næst

```
CHANGE "plads", "ts"
```

```
CHANGE "tal", "ts"
```

(Husk, **antal** skal ikke ændres)

Forsyn alle variabler i proceduren **ombyt** med \$-tegn, og udskift DATA-sætningernes indhold til ord.

Computeren kan stadig fortolke det logiske udtryk i linie 540, fordi et ord opfattes ved tegnenes ASCII-talværdi. Se appendiks A om ASCII koder.

Bemærk, at computeren behandler bogstaverne i rækkefølge. Hvis de første bogstaver er ens, sammenholdes de næste, osv. Eksempelvis er ordet **"abe"** 'mindre end' **"and"**, fordi **b** står før **n** i alfabetet.

Pas på med sammenligning af store og små bogstaver. Prøv.

### FILBEHANDLING

Vi har set på, hvordan man gemmer en kopi af et program på diskette eller kassette med kommandoen **SAVE**. En kopi af det gemte program kan senere hentes tilbage i arbejdslageret med kommandoen **LOAD**.

Der er også andre måder at gøre det på. Se f.eks. under **LIST - ENTER - MERGE** i kapitel 4.

Det næste program viser en af mange forskellige måder, hvorpå man kan gemme data (f.eks. tallister/navnelister eller en blanding af tal og tekst). Det gøres i en *fil* (eng. file). I kapitel 6 findes en detaljeret gennemgang af nogle eksempler på fil-behandling.

Programmet består af

Hovedprogram

procedurerne

```
tal'til'fil(<filnr>,<filnavn$>,tal(),antal)
```

```
tal'fra'fil(<filnr>,<filnavn$>,<REF tal()>)
```

De to procedurer sørger henholdsvis for at gemme tal-data på diskette/kassette og hente tal-data fra diskette/kassette.

Hovedprogrammet er et afprøvningsprogram, som blot gemmer indlæste tal på en fil, og dernæst henter dem tilbage igen til udskrift på dataskærmen.

Fremgangsmåden i procedurerne er, at de åbner op for en *datastrøm* til eller fra et område på en diskette eller kassette. Datastrømmen kendetegnes ved tallet <filnr>, og området på disketten kendetegnes ved sit <filnavn\$>. Derefter kan man skrive til datastrømmen, hvis den er åbnet i WRITE-tilstand, eller man kan læse fra datastrømmen, hvis den er åbnet i READ-tilstand. Datastrømmene vil forblive åbne, indtil de lukkes.

Gemme data:

```
OPEN FILE <filnr>,<filnavn$>,WRITE
```

```
PRINT FILE <filnr>: tal
```

```
CLOSE FILE <filnr>
```

Hente data:

```
OPEN FILE <filnr>,<filnavn$>,READ
```

```
INPUT FILE <filnr>: tal
```

```
CLOSE FILE <filnr>
```

### Program 19:

```
new
```

```
auto 100
```

```
0100 PROC tal'til'fil(filnr,filnavn$,tal(),antal)
0110 OPEN FILE filnr,filnavn$,WRITE
0120 FOR i:=1 TO antal DO
0130 PRINT FILE filnr: tal(i)
0140 ENDFOR i
0150 CLOSE FILE filnr
0160 ENDPROC tal'til'fil
0170
0180 PROC tal'fra'fil(filnr,filnavn$,REF tal())
0190 OPEN FILE filnr,filnavn$,READ
0200 i:=0
0210 REPEAT
0220 i:=i+1
0230 INPUT FILE filnr: tal(i)
0240 PRINT tal(i);
0250 UNTIL EOF(filnr)
0260 CLOSE FILE filnr
0270 ENDPROC tal'fra'fil
0280
```

```
0290
```

```
0300 // tal gemmes og hentes fra fil
```

```
0310 DIM tal(100)
```

```
0320 PAGE
```

```
0330 PRINT "Skriv tallene efterfulgt af <RETURN>"
```

```
0340 PRINT "Afslut med tallet 99999 :"
```

```
0350 nr:=0
```

```
0360 REPEAT
```

```
0370 nr:=nr+1
```

```
0380 INPUT "": tal(nr);
```

```
0390 UNTIL tal(nr)=99999
```

```
0400 nr:=nr-1 //sidste tal gemmes ikke
```

```
0410 PAGE
```

```
0420 FOR i:=1 TO nr DO PRINT tal(i);
```

```
0430 PRINT
```

```
0440 PRINT "TRYK EN TASTE FOR AT SKRIVE TIL FIL"
```

```
0450 WHILE KEYS=CHRS(0) DO NULL
```

```
0460
```

```
0470 tal'til'fil(2,"taldata",tal(),nr)
```

```
0480
```

```
0490 PAGE
```

```
0500 PRINT "TRYK EN TASTE FOR AT HENTE FRA FIL"
```

```
0510 WHILE KEYS=CHRS(0) DO NULL
```

```
0520 PAGE
```

```
0530
```

```
0540 tal'fra'fil(3,"taldata",tal())
```

```
0550
```

```
0560 END
```

### Bemærk:

- \* Procedurer kan placeres hvor som helst i et COMAL program. Både i begyndelsen (som her) og i slutningen.
- \* Hvis dataene skal lagres på kassette, skal filnavne forsynes med enhedsangivelse. Filnavnet skal være af formen: "**cs:taldata**".
- \* Data skal hentes under samme filnavn, som de gemtes med. Strømnummeret behøver derimod ikke være det samme.

Fordelen ved at gemme data på filer er, at dataene ikke knyttes til et bestemt program som i DATA sætninger. De samme data kan udnyttes i uafhængige programmer.

Bemærk specielt om **tal'til'fil**: I procedurekaldet skal angives, hvor mange (**antal**) tal, der skal gemmes.

Bemærk om **tal'fra'fil**: Computeren holder op med indlæsning fra filen, når der ikke er flere data. Da får funktionen **EOF(<filnr>)** værdien **TRUE**, og **UNTIL** betingelsen er dermed opfyldt.

Med **PRINT FILE** ordren gemmes dataene i ASCII-kode på tekstform. **INPUT FILE** henter dataene. Denne kombination fungerer både på diskette og kassette. Hvis man har diskette, bør man dog benytte henholdsvis **WRITE FILE** og **READ FILE** i stedet for, da dataene derved lagres hurtigere og mere kompakt i binær kode.

**Øvelser:**

1. Afprøv programmet med vilkårlige tal. Udskift navn og numre på filen. Undersøg lovlige strømnumre.
2. Brug programmet til at lave et datasæt. Benyt dette datasæt som talmateriale i **program 18** i stedet for tallene i DATA linierne. I **program 18** skal linierne 100-200 altså slettes og erstattes af linier, der indlæser tallene fra den netop oprettede datafil.
3. Lav et program, som gemmer strenge på en fil. Indskriv oplysningerne fra DATA sætningerne i program 12 deri.

Benyt derefter denne fil som datafil i stedet for **program 12's** DATA sætninger.

**FEJLHÅNDTERING**

Det er vigtigt, at programmer opbygges, så de ikke uden videre stopper midt i udførelsen, hvis man ikke lige netop gør, som programøren har tænkt.

En af de hyppigste årsager til uønskede stop er, at den, der benytter programmet, kommer til at indtaste et BOGSTAV i en INPUT sætning, som forventer et TAL som svar.

I COMAL er der en fejlhåndteringsstruktur, som er i stand til at tage vare på dette problem og mange andre fejlmuligheder. Anvendelse af fejlhåndteringsstrukturen uddybet i kapitel 4. Her tager vi kun fat på ovennævnte fejlmulighed.

Strukturen er:

**TRAP**

(sætninger med fejlmuligheder)

**HANDLER**

(udfør, hvis fejl)

**ENDTRAP**

Hvis der opstår en fejl i sætningerne mellem TRAP og HANDLER, springer computeren ned og udfører sætningerne mellem HANDLER og ENDTRAP, samtidigt med, at et fejlnummer ERR genereres. Fejlnummeret ERR kan være afgørende for, hvilke sætninger man ønsker udført i HANDLER-delen.

I det næste programeksempel er fejlhåndteringsstrukturen indsat i en COMAL-løkke, vi tidligere har benyttet, men ikke uddybet; nemlig **LOOP - ENDLOOP** løkken. Den skal sørge for, at INPUT-sætningen udføres igen, hvis der er forekommet fejl.

Bemærk følgende om de forskellige loop-strukturer:

I **WHILE - ENDWHILE** står betingelsen for udførelse forrest i løkken.

I **REPEAT - UNTIL** står betingelsen sidst i løkken.

I **LOOP - ENDLOOP** står betingelsen et vilkårligt sted inde i løkken. Hvis den eventuelle betingelse er opfyldt, springer computeren ud af løkken og fortsætter med linien efter **ENDLOOP**.

Strukturen for **LOOP - ENDLOOP**:

**LOOP**

**EXIT WHEN** <betingelse> (eller blot **EXIT** uden betingelse)

**ENDLOOP**

Programmet består af den generelle indlæsningsprocedure med fejlhåndteringen, og et kort hovedprogram til afprøvning af proceduren.

**Program 20:**

```
new
auto 100

0100 PROC tal'Input(linie,pos,dpos,tekst$,REF tal)
0110 // tal-sikkert input
0120 // kun <STOP> afbryder
0130
0140 LOOP
0150
0160 TRAP
0170
0180 PRINT AT linie,pos: SPC$(LEN(tekst$)+dpos);" "
0190 INPUT AT linie,pos,dpos: tekst$: tal
0200
0210 EXIT //hvis Input er OK
0220
0230 HANDLER
0240
0250 CASE ERR OF
0260 WHEN 2
0270 PRINT AT 24,1: "Tallet var for stort"
0280 WHEN 206
0290 PRINT AT 24,1: "Et TAL forventes "
0300 OTHERWISE
0310 PRINT AT 24,1: "Hvad skete der ? "
0320 ENDCASE
```

```

0330   FOR pause:=1 TO 1000 DO NULL
0340   PRINT AT 24,1: SPC$(20)
0350
0360   ENDTRAP
0370
0380   ENDLOOP
0390
0400 ENDPROC tal'input
0410
0420
0430 // test af inputfejl
0440 PAGE
0450 LOOP
0460   tal'input(10,3,10,"Indtast et tal: ",tal)
0470   PRINT AT 12,3: SPC$(15)
0480   PRINT AT 12,3: tal
0490 ENDLOOP
0500 END

```

#### Bemærk:

- \* Se appendiks F om fejlnumre og fejltekster
- \* Med sætningen **EXIT** beordres computeren til at hoppe ud af LOOP-løkken, hvis input er i orden.
- \* Strengfunktionen **SPC\$(antal'tomme'felter)**.
- \* Linie 180 gør INPUT feltet rent og sætter en \* to pladser efter INPUT feltets afslutning.

#### Øvelser:

1. Afprøv programmet med både tal og ord. Prøv også at taste <RETURN> uden noget input.
2. LOOP-løkken kan erstattes af en REPEAT-løkke. Det kan evt. gøres med programlinierne:

```

ingen'fejl:=FALSE
REPEAT
  ingen'fejl:=TRUE
UNTIL ingen'fejl

```

Hvor skal disse linier indsættes?

3. Erstat CASE-fejtektesterne med systemets fejlmeddelelse ERRTEXT\$: **PRINT AT 24,1: ERRTEXT\$.**
4. Svendepøve:

Tegnet \* i linie 180 er en spidsfindighed. Hvad kan der ske, hvis dette tegn ikke er der? Prøv.

Fortsat god fornøjelse med den nye COMAL-kapsel. Den kan bringe mange udfordringer med sig.

For at hjælpe dig til at klare dem, kommer der nu to kapitler, som er kilder til værdifulde informationer om alle COMAL-faciliteterne (kapitel 4) samt om alle de maskinkode **pakker**, som er til rådighed i din COMAL kapsel (kapitel 5).



## Kapitel 4 -

# COMAL OVERSIGT

### KOMMANDOER, SOM BENYTTES FØR OG UNDER INDTASTNING AF ET PROGRAM:

#### NEW - AUTO - RENUM

##### NEW

er en kommando, som bevirker, at det program og de data, der i øjeblikket er i systemets arbejdslager, slettes. Systemvariabler stilles til opstartværdier. Pakkenavne og tilhørende variabler slettes også.

##### AUTO

er en kommando, som sørger for automatisk linienummerering under indtastning af et program. Lovlige linienumre: 1 - 9999 Under indtastning afsluttes hver linie med <RETURN>, og programmet frembringer næste linies nummer. Den automatiske linienummering afsluttes ved tryk på <RUN/STOP>.

##### Eksempler:

**AUTO** Giver linienumrene: 10, 20, 30,...

**Bemærk:** Hvis et program allerede er i arbejdslageret, vil nummereringen starte ved sidste linienummer + 10.

**AUTO 1000** Giver linienumrene: 1010, 1020,...

**AUTO 100,2** Giver linienumrene: 100, 102, 104,...

##### Bemærk:

Et passende liniemellemrum på f.eks. 10 giver mulighed for at ændre i programmet ved at tilføje linier mellem to allerede eksisterende linienumre.

Hvis et frembragt linienummer eksisterer i forvejen, vil det fremtræde i omvendt skrift. Dermed advares brugeren mod uønskede overskrivninger.

##### RENUM

er en kommando, som giver nye linienumre til programmet i systemets arbejdsområde. Omnummereringen kan starte fra en vilkårlig programlinie.

**Eksempler:****RENUM:** Nye linienumre: 10, 20, 30,...**RENUM 2000,5:** Nye linienumre: 2000, 2005, 2010,...**RENUM 300;4000,10:** Linienumrene fra og med 300 ændres til 4000, 4010,...**KOMMANDOER, SOM BENYTTES TIL  
REDIGERING I PROGRAM:****EDIT - FIND - CHANGE - DEL - SCAN****EDIT**

er en kommando, som bevirker, at programlinier udskrives uden indrykning, én ad gangen. Benyttes særligt under retning i programlinier, som fylder mere end en linie på skærmen. Hvis LIST-ordren anvendes i stedet for, kan linien komme til at indeholde forkert placerede mellemrumstegn. Efter retning i linien trykkes på <RETURN>, og udskriften vil fortsætte med næste linie, hvis flerlinieredigering er angivet.

**Eksempler:**

**EDIT** Redigering af alle linier, én ad gangen  
**EDIT 130** Redigering af linie 130  
**EDIT 210-290** Redigering af linienumrene 210 - 290  
**EDIT farvekoder:** Redigering af proceduren **farvekoder**

**FIND**

er en kommando, som benyttes under redigering til at finde et navn eller en deltekst i et program. Når delteksten er fundet, udskrives systemet programlinien med markøren placeret over deltekstens første bogstav. Efter en eventuel rettelse trykkes på <RETURN>, og systemet vil lede efter næste forekomst af den givne deltekst.

**Eksempler:**

**FIND "Jens"** Systemet gennem søger hele programmet efter ordet **Jens**.  
**FIND 200-500 "John"** Systemet leder efter ordet **John** i linierne 200 - 500.  
**FIND farvekode "sort"** Systemet leder efter ordet **sort** i proceduren **farvekode**.

**CHANGE**

er en kommando, som benyttes til at opsøge og ændre en deltekst. Når delteksten, som ønskes ændret, er fundet, udskrives systemet

den pågældende programlinie med delteksten blinkende som en markør.

Der er nu tre muligheder:

1. Ja, ændringen skal foretages: **Tast <RETURN>**.
2. Linien ønskes omredigeret uden automatisk ændring:  
**Tryk på <C=> tasten.**  
  
Herefter ændres linien som under EDIT-kommandoen.  
**Tryk på <RETURN>**.  
Herefter fortsættes søgningen.
3. Nej, udskiftningen skal ikke foretages her:  
**Tryk på n eller N.**  
Herefter fortsættes søgningen.  
Ved tryk på <STOP> afbrydes CHANGE operationen.

**Eksempler:**

**CHANGE "rød","gul"** Søgeteksten **rød** udskiftes med erstatningsteksten **gul** overalt i programmet.  
**CHANGE 50-200 "x1","xstart"** Udskiftning i linierne 50 - 200  
**CHANGE firkant "op","hen"** Udskiftning i proceduren **firkant**

**DEL**

er en kommando, som benyttes til at slette programlinier.

**Eksempler:**

**DEL 20** Linie 20 slettes.  
**DEL 40,200-280** Linierne 40 og 200 - 280 slettes.  
**DEL udskrift** Proceduren **udskrift** slettes.

**SCAN**

er en kommando, som får systemet til at løbe programmet i arbejdslageret igennem (eng. *prepass*). Programstrukturen efterses, og en eventuel strukturfejl meddeles. Efter et SCAN uden resulterende fejlmeddelelser kan godkendte procedurer og funktioner kaldes som kommandoer.

**Eksempler:**

Indtastet program:

```
0100 tal=0
0110 repeat
0120 print tal
0130 tal:=+2
0140 print "Du så nogle llge tal"
0150 end
```

**SCAN:** Systemet udskriver:  
 at 150: "UNTIL" missing  
 tilføj linien:  
 135 until tal>20

Programudseende efter nyt SCAN:

```
0100 tal:=0
0110 REPEAT
0120 PRINT tal
0130 tal:=tal+2
0135 UNTIL tal>20
0140 PRINT "Du så nogle lige tal"
0150 END
```

## ANDRE KOMMANDOER:

### SETEXEC

er en kommando, som reelt består af to kommandoer:

**SETEXEC-** og **SETEXEC+**, som h.h.v. slår EXEC fra og til.

Ved systemopstart udføres automatisk en **SETEXEC-**, så nøgleordet EXEC ikke udskrives i forbindelse med procedurekald.

Efter en **SETEXEC+** kommando vil EXEC udskrives.

#### Eksempel:

Programstump ved opstart:

```
0100 PRINT "Tal indlæses og udskrives"
0110 indlæsning
0120 udskrift
0130 END
0140
0150 PROC indlæsning
0160 INPUT "Skriv tal: ": tal
0170 ENDPROC indlæsning
0180 PROC udskrift
0190 PRINT tal
0200 ENDPROC udskrift
```

Efter **SETEXEC+**:

```
0110 EXEC indlæsning
0120 EXEC udskrift
```

## KOMMANDOER, SOM BENYTTES TIL OVERSIGT OVER DISKETTEINDHOLD OG LAGERFORBRUG:

### SIZE - CAT - DIR

#### SIZE

er en kommando, som forårsager udskrift af den nuværende fordeling af oktetter i arbejdslageret.

#### Eksempel:

##### SIZE

Systemsvar:

prog	data	fri
13501	02466	14747

hvor **prog** er programmets størrelse, og **data** er den plads, som variable og parametre optager.

#### CAT

er en kommando, som forårsager udskrift af et katalog over diskettens indhold. Hvis flere diskettestationer er tilsluttet, angives stationnummeret i kommandoen.

#### Eksempler:

<b>CAT</b>	Udskrift af alle filnavne
<b>CAT "lst.*"</b>	Udskrift af filer, der starter med <b>lst</b> .
<b>CAT "?est??"</b>	Udskrift af filer, hvis navne består af 6 tegn og 2-4 tegn er <b>est</b>
<b>CAT "*"=seq"</b>	Udskrift af alle sekventielle filer
<b>CAT "2:"</b>	Udskrift af indholdet af disketten på station 2, hvis ekstra diskettestation er tilsluttet. Denne ekstra diskettestation må indstilles som apparat nr. 9. Det kan ved en omskifter (dip switch) inden i diskstationen eller via software. Se dokumentationen for diskettestationen.

#### Bemærk:

Tryk på mellemrumstasten vil standse udskriften, der genoptages ved et nyt tryk på mellemrumstasten.

#### DIR

kan anvendes som kommando eller sætning, der (som CAT) bevir-

ker udskrift af en fortegnelse over indholdet af en diskette. DIR kan optræde som sætning i et program. Det kan CAT ikke.

## LIST - ENTER - MERGE - DISPLAY

### LIST

er en kommando, som benyttes til at udskrive en oversigt (en liste) af programmet i arbejdslageret. LIST benyttes også til at lagre programdele på diskette eller kassette. Programmet gemmes da som en sekventiel fil i ASCII-format. Programkopier, der er gemt med LIST-kommandoen, må senere hentes med ENTER- eller MERGE-kommandoen. De kan IKKE hentes med LOAD.

#### Eksempler:

<b>LIST</b>	Alle programlinier udskrives
<b>LIST 200-400</b>	Programlinierne 200-400 udskrives
<b>LIST 300-</b>	Programmet udskrives fra linie 300
<b>LIST 10,30,100-200</b>	er også mulig
<b>LIST demoproc</b>	Proceduren med navnet <b>demoproc</b> udskrives

Hvis LIST-ordren forsynes med et navn i anførselstegn, vil udskrivningen foregå til diskette/kassette:

<b>LIST "programnavn"</b>	Hele programmet gemmes under navnet <b>programnavn</b>
<b>LIST demoproc "lst.demo"</b>	Proceduren <b>demoproc</b> gemmes under navnet <b>lst.demo</b> . Tilføjelsen <b>lst.</b> er unødvendigt i navnet, men vedhæftes som en påmindelse om, at programmet er gemt med LIST.

#### Bemærkninger:

Udskriften kan gøres langsommere ved at holde <CTRL>-tasten nede under udskriften.

Udskriften kan *standses midlertidigt* med et tryk på mellemrumstasten, og *fortsættes* ved et nyt tryk på mellemrumstasten.

Et tryk på <STOP>-tasten afbryder helt udskriften.

Udskriften kan dirigeres til printer med kommandoen **LIST "lp:"**

Hvis en programlinje strækker sig over flere skærmlinier, vil LIST bevirke, at den brydes ved indrykningen. Placer da markøren på den

linie, som eventuelt skal rettes, og tryk på <CTRL-A>. Linien sammentrækkes derved uden indrykningsbrud.

### ENTER

er en kommando, der henter et program, som tidligere er gemt på diskette/kassette med LIST-kommandoen. **obs:** ENTER bevirker i modsætning til MERGE, at et eventuelt program i arbejdslageret slettes.

#### Eksempler:

<b>ENTER "lst.navn"</b>	Programmet <b>lst.navn</b> hentes fra diskette
<b>ENTER "cs:lst.prog3"</b>	Programmet <b>lst.prog3</b> hentes fra kassette

#### Bemærk:

Et program, som er lagret med SAVE-ordren kan IKKE hentes med ENTER.

### MERGE

er en kommando, som benyttes til at hente programdele fra diskette/kassette og indflette dem i et program i arbejdslageret. Programdelene skal være gemt med LIST-kommandoen.

#### Eksempler:

<b>MERGE "lst.omkreds"</b>	Programmet <b>lst.omkreds</b> hentes ind og placeres med linienumre umiddelbart efter sidste linie i et i forvejen eksisterende program.
<b>MERGE 1000,5 "lst.begynd"</b>	Programmet <b>lst.begynd</b> hentes ind og placeres i linie 1000, 1005, 1010... Pas på overskrivning af allerede eksisterende linier.

### DISPLAY

er en kommando, som bevirker udskrift UDEN LINIENUMRE af et program eller en del af et program.

#### Eksempler:

<b>DISPLAY</b>	Hele programmet udskrives på skærmen
<b>DISPLAY 20-90 "lp:"</b>	Indholdet af linie 20-90 udskrives på printer

**DISPLAY sorter "dsp.sorter"** Indholdet af proceduren **sorter** lagres på diskette under navnet **dsp.sorter**

#### Bemærk:

Et program, der er gemt på diskette eller bånd uden linienumre med DISPLAY-kommandoen, kan ikke hentes med ENTER eller MERGE. Det kan derimod hentes som en almindelig sekventiel ASCII-fil med ordren INPUT FILE.

## SAVE - LOAD

### SAVE

er en kommando, som gemmer en kopi af programmet i arbejdslageret på diskette/kassette i sammenpresset, binær form. Et SAVEd program hentes senere med en af ordrerne LOAD, RUN eller CHAIN.

#### Eksempler:

**SAVE "programnavn"** Programmet gemmes på diskette under navnet **programnavn**.  
**SAVE "cs:racerløb"** Programmet gemmes på kassette under navnet **racerløb**.

#### Bemærk:

Eventuelle programpakker, som er forbundet til COMAL programmet med LINK-ordren, gemmes sammen med COMAL programmet som én fil. Når programmet senere hentes med f.eks. LOAD, hentes automatisk både COMAL programmet og maskinkodepakken.

### LOAD

er en kommando, som henter et program fra diskette eller kassette ind i arbejdslageret. Programmet skal tidligere være gemt med SAVE-kommandoen. LOAD-kommandoen bevirker, at et eventuelt tidligere program og variabler i arbejdslageret slettes.

#### Eksempler:

**LOAD "programnavn"** En kopi af programmet med navnet **programnavn** hentes fra diskette ind i arbejdslageret.  
**LOAD "cs:"** Fra kassettebåndet hentes en kopi af næste program.

## RUN - CHAIN - CON

### RUN

er en kommando, som bringer programmet i arbejdslageret til udførelse. Ved starten nulstilles alle variabler, og programstrukturen undersøges for eventuelle fejl. Et program kan også hentes direkte fra diskette eller kassette og bringes til udførelse med RUN-kommandoen.

#### Eksempler

##### RUN

Programudførelse startes (programmet "køres").

##### RUN "programnavn"

**programnavn** hentes ind fra diskette og bringes til udførelse.

### CHAIN

er en sætning og kommando, som henter et program fra diskette eller kassette og bringer det til udførelse. Et eventuelt eksisterende program slettes fra arbejdslageret.

Brugt som kommando, virker CHAIN "<filnavn>" som RUN "<filnavn>".

Som sætning benyttes CHAIN særligt ved opdeling af et stort program i mindre, uafhængige dele.

#### Eksempler:

##### CHAIN "cs:navn"

Programmet **navn** hentes ind fra kassette og bringes til udførelse

#### Program eksempel:

```
INPUT "Vælg programnummer ":nr
CASE nr OF
  WHEN 1
    CHAIN "program 1"
  WHEN 2
    CHAIN "program 2"
  OTHERWISE
    CHAIN "program 3"
ENDCASE
```

### CON

er en kommando, som medfører, at programudførelse fortsættes i et afbrudt program. Programmet kan være afbrudt af en fejl, ved en STOP sætning eller et tryk på STOP-tasten. Mens programmet er afbrudt, er det tilladt at ændre i indholdet af eksisterende variabelnavne, men man må ikke tilføje nye variabelnavne eller nye linier til programmet. Det er heller ikke muligt at slette eller ændre i pro-

gramlinier under afbrydelsen, hvis programkørsel ønskes fortsat med CON-kommandoen.

### STATUS - STATUS\$

STATUS er en kommando, som får systemet til at udskrive status af diskoperativsystemet og nulstille fejlflaget. STATUS\$ er en streng-funktion, som indeholder statusmeddelelsen. STATUS er det samme som PRINT STATUS\$.

#### Eksempel:

Efter opstart vil kommandoen

#### STATUS

gøre, at systemet svarer

**73,cbm dos v2.6 1541,00,00**

afhængig af diskteststationens type.

### VERIFY

er en kommando, som benyttes til at bekræfte, at programmet på diskette/kassette (gemt med SAVE-kommandoen) er identisk med programmet i computerens arbejdslager.

**Advarsel:** Pas på ikke at ændre i programmet i arbejdslageret, før VERIFY udføres (stav rigtigt!).

#### Eksempel:

**VERIFY "testprog"** COMAL-systemet melder "verify error", hvis ikke det gemte program **testprog** og programmet i arbejdslageret er nøjagtig ens.

## COPY - DELETE - RENAME - PASS

### COPY

er en kommando og sætning, som bruges til at kopiere diskettefiler.

#### Eksempler:

**COPY "gammel'fil","ny'fil"**

Systemet laver en kopi af programmet **gammel'fil** og gemmer den på samme diskstation under navnet **ny'fil**.

**COPY "0:program 3","1:program 3"**

Systemet laver en kopi af **program 3** fra diskstation **0:** og gemmer den på diskstation **1:** under samme navn.

### DELETE

er en kommando og sætning, som sletter en fil på diskette.

#### Eksempel:

**DELETE "testdata"**

Filen **testdata** slettes.

**DELETE "test"**

Alle filer, hvis navn begynder med **test**, slettes.

### RENAME

er en kommando og sætning, som benyttes til at ændre et filnavn.

#### Eksempel:

**RENAME "gammel","ny"**

Diskettefilen med navnet **gammel** tildeles det nye navn **ny**.

### PASS

er en kommando og sætning, som videresender ordrer til diskoperativsystemet.

#### Eksempler:

**PASS "n0:procedurebib,a1"** Klargør (formatterer) en ny diskette på diskstation **0**. Disketten får navnet **procedurebib** og kendenumeret **a1**.

**PASS "n2:disknavn,01",9** En ny diskette formatteres på den ekstra diskteststation (nr. 2) med enhedsnummer **9**.

**PASS "v"** Ryd op: Filerne på disketten samles, og eventuelle åbne filer lukkes. **v** står for det engelske ord *validate*.

#### Bemærk:

Der er flere mulige ordrer, som kan videresendes til diskoperativsystemet ved hjælp af PASS, men for disse findes mere velegnede COMAL-ordrer.

## SELECT INPUT - SELECT OUTPUT

### SELECT INPUT

er en kommando og sætning, der bevirker, at efterfølgende indlæsning, som normalt vil være fra tastaturet, i stedet foregår fra den angivne, sekventielle ASCII-fil. Denne indlæsning kan f.eks. standses ved tryk på <STOP>-tasten eller ved END-OF-FILE og fejl i programmet. Herefter vender input igen tilbage til tastaturet.

**INPUT**-sætninger, **KEY\$** og **Inkey\$** tager også deres input fra **SELECT INPUT** filen. COMAL systemet opfatter endvidere denne indlæsning som om, den kommer fra tastaturet og ekkoer den derfor på sædvanlig vis på tekstskræmen.

Hvis **SELECT INPUT** anvendes som kommando, kan den benyttes til at omdefinere funktionstaster

**SELECT INPUT "kb:"** tastaturinput, som ved opstart eller genstart af COMAL-systemet

**SELECT INPUT "styrefil"** **styrefil** vil blive indlæst, som om den kom direkte fra tastaturet.

### SELECT OUTPUT/SELECT

er en kommando og sætning, som benyttes til at vælge, hvortil efterfølgende udskrift skal foregå. Hvis man blot skriver SELECT, vil systemet automatisk tilføje OUTPUT i programlistningen.

**SELECT OUTPUT "ds:"** udskrift til dataskærmen, som ved opstart af computeren

**SELECT OUTPUT "lp:"** Udskrift dirigeres til printer.

**SELECT OUTPUT "0:navnefil"** En sekventiel fil med navnet **navnefil** oprettes på diskstation 0, og efterfølgende udskrift dirigeres til filen.

#### Bemærkninger:

**SELECT OUTPUT** kan forkortes til **SELECT**. Systemet tilføjer da automatisk **OUTPUT**.

Udskrift vil automatisk vende tilbage til dataskærmen efter LIST-kommandoen.

Selv om udskrift er dirigeret væk fra dataskærmen, vil tekst, angivet i INPUT-sætninger, stadig udskrives på skærmen.

## KOMMANDOER I FORBINDELSE MED SYSTEMOPSTART:

### BASIC - SYS til COMAL

**BASIC**-kommandoen dirigerer computeren til Basic operativsystemet. Computeren dirigeres tilbage til COMAL systemet med ordren **SYS 50000**

Begge ordrer bevirker, at al information i arbejdslageret slettes.

### KOMMANDOER OG SÆTNINGER VEDRØRENDE BRUG AF PROGRAMPAKKER I MASKINKODE

(Se i øvrigt kapitel 8 om COMAL og programmer i maskinkode.):

### USE - LINK - DISCARD

#### USE

er en kommando og sætning, som forbinder en navngiven program-pakke i maskinkode med COMAL-programmet i arbejdslageret. Pakkens navne gøres hermed kendte for COMAL-fortolkeren.

Ordren benyttes f.eks. til at gøre COMAL-kapslens pakker tilgængelige (kapitel 5).

#### Eksempel:

**USE graphics** Pakken **graphics** aktiveres.

#### LINK

er en kommando, som henter en fil med en maskinkodepakke fra diskette ind i arbejdslageret. Pakkens navne gøres kendte med **USE**-ordren.

#### Eksempel:

**LINK "obj.driver"** Objektcode-filen med navnet **obj.driver** hentes.

**USE driver** Den ovenfor LINKede fil indeholder pakken med navnet **driver**, som nu aktiveres.

#### Bemærk:

Et maskinkodeprogram, som er forbundet til et COMAL program med kommandoen **LINK**, gemmes sammen med COMAL programmet som én fil ved **SAVE**-kommandoen. Et senere **LOAD** vil automatisk hente både COMAL programmet og maskinkodeprogrammet.

#### DISCARD

er en kommando, som gør, at alle programpakker i maskinkode fjernes fra arbejdslageret.

COMAL-programmet går ikke tabt, men fortolkerens navnetabel er først intakt igen efter et **RUN** eller **SCAN**.

## SÆTNINGER, SOM BENYTTES UNDER INDLÆSNING OG UDSKRIFT

### INPUT - INPUT AT - KEY\$

#### INPUT

er en sætning, hvormed data indlæses i et program under udførelse. Efter en INPUT-sætning standser systemet og afventer brugerens svar. Markøren blinker ved indtastningsfeltets begyndelse. Alle svar må afsluttes med et tryk på <RETURN>-tasten.

#### Eksempler:

**INPUT "Antal ": tal)** Systemet forventer **tal** som svar.

**INPUT "Hvad hedder du? ": navn\$** Systemet afventer indtastning af en tekst.



**INPUT "Punktet (X,Y) ": x,y**

Flere tal kan indtastes i samme INPUT sætning.

**INPUT "Varenummer: ": nr;**

Et (;) eller (,) efter svarvariablen forhindrer lineskift efter svaret.

### INPUT AT

virker som INPUT med mulighed for at placere svarfeltet på skærmens 25 linier og 40 kolonner.

### Eksempler:

**INPUT AT 4,10: "Nummer ": nr;**

Indlæsningsbeskeden starter på linie 4, kolonne 10.

**INPUT AT 4,7,15: "Navn ": text\$**

Indlæsningsbeskeden starter på linie 4, kolonne 7. Selve indtastningsfeltet afgrænses til de 15 efterfølgende tegn, som isoleres fra det øvrige skærmbillede.

### Specielt:

Et 0 som linie eller kolonne betyder **nuværende**.

### Eksempel:

**INPUT AT 0,0,10: "By ": by\$** Indlæsningsbeskeden starter nuværende linie og kolonne, men med svarfeltet afgrænset til 10 tegn.

Se også INPUT FILE og SELECT INPUT.

### KEY\$

er en funktion, som aflæser indtastningsbufferen for det sidst indtastede tegn. Hvis der ikke er trykket på nogen taste, returnerer den værdien **chr\$(0)** eller **""0""**.

Programafviklingen standses ikke, i modsætning til INPUT-sætningen og funktionen **Inkey\$** i pakken **system**.

### Anvendelseseksempler:

**WHILE KEY\$=CHR\$(0)"0" DO NULL** Programmet "hænger" i samme linie, indtil brugeren trykker på en vilkårlig taste.

**DIM svar OF 1**

**PRINT "Svar ja/nej"**

**REPEAT**

**svar\$=KEY\$**

**UNTIL svar\$ IN "jJnN"**

Systemet afventer et tryk på j, J, n eller N.

## PRINT - PRINT AT - PRINT USING - TAB - ZONE

### PRINT

er en kommando og sætning, som benyttes til at udskrive data på skærmen eller andre udskriftmedier. Hvis PRINT-linien indeholder flere udskriftemner, adskilles disse med et semicolon (;), som giver et mellemrum mellem hvert emne, eller et komma (,), hvis effekt er bestemt af ZONE-ordren. PRINT kan under indtastning forkortes til ;-

### Eksempler:

**PRINT "Resultat: ";fart;"m/s"** Tekst og tal kan sammenblandes i udskriften.

**PRINT**

udskrift af en tom linie

**PRINT tekst\$;** Lineskift forhindres ved at afslutte PRINT-linien med et (;) eller (,).

### PRINT AT

er en kommando og sætning, som gør det muligt at skrive på en vilkårlig tegnposition på skærmen. Linienumre 1 - 25, kolonne 1 - 40.

### Eksempel:

**PRINT AT 3,12: "Navnet er"; navn\$** Udskriften begynder i 3. linie, kolonne 12.

### Specielt:

Et 0 som linie eller position betyder **nuværende**.

### Eksempel:

**PRINT AT 0,30: "COMAL"** Skriv på nuværende linie, kolonne 30.

### PRINT USING

er en kommando og sætning, som benyttes til at udskrive i et veldefineret format.

### Eksempler:

**PRINT USING "Pris ###.##": beløb** beløbet udskrives i formatet bestemt ved #-tegnene og decimal-punktum. Her med plads til 3 cifre før punktum, og 2 cifre efter.

De forskellige PRINT muligheder kan kombineres:

**PRINT AT 10,15: USING "Fart =-#. #": speed****Bemærk:**

Hvis tallet er for stort til at passe i formatet, vil udskriften bestå af stjerner \*\*\*\*\*.

**TAB**

er en systemfunktion, som benyttes i forbindelse med PRINT-ordren. TAB er forkortelse for **TAB**ulator.

**Eksempel:**

**PRINT "Varenummer: ",TAB(25),nr** Efter at teksten **Varenummer:** er udskrevet, vil systemet flytte markøren til kolonne 25, hvor **nr** udskrives.

Se også PRINT FILE og SELECT OUTPUT.

**ZONE**

er en sætning og funktion, som i forbindelse med kommategnet (,) benyttes til at definere intervaller i PRINT-udskrifter. Ved opstart og efter hver NEW (eller RUN) kommando er ZONE lig 0.

**Eksempler:**

Lige efter opstart:

**PRINT 23,56,89** vil udskrives som  
**235689** uden mellemrum, da ZONE er lig 0.  
**ZONE 5** Intervaller sættes til 5.  
**PRINT 23,56,89** vil nu udskrives som  
**23 56 89** Det første tal startende i kolonne 1, det næste i kolonne 6, det næste i 11 osv.  
**huskzone:=ZONE** ZONE kan benyttes som funktion til f.eks. at tildele variablen **huskzone** den nuværende ZONE-værdi.

**PAGE - CURSOR****PAGE**

er en kommando og sætning, som benyttes til at slette skærm billedet. Hvis udskrift er valgt til en printer, føres papiret en side frem.

**CURSOR**

er en kommando og sætning, som benyttes til at placere markøren

(eng. cursor) på skærmen. 1,1 er i øverste, venstre hjørne og 25,40 i nederste, højre hjørne.

**Eksempler:**

**CURSOR 15,30** Placer markøren på linie 15, kolonne 30.

**CURSOR 0,10** Flyt markøren til nuværende linie, kolonne 10. Et 0 betyder **nuværende**.

Bemærk angivelsen af placeringen på skærmen. CURSOR, INPUT AT og PRINT AT benytter Linie, Kolonne princippet, i modsætning til højopløsningsgrafikken, hvor punkter angives med X,Y-koordinater.

**READ - DATA - RESTORE - Etiket: - EOD****READ**

er en sætning, som benyttes til at læse værdier fra en DATA-sætning. Hvis READ-sætningen indeholder flere variabelnavne, adskilles disse med et komma (,).

**Eksempel:**

**READ navn\$,gade\$,nr,postnr,by\$**

**DATA**

er en sætning, der indeholder de værdier, som variabelnavnene tildeles i en READ-sætning. DATA-sætninger udføres ikke, og kan derfor placeres hvor som helst i et program. Der må dog tages hensyn til, at en DATA-sætning er lokal inden i en lukket procedure eller funktion.

**Eksempler:**

DATA-sætningen kan indeholde både tekst og talværdier. Tekst omsluttet af anførselstegn ":

**DATA "Jens Jensen","Nygade",12,8700,"Horsens"**

**DATA 230,Se6,%11100110**

En DATA-sætning kan indeholde både decimaltal, hexadecimale tal og binære tal.

**RESTORE**

er en kommando og sætning, som sætter DATA-pilen til at pege på den første DATA-sætning i et program. Hvis RESTORE anvendes

sammen med en **etikette** vil DATA-pilen pege på den første sætning lige efter **etiketten**.

#### Etikette:

er et vilkårligt navn (eng. **Label**), som benyttes til at navngive et sted i programmet. Etiketten udføres ikke som en selvstændig ordre, men kan benyttes i forbindelse med RESTORE (og GOTO). Se sammenfattende eksempel efter EOD.

#### EOD

er en logisk systemfunktion, som benyttes under READ-indlæsning fra DATA-sætninger. EOD betyder **End Of Data**. Så længe, der er DATA-værdier tilbage i listen, er EOD lig FALSE. Når sidste DATA-værdi er læst, sættes EOD lig TRUE. DATA-værdier kan være lokale i en lukket procedure eller funktion.

#### Sammenfattende eksempel:

```
DATA "skruer",112,"søm",50
legetøj:
DATA "biler",220,"dukker",35
DATA "bolde",76,"sjippetove",24
DIM navn$ of 20
RESTORE legetøj
WHILE NOT EOD DO
  READ navn$,antal
  PRINT "Der er";antal;navn$;"tilbage"
ENDWHILE
```

#### Bemærk:

DATA-sætningerne står først i programmet, så de er nemme at få øje på og udskifte.

En etikette **legetøj**: er sat før DATA sætningerne indeholdende angivelse af legetøj.

**RESTORE legetøj** sikrer, at READ begynder i den efterfølgende linie.

Indlæsning og udskrift af legetøjsbeholdningen fortsætter, indtil EOD sættes til TRUE, når der ikke er flere DATA-værdier tilbage.

## KOMMUNIKATION MED FILER

### MOUNT - CREATE

#### MOUNT

er en kommando og sætning, som initialiserer en diskette, der netop er sat i diskettestationen. Kassettebånd behøver ikke initialiseres.

For en sikkerheds skyld bør det være en vane at MOUNT disketter for hver udskiftning.

#### Eksempler:

<b>MOUNT</b>	Diskettestation initialiseres (samme som <f2>)
<b>MOUNT "1:"</b>	Diskettestation 1 initialiseres

#### CREATE

er en kommando og sætning, som opretter en fil med direkte tilgang på diskette. En fil kan også oprettes under åbningen med en OPEN-ordre, men kommunikationen med filen forløber ca. 10 gange hurtigere, når filen er skabt ved hjælp af CREATE i forvejen.

#### Eksempel:

**CREATE "varer",300,42** Der oprettes en fil med navnet **varer**, med 300 poster, hver af 42 tegns (okteters) længde

## OPEN FILE/OPEN - READ - WRITE - APPEND - RANDOM

#### OPEN FILE/OPEN

er en kommando og sætning, som benyttes til at åbne adgang til en fil på en ydre enhed, f.eks. diskette, kassette, printer m.m. Der kan være flere sekventielle filer åbne på samme tid, men med forskellige strømnumre. Man kalder det for strømnummer, fordi der åbnes for en datastrøm til eller fra filen. Hvis man under indtastning undlader ordet FILE, tilføjes det automatisk af systemet.

Der er mange måder at åbne filer på. Se kapitel 6. Her anføres kun nogle få eksempler på måder med brug af READ, WRITE, APPEND og RANDOM.

#### Eksempler:

**OPEN FILE 3, "datafil",WRITE**

Filen med navnet **datafil** og **strømnummeret 3** åbnes for **tilskrivning** af data. Strømnummeret 3 er hermed reserveret til denne fil, indtil den lukkes med **CLOSE FILE 3**.

**OPEN FILE 7, "cs:navne",READ**

Kassettefilen **navne** åbnes for **læsning**. Filen identificeres ved **strømnummeret 7**.

**OPEN FILE 15,"data",APPEND**

En allerede eksisterende, sekventiel diskfil med navnet **data** åbnes for **tilføjelse** af nye data efter de eksisterende. Filen identificeres ved **strømnummeret 15**.

**OPEN FILE 5,"text", RANDOM 42**

Filen **text** åbnes. RANDOM angiver, at det er en fil med **direkte tilgang** til dens poster. Hver post vil fylde 442 bytes på disketten, selv om den ikke fyldes helt ud med tegn.

**OPEN FILE 4,"lp:", WRITE**

En fil åbnes på **printer**.

**PRINT FILE - INPUT FILE****PRINT FILE**

er en kommando og sætning, som benyttes til udskrift af data i ASCII-format til en tidligere åbnet (OPEN-ordren) fil på diskette, kassette eller anden enhed. Filen identificeres ved dens strømnummer.

Ved udskrift med PRINT FILE adskilles de enkelte dataelementer med **<CR>**, dvs. ASCII-kode 13.

Filer, som er tilskrevne med PRINT FILE, læses med ordren INPUT FILE.

**Eksempler:****PRINT FILE 2: vare\$**

Variablens indhold udskrives på den sekventielle fil med strømnummeret 2. Udskriften afsluttes med **<CR>** efter **vare\$**. Filen er åbnet med OPEN 2,...,WRITE eller APPEND.

**PRINT FILE 4,7: navn\$**

Variablens indhold udskrives til filen med **strømnummer 4, post nummer 7** på fil med **direkte tilgang** (åbnet med RANDOM).

**PRINT FILE 4,7,30: navn\$**

Variablens indhold udskrives med start i post 7, position 30. Hvis ingen position er angivet, startes forfra i posten.

**INPUT FILE**

er en kommando og sætning, der bruges til at indlæse data fra en fil, som er åbnet ved OPEN nr,navn\$,READ eller RANDOM. Filen skal indeholde data i ASCII-format, skrevet med PRINT FILE-ordren.

**INPUT FILE 2: vare\$****Eksempler:**

Variablens værdier indlæses fra den sekventielle fil med strømnummeret 2. Filen skal være åbnet som READ type

**INPUT FILE 4,7: navn\$**

Variablens værdi indlæses fra fil 4, post 7. Filen skal være åbnet som RANDOM type.

**WRITE FILE - READ FILE****WRITE FILE**

er en kommando og sætning, som udskriver data i kompakt, binær format til en fil. Filen er sekventiel, hvis den er åbnet som WRITE eller APPEND type; og har direkte tilgang, hvis den er åbnet som RANDOM. WRITE FILE er at foretrække, hvor det er muligt, fremfor PRINT FILE, da den binære form fylder mindre, og tilgangen er hurtigere. Det er ikke muligt at benytte WRITE FILE til lagring af data på kassette.

**Eksempler:****WRITE FILE 2: for\$,efter\$,tilf**

Variablernes værdier udskrives i binær form til filen med strømnummeret 2. Filen må tidligere være åbnet med ordren OPEN 2,...,WRITE, APPEND eller RANDOM. Hvis RANDOM, fortsættes fra den i forvejen adresse-rede post.

**WRITE FILE 3: tabelværdier()**

Hele talsættet **tabelværdier()** udskrives på fil 3.

**WRITE FILE 4,12: nr,tekst\$,andets**

Variablernes værdier udskrives i binær form til en fil med direkte tilgang. Strømnummer 4, post 12. Filen skal tidligere være åbnet som OPEN 4,...,RANDOM.

**READ FILE**

er en kommando og sætning, der bruges til at indlæse data fra en fil, som tidligere er åbnet med ordren **OPEN nr,navn\$,READ** eller **RANDOM**. Filen skal indeholde data i binær form, indskrevet med ordren WRITE FILE.

**Eksempler:****READ FILE 2: for\$,efter\$,tilf**

Dataværdierne indlæses fra den sekventielle fil med strømnummeret 2. Filen skal tidligere være åbnet som READ type.

**READ FILE 4,12: nr,tekst\$,andets\$**

Dataværdierne indlæses fra fil nr 4, post 12. Filen skal tidligere være åbnet som RANDOM type med direkte tilgang.

**CLOSE FILE/CLOSE**

er en kommando og sætning, der lukker filer, som er åbnet med OPEN-ordren. Der kan opstå alvorlige fejl, hvis man forsøger at kopiere eller på andre måder omarrangere åbne filer. Hvis man under indtastning undlader ordet FILE, anbringes det automatisk af systemet.

**Eksempler:**

**CLOSE** Alle åbne filer (bortset fra SELECT'ede filer) lukkes.

**CLOSE FILE 2** Filen med **strømnummer 2** lukkes.

**EOF**

er en logisk systemfunktion, som benyttes under indlæsning fra en fil. EOF står for "End of File". EOF skal anføres med angivelse af strømnummer: **EOF(<strømnummer>)**. Så længe, der er elementer tilbage i filen, har EOF værdien FALSE (=0). Når sidste element er læst, sættes EOF lig TRUE (=1). EOF = CHR\$(255).

**Eksempel:**

```
nr:=0
WHILE NOT EOF(2) DO
  nr:=1
  READ FILE(2): tal(nr)
ENDWHILE
```

Data indlæses fra filen med strømnummeret 2. Indlæsningen stopper, når der ikke er flere elementer tilbage.

**UNIT - UNITS****UNIT**

er en kommando og sætning, som bruges til at definere hvilken sta-

tion, der skal benyttes ved filbehandling, hvis ikke filnavnet indeholder et valg af station. Ved opstart vælges diskettestation nummer 0 automatisk som enhed. Se i øvrigt kapitel 7 om Ydre Enheder.

Man kan vælge mellem lagerenheder:

**cs:** kassette  
**0:** diskettestation nr 0 (som ved opstart)  
**1:** diskettestation nr 1  
**2:** ekstra diskettestation (enhed 9)

**Eksempel:**

**UNIT "cs:"** kassette er primær enhed

**UNITS**

er en systemfunktion, der udskriver navnet på den enhed, som benyttes, hvis ikke andet er angivet i filnavnene.

**Eksempel:**

**PRINT UNITS** Systemet svarer f.eks. med 0:

**SÆTNINGSSTRUKTURER**

Betingelsessætninger  
 Gentagelsessætninger  
 Fejlhåndtering  
 Procedurer og funktioner

**BETINGELSESSÆTNINGER****IF - THEN - ELIF - ELSE - ENDIF**

er sætninger, som bruges i IF-THEN strukturer. En IF-THEN sætning kan formuleres på mange forskellige måder, men grundreglen er fast: Hvis et <logisk udtryk> er **sandt**, så udføres de tilhørende sætninger. Tit siger man om det samme, at hvis en given <betingelse> er **opfyldt**, så udføres de tilhørende sætninger.

**Eksempel 1:**

**IF <logisk udtryk> THEN <sætning>**

er en enkeltlinieveersion: Hvis det <logiske udtryk> er sandt, udføres <sætningen> efter THEN. Ellers fortsættes blot med næste linie.

**IF svar\$="ja" THEN udskriv data**

**Eksempel 2:**

```

IF <logisk udtryk> THEN
  <sætning>
  <sætning>
  ...
ENDIF

```

Flerlinieversion: Hvis udtrykket er sandt, udføres sætningerne mellem THEN og ENDIF. Ellers hoppes til linien efter ENDIF.

```

IF tal>=0 THEN
  kvadratrod:(SQR=tal)
  PRINT "Kvadratrod af";tal;"er";kvadratrod
ENDIF

```

**Eksempel 3:**

```

IF <logisk udtryk> THEN
  <sætning>
  ...
ELSE
  <sætning>
  ...
ENDIF

```

Hvis udtrykket er sandt, udføres sætningerne mellem THEN og ELSE. Ellers udføres sætningerne mellem ELSE og ENDIF.

```

IF svar$ IN "æiouyæå" THEN
  PRINT svar$;"er en vokal"
  PRINT "Hvad med et nyt forsøg?"
ELSE
  PRINT svar$;"er ikke en vokal"
  PRINT "Bogstaverne: æiouyæå er vokaler"
  PRINT "alle andre bogstaver er konsonanter"
ENDIF

```

**Eksempel 4:**

```

IF <betingelse1> THEN
  <sætning>
  ...
ELIF <betingelse2> THEN
  <sætning>
  ...
ELIF <betingelse3> THEN
  <sætning>
  ...
ENDIF

```

ELIF er forkortelse for ELSE IF. Hvis <betingelse1> er opfyldt, udføres sætningerne mellem THEN og første ELIF, hvorefter programudførelse fortsætter efter ENDIF. Hvis <betingelse1> ikke er opfyldt, undersøges <betingelse2>. Hvis <betingelse2> er opfyldt, udføres sætningerne ned til næste ELIF, hvorefter udførelsen hopper til linien efter ENDIF. Ellers undersøges <betingelse3> osv.

```

IF tal=0 THEN
  tilføj'data
ELIF tal=1
  slet'data
ELIF tal=2
  udskriv'data
ENDIF

```

**Eksempel 5:**

```

IF <betingelse1> THEN
  <sætning>
  ...
ELIF <betingelse2> THEN
  <sætning>
  ...
ELSE
  <sætning>
  ...
ENDIF

```

Hvis ingen af betingelserne er opfyldte, udføres sætningerne mellem ELSE og ENDIF.

```

IF a$="post" AND b$="hus" THEN
  PRINT "Ja, netop"
  PRINT "Ordet skulle være ";a$+b$
ELIF a$="hus" AND b$="post" THEN
  PRINT "Prøv at bytte om på ordene"
ELSE
  PRINT "Ordene passede ikke sammen"
  PRINT "Se på tegningen igen"
  PRINT "og forsøg endnu en gang"
ENDIF

```

**CASE - OF - WHEN - OTHERWISE - ENDCASE**

er sætninger, som benyttes i **CASE-strukturen** til at dirigere programudførelse i en situation med mange valgmuligheder.

**Eksempel:**

```

CASE <udtryk> OF
  WHEN <1. værdi>
    <sætning>
    ...
  WHEN <2. værdi>
    <sætning>
    ...
  WHEN <3. værdi>
    <sætning>
    ...
  OTHERWISE (kan udelades)
    <sætning>
    ...
ENDCASE

```

**Eksempel 1:**

```

CASE svar OF
WHEN 1
  PRINT "Hm..."
WHEN 2
  tegn'linie
WHEN 3,4
  tegn'polygon
OTHERWISE
  tegn'cirkel
ENDCASE

```

Alt afhængig af **sva**r's værdi udføres de forskellige procedurer. Hvis svar er lig 1, 2, 3 eller 4, udføres sætningerne under det tilhørende WHEN. Ellers udføres sætningerne under OTHERWISE. Strukturen afsluttes med ENDCASE.

**Eksempel 2:**

```

CASE TRUE OF
WHEN nævner>0
  PRINT "Nævneren er positiv."
WHEN nævner=0
  PRINT "Pas på!"
  PRINT "Nævneren er nul."
WHEN nævner<0
  PRINT "Nævneren negativ."
  PRINT "Fortegnet skifter!"
  nævner:nævner*(-1)
ENDCASE

```

**GENTAGELSESSÆTNINGER****REPEAT - UNTIL**

er sætninger, som benyttes i **REPEAT-strukturen**. Sætningerne inden i REPEAT-UNTIL løkken gentages, indtil det logiske udtryk i UNTIL-sætningen er sandt.

**Eksempel 1:**

```

REPEAT <sætning> UNTIL <logisk udtryk>

```

er en enkeltlinieversion: <sætning> udføres, indtil <logisk udtryk> er sandt.

```

REPEAT læs'fil UNTIL tekst$="Jens" OR EOF(nr)

```

Proceduren læs'fil vil blive udført, indtil det logiske udtryk er sandt. Her er **udtrykket**, at variabelen **tekst\$** er lig "Jens", eller **EOF(nr)** er sand (når der ikke er mere tekst på den læste fil).

**Eksempel 2:**

```

REPEAT
  <sætning>
...
UNTIL <logisk udtryk>

```

Flerelinieverson: sætningerne mellem REPEAT og UNTIL udføres, indtil det logiske udtryk er sandt.

```

REPEAT
  INPUT "Nyt tal ": a
UNTIL a<0

```

**WHILE - DO - ENDWHILE**

er sætninger, som benyttes i **WHILE-strukturen**. Sætningerne inden i WHILE-ENDWHILE løkken gentages, så længe det logiske udtryk i WHILE-sætningen er sandt.

**Eksempel 1:**

```

WHILE <logisk udtryk> DO <sætning>

```

er en enkeltlinieverson: Så længe <logisk udtryk> er sandt, udføres <sætning>.

```

WHILE navn$<>"Ole" DO gæt'navn

```

Så længe **navn\$** er forskellig fra "Ole", gentages kaldet af proceduren **gæt'navn**.

**Eksempel 2:**

```

WHILE <logisk udtryk> DO
  <sætning>
...
ENDWHILE

```

Så længe <logisk udtryk> er sandt, udføres sætningerne mellem DO og ENDWHILE.



```

b:=1
WHILE KEY$=CHR$(0) DO
  b:=2*b
  PRINT 1/b
ENDWHILE

```

Så længe ingen taste er trykket, udskrives nye tal i talrækken.

Bemærk, at nøgleordet ENDWHILE ikke må benyttes i enkeltlinieversionen.

## FOR - TO - STEP - DO - ENDFOR

er sætninger, som benyttes i **FOR - ENDFOR strukturen**. Sætningerne inden i FOR-løkken gentages et forud bestemt antal gange, hvorefter programudførelsen fortsætter med linien efter ENDFOR. Løkkevariablen <tæller> er lokal.

### Eksempel 1:

```
FOR <tæller>:=<start> TO <slut> DO <sætning>
```

er en enkeltlinieversion: Løkken gennemløbes <slut>-<start>+1 gange med <tæller> lig <start>, <start>+1, ..., indtil <slut> passerer.

```
FOR n:= 1 TO 30 DO PRINT a(n);
```

### Eksempel 2:

```

FOR <tæller>:=<start> TO <slut> DO
  <sætning>
...
ENDFOR <tæller>

```

```

FOR nr:=1 TO 10 DO
  INPUT "Navn : ":navn$(nr)
  INPUT "tekst: ":tekst$(nr)
ENDFOR nr

```

FOR-løkken gentages 10 gange med variablen nr lig 1,2,...,10.

### Eksempel 3:

Version med STEP-parameter:

```

FOR vinkel:=0 TO 6.3 STEP 0.1 DO
  PRINT COS(vinkel);SIN(vinkel)
  PRINT COS(vinkel)^2=SIN(vinkel)^2
ENDFOR vinkel

```

Som angivet ved STEP-parametren, vil vinkel antage værdierne 0, 0.1,...,6.3.

```

FOR I#:=max TO min STEP -1 DO
  moveto(0,0)
  drawto(x(I#),y(I#))
ENDFOR I#

```

Heltalsvariablen I# øger hastigheden. STEP-parametren kan også være negativ.

### Bemærk:

Nøgleordet ENDFOR benyttes ikke i enkeltlinieversionen. Enkeltlinieversionen kan også bruges som kommando.

## LOOP - EXIT - EXIT WHEN - ENDLOOP

er sætninger, som benyttes i **LOOP-ENDLOOP strukturen**. Sætningerne i LOOP-løkken gentages, indtil en EXIT eller EXIT WHEN sætning udføres. Derefter fortsætter programudførelse med linien efter ENDLOOP. Der kan være 0, 1 eller flere EXIT's i en LOOP-løkke.

### Eksempel:

```

LOOP
  <sætning>
...
EXIT WHEN <logisk udtryk>
  <sætning>
...
ENDLOOP

LOOP
  INPUT "Tekst ": tekst$
  EXIT WHEN tekst$="slut"
  WRITE FILE 3: tekst$
  udfør'test
ENDLOOP

```

Tekst indlæses, udskrives på fil 3 og undersøges i proceduren udfør'test, indtil teksten "slut" indlæses.

## FEJLHÅNDTERING

### TRAP - HANDLER - ENDTRAP

er sætninger, som benyttes til at styre den videre programafvikling efter eventuelle fejl. Hvis der indtræder fejl i sætningerne mellem TRAP og HANDLER (TRAP-delen), udføres sætningerne mellem HANDLER og ENDTRAP (HANDLER-delen). Ellers fortsættes med linien efter ENDTRAP. Man undgår altså, at programmet standser på et ubejligt tidspunkt.

**Eksempel:**

```

TRAP
  INPUT "Nr. ":nr
  HANDLER
    check'fejl
  ENDTRAP

```

Hvis der opstår fejl i indlæsningen, vil systemet hoppe ned i HANDLER delen og udføre proceduren **check'fejl**.

**ERR - ERRFILE - ERRTEXT\$**

er systemfunktioner, som bruges i forbindelse med **TRAP-strukturens HANDLER del** til at identificere opståede fejl. Se appendiks F om fejlnumre og fejlttekster.

**ERR** indeholder fejlnummeret.

**ERRFILE** indeholder nummeret på en eventuel fil, der var i brug, da en indlæsnings-eller udskrivnings fejl opstod.

**ERRTEXT\$** indeholder teksten med fejlmeddelelsen.

**Eksempel 1:**

```

TRAP
  INPUT "Eksponent ":eksp
  PRINT 10^eksp
  HANDLER
    PRINT ERRTEXT$
    CASE ERR OF
      WHEN 2
        PRINT "Eksponent for stor"
      WHEN 206
        PRINT "Eksponent er et tal"
      OTHERWISE
        PRINT "Forsøg igen"
    ENDCASE
  ENDTRAP

```

**Eksempel 2:**

```

TRAP
  INPUT "Filnavn: ":navn$
  OPEN FILE 2,navn$,READ
  OPEN FILE 3,"gemmefil",WRITE
  overfør(navn$,"gemmefil")
  HANDLER
    CLOSE
    IF ERRFILE=2 THEN
      PRINT "Fejl i indlæsningen"
    ELIF ERRFILE= THEN
      PRINT "Fejl under udskrift"
    ELSE
      PRINT "Ikke input/output fejl"

```

```

ENDIF
PRINT ERR,ERRTEXT$
ENDTRAP

```

**REPORT**

er en kommando og sætning, som bruges i forbindelse med **TRAP-strukturen**. REPORT kan benyttes på flere måder til at frembringe en fejl og dirigere den videre fejlhåndtering. REPORT kan forekomme både med og uden argument:

<b>REPORT</b>	Forrige fejl gentages (kun som sætning).
<b>REPORT fejlnr</b>	Meld en fejl med fejlnr.
<b>REPORT fejlnr,fejlttekst\$</b>	Meld fejlnr og fejlttekst\$.

Ordren har forskellig effekt, alt afhængig af, hvor den optræder i strukturen.

REPORT **udenfor TRAP-ENDTRAP** strukturen: Fejlen meldes til systemet, som så vil reagere på fejlen.

REPORT i strukturens **TRAP del**: Programudførelsen dirigeres til HANDLER delen, hvor bruger-programmet behandler fejlen.

REPORT i strukturens **HANDLER del**: Programudførelse dirigeres til en ydre HANDLER struktur, hvis en sådan findes. Ellers meldes fejlen til systemet med fejlmeddelelse på skærmen.

**Eksempel:**

```

TRAP
  INPUT "Navn : ":navn$
  INPUT "Alder: ":alder
  HANDLER
    IF ERR OR ERR=2 OR ERR=206 THEN
      alder:=0
    ELSE
      REPORT
    ENDIF
  ENDTRAP

```

REPORT kan genkalde fejl: Hvis der ved **alder** ikke er svaret med et tal, eller tallet er for stort, sættes alder lig 0. Ellers meldes fejlen til systemet.

**GOTO - <Etikette>**

**GOTO** er en sætning, som får programudførelsen til at fortsætte et nærmere angivet sted. Dette sted angives ved en <Etikette>. Dvs. et navn efterfulgt af et kolon (:). Man kan ikke med GOTO hoppe ud af en procedure eller ind i en sammenhængende struktur.

**Eksempel:**

```
FOR nr:=1 TO 10 DO
  READ FILE 2: tal
  IF tal<1e-37 THEN GOTO for'lille
  PRINT 1/tal
ENDFOR nr
for'lille:
PRINT "Divisor for lille"
```

**<Etikette>**

er et navn (eng. *label*), som bruges til at identificere en programlinje. Programlinjen udføres ikke. Udførelsen fortsætter i linjen efter <etiketten>. Etiketter benyttes i forbindelse med GOTO og RESTORE.

**Eksempler:**

Se **GOTO** eksempel.

```
DATA 2,4,5,2,1
toclfre:
DATA 12,34,18,54,22
RESTORE toclfre
WHILE NOT EOD
  READ tal(nr),
ENDWHILE
```

Indlæsningen af tal fra DATA-sætningerne starter med tallet **12** på grund af sætningen **RESTORE toclfre**.

**PROCEDURER****PROC - ENDPROC**

er sætninger, som benyttes i **PROC-ENDPROC** strukturen til at omslutte en række sætninger, der tilsammen udgør en **procedure**. En procedure er et delprogram, som kendetegnes ved et navn i procedurehovedet: **PROC** <navn>. Proceduren udføres kun, hvis den kaldes et andet sted fra med samme navn, som angives i PROC-hovedet.

COMAL programmer bør opbygges af procedurer, der i deres simpleste form blot benyttes til en opdeling af et større program i mindre, overskuelige enheder. Mere avancerede anvendelser med

parameteroverførsel og brug af mulighederne REF, CLOSED, IMPORT og EXTERNAL gør procedurer til et virkelig stærkt programmeringsværktøj.

**Eksempel 1:**

```
// HOVEDPROGRAM
<sætning>
...
<navn1>
<sætning>
...
<navn2>
<sætning>
...
<navn1>
<sætning>
...
END // HOVEDPROGRAM

PROC <navn1>
<sætning>
...
ENDPROC <navn1>
PROC <navn2>
<sætning>
...
ENDPROC <navn2>
```

Procedurens sætninger omslutes af **PROC** <navn> og **ENDPROC** <navn>. Proceduren kaldes fra forskellige steder i hovedprogrammet "med navns nævnelse".

<pre>// HOVEDPROGRAM opstart indlæsning  PROC opstart   USE system   textcolors(0,2,1)   DIM tal(10)   PAGE ENDPROC opstart  PROC indlæsning   FOR nr:=1 TO 10 DO     PRINT "Indlæs alder(",nr,") ",     INPUT "": tal(nr)   ENDFOR nr ENDPROC indlæsning</pre>	<p>Hovedprogrammet består af to programlinjer, som hver kalder en procedure</p>
---	---

**Eksempel 2:**

```

<sætning>
...
udskrift(medlem,alder,navn$)
<sætning>
...
...
PROC udskrift(nr,år,tekst$)
  PRINT
  PRINT "Medlemsnummer: ",nr
  PRINT "Alder      : ",år
  PRINT "Navn      : ",tekst$
ENDPROC udskrift

```

**Bemærk om eksempel 2:**

I hovedprogrammet kaldes proceduren **udskrift**. De værdier, som er indeholdt i de aktuelle parametre **medlem**, **alder** og **navn\$**, overføres til de formelle parametre **nr**, **år** og **tekst\$**, som optræder i procedurehovedet.

Variabelnavnene på de formelle parametre er lokale i proceduren **udskrift**.

Denne form for værdioverførsel er én-vejs: Man kan kun føre værdier ind i proceduren, ikke ud.

**Bemærk om procedurer:**

Når en procedure har været RUNnet eller SCANnet, kan den benyttes som kommando.

En procedure kan kalde en anden procedure, endog sig selv.

En procedure kan indsættes i en anden procedure og dermed gøres lokal inden for netop denne procedure. Tilsvarende vil en funktion og en etikette være lokal inden i en procedure/funktion.

Kommandoen **SETEXEC**+ vil bevirke, at ethvert procedurekald begynder med ordet **EXEC**. Se **SETEXEC**.

**REF - CLOSED - IMPORT****REF**

er en parametertype, som bruges ved procedurekald. Et REF foran en parameter i procedurehovedet angiver, at navnet kun skal være et synonym for det tilsvarende navn i procedurekaldet. Der afsættes altså ikke plads i computerens arbejdslager til et nyt navn med tilhørende værdi. Værdien får blot et nyt, foreløbigt navn. Begge navne refererer til samme værdi. Hermed spares lagerplads, hastigheden øges og parameterværdier kan føres begge veje, både ind og ud af proceduren.

**Eksempel:**

```

<sætning>
...
indlæs(klasse,navn$())
<sætning>
...
PROC indlæs(REF nr,REF a$())
  INPUT "Hvilken klasse: "; nr
  PRINT "Skriv elevnavne."
  i:=0
  REPEAT
    i:=i+1
    INPUT "Navn: ";a$(i)
  UNTIL a$(i)=""
ENDPROC indlæs

```

Mens proceduren **indlæs** udføres, vil navnene **klasse** og **nr** referere til samme værdi på grund af REF foran **nr**. Tilsvarende gælder for navnene **navn\$** og **a\$**. Begge refererer til strengværdierne i en én-dimensional tabel.

**CLOSED**

er en ordre, som benyttes til at erklære alle variabelnavne i en procedure for lokale. Det vil sige, at proceduren "lukkes af" fra det øvrige program, på nær overførsel af parameterværdier i procedurehovedets parentes. Derved forhindres sammenblanding og navnekonflikt mellem procedures navne og navne i det øvrige program. Et navn kan f.eks. bruges lokalt i proceduren uden at anfægte indholdet af en variabel med samme navn uden for proceduren.

**Eksempel:**

```

<sætning>
...
minmax(10,tal(),min,max)
PRINT min;max
<sætning>
...
PROC minmax(n,a(),REF b,REF c) CLOSED
  b:=a(1);c:=a(1)
  FOR i:=2 TO n DO
    IF a(i)<b THEN b:=a(i)
    IF a(i)>c THEN c:=a(i)
  ENDFOR i
ENDPROC minmax

```

Proceduren **minmax** er lukket med **CLOSED**, så den kan benyttes uden at tage hensyn til dens variabelnavne.

**IMPORT**

er en sætning, som benyttes i lukkede procedurer til at indføre

variabler, procedurer og funktioner udefra. Dermed gøres de tilgængelige til brug i den ellers lukkede procedure.

#### Eksempel:

```
<sætning>
...
udskriv(points())
<sætning>
...
PROC udskriv(tal()) CLOSED
IMPORT antal, t(), sorter
DIM prod(antal)
FOR nr#:=1 TO antal DO
  prod(nr#):=tal(nr#)*t(nr#)
  PRINT nr#;prod(nr#)
ENDFOR nr#
sorter(tal(),antal)
sorter(t(),antal)
FOR nr#:=1 TO antal DO
  PRINT nr#;tal(nr#)*t(nr#)
ENDFOR
ENDPROC udskriv
```

Selvom proceduren **udskriv** er lukket, gøres variablen **antal**, tabellen **t()** og proceduren **sorter** tilgængelige ved hjælp af **IMPORT**-sætningen.

## EXTERNAL - MAIN

### EXTERNAL

er et nøgleord, der bruges til at tilkendegive, at en given procedure er en **ydre procedure**, som må hentes fra diskette, når den skal bruges i programmet. Når man laver en procedure, der tænkes benyttet som EXTERNAL procedure, skal den lukkes med CLOSED ordren og gemmes med kommandoen SAVE. Den gemte procedure kan senere hentes ind fra diskette og bruges i et andet program, hvis den i dette program erklæres for EXTERNAL. På denne måde er det muligt at opbygge et menustyret program, som benytter eksterne procedurer. Procedurene kan så hentes ind i arbejdslageret efterhånden som de benyttes.

#### Eksempel:

```
PROC test(a,b$,REF check) CLOSED
IF a=0 AND b$ IN "abcd" THEN check:=TRUE
ENDPROC test
```

Proceduren **test** er lukket og gemmes på diskette med kommandoen **SAVE test "ext.test"**.

Den kan senere benyttes i et andet program.

```
// Programstart
<sætning>
...
test(nr,tekst$,fejl)
<sætning>
...
PROC test(nr,tekst$,REF fejl) EXTERNAL "ext.test"
// Programslut
```

Dette program vil hente proceduren **test** ind fra diskette, bruge den og glemme den igen.

Linien med EXTERNAL erklæringen kan anbringes i programmet, hvor det findes mest hensigtsmæssigt.

### MAIN

er en kommando, som bruges til at bringe systemet tilbage til hovedprogrammet, hvis det skulle standse under udførelse af en EXTERNAL procedure. Hvis udførelsen stoppes i den ydre procedure, vil LIST og andre editeringsordrer virke på den ydre procedure, indtil MAIN fjerner den og bringer hovedprogrammet tilbage.

## FUNKTIONER

### FUNC - ENDFUNC - RETURN

er sætninger, som benyttes i **FUNC-ENDFUNC strukturen**. Strukturen består af en række sætninger, der tilsammen udgør en **brugerdefineret funktion**. Funktioner skal indledes med **FUNC <navn>** og afsluttes med **ENDFUNC <navn>**. Den værdi, som funktionen returnerer, skal være angivet i **RETURN**-sætningen.

Funktioner kan være reelle funktioner, heltalsfunktioner eller strengfunktioner. En funktion udføres kun, hvis den kaldes et andet sted fra med det samme navn, som angives i funktionshovedet (**FUNC <navn>**).

For funktioner gælder, at de kan forsynes med alle de egenskaber, som gør **procedurer** værdifulde: **REF**, **CLOSED**, **IMPORT** og **overførsel af parametre**. Se også under disse nøgleord i kapitel 4. Endvidere er funktioner benyttet i kapitel 3 og i appendiks C og E.

Specielt gælder, at alle funktioner efter at være strukturercheckede (med SCAN eller RUN) kan kaldes fra direkte kommandoer.

**Eksempel 1:**

```
// Hovedprogram
// reel funktion
<sætninger>
PRINT gennemsnit(a,b)
<sætninger>

FUNC gennemsnit(x,y)
RETURN (x+y)/2
ENDFUNC gennemsnit
```

**Eksempel 2:**

```
// Hovedprogram
// heltalsfunktion
<sætninger>
første#:=vokaler#("COMAL")
anden#:=vokaler#("og funktioner")
<sætninger>
FUNC vokaler#(tekst$) CLOSED
antal#:=0
FOR i#:=1 TO LEN(tekst$) DO
IF tekst$(i#) IN "aeiouyæøåAEIOUYÆØÅ" THEN antal#:+1
ENDFOR i#
RETURN antal#
ENDFUNC vokaler
```

**Eksempel 3:**

```
// Hovedprogram
// strengfunktion
<sætninger>
PRINT mystisk$("hemmeligt")
<sætninger>
FUNC mystisk$(a$)
dobbel:= 2*LEN(a$)
DIM b$ OF 1, c$ OF dobbelt
c$:=a$
FOR i:=1 TO dobbelt STEP 2 DO
b$:=CHR$(RND(65,93))
c$:=c$(i)+b$+c$(i+1:)
ENDFOR i
RETURN c$
ENDFUNC mystisk$
```

**Eksempel 4:**

```
PRINT snup$(0,"klip en hæl")
FUNC snup$(først,a$)
længde:=LEN(a$)
IF længde>1 THEN
IF først THEN
```

```
RETURN a$(2:)
ELSE
RETURN a$(længde-1)
ENDIF
ELSE
RETURN ""
ENDIF
ENDFUNC snup$
```

Hvis **først**<>0, fjerner funktionen det første bogstav i den anførte sætning. Hvis **først**=0, fjernes det sidste bogstav.

**ANDRE FUNKTIONER****ABS - INT - SGN - SQR - PI****ABS**

er en funktion, som udregner et udtryks absolutte værdi. Også kaldet udtrykkets numeriske værdi. Er udtrykkets talværdi negativ, skifter talværdien fortegn. ellers ikke

**Eksempler:**

```
ABS(3.25)    giver 3.25
ABS(-7.46)   giver 7.46
ABS(tal-7)   resultatet afhænger af værdien af tal.
```

**INT**

er en funktion, som udregner den hele del af værdien af et udtryk; dvs. det største hele tal, som er mindre end eller lig det angivne udtryks talværdi.

**Eksempler:**

```
INT(3.25)    giver 3
INT(-7.46)   giver -8
INT(1/2)     giver 0
```

**SGN**

er en funktion, som har værdien +1, 0 eller -1, når talværdien af et angivet udtryk er henholdsvis positiv, nul eller negativ.

**Eksempler:**

```
SGN(327.54)  giver +1
SGN(-45.7)   giver -1
SGN(0)       giver 0
SGN(x/7-y)   resultatet afhænger af x og y.
```

**SQR**

er en funktion, som svarer til kvadratroden af et ikke negativt tal.

**Eksempler:**

<b>SQR(16)</b>	giver 4
<b>SQR(4.9e x 09)</b>	giver 70000
<b>SQR(x<sup>2</sup>+y<sup>2</sup>)</b>	resultatet afhænger af x og y.

**PI**

er en systemkonstant, som er tildelt værdien 3.14159266. PI benyttes ved angivelse af vinkel mål i radianer, hvor PI radianer svarer til 180 grader.

**COS - SIN - TAN - ATN****COS**

er en funktion, som udregner cosinus til et tal. Dette tal må udtrykkes i radianer.

---

**X grader = X\*PI/180 radianer**  
**X radianer = X\*180/PI grader**

---

**Eksempler:**

<b>COS(PI/2)</b>	giver 0
<b>COS(2.5)</b>	giver -0.801143616
<b>COS(v*PI/180)</b>	resultatet afhænger af værdien af v

**SIN**

er en funktion, som udregner sinus til et tal. Dette tal må udtrykkes i radianer. Se under COS.

**Eksempler:**

<b>SIN(PI/6)</b>	giver 0.5
<b>SIN(vinkel)</b>	resultatet afhænger af værdien af vinkel

**TAN**

er en funktion, som udregner tangens til et tal. Dette tal må udtrykkes i radianer. Se under COS.

**Eksempler:**

<b>TAN(-PI/4)</b>	giver -1
<b>TAN(1.8)</b>	giver -4.28626168

**ATN**

er en funktion, som udregner **arcus tangens** (invers tangens) til et tal. Resultatet er et tal, udtrykt i radianer.

**Eksempler:**

<b>ATN(1)</b>	giver 0.785398163 (PI/4)
<b>ATN(-200)</b>	giver -1.56579637

**LOG - EXP****LOG**

er en funktion, som udregner den naturlige logaritme til et positivt tal. LOG betegner altså logaritmefunktionen med grundtal e, hvor e med god tilnærmelse er lig 2.71828183. LOG er den omvendte funktion til EXP.

**Eksempler:**

<b>LOG(1)</b>	giver 0
<b>LOG(10)</b>	giver 2.30258509
<b>LOG(-2)</b>	er ikke defineret
<b>LOG(EXP(x))</b>	giver x

**EXP**

er en funktion, som udregner eksponentialfunktionsværdien af et tal. EXP(x), e i x'te, hvor e er grundtallet for den naturlige logaritme-funktion. EXP er den omvendte funktion til LOG.

e = 2.71828183 er en god tilnærmelse.

**Eksempler:**

<b>EXP(1)</b>	giver 2.71828183 (= e)
<b>EXP(3)</b>	giver e i 3. = 20.0855369
<b>EXP(t-a*2)</b>	resultatet afhænger af t og a
<b>EXP(LOG(x))</b>	giver x

**CHR\$ - STR\$ - SPC\$****CHR\$**

er en strengfunktion, som er lig det tegn, der svarer til argumentets ASCII nummer. Den omvendte operation udføres med funktionen ORD.

Se appendiks A for C64 ASCII koder.

**Eksempler:**

<b>CHR\$(65)</b>	giver tegnet <b>a</b>
<b>CHR\$(147)</b>	giver tegnet <b>slet skærm</b>
<b>CHR\$(&lt;tal&gt;)</b>	resultatet afhænger af <b>tal</b>
<b>CHR\$(ORD("B"))</b>	giver tegnet <b>B</b>

**STR\$**

er en strengfunktion, som omdanner et numerisk udtryk til en streng. Den omvendte funktion udføres med funktionen **VAL**.

**Eksempler:**

<b>STR\$(1.34)</b>	giver strengen "1.34"
<b>STR\$(2-5)</b>	giver strengen "-3"
<b>STR\$(VAL("7"))</b>	giver strengen "7"

**SPC\$**

er en strengfunktion, som returnerer det angivne antal blanke tegn (»tomme pladser«).

**Eksempler:**

<b>PRINT "1",SPC\$(10),"2"</b>	<b>Mellem 1 og 2 udskrives 10 mellemrum</b>
<b>tekst\$ = "a" + SPC\$(8) + "jk"</b>	<b>tekst\$ sættes lig "a jk"</b>
<b>blanke\$ = SPC\$(LEN(navn\$))</b>	<b>blanke\$ består af lige så mange mellemrum, som der er tegn i navn\$.</b>

**ORD - VAL - LEN****ORD**

er en funktion, hvis værdi er ASCII-værdien af det første tegn i den angivne streng. Den omvendte operation udføres med funktionen **CHR\$**.

Se appendiks A for C64 ASCII værdier.

**Eksempler:**

<b>ORD("F")</b>	giver 198
<b>ORD("dreng")</b>	giver 68
<b>ORD(by\$)</b>	resultatet afhænger af by\$
<b>ORD(CHR\$(8))</b>	giver 8

**VAL**

er en funktion, som omdanner en lovlig streng til dens tilsvarende talværdi. For at være lovlig skal strengen bestå af tallene 0,...,9, for-

tegn =-, decimalpunktum . eller e i angivelse af eksponentiel notation. Den omvendte operation udføres med funktionen **STR\$**.

Hexadecimal og binær notation er tilladt.

**Eksempler:**

<b>VAL("123")</b>	giver tallet 123
<b>VAL("2"+"3")</b>	giver tallet 23
<b>VAL("4e12")</b>	giver tallet 4e+12
<b>VAL("abe")</b>	ulovligt
<b>VAL(STR\$(2))</b>	giver tallet 2
<b>VAL("\$fe")</b>	giver tallet 254

**LEN**

er en funktion, hvis værdi er længden af den angivne streng.

**Eksempler:**

<b>LEN("abcd")</b>	giver tallet 4
<b>LEN(navn\$)</b>	resultatet afhænger af navn\$
<b>LEN("")</b>	giver tallet 0
<b>LEN("a ki")</b>	giver tallet 5

**TRUE - FALSE****TRUE**

er en systemkonstant, som altid er lig 1.

**FALSE**

er en systemkonstant, som altid er lig 0.

**TIME**

er en kommando, sætning og funktion, som bruges i forbindelse med systemets indbyggede ur.

Uret angiver tiden i jiffies.

---

1 sekund = 60 jiffies.  
 1 døgn = 5184000 jiffies  
 (Herefter nulstilles uret)

---

**TIME** kan bruges til at stille uret, eller til at aflæse tiden siden forrige nulstilling.



**Eksempler:****TIME 0**

Uret nulstilles.

**TIME 3600**

Uret stilles på 3600 jiffies, dvs. 1 minut.

**sek:=INT(TIME/60)****sek** sættes lig antal sekunder siden sidste nulstilling.**RANDOMIZE - RND****RANDOMIZE**

er en kommando og sætning, som bruges til at stille generatoren af tilfældige tal et tilfældigt sted i talrækken. De tilfældige tal frembringes med funktionen **RND**.

**Eksempler:****RANDOMIZE,**

Begyndelsesplaceringen i talrækken bestemmes af tiden i jiffies siden sidste **TIME**. Da antallet af jiffies (1/60 sek.) må antages at være helt tilfældigt, vil placeringen være tilfældig.

**RANDOMIZE 6**

Hvis **RANDOMIZE** efterfølges af et tal, vil dette tal angive startstedet, hver gang tal frembringes fra den tilfældige række. Dette gør, at den samme talrække frembringes hver gang **RND** benyttes.

**RND**

er en funktion, som frembringer et tilfældigt reelt tal fra en talrække af ligeligt fordelte, »tilfældige« tal.

Med **RANDOMIZE** stilles generatoren et tilfældigt sted i denne talrække, således at tallene virkeligt er tilfældige.

**Eksempler:****tal:=RND**

Et tilfældigt reelt tal mellem 0 og 1 frembringes.  $0 \leq \text{RND} < 1$

**nr:=RND(-10,30)**

Et tilfældigt af tallene -10,-9,...,29,30 frembringes.

**PRINT RND(min,max)**

Et tilfældigt helt tal mellem **min** og **max** udskrives.

**ESC - TRAP ESC**

er nøgleord, som har betydning for, hvordan **<STOP>** tasten virker.

**ESC** er en systemfunktion, hvis værdi afhænger af, om sætningen **TRAP ESC+** eller sætningen **TRAP ESC-** er udført:

Hvis **TRAP ESC+** er udført (hvad der f.eks. automatisk sker ved opstart af systemet) vil et tryk på **<STOP>** tasten afbryde programudførelse. **ESC** funktionen har ingen betydning.

Hvis **TRAP ESC-** er udført, vil et tryk på **<STOP>** tasten IKKE afbryde programmet. **ESC** vil have værdien **FALSE**, indtil der trykkes på **<STOP>** tasten. Derefter vil den forblive **TRUE**, indtil værdien af **ESC** aflæses i programmet.

**Eksempelsekvens:****TRAP ESC-**

**<STOP>** tasten sættes ud af funktion og **ESC** tildeles værdien **FALSE**. **ESC** sættes lig **TRUE**. **ESC** sættes lig **FALSE**. **<STOP>** tasten gøres funktionsdygtig.

**<STOP> trykkes ned:****dummy:=ESC****TRAP ESC+****OPERATORER**

Se appendiks C for en mere detaljeret gennemgang af operatorer.

**DIV - MOD****DIV**

er en operator, som giver den heltallige kvotient ved division. **x DIV y** er det samme som **INT(x/y)**.

**Eksempler:****5 DIV 2**

giver 2

**74 DIV 10**

giver 7

**(x+3) DIV y**

resultatet afhænger af x og y.

**MOD**

er en operator, som udregner **resten** efter division. **x MOD y** er det samme som **x-INT(x/y)\*y**.

**Eksempler:****5 MOD 2**

giver 1

**74 MOD 10**

giver 4

**8.25 MOD 2.1**

giver 1.95

**(4-x) MOD z**

resultatet afhænger af x og z.

## LOGISKE OPERATORER

### NOT - AND - AND THEN - OR - OR ELSE

#### NOT

er en logisk operator, som skifter sandhedsværdien af et udtryk.

#### Sandhedstabel:

a	NOT a
TRUE	FALSE
FALSE	TRUE

#### Eksempler:

```
WHILE NOT EOF(2) DO
  READ FILE 2: tal
  PRINT tal;
ENDWHILE
```

Løkken gennemløbes, indtil der ikke længere er data på filen med strømnummeret 2.

#### IF NOT ok THEN læs'status(ok)

Proceduren **læs'status** udføres, indtil variablen **ok** antager værdien TRUE (<>1).

#### AND

er en logisk operator, som bestemmer sandhedsværdien af et sammensat udtryk, **a AND b**. Det sammensatte udtryk er kun sandt, hvis begge deludtryk er sande.

#### Sandhedstabel:

a	b	a AND b
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

#### Eksempler:

7=2 AND 3=3

giver værdien FALSE

### WHILE udtryk1 AND udtryk2 DO lav'tegning

Hvis både **udtryk1** og **udtryk2** er sande, udføres proceduren **lav'tegning**. Ellers udføres den ikke.

#### AND THEN

er en logisk operator, som er en udvidelse af operatoren AND: **a AND THEN b**. Der gælder samme regler for AND THEN som for AND; men hvis det første udtryk **a** er falsk, udregnes udtrykket **b** ikke, da det allerede er givet, at det samlede udtryk vil være falsk.

#### Eksempel:

```
a$="test";i:=1
længde:=LEN(a$)
WHILE i<=længde AND THEN a$(i)<>"." DO i:=i+1
```

For i:=5 vil der opstå kørsels-fejl i det logiske udtryk **a\$(i)<>".**", hvis ikke dette tilfælde bortcensureres af den første betingelse.

#### OR

er en logisk operator, som bestemmer sandhedsværdien af et sammensat udtryk, **a OR b**. Det sammensatte udtryk er sandt, hvis blot et af udtrykkene **a** eller **b** er sande.

#### Sandhedstabel:

a	b	a OR b
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

#### Eksempler:

7=2 OR 3=3: giver værdien TRUE.

```
REPEAT
  <sætning>
```

```
UNTIL nr>4 OR svar$ IN "J"
```

Sætningerne i REPEAT-løkken gentages, indtil **nr>4** eller **svar\$** er et J eller et j.

## OR ELSE

er en logisk operator, som er en udvidelse af operatoren OR: **a OR ELSE b**. Der gælder samme regler for OR ELSE som for OR; men hvis det første udtryk **a** er sandt, udregnes udtrykket **b** ikke, da det allerede er givet, at det samlede udtryk vil være sandt.

### Eksempel:

**IF a#=0 OR ELSE b/a#>100 THEN nyt'problem**

Hvis **a#** er lig 0, er det første logiske udtryk sandt. I dette tilfælde er det overflødigt at vurdere det sidste udtryk, som ville indebære en ulovlig division.

## IN

er en operator, som angiver positionen af en søgestreng i en given tekst: **søgestreng IN tekst**.

Værdien er nummeret i teksten af søgestrengens første tegn. Hvis søgestrengen ikke findes, angives værdien 0.

**IN** kan derfor bl.a. benyttes til at vurdere, om et svar er indeholdt i acceptable svarmuligheder.

### Eksempler:

**x:="gram" IN "programmering"**      **x** får værdien 4.

**PRINT "male" IN "Comalprogram"**      0 udskrives.

**IF svar\$ IN "nN" THEN STOP** Hvis **svar\$** består af bogstavet **n** eller **N**, er udtrykket sandt, og programmet stopper.

### Specielt eksempel:

Hvis søgestrengen er tom, dvs. lig "", angiver **IN** **tekstens længde + 1**.

**lang:="" IN "Comal til CBM"**      **lang** får værdien 14.

## BITAND - BITOR - BITXOR

### BITAND

er en boolesk operator, som udfører et AND på hvert bit i den binære repræsentation af to tal: **a BITAND b**.

Alle tal, som skal sammenlignes med operatorene BITAND, BITOR eller BITXOR, skal være hele tal i intervallet 0-65535, dvs. binært mellem %0000000000000000 og %1111111111111111.

### Regler:

BITAND	a	b	00	01	10	11	Eks %	1	0	0
								AND	AND	AND
	00		00	00	00	00	%	1	1	0
	01		00	01	00	01				
	10		00	00	10	10	%	1	0	0
	11		00	01	10	11				

### Eksempler:

**%0011 BITAND %0101**      giver %0001 (decimalt lig 1)

**17 BITAND 18**      giver 16

**\$fe BITAND 5**      giver 4

### IF PEEK(userport) BITAND %1100 THEN lav'registrering

Hvis indholdet i hukommelsesadressen **userport** har bitmønsteret **%00001100**, så udføres proceduren **lav'registrering**.

## BITOR

er en boolesk operator, som udfører et OR på hvert enkelt bit i den binære repræsentation af to tal, **a BITOR b**.

### Regler:

BITOR	a	b	00	01	10	11	Eks. %	1	0	1
								OR	OR	OR
	00		00	01	10	11	%	1	0	0
	01		01	01	11	11				
	10		10	11	10	11	%	1	0	1
	11		11	11	11	11				

### Eksempler:

**%1010 BITOR %0110**      giver %1110 (decimalt lig 14)

**23 BITOR \$1b**      giver 31

**BITXOR**

er en boolesk operator, som udfører et XOR (dvs. et "eksklusivt eller") på hvert enkelt bit i den binære repræsentation af to tal, **a BITOR b**.

**Regler:**

BITXOR	a	b	00	01	10	11	Eks. %	1	0	1
								XOR	XOR	XOR
	00		00	01	10	11	%	1	0	0
	01		01	00	11	10				
	10		10	11	00	01	%	0	0	1
	11		11	10	01	00				

**Eksempler:**

**%0011 BITXOR %1010:** giver %1001 (decimalt lig 9)  
**17 BITXOR 8** giver 25.

**ANDRE ORDRER**

//

er en sætning, som muliggør kommentarer i et program. Kommentarsætningen udføres ikke, men bruges i programmet til at forklare dets virkemåde. Det gør programmet nemmere forståeligt for senere programmører.

Kommentarlinier optager plads i lageret, men forøger ikke et programs udførelsestid.

**Eksempler:**

// grafikvinduet slettes

**a\$=b\$(1)+b\$(LEN(b\$)) //a\$=b\$'s første og sidste tegn**

**TRACE**

er en kommando, som bruges til at spore de procedurekald eller funktionskald, der er aktive. **TRACE** kan f.eks. benyttes til at finde årsagen til en opstået fejl.

**Eksempel:**

Et program tænkes standset i en procedure i linie 740 på grund af en fejl:

**TRACE**

programmet standsede i

0740 a\$=tegn\$(1:3)

inden i

0700 PROC udskrift(nr,tegn\$)

som blev kaldt i

0030 udskrift(2,"k")

**DIM**

er en kommando og sætning, som benyttes til at **afsætte plads** i lageret **til tabeller**, indeholdende tal eller tekst.

Som sætning anføres den normalt som en global, indiceret variabel i begyndelsen af et program, men kan også benyttes lokalt i en lukket procedure.

**Tabeller med tal:**

**DIM tabel(50)**

Tabellen kan indeholde reelle tal med indeks 1, 2,...,50.

**DIM x(20),y(20)**

En DIM-sætning kan indeholde flere tabeller, adskilt med et komma (,).

**DIM point(-10:20)**

Tabel med indeks -10,-9,...,0,...,20

**DIM plads(10,40,40)**

Tre-dimensional tabel

**DIM pris(0:100,5:10)**

To-dimensional tabel med indices 0,...,100 og 5,...,10

**Bemærk:**

Hvis tabellen i DIM sætningen ikke indeholder en nedre indeksgrænse, sættes den til 1.

Ved oprettelsen i DIM sætningen sættes alle tabelværdier lig 0.

**Tabeller med strenge:**

**DIM navn\$ OF 30**

Der afsættes plads til 30 tegn i strengen **navn\$**.

**DIM vare\$(10) OF 20**

Der afsættes plads til 10 **vare\$**-navne. Hvert navn må indeholde højst 20 tegn.

**DIM tekst\$(0:10,2:5) OF 80**

**tekst\$** er en to-dimensional tabel af ord på højst 80 tegn.

**Bemærk:**

Første gang en streng tildeles en værdi, oprettes plads i lageret til 40 tegn, hvis den ikke forinden er erklæret med en DIM sætning.

Ved opstart sættes alle strenge lig den tomme streng, "".

## PEEK - POKE

### PEEK

er en funktion, som henter indholdet i en angiven lageradresse. Resultatet er et helt tal mellem 0 og 255. Et kort med angivelse af aktuelle lageradresser findes bl.a. i kapitel 8 og i "Commodore 64, Programmers Reference Guide", side 310-334.

#### Eksempler:

**linie:=PEEK(214)**      Fra adresse 214 hentes det linie-nummer, som markøren står på

**PRINT PEEK(\$dd00)**      Skriv indholdet af Paralleporten.

### POKE

er en kommando og sætning, som bruges til at anbringe tal direkte ned i en lageradresse, POKE adresse,tal.

Man må være forsigtig med brugen af POKE, da forkerte tal til uhensigtsmæssige adresser kan få computeren til at opføre sig mærkeligt!

#### Eksempler:

**POKE 198,0**      Tælleren på tastaturets buffer nulstilles. Dvs. bufferen tømmes.

**POKE \$dd03,%11110000:**      Parallellportens retningsregister har den hexadecimale adresse \$dd03. Adressen får indholdet %11110000, som sætter bit 0-3 til indgange og bit 4-7 til udgange.

### SYS

er en kommando og sætning, der dirigerer programudførelsen hen til en maskinkode subrutine, som begynder i den angivne adresse.

#### Eksempel:

**SYS 4000**      udfør maskinkoderutinen med start-adresse 4000.

**SYS 50000**      Systemet foretager en COMAL opstart (normalt fra Basic).

### NULL

er en kommando eller sætning, som bruges til at gøre **ingenting!** Den kan f.eks. bruges i pauser og andre steder, hvor man ønsker, at computeren skal vente.

#### Eksempler:

**FOR pause:=1 TO 1000 DO NULL**  
**WHILE KEYS=CHRS(0) DO NULL**

## STOP - END

### STOP

er en sætning, som bruges til at standse udførelsen af et program.

**STOP** kan placeres hvor som helst i et program, og der kan være flere STOP-sætninger i et program. Efter at programmet er standset, kan variabelers indhold aflæses og eventuelt ændres. Med kommandoen **CON** kan programmet derefter bringes til at fortsætte med linien efter STOP-sætningen. Blot må der ikke foretages programændringer.

#### Eksempler:

**STOP**      Programmet stopper med meddelelsen: **STOP at XXXX**

**STOP "udskrift slut"**      Programmet stopper med meddelelsen: **udskrift slut.**

### END

er en sætning, der helt standser programudførelse og markerer afslutning på et program. **END** kan placeres hvor som helst i et program. I modsætning til STOP kan programmet ikke fortsættes med CON kommandoen.

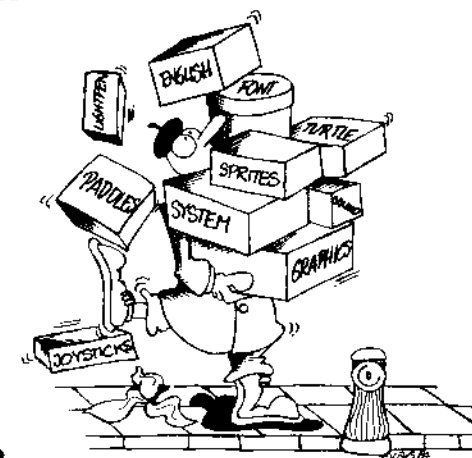
#### Eksempler:

**END**      Programmet afsluttes med meddelelsen: **END at XXXX**

**END "Slut finale"**      Programmet afsluttes med meddelelsen: **Slut finale**

## Kapitel 5 -

# COMAL PAKKER



### HVAD ER EN PAKKE?

I COMAL kapslen er anbragt 11 programpakker med nyttige procedurer, skrevet i maskinkode. De skal hjælpe dig til endnu bedre at udnytte de muligheder, der er med din Commodore 64 og COMAL.

En pakke (og dermed dens indbyggede procedurer og funktioner) gøres tilgængelig med kommandoen eller sætningen:

**USE <pakkenavn>**

hvor **pakkenavn** er en af de 11 nedenstående betegnelser.

Når en pakke er aktiveret, kan dens procedurer og funktioner kaldes med navn, ligesom almindelige COMAL procedurer. Alle procedurer kan benyttes som kommandoer, såvel som sætninger i programmer. Man må gerne aktivere flere pakker på samme tid.

### Oversigt over pakker:

- |                    |  |
|--------------------|--|
| 1. <b>english</b>  | , engelske fejlmeddelelser             |
| 2. <b>dansk</b>    | , danske fejlmeddelelser               |
| 3. <b>graphics</b> | , procedurer til X-Y grafik            |
| 4. <b>turtle</b>   | , procedurer til turtle (Logo) grafik  |
| 5. <b>sprites</b>  | , procedurer til bevægelige figurer    |
| 6. <b>sound</b>    | , procedurer til styring af lyd        |
| 7. <b>system</b>   | , procedurer til konfigurationsændring |

- 8. **font** , procedurer til definition af tegnsæt
- 9. **paddles** , procedure til aflæsning af paddles
- 10. **joysticks** , procedure til aflæsning af joysticks
- 11. **lightpen** , procedurer til kontrol af lyspen

## PAKKEN ENGLISH

### USE english

Alle COMAL fejlmeddelelser vil være på engelsk. Ved opstart udføres kommandoen **USE english** automatisk. Derfor starter computeren med at meddele sig på engelsk. Pakken indeholder ingen procedurer.

## PAKKEN DANSK

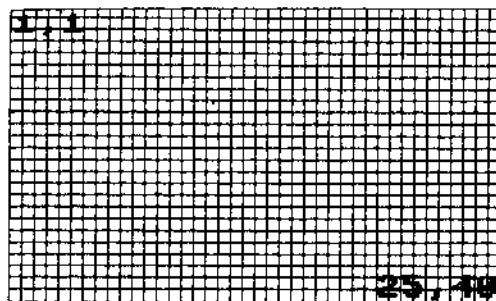
### USE dansk

Alle COMAL fejlmeddelelser vil være på dansk. Pakken indeholder ingen procedurer.

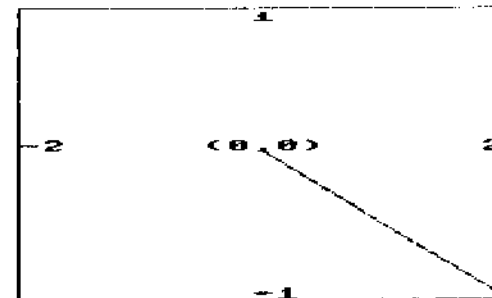
## GRAFIK MED COMAL

På Commodore 64 kan man arbejde med to forskellige skærme: en tekstskeerm og en grafiskskeerm.

Det skal forstås sådan, at computeren i sit lager har to "landkort": et over, hvordan tekstskeermen ser ud, og et over, hvordan grafiskskeermen ser ud. Kun ét af disse kort kan vises som skærmbillede ad gangen.



Normalt er det tekstskeermen, der vises. Den består af 25 linier; hver med plads til 40 tegn. Position 1,1 er i øverste, venstre hjørne og position 25,40 er nederste, højre hjørne. Tekstskeermen har altså i alt  $25 \times 40 = 1000$  forskellige pladser. På hver kan der placeres bogstaver, tal, grafiske symboler eller andre tegn.



Grafiskskeermen består af  $320 \times 200 = 64000$  prikker. 320 prikker vandret og 200 prikker lodret. Punkterne er nummererede i et koordinatsystem med koordinater (X,Y). Selve den fysiske billedskeerm har altid punktet (0,0) i nederste, venstre hjørne og (319,199) i øverste, højre hjørne.

De procedurer og funktioner, som bruges til at tegne på den grafiske skærm, gøres tilgængelige med ordren:

### USE graphics

eller med

### USE turtle

## GRAFIKSKÆRMEN

Man kan arbejde med grafiskskeermen på to måder:

- graphicscreen(0)** , højopløsningsgrafik
- graphicscreen(1)** , flerfarve grafik

Begge ordre gør grafiskskeermen synlig på billedskeermen og gemmer tekstskeermen til senere brug. Forskellen på de to måder er antallet af mulige farvekombinationer. Se **graphicscreen** ordren for uddybning.

Benyt høj opløsning, hvis du vil lave tegninger med tynde streger og kun én farve foruden baggrundsfarven.

Hvis flere farver er vigtigere end detaljer, benyttes flerfarvegrafik tilstanden.

Et program, der tegner en gul ramme omkring billedskeermen, kan se således ud:

```
USE graphics
graphicscreen(1)
pencolor(7)
drawto(319,0)
drawto(319,199)
```

```
drawto(0,199)
drawto(0,0)
WHILE KEYS=CHRS(0) DO NULL
```

Programmets sidste linie fastholder grafikskærmen som billede, idet computeren afventer, at en eller anden taste trykkes ned. Så snart en taste er nedtrykket, slutter programmet. Tekstskærmen vises igen, mens grafikskærmen skjules.

---

Efter at ordren **USE graphics** nu har været udført, kan du med funktionstasterne <f1> og <f5> vælge hvilket billede, du vil se:

```
<f1>      , tekstskærmen vises
<f5>      , grafikskærmen vises
```

Desuden kan funktionstasten <f3> bruges. Hvis en grafikpakke ikke er aktiveret, vil et tryk på tasten bevirke, at kommandoen **USE turtle** udføres med delt skærm:

```
<f3>      , delt skærm: grafikskærm med 4 linier
           , rullende tekst foroven
```

Et tryk på <CTRL-u> slår skiftevis disse tasters grafikvirkning fra eller til.

---

Med COMAL grafik er du ikke bundet til at arbejde med koordinater fra (0,0) til (319,199). Oven på skærbilledet kan du med ordren **window** lægge dit eget koordinatsystem. Alle grafikordrer på nær ordren **viewport** og **sprite**-ordrerne vil adlyde dette koordinatsystem.

#### Programeksempel:

```
USE graphics
graphicscreen(1)
window(-2,2,-1,1)
moveto(0,0)
drawto(2,-1)
WHILE KEYS=CHRS(0) DO NULL
```

Ordren **window(-2,2,-1,1)** lægger et koordinatsystem over billedskærmen. Punktet (-2,-1) er nu skærmens nederste, venstre punkt, og (2,1) det øverste, højre punkt.

Ved opstart med ordren **USE graphics** vælges koordinatsystemet **window(0,319,0,199)**, som stemmer overens med billedskærmens koordinater. Ordren **USE turtle** udfører automatisk **window(-160,159,-100,99)**, så (0,0) vil ligge midt på skærbilledet.

Hvis du vil skrive tekst på grafikskærmen, må du benytte den særlige skriveordre: **plottext**.

#### Eksempel:

```
USE graphics
graphicscreen(1)
plottext(0,100,"COMAL grafik")
WHILE KEYS=CHR(0) DO NULL
```

I kapitel 3 findes flere eksempler på anvendelse af grafikprocedurer. Yderligere eksempler findes på disketten og kassetten, der følger med COMAL kapslen.

## GRAFIK OVERSIGT

Pakkerne **graphics** og **turtle** indeholder disse ordre:

#### Definition af grafisk arbejdsområde:

**viewport - window**

#### Valg af grafikskærm og farvegrafik tilstand:

**graphicscreen**

#### Valg af billedskærm:

**textscreen - fullscreen - splitscreen**

#### Sletning af grafisk skærbillede:

**clearscreen - clear**

#### Farvevalg:

**textcolor - textbackground - textborder**

**pencolor - background - border**

**getcolor**

#### (X,Y) grafik:

**plot**

**drawto - moveto**

**draw - move**

**setxy**

**circle - arc**

**xcor - ycor**

#### Intelligent farve udfyldning:

**fill - paint**

#### Relativ grafik:

**showturtle - hideturtle**

**turtlesize**

**home**

**setheading - heading**

**penup - pendown**

**left - right**

**forward - back**

**arc1 - arc**

#### Tekst på grafikskærmen:

**textstyle - plottext**



Oplysning om grafisk tilstand:  
inq

Gemme og hente det grafisk billede:  
savescreen - loadscreen  
printscreen

Desuden er det i pakken **turtle** tilladt at forkorte nedenstående procedurenavne:

bk	=	back
bg	=	background
cs	=	clearscreen
fd	=	forward
ht	=	hideturtle
lt	=	left
pc	=	pencolor
pd	=	pendown
pu	=	penup
rt	=	right
seth	=	setheading
st	=	showturtle
textbg	=	textbackground

## UDDYBNING:

**viewport(<vxmin>,<vxmax>,<vymin>,<vymax>)**  
er en procedure, som afgrænser det område på skærbilledet, hvor man kan definere et koordinatsystem og tegne.

Parametrene <vxmin>, <vxmax>, <vymin> og <vymax> refererer altid til selve det fysiske skærbillede med (0,0) i nederste, venstre hjørne og (319,199) i øverste, højre hjørne; uafhængigt af om der med **window** er valgt et andet koordinatsystem.

### Eksempel:

**viewport(0,159,0,99)** Der kan ikke tegnes uden for skærbilledets nederste, venstre fjerdedel

**window(<wxmin>,<wxmax>,<wymin>,<wymax>)**  
er en procedure, som definerer koordinatsystemet i den gældende viewport. Punktet i viewportens nederste, venstre hjørne tildeles koordinaterne (<wxmin>,<wymin>). Punktet i øverste, højre hjørne får koordinaterne (<wxmax>,<wymax>). Alle efterfølgende grafikordrer vil adlyde dette koordinatsystem (pånær viewport og spriteordrer), indtil et nyt defineres.

Ved opstart med **USE graphics** er **viewporten** hele skærbilledet og koordinatsystemet defineret ved **window(0,319,0,199)**

Ved opstart med **USE turtle** er **viewporten** hele skærbilledet og koordinatsystemet defineret ved **window(-160,159,-100,99)**, således at punktet (0,0) er midt i skærbilledet.

### Eksempel:

**window(-1000,2000,-100,200)**

**graphicscreen(<tilstand>)**

er en procedure, som gør grafikskærmen til skærbillede, mens tekstskærmen er usynlig.

Grafikskærmen kan gøres tilgængelig i to forskellige tilstande:

**graphicscreen(0)** , højopløsnings grafik  
**graphicscreen(1)** , flerfarve grafik

Forskellen på de to tilstande er behandlingen af farver. Skærbilledets punkter er ikke uafhængige, når det drejer sig om farvetildeling:

*I tilstand 0 (højopløsningsgrafik) hører punkterne sammen i blokke på 8 vandret x 8 lodret = 64 punkter. Inden for hver blok må der kun være to forskellige farver, hvoraf den ene er baggrundsfarven. Forsøger man at angive en tredje farve til et punkt i blokken, vil hele blokken få den farve.*

*I tilstand 1 (flerfarve grafik) er opløsningen i vandret retning ikke så god, idet punkterne er forbundet parvis. Det betyder, at hver blok består af 4 x 8 par. Hvert af disse par kan tildeles en farve. Hvis det ene punkt tildeles en farve, antager det andet punkt automatisk samme farve. Inden for hver blok kan der vises fire forskellige farver på samme tid, hvoraf den ene er baggrunden. Forsøger man med en femte farve i blokken, vil den fjerde farve også ændres til den nye farve.*

### textscreen

er en procedure, som gør tekstskærmen til skærbillede, mens grafikskærmen er usynlig.

Det kan være nødvendigt i et program at skifte frem og tilbage mellem tekstsider og grafiksider, hvis programmet f.eks. både skal kunne tegne og indeholde INPUT-sætninger. Det kan forekomme besværligt, men sikrer til gengæld, at en tegning ikke forstyrres af uvedkommende tekst.

**fullscreen**

er en procedure, som gør, at hele skærbilledet udfyldes af grafik-skærmen. Instruktionen benyttes f.eks. i turtle grafik til at gå fra den delte skærm (**splitscreen**).

**splitscreen**

er en procedure, som gør grafikskærmen synlig med en rullende kopi fra tekstskærmen af de fire tekstlinier omkring markøren.

Brugt som kommando udfører **USE turtle** automatisk **splitscreen**, men ikke som programsætning.

**clearscreen**

er en procedure, som sletter hele det grafiske billede; uanset den aktuelle tegneramme (**viewport**).

**clear**

er en procedure, som sletter det grafisk billede inden for den aktuelle tegneramme.

**Eksempel:**

**viewport(0,100,0,100)**    Kun de 101 x 101 prikker  
**clear**                        i nederste, venstre hjørne af  
                                     skærbilledet slettes.

---

FARVER: I de følgende procedurer med farveangivelser gælder, at variabelen <farve> skal være et heltal mellem -1 og 15. (Bemærk: -1 betyder baggrundsfarven.) Se i øvrigt appendiks B om farver og farvekoder.

---

**textcolor(<farve>)**

er en procedure, som definerer skriftens farve på tekstskærmen.

**Eksempel:**

**textcolor(0)** sort tekst vælges

**textbackground(<farve>)**

er en procedure, som definerer tekstskærmens baggrundsfarve.

**textborder(<farve>)**

er en procedure, som definerer tekstskærmens kantfarve.

**pencolor(<farve>)**

er en procedure, som definerer pennens farve.

**Eksempler:**

**pencolor(7)** gul er valgt som tegnefarve

**pencolor(-1)** baggrundsfarven er tegnefarve

**background(<farve>)**

er en procedure, som definerer grafikskærmens baggrundsfarve.

**border(<farve>)**

er en procedure, som definerer grafikskærmens kantfarve.

**getcolor(<x>,<y>)**

er en funktion, hvis værdi er lig farven af punktet (<x>,<y>).

Hvis (<x>,<y>) er uden for tegnerammen, sat i proceduren **viewport**, har **getcolor(<x>,<y>)** værdien -1.

**getcolor** ændrer ikke på nuværende penposition (x,y).

**Eksempler:**

**PRINT getcolor(1,2)**

**IF getcolor(0,0)<0 THEN flyt'centrum**

**plot(<x0>,<y0>)**

er en procedure, som sætter en prik på penposition (<x0>,<y0>).

**Eksempel:**

**plot(4.3,56)**

**drawto(<x>,<y>)**

er en procedure, som tegner en streg fra nuværende penposition til punktet (<x>,<y>), som bliver den nye penposition.

**Eksempler:**

**drawto(100,200)**

**drawto(-20,4000)**

**moveto(<x>,<y>)**

er en procedure, som uden at tegne flytter pennen til punktet (<x>,<y>).

**Eksempel:**

**moveto(200,-25)**

**draw(<dx>,<dy>)**

er en procedure, som tegner en streg fra nuværende penposition (<x0>,<y0>) til punktet med koordinaterne (<x0>+<dx>,<y0>+<dy>) og ændrer penpositionen til slutpunktet.

**Eksempler:**

**draw(0,100)** lodret streg, 101 enheder lang

**draw(-1.5,0.4)**

**move(<dx>,<dy>)**

er en procedure, som uden at tegne flytter pennen fra nuværende position (<x0>,<y0>) til punktet med koordinaterne (<x0>+<dx>,<y0>+<dy>).

**Eksempler:**

**move(-3,20)**

**move(-2000,0)**

**setxy(<x>,<y>)**

er en procedure, som flytter pennen til punktet med koordinaterne (<x>,<y>). Hvis pennen er nede, tegnes en streg, lige som ved **drawto**(<x>,<y>). Hvis pennen er oppe, flyttes den som ved **moveto**(<x>,<y>).

**circle(<x0>,<y0>,<r>)**

er en procedure, som tegner en cirkel med centrum i (<x0>,<y0>) og radius <r>.

Om cirklen fremtræder "cirkelrund" afhænger af dit valg af tegne-område på skærmen, koordinatsystem og indstilling af højde/bredde-forholdet på TV eller monitor skærmen. Hvis koordinatsystemet er valgt i tegneområdet, så

$$\frac{\langle wxmax \rangle - \langle wxmin \rangle}{\langle wymax \rangle - \langle wymin \rangle} \cdot \frac{\langle vymin \rangle - \langle vymax \rangle}{\langle vxmax \rangle - \langle vxmin \rangle} = 1$$

bør cirklen fremtræde "rund" på skærmen. Hvis ikke, må det om muligt korrigeres med højdejusteringen på fjernsynet/monitoren.

**Eksempel 1:**

Opstart med **USE graphics** medfører automatisk:

**viewport(0,319,0,199)**

**window(0,319,0,199)**

**højde/bredde-forholdet er da lig 1, og**

**circle(160,100,99)**

**vil tegne en rund cirkel midt på skærmen**

**Eksempel 2:**

**viewport(200,300,80,180)**

**window(-1,1,-1,1)**

**circle(0,0,1)**

giver en rund cirkel øverst til højre på skærmen.

**arc(<x0>,<y0>,<r>,<a0>,<da>)**

er en procedure, som tegner en cirkelbue med centrum (<x0>,<y0>), radius <r>, startvinkel <a0> grader og buevinkel <da> grader.

**Eksempler:**

**arc(100,100,50,45,90)**

**arc(-20,25,30,15,-60)**

**xcor og ycor**

er funktioner, hvis værdi er henholdsvis pennens aktuelle x og y koordinat.

**Eksempler:**

**PRINT xcor;ycor**

**plotttext(xcor,ycor,"Figur 1")**

**fill(<x>,<y>)**

er en procedure, som benytter penfarven til at udfylde et område. Det område, som udfyldes med penfarven, skal indeholde punktet (<x>,<y>), og det afgrænses af en kant med en farve forskellig fra områdets egen farve eller af viewport-tegnerammen.

**fill** ændrer ikke på penpositionen.

Se sammenfattende eksempel under proceduren

**paint(<x>,<y>).**

**Eksempel:**

**fill(10,56)**

**paint(<x>,<y>)**

er en procedure, som benytter penfarven til at udfylde et område. Det område, som udfyldes med penfarven, skal indeholde punktet (<x>,<y>), og det afgrænses af en kant med penfarven, eller af viewport-tegnerammen.

**paint** ændrer ikke på penpositionen.

**Eksempler:**

```
paint(-10,4)
pencolor(-1)
paint(100,20) Et område "viskes" rent
```

Sammenfattende eksempler, som belyser forskellen på **fill** og **paint**:

```
USE graphics
graphicscreen(1)
pencolor(7)
drawto(319,199)
fill(10,100) // hvis paint(10,100) ingen forskel
pencolor(1)
circle(100,100,70)
fill(100,100) // hvis paint(100,100) forskel!
WHILE KEYS=CHRS(0) DO NULL
```

**showturtle**

er en procedure, som gør, at skildpadden vises på grafiskskærmen. Det engelske ord "turtle" betyder "skildpadde".

**USE turtle** bevirker automatisk, at skildpadden vises.

**hideturtle**

er en procedure, som gør, at skildpadden ikke vises på grafikskærmen.

**turtlesize(<størrelse>)**

er en procedure, som definerer størrelsen af tegnetrekanten (skildpadden).

<størrelse> skal være et tal mellem 0 og 10. Ved grafikopstart sættes størrelse automatisk til 10.

**home**

er en procedure, som anbringer skildpadden i punktet (0,0) med tegneretning opad.

**setheading(<retning>)**

er en procedure, som definerer tegneretningen. Hvis skildpadden vises, vil den dreje "hovedet" i tegneretningen.

<retning> angives i grader:  
0 svarer til lodret opad.  
90 svarer til vandret, mod højre.  
-90 svarer til vandret, mod venstre.

Ved **USE turtle** opstart sættes retning automatisk til 0.

**heading**

er en funktion, som antager værdien af den nuværende tegneretning. Tegneretningen angives i grader med 0 lodret opad og 90 grader er vandret, mod højre.

**Eksempler:**

**forward(20)**: Efter at have tegnet en lige streg,  
**arc(50,30)**: drejer pennen "blødt" 30 grader til venstre.

**penup**

er en procedure, som løfter pennen.

**pendown**

er en procedure, som sænker pennen. Det betyder, at skildpadden tegner, når den flytter sig.

Ved grafikopstart udføres automatisk **pendown**.

**left(<vinkel>)**

er en procedure, som drejer skildpadden <vinkel> grader til venstre i forhold til nuværende tegneretning.

**right(<vinkel>)**

er en procedure, som drejer skildpadden <vinkel> grader til højre i forhold til nuværende tegneretning.

**forward(<længde>)**

er en procedure, som flytter skildpadden <længde> enheder frem i nuværende tegneretning. Hvis pennen er nede, tegnes en streg.

**back(<længde>)**

er en procedure, som flytter skildpadden <længde> enheder til-

bage i forhold til nuværende tegneretning. "Skildpadden bakker". Hvis pennen er nede, tegnes en streg.

### Sammenfattende eksempel:

Tryk på <f3>-tasten: (USE turtle kommando)

Skriv direkte på de fire øverste linier:

```
left(90)
forward(70)
right(130)
forward(80)
left(40)
back(100)
hideturtle
```

Skildpadden har nu tegnet et 4 - tal

### arcl(<r>,<da>)

er en procedure, som tegner en venstrebue med krumningsradius <r> og buevinkel <da> grader. Udgangspunktet er skildpaddens nuværende position og tegneretning.

### Procedureeksempel:

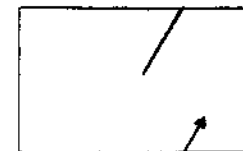
```
PROC blød'ramme(xmin,ymin,bredde,højde)
  IF bredde>20 AND højde>20 THEN
    breddebredde=20
    højde=højde-20
    moveto(xmin+10,ymin)
    setheading(90)
    forward(bredde)
    arcl(10,90)
    forward(højde)
    arcl(10,90)
    forward(bredde)
    arcl(10,90)
    forward(højde)
    arcl(10,90)
  ENDIF
ENDPROC blød'ramme
```

### arcr(<r>,<da>)

er en procedure, som tegner en blød højrebue med krumningsradius <r> og buevinkel <da>. Udgangspunktet er pennens nuværende position og tegneretning.

**arcr(<r>,<da>)** svarer til **arcl(-<r>,-<da>)**.

### Eksempel: arcr(3.45,50)



### wrap

er en procedure, som sætter grafikskærmen til **grafisk omløb**. Det betyder, at hvis pennen f.eks. forsvinder ovenud af skærmen, vil den dukke op med samme x-koordinat fornedet på skærmen. Der gælder tilsvarende forhold, hvis pennen et andet sted bevæger sig uden for skærmen.

Ved **USE turtle** opstart udføres proceduren **wrap** automatisk. Det gør den IKKE ved **USE graphics** opstart.

### nowrap

er en procedure, som **fjerner grafisk omløb**. Grafisk omløb af skærmen indstilles med proceduren **wrap**.

### textstyle(<bredde>,<højde>,<retning>,<tilstand>)

er en procedure, som bruges til at definere, hvordan udskrift af tekst på grafikskærmen skal se ud. Selve udskriften sker med proceduren **plottext**.

Parametrene <bredde>, <højde>, <retning> og <tilstand> skal alle være hele tal.

<bredde>	bogstavbredden (1 svarer til normaltekst)
<højde>	bogstavhøjden (1 svarer til normaltekst)
<retning>	0 , teksten drejes 0 grader (normal tekst) 1 , teksten drejes 90 grader 2 , teksten drejes 180 grader 3 , teksten drejes 270 grader
<tilstand>	=0 , såvel selve teksten som dens baggrundsfarve tegnes. Det betyder, at tekstfeltet rengøres før udskriften.  1, kun selve tekstens tegn skrives. Det betyder, at et <b>a</b> placeret oven i et <b>b</b> på samme plads ikke vil slette hele <b>b</b> 'et. Man vil kunne skimte resterne af <b>b</b> .

Hvis en parameter sættes lig **-1** benyttes den nuværende parameter-værdi.

Ved opstart er automatisk valgt **textstyle(1,1,0,0)** svarende til skrift

i samme størrelse som skriften på tekstskræmen; i normal vandret skriveretning; udskriftstilstanden er, at såvel tekst som dens baggrundsfarve udskrives.

#### Eksempel:

**textstyle(2,1,2,0)** Al senere tekst vil udskrives på hovedet med tegn af dobbelt bredde

**textstyle(3,2,-1,-1)** Kun skriftstørrelse ændres

#### plottext(<x>,<y>,<text\$>)

er en procedure, som udskriver den angivne tekst med udgangspunkt i punktet (<x>,<y>).

Bogstavsstørrelsen, skriveretning og skrivemåde er tidligere defineret med proceduren **textstyle**.

**plottext** ændrer ikke på penpositionen.

#### Eksempler:

```
plottext(100,150,"COMAL")
text$="Kan du kende mig"
textstyle(1,3,1,0)
plottext(200,10,text$)
```

#### inq(<nr>)

er en funktion, som bruges til at indhente oplysning om de forskellige grafiske variabelers øjeblikkelige tilstand. Parameteren <nr> skal være et helt tal mellem 0 og 33.

<nr>	Oplysning om	tilstand	påvirkes af
0	grafik skærm	0 eller 1	graphicscreen
1	tekstkant	0 - 15	textcolor, border
2	tekstbaggrund	0 - 15	textcolor, textbackground
3	tekstfarve	0 - 15	textcolors
4	grafikkant	0 - 15	border
5	grafikbaggrund	0 - 15	background
6	penfarve	0 - 15	pencolor
7	gr.tekst bredde	1 - 254	textstyle
8	gr.tekst højde	1 - 254	textstyle
9	gr.tekst retn.	0 - 3	textstyle
10	gr.tekst måde	0 eller 1	textstyle
11	skildp. vises	TRUE,FALSE	showturtle, hideturtle
12	indenfor ramme	TRUE,FALSE	de fleste tegneprocedurer
13	tekstskaerm ses	TRUE,FALSE	... screen
14	delt skaerm ses	TRUE,FALSE	... screen
15	grafisk omløb	TRUE,FALSE	wrap, nowrap
16	er pennen nede	TRUE,FALSE	penup, pendown
17	x - position	0-319	de fleste tegneprocedurer
18	y - position	0-199	de fleste tegneprocedurer
19	vxmin	0-319	viewport
20	vxmax	0-319	viewport
21	vymin	0-199	viewport
22	vymax	0-199	viewport

<nr>	Oplysning om	tilstand	påvirkes af
23	wxmin	reelt tal	window
24	wxmax	reelt tal	window
25	wymin	reelt tal	window
26	wymax	reelt tal	window
27	COS(penretning)	-1.0 - 1.0	seth,left,right,home,arcl,arcr
28	SIN(penretning)	-1.0 - 1.0	seth,left,right,home,arcl,arcr
29	skildp. stør.	0.0 - 10.0	turtlesize
30	x-forhold	reelt tal	+(wymax-wxmin)/(vxmax-vxmin)
31	y-forhold	reelt tal	+(wymax-wymin)/(vymax-vymin)
32	x-text slut	reelt tal	plottext
33	y-text slut	reelt tal	plottext

#### savescreen(<filnavn\$>)

er en procedure, som gemmer en kopi af det nuværende grafikbillede på diskette eller bånd. Filen gemmes under navnet <filnavn\$>.

Filens indhold:

**Højopløsningsbillede (fylder 36 blokke a 256 oktetter):**

```
0
baggrundsfarve
kantfarve
1000 oktetter til farve 0 og 1
8000 oktetter til bitmønstrer
```

**Flerfarvebillede (fylder 40 blokke a 256 oktetter):**

```
1
baggrundsfarve
kantfarve
1000 oktetter til farve 1 og 2
1000 oktetter til farve 3
8000 oktetter til bitmønstrer
```

#### Eksempler:

**savescreen("gr0.tegn")** gemmer et højopløsningsbillede.

**savescreen("gr1.cirkler")** gemmer et flerfarvebillede.

#### loadscreen(<filnavn\$>)

er en procedure, som henter et tidligere gemt grafikbillede ind på grafikskærmen fra diskette. Se **savescreen**.

#### Eksempler:

**loadscreen("gr0.tegn")**

**loadscreen("gr1.cirkler")**

**printscreen(<filnavn\$>,<position>)**

er en procedure, som udskriver indholdet i den aktuelle viewport til filen med navnet <filnavn\$>.

<position> er et heiltal mellem 0 og 479. Det angiver horisontal adresse på MPS801 printeren. <position> 6 svarer til, at billedet placeres med start efter 1. tegn inde på papiret, da et tegn fylder 6 adresseenheder vandret.

Proceduren er hovedsagelig tænkt som en mulighed for at få tegnet det grafiske billede på printer, men kan også benyttes til f.eks. at lagre billedet på diskette til senere brug.

Overførsel til printer vil kun kunne fungere på en Commodore MPS 801 kompatibel printer.

**Højopløsningsgrafik:**

Sværtningsgrad	Farver
0/4	baggrundsfarven
4/4	alle andre farver

**Flerfarvegrafik:**

Farverne vil udskrives efter en gråtoneskala:

Sværtningsgrad	Farver
0/4	1: hvid
1/4	3: turkis, 7: gul, 13: lysegrøn, 15: lysegrå
2/4	4: violet, 5: grøn, 8: orange, 10: lyserød, 12: grå, 14: lyseblå
3/4	2: rød, 6: blå, 9: brun, 11: mørkegrå
4/4	0: sort

**Eksempler:**

**printscreen("lp:",79)** Grafikskærmen udskrives på MPS 801 printer. Startposition vandret efter 13. tegn

**printscreen("hoved",19)** Grafikskærmens indhold gemmes på diskette under navnet **hoved**.

Filen kan ikke hentes med proceduren **loadscreen**, men som en almindelig sekventiel fil. Følgende programstump henter den gemte fil og udskriver den på printer:

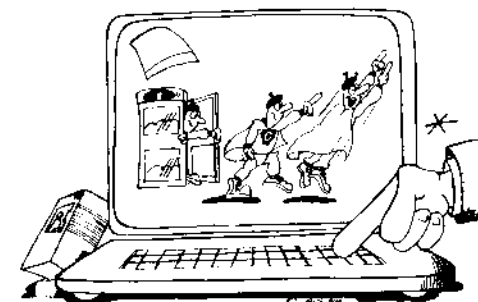
```
OPEN FILE 2,"hoved",READ
SELECT OUTPUT "lp:"
WHILE NOT EOF(2) DO PRINT GETS(2,5000)
CLOSE FILE 2
SELECT OUTPUT "ds:"
```

**SPRITES**

På Commodore 64 er det muligt at definere en lille grafisk figurenhed, som kan flyttes rundt på grafikskærmen. En sådan figur kaldes en **sprite** (udtales "sprait").

Der kan være op til 8 sprites på skærmen ad gangen. Det giver mulighed for livlige grafikbilleder, idet hver sprite kan tildeles sin egen farve og flyttes rundt uafhængigt af de andre sprites og det tegnede grafiske billede. Man kan også vælge at lade spriteene have indflydelse på hinanden.

I pakken **sprites** er lagt COMAL procedurer og funktioner, som gør det nemt at styre sprites.



Pakken gøres tilgængelig med ordren

**USE sprites**

Forestil dig, at du arbejder med sprites på følgende måde:

Vi har en scene	(computerens skærm)
med kulisser	(grafik-baggrunden)
På scenen er der skuespillere	(sprites)
som kan bevæge sig rundt	(ordren movesprite)
mens de udfører en handling	(ordren animate)
Skuespillerne kan komme ind	(showsprite)
på scenen og forlade den igen	(ordren hidesprite)
Skuespillerne kan bevæge sig,	(ordren priority)
foran og bagved hinanden	
foran og bagved rekvisitter	(grafiske tegninger)
Skuespillet dirigeres af stikord.	(COMAL sætninger)

En sprite skabes altid i et mønster på 504 prikker: 24 vandrette og 21 lodrette. En tegning må defineres ved, at hver prik tildeles en farve. En højopløsningssprite har to farver: baggrundsfarven og forgrundsfarven. Da hver prik svarer til en bit i computerens lager, kan en prik tildeles forgrundsfarven ved at anbringe et 1-tal i den tilsvarende bit. I de øvrige prikker anbringes baggrundsfarven ved hjælp af tallet 0.

Lad os gå i gang med at lave en sprite og bevæge den rundt på skærmen. Dette korte program viser, hvordan det kan gøres:

```

0100 DATA %00000000,%00000000,%00000000
0110 DATA %00000000,%00000000,%00000000
0120 DATA %00000000,%00000000,%00000000
0130 DATA %00001110,%00001110,%00000000
0140 DATA %00001111,%00001110,%00000000
0150 DATA %00000111,%00111100,%00000000
0160 DATA %00000011,%00110000,%00000000
0170 DATA %00000001,%11100000,%00000000
0180 DATA %00000011,%11100000,%00000000
0190 DATA %00000111,%11100000,%00000000
0200 DATA %00000011,%11100000,%00000000
0210 DATA %00110001,%11000000,%00000000
0220 DATA %00111111,%11100000,%00000000
0230 DATA %00001111,%11110000,%00000000
0235 DATA %00000111,%11110000,%00000000
0240 DATA %00000111,%11100000,%00000000
0250 DATA %00000111,%11100000,%00000000
0260 DATA %00011111,%11111000,%00000000
0270 DATA %00111110,%01111100,%00000000
0280 DATA %00000000,%00000000,%00000000
0300 DATA %00000000,%00000000,%00000000
0310
0320 USE graphics
0330 graphicscreen(0)
0340 USE sprites
0350 DIM tegning$ OF 64
0360 FOR i:=1 TO 63 DO
0370 READ oktet
0380 tegning$+CHR$(oktet)
0390 ENDFOR i
0400 farve:=1
0410 tegningnr:=1
0420 spritenr:=1
0430 define(tegningnr,tegning$+"0")
0440 identify(spritenr,tegningnr)
0450 spritecolor(spritenr,farve)
0460 spritepos(spritenr,50,100)
0470 showsprite(spritenr)
0480
0490 WHILE KEYS=CHR$(0) DO NULL
0500
0510 movesprite(spritenr,250,150,200,0)
0520
0530 WHILE KEYS=CHR$(0) DO NULL

```

DATA sætningerne linie 100-300 indeholder definitionen af en tegning.

Disse tal (som i COMAL direkte kan skrives i to-tals systemet) indlæses med deres CHR\$-værdi i FOR-ENDFOR løkken, sådan at tekststrengen **tegning\$** indeholder tegningens udseende.

I linie 430 defineres denne tegning til at have nummeret 1. Det

ekstra "0" angiver, at det drejer sig om en tegning i højopløsnings-grafik (i modsætning til flerfarvegrafik).

I linie 440 bestemmes, at sprite nr 1 skal se ud som tegning nr 1.

I linie 450 bestemmes farven af spriten med nummeret 1.

(farve:=1, altså hvid)

I linie 460 anbringes sprite nr 1 på skærbilledet, så øverste, venstre hjørne har (x,y)-koordinaterne (50,100).

Linie 470 gør, at spriten ses på skærmen.

Når du er blevet træt af at se på kaninen, trykkes på en taste.

Linie 510 gør, at spriten bevæger sig hen til punktet (250,150). Det sker i 200 trin. Det sidste 0 i **movesprite**-procedurekaldet vender vi tilbage til senere.

Når du igen trykker på en tilfældig taste, er programmet færdigt.

Det var dit første forsøg med en sprite. Forsøg nu selv at give kaninen en anden farve og bevæg den fra og til nye punkter på skærmen.

## SPRITEN FORSTØRRES

Prøv at tilføje programlinien

```
465 spritesize(spritenr,TRUE,TRUE)
```

Kør programmet igen. Kaninen er blevet dobbelt så høj og dobbelt så bred.

## FLERE SPRITES

Tilføj programlinierne

```

472 identify(2,tegningnr)
474 spritecolor(2,0)
476 spritepos(2,80,100)
478 showsprite(2)

```

Afprøv programmet.

Kan du få den nye sprite til at bevæge sig?

Prøv at lade de to sprites starte i hver sin side af skærmen. Bevæg dem mod hinanden, så de bytter plads.

Du bemærkede sikkert, at sprite nr 1 passerede hen foran sprite nr 2. Spriten med det laveste spritenr vil altid have første prioritet. Det betyder, at hvis de to sprites overlapper, vil spriten med laveste nr. ses foran.

## TO SPRITES STØDER SAMMEN

Det sidste tal i **movesprite**-kaldet bestemmer, hvordan spriten skal bevæge sig i forhold til andre sprites og andre grafiske tegninger på skærmen. Hidtil har det været et 0.



Hvis det sidste tal i linie 510 ændres til 1, vil spriten opfatte, når den støder sammen med en anden sprite. Begge sprites vil standse. Prøv.

## LAGR EN TEGNING

Du kan gemme en tegning med ordren:

```
saveshape(<tegningnr>,<filnavn$>)
```

på diskette eller på kassettebånd (Husk da **cs:** i filnavnet.). Tegningen kan så senere hentes ind i et andet program med ordren **loadshape(<tegningnr>,<filnavn\$>)**. Det er selvfølgelig tidsbesparende, da det så ikke vil være nødvendigt med alle DATA-sætningerne med tilhørende indlæsning af tegningens udseende.

Det følgende program definerer tegningen af kaninen og gemmer denne tegning på diskette under navnet **sp0.kanin**. Hvis du kører dette program, vil du bl.a. i det efterfølgende programeksempel kunne erstatte linierne 100-310, 360-400 og 430 med den enkelte linie

```
430 loadshape(tegningnr,"sp0.kanin")
```

Men først skal tegningen gemmes:

**0100 til 0300:** DATA sætningerne med tegningens indhold (Se forrige program.)

```
0310
0320 USE sprites
0330 DIM tegning$ OF 64
0340 FOR i:=1 TO 63 DO
0350 READ oktet
0360 tegning$+=CHR$(oktet)
0370 ENDFOR i
0380 tegningnr:=1
0390 define(tegningnr,tegning$+"0")
0400 saveshape(tegningnr,"sp0.kanin")
```

## SPRITES SAMMEN MED ANDEN GRAFIK

Det følgende program viser, hvordan en sprite kan opfatte sammenstød med en grafisk tegning og vente på, at sammenstødet sker. Efter sammenstødet fortsætter spriten i en anden retning.

**0100 til 0300:** DATA sætningerne med tegningens indhold (Se forrige program.)

```
0310
0320 USE graphics
0330 graphicscreen(0)
0340 USE sprites
0350 farve:=1
```

```
0360 DIM tegning$ OF 64
0370 FOR i:=1 TO 63 DO
0380 READ oktet
0390 tegning$+=CHR$(oktet)
0400 ENDFOR i
0410 tegningnr:=1
0420 spritenr:=2
0430 define(tegningnr,tegning$+"0")
0440 identify(spritenr,tegningnr)
0450 spritecolor(spritenr,farve)
0460 spritepos(spritenr,50,100)
0470 showsprite(spritenr)
0480
0490 WHILE KEYS=CHR$(0) DO NULL
0500
0510 lav'kasse
0520 movesprite(spritenr,250,150,200,4)
0530 WHILE NOT datacollision(spritenr,TRUE) DO NULL
0540 priority(spritenr,TRUE)
0550 movesprite(spritenr,130,180,50,0)
0560
0570 WHILE KEYS=CHR$(0) DO NULL
0580
0590 PROC lav'kasse
0600 pencolor(8)
0610 moveto(100,10); draw(50,0)
0620 draw(0,150); draw(-50,0); draw(0,-150)
0630 fill(105,15)
0640 ENDPROC lav'kasse
```

I linie 520 er det sidste tal i **movesprite**-kaldet 4. Det gør, at spriten er "opmærksom" på sammenstød med grafiktegninger. Hvis 4 ændres til 0, vil kaninen bevæge sig forbi kassen uden at ænse den.

I linie 530 ventes der, indtil et sprite-grafik sammenstød finder sted.

I linie 540 betemmes, at spriten skal ses bagved grafiktegningen. Prøv at ændre TRUE til FALSE og kør programmet.

## SPRITE TEGNEFILM

Ved at udskifte to (eller flere) tegninger hurtigt efter hinanden kan man få kaninen til at udføre en handling, mens den bevæger sig.

Vi starter med at ændre ganske lidt på den tegning, der allerede er af kaninen. (Det gøres nemmest ved at LISTE DATA-sætningerne og ændre direkte i dem).

Derefter angives en handlingsrækkefølge, som bringes til udførelse med ordren **animate(<spritenr>,<handling\$>)**.

Det færdige program kan se sådan ud:

```

0100 DATA %00000000,%00000000,%00000000
0110 DATA %00000000,%00000000,%00000000
0120 DATA %00000000,%00000000,%00000000
0130 DATA %00001110,%00000000,%00000000
0140 DATA %00001111,%00001110,%00000000
0150 DATA %00000111,%00111111,%00000000
0160 DATA %00000011,%00110111,%00000000
0170 DATA %00000001,%11100000,%00000000
0180 DATA %00000011,%11100000,%00000000
0190 DATA %00000111,%11110000,%00000000
0200 DATA %00000011,%11100000,%00000000
0210 DATA %00000001,%11000000,%00000000
0220 DATA %00000011,%11100000,%00000000
0230 DATA %00111111,%11110000,%00000000
0240 DATA %00111111,%11110000,%00000000
0250 DATA %00000111,%11100000,%00000000
0260 DATA %00000111,%11100000,%00000000
0270 DATA %00011111,%11111000,%00000000
0280 DATA %00111110,%01111100,%00000000
0290 DATA %00000000,%00000000,%00000000
0300 DATA %00000000,%00000000,%00000000
0310
0320 DATA %00000000,%00000000,%00000000
0330 DATA %00000000,%00000000,%00000000
0340 DATA %00000000,%00000000,%00000000
0350 DATA %00001110,%00001110,%00000000
0360 DATA %00001111,%00011110,%00000000
0370 DATA %00000111,%00111100,%00000000
0380 DATA %00000011,%00110000,%00000000
0390 DATA %00000001,%11100000,%00000000
0400 DATA %00000011,%11100000,%00000000
0410 DATA %00000111,%11110000,%00000000
0420 DATA %00000011,%11100000,%00000000
0430 DATA %00110001,%11000000,%00000000
0440 DATA %00111111,%11100000,%00000000
0450 DATA %00001111,%11110000,%00000000
0460 DATA %00000111,%11110000,%00000000
0470 DATA %00000111,%11100000,%00000000
0480 DATA %00000111,%11100000,%00000000
0490 DATA %00011111,%11111000,%00000000
0500 DATA %00111110,%01111100,%00000000
0510 DATA %00000000,%00000000,%00000000
0520 DATA %00000000,%00000000,%00000000
0530
.tp20
0540 USE graphics
0550 graphicscreen(1)
0560 USE sprites
0570 farve:=1
0580 spritenr:=1
0590 DIM tegning$ OF 64, handling$ OF 64
0600 FOR tegningnr:=1 TO 2 DO
0610   tegning$:= ""
0620   FOR i:=1 TO 63 DO

```

```

0630   READ oktet
0640   tegning$:=CHR$(oktet)
0650   ENDFOR i
0660   define(tegningnr,tegning$+"0")
0670   ENDFOR tegningnr
0680
0690   identify(spritenr,1)
0700   spritecolor(spritenr,farve)
0710   spritepos(spritenr,50,100)
0720   showsprite(spritenr)
0730   handling$:= ""1""+"4""+"2""+"5""
0740   animate(spritenr,handling$)
0750   movesprite(spritenr,350,150,300,0)
0760
0770   WHILE KEY$=CHR$(0) DO NULL

```

Vi håber, at dette lille program gav dig mod på større og mere indviklede dramaer.

Handlingsrækkefølgen defineres i linie 730. Der står oversat: Vis tegning 1 i 4 tidsenheder, vis tegning 2 i 5 tidsenheder. Bliv ved med at gentage denne handling, indtil spriten standser sin bevægelse.

Se i oversigten under **animate** for en uddybning af handlingsrækkefølger.

## EN SPRITE I FLERE FARVER

Hidtil har tegnerne været tegninger i højopløsningsgrafik (angivet ved et **"0"** i **define(<tegningnr>,<tegning\$>+"0")**).

Tegningen er ensfarvet, men kan fint benyttes på såvel en højgrafikskærm (**graphicscreen(0)**) som en flerfarveskærm (**graphicscreen(1)**).

En tegning kan imidlertid godt have flere farver, men det er lidt mere kompliceret at lave en sådan, hvis man da ikke har mulighed for at benytte programmet **"spriteeditor"**, som følger med COMAL kapslen på diskette. Se vejledning i brug af programmet **spriteeditor** i appendiks H.

Når man definerer en sprite-tegning i flere farver, skal man huske på, at skærmens horisontale naboprikker i flerfarvegrafik hører sammen to og to. I forbindelse med sprites i flerfarvegrafik er det sådan, at følgende talpar bestemmer spritens farve:

```

00 Gennemslagtig
01 Farve 2
10 Forgrundsfarve 1
11 Farve 3

```

En sprite kan altså have 4 forskellige farver, hvoraf den ene er "gennemslagtig". Forgrundsfarven bestemmes med **spritecolor** proceduren. Farverne 2 og 3 bestemmes med **spriteback** proceduren.

Ligesom ved tegninger i højgrafik er det en god idé at starte med at lave en tegning på et stykke ternet papir. På papiret afmærkes felterne parvis med en af de 4 mulige farver. Derefter indskrives tegningen i DATA-sætninger som før. Blot må man være noget mere omhyggelig med at tildele punktparrene de rigtige talkombinationer.

Et program med sprites i flere farver:

```
0010 DATA %00 00 00 00,%00 00 00 00,%00 00 00 00
0020 DATA %00 00 10 10,%00 00 00 00,%00 00 00 00
0030 DATA %00 00 10 10,%00 00 00 00,%00 00 00 00
0040 DATA %00 00 01 01,%01 01 01 01,%01 01 00 00
0050 DATA %00 00 01 01,%01 01 01 01,%01 01 00 00
0060 DATA %00 00 01 01,%01 01 01 01,%01 01 00 00
0070 DATA %00 00 10 10,%10 10 10 10,%10 10 00 00
0080 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0090 DATA %00 00 10 00,%00 10 10 11,%11 10 00 00
0100 DATA %00 00 10 00,%00 10 10 11,%11 10 00 00
0110 DATA %00 00 10 00,%00 10 10 11,%11 10 00 00
0120 DATA %00 00 10 00,%00 10 10 11,%11 10 00 00
0130 DATA %00 00 10 00,%00 10 10 01,%11 10 00 00
0140 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0150 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0160 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0170 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0180 DATA %00 00 10 10,%10 10 10 11,%11 10 00 00
0190 DATA %11 11 11 11,%11 11 11 01,%01 11 11 11
0200 DATA %11 11 11 11,%11 11 11 01,%01 11 11 11
0210 DATA %11 11 11 11,%11 11 11 01,%01 11 11 11
0220
0230 USE graphics
0240 graphicscreen(1)
0250 USE sprites
0260 DIM tegning$ OF 64
0270 FOR I:=1 TO 63 DO
0280   READ oktet
0290   tegning$+=CHR$(oktet)
0300 ENDFOR I
0310
0320 tegningnr:=1
0330 define(tegningnr,tegning$+"1")
0340 background(0)
0350 spriteback(2,12)
0360 RANDOMIZE
0370 FOR spritenr:=0 TO 7 DO
0380   spritecolor(spritenr,RND(3,10))
0390   spritepos(spritenr,spritenr*40,50)
0400   identify(spritenr,tegningnr)
0410   showsprite(spritenr)
0420   spritesize(spritenr,1,1)
0430 ENDFOR spritenr
0440 FOR I:=1 TO 100 DO plot(RND(0,319),RND(50,199))
0450 WHILE KEYS=CHR$(0) DO NULL
```

I linie 240 vælges flerfarvegrafik tilstand.

I linie 330 defineres tegningen som en flerfarve tegning ved hjælp af ""1"" i procedurekaldet.

I linie 340 vælges grafikkærmens baggrundsfarve

I linie 350 vælges spritenes 2. og 3. farver.

I linie 380 vælges en tilfældig forgrundsfarve for hver sprite.

I linie 420 sættes alle til dobbelt størrelse.

I linie 440 sættes stjerner på himlen.

## SPRITE OVERSIGT

Pakken **sprites** indeholder 23 procedurer og funktioner:

Definition af tegning og sprite:

define(<tegningnr>,<tegning\$>)

identify(<spritenr>,<tegningnr>)

Sritens farve(r):

spritecolor(<spritenr>,<farve>)

spriteback(<farve2>,<farve3>)

Sritens størrelse:

spritesize(<spritenr>,<xdobbel>,<ydobbel>)

Sritens position og bevægelse:

spritepos(<spritenr>,<x>,<y>)

movesprite(<spritenr>,<x>,<y>,<trin>,>0måde>)

startsprites

stopsprite(<spritenr>)

moving(<spritenr>)

spritex(<spritenr>)

spritey(<spritenr>)

animate(<spritenr>,<handling\$>)

Synlighed:

showsprite(<spritenr>)

hidesprite(<spritenr>1)

priority(<spritenr>,<grafik'foran>)

Kollisionscheck:

spritecollision(<spritenr>,<ja/nej>)

datacollision(<spritenr>,<ja/nej>)

Oplysninger om sprites:

spriteinq(<spritenr>,<egenskab>)

En sprite gøres til en grafisk tegning:

stampsprite(<spritenr>)

**Spritetegninger og lagring:****saveshape**(**<tegningnr>**,**<filnavn\$>**)**loadshape**(**<tegningnr>**,**<filnavn\$>**)**linkshape**(**<tegningnr>**)(Husk **cs**: for båndfiler.)**define**(**<tegningnr>**,**<tegning\$>**)

er en procedure, som definerer en ny tegning. **<tegning\$>** er en streng med en længde på 64 tegn. Den indeholder tegningens udseende (se eksemplerne i begyndelsen af dette afsnit). Det sidste tegn skal være **"0"** eller **"1"**. Den definerede tegning tildeles nummeret, angivet med parametren **<tegningnr>**.

Der kan være op til 32 tegninger ad gangen. Parameteren **<tegningnr>** skal derfor være et helt tal mellem 0 og 31. En tegning må gerne benyttes til at identificere flere sprites.

**Eksempel:**

**define(23,hus\$)** Indholdet i strengen **hus\$** definerer tegningen med nummeret **23**

**identify**(**<spritnr>**,**<tegningnr>**)

er en procedure, som angiver, at spriten med nummeret **<spritnr>** skal vises som tegningen med nummeret **<tegningnr>**. Der kan være op til 8 forskellige sprites på skærmen ad gangen. **<spritnr>** skal være et helt tal mellem 0 og 7. Samme tegning kan godt være grundlag for flere sprites.

Spriten med laveste **<spritnr>** har højst prioritet og vises derfor forrest, hvis flere sprites overlapper på skærmen.

Hvis grafik-skildpadden vises på skærmen, har den altid spritenummeret **7** og tegningen nummereret **31**.

**Eksempel:**

**identify(0,23)** Sprite nummer **0** vises som tegning nr **23**

**spritecolor**(**<spritnr>**,**<farve>**)

er en procedure, som tildeler spriten med nummeret **<spritnr>** den angivne farve. **<spritnr>** er et helt tal mellem 0 og 7, og **<farve>** er et helt tal mellem 0 og 15. I højopløsningsgrafik får spriten denne farve. I flerfarvegrafik er det **farve1**.

**Eksempel:**

**spritecolor(0,8)** Sprite nummer **0** får farve nummer **8**.

**spriteback**(**<farve2>**,**<farve3>**)

er en procedure, som definerer tillægfarverne i flerfarvegrafik. En flerfarve-sprite kan have fire farver:

gennemsigtig (overtegner ikke andre farver)

forgrundsfarven, sat med **pencolor** (=farve1)tillægfarverne, sat med **spriteback** (=farve2 og farve3)**Eksempel:**

**spriteback(2,7)** Tillægfarverne er rød og gul

**Specielt om sprites med flere farver:**

I en flerfarve tegning hører punkterne parvis sammen. Hver farve (baggrunds-, forgrunds- og tillægfarver) kendetegnes ved bitmønstreret

Bitpar	vist farve	sættes med
00	gennemsigtig	grafikordrer
01	farve 2	spriteback
10	farve 1	spritecolor
11	farve 3	spriteback

Hvis grafikken er prioriteret over sprites

(eks. **priority(<spritnr>,TRUE)**) vil **farve2** med bitmønstreret **01** også være baggrundsfarve.

**Farve2** giver ikke melding om kollision med en anden sprite (**spritecollision**) eller med grafiktegninger (**datacollision**).

**spritesize**(**<spritnr>**,**<xdobbel>**,**<ydobbel>**)

er en procedure, som bestemmer, om **<spritnr>** skal vises i dobbelt størrelse. Normalt fylder en sprite 24 prikker i x-retningen og 21 prikker i y-retningen. Hvis **<xdobbel>** sættes lig et tal forskelligt fra **0** (=TRUE), vil spriten vises i dobbelt bredde. Tilsvarende gælder for **<ydobbel>**.

**Eksempler:**

**spritesize(5,0,1)** Sprite **5** dobbelt højde

**spritesize(2,TRUE,TRUE)** Sprite **2** dobbelt så stor

**spritepos**(**<spritnr>**,**<x>**,**<y>**)

er en procedure, som placerer spritens øverste venstre hjørne i punktet med skærmkoordinaterne (x,y).

Spritepositioner angives altid i skærmkoordinatsystemet, uafhængigt af et eventuelt andet koordinatsystem, defineret med grafikordren **window**. Spritekoordinater er altså i koordinatsystemet (-32768...32767, -32768...32767), hvoraf kun punkterne (0...319,0...199) vises på skærmen.

**Eksempel:**

**spritepos(0,25,50)** sprite **0** placeres i punktet (25,50)

**movesprite(<spritenr>,<x>,<y>,<trin>,<måde>)**

er en procedure, som bevæger <spritenr> fra nuværende position til punktet (x,y). Bevægelsen sker i <trin> flytninger, hvor hver flytning tager 1/50 sekund på computere efter europæisk PAL standard. (På computere efter amerikansk NTSC standard tager hver flytning 1/60 sekund). Tiden er i begge tilfælde den tid, der går mellem opdateringer af computerens skærbillede.

Parameteren <trin> er altså et udtryk for, hvor mange tidsenheder flytningen skal vare. Desto færre trin, desto hurtigere bevægelse. Altså bestemmer <trin> spritens bevægelseshastighed:

1. Hvis <trin> altid fastholdes på samme værdi, vil hastigheden være proportional med afstanden til slutpunktet.
2. Hastigheden er den samme, uafhængig af afstanden til slutpunktet, hvis <trin> f.eks. defineres ved:

```
FUNC trin(spritenr,x,y)
  fart:=10
  dx:=x-sprite(x,spritenr)
  dy:=y-sprite(y,spritenr)
  dist:=SQRT(dx*dx+dy*dy)
  RETURN fart*dist
ENDFUNC trin
```

Hvis denne funktion benyttes til at bestemme parametren **trin**, vil hastigheden altid være den samme. I dette tilfælde 1 skærmenehed pr. 10 tidsenheder.

3. Hastigheden kan gøres uafhængig af **x-afstanden** (tilsvarende med **y-afstanden**). Man opnår, at spriten glider med konstant hastighed i én dimension.

Det sikres, hvis <trin> f.eks. bestemmes med denne funktion:

```
FUNC trin(spritenr,x)
  fart:=10
  dist:=ABS(x-sprite(x,spritenr))
  RETURN fart*dist
ENDFUNC trin
```

Hvis specielt **trin** er lig 0, føres spriten straks til (<x>,<y>) uanset værdien af <måde>. Derefter flytter spriten sig ikke yderligere, men kan godt bringes til at udføre en handling ved hjælp af ordren **animate**.

Parametren <måde> har betydning for, hvornår flytningen begynder, og om der skal tages hensyn til kollision med andre sprites og grafiske tegninger. <måde> er et helt tal mellem 0 og 7:

	<måde>	virkning
0	Start nu,	ikke kollisioncheck
1	Afvent startsignal,	ikke kollisioncheck
2	Start nu,	check sprite/sprite kollision
3	Afvent startsignal,	check sprite/sprite kollision
4	Start nu,	check sprite/grafik kollision
5	Afvent startsignal,	check sprite/grafik kollision
6	Start nu,	check kollision generelt
7	Afvent startsignal,	check kollision generelt

**Bemærk:**

Proceduren **movesprite** sætter bevægelsen i gang. COMAL systemet venter ikke på, at bevægelsen skal standse, men fortsætter med næste programlinje. Det muliggør, at andre spritebevægelser kan startes, tekst udskrives m.m. Der kan foregå mange ting på samme tid.

Hvis programudførelsen på den anden side ikke må gå videre, mens bevægelsen sker, må der tilføjes en "vente"-linje. F.eks.

```
WHILE moving(<spritenr>) DO NULL
```

eller

```
WHILE NOT datacollision(<spritenr>,TRUE) DO NULL
```

**Eksempler:**

**movesprite(2,200,130,100,0):** Bevæg sprite nr 2 til punktet (200,130) i 100 trin (d.v.s. på 100 tidsenheder). Start nu; uden kollisioncheck.

**movesprite(0,250,-10,300,6):** Bevæg sprite nr 0 til punktet (250,-10) i 300 trin. Start nu; check om spriten støder på andre sprites eller grafiktegnninger.

Bemærk, at brugen af diskettstationen kan forstyrre spritenes bevægelse, især hvis flere sprites er i gang, da diskoperationer kræver brug af interrupt. Brug evt. **stampsprite** til at fastholde spritebilleder.

**startsprites**

er en procedure, der starter bevægelsen af de sprites, som afventer startsignal. Se **movesprite** proceduren.

**stopsprite(<spritenr>)**

er en procedure, som standser bevægelsen af spriten med det angivne nummer.

**moving(<spritenr>)**

er en funktion, som har værdien TRUE (=1), hvis den angivne sprite bevæger sig. Ellers har funktionen værdien FALSE (=0).

**Eksempel:**

**IF NOT moving(2) THEN movesprite(2,0,190,50,0)**

Hvis sprite 2 ikke bevæger sig, skal den med det samme flyttes op i punktet (0,190).

**spritex(<spritnr>)** og **spritey(<spritnr>)**

er funktioner, hvis værdi er <spritnr>'s aktuelle x- og y-position.

**Eksempler:**

**x'forskel:=x-spritex(4)**

**y'forskel:=y-spritey(4)**

**IF spritey(3)>200 THEN movesprite(3,spritex(3),20,200,0)**

Hvis sprite nr. 3 støder mod skærmens øverste kant, skal den falde "til gulvet".

**animate(<spritnr>,<handling\$>)**

er en procedure, som får den angivne sprite til automatisk at udføre en given handling. Den ønskede handling må anføres i strengen <handling\$>.

Antal tegn i handlingsrækkefølgen skal være et lige tal på højst 64. Altså højst 32 handlinger.

Mulige handlinger:

**CHR\$(<tegningnr>)+CHR\$(<tid>):** Tegningen med det angivne nr. skal vises i den angivne tid.

NB:  $0 \leq \text{<tegningnr>} \leq 31$  og  $0 \leq \text{<tid>} \leq 255$  tidsenheder.

Se proceduren **movesprite** for redegørelse for tidsfastsættelsen.

Hvis <tid> specielt er lig 0, vil spriten gå i en ventetilstand, som kun kan afbrydes af ordren **startsprites** eller af en "g"-handling.

<b>"p"+CHR\$(&lt;tid&gt;):</b>	Hold pause i den angivne tid.
<b>"g"+CHR\$(&lt;spritnr&gt;):</b>	Genstart den angivne sprite, hvis den er i ventetilstand.
<b>"s"+CHR\$(&lt;spritnr&gt;):</b>	Den angivne sprite vises.
<b>"h"+CHR\$(&lt;spritnr&gt;):</b>	Den angivne sprite skjules.
<b>"x"+CHR\$(&lt;xdobbel&gt;):</b>	Hvis <xdobbel> er <b>TRUE</b> (dvs. <>0) fordobles spritens bredde. Hvis <xdobbel> er lig <b>FALSE</b> (dvs. =0), er spriten 24 prikker bred.
<b>"y"+CHR\$(&lt;ydobbel&gt;):</b>	Tilsvarende som ved "x"+CHR\$(<xdobbel>).
<b>"c"+CHR\$(&lt;farve&gt;):</b>	Spriten får den angivne farve, hvor $0 \leq \text{<farve>} \leq 15$ .

Handlingen startes ikke af sig selv, men af proceduren **movesprite**.

Handlingen, som strengen foreskriver, udføres fra venstre mod højre, når spriten ikke er i ventetilstand. Når den sidste handling er udført, startes forfra. Således fortsættes, indtil spriten ikke længere bevæger sig: **movesprite** bevægelsen er ophørt, eller en **animate(<spritnr>,"")** udføres.

Ligesom ved **movesprite**-proceduren afventer COMAL systemet ikke udførelsen af handlingen, men fortsætter i næste programlinje.

Bemærk, at **CHR\$(<værdi>)** betyder det samme som **"<værdi>"**, så

**handling\$="s"+CHR\$(1)+"p"+CHR\$(10)+"h"+CHR\$(1)+"p"+CHR\$(10)**

er det samme som

**handling\$="s"1"p"10"h"1"p"10"**

**Eksempler:**

**animate(1,"s"1"p"10"h"1"p"10")**

**movesprite(1,100,100,0,0)**

**WHILE KEYS=CHR\$(0) DO NULL**

**animate(1,"")**

Sprite nr. 1 vil med det samme flytte til punktet (100,100) og blinke i 10 tidsenheders intervaller, indtil der trykkes på en taste.

**animate(3,"1"4"2"4"3"4")**  
**movesprite(3,300,180,500,0)**

Mens sprite nr. 3 bevæger sig til punktet (300,180), vises den først i 4 tidsenheder som tegning nr. 1. Dernæst vises den i 4 tidsenheder som tegning nr. 2, efterfulgt af tegning nr. 3. Og så forfra igen. Tegnefilm!

**showsprite(<spritnr>)**

er en procedure, som gør den angivne sprite synlig (hvis den er placeret på skærmen).

**hidesprite(<spritnr>)**

er en procedure, som skjuler den angivne sprite.

**priority(<spritnr>,<grafik'foran>)**

er en procedure, som bestemmer den angivne sprite's prioritet i forhold til grafiske tegninger på skærmen. Hvis <grafik'foran> har værdien **TRUE** (=1), vises grafik foran spriten, hvis der er overlap. Hvis værdien er **FALSE** (=0), vil spriten vises foran grafik. Ved **USE sprites** sættes værdien automatisk til **FALSE**.

**Eksempel:**

**priority(6,1)** Sprite nr. 6 vises bag grafik

**spritecollision(<spritenr>,<ja'nej>)**

er en funktion, som bruges til at fastlægge, hvornår den angivne sprite kolliderer med en anden sprite, eller om den tidligere er kollideret med en.

Hvis <ja'nej>=TRUE, er **spritecollision** FALSE, indtil sammenstød finder sted.

Hvis <ja'nej>=FALSE, vil **spritecollision** være TRUE, hvis sammenstød tidligere har fundet sted.

Sammenstød finder sted ved overlap af farver forskellig fra baggrundsfarven. Se specielt om flerfarvegrafik i bemærkning efter **spriteback**-proceduren.

**Eksempler:**

**WHILE NOT spritecollision(2,TRUE) DO NULL**

Lav intet, før sprite nr. 2 støder sammen med en anden sprite.

**IF spritecollision(4,0) THEN spritecolor(4,2)**

Hvis sprite nr. 4 tidligere er stødt sammen med en anden sprite, skal den farves rød.

**datacollision(<spritenr>,<ja'nej>)**

er en funktion, som bruges til at fastlægge, hvornår den angivne sprite kolliderer med grafiktegninger, eller om den tidligere er kollideret med grafik.

Hvis <ja'nej>=TRUE, er **datacollision** FALSE, indtil sammenstød finder sted.

Hvis <ja'nej>=FALSE, vil **datacollision** være TRUE, hvis sammenstød tidligere har fundet sted.

Sammenstød finder sted ved overlap af farver forskellig fra baggrundsfarven. (Se under **spritecollision**.)

**spriteinq(<spritenr>,<egenskab>)**

er en funktion, som anvendes til at få oplysninger om den angivne sprite. Talværdien af parametren <egenskab> bestemmer, hvilken egenskab, man får oplysning om.

<egen- skab>	Funktion	Mulig værdi	Sættes med
0	synlig	TRUE/FALSE	hide- og showsprite
1	Flerfarve2 (01)	0..15	spriteback
2	Flerfarve1 (10)	0..15	spritecolor
3	Flerfarve3 (11)	0..15	spriteback
4	dobbelt brede	TRUE/FALSE	spritesize
5	dobbelt højde	TRUE/FALSE	spritesize
6	flerfarve	TRUE/FALSE	define,identify
7	grafik/sprite prioritet	TRUE/FALSE	priority
8	tegning nummer	0..31	identify
9	resterende varighed	0..215	movesprite
10	sprite/sprite kollision	TRUE/FALSE	movesprite
11	sprite/grafik kollision	TRUE/FALSE	movesprite
12	bevægelsesmåde	0..7	movesprite
13	antal handlinger	0..32	animate
14	næste handlings nummer	0..32	animate

Bemærk, at **TRUE** og **FALSE** har talværdierne 1 og 0.

**Eksempel:**

**FOR nr:=1 TO 14 DO PRINT spriteinq(nr)**

**stampsprite(<spritenr>)**

er en procedure, som bruges til at gøre den angivne sprite til en grafisk tegning. Spriten 'sættes fast' på grafikken.

Normalt er en sprite ikke en del af en grafisk tegning og vil derfor ikke udskrives sammen med den øvrige grafik (procedurerne **print-screen** og **savescreen**). **stampsprite** gør imidlertid en kopi af spriten til en del af det grafiske billede. Proceduren benyttes f.eks., hvis man ønsker, at grafik-skildpadden skal være med på en senere tegning af grafikbilledet.

**Eksempel:**

**FOR spritenr:=7 TO 0 STEP -1 DO stampsprite(spritenr)**

En kopi af alle synlige sprites fastgøres til grafikskærmen.

**saveshape(<tegningnr>,<filnavn\$>)**

er en procedure, som gemmer en kopi af den angivne tegning på diskette eller bånd (husk da **cs:** i filnavnet) under navnet <filnavn\$>. Selve tegningen skal være indeholdt i en streng på 64 tegn.

**Eksempel:**

**define(2,tegning\$)**

**saveshape(2,"sp0.blomst")**

Den figur, som er indeholdt i strengen **tegning\$**, gemmes under navnet "sp0.blomst". 0 er medtaget i navnet for at angive, at det drejer sig om en tegning i højopløsningsgrafik.

**loadshape(<tegningnr>,<filnavn\$>)**

er en procedure, som fra disketten eller bånd henter en kopi af filen med navnet <filnavn\$>. Filen skal tidligere være gemt med proceduren **saveshape**. <filnavn\$> skal indeholde en streng med definitionen på en sprite-tegning. Denne tegning tildeles nummeret <tegningnr>.

**Eksempel:**

**loadshape(1,"sp0.blomst")** Filen **sp0.blomst** indeholder en streng med en tegning, som i programmet herefter kendes ved nummeret 1.

**linkshape(<tegningnr>)**

er en procedure, som hægter en kopi af den angivne tegning på COMAL programmet. Når programmet gemmes med ordren **SAVE**, vil tegningen blive gemt sammen med det, og derefter hentes ind sammen med det med ordren **LOAD**.

Om ønskeligt frigøres tegningen fra COMAL programmet med ordren **DISCARD**.

Tegningen skal tidligere være hentet med proceduren **loadshape**. Denne tegning tildeles nummeret <tegningnr>.

**Eksempel:**

**linkshape(7)** Tegningen med nummeret 7 hægtes på COMAL programmet i arbejdslageret.

**LYD OG MUSIK**

Alle, der kender Commodore 64's lydegenskaber, vil blive glædeligt overraskede over at lære, hvor let COMAL kapslen gør udnyttelsen af Commodore 6581 lydsynthesizer (SID) chip'en. Med chip'en kan man benytte 3 stemmer ad gangen (ligesom at spille flerhændigt på klaver). I tilgift er der betydelig frihed til at vælge, hvordan de enkelte toner skal lyde. Man kan bestemme frekvens (tonehøjden), lydstyrke, lydtype, modulation og filtrering. Afsnittet her må kun betragtes som en indledning til et meget spændende emne. Man kunne tilegne en helt bog til udnyttelsen af Commodore 64's lyd- og lydegenskaber.

Med COMAL ordren

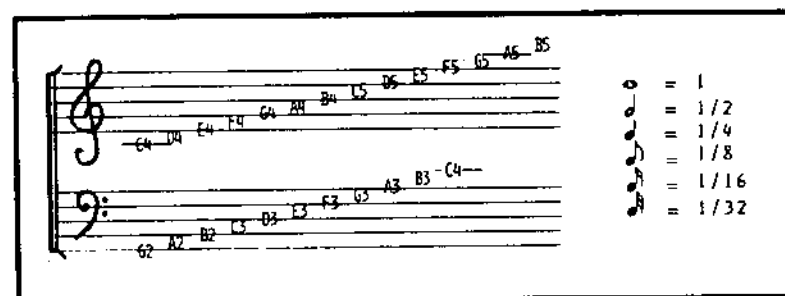
**USE sound**

gøres adskillige procedurer og funktioner tilgængelige. Brug disse procedurer og naturlige COMAL-sætninger til at "dirigere orkestret".

Musikskalaens enkelte toner er givet beskrivende navne. F.eks.

angives det mellemste c som strengen "c4". Noderne i oktaven, der begynder med "c4", benævnes: "c4", "c4#", "d4", "d4#", ..., "b4", "b4#". Noderne i næste oktav kendetegnes ved et efterstillet 5-tal: "c5", "b5#". "f5#" står f.eks. for tonen "fis 2". Tilsvarende med de andre oktaver. Noderne med et #-tegn er halve toner.

Skønt denne vejledning ikke er ment som et musikkursus, hjælper det sikkert med en oversigt over sammenhængen mellem nodernes placering på musikskalaen og strengenes betydning for Commodore 64. For at spille korrekt må computeren kende både tonens højde og dens varighed. Den følgende figur viser nogle af tonernes placering og standardsymbolerne for deres varighed:



Noderækken går fra og med "c0" til og med "a7#" på computere med europæisk PAL standard.

I dette afsnit behandler vi kort 6 programmer, som dels findes på den medfølgende demonstrationsdiskette og dels som udskrift i appendiks H. De benævnes som følger:

**playscore:** Begynd med dette program for at få en idé om **lydpakkens** muligheder. Når du har gennemgået lektionsprogrammerne 1-5, vil det være en god idé at vende tilbage og studere programmet.

**musik 1:** I denne første lektion vises, hvordan enkelttoner spilles.

**musik 2:** Op til tre forskellige stemmer kan principielt spilles på samme tid. Her spilles de efter hinanden.

**musik 3:** Her spilles en melodi med kun en stemme. Gennemlæs programudskriften for at lære, hvordan musikprogrammet er opbygget. Lav en ny melodi.

**musik 4:** I dette program kan man ændre et antal parametre, som har betydning for en stemmes lydbillede: **lydstyrke**, **lydtype** og **adsr** (attack-decay-sustain-release) **lydformen**.



**musik 5:** Opsætning af et musikstykke: synkroniseret, flerstemmig afspilning.

Så går vi i gang med lektionerne. LOAD, RUN og LIST det første af de fire små undervisningsprogrammer. Bemærk først, at ordren **USE sound** er nødvendig i programmet for at aktivere lyd-procedurerne. I udskriften skal man særligt lægge mærke til linierne 150-320:

```
0150 INPUT AT 8,1: "Stemme: ": stemme
0160 INPUT AT 9,1: "Tone: ": node$
0170 spil(stemme,node$)
0190
0200 PROC spil(stemme,node$)
0210 IF node$ <> "z" THEN
0220   note(stemme,node$)
0230   gate(stemme,1) // stlg og fald
0240 ENDIF
0250 pause(16) // hold tonen
0260 gate(stemme,0) // kling ud
0270 ENDPROC spil
0280
0290 PROC pause(sek'32)
0300 TIME 0
0310 WHILE TIME < 1.875*sek'32 DO NULL
0320 ENDPROC pause
```

Først indlæses **stemmens nummer** (1, 2 eller 3), og derefter strengen med den ønskede **node** (c0,c0#,d0,...,a7#). I linie 170 kaldes proceduren **spil** med overførsel af de indtastede værdier.

Hvis den indtastede **node\$** er tegnet **z**, spilles tonen ikke. Brug **"z"**, når der skal være en pause i musikken. Som andre noder, skal også den angives med en varighed. Hvis **node\$** er en "rigtig" tone, spilles den.

Det gøres som følger. Proceduren **note(stemme,node\$)** anbringer stemme- og nodevalg i computerens lydregister. Med proceduren **gate(stemme,1)** begynder computeren at spille tonen. Tonens attack-tid (ansatstid) og decay-tid (faldtid) fra maksimal styrke til det niveau (sustain), hvor den holdes, udføres med det samme. Brugeren må tilføje en **pause**-procedure, som bestemmer, hvor længe tonen holdes. Til sidst afbryder sætningen **gate(stemme,0)** denne fase, og tonen klinger ud (release) som angivet i **adsr** ordren. Mere om **adsr** senere.

Proceduren **pause** bevirker, at COMAL holder pause i **sek'32** 32. dele sekunder. I programmets linie 250 kaldes **pause** med værdien **16**. Det svarer til en pause på 16/32 1/2 sekund.

For at få to toner samtidigt kan man f.eks. tilføje programlinierne

```
225 note(2,"c5")
235 gate(2,1)
265 gate(2,0)
```

Svar herefter på programkørslen med stemme **1** og node **c4**. Programmet spiller da to toner med forskellige stemmer en oktav fra hinanden.

Afprøv programmet **musik 2**. Her udnyttes procedurerne **spil** og **pause** igen. Desuden er der følgende sekvens:

```
0130 FOR stemme:=1 TO 3 DO
0140   soundtype(stemme,3)
0150 ENDFOR stemme
0160
0170 INPUT AT 7,1: "Tonevalg: ": node$
0180
0190 FOR stemme:=1 TO 3 DO
0200   PRINT AT 10,1: "Stemme ":stemme
0210   spil(stemme,node$)
0220   spil(stemme,"z")
0230 ENDFOR stemme
```

Linie 130-150 definerer **lydtype**en for hver af de tre stemmer. Proceduren **soundtype** har to parametre. Den første er **stemme**-nummeret (1,2 eller 3), og den anden er **lydtype**en (0,1,2,3 eller 4). Tallene står for:

lydtype 0: Ingen lyd  
 lydtype 1: trekant form  
 lydtype 2: savtak form  
 lydtype 3: firkant pulser  
 lydtype 4: hvid støj

Det kræver nogen øvelse at udvælge den bedst egnede lydform til at opnå den ønskede lydeffekt. I linie 130-150 sættes alle stemmer til **firkant pulser**.

I linie 170 indlæses en node. I linie 190-230 spilles alle tre stemmer efter hinanden, så man kan få et indtryk af forskellen. Bemærk, at proceduren **spil(stemme,node\$)** benyttes på samme måde som tidligere. Bemærk også, at vi med noden **"z"** har indlagt en pause mellem hver af de tre stemmer. Prøv at fjerne linie 220 og hør, hvordan det lyder, når programmet køres.

Hent og kørs nu programmet **musik 3**. Få det udskrevet og læg særligt mærke til linierne 330-500:

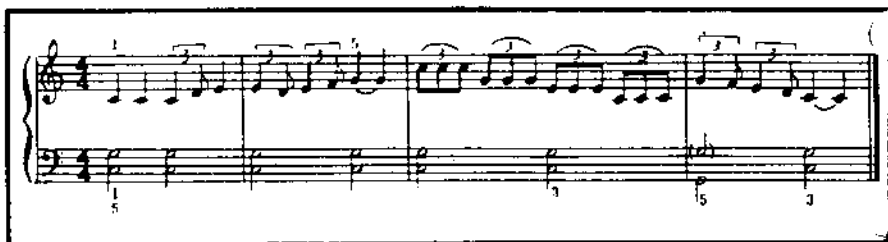
```
0330 PROC spil'melodi // Ro, ro, ro din baad...
0340
0350 melodien:
0360 DATA "c4",8,"z",2,"c4",8,"z",2,"c4",8,"d4",4
0370 DATA "e4",8,"z",8,"e4",8,"d4",4,"e4",8
0380 DATA "f4",4,"g4",16,"z",8,"c5",4
0390 DATA "c5",4,"c5",4,"g4",4,"g4",4
```

```

0400 DATA "g4",4,"e4",4,"e4",4,"e4",4
0410 DATA "c4",4,"c4",4,"c4",4,"z",8,"g4",8
0420 DATA "f4",4,"e4",8,"d4",4,"c4",8
0430
0440 RESTORE melodien
0450 WHILE NOT EOD DO
0460   READ node$,tid
0470   spil(stemme,node$,tid)
0480 ENDWHILE
0490
0500 ENDPROC spil'melodi

```

Denne procedure spiller en enkel melodi (Ro, ro, ro din båd):



Proceduren begynder med at nulstille DATA-pegepinden (RESTORE melodien), så DATA-sætningerne læses fra melodien begyndelse, hver gang melodien spilles. DATA linierne indeholder noderne (parvis tonehøjde og varighed). Sammenlign med nodebladet ovenfor: Den første tone er **c** med noden **c4**. Den har en kvart tonelængde, og varigheden skal derfor sættes til **8** ( $8/32 = 1/4$ ). Bemærk, at det første element er en **streng**, mens det andet er et heltal. Efter den første tone skal der være en kort pause, sådan at tonerne ikke flyder sammen. Det ordnes med **"z",2**. Herefter fortsættes med næste tone, som også er et c, osv. Det er ikke altid nødvendigt med en tvungen pause mellem tonerne. Eksperimenter dig frem, indtil du lærer at frembringe det tilsigtede.

Der er mange måder at håndtere musik på. Man kunne forestille sig melodiliniere som lange DATA-strenger. En specielt konstrueret procedure kunne så "pille" noderne, varigheden og pauserne ud. Man kunne lave tids-variablerne om til heltalsvariabler for at spare plads, hvis man har komponeret et digert værk. Hvis musikstykket indeholder gentagelser, er det en klar fordel at angive hver musikdel som en selvstændig procedure. En "dirigent procedure" kan så spille værket ved at kalde underprocedurerne efter tur.

Selve afspilningen foregår i linie 450 - 490. Data læses parvis ind og overføres til **spil**-proceduren. Denne proces fortsætter, indtil der ikke er flere noder (EOD lig TRUE). Bemærk, at WHILE strukturen sikrer, at computeren ikke starter med at spille, hvis der slet ikke er nogle noder i DATA-sætningerne.

Gå nu i gang med **musik 4**. I dette program vil du støde på flere

procedurer fra **sound** pakken. De følgende linier er specielt interessante:

```

0180 INPUT AT 11,1: "STEMME (1/2/3)? ": stemme
0190 INPUT AT 13,1: "STYRKE (0-15)? ": styrke
0200 INPUT AT 15,1: "LYDTYPE (1/2/3/4)? ": type
0210 soundtype(stemme,type)
0220 volume(styrke)

0420 INPUT AT 21,1: "A,D,S,R? ": a,d,s,r
0430 adsr(stemme,a,d,s,r)
0440
0450 spil'melodi

```

I linie 180-200 indlæses **stemmens nummer**, **lydstyrken** og **lydstypen** for den valgte stemme. Proceduren **volume(styrke)** fra pakken benyttes til at regulere lydstyrken mellem tavshed (0) og fuld styrke (15).

I linie 430 bestemmes **lydkurvens parametre**. De har betydning for lydens intensitetsforløb, mens tonen spilles. Selve lyden består af signaler, hvis type angives med proceduren **soundtype**. Proceduren **adsr** giver brugeren mulighed for at bestemme lydkurvens udseende (envelope): Hvor hurtigt tonen stiger til maksimal intensitet (attack), falder (decay) til det niveau (sustain), hvor den holdes, indtil den klinger af (release). Sekvensen kaldes attack-decay-sustain-release forløbet. Hvor længe sustain-fasen varer bestemmes igen af proceduren **pause**. Intensitetsforløbets udseende bestemmes altså af de tal, som frit kan vælges mellem 0 og 15:

**attack:** bestemmer, hvor hurtigt tonen stiger til fuld intensitet. Dette tal skal være lille for at opnå en "klaver", "banjo" eller "cembalo" lyd, da lyden af anslåede strengeinstrumenter er karakteriseret ved et hurtigt, hørbart attack, når strengen slås an.

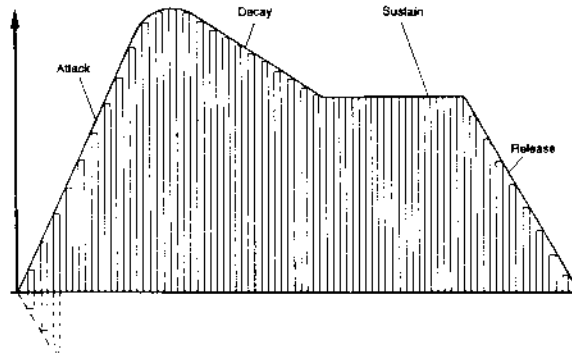
**decay:** bestemmer, hvor hurtigt tonen falder ned til sustain niveauet. Ved at ændre dette tal ændres på strengeinstrumentets type.

**sustain:** bestemmer intensitetsniveauet, hvor tonen fastholdes, f.eks. med en pause-procedure.

**release:** bestemmer efterklangstiden; hvor hurtigt tonen klinger ud ved sustain-periodens ophør.

Den sidste lektion viser, hvordan man kan sætte flere stemmer på samtidigt ved hjælp af proceduren **playscore**. I eksemplet er kun anvendt én stemme, nemlig stemme 1, men det kan ændres ved at tilføje et par nye linier. Se senere.

Noderne indlæses og omsættes med funktionen **frequency** til et



frekvens-tal. Alle disse tal lagres i en tabel **tone {}** sammen med de tilhørende pauser: en **ads'pause** for ads-fasen og en **r'pause** for r-fasen (inklusive tonemellemrum). Tallene anbringes i stemme 1's register ved hjælp af proceduren **setscore**, hvorefter selve afspilningen startes med proceduren **playscore**.

Mens melodien spilles, udskriver COMAL nogle tal. Det sker blot for at illustrere, at COMAL kan foretage beregninger m.m., mens baggrundsmusikken spiller. Når baggrundsmusikken er slut, antager funktionen **waitscore** værdien TRUE (=1). Derfor vil udskriften af tal fra WHILE-ENDWHILE løkken også ophøre.

```
0090 nr:=0
0100 WHILE NOT EOD DO
0110   nr:=1
0120   READ node$,tid
0130   tone#(nr):=frequency(node$)
0140   ads'pause#(nr):=tid*2
0150   r'pause#(nr):=tid*2
0160 ENDWHILE
0170
0180 tone#(nr+1):=0
0190 setscore(1,tone#(),ads'pause#(),r'pause#())
0200 playscore(1,0,0)
0210
0220 tal:=0
0230 WHILE NOT waitscore(1,0,0) DO
0240   tal:=tal+1
0250   PRINT tal;
0260 ENDWHILE
```

Tilføj linierne

```
192 setscore(2,tone#(),ads'pause#(),r'pause#())
194 setscore(3,tone#(),ads'pause#(),r'pause#())
```

og ændr linie 200 og linie 230 til henholdsvis

### **playscore(1,1,1) og waitscore(1,1,1)**

De tre stemmer vil da spille melodien samtidigt (synkroniseret). Programmet slutter, når alle tre stemmer er færdige.

Kan du lave en kanon nu med forsinkelse mellem de forskellige stemmer?

Læg mærke til, at når pakken først tages i brug ved eksekvering af **use sound**, bliver følgende parameterværdier valgt:

```
adsr(1,0,4,12,10)
adsr(2,10,8,10,9)
adsr(3,0,9,0,9)
FOR stemme:=1 TO 3 DO
  pulse(stemme, 2048)
  setfrequency(stemme,0)
ENDFOR stemme
volume(15)
soundtype(1,1) // piano
soundtype(2,2) // violn
soundtype(3,3) // bækken
```

Vi afslutter dette afsnit om lyd med en oversigt over de procedurer og funktioner, som er indeholdt i pakken **sound**:

```
volume(<styrke>)
note(<stemme>,<node$>)
gate(<stemme>,<start'stop>)
soundtype(<stemme>,<lydtype>)
adsr(<stemme>,<attack>,<decay>,<sustain>,<release>)
setscore(<stemme>,<frekvens()>,<pause1()>,<pause2()>)
playscore(<stemme1>,<stemme2>,<stemme3>)
stopplay(<stemme1>,<stemme2>,<stemme3>)
waitscore(<stemme1>,<stemme2>,<stemme3>)
frequency(<node$>)
setfrequency(<stemme>,<frekvens'tal>)
sync(<stemmekombination>,<ja'nej>)
filterfreq(<frekvens'tal>)
filter(<stemme1>,<stemme2>,<stemme3>,<ydre>)
filtertype(<low>,<band>,<high>,<3-afbrudt>)
pulse(<stemme>,<pulsbredde>)
ringmod(<stemmekombination>,<ja'nej>)
resonance(<resonansgrad>)
env3
osc3
```

## UDDYBNING

### **volume(<styrke>)**

er en procedure, som kontrollerer den fælles lydstyrke for alle tre stemmer. **<styrke>** er et heltal mellem 0 og 15.

**Eksempel:**

**volume(15):** fuld lydstyrke

**note(<stemme>,<node\$>)**

er en procedure, som benyttes til at angive den tone <node\$>, som stemmen med nummeret <stemme> skal spille. <stemme> kan være 1, 2 eller 3; <node\$> er en streng med mulige værdier: "c0", "c0#", "d0", ..., "a7#" på maskiner efter europæisk PAL standard. På maskiner efter NTSC standard kan tonerne gå op til "b7". Bogstaverne angiver tonen, og tallet angiver oktavnummeret. Et efterstillet # angiver halvtonerne.

**Eksempel:**

**note(2,"d5")** stemme 2 skal spille tonen d5

**gate(<stemme>,<start'stop>)**

er en procedure, som enten starter en afspilning af stemmen med nummeret <stemme> eller stopper afspilningen. <start'stop> lig 1 starter, og <start'stop> lig 0 standser.

**Eksempel:**

**gate(3,1)** stemme 3 startes

**soundtype(<stemme>,<lydtype>)**

er en procedure, som benyttes til at angive hvilken <lydtype> <stemme> skal være. <lydtype> er den form, som det periodiske lydsignal skal bestå af:

<lydtype> lig

- 0: ingen lyd
- 1: trekanter
- 2: savtakker
- 3: firkanter
- 4: "hvid" støj

**Eksempel:**

**soundtype(1,3)** stemme 1 skal være firkanter.

**adsr(<stemme>,<attack>,<decay>,<sustain>,<release>)**

er en procedure, som bestemmer intensitetsforløbs udseende. Se tidligere under musiklektion 4. Bemærk specielt, at <sustain> angiver et lydniveau mellem 0 og det maksimale lydniveau (bestemt med **volume**), hvorimod <attack>, <decay> og <release> er tidsforløb.

Tal	<attack>	<decay> og <release>:
0:	2 msek	6 msek
1:	8	24 -
2:	16	48 -
3:	24	72 -
4:	38	114 -
5:	56	168 -
6:	68	204 -
7:	80	240 -
8:	100	300 -
9:	250	750 -
10:	500	1.5 sek
11:	800	2.4 -
12:	1 sek	3 -
13:	3	9 -
14:	5	15 -
15:	8	24 -

<sustain> kan have værdierne 0, 1, ..., 15

**Eksempel:**

**adsr(1,13,13,8,13)** stemme 1 lydforløbet angives

**setscore(<stemme>,<frekvens()>,<pause1()>,<pause2()>)**

er en procedure, som benyttes til at anbringe en melodi i en given stemmes register. <stemme> er et helt tal: 1,2 eller 3. <frekvens()> er en tabel, som må indeholde de enkelte toners frekvenstal. De tilhørende pauser lagres i tabellene <pause1()>, som indeholder **ads**-pauselængden, og i <pause2()>, som indeholder pausen for **r**-fasen. Funktionen **frequency** benyttes ved omsætningen fra nodeangivelse til frekvenstal. Selve afspilningen af melodien påbegyndes med proceduren **playscore**.

**Eksempel:**

**setscore(2,frekvenser(),ads'pause(),r'pause())**

Frekvenser med tilhørende pauser anbringes i stemme nr. 2's register.

**playscore(<stemme1>,<stemme2>,<stemme3>)**

er en procedure, som benyttes til synkroniseret start af stemmerne. Et 1-tal ud for den angivne <stemme?> starter afspilningen:

**Eksempel:**

**playscore(1,1,0)** stemme 1 og 2 startes

**stopplay(<stemme1>,<stemme2>,<stemme3>)**

er en procedure, som standser afspilningen af de angivne stemmer. Hvis <stemme?> er TRUE (=1), standses afspilningen.

**Eksempel:**

**stopplay(0,1,1)** stemme 2 og 3 standser.

**waitscore(<stemme1>,<stemme2>,<stemme3>)**

er en funktion, som får værdien TRUE (=1), hvis afspilningen af den angivne stemmekombination er slut.

**Eksempel:**

**WHILE NOT waitscore(1,1,0) DO NULL**

gør intet, før afspilningen af stemme 1 og 2 er slut.

**frequency(<node\$>)**

er en funktion, som returnerer den heltalsværdi, som SID chip'en skal have for at spille en tone. Den benyttes mest til at udregne en tabel til proceduren **setscore**. Værdierne er mellem -32768 og 32767, så man kan ikke direkte omsætte mellem oktaver ved at dividere tallene med 2. <node\$> skal indeholde en streng med tonen i området 'c0' til 'a7#'.

**Eksempel:**

**frequency("c4")**: tonen "c4" omregnes til talværdi.

**setfrequency(<stemme>,<frekvens'tal>)**

er en procedure, som benyttes til at definere <stemme>'s frekvens. Tallet <frekvens'tal> skal være i området 0 - 65535. Tallene svarer ikke direkte til frekvenser.

**Eksempel:**

**setfrequency(2,2000)**

**sync(<stemmekombination>,<ja'ne>)**

er en procedure, som sørger for synkronisering i henhold til den angivne <stemmekombination>, hvis <ja'ne> er lig 1. Ellers synkroniseres ikke.

**Bemærk: stemmekombination  
nummer:**

1  
2  
3

**svarer til synkronisering  
mellem stemmerne:**

1 og 3  
1 og 2  
2 og 3

**Eksempel:**

**sync(1,1)** stemme 1 og 3 synkroniseres

**filterfreq(<frekvens'tal>)**

er en procedure, som benyttes til at bestemme afskæringsfrekvensen for filtret. Parameteren <frekvens'tal> skal være i området 0 til 2047, svarende til en frekvens mellem 30 Hz og 12000 Hz.

**Eksempel:**

**filterfreq(1500)**

**filter(<stemme1>,<stemme2>,<stemme3>,<ydre>)**

er en procedure, som bruges til at udvælge, hvilke stemmer, der skal filtreres. Dvs. dæmpes. Et 1-tal ud for <stemme?> betyder, at <stemme?> skal filtreres.

**Eksempel:**

**filter(0,1,1,1)**: stemme 1 skal IKKE filtreres

**filtertype(<low>,<band>,<high>,<3'afbrudt>)**

er en procedure, som benyttes til at udvælge filtertypen.

Hvis <low> er lig 1 benyttes et "low-pass" filter, som dæmper toner i diskantområdet. Alle frekvenser over filterfrekvensen (stillet i **filterfreq**) dæmpes 12 dB pr. oktav.

Hvis <band> er lig 1, dæmpes til begge sider omkring filterfrekvensen; 6 dB pr. oktav.

Hvis <high> er lig 1, dæmpes de lave frekvenser med 12 dB pr. oktav.

Hvis <3'afbrudt> er lig 1, vil stemme 3 ikke høres. Det kan udnyttes til synkronisering og ringmodulation.

Flere filtre kan vælges på samme tid.

**Eksempel:**

**filtertype(1,0,1,0)** giver et "notch-filter" med den modsatte effekt af et "band-pass filter": dæmpning omkring filterfrekvensen

**pulse(<stemme>,<pulsbredde>)**

er en procedure, som benyttes til at angive forholdet mellem den tid, hvor en firkant-puls er høj og den tid, den er lav. Desto mere dette forhold afviger fra 1:1, desto mere nasal og spids bliver lyden. <pulsbredde> er et tal mellem 0 og 4096. Ved 2048 er forholdet 1:1.

**Eksempel:**

**pulse(1,2048)** forholdet høj/lav er lig 1.

**ringmod(<stemmekombination>,<ja'nej>)**

er en procedure, som bruges til at bestemme, om der skal være ringmodulation; <stemmekombination> afgør mellem hvilke stemmer (se **sync**). Hvis <ja'nej> er TRUE (=1) udføres modulationen; ikke hvis <ja'nej> lig FALSE (=0).

Ved ringmodulation frembringes to nye stemmer med frekvenser hhv. lig summen og differensen mellem de oprindelige stemmer.

**resonance(<resonansgrad>)**

er en procedure, som benyttes til at angive i hvor høj grad enkelte frekvenser kraftigt skal fremhæves. Desto større resonansgrad, desto kraftigere fremhæves frekvenserne omkring frekvensen valgt med proceduren **setfrequency**. Det sætter et syntetisk præg på lyden. Parameteren <resonansgrad> skal være et heiltal mellem 0 og 15.

**env3**

er en funktion uden parametre. Den returnerer intensitetskurvens højde for stemme nummer 3. Funktionsværdierne ligger i intervallet 0 - 255.

**Program til tegning af Intensitetsforløbet:**

```
USE sound
USE graphics
graphicscreen(0)
volume(10)
soundtype(3,1)
note(3,"a4")
adsr(3,13,13,8,13)
gate(3,0)
WHILE env3<>0 DO NULL
TIME 0
gate(3,1)
WHILE TIME<60*10 DO
drawto(TIME/5,env3/256*199)
```

**ENDWHILE**

```
gate(3,0)
```

```
WHILE TIME/5<320 DO
```

```
drawto(TIME/5,env3/256*199)
```

```
ENDWHILE
```

```
WHILE KEY$=CHR$(0) DO NULL
```

**osc3**

er en funktion uden parametre, som returnerer en værdi mellem 0 og 255. Tallet angiver udsvinget for stemme 3's aktuelle lydtype. For en **trekant** svinger tallene fra 0 til 255 og tilbage til 0 igen. For en **savtak** stiger tallene fra 0 til 255 for derefter pludseligt at falde til 0. For **firkant**-puls skifter tallene mellem 0 og 255. **Hvid støj** giver tilfældige tal mellem 0 og 255.

---

Bemærk, at lyden fortsætter med at spille efter at COMAL programmet er standset. Lyden stoppes kun, hvis en melodi er færdigspillet, eller hvis COMAL programmet giver en fejlmeldelse eller kommunikerer med diskettstationen. Disse ordre benytter også **interrupt**'en, som benyttes til lydfrembringelse.

---

**PAKKER TIL BRUG SAMMEN MED KONTROLPORTENE**

COMAL kapslen indeholder 3 pakker, som alle kan benyttes i forbindelse med de to indgangsstik, der sidder på siden af Commodore 64 (på SX 64 sidder de bagpå). Disse to indgangsstik kaldes kontrolport 1 og kontrolport 2.

Kontrolportene virker på den måde, at ændringer i indstillingen af tilsluttet apparatur omsættes til tal, som computeren kan arbejde med. Commodore 64 kan forbindes til forskelligt tilbehør. (Se f.eks. kapitel 7 om Ydre Enheder.)

I dette afsnit beskæftiger vi os med

**paddles**

```
joysticks
```

```
lyspen (eng. lightpen)
```

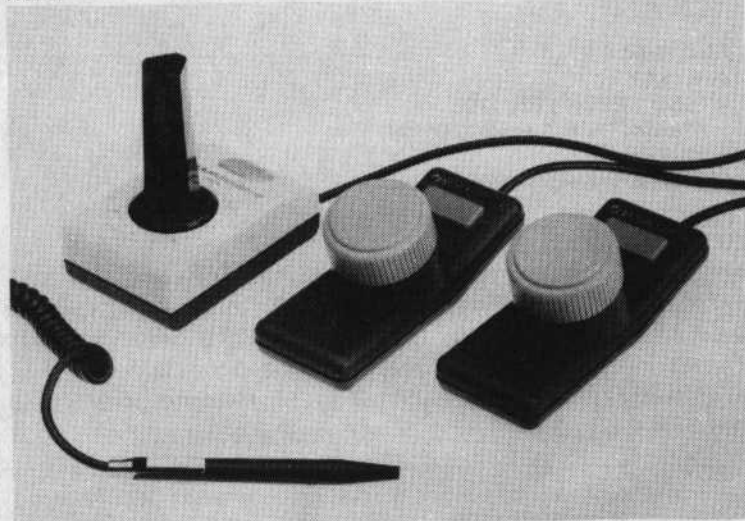
Dette tilbehør kan købes hos din Commodore forhandler.

COMAL pakkerne indeholder procedurer, som gør det lettere at anvende dette tilbehør.

**PADDLES**

Pakken **paddles** gøres tilgængelig med ordren

```
USE paddles
```



Til én kontrolport forbindes et paddle-par, som vi kalder **a-paddlen** og **b-paddlen**. Hver paddle er udstyret med en drejeknap, som styrer signalstyrken, og en trykknop (affyringsknappen), som kortslutter en portindgang, når den trykkes ned.

Pakken indeholder en enkelt procedure:

```
paddle(<portnr>,<a'paddle>,<b'paddle>,<a'knap>,<b'knap>)
```

som omsætter signalerne fra en kontrolport til tal.

- \* Parametren **<portnr>** skal indeholde nummeret på den kontrolport, paddleparret er forbundet til. Altså: **1** eller **2**.
- \* Variableerne **<a'paddle>** og **<b'paddle>** indeholder talværdien af drejeknapindstillingen på henholdsvis a-paddlen og b-paddlen:  
 $0 \leq \text{<a'paddle>} \leq 255$  og  $0 \leq \text{<b'paddle>} \leq 255$
- \* Variablen **<a'knap>** indeholder talværdien **1**, hvis a-trykknappen er trykket ned, ellers er **<a'knap>** lig **0**. Tilsvarende gælder for **<b'knap>**.

Eksempel:

**USE paddles**

```
paddle(2,a'paddle,b'paddle,a'knap,b'knap)
PRINT a'paddle;b'paddle;a'knap;b'knap
```

Signalværdierne hentes fra kontrolport 2 og udskrives i næste linie. Progameksempel, som viser en leg med brug af et paddlepar:

## 0010 USE paddles

0020

0030 DIM format\$ OF 40

0040 format\$="### # ### #"

0050

0060 PAGE

0070 INPUT AT 2,1: "kontrolport nr > ": portnr

0080

0090 DIM vinder\$ OF 1

0100 vinder\$="c"

0110 PRINT AT 9,2: "Hvem stiller hurtigst paddlen ind "

0120 PRINT AT 10,2: "og trykker affyringsknappen ned"

0130 PRINT AT 13,2: "Tryk taste for start"

0140 RANDOMIZE

0150 WHILE KEYS=CHRS(0) DO NULL

0160 tal:=RND(0,255)

0170 PRINT AT 15,2: "Tallet er : ",tal

0180

0190 REPEAT

0200 paddle(portnr,a'paddle,b'paddle,a'knap,b'knap)

0210 PRINT AT 5,1: " a'paddle a'knap b'paddle b'knap"

0220 PRINT AT 6,1: USING format\$: a'paddle,a'knap,b'paddle,b'knap

0230

0240 IF tal=a'paddle AND a'knap THEN vinder\$="a"

0250 IF tal=b'paddle AND b'knap THEN vinder\$="b"

0260 UNTIL vinder\$ IN "ab"

0270

0280 PRINT AT 17,2: vinder\$+" var hurtigst"

## JOYSTICKS

Pakken **joysticks** gøres tilgængelig med ordren

### USE joysticks

Til en kontrolport forbindes en joystick. En joystick er en styrepind, som kan være i hvile eller skubbes i 8 forskellige retninger:

retning	COMAL - talværdi
op	1
op-venstre	8
op-højre	2
venstre	7
midten	0
højre	3
ned-venstre	6
ned-højre	4
ned	5

Desuden er der på joystick'en en trykknop (affyringsknappen), som sender et signal til datamaten, når den trykkes ned.

**Pakken indeholder en enkelt procedure:**

**joystick(<portnr>,<retning>,<knap>)**

som omsætter signalerne fra en kontrolport med en joystick til talværdier.

- \* <portnr> skal indeholde nummeret på den port, joysticken er forbundet til: **1** eller **2**.
- \* <retning> er en variabel, som indeholder talværdierne **0 - 8**. Disse talværdier angiver styrepindens retning. Se ovenfor.
- \* <knap> er en variabel med værdien **1**, når affyringsknappen er trykket ned, ellers er <knap> lig **0**.

**Eksempel:**

**USE joysticks**

**joystick(2,retning,knap)**

**PRINT retning;knap**

Signalværdierne hentes fra kontrolport 2 og udskrives i næste linie.

Programeksempel, som viser, hvordan en joystick kan bruges til at tegne med:

```

0100 PAGE
0110 PRINT "JOYSTICK SOM TEGNER"
0120 PRINT
0130 PRINT "Styrepinden bestemmer tegneretning"
0140 PRINT "Affyringsknappen skifter farve"
0150 PRINT
0160 PRINT "Tryk <STOP>, hvis programmet skal stoppe"
0170 PRINT "Tryk <f5> for at se tegningen igen,"
0180 PRINT "og <f1> for at komme tilbage til teksten"
0190 PRINT
0200 INPUT "Joystick i Port nr: (1 eller 2) :": portnr
0210 IF portnr<1 OR portnr>2 THEN portnr:=2
0220
0230 USE turtle
0240 USE joysticks
0250 graphicscreen(1)
0260 background(1)
0270 pencolor(5)
0280
0290 LOOP
0300 joystick(portnr,retning,knap)
0310 IF retning THEN
0320   setheading((retning-1)*45)
0330   forward(1)
0340 ENDIF

```

```

0350 IF knap THEN //skift farve
0360   pencolor((inq(6)+1) MOD 16)
0370 ENDIF
0380 ENDLOOP

```

**LYSPEN**

For at forstå, hvordan en lyspen virker, må man forstå, hvordan et billede på TV- eller monitorskærmen dannes. Billedet skabes af en elektronstråle, som med stor hastighed og varierende styrke fejer hen over skærbilledet. Da et skærbillede opdateres 50 eller 60 gange pr. sekund, når øjnene ikke at opfatte denne bevægelse. En lyspen indeholder i spidsen en fotodiode, som registrerer, når den rammes af lys. Når elektronstrålen er ud for lyspennen, vil fotodioden spore det. Der udsendes med det samme et signal. Fra denne oplysning kan computeren beregne, hvor lyspennen berører skærmen.

Lyspennen skal altid forbindes til kontrolport 1.

Dernæst gøres pakken **lightpen** tilgængelig med ordren

**USE lightpen**

Lyspennens virker bedst, når skærmkanten er mørk og baggrunden lys.

Hvis ikke nedenstående programstump virker, forsøg da at stille på skærmens lys- og kontrastforhold.

Ved hjælp af programmet undersøges lyspennens opfattelse af koordinaterne på skærmområdet. Skriv programmet af og afprøv det:

```

0010 PAGE
0020 USE lightpen
0030 USE system
0040 textcolors(0,14,6)
0050
0060 offset(0,0)
0070 REPEAT readpen(x,y,ok) UNTIL ok
0080
0090 PRINT x;y

```

Programmet indeholder 2 procedurer fra lyspen pakken:

I linie 60 bestemmes, at lyspennens måling af et punkts koordinater foreløbigt ikke skal forskydes.

I linie 70 registreres, hvor lyspennen berører skærmen.

Lad pennen glide langsomt fra den mørke kant i nederste venstre hjørne ind i det lyse område. Programmet vil da udskrive lyspennens opfattelse af koordinaterne for dette punkt. Prøv nogle gange, indtil



koordinaterne er fundet med rimelig nøjagtighed. Koordinaterne kaldes lyspennens **offset** fra (0,0). Vi benævner koordinatsættet (<xoff>,<yoff>). (<xoff>,<yoff>) er forskellig fra skærm til skærm på grund af forsinkelse i den elektriske registrering af lyspennens placering.

I programmets linie 60 var **offset** sat til (0,0). Ændr nu (0,0) til det fundne (<xoff>,<yoff>).

Når du kører det ændrede program og fører lyspennen ude fra og ind i hjørnet, skulle lyspennen gerne måle punktet til (0,0). Hvis ikke, har du her et mål for den nøjagtighed, hvormed lyspennen er i stand til at registrere punkter. Finjuster eventuelt.

Undersøg nu lyspennens koordinatområde. Det skulle strække sig fra (0,0) til ca.(319,199).

Efter denne indledende indstilling er der gjort klar til større opgaver.

I første eksempel udnyttedes, at computeren ved ordren **USE lightpen** automatisk satte vigtige parameterværdier til deres opstartsværdier. Det gælder f.eks. den tid, som lyspennen skal holdes stille på skærmen, før dens berøringspunkt bestemmes (proceduren **delay**). Det gælder bl.a. også den tid, der skal gå efter bestemmelsen af et punkts koordinater, til computeren er i stand til at begynde på en ny bestemmelse (proceduren **timeon**). Et program, som tegner på grafiskskærmen, må være i stand til at bestemme punkters koordinater forholdsvis hurtigt. Derfor må **delay** og **timeon** tiderne sættes til små værdier. Lægger man mere vægt på nøjagtighed, må større værdier anvendes.

Programmet kan se således ud:

```
0010 PAGE
0020 USE lightpen
0030 USE graphics
0040 graphicscreen(0)
0050 border(0)
0060 background(14)
0070 pencolor(6)
0080
0090 xoff:=52; yoff:=-51 // eksempelvis
0100 offset(xoff,yoff)
0110
0120 delay(1)
0130 timeon(1)
0140
0150 REPEAT readpen(x,y,ok) UNTIL ok
0160 moveto(x,y)
0170 LOOP
0190 REPEAT readpen(x,y,ok) UNTIL ok
0230 drawto(x,y)
0250 ENDLOOP
```

Forsøg at ændre på værdierne i linie 120 og 130. Hvilken virkning har det?

Bemærk, at alle linier "hænger sammen". Hvordan gør man, hvis pennen skal kunne hæves?

Hvis man skal bestemme lyspennens placering på tekstskærmen, omsættes lyspennens koordinatsæt til placeringen af et tegn (<linie>,<kolonne>). Tekstskærmen har 25 linier, hver med 40 kolonner.

I det følgende eksempel benyttes to brugerdefinerede COMAL-funktioner (**FUNC linie(y)** og **FUNC kolonne(x)**) til omsætningen. For at funktionerne skal virke korrekt kræves, at lyspennens koordinater er korrigeret med **offset** proceduren, så lyspennen måler nederste venstre hjørne til (0,0).

Programmet illustrerer, hvordan en lyspen kan anvendes til at vælge fra en menu med tegn, ord eller andre valgmuligheder. I dette tilfælde er opgaven at udvælge ord fra ordlisten nederst og sammensætte dem til en sætning på højst 40 tegn:

```
0010 PAGE
0020 DIM text$(25,4) OF 10
0030 DIM navn$ OF 10, alt$ OF 40
0040 ZONE 10
0050 I:=8
0060
0070 USE system
0080 textcolors(0,14,6)
0090
0100 opstil'ord
0110
0120 USE lightpen
0130 delay(60)
0140 timeon(60)
0150 accuracy(10,2)
0160 xoff:=52; yoff:=-51 // justeres efter monitor
0170 offset(xoff,yoff)
0180
0190 udpeg'ord
0200
0210
0220 PROC opstil'ord
0230 CURSOR I,1
0240 FOR I:=1 TO 5 DO
0250   FOR J:=1 TO 3 DO
0260     READ text$(I,J)
0270     PRINT text$(I,J),
0280   ENDFOR J
0290   PRINT
0300 ENDFOR I
0310 text$(6,1):="slut"
0320 PRINT text$(6,1)
0330 PRINT AT 6,1: "Udpeg ordene med lyspennen"
```

```

0340 ENDPROC opstil'ord
0350
0360 PROC udpeg'ord
0370 REPEAT
0380   REPEAT readpen(x,y,ok) UNTIL ok
0390   IF y<199-(l-1)*8 THEN // fra linie l
0400     navn$:=text$(linie(y)-l+1,kolonne(x) DIV 10+1)
0410     IF navn$<>"slut" THEN alt$:= " "=navn$
0420     PRINT AT 2,1: alt$
0430   ENDIF
0440   WHILE penon DO NULL
0450   UNTIL navn$="slut"
0460   CURSOR 20,1
0470 ENDPROC udpeg'ord
0480
0490 FUNC linie(y)
0500   RETURN (200-y) DIV 8+1
0510 ENDFUNC linie
0520
0530 FUNC kolonne(x)
0540   RETURN x DIV 8+1
0550 ENDFUNC kolonne
0560
0570 DATA "Peter","tager","nok"
0580 DATA "katten","spiser","af"
0590 DATA "maden","regner","altid"
0600 DATA "alting","husker","aldrig"
0610 DATA "bogen","glemmer","snart"

```

I linie 150 benyttes proceduren **accuracy** fra lyspen-pakken. **accuracy(<dx>,<dy>)** bestemmer opløsningen i x- og y-retningen. Tilføj selv flere DATA-sætninger.

## OVERSIGT OVER PAKKEN LIGHTPEN

Pakken indeholder fem procedurer og en funktion:

```

offset(<xoff>,<yoff>)
penon
readpen(<x>,<y>,<ok>)
timeon(<tid>)
delay(<tid>)
accuracy(<dx>,<dy>)

```

### offset(<xoff>,<yoff>)

er en procedure, som bruges til at forskyde lyspennens koordinatsæt, så det passer med grafikskærmens koordinatsæt. Denne forskydning er forskellig fra skærm til skærm og må justeres efter forholdene.

Opstartsværdier: **<xoff>= 75** og **<yoff>= -45**.

### Eksempel:

**offset(52,-51)** Lyspennens koordinater forskydes, så (0,0) er i nederste venstre hjørne.

### penon

er en funktion, som har værdien **TRUE (=1)**, hvis pennen berører skærmen. Ellers er **penon** lig **FALSE (=0)**.

### readpen(<x>,<y>,<ok>)

er en procedure, som aflæser skærmkoordinaterne og leverer dem i variablene **<x>** og **<y>**. Variablen **<ok>** har værdien **TRUE**, hvis pennen berører skærmen (som funktionen **penon**).

### Eksempel:

```

REPEAT readpen(x,y,ok) UNTIL ok
PRINT x,y

```

Aflæs skærmkoordinater, når pennen berører skærmen. Udskriv koordinaterne i næste linie.

### delay(<tid>)

er en procedure, som bruges til at angive den **tid**, lyspennen skal holdes stille på skærmen, før skærmen betragtes som berørt. Lyspennen skal holdes stille inden for det område, der angives med proceduren **accuracy**.

**<tid>** angives i 1/60 sekunder ("jiffies").

Opstartsværdi: **<tid>=10** (dvs. 10/60 1/6 sekund)

### timeon(<tid>)

er en procedure, som bruges til at angive den **tid**, der skal gå fra én skærmaflæsning, til den næste er mulig.

**<tid>** angives i 1/60 sekunder.

Opstartsværdi: **<tid>=30** (dvs. 30/60 = 1/2 sekund)

### accuracy(<dx>,<dy>)

er en procedure, som bruges til at angive størrelsen af det område, lyspennen må bevæge sig i og alligevel betragtes som ubevægelig. Desto mindre område, desto roligere skal lyspennen holdes.

Opstartsværdier: **<dx>=4** og **<dy>=2**

### Eksempel:

**accuracy(10,8)** Pennen betragtes som ubevægelig, hvis den holdes inden for et 10x8 felt.

## PAKKEN SYSTEM

### USE system

Pakken system indeholder bl.a. procedurer, som bruges til at fastlægge, hvordan skærm-billede, tastatur og printer-forbindelsesled skal fungere. Desuden indeholder pakken funktioner, som oplyser om systemet, skærm-billede og tastatur:

```
textcolors(<kant>,<baggrund>,<tekst>)
keywords'in'upper'case(TRUE eller FALSE)
names'in'upper'case(TRUE eller FALSE)
quote'mode(TRUE eller FALSE)
inkey$
settime(<klokkeslet$>)
gettime$
getscreen(<skærm$>)
setscreen(<skærm$>)
hardcopy(<enhed$>)
currow og curcol
bell(<varighed>)
free
defkey(<nr>,<tekst$>)
showkeys
serial(TRUE eller FALSE)
setprinter(<attributter$>)
setrecorddelay(<varighed>)
setpage(<heltal>)
```

**textcolors(<kant>,<baggrund>,<tekst>)**

er en procedure, som bruges til at definere farvekombination af kant, baggrund og tekst. Ved opstart udføres automatisk **textcolors(14,6,14)** på Commodore 64. På SX-64 gælder, at ved opstart er **textcolors(3,1,6)**.

#### Eksempler:

```
textcolors(0,2,1)    sort kant, rød baggrund og hvid efterfølgen-
                     de tekst
textcolors(12,11,15) gråtoner
textcolors(-1,5,-1)  Kun baggrunden skifter (her til grønt).
```

**keywords'in'upper'case(TRUE eller FALSE)**

er en procedure, som fastlægger, om nøgleord skal skrives med stort (TRUE) eller med småt (FALSE). Ved opstart gælder TRUE.

#### Eksempel:

**keywords'in'upper'case(FALSE):** Nøgleord skrives med småt.

**names'in'upper'case(TRUE eller FALSE)**

er en procedure, som fastlægger, om navne skal skrives med stort (TRUE) eller med småt (FALSE). Ved opstart gælder FALSE.

#### Eksempel:

**names'in'upper'case(TRUE)** Navne skrives med stort.

**quote'mode(TRUE eller FALSE)**

er en procedure, som fastlægger, om kontrolkoder og andre ikke synlige ASCII-tegn i strengkonstanter skal vises i omvendt skrift (TRUE) eller ved deres ASCII værdi omsluttet af anførselstegn (FALSE). Efter opstart gælder FALSE.

#### Eksempler:

PRINT sætning efter **quote'mode(TRUE):**

```
PRINT "****Hej"
    efter quote'mode(FALSE):
PRINT ""2"Hej"
```

**inkey\$**

er en funktion, som indlæser tegn fra tastaturet. Funktionen **inkey\$** virker som **KEY\$** med den forskel, at **inkey\$** med blinkende markør afventer et tryk på en taste.

#### Eksempler:

```
svar$:=inkey $
PRINT inkey$
```

**settime(<klokkeslet\$>)**

er en procedure, som benyttes til at stille uret i computeren (CIA#1 real time clock). Ved opstart nulstilles uret som ved **settime("00:00:00.0")**.

Formatet i **klokkeslet\$**-strengen er:

<b>hh:mm:ss.t/ff</b>	hh er timetallet (0 - 24)
eller:	mm er minuttallet (0 - 59)
<b>hh:mm:ss</b>	ss er sekundtallet (0 - 59)
eller:	t er tiendedele sekunder (0 - 9)
<b>hh:mm</b>	eller: hvis et talfelt udelades,
<b>hh</b>	tildeles det værdien 0
	ff er frekvensen (50 eller 60); kan udelades
	(opstart: 50 Hz)

SX-64 er udstyret med switch-modestrømforsyning, der altid kører 60 HZ. Husk derfor ff-parameteren.

#### Eksempler:

```
settime("07:30:15")
```

```
settime("10:20")
```

```
settime("0"): Uret nulstilles.
```

#### gettime\$

er en funktion, som angiver klokkeslettet i formatet hh:mm:ss.t

#### Eksempler:

```
PRINT gettime$: svar er f.eks.: 9:32:50.4
```

```
digitalur:
```

```
PAGE
```

```
USE system
```

```
LOOP
```

```
PRINT AT 1,30: gettime$,
```

```
ENDLOOP
```

#### getscreen(<skærm\$>)

er en procedure, som tager en kopi af det, der vises på tekstskærmen, og gemmer det i strengen **skærm\$**. **skærm\$** skal have plads til mindst 1505 tegn. Det gøres ved f.eks. **DIM skærm\$ OF 1505**

Indholdet af strengen **skærm\$(1:1505)**:

skærm\$(1)	kantfarve
2)	baggr. farve
3)	markør farve
4)	markør: linie - 1
5)	markør: kolonne - 1
6:1505)	tekst- og farveinformation

Tekst- og farveinformation består af 500 sekvenser á 3 oktetter:

tegn 1	For hver to tegn
tegn 2	lagres deres farve.
2. 1.	Hver farve har 4 bits
farve	- tilsammen en oktet.

Se programeksempler efter **setscreen(<skærm\$>)**.

#### setscreen(<skærm\$>)

er en procedure, som laver et billede på tekstskærmen. Billedinformationen er indeholdt i strengen **skærm\$**. Strengen skal indeholde mindst 1505 tegn. Se **getscreen(<skærm\$>)**.

#### Programeksempel 1:

```
DIM a$ of 1505, b$ of 1505
USE system
...
...
getscreen(a$)
...
...
getscreen(b$)
a$:=a$(1:725)+b$(726:1505)
...
setscreen(a$)
```

#### Bemærk:

På to valgte tidspunkter under programudførelsen gemmes tekstskærmens indhold, henholdsvis i strengene **a\$** og **b\$**. Senere sammensættes en streng af **a\$**'s første 725 tegn (dvs. farve- og markørinformation, samt **a\$**-skærm billedets første 12 linier) og af **b\$**'s sidste 780 tegn (dvs. **b\$**-skærmens 13 nederste linier). Det sammensatte billede præsenteres til slut på skærmen.

#### Programeksempel 2:

```
PROC hjælp CLOSED
DIM s1$ OF 1505, s2$ OF 1505
USE system
getscreen(s1$) // red skærm billedet
OPEN FILE 10, "hjælpe skærm", READ
READ FILE 10: s2$
CLOSE FILE 10
setscreen(s2$) // præsenter hjælp
WHILE KEYS=CHRS(0) DO NULL
setscreen(s1$) // gammelt billede igen
ENDPROC hjælp
```

#### hardcopy(<enheds\$>)

er en procedure, der udskriver tekstskærmens indhold på den enhed, som angives i strengen **enheds\$**. Udskriften begynder med et linieskift.

#### Eksempel:

```
hardcopy("lp:") Tekstskærmens indhold udskrives på printer.
Ordren har samme effekt som <CTRL-P>.
```

#### currow og curcol

er to funktioner, som returnerer henholdsvis markørens nuværende linie og nuværende kolonne.

**Eksempler:**

**række:=currow; kolonne:=curcol**  
**PRINT AT 0,curcol-5: navn\$**

**bell(<varighed>)**

er en funktion, som aktiverer COMAL's "klokke". Størrelsen **varighed** skal være et heltal mellem 1 og 255. 1 svarer til 0.15 sekunder. Ved opstart udføres f.eks. **bell(3)**.

**Eksempel:**

**bell(10)** Klokke i 1.5 sek.

**free**

er en funktion, der returnerer antal frie oktetter i arbejdslageret. En mere fyldestgørende oversigt over arbejdslagerets udnyttelse fås med kommandoen **SIZE**, men da **SIZE** er en kommando, kan den ikke benyttes som sætning under programkørsel.

**Eksempel:**

**PRINT free**

**defkey(<nr>,<tekst\$>)**

er en procedure, som benyttes til at omdefinere funktionstasternes betydning. Tasterne har numrene 1,...,8,11,...18. Numrene 1 - 8 er normalt aktive som angivelse af funktionstasterne <f1> - <f8>, men når et program er under udførelse, vil funktionstasterne svare til numrene 11 - 18. Strengen **tekst\$** kan højst bestå af 32 tegn.

Proceduren **showkeys** vil udskrive en liste over funktionstasternes betydning.

**Eksempler:**

Ved opstart udføres f.eks.

**defkey(6,"LIST ")** Et tryk på <f6> giver ordet LIST

<f3> og <f4> kan f.eks. efter omdefinition udnyttes under skrivning af procedurer:

**defkey(3,"AUTO"13""13"PROC ")**

**defkey(4,"ENDPROC"13""141"SCAN"13""")**

<f3> vil bevirke:

AUTO

0010

0020 PROC

<f4> vil bevirke:

XXXX ENDPROC

(Afbryde AUTO-nummerering)

SCAN

(som efterser procedurens struktur og muliggør procedurens anvendelse som kommando)

**Programeksempel:**

**USE system**

**defkey(15,"COMAL for Folket"13""")**

**INPUT "Hvad siger du ": tekst\$**

**PRINT tekst\$**

Hvis <f5> tasten trykkes ned som svar på INPUT-sætningen, vil systemet opfatte det som om, teksten er indlæst fra tastaturet, og udskrive den.

**showkeys**

er en procedure, som udskriver en liste over funktionstasternes betydning.

**serial(TRUE eller FALSE)**

er en procedure, som dirigerer, om kommunikation er til den serielle port eller til Commodore IEEE-488 modulet, hvis dette modul er tilsluttet i computeren.

**Eksempler:**

**serial(TRUE)** til den serielle port

**serial(FALSE)** til Commodore IEEE-488 modulet

**setprinter(<attributter\$>)**

er en procedure, som bruges til at vælge enhedsnummer og attributter for den tilkoblede printer. Udskrift til printeren (lp:) vil derefter foregå efter reglerne, angivet ved attributterne. Angivelsen sker i en streng ved procedurekaldet.

Mulige printer attributter:

/a- omsæt ikke fra C64 ASCII til standard ASCII

/a+ omsæt fra C64 ASCII til standard ASCII

/l- ikke linieskift efter vogn retur

/l+ linieskift efter hver vogn retur

/t- overse 'time out' signal og fortsæt udskrift

/t+ afbryd med fejlmelding, hvis tiden udløber

Sekundære adresser for Commodore MPS 801 (delvis også MPS 802) printer: (Se i vejledningen for andre printere.)

```
/s-   ingen sekundær adresse benyttes
/s0   skriv data, som de modtages
/s1   skriv data i et tidligere defineret format
/s2   gem format oplysninger
/s3   antal linier pr. side
/s4   tillad forklarende udskriftsmeddelelser
/s5   definer et programmerbart tegn
/s6   antal tomme linier mellem hver beskrevet linie
/s7   skriv med små bogstaver
```

Ved opstart i COMAL er "lp:" defineret ved enhedsnummer og attributterne: **u4:/a-/l-/t+/s7**.

Printer MPS 801 kan på bagpanelet stilles som enhed 4 eller enhed 5.

#### Eksempler:

**setprinter("u5:/s0")** "lp:" betyder herefter enhed 5; store bogstaver i udskrift.

**setprinter("lp:/a+/l-")** omsæt til ASCII, ikke lineskift

Procedure til definition af antal linier pr. side på MPS 802 printer:

```
PROC side'802(linier'pr'side) CLOSED
  OPEN FILE 1,"lp:/s3",WRITE
  OPEN FILE 2,"lp:",WRITE
  PRINT FILE 1: CHR$(linier'pr'side),
  PRINT FILE 2: CHR$(147),
  CLOSE
ENDPROC side'802
```

#### setrecorddelay(<varighed>)

er en procedure, som laver en COMAL pause under skrivning til en randomfil. <varighed> angives i millisekunder. Diskoperativsystemet skal have tid til at skrive en blok ned på disketten, før COMAL systemet kan få lov til at sende en ny positioneringsordre. Det er sjældent nødvendigt at bruge proceduren. Ved opstart udføres automatisk en **setrecorddelay(50)**, med mindre Commodore IEEE modulet er isat sammen med COMAL kapslen. Da vil der udføres en **setrecorddelay(0)**.

#### setpage(<heltal>)

er en procedure, som definerer hvilken side (eng. *overlay*), ordrene PEEK og POKE referer til. Se i øvrigt kapitel 8 for mere information. Hjælpeprogrammet **showlibs** på demonstrationsdisketten/kassetten udnytter bl.a. denne procedure.

## PAKKEN FONT

Pakken **font** indeholder 6 procedurer, som benyttes til at definere nye skærmtegn. Man kan udskifte et helt tegnsæt eller blot enkelte tegn.

Pakken aktiveres ved ordren

### USE font

Pakken har indflydelse på 4 tegnsæt med numrene:

0: Brugerdefineret	læse/skrive
1: Brugerdefineret	læse/skrive
2: store bogstaver/grafiske tegn	læse
3: store/små bogstaver	læse

På Commodore 64 arbejdes med dobbelt tegnsæt. Normalt benytter COMAL tegnsæt nummer 3. Ved at trykke på <SHIFT C=> skiftes frem og tilbage mellem tegnsæt nummer 2 og tegnsæt nummer 3. Disse to tegnsæt findes fast i computerens ROM-lager, så dem kan man ikke ændre på.

Med font-pakken er det muligt at tilføje et nyt dobbelt tegnsæt med numrene 0 og 1. Dette tegnsæt anbringes i et beskyttet område af computerens RAM-arbejdslager.

Der er nu flere muligheder at vælge imellem:

1. Man kan flytte en kopi af computerens normale tegnsæt op som det brugerdefinerede tegnsæt, og ændre på enkelte tegn.
2. Man henter et helt nyt tegnsæt ind fra en diskette eller bånd og anbringer det som det brugerdefinerede tegnsæt, som med øjeblikkelig virkning træder i kraft. Det kræver selvfølgelig, at man har en diskettestation og en diskette med et nyt tegnsæt, men det er også muligt at skabe dit eget tegnsæt og derefter anbringe det på diskette til senere brug.

#### Bemærkninger:

- \* de benyttede tegnsæt er **skærmtegn**, hvis tegnkode afviger fra ASCII-værdierne. Se appendiks A for standard skærmkoder og ASCII-koder.

Følgende kommando vil udskrive samtlige standard skærmtegn (skærm billedet starter i adresse 1024):

**for i=0 to 255 poke 1024+i,i**

- \* Det brugerdefinerede tegnsæt benyttes også af proceduren **plottext** fra grafik-pakken.

- \* Da en printer skriver med sit eget tegnsæt, vil **font** ingen virkning have på PRINT og LIST udskrift på printer. Derimod vil <CTRL-D> (**printscreen**) bevirke, at en nøjagtig kopi af grafikskærm billedet udskrives på MPS 801 kompatibel printer.

### Eksempel på udskiftning af et tegn:

Først henter vi et tegn fra standard tegnsættet for at se, hvordan det er lagret i et rastermønster på 8x8 prikker.

Nedenstående program kan f.eks. benyttes:

Tegnet hentes med proceduren **getcharacter**. Resten af programmet er tilføjet for at få en pæn udskrift med tegnet i en 8x8 matrix. Vi lader strengfunktionen **bin\$** omsætte de enkelte tegn i det hentede rastermønster til et binært tal. Disse tal opskrives pænt under hinanden og danner tegnets bitmønster:

```
0010 // save "@0:hent'tegn"
0020 USE font
0030 DIM raster$ OF 8
0040 PAGE
0050 INPUT "Tegnsæt : ": valg#
0060 INPUT "Tegn nr. : ": tegn#
0070 PRINT
0080 getcharacter(valg#,tegn#,raster$)
0090 FOR i:=1 TO 8 DO
0100   PRINT TAB(12),bin$(ORD(raster$(i)))
0110 ENDFOR i
0120
0130 FUNC bin$(tal) CLOSED
0140   DIM bintals OF 8
0150   bintals:=""00000000"
0160   bit:=1
0170   FOR i#:=8 TO 1 STEP -1 DO
0180     IF tal BITAND bit THEN bintals(i#):="1"
0190     bit:=bit/2
0200   ENDFOR i#
0210   RETURN bintals
0220 ENDFUNC bin$
```

Afprøv programmet. Vælg et tegn fra det dobbelte standard tegnsæt: 2 eller 3. Da der endnu ikke er anbragt brugerdefinerede tegn, vil et forsøg på at hente fra tegnsæt 0 eller 1 resultere i en fejlmeddelelse. Tegnet **a** har f.eks. nummeret 1 i tegnsæt nummer 3.

Herefter laver vi et brugerdefineret tegnsæt ved blot at flytte en kopi af standard tegnsættet op som brugerdefineret tegnsæt. Da der ikke i forvejen er anbragt et brugerdefineret tegnsæt i det beskyttede lagerområde, vil kommandoen **linkfont** have denne effekt. Skriv derfor:

```
use font
linkfont
```

Herved skabes et brugerdefineret tegnsæt (0 og 1), som benyttes med det samme. Desuden skjules det gamle skærm billede, fordi der samtidig skabes et ekstra skærm billede, som benyttes sammen med det nye tegnsæt. Man kan altid vende tilbage til standard tegnsættet og standard skærm billedet med ordren DISCARD (som bevarer et eventuelt program i lageret) eller med NEW.

Man kan nu ændre på tegnene i det dobbelte tegnsæt 0 - 1. Det næste lille program indlæser tegnets udseende fra DATA-sætninger og anbringer det med ordren **putcharacter** som det nye tegn i det valgte tegnsæt.

Eksemplets DATA-sætninger danner et **a**. Dette tegn kan erstatte et hvilket som helst tegn i tegnsæt 0 eller 1. Hvis du ikke har dansk tegnsæt på din computer, kan det anbringes som tegn nummer 28 i tegnsæt nummer 1. Da vil et tryk på "pund"-tasten resultere i et **a** på skærmen.

Forsøg med udskiftning af nogle tegn. Bemærk den øjeblikkelige virkning på skærm billedet.

```
0010 // save "@0:gem'tegn"
0020 USE font
0030 DIM raster$ OF 8
0040 FOR i:=1 TO 8 DO
0050   READ oktet
0060   raster$(i):=CHR$(oktet)
0070 ENDFOR i
0080 PAGE
0090 INPUT "Tegnsæt : ": valg#
0100 INPUT "Tegn nr. : ": tegn#
0110 putcharacter(valg#,tegn#,raster$)
0120
0130 DATA %00000000
0140 DATA %00000000
0150 DATA %00111110
0160 DATA %01101110
0170 DATA %01111110
0180 DATA %01110110
0190 DATA %01111100
0200 DATA %00000000
```

## Udskiftning af helt tegnsæt

Hvis man har et dobbelt tegnsæt på diskette, hentes det ind med ordrene:

```
discard (for at slette tidligere linkfont)
use font
loadfont(<filnavn$>)
```

hvor <filnavn\$> er diskette- eller båndfilens navn. Herefter vil det nye tegnsæt og skærbillede anvendes som det aktuelle tegnsæt og skærbillede.

**Pakken font indeholder procedureerne:**

```
linkfont
loadfont(<filnavn$>)
savefont(<filnavn$>)
keepfont
getcharacter(<tegn$>,<tegn>,<raster$>)
putcharacter(<tegn$>,<tegn>,<raster$>)
```

## UDDYBNING:

### linkfont

er en procedure, som benyttes ved definition af et nyt dobbelt tegnsæt med numrene 0 og 1. Proceduren bør kun benyttes som kommando, da programkørsel ikke kan fortsætte efter en **linkfont**-sætning.

- \* Der reserveres plads øverst i arbejdslageret til det nye tegnsæt og ekstra skærbillede (4000 oktetter til tegnsæt og 1000 oktetter til skærbillede).
- \* Ekstraskærmen bliver det aktuelle skærbillede og slettes.
- \* Da variabeltabellen i arbejdslageret slettes ved overskrivning med det nye tegnsæt, er alle COMAL og pakkenavne ikke-erklærede.
- \* Hvis **linkfont** ikke har været kaldt før, enten direkte eller indirekte gennem **loadfont**, kopieres standardtegnsettet (2-3) over som det nye tegnsæt (0-1).
- \* Hvis **linkfont** har været kaldt før, sker ingenting. Man kan altså ikke overskrive et eksisterende brugerdefineret tegnsæt med en ny **linkfont**-kommando. Det brugerdefinerede tegnsæt må først fjernes med ordrene DISCARD eller NEW. Enkelttegn kan derimod udskiftes med ordren **putcharacter**.
- \* Det dobbelte tegnsæt opfattes som en del af et eventuelt COMAL-program. Når programmet gemmes med SAVE kommandoen, gemmes brugertegnsettet sammen med det som én fil. Når programmet senere hentes ind med LOAD, indlæses tegnsættene også og anvendes (også før programmet køres!).

### loadfont(<filnavn\$>)

er en procedure, som indlæser et tegnsæt med navnet <filnavn\$> fra diskette eller bånd. Først udfører **loadfont** automatisk en **linkfont**, som reserverer plads i arbejdslageret til det hentede tegnsæt og ekstra skærbillede.

**loadfont** erstatter et eventuelt eksisterende brugertegnsæt med det nye indlæste.

Det nye tegnsæt og skærbillede anvendes herefter som det aktuelle tegnsæt og skærbillede.

### savefont(<filnavn\$>)

er en procedure, som kopierer det brugerdefinerede dobbelt-tegn-sæt i arbejdslageret og gemmer det på diskette eller bånd under navnet <filnavn\$>.

### keepfont

er en procedure, som benyttes til at "fastfryse" et brugerdefineret tegnsæt, så det ikke kan slettes ved DISCARD eller NEW. Man vil være nødt til at slukke for computeren for at vende tilbage til computerens standard tegnsæt.

- \* **loadfont** virker stadig, så et hermed nyt indlæst tegnsæt vil "fastfryses".
- \* Efter **keepfont** gemmes tegnsættene IKKE sammen med et COMAL program ved kommandoen SAVE.

### getcharacter(<tegn\$>,<tegn>,<raster\$>)

er en procedure, som henter et raster-billede af tegnet med skærmkoden <tegn> fra <tegn\$>. Billedet hentes i strengvariablen <raster\$>, som er 8 tegn langt.

Tilladte værdier:

```
<tegn$>: 0, 1, 2 og 3
<tegn> : 0, 1, ..., 255
```

### Eksempler:

**getcharacter(3,1,raster\$)** Tegnet **a** hentes fra tegnsæt 3.

### DIM a\$ OF 8

Tegnet **D**'s rasterbillede hentes og udskrives.

### USE font

```
getcharacter(2,4,a$)
PRINT a$
```

Udskrift: **XLFFFLX**



**putcharacter(<tegn>,<tegn>,<raster>)**

er en procedure, som udskifter tegnet med skærmboden <tegn> i <tegn> med raster-mønstret i strengen <raster>.

Tilladte værdier:

<tegn> 0, 1  
<tegn> 0, 1, ..., 255

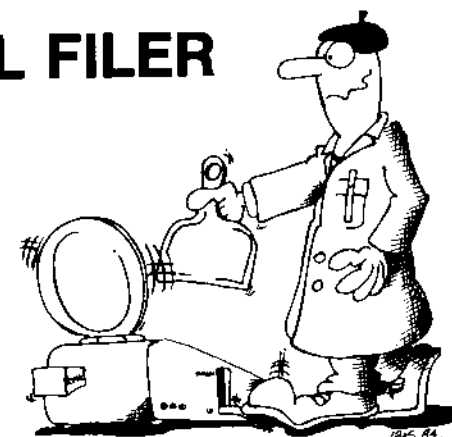
**Eksempler:**

**putcharacter(1,5,""0""0"<FFF<"0"" )**

I det ekstra tegnsæt 1 anbringes tegnet 0 som skærmtegn nummer 5.

## Kapitel 6 -

## COMAL FILER



### HVAD ER EN FIL?

Efter som man begynder at bruge sin computer på flere og flere områder, vil det være bekvemt at kunne oprette registre til at gemme informationer. Det kan være breve, forretningsaftaler, økonomiske oversigter, adresselister, måle- og beregningsresultater, bogfortegnelser eller andre data, som skal benyttes flere gange.

Selvfølgeligt er det muligt at købe kommercielt fremstillet "database"-programmer til disse opgaver. Ikke desto mindre foretrækker mange computerejere at skrive deres egne programmer, som skræddersys til at opfylde netop deres behov.

Ordet *fil* stammer fra det engelske ord *file*, som betyder *register*. En *fil* er en samling data, der er organiseret til opbevaring og genbrug. I forbindelse med mikrocomputere opbevares dataene på et *Datasette-bånd* eller en *diskette*. Da seriøs filhåndtering normalt kræver, at man benytter en diskette, vil dette kapitel hovedsageligt omhandle filbehandling med en diskettestation.

Filer kan organiseres på mange måder. Det er tit tilstrækkeligt at gemme emnerne som en række *poster* umiddelbart efter hinanden på en *sekventiel fil*. Sekventielle filer er nemme at håndtere og kræver ikke detaljeret, forudgående planlægning med hensyn til, hvor meget hver enkelt post skal fylde. Brugen af sekventielle filer gør det på den anden side nødvendigt at læse hele filen ind i computerens arbejdslager, når man skal hente oplysninger fra den. Og man skal for det meste gemme hele filen igen, hvis der er rettet i dens oplysninger. Det er tilstrækkeligt at arbejde med sekventielle filer, hvis filerne blot ikke bliver for store.

Hvis man arbejder med filer med *direkte tilgang* (eng. *random access files*), er det ikke nødvendigt at håndtere hele filen på én gang. Ved filer med direkte tilgang behøver man kun at hente den lille del af filen ind, som ønskes udskrevet, ændret eller på anden måde behandlet. I dette tilfælde skal man imidlertid nøje planlægge og afsætte plads til hver *post* (eng. *record*). Hvis man på forhånd ved, hvor meget hver post fylder, er det muligt kun at hente eller gemme en enkelt post ad gangen. Tilgangen til informationen på disketten kan således gøres væsentligt hurtigere ved at bruge filer med direkte tilgang. Filer med direkte tilgang kan ikke anvendes på Datasettebånd. Filer med direkte tilgang til de enkelte poster benyttes til opbevaring og håndtering af store, velorganiserede datamængder.

I dette kapitel vil vi omtale flere vigtige eksempler på brug af COMAL filer:

- \* gemme og hente *programmer og procedurer*
- \* en *adresse liste* på en sekventiel fil
- \* et *varelager* på en fil med direkte tilgang
- \* overflytning af filer mellem disketter

Kapitlets demonstrationsprogrammer findes på disketten (eller båndet), som følger med COMAL kapslen.

## GEM OG HENT PROGRAMMER OG PROCEDURER

Når man rigtigt kommer i gang med at skrive programmer, sker der det, at de bliver større og større. Men man opdager også, at mange af de udførte operationer går igen: gemme data, hente data, udskrive tabeller, tegne titelbillede, hente et brugersvar fra tastaturet, osv. Det vil snart falde helt naturligt at anbringe disse operationer i COMAL procedurer. Senere kan de samme procedurer benyttes igen med ingen eller kun få ændringer.

Det vil være passende her at opsummere nogle af de COMAL kommandoer til diskettstationen, som gør opbygningen af nye programmer og procedurer særlig nemt.

	HELE PROGRAMMER	PROGRAM UDSNIT
GEMME:	SAVE "<filnavn>"	LIST <udsnit> "<filnavn>"
	LIST "<filnavn>"	
	SAVE "testfil"	LIST udskrift "udskrift.1"
	LIST "testfil"	LIST 100-790 "udskrift.2"
HENTE:	LOAD "<filnavn>"	MERGE <fra linie> "<filnavn>"
	ENTER "<filnavn>"	ENTER "<filnavn>"
	LOAD "testfil"	MERGE "udskrift.1"
	ENTER "testfil"	MERGE 1100 "udskrift.1"
		MERGE 700,5 "udskrift.2"
		ENTER "udskrift.1"

Det skal også nævnes, at kommandoen DISPLAY kan benyttes til at gemme hele programmer og procedurer eller dele deraf på diskette. Kommandoen **DISPLAY 10-100 "afsnit3"** gemmer programlinierne 10-100 uden linienumre som en almindelig sekventiel fil. Filen kan (kun) hentes igen med ordrene INPUT FILE eller GET\$. Andre skrivemåder (som ved LIST) er også tilladte. DISPLAY kommandoen kan bruges til at uddrage en sekventiel fil fra et COMAL program. Denne fil kunne så senere hentes ind i et tekstbehandlingssystem som f.eks. EASYSCRIPT.

Lad os hurtigt gennemgå, hvordan man gemmer og henter *hele programmer*. Betragt venstre søjle i ovenstående tabel:

Kommandoerne SAVE og LOAD bør allerede være kendte. SAVE benyttes til at overføre en kopi af det COMAL program, som i øjeblikket er i arbejdslageret, til diskette. Kopien gemmes på disketten i kompakt, binær form. Syntaksen er **SAVE "<filnavn>"**, hvor <filnavn> er dit valg af navn til programmet på disketten.

Hvis et filnavn begynder med @ (kaldet "snabel-a", "commercial-a" eller meget uofficielt "hundelorten"), har navnet en speciel betydning. Hvis en fil med det efterfølgende navn allerede eksisterer på diskette, vil den slettes, og den nye fil vil blive gemt på dens plads under det samme filnavn. Det anbefales, at man altid bruger **SAVE"@0:<filnavn>"** for at gemme programmerne på drive 0 oveni en eksisterende fil.

Kommandoen LOAD er SAVE's modsatte operation. Syntaksen er **LOAD "<filnavn>"**, og en kopi af COMAL programmet med dette navn vil hentes fra disketten ind i arbejdslageret. Hvis man henter et program med LOAD kommandoen, vil et eventuelt eksisterende program i computerens arbejdslager slettes! Bemærk, at kun programfiler (angivet ved prg i diskettens programkatalog) kan hentes med LOAD.

En programkopi kan også gemmes med kommandoen LIST, hvorved kopien gemmes som en sekventiel ASCII-fil. Den må senere hentes ind i arbejdslageret med kommandoerne ENTER eller MERGE.

Kommandoerne RUN og CHAIN vil også kunne benyttes til at hente programmer ind i det indre lager fra diskette. Se en mere uddybende beskrivelse af disse ordrer i kapitel 4.

Yderligere vil en *lukket procedure*, som er gemt med SAVE, kunne hentes ind under programudførelse som en **EXTERNAL procedure**. (Se under EXTERNAL og CLOSED i kapitel 4.)

Lad os nu kaste et blik på søjlen **PROGRAM UDSNIT**. Informationen der kan spare dig megen tid, så læg godt mærke til den.

Forestil dig, at du har lavet en genial procedure **kvik'save**, som lynhurtigt udvælger og gemmer en liste med varer og priser på diskette. Den procedure er så generel, at den kan anvendes i mange

andre programmer. Hvis den er på mere end nogle få liniers længde, er det besværligt at skulle taste den ind, hver gang den benyttes. Den procedure skal gemmes på diskette med kommandoen **LIST kvik'save "prc.kvik'save"**. Hvis du gør det og bagefter forlanger et katalog over disketten med kommandoen CAT, vil du bemærke, at filen **prc.kvik'save** er gemt som en sekventiel fil og dermed ikke som en programfil. Den kan altså ikke hentes med LOAD som en programfil. For at få den anbragt i et program i arbejdslageret må man bruge enten MERGE eller ENTER. Disse kommandoer vil blive beskrevet om lidt.

---

Det er også muligt at gemme et programudsnit ved at angive de tilhørende linienumre. F.eks. kan linierne 1000-1090 gemmes med kommandoen **LIST 1000-1090 "1st.forgrening"**. En udskrift af diskettekataloget vil bekræfte, at udsnittet er gemt som en sekventiel fil, som igen kan hentes ind i andre programmer med MERGE eller ENTER.

---

Her kommer så fremgangsmåden. Når den gemte procedure (eller programudsnit) skal bruges igen, kan det gøres på følgende vis:

- \* Skriv **merge "prc.kvik'save"**. En kopi af proceduren vil nu hentes ind fra disketten og anbringes efter det program, som befinder sig i arbejdslageret. Det vil starte med et linienummer 10 større end programmets sidste linie, *uafhængigt af* hvilke linienumre det måtte være gemt med på disketten!
- \* Man kan også skrive **merge 1100 "prc.kvik'save"**. I så tilfælde vil proceduren blive anbragt i det eksisterende program fra linie 1100 og fremefter. Linienumre vil være 1100, 1110, 1120,... Hvis man ønsker linienumrene i intervaller på 5, behøver man blot at skrive **merge 1100,5 "prc.kvik'save"**.

---

**ADVARSEL:** Vær forsigtig med at MERGE en procedure ind midt i et eksisterende program. Du må være sikker på, at der er plads nok til proceduren med de valgte linienumre. Ellers vil proceduren blandes sammen med de øvrige instruktioner eller slette i forvejen eksisterende linier, hvis linienumrene falder sammen. Med kommandoen RENUM kan man som regel skabe den nødvendige plads i forvejen.

---

- \* I tilfælde af, at du har gemt et helt program med LIST, foretrækker du måske at hente det ind med kommandoen ENTER. Hvis man f.eks. skriver **ENTER "udskrift.1"**, bliver en kopi af den sekventielle fil **udskrift.1** læst ind i programlageret, efter at et eventuelt eksisterende program er slettet.

Hvis du har arbejdet med andre programmeringssprog og operativsystemer, vil du forstå at værdsætte, hvor nyttige disse muligheder er under udarbejdelsen af programmer.

## SEKVENTIELLE FILER

I kapitel 3, program 19 så vi et simpelt eksempel på, hvordan sekventielle filer kunne udnyttes til at opbevare et talmateriale. Vi gennemgik, hvordan en fil skal åbnes, før man gemmer/henter data på/fra den. Derefter skal filen lukkes.

Strukturerne var:

### åbne en fil, gemme data og lukke filen

```
OPEN FILE <filnr>,<filnavn$>,WRITE
...
...
PRINT FILE <filnr>: <et dataelement>
...
...
CLOSE FILE <filnr>
```

### åbne en fil, hente data og lukke filen

```
OPEN FILE <filnr>,<filnavn$>,READ
...
...
INPUT FILE <filnr>: <et variabelnavn>
...
...
CLOSE FILE <filnr>
```

Vi vil her bygge videre på dette indledende eksempel og vende opmærksomheden mod et praktisk problem: At lave et *kartoteks-program* til opbevaring af navne, adresser og telefonnumre. Det følgende eksempel viser, hvilken slags oplysninger vi vil gemme og de anvendte variabelnavne:

<b>Eksempel</b>	<b>strengvariabel</b>
Peter Jensen	navn\$()
Fynsgade 5	gade\$()
8800 Viborg	by\$()
06-621234	telefon\$()

Bemærk her, at vi har gjort plads til 100 dataposter, hver med 4 felter: navne, adresser og telefonnumre. Vort program skal kunne behandle op til 100 navne med adresser og telefonnumre. Hvert hold oplysninger kaldes en

post, og de individuelle variabler kaldes postens *felter*. I vort tilfælde består hver post altså af 4 felter.

Programmet skal kunne:

- \* HENTE data-filen ind i arbejdslageret
- \* LAVE en data-post med navn, adresse og telefonnr
- \* UDSKRIVE alle poster på filen
- \* SØGE i filen efter bestemte poster
- \* SORTERE posterne i alfabetisk orden
- \* ÆNDRE en post
- \* SLETTE en post
- \* GEMME en fil på diskette

Selvfølgelig er der andre operationer, man kunne forestille sig udført på en fil, men for overskuelighedens skyld indskrænker vi os til de følgende. Når du har forstået de beskrevne procedurer, vil du snart være i stand til at udvide og ændre i programmet, sådan at det lige præcist opfylder dine behov. Prøv også at lave programmet mere brugervenligt, så forkerte ordrer ikke får programmet til at standse. Alle, der har modtaget denne vejledning sammen med COMAL kapslen, har samtidigt modtaget demonstrationsdisketten eller båndet med dette program **adr liste demo**. Programlistningen findes desuden i appendiks H.

Programmet begynder med en linie, som henviser til filens navn:

```
0010 // save "@0:adr liste demo"
```

Bemærk, at vi har brugt en kommentarsætning (//) og medtaget **save** foran filnavnet. Dette lille trick gør det nemmere at gemme videreudviklinger af programmet, som arbejdet skrider frem. Man behøver blot flytte markøren op til denne linie. Derefter slettes linienummer og kommentarstreger ved at indtaste nogle mellemrumstegn (eller ved brug af <INST/DEL>). Når du trykker på <RETURN>, gemmes den nye version. @ symbolet foran navnet bevirker som tidligere nævnt, at en eventuel programfil med samme navn overskrives. Det anbefales at bruge drive nummeret, her drive 0, når "snabel-a" bruges i et filnavn.

---

ADVARSEL: Vær forsigtig med brugen af dette lille kneb; man kan miste en programfil. Vær sikker på, at du har en sikkerhedskopi af dit program og opdater den fra tid til anden. Afhold dig fra at benytte demonstrationsdisketten eller -båndet til at gemme nye versioner. LOAD demonstrationsprogrammet og gem senere ændringer på en anden diskette eller et andet bånd.

---

De følgende linier sørger for at **dimensionere** programmets tabeller og strenge:

```
0020 DIM svar$ OF 1, navn$(100) OF 40
0030 DIM gade$(100) OF 40, by$(100) OF 40
0040 DIM telefon$(100) OF 20, flag$ OF 40
0050 DIM lede$ OF 40, streng$ OF 150
0060 nummer:=0 // antal gemte dataposter
```

Bemærk her, at vi har gjort plads til 100 dataposter, hver med 4 felter navn, gade, by og telefonnummer. Hver af disse felter må bestå af op til 40 tegn. Dette valg betyder, at den sekventielle fil vil fylde op til 100x4x40 eller 16 kilooktetter i lageret. Da der er ca. 30 kilooktetter til rådighed, og programmet kun fylder ca. 4 kilo, er der rigelig plads, hvis du skulle få brug for at erklære et øget antal poster.

Herefter følger indledningen, som beskriver programmet:

```
0070 PAGE
0080 PRINT "Dette program skal vise brugen af"
0090 PRINT "SEKVENTIELLE FILER. Det kan benyttes"
0100 PRINT "til at lave en liste over navne,"
0110 PRINT "adresser og telefonnumre."
0120 PRINT "Hver oplysning har formatet:"
0130 PRINT
0140 PRINT "   navn"
0150 PRINT "   gade"
0160 PRINT "   by"
0170 PRINT "   telefonnummer"
0180 PRINT
0190 PRINT
0200 PRINT "Tryk en eller anden taste..."
0210
0220 afvent'taste
0230
```

Sætningen **PAGE** rengør skærm billedet, som derefter fyldes med information. Derefter kaldes proceduren **afvent'taste**. Denne procedure kan måske også anvendes i dine egne programmer:

```
2240 PROC afvent'taste
2250 PRINT
2260 PRINT "< >...";
2270 REPEAT
2280   svar$:=KEY$
2290 UNTIL svar$<>CHR$(0)
2300 PRINT AT 0,2: svar$
2310 ENDPROC afvent'taste
```

Linien < >... udskrives på skærmen som tegn på, at computeren venter på et svar fra brugeren. Hvis det ikke falder i din smag, kan du slette det. **REPEAT - UNTIL** løkken udføres, indtil en taste trykkes ned. Først da vil indholdet af COMAL funktionen **KEY\$** blive forskellig fra **CHR\$(0)**. Når en taste trykkes ned på tastaturet, vil **KEY\$** opfange det indtastede tegn og videregive det til variabelen **svar\$**. Da **svar\$** nu ikke længere er **CHR\$(0)**, hoppes ud af **REPEAT - UNTIL** løkken og fortsættes i næste linie. **PRINT AT 0,2: svar\$** sætningen gør, at det indtastede svar vises inden i de kantede parenteser < >.

```
0240 LOOP // hovedprogram løkke
0250 vis'menu
0260 flag$=""
0270 afvent'taste
0280 CASE svar$ OF
0290 WHEN "1"
0300     hent'fil
0310 WHEN "2"
0320     lav'post
0330 WHEN "3"
0340     list'fil
0350 WHEN "4"
0360     led'i'fil
0370 WHEN "5"
0380     sorter'fil
0390 WHEN "6"
0400     udskift'post
0410 WHEN "7"
0420     slet'post
0430 WHEN "8"
0440     gem'fil
0450 OTHERWISE
0460     PRINT "Ulovligt svar.."
0470     afvent'taste
0480 ENDCASE
0490 ENDLOOP
```

Denne løkke er den centrale, styrende del af programmet. Først kaldes proceduren **vis'menu**:

```
0500
0510 PROC vis'menu
0520 PAGE
0530 PRINT "-----HOVED MENU ----"
0540 PRINT
0550 PRINT
0560 PRINT " <1> HENT filen"
0570 PRINT " <2> LAV en post"
0580 PRINT " <3> LIST filen"
0590 PRINT " <4> LED I filen"
0600 PRINT " <5> SORTER alfabetisk"
0610 PRINT " <6> UDSKIFT en post"
```

```
0620 PRINT " <7> SLET en post"
0630 PRINT " <8> GEM den nye fil"
0640 PRINT
0650 PRINT
0660 PRINT "Antal poster: ";nummer
0670 IF nummer=0 THEN flag$="HENT eller LAV en fil"
0680 PRINT
0690 PRINT flag$
0700 ENDPROC vis'menu
```

Denne procedure sletter det gamle skærbillede og udskriver en menu med valgmulighederne. Desuden udskrives antal oprettede poster. Strengvariabelen **flag\$**, som i programmet benyttes til at give brugeren forskellige oplysninger, sættes lig **HENT eller LAV en fil**, hvis der ikke endnu er oprettet poster i lageret. Denne besked udskrives under menuen som vejledning. Tilbage i hovedprogrammet (linie 260) sættes **flag\$** igen lig den tomme streng, så den senere kan benyttes til andre formål. Derefter ventes igen på, at brugeren taster et tal for at vælge mellem fortsættelsesmulighederne. Hvis der ikke svares med et tal mellem 1 og 8, skriver programmet **Ulovligt svar...** og afventer tryk på en tilfældig taste.

Lad os nu se på hver af de 8 procedurer til håndtering af filer. Den første procedure **HENTER** filen, hvis brugeren fra menuen vælger mulighed 1:

```
0710
0720 PROC hent'fil
0730 OPEN FILE 1,"adresser",READ
0740 INPUT FILE 1: nummer
0750 FOR nr:=1 TO nummer DO
0760     INPUT FILE 1: navn$(nr)
0770     INPUT FILE 1: gade$(nr)
0780     INPUT FILE 1: by$(nr)
0790     INPUT FILE 1: telefon$(nr)
0800 ENDFOR nr
0810 CLOSE FILE 1
0820 ENDPROC hent'fil
0830
```

Denne procedure kan selvfølgelig først bruges, efter at en fil er lavet og gjort tilgængelig på disketten. Normalt vil det være det første valg, man træffer ved brugen af programmet. Det er vigtigt at forstå proceduren **hent'fil**, for den viser, hvordan en sekventiel fil læses fra diskette ind i arbejdslageret. Det første, denne procedure gør, er at åbne filen med nummeret 1 og navnet **adresser**. Filen åbnes, så man kan læse fra den (**OPEN FILE 1,"adresser",READ**).

Skulle man få lyst til at give filen et andet navn, ændres navnet blot her og alle andre steder i programmet. Den nemmeste måde at gøre det er ved brug af **CHANGE** kommandoen (**change "adresser","nyt navn"**).

Vi har kaldt det første element i filen for **nummer** svarende til antal poster i filen. Dette nummer anbringes af proceduren **gem'fil** (se senere). Nu, hvor antal poster i filen er kendt, læser resten af oplysningerne ind i arbejdslageret ved hjælp af en FOR - ENDFOR løkke. (I kapitel 3 anvendtes funktionen EOF(<filnr>) til at meddele, når alle oplysninger er hentet fra filen).

Læg mærke til, at indlæsningen sker med INPUT FILE - sætninger, og at fil 1 skal lukkes med ordren CLOSE FILE 1 efter brug.

Hvis mulighed 2 vælges fra menuen, kaldes nedenstående procedure, som benyttes til at lave en ny post til filen:

```
0840 PROC lav'post
0850 PAGE
0860 PRINT ":::: LAV EN NY POST :::"
0870 PRINT
0880 PRINT
0890 IF nummer=100 THEN flag$="ikke mere plads til data"
0900 IF flag$="" THEN
0910     nummer:=1
0920     INPUT "Navn   ": navn$(nummer)
0930     INPUT "Gade   ": gade$(nummer)
0940     INPUT "By     ": by$(nummer)
0950     INPUT "Telefon ": telefon$(nummer)
0960 ENDIF
0970 ENDPROC lav'post
0980
```

Proceduren starter med at slette det gamle skærbillede og så fortælle brugeren, hvad der sker. Hvis der ikke er plads til flere poster på filen (hvis **nummer** er lig 100), sættes **flag\$**-beskeden til **ikke mere plads til data**, og de næste linier vil ikke blive udført, da **flag\$** er forskellig fra "". Hvis der derimod er færre end 100 poster, øges posttælleren (**nummer:=1**), og indtastningen af de fire personoplysninger kan begynde. Derefter vendes igen tilbage til programmets hovedløkke.

Valgmulighed 3 er en udskrift af alle oplysninger på filen:

```
0990 PROC list'fil
1000 PAGE
1010 PRINT ":::: UDSKRIFT AF FILEN :::"
1020 PRINT
1030 IF nummer=0 THEN
1040     flag$="Der er ingen filer!"
1050     PRINT
1060 ELSE
1070     FOR nr:=1 TO nummer DO skriv'post(nr)
1080 ENDIF
1090 ENDPROC list'fil
1100
```

Skærbilledet slettes, og brugerbeskeden udskrives. Hvis der ikke er nogle filer i arbejdslageret (**nummer** er lig 0). Ved hjælp af **flag\$** sendes en advarsel til brugeren. Hvis der derimod er filer i lageret, udskrives posterne enkeltvis. Efter udskrift af hver post afventer systemet, at en taster trykkes ned. Hvis en eller anden taster holdes nede hele tiden, vil samtlige oplysninger rulle frem på skærmen.

Hvis man vil *søge* efter en bestemt oplysning, vælger man mulighed 4 fra menukortet. Det er selvfølgelig stadig en betingelse, at der er en fil i arbejdslageret at søge i. Man kan benytte en hvilken som helst oplysning som *søgeord*; navn, gade, gadenummer, by eller telefonnummer:

```
1110 PROC led'i'fil
1120 PAGE
1130 PRINT ":::: LED I FILEN :::"
1140 PRINT
1150 PRINT
1160 flag$="Jeg leder i filen"
1170 INPUT "Led efter ordet ": led'efter$
1180 FOR nr:=1 TO nummer DO
1190     streng$=navn$(nr)+gade$(nr)+by$(nr)+telefon$(nr)
1200     IF led'efter$ IN streng$ THEN skriv'post(nr)
1210 ENDFOR nr
1220 flag$=""
1230 ENDPROC led'i'fil
1240
```

Efter at have slettet skærbilledet og informeret brugeren, beder proceduren brugeren om at indtaste et søgeord, som anbringes i variabelen **led'efter**. Alle poster undersøges. Hvis søgeordet findes blandt en posts oplysninger, udskrives postens indhold ved hjælp af proceduren **skriv'post(nr)**:

```
1250 PROC skriv'post(nr)
1260 PRINT
1270 PRINT AT 0,10: "-----(",nr,")"
1280 PRINT AT 0,10: navn$(nr)
1290 PRINT AT 0,10: gade$(nr)
1300 PRINT AT 0,10: by$(nr)
1310 PRINT AT 0,10: telefon$(nr)
1320 PRINT
1330 afvent'taste
1340 ENDPROC skriv'post
```

Næste procedure, som vælges fra menukortet ved at taste 5, **sorterer** oplysningerne i alfabetisk navneorden. Det kræver dog, at navnene er indtastet korrekt med efternavn som det første felt i hver post. Man kan selvfølgelig også lade proceduren sortere oplysningerne i henhold til andre personoplysninger, f.eks. gadenavne. Det kræver kun en lille ændring i proceduren:

```

1360 PROC sorter'fil
1370 PAGE
1380 PRINT ":::: SORTER ALFABETISK::::"
1390 PRINT
1400 PRINT
1410
1420 PROC ombyt(REF a$,REF b$) CLOSED
1430   c$:=a$; a$:=b$; b$:=c$
1440 ENDPROC ombyt
1450
1460 REPEAT
1470   ingen'ombytning:=TRUE
1480   FOR nr:=1 TO nummer-1 DO
1490     PRINT AT 10,1: "Sortering... ",nr
1500     IF navn$(nr)=navn$(nr+1) THEN
1510       ombyt(navn$(nr),navn$(nr+1))
1520       ombyt(gade$(nr),gade$(nr+1))
1530       ombyt(by$(nr),by$(nr+1))
1540       ombyt(telefon$(nr),telefon$(nr+1))
1550       ingen'ombytning:=FALSE
1560     ENDIF
1570   ENDFOR nr
1580 UNTIL ingen'ombytning
1590 ENDPROC sorter'fil
1600

```

Ligesom i kapitel 3 program 19 har vi her benyttet *boblesortering* som sorteringsalgoritme. I forhold til den version har vi lavet den ændring, at **ombyt** proceduren her er anbragt inden i proceduren **sorter'fil**. Det er gjort for at vise et eksempel på, hvordan en procedure kan være lokal inden i en anden procedure.

Boblesortering bygger som tidligere nævnt på sammenligning af navnene parvis. Hvis det højst nummererede navn i parret kommer alfabetisk før nummeret foran, bytter de to poster plads. På den medfølgende demonstrationsdiskette/bånd er anbragt en **quick'sort** procedure, som er noget mere effektiv, men også lidt sværere at gennemskue. Hvis du har mod på det og behov for hurtig sortering, er det et forsøg værd at indsætte **quick'sort** proceduren i stedet for den her anførte.

Det vil af og til være nødvendigt at *ændre* indholdet af en post på filen. Det gøres ved at vælge 6 fra menukortet. Proceduren **udskift'post** foretager ændringen:

```

1610 PROC udskift'post
1620 PAGE
1630 PRINT ":::: UDSKIFT EN POST :::"
1640 PRINT
1650 PRINT
1660 INPUT "Hvilken post ? ": nr
1670 IF nr<=nummer THEN

```

```

1680   skriv'post(nr)
1690   INPUT AT 14,1: "Er det den rigtige (j/n)? ": svar$
1700   PRINT
1710   PRINT
1720   IF svar$ IN "J" THEN
1730     INPUT "Navn: ": navn$(nr)
1740     INPUT "Gade: ": gade$(nr)
1750     INPUT "By: ": by$(nr)
1760     INPUT "Telefon: ": telefon$(nr)
1770   ENDIF
1780 ELSE
1790   flag$="Der er kun "+STR$(nummer)+" poster"
1800 ENDIF
1810 ENDPROC udskift'post
1820

```

Det skulle være ligetil at følge fremgangsmåden i proceduren. Man skal simpelthen vælge, hvilken post, der ønskes ændret og så derefter foretage ændringen. Bemærk igen, hvordan **flag\$** udnyttes til at få anbragt en fejludskrift under menuen.

Hvis man vælger 7 fra menuen, betyder det, at en post ønskes slettet. Denne operation udføres af nedenstående procedure:

```

1830 PROC slet'post
1840 PAGE
1850 PRINT ":::: SLET EN POST :::"
1860 PRINT
1870 PRINT
1880 INPUT "Hvilken post ? ": post
1890 IF post>nummer THEN
1900   flag$="Benyt et mindre nummer!"
1910 ELSE
1920   skriv'post(post)
1930   PRINT
1940   INPUT "Er det den rigtige (j/n)? ": svar$
1950   PRINT
1960   IF svar$ IN "J" THEN
1970     FOR nr:=post TO nummer-1 DO
1980       navn$(nr):=navn$(nr+1)
1990       gade$(nr):=gade$(nr+1)
2000       by$(nr):=by$(nr+1)
2010       telefon$(nr):=telefon$(nr+1)
2020     ENDFOR nr
2030     nummer:=1
2040   ENDIF
2050 ENDIF
2060 ENDPROC slet'post
2070

```

Efter at en fil er hentet ind arbejdslageret, sorteret eller ændret, skal man kunne gemme den igen til senere brug. Vælg mulighed 8 for at aktivere den følgende procedure:

```

2080 PROC gem'fil
2090 PAGE
2100 PRINT ":::: GEM FILEN ::::"
2110 OPEN FILE 1,"@addresser",WRITE
2120 PRINT FILE 1:nummer
2130 PRINT
2140 PRINT
2150 FOR nr:=1 TO nummer DO
2160   PRINT FILE 1: navn$(nr)
2170   PRINT FILE 1: gade$(nr)
2180   PRINT FILE 1: by$(nr)
2190   PRINT FILE 1: telefon$(nr)
2200 ENDFOR nr
2210 CLOSE FILE 1
2220 ENDPROC gem'fil
2230

```

Før oplysningerne kan gemmes på diskette, må filen først åbnes ved brug af filnummer (1 i dette tilfælde), filnavn (her **addresser**) og tilstanden WRITE.

Det vil ikke være nogen stor sag at gøre filnavnet udskifteligt med en INPUT-sætning; f.eks. **INPUT "Filnavn? ": filnavn\$** tidligt i proceduren. Husk blot også på at indføre en valgmulighed i proceduren **hent'fil**.

Proceduren **gem'fil** fortsætter, efter at være åbnet, med først at gemme filens antal poster (**PRINT FILE 1: nummer**). Som du måske husker, er det den første information, som proceduren **hent'fil** læser, når filen skal hentes igen. Derefter følger **PRINT FILE** sætninger, som udskriver hver post til den sekventielle fil. Til slut skal filen lukkes med sætningen **CLOSE FILE 1**.

Da samtlige oplysninger er udskrevet på filen ved brug af **PRINT FILE** sætninger, er oplysningerne adskilt af et **<CR>** tegn.

## FILER MED DIREKTE TILGANG - ET VAREKARTOTEK

Til at illustrere brugen af *filer med direkte tilgang* også kaldet *random* filer ("random" betyder "tilfældig" eller "vilkårlig" på engelsk) har vi valgt et enkelt program, som fører fortegnelse over et varelager. Programmet **random fil demo** findes på demonstrations-disketten. Hvis du har en diskettestation, er det en god idé at hente programmet ind i arbejdslageret og afprøve programmet, inden du læser videre.

Den første linie tjener som identifikation af programmet og letter indtastningen, når programmet gemmes på diskette. Derefter følger **dimensioneringen** af de benyttede strengvariabler og angivelsen i variabelen **maxantal** af det maksimale antal poster, programmet vil behandle. Dette antal kan ændres efter eget valg:

```

0010 // save "@:0random fil demo"
0020 DIM kode$ OF 30, vare$ OF 30
0030 DIM antal$ OF 30, stykpris$ OF 30
0040 maxantal:=25

```

Så følger en kort beskrivelse af programmet. Beskrivelsen udskrives på skærmen ved programmets start:

```

0050 PAGE
0060 PRINT ":::: FIL MED DIREKTE TILGANG :::"
0070 PRINT
0080 PRINT
0090 PRINT "Dette program er en enkel illustration"
0100 PRINT "af, hvordan man gemmer og henter"
0110 PRINT "information i en fil med direkte"
0120 PRINT "tilgang (en RANDOM fil)."

```

Efter denne indledning fortsætter udførelsen til den centrale løkke i hovedprogrammet, så snart der trykkes på en taste. Her benyttes den samme procedure **afvent'taste**, som anvendtes i det forrige program.

```

0270 REPEAT
0280   vis'menu
0290   IF svar$="1" THEN lav'post
0300   IF svar$="2" THEN hent'post
0310 UNTIL svar$="3"
0320

```

Denne løkke udskriver et lille menukort. Afhængigt af det valgte dirigeres fortsættelsen til én af procedurerne **lav'post** eller **hent'post**. Hvis man da ikke vælger at slutte programmet:

```

0330 PROC vis'menu
0340 PAGE
0350 PRINT ":::: RANDOM FIL DEMO - MENU :::"
0360 PRINT
0370 PRINT

```



```

0380 PRINT AT 0,5: "<1> LAV en post"
0390 PRINT AT 0,5: "<2> HENT en post"
0400 PRINT AT 0,5: "<3> slut"
0410 PRINT
0420 PRINT
0430 afvent'taste
0440 ENDPROC vis'menu
0450

```

Hvis 1 vælges, opretter programmet en fil med direkte tilgang til posterne. Filen tænkes at indholde en varefortegnelse:

```

0460 PROC lav'post
0470 PAGE
0480 PRINT "::::: LAV EN POST :::::"
0490 PRINT
0500 PRINT
0510 INPUT "Indtast postnummer: ": nr
0520 PRINT
0530 PRINT
0540 IF nr>0 AND nr<=maxantal THEN
0550 INPUT "varenummer: ": kode$
0560 INPUT "varenavn : ": vare$
0570 INPUT "antal : ": antal$
0580 INPUT "stykpris : ": stykpris$
0590 OPEN FILE 1,"0:varelager",RANDOM 128
0600 WRITE FILE 1,nr: kode$,vare$,antal$,stykpris$
0610 CLOSE
0620 ENDIF
0630 ENDPROC lav'post
0640

```

Den første del af proceduren holder styr på den påtænkte posts nummer (1 til maxantal). Hvis der indtastes et lovligt postnummer, udføres sætningerne i IF - ENDIF strukturen. Her indtastes de enkelte oplysninger til varefortegnelsen, hvorefter filen **varelager** åbnes.

Filen **varelager** åbnes som en fil med direkte tilgang til en hvilken som helst af dens poster. Filen åbnes med nummeret 1. De første tegn i filens navn 0: henviser til, at filen åbnes på den primære diskettstation. Hvis en ekstra diskettstation forbindes til Commodore 64, benævnes den som enhed 2: (Det har at gøre med kompatibiliteten med andre Commodore computere i 4000 og 8000 serien, som kan have dobbelte diskettstationer med numrene 0 og 1).

Det markeres til sidst i OPEN FILE sætningen, at filen åbnes som RANDOM med en postlængde på 128 tegn. Det er den påkrævede, mindste postlængde, når hver af de 4 oplysninger ifølge programmet indledende erklæringer må bestå af op til 30 tegn, i alt 120 tegn. Se nedenstående bemærkninger om postlængder.

Herefter udskrives alle oplysninger til posten med det angivne **nr**.

Bemærkninger om filer med direkte tilgang og binær repræsentation af data:

- Med ordren WRITE FILE lagres postens data i binær form på diskette, hvor tal og tekst fylder et bestemt antal oktetter:  
**heltal:** fylder 2 oktetter  
**reelle tal:** fylder 5 oktetter  
**streng:** fylder **strengens længde + 2**

De 2 ekstra oktetter ved lagring af streng tilføjes af COMAL systemet som internt regnskab med strenglængden.

- Commodore diskettstationerne 1541 og 2031 tillader kun én RANDOM fil åben ad gangen.
- I kataloget over en diskettes indhold vil en fil med direkte tilgang optræde som **røl**, en relativ fil.
- En direkte fil skal altid lukkes med ordren CLOSE uden filnummer.

Når man vil hente lagret information fra filen **varelager**, kaldes den følgende procedure med valgmulighed 2 fra menuen:

```

0650 PROC hent'post
0660 PAGE
0670 PRINT "::::: HENT EN POST FRA FILEN :::::"
0680 PRINT
0690 PRINT
0700 INPUT "Indtast postnummer (1-25): ": nr
0710 PRINT
0720 IF nr>0 AND nr<maxantal THEN
0730 OPEN FILE 2,"0:varelager",RANDOM 128
0740 READ FILE 2,nr: kode$,vare$,antal$,stykpris$
0750 CLOSE
0760 PRINT
0770 PRINT
0780 PRINT "Lageremne";nr;"er:"
0790 PRINT
0800 PRINT "varenummer: ";kode$
0810 PRINT "varenavn : ";vare$
0820 PRINT "antal : ";antal$
0830 PRINT "stykpris : ";stykpris$
0840 afvent'taste
0850 ENDIF
0860 ENDPROC hent'post
0870

```

I denne procedure anmodes brugeren om at indtaste et postnummer. Hvis nummeret er gyldigt, åbnes filen, og de fire dataelementer hentes fra posten med sætningen READ FILE. Efter at filen igen er lukket med CLOSE, udskrives oplysningerne på skærmen.

## MULIGE UDVIDELSER:

- \* Programmet kunne f.eks. udvides med en tæller, som holder regnskab med det totale antal poster på filen. Den skulle læses, så snart programmet startede og opdateres, hver gang en ny post indlæstes.  
Af andre tilføjelser, som vil styrke dette enkle program, kan nævnes:
- \* Før programmet benyttes første gang, kan filen **varelager** oprettes i sin maksimale størrelse én gang for alle (nu udvides den jo efterhånden, som nye poster kommer til). Det sker med kommandoen **CREATE**. Skriv f.eks.

**CREATE "varelager",25,128**

Herved opnås, at man på forhånd ser, om der overhovedet er plads til varefortegnelsen på disketten. Desuden bliver tilgangen til disketten ca. 10 gange hurtigere, fordi systemet ikke samtidig skal udvide filen.

- \* Alle posterne kunne nulstilles med kendt information. Det vil forhindre, at man kommer til at læse på en ikke veldefineret post. Desuden kunne det udnyttes til at give advarsel om mulig overskrivning af en post med NYTTIG information. En mulig nulstillingssekvens:

**OPEN FILE 1: "varelager",RANDOM 128  
FOR nr:=1 TO 25 DO WRITE FILE 1: SPCS(126)  
CLOSE**

(Varelageret nulstilles med mellemrumstegn. Man skal naturligvis være helt sikker på, at der ikke står noget i forvejen i denne fil, som ikke ønskes slettet!)

## FLYTNING AF EN SEKVENTIEL FIL - RUB OG STUB

Det sidste program i dette kapitel viser hvordan en sekventiel fil kan flyttes fra én diskette til en anden. Filer skrevet i maskinkode kan f.eks. være besværlige at flytte på anden måde.

Den centrale sætning er **GETS(<filnr>,<oktetter>)**. Ved hjælp af den læses ALT på disketten - også adskilletegn, som ikke indlæses med sætningen **INPUT FILE**.

Programmet åbner en brugervalgt sekventiel fil, læser hele indholdet ind i variabelen **tal\$** (dog maksimalt 5000 tegn), anmoder brugeren om at skifte diskette og udskriver derefter **tal\$** indhold til en fil med samme navn, men på den nye diskette:

```
0010 PAGE
0020 DIM navn$ OF 40
0030 INPUT "Indtast filnavn: ": navn$
0040 OPEN FILE 2,navn$,READ
0050 DIM tal$ OF 5000
0060 WHILE NOT EOF(2) DO
0070   tal$+GETS(2,1000)
0080 ENDWHILE
0090 CLOSE FILE 2
0100 PRINT tal$
0110 PRINT "Efter disketteskift trykkes en taster ned"
0120 dummy$=KEY$
0130 WHILE KEY$=CHR$(0) DO NULL
0140 OPEN FILE 3,"@0:"+navn$,WRITE
0150 PRINT FILE 3: tal$,
0160 CLOSE FILE 3
```

## FILTYPER

Som det vil fremgå ved udskrivning af en diskettes katalog, er der forskellige muligheder for lagring af filer:

<b>prg</b>	program fil
<b>seq</b>	sekventiel fil
<b>rel</b>	relativ fil (random fil)
<b>usr</b>	sekventiel fil (bruger fil)

Denne opdeling begrænser filernes udnyttelse. Hvis man f.eks. prøver at hente en fil med direkte tilgang (**rel**) med kommandoen **LOAD**, reagerer COMAL med en fejlmeddelelse. En bestemt slags filer kan udvælges fra kataloget med kommandoen **dir** **"\*=prg"**, hvorved navnene på alle programfiler udskrives.

Det vil være nyttigt for dig at lade filnavnene indeholde en angivelse af filtypen, hvis du benytter filer og diskettestation i større udstrækning.

Du kommer til at arbejde med tegnsæt, figurtabeller til spritetegninger, LISTede sekventielle filer, som indeholder programmer, procedurer eller funktioner, eksterne procedurer, udskriftsfil, tekstfiler fra tekstbehandlingsanlæg, datafiler m.m.

For at skelne disse fra hinanden, og for at gøre det muligt at udskrive alle filer af en bestemt type ved hjælp af **dir** eller **cat**, er det tilrådeligt at tilføje en *filtypekode* til hvert navn. Alt afhængig af stil kan man skrive en kode på 3 eller 4 bogstaver foran selve filnavnet; eller man kan sætte koden bagefter filnavnet. F.eks. kunne man angive, at filen **mit program** er en LISTet fil ved at give filen navnet:

**mit program.lst** eller **list.mit program**

En tekstfil fra et tekstbehandlingsprogram skelnes fra andre filer, hvis man tilføjer en **text** angivelse:

**brev.txt** eller **text.brev**

Vi foreslår følgende filtyper anvendt:

<b>.lst</b>	<b>lst.</b>	LIST fil
<b>.dsp</b>	<b>dsp.</b>	DISPLAY fil
<b>.obj</b>	<b>obj.</b>	Objektcode fil
<b>.ext</b>	<b>ext.</b>	Ekstern procedure
<b>.bas</b>	<b>bas.</b>	Basic program
<b>.txt</b>	<b>txt.</b>	Tekstfil
<b>.gr0</b>	<b>gr0.</b>	Tegning i højopløsningsgrafik
<b>.gr1</b>	<b>gr1.</b>	Tegning i flerfarvegrafik
<b>.sp0</b>	<b>sp0.</b>	Spritetegning i højgrafik
<b>.sp1</b>	<b>sp1.</b>	Flerfarve spritetegning
<b>.src</b>	<b>src.</b>	Kildetekst (source code)
<b>.prc</b>	<b>prc.</b>	Procedure
<b>.fnc</b>	<b>fnc.</b>	Funktioner
<b>.fnt</b>	<b>fnt.</b>	Tegnsæt (eng. font)

Valget er selvfølgelig dit, men det letter programudvekslingen mellem COMAL brugere, hvis man benytter den samme notation.

En af fordelene ved filtypeangivelser er, at alle filer af samme type kan udskrives i kataloget, hvis man f.eks. skriver:

**dir "lst.\*"**

Hvis filtypebetegnelserne er tilføjet foran filnavnet, udskrives alle LIST'ede filer.

Benyttes konventionen med efterstillet filtypeangivelse, vil alle sprite-filer med et beskrivende navn på 5 tegn udskrives, hvis følgende kommando benyttes:

**dir "?????.sp?"**

## FILER OG SKÆRM, TASTATUR OG DISKETTESTATION

Een af COMAL sprogets stærkeste egenskaber, når det drejer sig om filer, er dets evne til at kommunikere med Commodore 64'ens indlæsnings- og udskriftsmedier. Hidtil har vi kun omtalt kommunikation med diskettestation og Datasette, men skærm billede og tastatur m.m. betragtes også af COMAL som filer. Enhedsangivelsen skal blot medtages i filnavnet.

Mulige enhedsangivelser:

<b>kb:</b>	Tastatur (eng. keyboard)
<b>ds:</b>	Skærm billede (eng. display screen)
<b>lp:</b>	Linieskriver (eng. line printer)
<b>sp:</b>	Seriell port
<b>cs:</b>	Datasettebånd (eng. cassette)
<b>u&lt;tilbehør&gt;:</b>	Enhedsnummer for tilbehør <tilbehør> er et nummer: 0 - 31
<b>&lt;disknr&gt;:</b>	Diskstationens nummer (0 ved opstart) <disknr> er et nummer: 0 - 15

Læg mærke til, at <tilbehør> skal være et tal i intervallet fra 0-31, og <disknr> er et tal fra 0-15. F.eks. svarer **select output "u4:"** til, at man vælge enhed nr. 4 på Commodore-IEEE seriellbussen (altså i dette tilfælde den førstprioriterede printer).

Ordren **unit "ds:"** dirigerer COMAL til at betragte display'et som output-enheden.

Det kan også lade sig gøre at afsløre den aktuelle enhedsangivelse ved hjælp af den reserverede strengvariabel **unit\$**. For eksempel:

```
print unit$
if unit$<>"lp:" then
unit "cs:"
```

En speciel egenskab ved håndtering af filer er tegnet @, som kan være det første tegn i et filnavn. Derved overskrives filen med samme navn, hvis den eksisterer i forvejen på disketten. Hermed slipper man for først at slette en gammel version, før en ny lagres. På den anden side må tegnet bruges med forsigtighed for at hindre uønsket sletning af filer. Brug diskstationsnummer i filnavn, når du bruger @.

## DATASSETTEN OG FILER

Skønt seriøs filhåndtering kræver, at man benytter en diskettestation, vil datasettebrugere alligevel kunne arbejde med SEKVENTIELLE filer efter samme fremgangsmåde, som beskrevet for disketter. Blot vil hastigheden være betydelig mindre. Dog kan READ FILE og WRITE FILE ikke benyttes.

## BRUG AF 1520 PRINTER-PLOTTER

En 1520 Printer-Plotter er tilbehør, som kan tilsluttes Commodore 64. Den kan benyttes både til udskrivning af programmer og til at tegne grafiske billeder af høj kvalitet med op til fire farver.

Det er nemt at aktivere Printer-Plotteren fra COMAL. Den forbindes til computerens serielle udgang (eller til den ekstra serielle ud-

gang på 1541 diskettstationens bagside). Prøv den følgende demonstration. Kontroller, at plotteren er tændt. Indtast et kort program, og skriv så:

**list "u6:"**

Dit program skulle nu udskrives på plotteren.

Andre ordrer til Printer-Plotteren skrives på tilsvarende måde. Husk blot på enhedsangivelsen **u6:**.

På demonstrationsdisketten og i appendiks H findes et program, som illustrerer brugen af plotteren fra COMAL: **1520 plotter demo**. Afprøv programmet og studer udskriften.

## RESUME

I dette kapitel er gennemgået adskillige vigtige emner angående brugen af COMAL filer:

- \* fil operationer på programmer og procedurer
- \* tal og tekster på sekventielle filer
- \* filer med direkte tilgang
- \* filtyper
- \* indlæsning- og udskrift med filer
- \* 1520 Printer-Plotter som fil

Nedenstående begreber skulle gerne være velkendte efter gennemarbejdning af kapitlet:

*filer*  
*lager enhed*  
*sekventiel fil*  
*fil med direkte tilgang*  
*post og felt*  
*dataelement*  
*boblesortering*  
*søgning*  
*filtyper*  
*specifikationer for tilbehør*

Nedenstående COMAL nøgleord er diskuteret:

**SAVE - LOAD**  
**LIST - ENTER - MERGE**  
**OPEN FILE - CLOSE FILE**  
**PRINT FILE - INPUT FILE**  
**WRITE FILE - READ FILE**  
**RANDOM**  
**CREATE**  
**GET\$**

I tilgift til eksemplerne i dette kapitel beskrives instruktionerne i større detalje i kapitel 4.

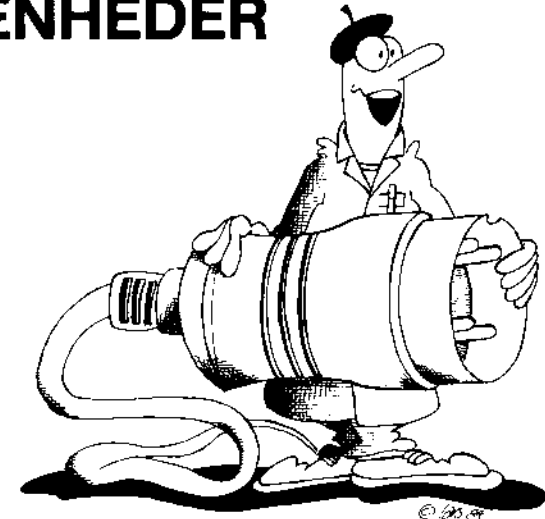
De følgende programmer er gennemgået og findes på demonstrationsdisketten (båndet):

**adr liste demo**  
**random fil demo**  
**filflyt**  
**plotter demo**

Den bedste måde at lære om filer på er ved at bruge dem til et konkret eksempel. Programmerne her kan være et udgangspunkt. Udvik dem og ændr i dem. Så finder du ud af, at det at kunne håndtere filer er en værdifuld færdighed med mange anvendelsesmuligheder.

## Kapitel 7 -

# YDRE ENHEDER



### INDLEDNING

Commodore 64 computeren er udstyret med adskillige muligheder for sammenkobling med ydre enheder. Sammenlignet med andre computere i samme klasse er Commodore 64 rigeligt forsynet med ind-og udlæsningsforbindelser, som er inkluderet i maskinens standard udrustning:

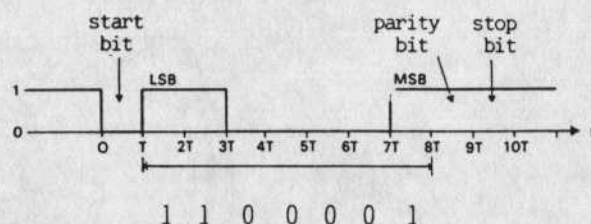
- \* IEEE seriel bus, der forbinder C64 med diskstation, printere eller andre enheder
- \* Forbindelsesled til Datasette båndenhed
- \* Parallel port til indlæsning og udskrift
- \* Kapselport, der forbinder computeren med spil, brugerprogrammer, eller sprogkapsler som f.eks. COMAL
- \* To kontrolporte til joysticks mm.

Som det ses af kapitel 5 om COMAL pakker, er det let at integrere brugen af joysticks, paddles eller en lyspen i programmerne. IEEE seriel bussen til kommunikation med diskstationer eller printere omtales i kapitel 6 om COMAL filer. Datasette ejere er også bekendte med, hvordan man gemmer og henter programmer og filer.

I dette kapitel vil vi rette opmærksomheden mod brugen af RS-232C forbindelsesled, IEEE kapsler og især parallelporten.

### RS-232C FORBINDELSSESLED

RS-232C er en industristandard, som definerer en bestemt type seriel kommunikation. Informationer transmitteres som en serie pulser langs en enkelt ledning. Figur 7.1 illustrerer det transmissionsmønster, som svarer til den serielle (standard) ASCII kode for bogstavet C. Dette bogstav har ASCII decimalkoden **67**, svarende til det binære tal **1000011**.



Figur 7.1: Bogstavet **C** transmitteres i seriel form i henhold til RS-232C standarden. Bemærk, at der kun sendes 7 bit, og at de sendes således, at mindst betydende bit kommer gennem ledningen først.

Informationerne kan sendes i *asynkron* form. Den tid, det tager at transmittere et helt tegn, deles i ti lige store tidsintervaller. To veldefinerede spændingsniveauer er bestemmende for, om signalet i et givet interval skal tolkes som *højt* eller *lavt*. I det følgende vil vi henviser til logiske niveauer, men det bør huskes, at i praksis optræder disse niveauer som spændingsvariationer i RS-232C kablet.

Hvert tegnsignal begynder med en *startbit*. I eksemplet vist i figur 7.1 er der tale om det logiske 0. Startbitten bruges til at synkronisere modtageren med afsenderen. Når startbitten registreres, starter den et ur med perioden **T**, som så koordinerer aflæsningen af *signal-linien*. Modtageren kan foretage periodiske aflæsninger for at afgøre, om den enkelte bit er et logisk 1 eller et logisk 0. Efter syv aflæsninger er tegnets binære kode til rådighed i modtagerens lagerregister.

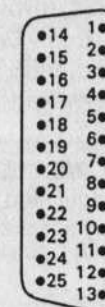
Den næste bit er *paritetsbitten*, som fortæller modtageren, om der overføres et lige eller et ulige antal 1'ere (eller 0'ler) i en given tegnkode. For systemer med *lige paritet* vil paritetsbitten være høj (logisk 1), hvis et *lige* antal *høje* bits (inklusive paritetsbitten) transmitteres, og paritetsbitten vil være lav (0), hvis et *ulige* antal sendes. Modtageren kan kontrollere dette for at få bekræftet, om der har fundet transmissionsfejl sted. Til slut sendes en *stopbit* for at tilkendegive afslutningen på tegnoverførslen.

RS-232C standarden specificerer også en *protokol*, som er udformet med henblik på at lette kommunikationen. Eksempelvis er der defineret signalerne CTS (eng.: Clear To Send; parat til at sende) og RTS (eng.: Request To Send; anmodning om at sende). End-

videre er spændingsniveauerne for logisk 1 og logisk 0 specificeret som henholdsvis -12 og +12 volt. Den fuldstændige specifikation findes i elektronikbøger. Følgende oplysninger skulle være tilstrækkelige til at sætte dig i stand til at begynde at bruge RS-232C forbindelsesledet med COMAL.

De elektriske forbindelser til en RS-232C port er standardiserede ved brug af DB-25 konnektoren:

ben	signal	kode
1	beskyttelsesjord	GND
2	signal ud (data)	SOUT
3	signal ind (data)	SIN
4	anmodning om at sende	RTS
5	parat til at sende	CTS
6	datasæt parat	DSR
7	signaljord	GND
8	signalspor	DCD
9-17	... ikke brugt ...	
18	klokke	RI
19	... ikke brugt ...	
20	dataterminal parat	DTR
21-25	... ikke brugt ...	



Figur 7.2: De standardiserede benforbindelser for RS-232C forbindelsesledet og benarrangementet for DB-25 konnektoren.

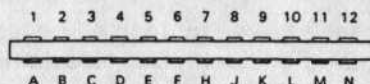
Alle til rådighed stående RS-232C kontrolsignaler bruges sjældent i faktiske kommunikationsopstillinger. Det er ofte tilstrækkeligt kun at bruge de to datakanaler **SIN** og **SOUT**. Et forbindelsesled af denne type kaldes somme tider et *tretråds-forbindelsesled*, da det kun består af en indgang, udgang og signal-jord.

Commodore 64 computeren kan håndtere tretråds-forbindelsesledet såvel som det komplette RS-232C forbindelsesled med alle kontrolsignaler. Dog afviger Commodore forbindelsesledet fra RS-232C standarden med hensyn til spændingsniveauer. Commodore 64 bruger 0 volt for logisk 1 og +5 volt for logisk 0. RS-232C er til rådighed på Commodore brugerporten som antydnet i figur 7.3:

Commodore brugerport	RS-232C signal beskrivelse	Signal retning	DB-25 standard forbindelser
A	GND		1
B	SIN	ind	3
C	SIN	ind	3
D	RTC	ud	4
E	DTR	ud	20
F	RI	ind	18
G	DCD	ind	8
H	CTS	ind	5
K	DSR	ind	6
L	SOUT	ud	2
M	SOUT	ud	2
N	GND		7

(NB: B og C bør forbindes med hinanden.)





Commodore 64 brugerportens benforbindelser.

Figur 7.3: Commodore RS-232C forbindelserne er tilgængelige på brugerporten på computerens bagside.

Det er meget vigtigt, at spændingsniveauerne for Commodore 64 RS-232C forbindelsesleddet er tilpasset de +/- 12 volt, der findes på andet udstyr. En standard omsætter kan købes hos Commodore forhandleren. I hobbylitteraturen findes diagrammer over udstyr, så man selv kan bygge et forbindelsesled.

**ADVARSEL:** Fejlagtig sammenkobling af RS-232C forbindelsesleddet med andet udstyr, der bruger +/- 12 volt, kan forårsage permanent skade på Commodore 64 computeren.

I COMAL kan man udvælge et antal parametre, som tilpasser kommunikationsudstyret til din Commodore 64. Det næste COMAL program viser, hvordan man kan *modtage* data ved hjælp af RS-232C forbindelsesleddet:

```
0010 OPEN FILE 1,"sp:300d8s1pe",READ
0020
0030 REPEAT
0040   a$:=GET$(1,1)
0045   PRINT a$,
0050 UNTIL a$=CHRS(255) OR KEY$<>CHRS(0)
0060
0070 CLOSE FILE 1
0080
0090 END "End"
```

Linie 10 åbner en logisk fil med strømnummeret **1** og præciserer følgende oplysninger: filen skal læse serielporten med en transmissionshastighed på 1200 baud (**b1200**), 8 data bits (**d8**), 1 stop bit (**s1**) og lige paritet (**pe**). I almindelighed anvendes følgende kodning for at specificere RS-232C forbindelsesleddet:

parameter	syntaks	værdier	opstartsværdi
baudrate	b<baud>	50-2400	b300
databits	d<num>	5-8	d7
stopbits	s<num>	0-2	s2
paritet	p<type>	n=ingen e=lige o=ulige	pn

### Eksempler:

"sp:" 300 baud, 7 databits, 2 stopbits, ingen paritetsbit  
 "sp:b600" 600 baud, 7 databits, 2 stopbits, ingen paritetsbit  
 "sp:b1200d8s1pe" 1200 baud, 8 databits, 1 stopbit, lige paritet

Bemærk, at den serielle datastrøm forbliver åben, og programmet fortsætter med at køre i REPEAT-UNTIL løkken i linie 30-50, indtil en tegnkode svarer til 255, eller indtil der trykkes på en vilkårlig taste.

Når data skal afsendes fra computeren via RS-232C udgangen, må filen åbnes tilsvarende, dog skal man bruge WRITE i stedet for READ. Bemærk også, at en åben RS-232C fil naturligvis skal lukkes (CLOSE) igen snarest muligt. Det er ikke muligt at anvende Datasette enheden eller IEEE serielbussen (altså diskettstationen) mens RS-232C forbindelsesleddet er i drift. Altså må data, som modtages, lagres i arbejdslageret eftersom de indlæber via RS-232C porten og senere gemmes på diskette. Tilsvarende skal du forberede en fil i arbejdslageret, OPEN RS-232C filen, sende dine data, og så lukke filen med CLOSE, inden du bruger diskettstationen.

### IEEE KAPSLER

Det er muligt at købe et udvalg af IEEE kapsler, som kan tilsluttes Commodore 64'ens modulstik. Disse kapsler fås hos din Commodore forhandler (spørg efter IEEE 488 kapslen) eller fra andre leverandører. F.eks. kan man få *Bus Card II* fra firmaet *Batteries Included* i Canada. Disse kapsler kan anvendes samtidig med din COMAL kapsel, da de er forsynede med et tilgængeligt kapselstik. IEEE kapslen tilsluttes modulstikket på din Commodore. Derpå kan COMAL kapslen monteres i stikket på IEEE modulet.

Hvis man har en Commodore IEEE interface, kan pakken **system** anvendes som hjælpemiddel. Se kapitel 5 under beskrivelsen af denne pakke for informationer om ordren **serial(<værdi>)**. Størrelsen **<værdi>** kan være **sand** eller **falsk**. Når den er **sand** dirigeres kommunikation til serielporten (hvor bl.a. diskettstationen kan tilsluttes). Er **<værdi>** falsk, dirigeres kommunikation via IEEE-488 modulet, hvis det er til rådighed.

Den største fordel ved brug af IEEE kapslen er, at du kan anvende Commodoren til at kommunikere med andre typer diskettstationer med høj kapacitet og høj hastighed, som f.eks. Commodore CBM 8050 og 8250 enhederne.

Hvis du har andre kapsler som f.eks. spilmoduler, regnearkprogrammer, og lign., skal man først fjerne COMAL-kapslen, før de bruges. Husk endelig i dette tilfælde at *slukke for strømforsyningen til alle enheder*, inden du skifter kapsel!

### PARALLELSTIKKET

Parallelstikket har mange navne: *brugerstik, brugerport, parallel*

port, user port og I/O port (I/O står for Input/Output). Det bruges til at sætte computeren i forbindelse med omverdenen. Man kan benytte det som UDGANG til kontrolformål, f.eks. styre en maskine (robot), tænde og slukke lys, automatisere et tog m.m. Man kan også bruge stikket som INDGANG til at indsamle oplysninger, f.eks. måle elektriske spændinger, temperatur og andre størrelser.

I dette afsnit vil vi beskrive et eksempel på anvendelse for at vise, hvordan det gøres. Vi har på ingen måde forsøgt en fuldstændig gennemgang af parallelportens muligheder. Der er skrevet flere bøger på dansk om dette emne. Ellers vil **Commodore 64 Programmer's Reference Guide** være et godt sted at finde detaljerede oplysninger. Den kan købes hos nærmeste Commodore forhandler. I det følgende er kun medtaget det mest nødvendige for at sætte dig i stand til at forstå eksemplerne og samtidig give dig lyst til at eksperimentere videre på egen hånd.

Selve den fysiske placering af parallelstikket er kantstikket yderst til højre, når computeren ses bagfra. Se figur 1.1 i kapitel 1. Parallelstikkets elektriske forbindelser er vist på figur 7.3 tidligere i dette kapitel. Den bedste måde at forbinde ydre elektriske signaler til stikket er med et 24-bens forbindelsesled. Det kan købes hos nærmeste Commodore forhandler eller i de fleste elektronikforretninger. Vi vil i eksemplerne benytte følgende forbindelser:

Forbindelse	Signal
ben 1 (og A)	Jord (=0 V)
ben 2	+ 5 V jævnspænding (max. 100 mA)
ben C	B port, bit 0
ben D	B port, bit 1
ben E	B port, bit 2
ben F	B port, bit 3
ben H	B port, bit 4
ben J	B port, bit 5
ben K	B port, bit 6
ben L	B port, bit 7

Computeren forbindes f.eks. nemt til ydre udstyr med et stykke fladkabel med 10 tråde. Lod de 10 ledninger til forbindelsesledet som vist ovenfor. I kablets anden ende loddes f.eks. et standard DB-25 miniature 25-bens forbindelsesled på. DB-25 leddet er vist på figur 7.2. Disse forbindelser er billige og nemme at få fat i.

25-bens leddet kan anbefales, fordi man senere måske får lyst til at tilslutte flere forbindelser til mere avancerede projekter. Forbind ben 1 på DB-25 til jord (=0 Volt), ben 2 til +5 Volt og ben 18-25 til port B bit 0-7.

**ADVARSEL:** Det er ikke tilrådeligt at udføre projekterne uden nogen forudgående kendskab til arbejdet med elektriske forbindelser. Forbind aldrig noget til computerens brugerstik, med mindre alle strømforsyninger er slukkede og taget ud af stikkontakten. Ukorrekt forbindelse til Commodore 64 kan ødelægge computeren. Skønt projekterne ikke er indviklede, er det bedst at rådføre sig med en elektronikkyndig, hvis man er usikker på fremgangsmåden.

For at vise hvordan man styrer en ydre enhed via parallelstikket, har vi valgt et bestemt, enkelt eksempel. Når du har sat dig ind i dette eksempel, får du sikkert lyst til at gå i gang med mere ambitiøse opgaver.

Vi forestiller os et lukket kredsløb af togs Skinner, et elektrisk tog og en station. Vi ønsker, at computeren skal tillade toget at køre frit, indtil det nærmer sig stationen. Det skal stoppe ved stationen, vente i en forud bestemt tid, og så fortsætte med et nyt gennemløb af skinesystemet.

For at være i stand til at styre denne proces må vi have følgende udstyr:

- \* Til at tænde og slukke for strømmen til skinnerne bruges en transistor og et relæ. Dem skaffer man sig nemmest i den lokale hobby-elektronik forretning.
- \* En føler skal registrere, når toget passerer lige før stationen. Til det kan man benytte en lille fototransistor og en smal lysstråle. Lysstrålen skal lyse tværs over skinnerne og ramme den følsomme del på fototransistoren. Transistorens kollektor forbindes til portens bit som beskrevet nedenfor, og emitteren jordforbindes.

Til at styre toget skal vi bruge to af benene på parallelstikket, svarende til to bit i port B. Vi kan vælge frit. Vi tager bit 0 til lysføleren og bit 1 til at starte og standse toget.

Hver bit i port B kan bruges både som indgang og som udgang. Retningen bestemmes ved at anbringe et passende tal i "data retningsregistret" for port B. Retningsregistret og port B har lageradresserne:

	decimal	hexadecimal
port B adresse	56577	\$dd01
port B retningsreg.	56579	\$dd03

Det tal, der gemmes i dataretningsregistret (ofte betegnet **ddr** fra engelsk *Data Direction Register*), bestemmer, om den enkelte bit i



port B skal være indgang eller udgang. Det er nemmest at forstå tallene, hvis vi benytter binære tal, dvs. 1'ere og 0'ere. Et 0-bit i ddr'et betyder, at den tilsvarende bit i port B skal være indgang. Et 1-bit i ddr'et betyder, at den tilsvarende bit i port B skal være udgang. F.eks., **%00000010** (decimalt = 2) anbragt i ddr'et gør port B bit 0 til indgang og bit 1 til udgang. Det er alt, hvad der kræves for at styre toget.

Da man i COMAL direkte kan skrive binære tal, er det unødvendigt for programmøren ved store tal at omregne det binære tal til ti-tal systemet. Man skal blot huske at anbringe et %-tegn foran tallet.

Programmet **tog demo** findes på COMAL demonstrationsdisketten og som udskrift i appendiks H.

Programmet starter med at udskrive følgende besked på skærmen:

### ELEKTRISK TOG DEMO

Toget skal starte ved stationen  
med passagelyset lige bag den  
sidste vogn. Start toget, og tryk  
derefter en taste ned for at  
overdrage kontrollen til computeren..

Den næste linie har udseendet:

```
0100 WHILE KEYS=CHRS(0) DO NULL
```

Denne linie bør være velkendt. Den opholder programmet, indtil en taste trykkes ned.

Hovedprogrammet starter i linie 200. Proceduren **definer'variabler** definerer adresserne på portb og dens dataretningsregister ddr, og variabelen **position** sættes til sin startværdi 1. Bemærk, at COMAL's hexadecimal angivelse af adresserne, kendetegnes ved \$-tegnet foran tallet. Læg også mærke til de beskrivende variabelnavne:

```
0690 PROC definer'variabler
0700   portb:=$dd01
0710   portb'ddr:=$dd03
0720   position:=1
0730 ENDPROC definer'variabler
```

Apostrofferne er nødvendige som bindeled mellem de enkelte ord. COMAL systemet tillader ikke todelte variabel- og procedurenavne. Variabelen **position** bruges til at styre pegepinden på skærm billedets "køreplan".

I hovedprogrammets næste linie kaldes proceduren **angiv'portb**:

```
0750 PROC angiv'portb
0760   POKE portb'ddr,2
0770   POKE portb,2
0780 ENDPROC angiv'portb
```

I denne procedure anbringes tallet 2 (binært **%00000010**) i **portb'ddr**-adressen. Derved gøres bit 0 til indgang og bit 1 til udgang. Bit 2-7 benyttes ikke i dette eksempel, men stilles alle til 0.

Toget startes i proceduren **start'tog**:

```
0480 PROC start'tog
0490   POKE portb,PEEK(portb) BITOR 2
0500   fremryk'pegepind
0510 ENDPROC start'tog
```

POKE sætningen anbringer tallet **PEEK(portb) BITOR 2** i portb-adressen. BITOR operatoren er beskrevet i detalje i kapitel 4. Denne ordre sikrer, at bit 1 er "høj". Signalet fra computeren forstærkes af transistoren og aktiverer relæet, som igen starter toget. Proceduren **fremryk'pegepind** bevæger pilen frem til det næste emne på skærbilledet eller tilbage til køreplanens begyndelse ved slutningen af hver omgang.

```
0800 PROC fremryk'pegepind
0810   PRINT AT 10=position,2: " "
0820   IF position<4 THEN
0830     position:=position+1
0840   ELSE
0850     position:=2
0860   ENDIF
0870   PRINT AT 10+position,2: ">"
0880 ENDPROC fremryk'pegepind
```

Den næste procedure er **udskrift**. I den udskrives 'køreplanen' på skærmen:

```
> toget er i gang
toget passerer lyset
toget venter ved stationen
Hvis en taste trykkes ned,
standser toget ved stationen....
```

Pilen viser, hvor langt toget (og programmet) er nået. Nu kommer vi til programmets hovedløkke:

```
0270 REPEAT
0280   check'lys
0290   pause(1.5)
0300   stop'tog
0310   pause(10)
```

```

0320 start'tog
0330 UNTIL KEYS<>CHRS(0)
0340 stop'tog
0350 PAGE
0360 PRINT "Det var den tur."

```

Programmet vil fortsætte i løkken, indtil en taste trykkes ned, hvorefter programmet standser toget med beskeden: **Det var den tur.**

Proceduren **check'lys** kontrollerer, om portb bit 1 er høj eller lav (1 eller 0):

```

0530 PROC check'lys
0540 WHILE PEEK(portb) BITAND 1<>1 DO NULL
0550 fremryk'pegepind
0560 ENDPROC check'lys

```

Operatoren BITAND er beskrevet i kapitel 4. I dette tilfælde vil det logiske udtryk **PEEK(portb) BITAND 1<>1** være **falsk**, hvis bit 1 er høj. Det sker, hvis lyset på fototransistoren afbrydes (toget kommer ind i lysstrålen). Modstanden i fototransistoren er lille, når transistoren belyses. Da kollektoren er forbundet til port B bit 0, og emitteren er jordforbundet, vil den lave modstand (omkring 100 ohm) også gøre, at bit 0 jordforbindes (gøres lav). Når transistoren ikke belyses, er modstanden stor (typisk 1 Mohm), og bit 0 er igen høj.

Før toget standses, venter programmet i 1.5 sekund i proceduren **pause**.

```

0590 PROC pause(sec)
0600 TIME 0
0610 WHILE TIME<sec*60 DO NULL
0620 ENDPROC pause

```

Proceduren nulstiller først et internt ur (TIME 0), hvorefter den venter i næste linie, indtil der går **sec** sekunder (TIME regner tiden i 1/60 sekund).

Toget standses i proceduren **stop'tog**, som blot ændrer **portb** bit 1 adressen til 0. Det ville selvfølgelig være mere livagtigt, hvis toget standsede ved langsomt at sænke farten. Det kunne gøres ved at tænde og slukke bit 0 i hurtig rækkefølge, således at den tid, hvor bit'en er tændt, gradvis mindskes. Hvis du beslutter dig for at prøve, må relæet udskiftes med et krafttransistor kredsløb til styring af strømmen til skinnerne.

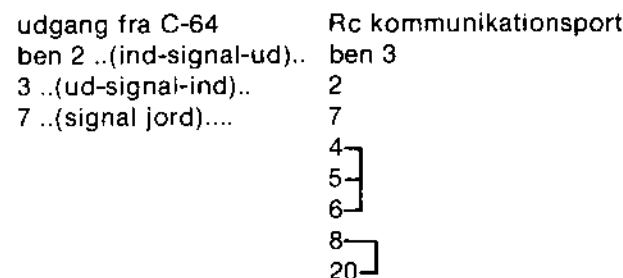
## FILOVERFØRSEL MELLEM COMPUTERE (C-64 og Piccoline)

Når man vil overføre filer fra den ene computer til den anden, er det naturligt at udnytte RS-232C grænsefladen til denne to-vejs kommunikationsopgave, fordi mange datamater understøtter denne standard. Bemærk her, at C-64 kræver brugen af et RS-232C mo-

dulstik, der ændrer Commodorens spændingsniveauer (0 og 5 volt) til RS-232C standardens plus/minus 12 volt. Dette modulstik koster mellem 300 og 700 kr, afhængig af fabrikat. I dette eksempel er der brugt et Handic RS-232 modul. Desuden er det nødvendigt, at computeren, som via kabel skal forbindes til C-64, er forsynet med en RS-232C grænseflade med både ind- og udgang.

## KABELFORBINDELSEN (tretrådsforbindelse):

Som nævnt tidligere i dette kapitel foreskriver RS-232C standarden, at DB-25 hanstik skal bruges i begge ender af forbindelseskablet. Dette kan virke lidt mærkeligt, fordi der kun er behov for 3 ledninger, og DB-25 er et 25 pol stik. Dette valg er nok alligevel klogt, dels fordi der skal kortsluttes nogle ben i Rc-enden af kablet, og dels fordi RS-232C forbindelserne til begge datamater anvender DB-25 standarden.



Figur 2 viser RS-232 modulet og forbindelseskablet.

## OVERFØRSEL AF PROGRAMFILER:

Selv om alle slags filer kan overføres via RS-232C forbindelsen, koncentrerer vi os først om én bestemt type, nemlig COMAL programfiler. Programmet der ønskes overført, læses ind i afsendercomputerens (*sender's*) arbejdslager. Programmet gennemgås, så alle programlinier, som man ikke er sikker på, at modtagercomputerens (*modtager's*) COMAL kan fortolke, gøres til kommentarsætninger ved at anbringe // først i linien efter linienumeret. Derved undgås, at modtagercomputeren melder om syntaksfejl under overførslen. Syntaksfejl kan enten afbryde overførslen eller forårsage, at programlinier går tabt. Ved at gøre tvivlsomme sætninger til kommentarer, sikres en fuldstændig overførelse. Senere kan man i ro og mag rette den modtagne programfil til ved hjælp af modtagerdatamats almindelige programredigeringsfaciliteter.

Bemærk i denne forbindelse, at UniComal's COMAL 80 på Com-

modore har redigeringsmuligheder, som på mange måder er langt bedre end dem, der findes på andre COMAL-computere. Specielt kan Commodore's CHANGE ordre være meget nyttig. Foretag derfor de flest mulige tillempninger af programmet på din 64'er.

Før overførslen må RS-232C transmissionsbetingelserne defineres. På RC Partner gøres det med et *KONFIG*-program, som "konfigurerer" kommunikationsporten: altså giver brugeren mulighed for at vælge de parametre, der skal bruges ved kommunikationen. På C-64 gøres det ved overførsels start. Ved denne prøveopstilling har vi benyttet:

transmissionshastighed	1200 baud	(b1200)
antal databits	7	(d7)
antal stop bits	1	(s1)
paritet	lige	(pe) - e for "even"

Bemærk, at denne opstilling normalt bør være det førstprioriterede valg af RS-232 parametre på Partneren. Det vil nok alligevel være klogt at bruge Partnerens *KONFIG* program for at kontrollere, at kommunikationsporten er indstillet korrekt.

### SEND PROGRAMMER FRA C64 TIL PARTNER:

Anbring COMAL programmet i C-64'ens arbejdslager, og ret det til.

- 1) Skriv: **new** på Partner for at slette et evt. eksisterende program fra lageret.
- 2) Tast: **enter "/1/com"** på Partner, og tryk på **<RETURN>**-tasten. Partneren afventer nu, at data indløber fra kommunikationsporten.
- 3) Indtast følgende på C-64:

**select output "sp:b1200d7s1pe".**

Tast derpå **llst**. Ved tryk på **<RETURN>**-tasten startes udskrivning af data til C-64'ens serielle port med den transmissionsform, som valgtes ved hjælp af **select output** sætningen.

- 4) Programmet føres nu over fra C-64 til Partner. Måske kan der forekomme enkelte syntaksfejl, som må erindres til senere tilpasning af programmet.
- 5) Når markøren blinker på C-64, er afsendelsen slut, og Partneren afventer nu et slut-signal (EOF-signal).
- 6) Skriv igen på C-64 (man kan bare flytte markøren og bruge samme linie igen): **select output "sp:b1200d7s1pe"**. Tast så

**print chr\$(26)** - dette tegn er slutsignalet til Partneren. Partnerens markør vil begynde at blinke som tegn på, at overførslen er slut.

- 7) Skriv til slut: **select output "ds:"** på C-64 for igen at vende tilbage til dataskærmen som output-enhed.
- 8) Det overførte program kan nu rettes til på Partneren.

### SEND PROGRAMMER FRA PARTNER TIL C-64:

COMAL programmet, der skal overføres, bringes på plads i Partnerens arbejdslager, og evt. kommentarsætninger indføjes for at forhindre meddelelser om syntaksfejl.

- 1) Skriv på C-64: **select input "sp:b1200d7s1pe"**.
- 2) Skriv på Partner: **llst "/1/com"**.
- 3) Når Partner-markøren blinker, er overførslen slut. Tryk **<STOP-RESTORE>** på C-64 for at afbryde forbindelsen til den serielle port. Programmet ligger nu klar i C-64'ens arbejdslager, bortset fra evt. rettelser eller ændringer.

### OVERFØRSEL AF SEKVENTIELLE ASCII FILER

Det kan også lade sig gøre at overføre vilkårlige sekventielle ASCII filer fra den ene datamaskine til den anden. Der kan være tale om filer med tekst, tal, eller andet, som man ønsker adgang til på den anden datamat. Til dette formål skal hver maskine være i stand til at køre 2 programmer: et *senderprogram* og et *modtagerprogram*. For at overføre sekventielle filer fra en *sender* til en *modtager*, skal man i almindelighed følge nedenstående fremgangsmåde.

- 1) Sørg for, at forbindelsesledningen er tilsluttet korrekt med RS-232 modulet på plads bag på din Commodore.
- 2) Kontroller, at RS-232 kommunikationskortet på den anden computer er indstillet på den rigtige baudhastighed (f.eks. 1200), antal data-bits (7), antal stop-bits (1) og paritet (even). Brug evt. lavere baudhastigheder, hvis overførslen ikke lykkes, eller der forekommer fejl under transmissionen. Baudhastigheden og de øvrige parametre skal være ens for begge datamater.
- 3) Sørg for, at den sekventielle fil, som du ønsker at overføre, er på plads på disketten i *sender*'en. Load *senderprogrammet*, så det er klar til at køre.

- 4) Indlæs *modtagerprogrammet* i *modtager*-datamaten. Kør dette program, så datamaten venter på data.
- 5) Kør nu *senderprogrammet*. Svar på spørgsmålet om filnavnet.
- 6) Datatransmissionen bør nu gå i gang. Bemærk, at 1200 baud svarer til ca. 150 tegn i sekundet, svarende til et par skærm linier hvert sekund. Overførsel af hver kilobyte tager således omkring 6-7 sekunder.
- 7) Transmissionen ophører, når *sender*'en transmitterer et *filafslutningstegn*. CHR\$(26) er End Of File tegnet, når man sender fra C-64 til Partneren; og CHR\$(127) er EOF, når man sender den anden vej. *Modtagerprogrammet* sørger for, at den modtagne fil bliver lagret på diskette/disk hos *modtager*'en.

## COMMODORE PROGRAMMER:

### C-64 MODTAGERPROGRAM

```

0010 // save "@0:modtag'C64"
0020 DIM a$ OF 1, b$ OF 25000
0030 modtag'c64
0040 vis'instruktion
0050
0060 PROC modtag'c64
0070  PAGE
0080  PRINT "-----VENTER PAA DATA ----"
0090  OPEN FILE 2,"sp:b1200d7s1pe",READ
0100  WHILE a$<>CHR$(127) DO
0110    a$=GET$(2,1)
0120    b$+a$
0130  ENDWHILE
0140  OPEN FILE 3,"@0:flyttefil",WRITE
0150  PRINT FILE 3: b$(1:LEN(b$)-1)
0160  CLOSE
0170 ENDPROC modtag'c64
0180
0190 PROC vis'instruktion
0200  PAGE
0210  PRINT
0220  PRINT "Den overførte fil er nu klar under"
0230  PRINT "navnet 'flyttefil' paa disketten."
0240  PRINT
0250  PRINT
0260  PRINT
0270  PRINT
0280  PRINT "      enter ""flyttefil""
0290  PRINT
0300  PRINT AT 4,0: "      "
0310 ENDPROC vis'instruktion
0320

```

Bemærk, at for større filer kan der opstå problemer, når strengen b\$ udskrives til disketten. Dette problem kan simplest løses ved at udskifte linie 150 med:

```
150 FOR i:$1 TO LEN(b$)-1 DO PRINT FILE 3: b$(i:i),
```

### C-64 SENDERPROGRAM:

```

0010 // Commodore 64: send til seriel port
0020 fra'64
0030
0040 PROC fra'64
0050  DIM a$ OF 20
0060  PAGE
0070  PRINT "Den nævnte fil føres direkte"
0080  PRINT "til den serielle port fra diskette"
0090  INPUT "Indtast filnavn: ": a$
0100
0110  OPEN FILE 2,"sp:b1200d7s1pe",WRITE
0120  OPEN FILE 3,a$,READ
0130  WHILE NOT EOF(3) DO
0140    a$=GET$(3,1)
0150    PRINT FILE 2: a$,
0160  ENDWHILE
0170  PRINT FILE 2: CHR$(26) // Rc EOF
0180  CLOSE
0190 ENDPROC fra'64

```

## REGNECENTRAL PARTNER PROGRAMMER

### RC-MODTAGERPROGRAM:

```

0010 // Rc Partner: hent fra seriel port
0020 iiirc
0030
0040 PROC iiirc
0050  DIM a$ OF 20
0060  PRINT CHR$(12)
0070  PRINT "Husk, COM-porten skal konfigureres først"
0080  PRINT "Den nævnte fil føres direkte fra den serielle"
0080  PRINT "port til diskette."
0090  INPUT "Indtast filnavn: ": a$
0100
0110  OPEN FILE 2,"/1/com", READ
0120  OPEN FILE 3,a$, WRITE
0130  WHILE NOT EOF(2) DO
0140    a$=GET$(2,1)
0150    PRINT FILE 3: a$,
0160  ENDWHILE
0170  CLOSE
0180  PRINT "Overførsel slut"
0190 ENDPROC iiirc

```

**RC-SENDERPROGRAM:**

```

0010 // Rc Partner: send til seriel port
0020 fraRC
0030
0040 PROC fraRC
0050 DIM a$ OF 20
0060 PRINT CHR$(12)
0070 PRINT "Husk, COM-porten skal konfigureres først"
0080 PRINT "Den nævnte fil føres direkte fra den serielle
port til diskette."
0090 INPUT "Indtast filnavn: ": a$
0100
0110 OPEN FILE 2,"/1/com", WRITE
0120 OPEN FILE 3,a$, READ
0130 WHILE NOT EOF(3) DO
0140   a$=GET$(3,1)
0150   PRINT FILE 2: a$,
0160 ENDWHILE
0170 PRINT FILE 2: CHR$(127) // C 64 Sluttegn
0180 CLOSE
0190 ENDPROC fraRC

```

Der er naturligvis mange andre muligheder, end dem, vi har skitseret her. F.eks. er de C-64 programmer, der vises i dette afsnit, blevet overført via brugerporten til et parallel kort i en Apple II til brug som en tekstfil. Desuden kan C-64 COMAL programmer føres frem og tilbage mellem C-64 og IBM PC'en, der også har en UniComal COMAL version. Commodore COMAL brugere får hermed endnu flere muligheder for at udveksle programmer og filer med andre datamater.

**KONTROLPORTENE**

Foruden de mange muligheder for kommunikation med ydre enheder, som vi har beskrevet i dette kapitel, har Commodore 64 computeren også to kontrolporte. Brugen af disse porte fra COMAL er allerede beskrevet i kapitel 5 i afsnittet om COMAL pakker.

Ud over de 2 x 5 kontaktindgange (JOYA0-3, JOYB0-3, KNAP A og KNAP B), som er til rådighed ved de to kontrolporte, er der også i alt 4 forskellige analoge indgange til rådighed via kontrolportene. Disse indgange er POTAX, POTAY, POTBX og POTBY. (Internt på SID'en er der faktisk kun 2 ADC'ere og en analog switch.) Ben og forbindelser er som følger:

pin	kontrolport A	kontrolport B
1	JOYA0	JOYB0
2	JOYA1	JOYB1
3	JOYA2	JOYB2
4	JOYA3	JOYB3
5	POTAY	POTBY

6	KNAP A	KNAP B
7	+ 5V	+ 5V
8	JORD	JORD
9	POTAX	POTBX

Maksimal belastning på + 5V er 50 mA.

Læg mærke til, at der skal bruges en standard DB-9 **hun** konektor for at forbinde eksperimenter til kontrolportene.

Kontaktindgangene giver besked til et program, om en given kontakt er **tændt** eller **slukket**. I kapitel 5 er der eksempler på brugen af disse signaler.

De analoge indgange går til *A/D omvendere*, som bruges til at omsætte potentiometrenes positioner til digitale værdier, der kan fortolkes af computeren. Konverteringsprocessen baseres på tidskonstanten for en modstand og kapacitor. Kapacitoren skal tilsluttes mellem POT forbindelsen til jord. Den oplades gennem en resistor (f.eks. et potentiometer), der er forbundet fra POT forbindelsen til +5 volt. Komponentværdierne kan anslås ved hjælp af ligningen:  $RC = 4.7E-4$ . I denne ligning er **R** potentiometrets maksimale modstand, og **C** er kapacitansen. Jo større kapacitator, jo lavere bliver usikkerheden i POT værdien. De anbefalede værdier for **R** og **C** er 470 kiloohm og 1000 pF. Læg mærke til, at der skal bruges separate potentiometre og kapacitorer for hvert POT ben.

Skønt POT indgangene i kontrolportene blev udformet til at måle et potentiometers indstilling, kan man anvende andre variable modstande. Hvis man f.eks. vil måle temperatur, udskifter man blot potentiometret med en termistor i det korrekte modstandsinterval. Andet udstyr til måling af modstand kan naturligvis bruges til at foretage automatisk måling af tryk, væskehøjde, lysstyrke eller andre fysiske størrelser.

Følgende program kan danne grundlaget for konstruktion af et simpelt *digitaltermometer*:

```

0010 // save "@0:temperatur"
0020 USE paddles
0030 // kondensatoren: 1000 pF
0040 // termistor: ca.100 k ved 20 grader
0050 a:=1; b:=0 // kalibreringsværdier (1 og 0 for test)
0060 PAGE
0070 PRINT "DIGITAL TERMOMETER"
0080 PRINT AT 5,1: "Termistor og kapacitor skal tilsluttes"
0090 PRINT "kontrolport 1..."
0100
0110 // hovedprogrammet
0120 REPEAT
0130   check'paddle(1)
0140   konverter(middel)
0150   tryk'temperatur

```

```

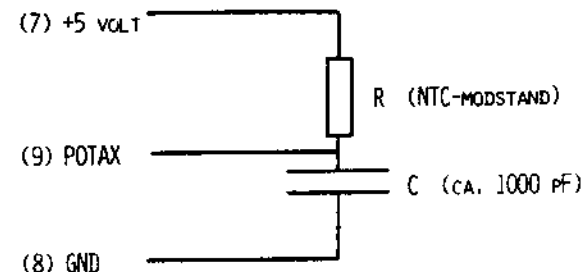
0160 UNTIL KEYS<>CHRS(0)
0170 END // hovedprogram
0180
0190
0200 PROC check'paddle(port)
0210   total:=0
0220   FOR I:=1 TO 50 DO
0230     paddle(port,a'paddle,b'paddle,a'knap,b'knap)
0240     total:=total+a'paddle
0250   ENDFOR I
0260   middel:=total/50
0270 ENDPROC check'paddle
0280
0290 PROC konverter(middel)
0300   temp:=a*middel+b
0310 ENDPROC konverter
0320
0330 PROC tryk'temperatur
0340   temp:=INT(temp*10)/10
0350   PRINT AT 10,10: "T =    o"
0360   PRINT AT 10,14: temp
0370 ENDPROC tryk'temperatur
0380

```

Første del af programmet (linie 10-100) består blot af indledende informationer, et budskab til skærmen og definition af konstanterne **a** og **b**. Læg mærke til, at disse er sat lig med hhv. 1 og 0 i linie 50. Dette betyder, at programmet nu blot vil trykke en paddelværdi uden omregning til temperatur. Disse værdier kan du først bestemme, når du har lavet en prøveopstilling og kalibreret føleren, som du vil bruge.

Bemærk strukturen i resten af programmet. Hovedprogrammet er fra linie 110-170. Den består blot af en REPEAT-UNTIL konstruktion, som henter informationer om paddleport 1 (**check'paddle(1)**), omdanner den fundne værdi ved en konverteringsrutine (**konverter**) og trykker temperaturen (**tryk'temperatur**). Proceduren **check'paddle(1)** benytter **paddle** pakkens procedure til at hente paddelværdien. For at opnå en stabil aflæsning, finder proceduren middelværdien af 50 målinger. Proceduren **konverter** omtaler vi om lidt. Proceduren **tryk'temperatur** udskriver blot den fundne temperaturværdi på skærmen. Man standser programmet ved at trykke på en vilkårlig taste.

Man bør begynde med at lave en prøveopstilling. Start evt. med en kapacitor på ca. 1000 pF og brug en termistor (NTC eller PTC modstand) med en værdi på omkring 100 kohm ved stuetemperatur. Til slut din prøveopstilling til kontrolport 1, som vist i nedenstående figur:



Figur 7.4: Mange forskellige følertyper kan tilsluttes kontrolportene, hvis man vil udnytte de indbyggede ADC'ere til at lave målinger.

Hvis du nu kører programmet, vil de målte "paddelværdier" vises på skærmen. Lav en graf, der viser temperaturen i grader celsius som funktion af paddelværdien. Hvis grafen er omtrent *lineær* i temperaturområdet, der har din interesse, kan man beregne **a** og **b** som følger:

Bestem to talsæt bestående af paddelværdier og tilsvarende temperaturer (P1,T1) og (P2,T2). Disse bør svare til punkter, der ligger på din graf. Konstanten **a** kan findes ved hjælp af formlen:

$$a = (T2 - T1)/(P2 - P1).$$

Vi har fundet følgende værdier i en opstilling, der anvender en NTC modstand: (183,25) og (215,20). Altså programmet viste 183 som paddelværdi, når følertemperaturen var 25 grader; og 215, når temperaturen var 20 grader. Dermed bliver **a** lig -0.178. For at finde **b** kan vi bruge formlen: **temp** = **a**\***middel** + **b**, som bruges i proceduren **konverter**. Ved at indsætte **middel** = 183 og **temp** = 25, finder vi **b** = 57.6.

Vil man kalibrere en NTC-føler over et større temperaturområde, kan man anvende funktioner som f.eks. eksponentialfunktionen for at opnå en bedre tilpasning med kalibreringsgrafen end den lineære funktion, som vi har anvendt i denne illustration. I givet fald må linie 300 udskiftes.

## RESUME

I dette kapitel har vi diskuteret en række af de righoldige muligheder, Commodore 64 er udstyret med til sammenkobling med forskelligt tilbehør. Læseren opfordres til selv at eksperimentere med RS-232C forbindelsesledet, den parallelle port og kontrolportene for at lære mere om dem.

Er man forsigtig med at lave en omhyggelig kontrol, inden man laver nogle elektriske tilslutninger, kan man roligt eksperimentere med computerens muligheder.

Der findes flere oplysninger om disse porte i den engelsksprogede håndbog **Commodore 64 Programmer's Reference Guide**. En lang række yderligere oplysninger er tilgængelige i den populære litteratur om mikrocomputere.

## Kapitel 8 -

# COMAL OG MASKINSPROG

### HVAD ER MASKINSPROG?

Hjernen i enhver mikrocomputer er en central mikroprocessor. Commodore 64 er ingen undtagelse. Enhver mikroprocessor forstår sit eget sæt af instruktioner. Disse instruktioner kaldes maskinsprogsinstruktioner. For at være mere præcis, maskinsprog er det ENESTE programmeringssprog, som Commodore 64 forstår. Det er så at sige dens modersmål.

I COMAL-kapslen er anbragt et meget stort maskinsprogsprogram, nemlig COMAL-systemet. Når maskinen tændes, ved den, hvad der skal ske, fordi COMAL-systemet da automatisk "køres". Det er COMAL, som sammen med et andet program, operativsystemet, tillader, at der indtastes fra tastaturet, og at skærmeditoreren anvendes. Når et COMAL-program kører, udføres der et passende stykke maskinsprog, som oversætter den COMAL-sætning, der skal udføres.

Dette kapitel er *ikke* en lærebog i maskinsprogsprogrammering på Commodore 64, men er en beskrivelse af den specielle metode, der skal anvendes for at bringe maskinsprogsprocedurer til at arbejde sammen med COMAL. Kapitlet er sværere tilgængeligt end denne vejlednings øvrige kapitler. Det forudsættes, at man har et habilt kendskab til computerbegreber på forhånd. Først gives en oversigt over maskinens hukommelse. Dernæst følger en trinvis gennemgang af, hvordan man opbygger sit eget maskinkodeprogram og benytter det i et COMAL program.

Maskinsprog læres lettest, hvis man kan få en som allerede kan maskinsprog, til at lære sig det. Næstbedst er det at købe bøger om maskinsprogsprogrammering med 6502-processoren. Det vil under alle omstændigheder være en god idé at anskaffe **Commodore 64, Programmer's Reference Guide**. Der findes efterhånden også bøger på dansk om 6502 maskinkode programmering, f.eks. **Programmering i assembler** af Jens Grysbjerg (Teknisk Forlag, Kbh., 1983).

Man skaber nemmest maskinsprogsrutiner med et **6510/6502-assembler** program, som oversætter fra symbolsk maskinsprog til maskinsprog; samt en diskettestation. På demo-disketten er en tekstfil ved navn **C64SYMB**, som indeholder en definition af alle symboler, der er relevante ved assemblerprogrammering med Commodore 64 og COMAL. Denne tekstfil bør inkluderes i enhver assembler-kildetekst med COMAL-pakker.

Det er også muligt at lave et maskinkodeprogram på den direkte

måde med at anbringe maskinsprogskoderne i lageradresserne. Det gøres ved hjælp af POKE-sætninger. Maskinkodeprogrammet, som således er anbragt i et ledigt RAM-område, startes fra COMAL med **SYS <startadresse>**. Maskinkodens sidste instruktion skal være **rts**, som får programudførelsen til at vende tilbage til COMAL. På denne måde kan man imidlertid ikke lave program-pakker i maskinkode, som kan LINKes til COMAL-programmer. Vi vil i dette kapitel holde os til opbygningen af maskinkoderutiner, som kan LINKes til et COMAL program.

Brug af maskinsprogsrutiner er en integreret del af COMAL-systemet. Ved designet af faciliteter til anvendelse af maskinkodede rutiner, er følgende tre mål forsøgt opnået:

- \* Maskinkodede rutiner skal være enkle at anvende, også for personer uden kendskab til maskinsprog.
- \* Adgang til maskinsprogsrutiner skal ske ved brug af navne. Herved skjules detaljer, såsom adresser.
- \* Maskinsprogsrutiner skal kunne påvirkes af kommandoer som **NEW** og **RUN**. Herved opnår man, at pakker ser ud som om, de er en integreret del af COMAL-systemet.

Der er tre kommandoer/sætninger i COMAL, som anvendes ved definition, brug og sletning af maskinkodede rutiner. Det er

```
LINK <filnavn>      // Indlæs en modul-fil.
USE <pakke>          // Definér procedurer.
DISCARD              // Slet alle moduler.
```

Disse kommandoer (USE kan også benyttes som programsætning) vil blive forklaret nærmere. Maskinsprogsrutiner anvender procedure- og funktionskaldsmekanismen i COMAL, og tillader derved alle parametertyper.

## MODULER

LINK-kommandoen henter et maskinsprogs-modul (bibliotek, eng. *library*), som er lavet af assembleren (objekt-fil). Dette modul indeholder information, som specificerer, hvor maskinkoden skal placeres i maskinen, i.e. på en absolut adresse i et bestemt lagerområde (eng. *memory-map*) udover selve maskinkoden. COMAL kan styre op til 10 sådanne moduler på én gang. Der er altid defineret mindst to moduler, som indeholder følgende:

(Modul 1)	(Modul 2)	
english	graphics	sound
dansk	turtle	joysticks
system	sprites	paddles
	font	lightpen

Disse moduler skal ikke LINKes, idet de allerede findes i COMAL-kapslen. Moduler kan slettes igen ved hjælp af DISCARD kommandoen. Dog kan de to ovennævnte standard moduler *ikke* slettes. Da moduler ikke har navne, kan man ikke slette et bestemt modul, men vil altid slette alle moduler. Moduler kan gøres permanente (ROMede), hvorved de ikke kan DISCARDes. Ikke-permanente moduler opfattes som om, de er en del af programmet. Ved SAVE gemmes alle ikke-permanente moduler sammen med COMAL-programmet i den samme PRG-fil. Ved LOAD, RUN eller CHAIN læses de ind igen (LINKes).

## PAKKER

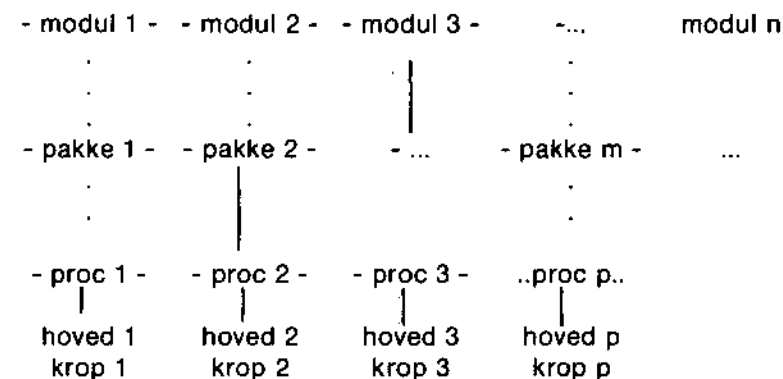
Et modul kan indeholde 0, 1 eller flere *pakker* (eng. *packages*).

## PROCEDURER OG FUNKTIONER

En pakke kan indeholde ingen, eller én eller flere procedurer eller funktioner. Til en procedure/funktionsdefinition hører to dele, nemlig:

- \* Et procedure-hoved, som angiver hvor mange og hvilke slags parametre, der skal overføres til proceduren.
- \* Selve procedure-kroppen, dvs. den maskinkode, der skal udføres ved et kald.

Denne tegning illustrerer den hierarkiske struktur:



USE-sætningen gør følgende: For hvert modul (begyndende fra det sidst indlæste modul) undersøges, om navnet efter USE findes i listen over pakkenavne i det pågældende modul. Hvis navnet findes, defineres de procedurer og funktioner, hvis navne findes i den pågældende pakkes liste over procedure- og funktionsnavne. Samtidig huskes på, hvor procedurehovederne kan findes.



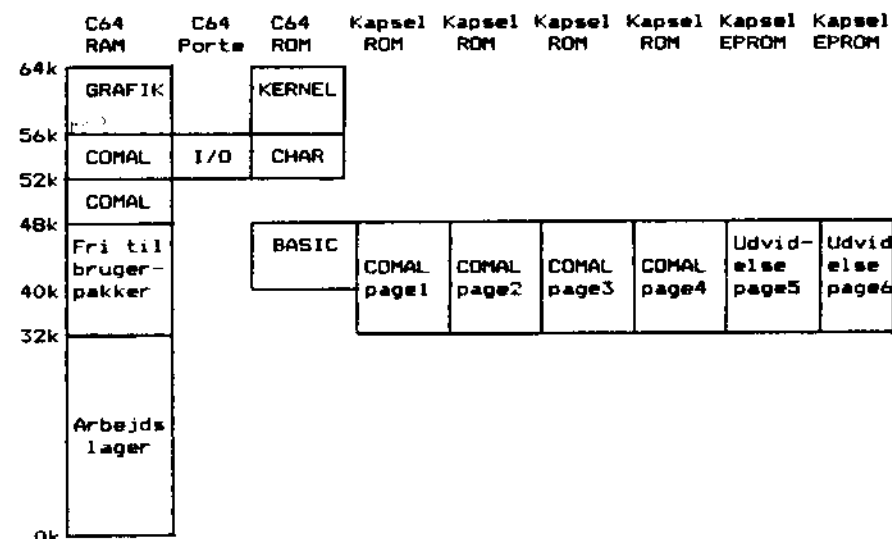
## SIGNALER

Når COMAL foretager operationer, der kan være af interesse for moduler eller pakker, gives et signal om denne operation. Modulet eller pakken kan så reagere eller lade være. Der er to slags signaler:

- \* Der udsendes et signal til en pakke, når der mødes en USE-sætning til den pågældende pakke. Signalet består i, at der kaldes en rutine, som er lokal til pakken. Som eksempel på, hvad en sådan rutine kan foretage sig, kan nævnes, at TURTLE-pakken vælger SPLITSCREEN, hvis en USE turtle kommando gives. Hovedformålet med rutinen er at initialisere pakkens variabler.
- \* Ved opstart, LINK, LOAD, DISCARD, NEW, RUN, CHAIN samt i nogle andre tilfælde, signaleres denne hændelse til alle moduler. Signalet består i, at der kaldes en rutine i modulet (som dermed er fælles for alle pakker i modulet). Formålet er, at et moduls pakker skal kunne integrere sig selv i COMAL-systemet (ved opstart, LINK, LOAD) og bringe COMAL-systemet tilbage til oprindelig tilstand (ved DISCARD, NEW). Hvis en pakke ønsker at benytte interrupt (IRQ), kan modulet hægte interrupt-rutinen ind ved LINK, og hægte den ud igen ved DISCARD.

## HVORDAN ER HUKOMMELSEN ORGANISERET?

Nedenstående diagram illustrerer al hukommelse i Commodore 64 (3 første søjler), hukommelse i COMAL-kapslen (4 næste søjler), og endelig EPROM-udvidelsen (2 sidste søjler), som kan programmeres af brugeren. Udvidelsen består af en tom EPROM-sokkel i COMAL-kapslen. Denne sokkel kan indeholde en 8KB-, 16KB-, eller 32KB-EPROM.



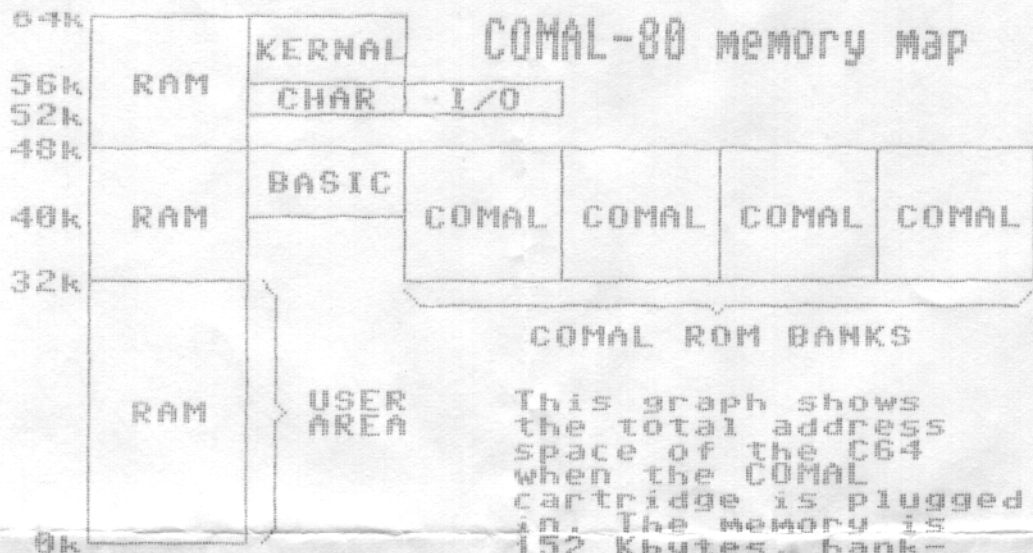
## RAM'en er opdelt således:

0- 1KB	Systemvariabler for KERNEL, COMAL, processorstak.
1- 2KB	Skærmhukommelse.
2-32KB	Lager for COMAL-program, navnetabel samt stak. Her kan også anbringes pakker, hvorved man tager af brugerlageret. Tegnsæt anbringes fra 27-32KB, hvis de anvendes.
32-48KB	Anvendes slet ikke. Her kan man frit anbringe pakker, som ikke skærer ned på brugerlageret.
48-52KB	COMAL-systemvariabler, variabler for standardpakker.
52-56KB	Variabler for funktionstaster, bevægende sprites, sprite-tegninger og farveinformation til grafik.
56-64KB	Grafik bitmap.

I/O-området indeholder input/output-porte. Herigennem foregår al kommunikation med omverdenen. Tekstskærmens farvehukommelse befinder sig her. Denne farvehukommelse deles (desværre) også med flerfarvegrafik.

## Der er følgende ROM'er i Commodore 64:

40-48KB	BASIC-fortolkeren.
52-56KB	Standard dobbelt tegnsæt.



This graph shows the total address space of the C64 when the COMAL cartridge is plugged in. The memory is 152 Kbytes, bank-switched.

56-64KB

KERNEL. Dette er Commodore 64's operativsystem. Det indeholder bl.a. rutiner til kommunikation med skærm, kassettebånd, diskettestationer og RS 232.

**COMAL-kapslen** er opdelt i fire sider (eng. *pages*) af hver 16KB, som alle befinder sig i adresseområdet 32-48KB. Herved opnås, at en 64KB COMAL-fortolker kun optager 16KB i Commodore 64.

Indholdet af kapsel-ROM'erne er følgende:

Page 1: Her starter COMAL, når maskinen tændes. Indeholder matematikrutiner, kommandoer, pakkerne ENGLISH, DANSK og SYSTEM.

Page 2: COMAL-editor, syntaksanalyse og kodegenerering, prepass (SCAN), recreator (LIST), kommandoer.

Page 3: Runtime-modul.

Page 4: Pakkerne GRAPHICS, TURTLE, SPRITES, FONT, SOUND, JOYSTICKS, PADDLES og LIGHTPEN.

### EPROM-udvidelsen i kapslen opfattes således, afhængig af EPROM-type:

8KB Page 5, adresseområde \$8000-\$9fff.  
 16KB Page 5, adresseområde \$8000-\$bfff.  
 32KB Page 5, adresseområde \$8000-\$bfff,  
 Page 6, adresseområde \$8000-\$bfff.

Ved opstart "kigger" COMAL for hver 4KB i page 5 og page 6 efter nogle bestemte bytes for at afgøre, om der er pakkemoduler. Derefter signaleres til modulerne, at maskinen er tændt.

### LAGERSTYRING

En 6510-mikroprocessor, som anvendes i Commodore 64, kan ikke adressere mere end 64 KB. Når COMAL-kapslen er aktiv, skal 6510'eren kunne adressere så meget som 152 KB! Derfor er det nødvendigt at anvende specielle tricks, for at kunne adressere mere lager. Tricket består i at kunne bestemme, hvad 6510'eren skal kunne "se" i et bestemt lagerområde. Hukommelsen deles op i *banks* (*pages*, *overlays*). Metoden kaldes ofte for *bank-switching* eller *memory management*. Eksempelvis er der i adresseområdet 52-56KB tre banks, nemlig RAM, I/O og Tegnsæt-ROM (se hukommelsesdiagrammet). I området 40-48KB er der hele 8 mulige banks.

Man vælger banks ved at skrive et bitmønster i en kontrolport. Der findes to sådanne kontrolporte, nemlig

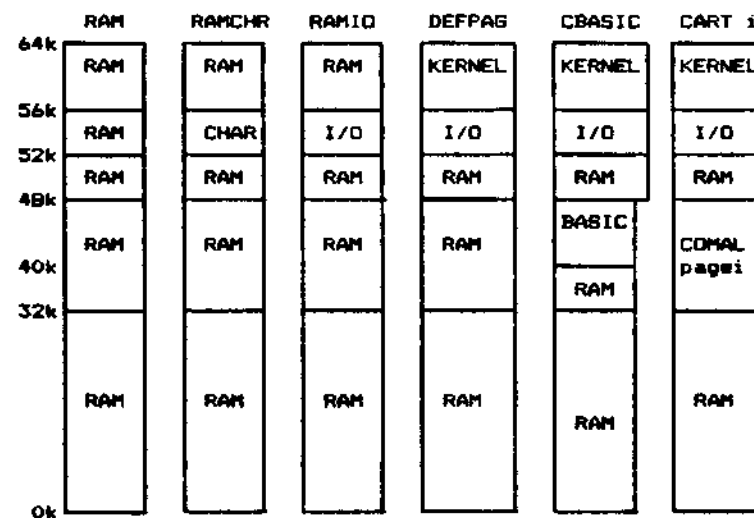
R6510

Styring af C64-memory map. Findes i Commodore 64, adresse \$0001. Den kan både skrives i og læses.

OVLAY

Styring af kapsel-banks. Findes i COMAL-kapslen, adresse \$de00 i bank I/O. Dvs. at portene skal være synlige, når denne port skal ændres. Den kan kun skrives (eng. *write-only*).

COMAL har systemrutiner, som manipulerer disse porte. Ved at bruge disse rutiner skal man kun specificere én byte for at bestemme memorymappen. Herunder er vist flere interessante memory maps (i,2,...,6):



### OPBYGNING AF MODULER

For at LINK, USE og DISCARD skal kunne virke, må placering af kode og formatet af pakkenavne, procedurer og procedurehoveder angives.

Dersom et modul skal placeres i RAM, har det følgende format:

```
.lib c64syb
*=<startadresse>
.byte <map>
.word end
.word <signal>
<pakketabel>
<maskinkode>
end
.end
```

Hvis modulet skal placeres i EPROM, har det følgende format:

```
.lib c64symb
*=<startadresse>
.word cold
.word warm
.byte 'CBM80comal'
.byte >*
.byte <map>+rommed
.word end
.word <signal>
<pakketabel>
<maskinkode>
end
.end
```

**.lib c64symb:** gør alle KERNEL- og COMAL-variabler kendte for modulet.

**<startadresse>:** er begyndelsesadressen for modulet i lageret.

**<map>:** angiver hvilken memory map, modulet skal anbringes i ved LINK. Denne memory map er automatisk aktiv ved kald af en procedure, funktion eller signal-handler i modulet.

**rommed:** angiver at modulet ikke kan DISCARD'es.

**end:** er modulets slutadresse + 1.

**<signal>:** er modulets signal-handler, som befinder sig i **<maskinkode>**.

**<pakketabel>:** er en liste over pakkenavne.

**<maskinkode>:** er al anden kode i modulet.

**En <pakketabel> har følgende format:**

```
.byte l1,'pakke1'
.word proct1,init1
.byte l2,'pakke2'
.word proct2,init2
.
.
.byte 0 ;Slut på tabellen
```

**l1:** er antallet af tegn i det i'te pakkenavn.

**'pakke1':** er navnet på den i'te pakke (i anførselstegn).

**proct1:** er adressen på den i'te tabel over procedurenavne.

**init1:** er adressen på initialiseringsrutinen for den i'te pakke.

**En tabel over procedurenavne skal have følgende format:**

```
procti .byte l1,'proc1'
        .word proch1
        .byte l2,'proc2'
        .word proch2
.
.
        .byte 0 ;Slut på tabellen
```

**procti:** er adressen på den i'te tabel over procedurenavne.

**lj:** er antallet af tegn i det j'te procedurenavn.

**'procl':** er navnet på den j'te procedure (i anførselstegn).

**prochj:** er adressen på det j'te procedurehoved.

**Et procedurehoved har dette format:**

```
prochj .byte proc,<kodeh,>kodeh,n
        .byte <parameter1>
        .byte <parameter2>
.
.
        .byte <parametern>
        .byte endprc
```

**Et funktionshoved har dette format:**

```
funchj .byte func+type,<kodeh,>kodeh,n
        .byte <parameter1>
        .byte <parameter2>
.
.
        .byte <parametern>
        .byte endinc
```

**kodeh:** er adressen på den assemblerkodede rutine.

**n** er antallet af formelle parametre.

**<parameterk>** er specifikation af den k'te parameter.

**type** er funktionstypen (real, int eller str)

**En parameterspecifikation er en af følgende:**

.byte	value+type	;Simpel værdiparameter
.byte	value+array+type,dim	;Talsæt værdiparameter
.byte	ref+type	;Simpel referenceparameter
.byte	ref+array+type,dim	;Talsæt referenceparameter

**type:** er parametertypen (real, int eller str)

**dim:** er antallet af dimensioner af en talsætparameter.

**real:** betyder af reel type (REAL).

**int:** betyder af heltallig type (INTEger).

**str:** betyder af streng type (STRing).

### Et eksempel på hvordan et procedurehoved kodes:

**FUNC pip(x,y#,REF z\$(,))** kan kodes som

```
.byte func+real,<pip,>pip,3      ;Reel funk. med tre parametre.
.byte value+real                  ;x
.byte value+int                   ;y#
.byte ref+array+str,2             ;REF z$(,)
.byte endfnc                      ;Ikke flere parametre
```

### PARAMETEROVERFØRSEL

Når COMAL-fortolkeren overgiver kontrollen til en assemblerkodet rutine, er alle aktuelle parametre (hvis der er nogen) beregnet. Samtidig er det checket, at parametrene's typer stemmer overens med de i procedurehovedet angivne parametre, samt at antallet af parametre er korrekt.

Man kan ikke på forhånd vide, hvor parameterværdien eller -variablen (ved REF) befinder sig i lageret. Derfor skal man kalde systemrutinen FNDPAR (FIND PARAMeter) for at få oplysning om adressen på en parameter. Herefter kan man selv behandle den.

**FNDPAR:**

**Ved kald:** .A er parametrens nummer.

**Ved return:** COPY1 indeholder parametrens adresse.

Alle registre er ændrede.

Bemærk: I COMAL-systemet er valgt den konvention, at heltal og reelle tal lagres i høj/lav-formatet, hvorimod adresser lagres i lav/høj-formatet. Dette gælder for parametre, men det gælder også for parametre til systemrutiner. Herunder beskrives formatet for hver enkelt parametertype.

### VALUE+REAL og REF+REAL

(COPY1)+0:			EkspONENT+128
+1:	5 bytes		Mantisse(1)
+2:	floating		Mantisse(2)
+3:	point		Mantisse(3)
+4:			Mantisse(4)

### VALUE+INT og REF+INT

(COPY1)+0:	2 bytes	Høje del
+1:	heltal	Lave del

### VALUE+STR og REF+STR

**m:** Maksimal længde af streng (dimensioneret længde).

**n:** Aktuel længde (Hvis VALUE+STR, så gælder  $m=n$ ).

(COPY1)+0:	m	Høje del Lave del
+2:	n	Høje del Lave del
+4:	m bytes	s\$(1:1) s\$(2:2) s\$(3:3) s\$(4:4)
+4+n-1:		s\$(n:n)(sidste tegn)
+4+m-1:		s\$(m:m)

**VALUE+ARRAY+REAL, VALUE+ARRAY+INT,  
VALUE+ARRAY+STR,  
REF+ARRAY+REAL, REF+ARRAY+INT, REF+ARRAY+STR**

Enhver tabel har en informationsblok:

**n** : Antal indices.

**adr**: Adressen på tabellens første element.

(COPY1)+0:	adr	Lave del Høje del
+2:	n	Antal indices
+3:	Nedre grænse for 1. indeks	Høje del Lave del
+5:	Øvre grænse for 1. indeks	Høje del Lave del
+7:	Nedre grænse for 2. indeks	Høje del Lave del
+9:	.. ..	
+3+(n-1)*4+2:	Øvre grænse for n'te ind.	Høje del Lave del
+3+n*4:		

Hvis en tabel A er erklæret som

**DIM a(1:3,6:8)**

er det anbragt i hukommelsen således:

adr+0: a(1,6)  
+1: a(1,7)  
+2: a(1,8)  
+3: a(2,6)  
+4: a(2,7)  
+5: a(2,8)  
+6: a(3,6)  
+7: a(3,7)  
+8: a(3,8)

hvor I er størrelsen (i bytes) af hvert element i tabellen.

Hvert element er organiseret akkurat som en simpel parameter.

## HVOR KAN MODULER ANBRINGES?

Moduler kan anbringes i RAM fra \$0900-\$7fff og fra \$8009-\$bfff.

Desuden kan pakker anbringes i EPROM i kapslen fra \$8000-\$bfff, dog skal startadressen være et multiplum af \$1000.

## HVOR KAN MODULETS VARIABLER ANBRINGES?

Variabler, som skal overleve fra kald til kald, må anbringes i selve modulet (for RAM-moduler).

EPROM-moduler kan anbringe variabler fra \$c855-\$c87a.

Dersom man har brug for mere og RS232 ikke anvendes, kan RSOBUF (256 bytes) anvendes. Hvis kasettebånd ikke anvendes, kan TBUFFR (192 bytes) anvendes. Desuden er zero-page lokationerne \$4c, \$56 og \$fb-\$ff til fri benyttelse.

Rutiner, som anvender variabler, der er lokale til det enkelte kald, kan anvende disse lokale variabler:

Navn	Adresse	
INF1	\$0038	
INF2	\$0039	
INF3	\$003a	
Q1	\$003b-\$003c	
Q2	\$003d-\$003e	
Q3	\$003f-\$0040	
Q4	\$0041-\$0042	
Q5	\$0043-\$0044	
COPY1	\$0045-\$0046	Benyttes også af FNDPAR
COPY2	\$0047-\$0049	
.tp10		
COPY3	\$0050-\$0051	
AC1	\$0061-\$0066	Benyttes også af FP-rutiner
AC2	\$0069-\$006f	Benyttes også af FP-rutiner
MOVEAD	\$007a-\$007b	
TXLO	\$007c	
TXTHI	\$007d	
RANGES	\$02e0-\$02ff	
TXTC	\$c760-\$c7af	

## SIGNALRUTINER

En signal-rutine er en subrutine, der afsluttes med en RTS-instruktion. I en signal-rutine er det tilladt at gøre alt hvad en procedure eller funktion må. Dersom man ikke har brug for en signal-rutine, kan en system-rutine ved navn DUMMY benyttes. Denne rutine består blot af en RTS-instruktion og udfører dermed intet.

En **USE-signal-rutine** har ingen parametre. Hver gang en **USE** <pakke> sætning mødes i et COMAL-program, kaldes denne ru-

tine. Hvis man ikke ønsker at initialisere pakken hver gang, må man anvende en variabel til at fortælle, at en pakke har været aktiveret med USE før.

En **modul-signal-rutine** har en parameter, idet .y-registret ved kaldet vil indeholde en værdi, som fortæller hvilken type signal, der er tale om. Parametren kan være en af disse:

#### POWER1

Gives ved opstart til alle ROM'ede moduler. Signalet skal anvendes til at initialisere modulet.

#### POWER2

Gives ved opstart efter at POWER1 er signaleret. Sædvanligvis ignoreres dette signal, men det kan anvendes til at lade et modul helt overtage styringen før COMAL starter.

#### LINK

Gives til en netop LINK'et pakke eller til de pakker, der indlæses ved LOAD, RUN <filnavn>, eller CHAIN. Ved dette signal kan modulet ændre vektorer i COMAL og operativsystem.

#### DSCRD

Gives før DISCARD eller NEW-kommandoen til alle moduler. Ved dette signal kan modulet ændre vektorer tilbage til, hvad de var før LINK.

#### NEW

Gives ved NEW-kommandoen.

#### CLRTAB

Gives når alle navne i et program gøres ikke-erklærede. Dette signal gives bl.a. ved RUN og CHAIN-kommandoerne. Når navnene er ikke-erklærede, kan man heller ikke kalde nogen procedure eller funktion i nogen pakke.

#### RUN

Gives ved RUN eller CHAIN-kommandoen.

#### WARM1

Gives ved "warm start", dvs. når der trykkes <STOP-RESTORE>.

#### CON

Gives ved CON-kommandoen.

#### ERROR

Gives efter at programmet er standset med en fejlmeddelelse.

#### STOP1

Gives efter at programmet er standset ved STOP eller END.

#### BASIC

Gives før COMAL forlades.

### En modul-signal-rutine følger som regel dette skema:

signal	cpy #link	;LINK-kommando ?
	beq slink	;Hop hvis det er
	cpy #dscrd	

### En modul-signal-rutine følger som regel dette skema:

signal	cpy #link	;LINK-kommando ?
	beq slink	;Hop hvis det er
	cpy #dscrd	;DISCARD ?
	beq sdscrd	;Hop hvis det er
	rts	;Ignorer alle andre signaler
;		
slink	..	;LINK-handler
	rts	;Tilbage til COMAL
sdscrd	..	;DISCARD-handler
	rts	;Tilbage til COMAL

#### FEJLGIVNING

Det er god programmeringskik at checke, om parametrene til en procedure eller funktion er lovlige. Hvis de ikke er det, så giv en fejlmeddelelse. Hvis man vil anvende COMAL-systemets egne fejlmeddelelser, kan dette gøres ved

Idx #5	;Giv fejl nummer 5
jmp runerr	;dvs. "value out of range"

Med denne metode kan man give standard-fejlmeddelelser med numrene 0 til 255. Se appendiks F med fejlmeddelelser. RUNERR svarer ganske nøje til COMAL-sætningen REPORT <fejl>, og kan fanges i en TRAP-struktur, hvis dette er ønskværdigt.

Der findes en mere *generel fejlgivningsmetode*. Hvis man ønsker at give følgende værdier til systemet eller en error-handler,

ERR	= 300
ERRFILE	= 0
ERRTEXT\$	= "ulovlig parameter værdi"

kan dette gøres med følgende rutine:





```

        .byte 6,'string'      ;funktionen string
        .word pstrin
        .byte 0              ;ikke flere procedurer
;
;proc hej
;
phej    .byte proc,<hej,>hej,0 ;ingen parametre
        .byte endproc       ;begynder i hej
;
;func add(a#,b#)
padd    .byte func+real,<add,>add,2 ;to parametre
        .byte value+int      ;a# er heltallig værdiparameter
        .byte value+int      ;b# er heltallig værdiparameter
        .byte endfnc
;
;func string$(tegn$,antal#)
pstrin   .byte func+str,<string,>string,2;to parametre
        .byte value+str      ;begynder i string
        .byte value+int      ;tegn$ er streng værdiparameter
        .byte endfnc         ;antal# er heltallig værdiparameter
;
;
;proc hej
;print "hej med dig"
;endproc hej
;
tekst    .byte 'hej med dig',13 ;tekst, der skal printes
tekstf   =*-tekst              ;længden af teksten
;
hej      ldy #0                ;begynd med 1. tegn
hejlp    lda tekst,y           ;hent tegn
        jsr cwrtn              ;print tegn til skærm
        iny                   ;næste tegn
        cpy #tekstf            ;færdig ?
        bne hejlp             ;hop, hvis ikke færdig
        rts                   ;vend tilbage til comal
;
;
;func add(a#,b#)
;return a#+b#
;endfunc add
;
add      lda #1                ;få fat i adressen på 1. parameter
        jsr fndpar             ;copy1 = adresse
        idx copy1              ;flyt adressen til copy2

```

```

        lda copy1+1
        stx copy2
        sta copy2+1
;
        lda #2                ;få fat i adressen på 2. parameter
        jsr fndpar
;
;copy1 peger på b# og copy2 peger på a#
;
        ldy #1                ;husk: heltal er i høj/lav format
        clc                   ;ingen mente
        lda (copy2),y         ;lave del af a#
        adc (copy1),y         ;plus lave del af b#
        tax                   ;flyttes over i .x
        dey                   ;.y:=0
        lda (copy2),y         ;høje del af a#
        adc (copy1),y         ;plus høje del af b# plus mente
        bvs ovrflw            ;hop, hvis aritmetisk overløb
;
; .x = lave del af a#+b#
; .a = høje del af a#+b#
;
;konverter fra heltal til reelt tal ;
;så læg resultatet på comal's stak.
;
        jsr pshint              ;konverter og push
        rts                   ;returner til comal med resultatet
;
ovrflw   ldx #2                ;"overflow"
        jmp runerr             ;report 2
;
;
;func string(tegn$,antal#) closed
;if antal#<then report 1 // argumentfejl //
;if len(tegn$)<>1 then report 1 // argumentfejl //
;dim r$ of antal# // plads til resultat //
;for i#=1 to antal# do // generer resultat //
;r$:=tegn$
;endfor i#
;return r$ // returner resultat //
;endfunc string
;
antal    =copy2                ;brug copy2 som antal
;
string    lda #2                ;få adressen på 2. parameter
        jsr fndpar
        ldy #0                ;test fortegn
        lda (copy1),y
        bmi argerr            ;hop, hvis <0
        sta antal+1           ;høje del i antal

```

```

    iny                ;y:=1
    lda (copy1),y
    sta antal          ;lave del i antal
;
;generer resultatet direkte på comal's evalueringsstak.
;
;stos peger på næste frie byte på stakken; stakken begrænses
;opadtil af sfree; test om der er plads til resultatet
;
    clc                ;slet menten
    adc stos           ;antal+stos
    tax                ;x:=lave del af antal+stos
    lda antal+1
    adc stos+1         ;a:=høje del af antal+stos
    bcs sterr          ;hop, hvis overløb
;
    tay
    txa                ;antal+stos+2
    adc #<2            ;menten vides at være = 0
    tax
    tya
    adc #>2
    bcs sterr          ;hop, hvis overløb
;
    cpx sfree          ;hvis antal+stos+2>=sfree,
    sbc sfree+1        ;så stack-overflow
    bcs sterr          ;hop, hvis stack-overflow
;
;undersøg tegn$.
;
    lda #1             ;få adressen på tegn$
    jsr fndpar
    ldy #2             ;aktuel længde skal være = 1
    lda (copy1),y
    bne argerr         ;høje del skal være = 0
    iny                ;y:=3
    lda (copy1),y
    cmp #1             ;lave del skal være = 1
    bne argerr
;
;få fat i tegn$(1:1)
;
    iny                ;y:=4
    lda (copy1),y
    ;a:=tegn$(1:1)
;
;skriv tegn$(1:1) antal gange på stakken.
;
    ldy #0
    sty q1              ;q1:=0 // løkkevariabel
    sty q1+1

```

```

;
strip    ldx antal+1    ;while q1<>antal do
          cpx q1+1
          bne str1
          ldx antal
          cpx q1
          beq strok
;
str1     sta (stos),y    ; r$(q1:q1):=tegn$(1:1)
;
          inc stos       ; tos:+1
          bne str2
          inc stos+1
;
str2     inc q1          ; q1:+1
          bne strip
          inc q1+1
          jmp strip      ;endwhile
;
;sæt længden af strengen til antal.
;
strok    lda antal+1
          sta (stos),y
          iny
          lda antal
          sta (stos),y
;
          clc            ;stos:=2 // plads til længden //
          lda stos
          adc #<2
          sta stos
          lda stos+1
          adc #>2
          sta stos+1
          rts
;
          ;vend tilbage til comal med resultatet
argerr   ldx #1          ;"argument error"
          jmp runerr
;
sterr    ldx #56         ;"out of memory"
          jmp runerr
;
end      .end            ;slut på kildetekst

```

For en grundigere gennemgang af brugen af pakker, som du selv kan lave til brug med Comal-kapslen, se bogen:

Jesse Knight, **Comal 2.0 Packages**

som kan fås ved henvendelse til:

Comal Users' Group  
5501 Groveland Ter.  
Madison, WI 53716-3251  
USA

## Appendiks A -

# COMMODORE 64 TEGNKODER

KODE	ASCII-TEGN	SKÆRMTEGN
	tilstand	tilstand
tekst	grafik	tekst grafik

0		Q	q
1		A	a
2		B	b
3	<STOP>	C	c
4		D	d
5	hvid	E	e
6		F	f
7		G	g
8	<SHIFT - C> låst	H	h
9	<SHIFT - C> åben	I	i
10	....	J	j
11	slet linien ud	K	k
12	ny side (printer)	L	l
13	<RETURN>	M	m
14	tekst tilstand aktiv	N	n
15		O	o
16	....	P	p
17	marker ned	Q	q
18	omvendt skrift start	R	r
19	marker hjem	S	s
20	<DEL>	T	t
21		U	u
22		V	v
23		W	w
24		X	x
25		Y	y
26		Z	z
27		[	[
28	red	\	\
29	marker til højre	^	^
30	grøn	_	_
31	blå	`	`
32	melletrum	mellemrum	
33	!	!	!
34	"	"	"
35	#	#	#
36	\$	\$	\$
37	%	%	%
38	&	&	&
39	'	'	'
40	(	(	(
41	)	)	)
42	*	*	*
43	+	+	+
44	,	,	,
45	-	-	-
46	.	.	.
47	/	/	/

KODE	ASCII-TEGN	SKÆRMTEGN	
	tilstand	tilstand	
	tekst	grafik	tekst

48	0	0	0
49	1	1	1
50	2	2	2
51	3	3	3
52	4	4	4
53	5	5	5
54	6	6	6
55	7	7	7
56	8	8	8
57	9	9	9
58	:	:	:
59	;	;	;
60	<	<	<
61	=	=	=
62	>	>	>
63	?	?	?
64	@	@	@
65	a	A	A
66	b	B	B
67	c	C	C
68	d	D	D
69	e	E	E
70	f	F	F
71	g	G	G
72	h	H	H
73	i	I	I
74	j	J	J
75	k	K	K
76	l	L	L
77	m	M	M
78	n	N	N
79	o	O	O
80	p	P	P
81	q	Q	Q
82	r	R	R
83	s	S	S
84	t	T	T
85	u	U	U
86	v	V	V
87	w	W	W
88	x	X	X
89	y	Y	Y
90	z	Z	Z
91	[	[	[
92	\	\	\
93	^	^	^
94	_	_	_
95	+	+	+
96	=	=	=
97	~	~	~
98			
99			
100			

KODE	ASCII-TEGN	SKÆRMTEGN	
	tilstand	tilstand	
	tekst	grafik	tekst

101	E	-	
102	F	-	
103	G	-	
104	H	-	
105	I	-	
106	J	-	
107	K	-	
108	L	-	
109	M	-	
110	N	-	
111	O	-	
112	P	-	
113	Q	-	
114	R	-	
115	S	-	
116	T	-	
117	U	-	
118	V	-	
119	W	-	
120	X	-	
121	Y	-	
122	Z	-	
123	[	-	
124	\	-	
125	^	-	
126	_	-	
127		-	
128	orange		
130			
131	<RUN>		
132			
133	<F1>		
134	<F3>		
135	<F5>		
136	<F7>		
137	<F2>		
138	<F4>		
139	<F6>		
140	<F8>		
141	<SHIFT-RETURN>		
142	grafiktilstand aktiv		
143			
144	sort		
145	markør op		
146	omvendt skrift slut		
147	som <CLR> (slet side)		
148	som <INST> (indsæt)		
149	brun		
150	lyserød		
151	mørkegrå		
152	grå		
153	lysegrøn		

128-255 er  
gentagelse af  
0 - 127, men i  
omvendt skrift



## KODE ASCII-TEGN

tilstand

tilstand

tekst

grafik

tekst

grafik

154	lyseblå	206	N	/
155	lysegrå	207	O	┐
156	violet (magenta)	208	P	┐
157	markør til venstre	209	Q	•
158	gul	210	R	—
159	turkis (cyan)	211	S	♥
160	mellemlum	212	T	
161	■	213	U	┐
162	■	214	V	X
163	—	215	W	o
164	—	216	X	•
165		217	Y	
166	■	218	Z	+
167		219	☐	+
168	■	220	☐	
169	☼	221	A	
170		222	☼	
171	└	223	☼	┐
172	└	224		┐
173	└	225	■	┐
174	└	226	■	┐
175	└	227	—	┐
176	└	228	—	┐
177	└	229		┐
178	└	230	■	┐
179	└	231		┐
180		232	■	┐
181		233	☼	┐
182		234		┐
183	■	235	└	┐
184	■	236	└	┐
185	■	237	└	┐
186	✓	238	└	┐
187	■	239	└	┐
188	■	240	└	┐
189	└	241	└	┐
190	■	242	└	┐
191	■	243	└	┐
192	—	244		┐
193	A	245		┐
194	B	246		┐
195	C	247	—	┐
196	D	248	■	┐
197	E	249	■	┐
198	F	250	✓	┐
199	G	251	■	┐
200	H	252	■	┐
201	I	253	└	┐
202	J	254	■	┐
203	K	255	☼	┐
204	L			
205	M			

Appendiks B -  
FARVER

Farve kode	Farve	Gråtone skala	ASCII værdi	Tastatur
0	sort	4/4	144	<CTRL-1>
1	hvid	0/4	5	<CTRL-2>
2	rød	3/4	28	<CTRL-3>
3	turkis (cyan)	1/4	159	<CTRL-4>
4	violet	2/4	156	<CTRL-5>
5	grøn	2/4	30	<CTRL-6>
6	blå	3/4	31	<CTRL-7>
7	gul	1/4	158	<CTRL-8>
8	orange	2/4	129	<C= 1>
9	brun	3/4	149	<C= 2>
10	lyserød	2/4	150	<C= 3>
11	mørkegrå	3/4	151	<C= 4>
12	grå	2/4	152	<C= 5>
13	lysegrøn	1/4	153	<C= 6>
14	lyseblå	2/4	154	<C= 7>
15	lysegrå	1/4	155	<C= 8>

FARVEKOMBINATIONER PÅ TV/MONITOR:  
(fra Commodore 64; Programmer's Reference Guide)

Hvordan passer farverne sammen?

+ = udmærket

o = rimeligt

= dårligt

skærm farve kode	tegn farvekode															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		+	+	+	+	o	+	+	+	+	+	+	+	+	+	+
1	+		+		+	+	+	+	+	o	+	+	+	+	+	+
2		+			o		+	+	+	+	+					o
3	+					o	+					o			o	
4	+	o														o
5	+	o		o								o		+		o
6	o	+	+	+										o	+	+
7	+		+			o			o	+	o	+	+			
8	o	+	+					+		+						o
9		+						+	+	+						+
10	o	o	+					o		+						o
11	+	+		o				+					+	+	o	+
12	+	+	o			o			o		+					+
13	+				+	o					+					
14	+	+		+		+					o					o
15	+	+	+		o	o	+			o	o	+	+	o		

## Appendiks C -

# TALBEHANDLING MED COMAL

COMAL systemet er i stand til at arbejde med 4 typer talkonstanter og talvariabler:

**reelle tal:** Eks: 3.232 , 4.6e-12 , PI, a , sum  
**heltal:** Eks: 71 , -3067, nr# , antal#  
**hexadecimale tal:** Eks: \$1a , \$d7 , \$ac00 , tal , sted  
**binære tal:** Eks: %1011 , %10011010 , byte , id

Bemærk, at decimaltal skrives med decimalpunkt . og ikke med komma , (f.eks. skrives halvanden som 1.5 og *ikke* som 1,5).

### Talområde:

2.93873588e-39	<= reelle tal	<= 1.70141183e+38
-32768	<= heltal	<= 32767
0 = \$00	<= hexadecimale tal	<= \$ffff = 65535
0 = %0	<= binære tal	<= %1111111111111111 = 65535

### Talberegninger sker efter følgende regler:

Et udtryk, hvis talværdi skal beregnes, kan indeholde en blanding af alle taltyper og talvariabler. Det kan indeholde en blanding af aritmetiske operatoren, sammenlignende operatoren og logiske operatoren. Der kan også indgå COMAL standardfunktioner og brugerdefinerede funktioner:

- \* et udtryk beregnes fra venstre mod højre
- \* dog må der tages hensyn til, at operatorerne har forskellig prioritet. Beregningen med operatoren med første prioritet skal udføres først:

### PRIORITET:

(nævnes i prioriteret rækkefølge)

1. ( ) parenteser

### Aritmetiske operatoren:

- |      |                  |               |
|------|------------------|---------------|
| 2. ↑ | potensopløftning | 2 ↑ 3 giver 8 |
| 3. * | multiplikation   | 2 * 3 giver 6 |

3.	/	division	7/2 giver 3.5
3.	DIV	heltalsdivision	54 DIV 8 giver 6
3.	MOD	rest efter division	23 MOD 7 giver 2
4.	+	addition	2+3 giver 5
4.	-	subtraktion	4-3 giver 1
4.	-	monadisk minus	-5+2 giver -3

**Logiske operatoren til bitvis sammenligning:**

(Se forklaring under de enkelte ord i kapitel 4):

5.	BITAND	bitvis logisk "og"
5.	BITOR	bitvis logisk "eller"
5.	BITXOR	bitvis logisk "eksklusiv eller"

**Sammenlignende operatoren:**

(Sammenligningen indgår i et logisk udtryk, hvis værdi er TRUE (=1), hvis sammenligningen er sand. Ellers har det logiske udtryk værdien FALSE (=0))

6.	<	mindre end	3*2<9 giver TRUE
6.	<=	mindre end eller lig med	4*3<=10 giver FALSE
6.	=	lig med	1=2 giver FALSE
6.	>=	større end eller lig med	17>3 giver TRUE
6.	>	større end	7>7 giver FALSE
6.	<>	forskellig fra	3*2<>6.01 giver TRUE

**Logiske operatoren:**

(Se forklaring under de enkelte ord i kapitel 4):

7.	NOT	logisk negation
8.	AND	logisk 'og'
8.	AND THEN	som AND
9.	OR	logisk 'eller'
9.	OR ELSE	som OR

**Standardfunktioner:**

INT(x)	Heldelen af x	INT(3.2) giver 3
ABS(x)	Absolutværdi af x	ABS(-2.5) giver 2.5
SGN(x)	Fortegn af x	SGN(-3) giver -1
SIN(x)	Sinus til x	SIN(PI/6) giver 0.5
COS(x)	Cosinus til x	COS(PI) giver -1
TAN(x)	Tangens til x	TAN(PI/4) giver 1
ATN(x)	Omvendt tangens	ATN(1) giver PI/4
LOG(x)	Naturlig logaritme	LOG(10) giver 2.3026
EXP(x)	Exponentialfunktion	EXP(2) giver 7.389
SQR(x)	Kvadratroden af x	SQR(9) giver 3

**Eksempler på brugerdefinerede talfunktioner:**

```

FUNC asin(x)
IF ABS(x)>1 THEN
RETURN x*PI/2
ELSE
RETURN ATN(x/SQR(1-x*x))
ENDIF
ENDFUNC asin

```

```

FUNC log10(x)
RETURN LOG(x)/LOG(10)
ENDFUNC log10

```

## Appendiks D -

# TASTATUR OG SKÆRMREDIGERING

### Specielle tasters betydning i COMAL:

<←>

Understregning

<CTRL>

har speciel betydning sammen med andre taster. Se senere.

<RUN/STOP>

afbryder programudførelse.

Indvirker på COMAL sætningen ESC.

<SHIFT/LOCK>

fastlåser <SHIFT> i øverste position

løses ved nyt tryk på tasten

<SHIFT>

Hvis en taste trykkes ned, mens <SHIFT> holdes nede, skrives de bogstaver/tegn, som står øverst på tasterne. Bogstaver skrives med stort. I semigrafisk tilstand vil tegnene til højre, foran på tasterne udskrives. <SHIFT> sammen med de andre specialtaster har endvidere en særlig effekt, som beskrives under disse taster.

<C=> Commodore tasten:

<C= SHIFT>

Ved hvert tryk skiftes mellem små/store bogstaver og store bogstaver/grafik tilstandene.

<C= tal>

Tryk på tallene 1-8 skifter til farverne med farvekoderne 8-15.

<C= tegn>

Tryk på taste med grafisk tegn giver tegnet til venstre, foran på tasten.



**Andre taster:****<CLR/HOME>**

flytter markøren op i øverste venstre hjørne af skærmen.

**<SHIFT-CLR/HOME>**

sletter skærm billedet.

**<INST/DEL>**

er slette tasten, som sletter tegnet umiddelbart til venstre for markøren, og teksten sammentrækkes.

**<SHIFT-INST/DEL>**

er indsætte tasten, som skubber tegnet under markøren en plads til højre, så et nyt tegn kan indsættes.

**<STOP-RESTORE>**

Hvis <STOP> og <RESTORE> tasterne trykkes ned samtidigt, nulstilles computeren uden at slette programmet i arbejdslageret (panik-tasten!).

**<RETURN>**

Den indtastede information sendes til COMAL systemet, behandles og lagres i hukommelsen.

**<CRSR>**

Der er to markørtaster, som bruges til at flytte markøren (engelsk: *cursor*) omkring på skærmen. Pilene angiver retningen. Hver taste har to funktioner afhængigt af, om <SHIFT> også er trykket ned.

**Funktionstasterne (<f1> - <f8>)**

Funktionstasterne kan programmeres til at udføre forskellige funktioner. Se under beskrivelsen af COMAL-pakken **system** (proceduren **defkey**).

Ved opstart har tasterne følgende funktion:

<f1>	RENUM + <RETURN>
<f2>	MOUNT + <RETURN>
<f3>	USE turtle + <RETURN>
<f4>	AUTO
<f5>	EDIT
<f6>	LIST
<f7>	RUN + <RETURN> + CHR\$(11) + <RETURN> (Se uddybning.)
<f8>	SCAN + <RETURN>

---

**Om <f7>:** Foruden normal kørsel af et program, kan tasten benyttes til at køre et program direkte fra kataloget, idet RUN, RETURN bevirker, at programmet forsøges kørt; men teksten **prg** i katalogets sidste søjle hører ikke med til et lovligt programnavn, så systemet reagerer med en fejlmeddelelse, og markøren placeres umiddelbart foran **prg**. ASCII-koden 11 sletter så resten af linien (dvs. **prg**). Nu vil linien være korrekt og kan bringes til udførelse med det sidste RETURN.

---

Under programudførelse har funktionstasterne andre værdier, nemlig ASCII værdierne 133 - 140.

Efter udførelse af en af ordrene **USE graphics** eller **USE turtle** har <f1>, <f3> og <f5> tasterne betydningen:

<f1>	textscreen (tekstskærmen vises)
<f3>	splitscreen (grafikskærm med 4 tekstlinier vises)
<f5>	graphicscreen (grafikskærmen vises)

**Kontrollasten <CTRL>:****<CTRL-tal>**

<CTRL> sammen med et af tallene 1 til 8 vil bevirke, at efterfølgende tekst skrives med den farve, der er anført foran på tasten. <CTRL> sammen med 9 og 0 bevirker, at omvendt skrift slås henholdsvis til og fra.

Se endvidere appendiks B om farver og kapitel 5 om proceduren **quote'mode** i COMAL-pakken **system**

**Ved COMAL REDIGERING bruges følgende CTRL-funktioner:****<CTRL> + <bogstaver>**

**<CTRL-a>:** Bruges under rettelse i en brudt programlinie. Hvis de første 1 til 4 tegn i den linie, markøren befinder sig i, er et linienummer, udskrives programlinien ubrudt. <CTRL-a> kan også bruges som UPS!-taste: Hvis der er foretaget rettelse i en linie og endnu ikke trykket på <RETURN>, vil tryk på <CTRL-a> bevirke, at linien udskrives i sin oprindelige form.

<CTRL-b>	: Flyt markøren et ord tilbage
<CTRL-c>	: Svarer til <STOP>
<CTRL-d>	: Udskriv grafiksiden til printer, idet udskriften skal begynde 13 tegn inde på papiret. Benyttes kun til MPS801 kompatibel matrixprinter.

<CTRL-e>	: Skift markørfarve til hvid
<CTRL-f>	: Flyt markøren et ord frem
<CTRL-k>	: Slet fra markøren til slutningen af linien
<CTRL-l>	: Flyt markøren hen til efter det sidste ikke-blanke tegn på linien
<CTRL-m>	: Svarer til <RETURN>
<CTRL-p>	: Foretag en <b>hardcopy</b> ("lp:"), dvs. en udskrift af tekstsiden til printer. Udskriften starter med et linieskift.
<CTRL-s>	: Svarer til <CLR/HOME>
<CTRL-u>	: Slå grafikens overlejring af funktionstasterne <f1>, <f3> og <f5> til/fra. Se under beskrivelsen af funktionstasterne
<CTRL-v>	: Stil farvevalg som ved <b>textcolors(6,6,1)</b> , d.v.s. blå kant, blå baggrund og hvid efterfølgende tekst
<CTRL-w>	: Stil farvevalg som ved <b>textcolors(11,15,0)</b> , d.v.s. gråtoneskala: mørkegrå kant, lysegrå baggrund med efterfølgende sort tekst
<CTRL-x>	: ændr kantfarve... Skal efterfølges af farvevalg med <CTRL tal> eller <C= tal>
<CTRL-y>	: ændr baggrundsfarve... Skal efterfølges af farvevalg med <CTRL tal> eller <C= tal>
<CTRL-z>	: De aktuelle kant-, baggrunds- og markørfarver huskes og benyttes efter nulstilling med <STOP-RESTORE>

## Appendiks E -

# TEKSTBEHANDLING MED COMAL

Tekstvariabler angives i COMAL med et navn på op til 80 tegn, efterfulgt af et \$-tegn. Det første tegn i navnet skal altid være et bogstav. Visse specialtegn må ikke indgå i navnet.

Eks. navn\$, by\$, text\$, fra\$, t\$, langt'navn\$.

Før en tekstvariabel kan benyttes, skal den erklæres (dimensioneres), dvs. man skal fortælle systemet, hvor mange tegn, der skal gøres plads til under dette variabelnavn.

Eks. DIM tekst\$ OF 80

DIM navn\$ OF 20

DIM svar\$ OF 1

En tekstvariabel kan indeholde en tekst på op til den dimensionerede længde. (Undtagelse: tegnet " må ikke anvendes alene. Hvis dette tegn skal med i en streng kan man dog bruge "" for at markere det. Hvis et tal omslutes af anførselstegn, bliver tilsvarende ASCII tegn en del af strengtildelingen.)

Hvis man undlader at dimensionere en tekstvariabel, vil første tildelingssætning automatisk udføre: **DIM variabelnavn\$ OF 40**. Hvis en tekstvariabel ikke dimensioneres, og navnet benyttes, før variabelen er tildelt en værdi, vil der gives en fejlmeddelelse.

### Eksempler på tekstbehandling:

Lad

slogan\$:="comal er sundt"

og

text\$:="en blomst er smuk"

Teksten kan analyseres med standardfunktioner og operatorer:

lang:=LEN(slogan\$)

lang tildeles værdien 14, da slogan\$ består af 14 tegn. Se den nærmere beskrivelse af funktionen LEN i kapitel 4.

position:="mal" IN slogan\$

position tildeles værdien 3, da teksten "mal" er indeholdt i

**ascii=CHRS(text\$)**

**slogan\$<text\$**

**slogan\$**, og første tegn i "mal" er 3. tegn i **slogan\$**. Se den nærmere beskrivelse af operatoren IN i kapitel 4.

**ascii** tildeles ASCII værdien af bogstavet **e** (= 69). Se ASCII værdierne for alle tegn i appendiks A.

det logiske udtryk vil være sandt (= TRUE = 1), da **c** kommer før **e** i alfabetet.

### Udvælgelse af deltekst:

**bogstav\$=text\$(8):**

eller

**bogstav\$=text\$(8:8)**

**bogstav\$** tildeles strengen "s", der er det 8. tegn i **text\$**.

**først\$=slogan\$(5)**

**først\$** tildeles teksten "comal", dvs. de 5 første tegn i **slogan\$**.

**sidst\$=text\$(14:)**

**sidst\$** tildeles teksten "smuk", dvs. de sidste fire tegn i **text\$**, eller

**sidst\$=text\$(LEN(text\$)-4:)**

**t\$=slogan\$(7:12)**

**t\$** tildeles teksten "er sun".

**t\$="programmer"(5:7)**

**t\$** tildeles teksten "ram".

**t\$=STR\$(1789)(2:3)**

**t\$** tildeles strengen "78".

**t\$=(text\$(4:9))(2:4)**

**t\$** tildeles strengen "lom", som er en delstreng af en delstreng.

**text\$(4:9):="bi"**

**text\$** vil efter denne sætning have udseendet "en bi er smuk".

### Udvælgelse af tekstdele fra indicerede strengvariabler:

**DIM navn\$(3) OF 20**

**navn\$(1):="Adam Smith"**

**navn\$(2):="Eva Smith"**

**navn\$(3):="Krystle Smith"**

**t\$=navn\$(2)(1:5)**

**t\$** tildeles strengen "Eva S".

**DIM vare\$(3,2) OF 10**

**vare\$(1,1):="bog"**

**vare\$(1,2):="blad"**

**vare\$(2,1):="bil"**

**vare\$(2,2):="log"**

**vare\$(3,1):="olie"**

**vare\$(3,2):="gas"**

**valg\$=vare\$(2,1)(2:3)**

**valg\$** tildeles strengen "il".

### Sammenkædning af strenge:

**sted\$="Odense"+"stadion"**

tekststrenge sammenkædes med et = tegn.

**besked\$=slogan\$+" og godt"**

**besked\$** tildeles strengen "comal er sundt og godt".

**hallo\$=navn\$(2):(3)+text\$(10:)+" og "+slogan\$(10:13)**

**hallo\$** tildeles strengen "Eva er smuk og sund"

**t\$=("vi og "+slogan\$(1:5))(4:8)** **t\$** tildeles strengen "og co".

### TEKSTFUNKTIONER:

Man kan definere egne strengfunktioner og uddrage delstrenge af strengfunktioner:

```
0010 FUNC stor$(lille$)
0020   FOR i#:=1 TO LEN(lille$) DO
0030     a:=ORD(lille$(i#))
0040     IF a>64 AND a<94 THEN
0050       a:=+128
0060       lille$(i#):=CHRS(a)
0070     ENDIF
0080   ENDFOR i#
0090   RETURN lille$
0100 ENDFUNC stor$
```

Eksempler på brug af funktionen **stor\$**:

**PRINT stor\$("Handlekraftig")** giver udskriften:

HANDLEKRAFTIG

**PRINT stor\$("Overskrift:")(4:8)** giver udskriften:  
RSKRI

I COMAL kan man hurtigt definere Basic-funktionen **mid\$**:

```
0010 FUNC mid$(a$,start,antal)
0020 RETURN a$(start:start+antal-1)
0030 ENDFUNC mid$
```

Denne funktion kan indsættes i tidligere Basic programmer.

## Appendiks F -

# COMAL FEJLNUMRE OG FEJLTEKSTER

Commodore 64 COMAL indeholder to nationaliteter af fejltekster som standard. Når maskinen tændes med COMAL, anvendes engelske fejlmeddelelser. Ellers vælges danske eller engelske fejlmeddelelser med ordrene

**USE dansk**

eller

**USE english**

Herefter vil alle fejlmeddelelser (bortset fra meddelelser fra diskoperativsystemet, som altid er på engelsk) være på det valgte sprog.

COMAL-systemet kan give fejlmeddelelser ved følgende tre lejligheder:

Ved indtastning af linier  
Ved strukturundersøgelse (brug af **scan**)  
Ved kørsel (run-time fejl)

Her følger en liste med fejlmeddelelser og deres eventuelle fejlnummer:

### DYNAMISKE SYNTAKS-FEJLMEDDELELSER (INDTASTNING):

<sprogelement> ikke forventet  
<sprogelement> not expectet  
Der forventedes ikke mere på linien.

<sprogelement> mangler  
<sprogelement> missing  
Der forventes flere sprogelementer på linien.

<sprogelement 1> forventet, ikke <sprogelement 2>  
<sprogelement 1> expected, not <sprogelement 2>  
Der forventes andre sprogelementer end det angivne.

## DYNAMISKE STRUKTUR-FEJLMEDDELELSER (PREPASS SCAN):

<sætning 1> uden <sætning 2>  
<sætning 1> without <sætning 2>

<sætning> mangler  
<sætning> missing

<sætning 1> forventet, ikke <sætning 2>  
<sætning 1> expected, not <sætning 2>  
Åbnings- og lukningssætning passer ikke sammen.

<sætning> ikke tilladt i styrestrukturer  
<sætning> not allowed in control structures  
DIM, DATA, IMPORT, PROC og FUNC må ikke forekomme i styrestrukturer.

import kun tilladt i lukket proc/func  
import allowed in closed proc/func only

forkert slags <sætning>  
wrong type of <sætning>  
Eks: tekst i WHEN linie, hvor numerisk udtryk forventes; forskellige variable i FOR og ENDFOR.

forkert navn i <sætning>  
wrong name in <sætning>  
ENDFOR, ENDPROC og ENDFUNC skal benytte samme navn som hhv. FOR, PROC og FUNC.

<navn>: navn allerede defineret  
<navn>: name already defined  
Indenfor et givet virkefelt må samme navn ikke forekomme med forskellig betydning: f.eks. a,a#,a\$.

<navn>: ukendt etikette  
<navn>: unknown label  
Der findes ikke en etikette til GOTO indenfor samme virkefelt. Eks: man kan ikke hoppe ud af en procedure ved hjælp af GOTO.

ulovlig goto  
illegal goto  
Man kan ikke hoppe ind i en struktur ved hjælp af GOTO.

## DYNAMISKE RUN TIME-FEJLMEDDELELSER:

<navn>: ukendt sætning eller procedure  
<navn>: unknown statement or procedure  
Eks: kald af pakke-procedure uden forudgående aktivering af pakken med USE pakkenavn.

<navn>: ikke en procedure  
<navn>: not a procedure  
Navnet er en variabel, funktion, pakke eller etikette, men ikke en procedure.

<navn>: ukendt variabel  
<navn>: unknown variable  
Variablen er ikke tildelt en værdi inden for det givne virkefelt.

<navn>: forkert type  
<navn>: wrong type  
Eks: Variablen er en streng men forsøges benyttet som talvariabel.  
Eks: Husk RESTORE etikette: skal stå umiddelbart foran en DATA-sætning.

<navn>: forkert funktionstype  
<navn>: wrong function type

<navn>: hverken tabel eller funktion  
<navn>: not an array nor a function  
Navnet står måske for simpel variabel, procedure eller andet.

<navn>: ikke en simpel variabel  
<navn>: not a simple variable  
Navnet står måske for en tabel, procedure eller andet.

<navn>: ukendt tabel eller funktion  
<navn>: unknown array or function  
Navnet er ikke erklæret eller defineret.

<navn>: forkert tabeltype  
<navn>: wrong array type  
Eks: To-dimensional tabel forsøges kaldt som én-dimensional.

<navn>: import fejl  
<navn>: import error  
Navnet IMPORT sætningen er ukendt eller af forkert type.

<navn>: ukendt pakke

<navn>: unknown package

COMAL systemet kender ikke pakken med det i USE anførte navn. Husk pakker fra diskette skal LINK'es.

<navn>: navn redefineret

<navn>: array redefined

Eks: Tabellen eller strengvariablen er tidligere defineret. Man kan ikke udvide tabellen eller strengvariablen.

<navn>: navn allerede defineret

<navn>: name already defined

Samme navn kan ikke bruges til forskellige typer.

<navn>: tekstvariabel ikke defineret

<navn>: string not dimensioned

<navn>: ikke en pakke

<navn>: not a package

#### **RUN TIME FEJL, SOM KAN TRAP'ES:**

0 report fejl  
report error

Report uden parametre skal kun benyttes i TRAP-strukturens HANDLER-del.

1 argument fejl  
argument error

Eks: kvadratroden af negativt tal; logaritmen til et ikke positivt tal

2 overloeb  
overflow

Der benyttes for store tal. Se appendiks C.

3 division med nul  
division by zero

4 deltekst-fejl  
substring error

Eks: Ved a\$: = tekst\$ (fra:til) skal gælde 1<fra til + 1.

Eks: Ved tekst\$ (fra:til): ==a\$ skal yderligere gælde til <=LEN(tekst\$).

5 uden for vaerdiomraade  
value out of range

6 step = 0  
step = 0

7 ulovlige graenser  
illegal bound

I DIM sætninger skal gælde: nedre grænse<=øvre grænse.

8 fejl i print using  
error in print using

Formatet mangler eller har forkert syntaks.

10 ulovlig indexvaerdi  
index out of range

Indeks overskrider grænserne for DIM-sætningen.

11 ulovligt filnavn  
invalid file name

Et filnavn må højst være 69 tegn langt.

13 verify fejl  
verify error

Der er uoverensstemmelse med programfilen og programmet i arbejdslageret. Husk nye navne eller fejlindtastninger vil ændre programmet.

14 program for stort  
program too big

15 daarlig comalkode  
bad comal code

Programfilen er ændret, eller der er transmissionsfejl.

16 ej save-fil  
not comal program file

Programmet er eventuelt et Basic-program.

17 program til anden comalversion  
program for other comal version

18 ukendt filattribut  
unknown file attribute

30 ulovlig farve  
invalid color

Bemærk, at -1<=farvekode<=15.

31 ulovlig grænse  
invalid boundary

I viewport: 0<=xmin<=xmax<=319 og 0<=ymin<=ymax<=199.

- 32 ulovlig tegning-nummer  
invalid shape number

Der gælder:  $0 \leq \text{tegning-nummer} \leq 47$  (hvis skildpadden vises, så 46).

- 33 tegningens længde skal være 64  
shape length must be 64

- 34 ulovlig sprite-nummer  
invalid sprite number

Bemærk, at  $0 \leq \text{sprite-nummer} \leq 7$  (hvis skildpadden vises, så 6).

- 35 ulovlig stemme  
invalid voice

Bemærk, at  $1 \leq \text{stemme} \leq 3$ .

- 36 ulovlig node  
invalid note

Se under **note** og **frequency**.

#### **RUN TIME FEJL, SOM IKKE KAN TRAP'ES:**

- 51 system fejl  
system error

Alvorlig fejl i Comal-systemet. Forsøg redning med NEW.

- 52 for lidt hukommelse  
out of memory

Der var for lidt plads til program, navne, data og funktionskald. Al hukommelse frigives ved fejlen uden at programmet ødelægges. Eks: for stor dybde i rekursive kald.

- 53 forkert dimension i parameter  
wrong dimension in parameter

Den aktuelle og den formelle parameter i procedurekald skal have samme dimension.

- 54 parameter skal være en tabel  
parameter must be an array

Hvis den formelle parameter er en hel tabel, skal den aktuelle parameter også være det.

- 55 for faa indices  
too few indices

Tabellen er kaldt med for få indices.

- 56 kan ikke tildele variabel  
cannot assign variable

Eks: En tildeling er forsøgt til et navn, som ikke er en tekstvariabel.

- 57 ikke implementeret  
not implemented

- 58 con ikke mulig  
con not possible

CON er ikke tilladt, når:

- 1) maskinen tændes
- 2) efter NEW, LINK, DISCARD og SCAN
- 3) efter en fejl
- 4) afbrudt kommando (f.eks. procedurekald)
- 5) programafslutning med END
- 6) programmet er ændret
- 7) der er tilføjet et nyt navn i navnetabellen

- 59 programmet er blevet modificeret  
program has been modified

Eks: Efter modificering kan en procedure først kaldes igen som direkte kommando, efter et nyt RUN eller SCAN.

- 60 for mange indices  
too many indices

- 61 funktionsværdi ikke returneret  
function value not returned  
RETURN er ikke udført i funktionen.

- 62 ikke en variabel  
not a variable

- 67 parameterlister afviger eller ikke lukket  
parameter lists differ or not closed

Den eksterne procedure skal være lukket, og stemme overens med kaldet med hensyn til parametre.

- 68 ingen lukket proc/func i fil  
no closed proc/func in file

Forsøg på at kalde en lukket procedure eller funktion, som ikke findes på den indlæste fil.

- 69 for faa parametre  
too few parameters

Der er for få parametre i procedure- eller funktionskaldet.

- 70      forkert indextype  
wrong index type  
Et strengudtryk er ikke et gyldigt indeks.
- 71      parameter skal være en variabel  
parameter must be a variable  
En REF parameter skal kaldes med en variabel og ikke en konstant.
- 72      forkert parametertype  
wrong parameter type  
Parameteren i procedurekaldet og procedurehovedet er ikke af samme type.
- 73      ikke-ram indlæsning  
non-ram load  
Der er forsøgt indlæst en pakke i et adresseområde uden ledig RAM-lager.
- 74      checksumfejl i objektfil  
checksum error in object file  
Der er en fejl i den LINKede objektfil.
- 75      hukommelsesområde beskyttet  
memory area is protected  
Et modul er forsøgt LINKet ind oven i anden pakke eller Comal-programmet.
- 76      for mange biblioteker  
too many libraries  
Et pakkemodul kaldes også et bibliotek. Antal biblioteker  $\leq 10$ , men vilkårligt antal pakker.
- 77      ikke en objektfil  
not an object file  
Der er forsøgt LINKet en ikke objekt fil.
- 78      ingen passende when  
no matching when  
Et CASE-udtryk passer ikke til nogen WHEN-linie. Tilføj evt. OTHERWISE.
- 79      for mange parametre  
too many parameters  
Der er for mange parametre i procedurekaldet.

**SYNTAKSFEJL:**

- 101      syntaksfejl  
syntax error  
Comal-systemet kan ikke finde mere passende fejlmeddelelse.
- 102      forkert type  
wrong type  
Sætningen indeholder udtryk af forkert type.
- 103      sætning for lang eller for kompliceret  
statement too long or too complicated
- 104      kun som sætning, ikke som kommando  
statement only, not command
- 106      linienumre er fra 1 til 9999  
line number range: 1 to 9999
- 108      procedure/funktion findes ikke  
procedure/function does not exist
- 109      struktureret sætning ikke tilladt her  
structured statement not allowed here  
Struktureringsætning er ikke tilladt i enkeltlinieversion af IF-, FOR-, WHILE og REPEAT sætning.
- 110      ikke en sætning  
not a statement  
Linien kan ikke begynde med det anførte tegn.
- 111      linienumre vil overskride 9999  
line numbers will exceed 9999  
Hvis AUTO, RENUM eller MERGE fortsætter, vil linienumrene overskride 9999.
- 112      kilde beskyttet!!!  
source protected!!!  
I Comal kan linien beskyttes mod LISTning. Se program på demo-disketten.
- 113      ulovligt tegn  
illegal character  
Symbolet kan ikke begynde med det anførte tegn.



114 fejl i konstant  
error in constant  
Syntaksen for den reelle, binære eller hexadecimale konstant er forkert.

115 fejl i eksponent  
error in exponent  
Den reelle konstants eksponentdel er ikke syntaktisk korrekt.

# **INPUT/OUTPUT- FEJLMEDDELELSER, SOM ALLE KAN TRAP'PES:**

200 ikke flere datalinier  
end of data  
Forsøg på at læse forbi den sidste DATA-værdi.

201 slut paa fil  
end of file  
Forsøg på at læse forbi den sidste post i en sekventiel fil.

202 fil allerede aaben  
file already open  
En fil med samme strømnummer er åben i forvejen.

203 fil ikke aaben  
file not open

204 ikke en inputfil  
not input file  
Man kan ikke læse fra en fil, der er åbnet med WRITE.

205 ikke en outputfil  
not output file  
Man kan ikke skrive på en fil, der er åbnet med READ.

206 numerisk konstant forventet  
numeric constant expected  
Forsøg på at indlæse en ikke-numerisk værdi.

207 ikke en random access fil  
not random access file

Filen er åbnet som en sekventiel fil, så man kan ikke adressere en individuel post.

208 enhed ikke tilstede  
device not present  
Den valgte enhed på den serielle bus er ikke tilsluttet.

209 for mange filer aabne  
too many files open  
Der må højst være 9 filer åbne ad gangen. Og kun én fil med direkte tilgang åben ad gangen.

210 læsefejl  
read error  
Under indlæsning fra den serielle bus er der ikke svaret inden for den afsatte tid.

211 skrivefejl  
write error  
Under skrivning til den serielle bus er der ikke svaret inden for den afsatte tid.

212 kort blok paa baand  
short block on tape

213 lang blok paa baand  
long block on tape

214 checksumfejl paa baand  
checksum error on tape

215 slut paa baand  
end of tape

216 fil ikke fundet  
file not found

217 ukendt enhed  
unknown device

218 ulovlig operation  
illegal operation

219 i/o afbrydelse  
i/o break

# **MEDDELELSER FRA DISKOPERATIVSYSTEMET (KUN ENGELSK):**

220	read error (Blok hovede findes ikke)
221	read error (Manglende synkroniseringsmærke)
222	read error (Datablokken er ikke til stede)
223	read error (Checksumfejl i datablok)
224	read error (Fejl i dekodning af oktet)
225	write error (Skriv/læs fejl)
226	write protect on (Disketten skrivebeskyttet)
227	read error (Checksum fejl i headeren)
228	write error (Lang datablok)
229	disk id mismatch (U-MOUNTed diskette el. lign.)
230	syntax error (Almindelig syntaksfejl)
231	syntax error (Ukorrekt DOS-kommando)
232	syntax error (For lang linie)
233	syntax error (Ukorrekt filnavn)
234	syntax error (Der er ikke angivet nogen fil)
239	syntax error (Ukorrekt pass-kommando)
250	record not present (Læsning forbi sidste post)
251	overflow in record (Postlængden overskredet)
252	file too large (Ikke plads til random-filen)
260	write file open (Allerede åben fil forsøgt åbnet)
261	file not open (Forsøgt adgang til ikke åbnet fil)
262	file not found (Filen findes ikke på diskettestationen)

263	file exists (Filen eksisterer i forvejen)
264	file type mismatch (Ikke samme filtype)
265	no block (Block reserveret i forvejen)
266	illegal track and sector (Spor/sekter findes ikke)
267	illegal system t or s (Ulovligt systemspor eller sektor)
270	no channel (Der er ingen kanal ledig)
271	dir error (Katalogfejl)
272	disk full (Disketten er fyldt op)
273	cbm dos vx.x yyyy (Diskette status)
274	drive not ready (Ingen diskette)

## Appendiks G -

# BRUGERKOMMENTARER OG RETTELSE

Disse sider er beregnet til kommentarer og rettelser fra dig. Når der er opnået yderligere erfaringer med brugen af denne COMAL håndbog, vil den blive genoptrykt. Det vil da være en stor fordel for alle brugere, at der kan tages hensyn til dine kommentarer eller rettelser, som evt. kan komme med i næste udgave. På forhånd mange tak for hjælpen!

Dine kommentarer kan sendes til:

COMMODORE DATA A/S  
ATT: COMAL 80 Håndbog  
Jens Juhtsvej 42  
8000 Århus C

## Appendiks H -

# COMAL PROGRAM- EKSEMPLER

### Pakken sound

```

0010 // musiklektion 1
0020 DIM node$ OF 3
0030 USE sound
0040
0050 LOOP
0060 PAGE
0070 PRINT "Stemmevalg (1,2 eller 3)"
0080 PRINT "Tonevalg (a2,c4,b3,...)"
0090 PRINT "Tallene angiver oktaven:"
0100 PRINT "'c4' svarer til C i 4. oktav (440 Hz)"
0110 PRINT "'f5#' er 'kvart f' i oktaven ovenover"
0120 PRINT AT 22,1: "LEKTION 1: Vi spiller en enkelt node"
0130 PRINT AT 20,1: "(Tryk <RUN/STOP> for at slutte...)"
0140 PRINT
0150 INPUT AT 8,1: "Stemme: ": stemme
0160 INPUT AT 9,1: "Tone: ": node$
0170 spil(1,node$)
0180 ENDLOOP
0190
0200 PROC spil(stemme,node$)
0210 IF node$("<z") THEN
0220     note(stemme,node$)
0230     gate(stemme,1) // stig og fald
0240 ENDIF
0250 pause(16) // hold tonen
0260 gate(stemme,0) // kling ud
0270 ENDPROC spil
0280
0290 PROC pause(sek'32)
0300 TIME 0
0310 WHILE TIME<1.875*sek'32 DO NULL
0320 ENDPROC pause

```

```

0010 // musiklektion 2
0020 DIM node$ OF 3
0030 USE sound
0040
0050 LOOP
0060 PAGE
0070 PRINT "Indtast en tone (a2,b5,c4,...)"
0080 PRINT "De 3 stemmer spilles efter hinanden"
0090 PRINT AT 22,1: "LEKTION 2: 3 stemmer"
0100 PRINT AT 20,1: "Tast <RUN/STOP> for at slutte.."
0110 PRINT
0120

```

```

0130 FOR stemme:=1 TO 3 DO
0140   soundtype(stemme,3)
0150 ENDFOR stemme
0160
0170 INPUT AT 7,1: "Tonevalg: ": node$
0180
0190 FOR stemme:=1 TO 3 DO
0200   PRINT AT 10,1: "Stemme ";stemme
0210   spil(stemme,node$)
0220   spil(stemme,"z")
0230 ENDFOR stemme
0240
0250 ENDLOOP
0260
0270 PROC spil(stemme,node$)
0280 IF node$() "z" THEN
0290   note(stemme,node$)
0300   gate(stemme,1) // stig og fald
0310 ENDIF
0320 pause(8)
0330 gate(stemme,0) // kling ud
0340 ENDPROC spil
0350
0360 PROC pause(sek'32)
0370 TIME 0
0380 WHILE TIME<1.875*sek'32 DO NULL
0390 ENDPROC pause

0010 // musiklektion 3
0020 DIM node$ OF 2, svar$ OF 5
0030 USE sound
0040
0050 LOOP
0060 PAGE
0070 PRINT "Lad os spille nogle toner"
0080 PRINT "og lave en enkel melodi..."
0090 PRINT AT 22,1: "LEKTION 3: Vi spiller en melodi"
0100
0110 FOR stemme:=1 TO 3 DO
0120   soundtype(stemme,3)
0130 ENDFOR stemme
0140
0150 INPUT AT 4,1: "begynd eller slut (b/s)? ": svar$
0160 IF svar$="s" THEN STOP
0170 INPUT AT 6,1: "stemme (1/2/3)? ": stemme
0180
0190 spil'melodi
0200
0210 ENDLOOP
0220
0230 PROC spil(stemme,node$)
0240 IF node$() "z" THEN
0250   note(stemme,node$)
0260   gate(stemme,1) // stig og fald
0270 ENDIF
0280 pause(tid)
0290 gate(stemme,0) // kling ud
0300 ENDPROC spil
0310
0320
0330 PROC spil'melodi // Ro, ro, ro din baad..

```

```

0340
0350 melodien:
0360 DATA "c4",8,"z",2,"c4",8,"z",2,"c4",8,"d4",4
0370 DATA "e4",8,"z",8,"e4",8,"d4",4,"e4",8
0380 DATA "f4",4,"g4",16,"z",8,"c5",4
0390 DATA "c5",4,"c5",4,"g4",4,"g4",4
0400 DATA "g4",4,"e4",4,"e4",4,"e4",4
0410 DATA "c4",4,"c4",4,"c4",4,"z",8,"g4",8
0420 DATA "f4",4,"e4",8,"d4",4,"c4",8
0430
0440 RESTORE melodien
0450 WHILE NOT EOD DO
0460   READ node$,tid
0470   spil(stemme,node$)
0480 ENDWHILE
0490
0500 ENDPROC spil'melodi
0510
0520 PROC pause(sek'32)
0530 TIME 0
0540 WHILE TIME<1.875*sek'32 DO NULL
0550 ENDPROC pause

0010 // musiklektion 4
0020 DIM node$ OF 2
0030 USE sound
0040
0050 LOOP
0060
0070 PAGE
0080 PRINT AT 22,1: "LEKTION 4: Lydstyrke, lydtype og form"
0090 PRINT AT 1,1: "Lydstyrken bestemmes for hver stemme,"
0100 PRINT "og man kan frit bestemme"
0110 PRINT "hvad slags lyd, hver stemme laver"
0120 PRINT "Det bestemmer parametrene i SOUNDTYPE"
0130 PRINT "og i ADSR proceduren"
0140 PRINT
0150 PRINT "Valgene har indflydelse,"
0160 PRINT "indtil parametrene omdefineres"
0170
0180 INPUT AT 11,1: "STEMME (1/2/3)? ": stemme
0190 INPUT AT 13,1: "STYRKE (0-15)? ": styrke
0200 INPUT AT 15,1: "LYDTYPE (1/2/3/4)? ": type
0210 soundtype(stemme,type)
0220 volume(styrke)
0230 PAGE
0240 PRINT "Du har valgt stemme";stemme;"og lydtype";type;". "
0250 PRINT "Lydstyrken er";styrke;". "
0260 PRINT
0270 PRINT "-----"
0280 PRINT "ADSR parametrene: attack, decay,"
0290 PRINT "sustain og release, bestemmer"
0300 PRINT "stemmens anslags- og udklingningstid:"
0310 PRINT
0320 PRINT " * "
0330 PRINT " * * "
0340 PRINT " * ***** "
0350 PRINT " * "
0360 PRINT " * "
0370 PRINT " A D S R "
0380 PRINT

```



```

0390 PRINT "A: anslagstid      D: faldtid"
0400 PRINT "S: opretholdt niveau R: efterklangstid"
0410 PRINT "-----"
0420 INPUT AT 21,1: "A,D,S,R? ": a,d,s,r
0430 adsr(stemme,a,d,s,r)
0440
0450 spil'melodi
0460
0470 ENDLOOP
0480
0490 PROC spil(stemme,node$)
0500 IF node$(">z ") THEN
0510   note(stemme,node$)
0520   gate(stemme,1) // anslags- og faldtid
0530 ENDIF
0540 pause(tid)
0550 gate(stemme,0) // efterklangstid
0560 ENDPROC spil
0570
0580 PROC spil'melodi // Ro, ro, ro din baad
0590 melodien:
0600 DATA "c4",8,"z ",2,"c4",8,"z ",2,"c4",8,"d4",4
0610 DATA "e4",8,"z ",8,"e4",8,"d4",4,"e4",8
0620 DATA "f4",4,"g4",16,"z",8,"c5",4
0630 DATA "c5",4,"c5",4,"g4",4,"g4",4
0640 DATA "g4",4,"e4",4,"e4",4,"e4",4
0650 DATA "c4",4,"c4",4,"c4",4,"z",8,"g4",8
0660 DATA "f4",4,"e4",8,"d4",4,"c4",8
0670
0680 RESTORE melodien
0690 WHILE NOT EOD DO
0700   READ node$,tid
0710   spil(stemme,node$)
0720 ENDWHILE
0730
0740 ENDPROC spil'melodi
0750
0760 PROC pause(sek'32)
0770 TIME 0
0780 WHILE TIME<1.875*sek'32 DO NULL
0790 ENDPROC pause

0010 // musiklektion 5
0020 DIM node$ OF 3
0030 DIM tone$(50), ads'pause$(50), r'pause$(50)
0040 USE sound
0050 volume(15)
0060 soundtype(1,2)
0070 adsr(1,6,6,8,6)
0080
0090 nr:=0
0100 WHILE NOT EOD DO
0110   nr:=1
0120   READ node$,tid
0130   tone$(nr):=frequency(node$)
0140   ads'pause$(nr):=tid*2
0150   r'pause$(nr):=tid*2
0160 ENDWHILE
0170
0180 tone$(nr+1):=0
0190 setscore(1,tone$( ),ads'pause$( ),r'pause$( ))

```

```

0200 playscore(1,0,0)
0210
0220 tal:=0
0230 WHILE NOT waitscore(1,0,0) DO
0240   tal:=1
0250   PRINT tal;
0260 ENDWHILE
0270 END
0280
0290 PROC pause(sek'32)
0300 TIME 0
0310 WHILE TIME<1.875*sek'32 DO NULL
0320 ENDPROC pause
0330
0340 DATA "c4",8,"c4",8,"c4",8,"d4",4
0350 DATA "e4",8,"e4",8,"d4",4,"e4",8
0360 DATA "f4",4,"g4",16,"c5",4
0370 DATA "c5",4,"c5",4,"g4",4,"g4",4
0380 DATA "g4",4,"e4",4,"e4",4,"e4",4
0390 DATA "c4",4,"c4",4,"c4",4,"g4",8
0400 DATA "f4",4,"e4",8,"d4",4,"c4",8

```

### SPRITEEDITOR

SPRITEEDITOR er på COMAL demonstrationsdisketten.

Dette program kan benyttes til at lave sprite-tegninger. En tegning, som er lavet og gemt på diskette med dette program, kan senere hentes ind i et andet program med ordren:

**loadshape(<tegningnr>,<filnavn>)**

SPRITEEDITOR programmet begynder med at vise denne side:



Hver af de runde prikker svarer til en prik på skærmen. Man flytter rundt mellem prikkerne ved hjælp af markør-pilene, og afmærker de prikker, som skal have en anden farve end baggrunden.

Man vælger fra menuen, som vises til højre på skærmen. Hvis der er brug for **uddybning**, trykkes på **H**, og nedenstående hjælpeside dukker op med forklaring til, hvilke taster man kan trykke på:

**Dansk uddybende version af hjælpeiden:**

O: sæt baggrundsfarve ved markør  
 1: sæt farve 1 ved markøren  
 X: udvid/udvid ikke spriten i x-retningen  
 Y: udvid/udvid ikke spriten i y-retningen  
 (forstørrelsen slås til/fra for hvert tryk)

L: indlæs tegning fra diskette  
 (hvis man vil ændre på en gemt tegning)

S: gem tegning på diskette  
 (svar derefter med filnavn og <RETURN>)

<HOME>: markør til 1. linie, 1. kolonne  
 <CLR>: slet tegningen på skærmen  
 CTRL-0: vælg ny baggrundsfarve  
 (svar derefter med farvenr og <RETURN>)

CTRL-1: vælg ny forgrundsfarve  
 (svar derefter med farvenr og <RETURN>)

Q: afslut programmet  
 M: slå flerfarvegrafik til og fra  
 (skifter ved hvert tryk)

brug markørpilene til at flytte markøren

kun til brug ved flerfarvegrafik

2: sæt farve 2 ved markøren  
 3: sæt farve 3 ved markøren  
 CTRL-2: vælg farve 2  
 (svar derefter med farvenr og <RETURN>)

CTRL-3: vælg farve 3  
 (svar derefter med farvenr og <RETURN>)

**Sekventielle filer:**

```

0010 // save "@adr. liste.demo"
0020 DIM svar$ OF 1, navn$(100) OF 40
0030 DIM gade$(100) OF 40, by$(100) OF 40
0040 DIM telefon$(100) OF 20, flag$ OF 40
0050 DIM led'efter$ OF 40, streng$ OF 150
0060 nummer:=0 // antal dataposter
0070 PAGE
0080 PRINT "Dette program skal vise brugen af"
0090 PRINT "SEKVENTIELLE FILER. Det kan benyttes"
0100 PRINT "til at lave en liste over navne,"
0110 PRINT "adresser og telefonnumre."
0120 PRINT "Hver oplysning har formatet:"
0130 PRINT
0140 PRINT "      navn"
0150 PRINT "      gade"
0160 PRINT "      by"
0170 PRINT "      telefonnummer"
0180 PRINT
0190 PRINT
0200 PRINT "Tryk en eller anden taste..."
0210
0220 afvent'taste
0230
0240 LOOP
0250   vis'menu
0260   flag$:=""
0270   afvent'taste
0280   CASE svar$ OF
0290     WHEN "1"
0300       hent'fil
0310     WHEN "2"
0320       lav'post
0330     WHEN "3"
0340       list'fil
0350     WHEN "4"
0360       led'i'fil
0370     WHEN "5"
0380       sorter'fil
0390     WHEN "6"
0400       udskift'post
0410     WHEN "7"
0420       slet'post
0430     WHEN "8"
0440       gem'fil
0450     OTHERWISE
0460       PRINT "Ulovligt svar.."
0470       afvent'taste
0480   ENDCASE
0490 ENDLOOP
0500
0510 PROC vis'menu
0520   PAGE
0530   PRINT "----- HOVED MENU -----"
0540   PRINT
0550   PRINT
0560   PRINT "      (1) HENT filen"
0570   PRINT "      (2) LAV en post"
0580   PRINT "      (3) LIST filen"
0590   PRINT "      (4) LED I filen"
0600   PRINT "      (5) SORTER alfabetisk"

```



```

0610 PRINT "      (6) UDSKIFT en post"
0620 PRINT "      (7) SLET en post"
0630 PRINT "      (8) GEM den nye fil"
0640 PRINT
0650 PRINT
0660 PRINT "Antal poster: ";nummer
0670 IF nummer=0 THEN flag$:="HENT eller LAV en fil..."
0680 PRINT
0690 PRINT flag$
0700 ENDPROC vis'menu
0710
0720 PROC hent'fil
0730 OPEN FILE 1,"adresser",READ
0740 INPUT FILE 1: nummer
0750 FOR nr:=1 TO nummer DO
0760     INPUT FILE 1: navn$(nr)
0770     INPUT FILE 1: gade$(nr)
0780     INPUT FILE 1: by$(nr)
0790     INPUT FILE 1: telefon$(nr)
0800 ENDFOR nr
0810 CLOSE FILE 1
0820 ENDPROC hent'fil
0830
0840 PROC lav'post
0850 PAGE
0860 PRINT "::::: LAV EN NY POST :::::"
0870 PRINT
0880 PRINT
0890 IF nummer=100 THEN flag$:="Ikke mere plads til data"
0900 IF flag$="" THEN
0910     nummer:=1
0920     INPUT "Navn      ": navn$(nummer)
0930     INPUT "Gade      ": gade$(nummer)
0940     INPUT "By        ": by$(nummer)
0950     INPUT "Telefon  ": telefon$(nummer)
0960 ENDIF
0970 ENDPROC lav'post
0980
0990 PROC list'fil
1000 PAGE
1010 PRINT "::::: UDSKRIFT AF FILEN :::::"
1020 PRINT
1030 IF nummer=0 THEN
1040     flag$:="Der er ingen filer!"
1050     PRINT
1060 ELSE
1070     FOR nr:=1 TO nummer DO skriv'post(nr)
1080 ENDIF
1090 ENDPROC list'fil
1100
1110 PROC led'i'fil
1120 PAGE
1130 PRINT "::::: LED I FILEN :::::"
1140 PRINT
1150 PRINT
1160 flag$:="jeg leder i filen"
1170 INPUT "Led efter ordet: ": led'after$
1180 FOR nr:=1 TO nummer DO
1190     streng$:=navn$(nr)+gade$(nr)+by$(nr)+telefon$(nr)
1200     IF led'after$ IN streng$ THEN skriv'post(nr)
1210 ENDFOR nr

```

```

1220 flag$:=""
1230 ENDPROC led'i'fil
1240
1250 PROC skriv'post(nr)
1260 PRINT
1270 PRINT AT 0,10: "-----(",nr,")"
1280 PRINT AT 0,10: navn$(nr)
1290 PRINT AT 0,10: gade$(nr)
1300 PRINT AT 0,10: by$(nr)
1310 PRINT AT 0,10: telefon$(nr)
1320 PRINT
1330 afvent'taste
1340 ENDPROC skriv'post
1350
1360 PROC sorter'fil
1370 PAGE
1380 PRINT "::::: SORTER ALFABETISK:::::"
1390 PRINT
1400 PRINT
1410
1420 PROC ombyt(REF a$,REF b$) CLOSED
1430     c$:=a$; a$:=b$; b$:=c$
1440 ENDPROC ombyt
1450
1460 REPEAT
1470     ingen'ombytning:=TRUE
1480     FOR nr:=1 TO nummer-1 DO
1490         PRINT AT 10,1: "Sortering... ",nr
1500         IF navn$(nr+1) < navn$(nr) THEN
1510             ombyt(navn$(nr),navn$(nr+1))
1520             ombyt(gade$(nr),gade$(nr+1))
1530             ombyt(by$(nr),by$(nr+1))
1540             ombyt(telefon$(nr),telefon$(nr+1))
1550             ingen'ombytning:=FALSE
1560         ENDIF
1570     ENDFOR nr
1580 UNTIL ingen'ombytning
1590 ENDPROC sorter'fil
1600
1610 PROC udskift'post
1620 PAGE
1630 PRINT "::::: UDSKIFT EN POST :::::"
1640 PRINT
1650 PRINT
1660 INPUT "Hvilken post ? ": nr
1670 IF nr<=nummer THEN
1680     skriv'post(nr)
1690     INPUT AT 14,1: "Er det den rigtige (j/n)? ": svar$
1700     PRINT
1710     PRINT
1720     IF svar$ IN "jJ" THEN
1730         INPUT "Navn:      ": navn$(nr)
1740         INPUT "Gade:      ": gade$(nr)
1750         INPUT "By:        ": by$(nr)
1760         INPUT "Telefon:  ": telefon$(nr)
1770     ENDIF
1780 ELSE
1790     flag$:="Der er kun "+STR$(nummer)+" poster"
1800 ENDIF
1810 ENDPROC udskift'post
1820

```



```

1830 PROC slet'post
1840 PAGE
1850 PRINT "::::: SLET EN POST :::::"
1860 PRINT
1870 PRINT
1880 INPUT "Hvilken post ? ": post
1890 IF post>nummer THEN
1900   flag$:="Benyt et mindre nummer!"
1910 ELSE
1920   skriv'post(post)
1930   PRINT
1940   INPUT "Er det den rigtige (j/n)? ": svar$
1950   PRINT
1960   IF svar$ IN "jJ" THEN
1970     FOR nr:=post TO nummer-1 DO
1980       navn$(nr):=navn$(nr+1)
1990       gade$(nr):=gade$(nr+1)
2000       by$(nr):=by$(nr+1)
2010       telefon$(nr):=telefon$(nr+1)
2020     ENDFOR nr
2030     nummer:=1
2040   ENDIF
2050 ENDIF
2060 ENDPROC slet'post
2070
2080 PROC gem'fil
2090 PAGE
2100 PRINT "::::: GEM FILEN :::::"
2110 OPEN FILE 1,"@adresser",WRITE
2120 PRINT FILE 1: STR$(nummer)
2130 PRINT
2140 PRINT
2150 FOR nr:=1 TO nummer DO
2160   PRINT FILE 1: navn$(nr)
2170   PRINT FILE 1: gade$(nr)
2180   PRINT FILE 1: by$(nr)
2190   PRINT FILE 1: telefon$(nr)
2200 ENDFOR nr
2210 CLOSE FILE 1
2220 ENDPROC gem'fil
2230
2240 PROC afvent'taste
2250 PRINT
2260 PRINT "< >...";
2270 REPEAT
2280   svar$:=KEY$
2290 UNTIL svar$()CHR$(0)
2300 PRINT AT 0,2: svar$
2310 ENDPROC afvent'taste

```

### Plotter demo program:

```

0010 // save "plotter demo"
0020
0030 DIM sc$ OF 1
0040
0050 setup'plotter
0060 //
0070 // MAIN PROGRAM
0080 //
0090 demo'size
0100 demo'color
0110 demo'case
0120 demo'rotation
0130
0140 square(100)
0150 blankline(2)
0160 dotlines(15)
0170 blank'line(8)
0180 circle(240,240,200)
0190 blank'line(14)
0200 spinsquares(150)
0210
0220 setup'plotter
0230
0240 END // MAIN PROGRAM
0250
0260
0270
0280 PROC demo'size
0290   FOR i:=0 TO 3 DO
0300     select'size(i)
0310     print'hello
0320     blank'line(1)
0330   ENDFOR i
0340 ENDPROC demo'size
0350
0360 PROC demo'color
0370   select'size(2)
0380   FOR i:=0 TO 3 DO
0390     switch'color(i)
0400     print'hello
0410   ENDFOR i
0420 ENDPROC demo'color
0430
0440 PROC demo'case
0450   blank'line(1)
0460   select'case(0) // upper case
0470   print'hello
0480   select'case(1) // lower case
0490   print'hello
0500 ENDPROC demo'case
0510
0520 PROC demo'rotation
0530   blank'line(2)
0540   rot'char(1)
0550   print'hello
0560   rot'char(0)
0570   print'hello
0580 ENDPROC demo'rotation
0590
0600 PROC dotlines(n)

```

```

0610 zero'pen("h")
0620 FOR i:=0 TO n DO
0630     plot("m",0,-i*20)
0640     dot'line(i)
0650     plot("d",400,-i*20)
0660 ENDFOR i
0670 blank'line(4)
0680 dot'line(0)
0690 ENDPROC dotlines
0700
0710 PROC circle(x0,y0,radius)
0720     plot("m",x0,y0)
0730     zero'pen("i")
0740     plot("r",radius,0)
0750     FOR v:=0 TO 360 STEP 5 DO
0760         t:=PI*v/180
0770         x:=radius*COS(t)
0780         y:=radius*SIN(t)
0790         plot("j",x,y)
0800     ENDFOR v
0810     blank'line(4)
0820 ENDPROC circle
0830
0840 PROC square(side)
0850     blank'line(3)
0860     plot("j",0,side)
0870     plot("j",side,side)
0880     plot("j",side,0)
0890     plot("j",0,0)
0900 ENDPROC square
0910
0920 PROC spinsquares(s)
0930     plot("m",240,240)
0940     zero'pen("i")
0950     FOR v:=0 TO 360 STEP 20 DO
0960         t:=PI*v/180
0970         draw'box(s,t)
0980     ENDFOR v
0990     blank'line(4)
1000 ENDPROC spinsquares
1010
1020 PROC draw'box(s,t)
1030     plot("j",s*COS(t),s*SIN(t))
1040     plot("j",s*SQR(2)*COS(t+PI/4),s*SQR(2)*SIN(t+PI/4))
1050     plot("j",s*COS(t+PI/2),s*SIN(t+PI/2))
1060     plot("j",0,0)
1070 ENDPROC draw'box
1080
1090 PROC blank'line(bl)
1100     plotter'on
1110     FOR i:=1 TO bl DO
1120         PRINT FILE 6:
1130     ENDFOR i
1140     plotter'off
1150 ENDPROC blank'line
1160
1170 PROC print'hello
1180     plotter'on
1190     PRINT FILE 6: "HELLO!"
1200     plotter'off
1210 ENDPROC print'hello

```

```

1220
1230
1240 // PLOTTER PROCEDURES
1250
1260 PROC plotter'on
1270     OPEN FILE 6,"u6:",WRITE
1280 ENDPROC plotter'on
1290
1300 PROC plotter'off
1310     CLOSE FILE 6
1320 ENDPROC plotter'off
1330
1340 PROC switch'color(pen)
1350     talk("2",STR$(pen))
1360 ENDPROC switch'color
1370
1380 PROC select'size(size)
1390     talk("3",STR$(size))
1400 ENDPROC select'size
1410
1420 PROC select'ascii
1430     talk("0","")
1440 ENDPROC select'ascii
1450
1460 PROC plot(sc$,x,y)
1470     talk("1",sc$+" "+STR$(x)+" "+STR$(y))
1480 ENDPROC plot
1490
1500 PROC zero'pen(zp$)
1510     // zp$ = h/i for abs/relative
1520     talk("1",zp$)
1530 ENDPROC zero'pen
1540
1550 PROC rot'char(rot)
1560     // rot=0/1 for hor/rot 90 deg CW
1570     talk("4",STR$(rot))
1580 ENDPROC rot'char
1590
1600 PROC dot'line(dash)
1610     // dash=0 to 15, 0 = unbroken
1620     talk("5",STR$(dash))
1630 ENDPROC dot'line
1640
1650 PROC select'case(nr)
1660     // nr=0/1 for upper/lower case
1670     talk("6",STR$(nr))
1680 ENDPROC select'case
1690
1700 PROC reset'plotter
1710     talk(7,"")
1720 ENDPROC reset'plotter
1730
1740 PROC setup'plotter
1750     select'case(1) // lower case
1760     switch'color(1) // blue
1770     rot'char(0) // horizontal
1780     dot'line(0) // unbroken
1790     select'size(1) // normal
1800 ENDPROC setup'plotter
1810
1820 PROC talk(sa$,text$)

```



```

1830 OPEN FILE 100,"u6:/s"+sa$,WRITE
1840 PRINT FILE 100: text$
1850 CLOSE FILE 100
1860 ENDPROC talk

```

### Brug af paralleporten:

```

0010 //save "@tog demo"
0020
0030 PAGE
0040 PRINT AT 2,2: "ELEKTRISK TOG DEMO"
0050 PRINT AT 4,2: "Toget skal starte ved stationen"
0060 PRINT AT 5,2: "med passagelyset lige bag den"
0070 PRINT AT 6,2: "sidste vogn. Start toget,og tryk"
0080 PRINT AT 7,2: "derefter en taste ned for at"
0090 PRINT AT 8,2: "overdrage kontrollen til computeren.."
0100 WHILE KEY$=CHR$(0) DO NULL
0110 PAGE
0120 PRINT AT 2,2: "ELEKTRISK TOG DEMO"
0130
0140 // Port B bit 0 forbindes til en fototransistors collector,
0150 // emitteren jordforbindes.
0160 // Bit 0 er 0 volt, mens fototransistoren belyses
0170 // Port B bit 1 forbindes til en transistor og et relæ.
0180 // Derved vil bit 1 lig +5 volt starte toget.
0190
0200 // Hovedprogram
0210
0220 definer'variabler
0230 angiv'portb
0240 start'tog
0250 udskrift
0260
0270 REPEAT
0280   check'lys
0290   pause(1.5)
0300   stop'tog
0310   pause(10)
0320   start'tog
0330 UNTIL KEY$() ""
0340 stop'tog
0350 PAGE
0360 END "Det var den tur"
0370
0380 // Alle procedurerne
0390
0400 PROC udskrift
0410   PRINT AT 12,4: "toget er i gang"
0420   PRINT AT 13,4: "toget passerer lyset"
0430   PRINT AT 14,4: "toget venter ved stationen"
0440   PRINT AT 18,4: "Hvis en taste trykkes ned,"
0450   PRINT AT 19,4: "standser toget ved stationen...."
0460 ENDPROC udskrift
0470
0480 PROC start'tog
0490   POKE portb,PEEK(portb) BITOR 2
0500   fremryk'pegepind
0510 ENDPROC start'tog
0520
0530 PROC check'lys
0540   REPEAT
0550     UNTIL PEEK(portb) BITAND 1=1
0560     fremryk'pegepind
0570 ENDPROC check'lys
0580
0590 PROC pause(sec)
0600   TIME 0

```

```

0610 WHILE TIME(sec*60 DO NULL
0620 ENDPROC pause
0630
0640 PROC stop'tog
0650 POKE portb, PEEK(portb) BITAND 253
0660 fremryk'pegepind
0670 ENDPROC stop'tog
0680
0690 PROC definer'variabler
0700 portb:=$dd01
0710 portb' ddr:=$dd03
0720 position:=1
0730 ENDPROC definer'variabler
0740
0750 PROC angiv'portb
0760 POKE portb' ddr, 2
0770 POKE portb, 2
0780 ENDPROC angiv'portb
0790
0800 PROC fremryk'pegepind
0810 PRINT AT 10+position, 2: " "
0820 IF position<4 THEN
0830 position:=position+1
0840 ELSE
0850 position:=2
0860 ENDIF
0870 PRINT AT 10+position, 2: ">"
0880 ENDPROC fremryk'pegepind

```

## Stikord

### A

ABS 133  
 accuracy 205  
 adresseliste 224  
 ADSR 186, 192  
 algoritme 79  
 anbring i lager 146  
 anbring tegn 215, 218  
 AND 140  
 AND THEN 141  
 animate 171, 180  
 arc 159  
 arcl 159  
 arcr 159  
 arcus tangensfunktion 135  
 aritmetiske operatorer 291  
 ASCII tegn 285  
 assembler 263  
 assemblersprog 23  
 ATN 135  
 attack 189  
 attributer 212  
 AUTO 33, 95

### B

back 161  
 background 157  
 bank-switching 268  
 Basic 9, 106  
 bell 210  
 betingelsessætninger 117  
 betinget udførelse 58  
 bit 23  
 BITAND 142  
 BITOR 143  
 BITXOR 144  
 boblesortering 84, 230  
 bogstaver 18  
 border 157  
 brugerport 247  
 byte 23

### C

CASE-OF-WHEN-OTHERWISE-  
 ENDCASE 119  
 CASE-strukturen 61  
 CAT 99

CHAIN 103  
 CHANGE 96  
 CHR\$ 135  
 Christensen, Børge 10  
 circle 158  
 cirkelbue 159  
 clear 156  
 clearscreen 156  
 CLOSE 116  
 CLOSE FILE 116  
 CLOSED 129  
 CLR/HOME 18, 296  
 COMAL filer 219  
 COMAL Users' Group 284  
 Commodore 10  
 Commodore-tasten 18, 295  
 CON 103  
 COPY 104  
 COS 134  
 CREATE 113 <sup>135</sup>  
 CRSR 18, 296  
 cs: 239  
 CTRL 295, 297  
 curcol 209  
 currow 209  
 CURSOR 110

### D

dansk 19, 150, 303  
 DATA 111  
 datacollision 182  
 dataretningsregister 249  
 data slut 112  
 Datassette 13, 15, 19, 40  
 datastrøm 88  
 DB-25 248  
 DDR 249  
 decay 189  
 define 176  
 definér taste 210  
 definér tegning 176  
 defkey 210  
 DEL 97  
 delay 202, 205  
 DELETE 105  
 deltekst 300  
 demoprogram 19  
 digitaltermometer 259  
 DIM 145  
 dimensionering 55  
 DIR 99  
 direkte tilgang 230, 232  
 DISCARD 49, 107  
 discard (pakker) 29

diskette, ny 21  
 diskettestation 15, 21, 42  
 diskettestation, ekstra 117  
 DISPLAY 101  
 DIV 139  
 DO 122  
 DOS fejl 313  
 draw 158  
 drawto 157  
 ds: 239  
 dynamisk lighedstegn 47

**E**

EDIT 96, 96  
 eksterne procedurer 86  
 ELIF 117  
 ELSE 117  
 eller 141  
 ellers 61, 119  
 ENDIF 117  
 enhed 116  
 END 37, 147  
 ENDCASE 119  
 ENDFOR 112  
 ENDFUNC 131  
 ENDLOOP 91  
 ENDPROC 126  
 ENDTRAP 90  
 ENDWHILE 121  
 english 150, 303  
 enhedsangivelse 239  
 ENTER 101  
 env3 196, 196  
 EOD 112  
 EOF 116  
 EPROM-udvidelse 268  
 erklæringssætning 55  
 ERR 124  
 ERRFILE 124  
 ERRTEXT\$ 124  
 ESC 139  
 Etikette: 112  
 EXIT 91  
 EXIT WHEN 91  
 EXP 135  
 EXTERNAL 130

**F**

FALSE 137  
 falsk 137  
 farveinformation 157  
 farver 289  
 fejl, disk 313

fejl, DOS 313  
 fejl, run-time 305  
 fejlgivning 277  
 fejlhåndtering 90, 123  
 fejlnumre 303  
 fejlrapport 125  
 fejlttekster 303  
 fil 87, 219  
 fil, indlæs 101  
 fil, kopier 104  
 fil, luk 116, 129  
 fil, skab 113  
 fil, skriv til 115  
 fil, slet 105  
 filafslutningstegn 256  
 filer 253  
 filflytning 236  
 filindlæsning 101  
 fill 159  
 filoverførsel 252  
 filter 195  
 filterfreq 195  
 filtertype 195  
 filtyper 237  
 FIND 96, 96  
 flerfarvebilleder 165  
 flyt pennen 158  
 flyt pen til 158  
 flytsprite 178  
 font 213  
 forskydning af lyspen origo 203  
 forstør sprite 183  
 FOR-TO-STEP-DO-ENDFOR 122  
 forgreninger 58  
 forgreningsblokke 44  
 formattering 22  
 fortsæt program 103  
 forward 161  
 FREE 210  
 fremad 161  
 frequency 189, 194  
 fullscreen 156  
 FUNC 131  
 funktioner 131, 265  
 funktionshoved 271  
 funktionstaster 296  
 funktionstaster, vis definitioner 211

**G**

gangetegn 291  
 gate 186, 192  
 gem program 102  
 gem skærmbillede 165  
 gem tegning 170, 183

gem tegnsæt 217  
 gentag 120  
 gentagelser 35, 64  
 gentagelsessætninger 120  
 GET\$ 219, 252  
 getcharacter 214, 217  
 getcolor 157  
 getscreen 208  
 gettime\$ 208  
 globale navne 75  
 GOTO 126  
 grafik 150  
 grafik oversigt 153  
 grafikmarkør 29  
 grafikskærmen 151  
 grafiktegn 285  
 graphicscreen 155  
 Grysberg, Jens 263

**H**

HANDLER 90  
 handlingsblokke 44  
 hardcopy 209  
 heading 161  
 hent fra lager 146  
 hent program 102  
 hent skærmbillede 165  
 hent tegn 214, 217  
 hent tegn fra fil 219  
 hent tegning 170, 184  
 hent tegnsæt 214, 216  
 hidesprite 181  
 hideturtle 160  
 hjem 160  
 home 160  
 hovedprogram 131  
 hukommelse 266  
 hvis 117  
 højniveausprog 24  
 højopløsningsbillede 165  
 højre 161

**I**

I/O port 248  
 identify 176  
 IF-THEN-ELIF-ELSE-ENDIF 117  
 IMPORT 129  
 IN 142  
 indeks 69  
 indflette 101  
 indicerede variable 69  
 initialiser diskette 112  
 indlæs data 48, 107

indtil 120  
 inkey\$ 207  
 INPUT 48, 107  
 INPUT AT 108  
 INPUT FILE 114  
 input/output fejlmeldinger 312  
 inq 164  
 INST/DEL 18, 296  
 INT 133  
 interrupt 179, 197, 266

**J**

Jensen, Jens Erik 11  
 jiffy 137  
 joysticks 199

**K**

kapslen 268  
 kapslen, montering 15  
 karakterkoder 285  
 kartoteksprogram 223  
 kassettefiler 89  
 katalog 99  
 kb: 239  
 keepfont 217  
 kernel 268  
 KEY\$ 108  
 keywords'in'uppercase 206  
 Kjær, Mogens 11  
 klokken 37  
 Knight, Jesse 284  
 koder, tegn 285  
 kommandoer 26  
 kommentarer 36  
 kontroller program 105  
 kontrolporte 197, 258  
 kontroltasten 297  
 kopier fil 104  
 kvadratrod 134  
 kæde af programmer 103  
 kør programmet 103

**L**

lagerstyring 268  
 lagerstørrelse 99  
 lagr tegning 170  
 lagring 40  
 Lassen, Helge 11  
 Laursen, Lars 11  
 ledig lagerplads 210  
 left 161  
 LEN 137

lightpen 201  
 lightpen, oversigt 204  
 linienummer 33  
 LINK 107  
 linkfont 214, 216  
 linkshape 184  
 LIST 100, 222  
 LOAD 102  
 loadfont 217, 217  
 loadscren 165  
 loadshape 170, 184  
 LOG 135  
 logaritmefunktionen 135  
 logiske operatorer 67, 140, 292  
 logiske udtryk 58  
 Logo 9  
 lokale navne 75  
 LOOP 91  
 loop-strukturen 91  
 lp: 239  
 lukkede procedurer 83, 221  
 lyd 184  
 lydkurven 189  
 lydniveau 189  
 lydstyrke 189, 191  
 lyspen 201  
 lyspen på skærm 205  
 lysstyrke 189  
 læs taste 108  
 løft pennen 161  
 Løfstedt, Benedict 10  
 løkkeblokke 44  
 løkker 64

**M**

MAIN 131  
 male med farve 160  
 markør 110, 209  
 maskinsprog 263  
 medens 65, 121  
 melodi, stands 194  
 melodi, vent på 190  
 memory management 268  
 MERGE 101  
 midtpunktsmetoden 79  
 mikroprocessor 23  
 MOD 139  
 modtag data 246  
 modtagerprogram 256  
 moduler 264, 269  
 modulus 139  
 monitor 16  
 MOUNT 112  
 move 158

movesprite 178  
 moveto 158  
 moving 179  
 MPS-801 166  
 musik 184, 319  
 musikopstart 186

**N**

names'in'uppercase 207  
 NEW 95  
 NEXT 122  
 NOT 140  
 note 186, 192, 192  
 nowrap 163  
 NULL 146  
 nulstil datapejler 111  
 numerisk værdi 133  
 nummererede variabler 69  
 nye numre 33, 95  
 nyt navn 105  
 nøgleord 36

**O**

offset 203, 204  
 og 140  
 oktet 23  
 omløb 163  
 omløb, ikke 163  
 OPEN 113  
 OPEN FILE 113  
 OR 141  
 OR ELSE 142  
 ORD 136  
 osc3 197  
 OTHERWISE 61, 119

**P**

paddles 197  
 PAGE 110  
 paint 160  
 pakke 149  
 pakkeeksempel 278  
 pakker 265  
 pakketabel 270  
 parallelport 247  
 parameteroverførsel 272  
 parameterspecifikation 271  
 paritetsbit 244  
 Partner 254  
 Pascal 9  
 pass 22, 105  
 PEEK 146

pen 29  
 pencolor 157  
 pendown 161  
 penfarve 157  
 penon 205  
 penup 161  
 PI 134  
 Piccoline 252  
 pilespids 29  
 playscore 185, 189, 193  
 plot 157  
 plottext 164  
 POKE 146  
 poster 219  
 potensopløftning 291  
 prepass scan 304  
 prg 237  
 programkontrol 105  
 PRINT 109  
 PRINT AT 109  
 PRINT FILE 114  
 PRINT USING 109  
 printer 13, 15, 166, 212  
 printer-plotter 239  
 printscreen 166, 214  
 prioritet 291  
 priority 181  
 PROC 126  
 procedure 52, 74, 126  
 procedure, lukket 221  
 procedurehoved 271  
 procedurer 24, 37, 265  
 program 31  
 programmeringsprog 45  
 programsætning 33  
 programudførelse 34  
 pulse 196  
 putcharacter 215, 218

**Q**

quote'mode 207

**R**

RAM opdeling 267  
 random filer 220, 232  
 RANDOMIZE 138  
 READ 111  
 READ FILE 115  
 readpen 205  
 redigér program 96  
 REF 128  
 REF ordren 76  
 register 219

regningsarterne 291  
 rekursiv procedure 78  
 rel 237  
 release 189  
 RENAME 105  
 renum 33, 95  
 REPEAT-UNTIL 120  
 REPORT 125  
 resonance 196  
 RESTORE 111  
 rettelser 26  
 RETURN 131, 296  
 right 161  
 ringmod 196  
 ringmodulation 196  
 RND 138  
 rodsøgning 79  
 RS-232 grænseflade 244  
 RUN 103  
 run-time fejl 305  
 RUN/STOP 295

**S**

sammenligningsoperatorer 292  
 sandt 137  
 SAVE 102  
 savefont 217  
 savescren 165  
 saveshape 170, 183  
 SCAN 36, 97  
 sekundære adresser 212  
 sekventielle filer 219, 236  
 SELECT 106  
 SELECT INPUT 105  
 SELECT OUTPUT 106  
 semicolon 27  
 senderprogram 256  
 seq 237  
 serial 211  
 SETEXEC 98  
 setfrequency 194  
 setheading 161  
 setpage 212  
 setprinter 211  
 setrecorddelay 212  
 setscore 190, 193  
 settime 207  
 setxy 158  
 SGN 133  
 SHIFT 18, 295  
 SHIFT LOCK 18  
 SHIFT/LOCK 295  
 showkeys 211  
 showsprite 181

showturtle 160  
 signaler 266  
 signallrutiner 275  
 sikkerhedskopi 41  
 SIN 134  
 sinusfunktionen 134  
 SIZE 99  
 skildpadden 29  
 skildpadden, skjul 160  
 skildpadden, vis 160  
 skildpaddestørrelse 160  
 skjul skildpadder 160  
 skjul sprite 181  
 skriv 109  
 skriv grafiktekst 164  
 skriv til fil 115  
 skærm dump 166  
 skærmredigering 295  
 skærmtegn 285  
 slet fil 105  
 slet grafikskærm 156  
 slet linier 97  
 slet pakker 29, 107  
 slet program 95  
 slet tekstskærm 110  
 sortering, boble 84, 230  
 sound 184, 319  
 soundtype 187, 192  
 sp: 239  
 SPC\$ 136  
 split screen 156  
 spor fejl 144  
 spriteback 176  
 spritecollision 182  
 spritcolor 176  
 spriteeditor 323  
 spritfarver 173  
 sprite, forstørrelse 183  
 spriteing 182  
 spriteoversigt 175  
 spritepos 177  
 sprites 167, 323  
 spritesammenstød 169  
 spritesize 177  
 SQR 134  
 stampsprite 183  
 standardfunktioner 292  
 stands melodi 194  
 stands program 147  
 stands sprite 179  
 startbit 244  
 start melodi 185, 189, 193  
 start program 103  
 startsprites 179  
 STATUS 104

STATUS\$ 104  
 STEP 122  
 stikforbindelser 14  
 stil uret 207  
 STOP 147  
 STOP/RESTORE 296  
 stopplay 194  
 stopsprite 179  
 STR\$ 136  
 streng 55  
 strengfunktioner 82  
 strenglængde 137  
 strengnavn 55  
 strengvariabel 55  
 strukturkontrol 36, 97  
 strømforsyning 17  
 størrelse, lager 99  
 sustain 189  
 SX-64 14  
 sync 194  
 syntaksfejl 311  
 SYS 106, 146, 264  
 system 206  
 sænk pennen 161  
 sætningsstruktur 117

## T

TAB 110  
 tabel 112  
 tabulering 110  
 talbehandling 291  
 talområde 291  
 talværdi fra streng 136  
 TAN 134  
 tangensfunktionen 134  
 tastaturet 18, 295  
 tegn 56  
 tegn linie 157  
 tegn punkt 157  
 tegn vektor 158  
 tegnefilm 171, 181  
 tegneramme 154  
 tegnkoder 285  
 tegnsæt, bruger 213  
 tegnsætpakken 213  
 tekstbaggrund 156  
 tekstbehandling 299  
 tekster 55  
 tekstfarve 156  
 tekstfunktioner 301  
 tekstkant 157  
 tekstskærm 155  
 tekststil 163  
 teksttabelle 71

tekstvariabler 299  
 termistor 260  
 textbackground 156  
 textborder 157  
 textcolor 156  
 textcolors 206  
 textscreen 155  
 textstyle 163  
 tilbehør 13  
 tildelingstegn 47  
 tilfældige tal 138  
 TIME 137  
 timeon 202, 205  
 tog demo 250  
 tomme sætninger 38  
 tone 190  
 total skildpaddetur 36  
 TRACE 144  
 TRAP 90  
 TRAP ESC 139  
 TRAP-HANDLER-ENDTRAP 123  
 tretrådsforbindelse 245, 253  
 trinklængde 122  
 TRUE 137  
 turtle grafik 28  
 turtle ordrer oversigt 31  
 turtlegrafik 28  
 turtlesize 160  
 TV, tilslutning 16  
 Tønder Statsseminarium 10

## U

udfyld med farve 160  
 udfør program 103  
 udskrift til printer 209  
 udskriv program 100, 222  
 udtryk 47  
 UniComal 10  
 UNIT 116  
 UNIT\$ 117, 239  
 UNTIL 120  
 uret, indstil 207  
 USE 106, 264  
 user port 248  
 usr 237  
 uX: 239

## V

VAL 136  
 varekartotek 232  
 variabel 47  
 variabeliste 39  
 vedhæft pakke 107  
 vedhæft tegning 184  
 vedhæft tegnsæt 214, 216  
 vent på melodi 190  
 VERIFY 105  
 viewport 154, 154  
 vindue 154  
 vis funktionstaster 211  
 vis klokken 37  
 vis skildpadden 160  
 volume 189, 191  
 vælg frekvens 194  
 vælg input 105  
 vælg output 106  
 vælg printer 211  
 vælg retning 161  
 vælg side 212

## W

waitscore 190, 194  
 WHILE 65  
 WHILE-DO-ENDWHILE 121  
 window 154  
 Wirth, Niklaus 10  
 wrap 163  
 WRITE FILE 115

## Y

ydre enheder 243  
 ydre procedurer 130

## Z

zone 27, 110

## A

åbn fil 113  
 Århus Universitet 10

## Specielle tegn

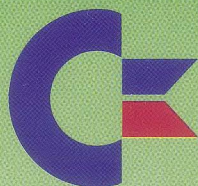
// 144





**COMAL**

for Commodore 64



**Commodore**  
Made in Denmark



**Commodore**

KAMPER BOGTRYKOFFSET HORSENS