EN DATA BECKER BOG

SZCZEPANOWSKI

AMSTRAD 664 & 6128

FØRSTE-BOGEN

En letfattelig indføring i 664/6128

DANSK / NORSK

udgave

Forlag: NCS, box 105, DK-6950 Ringkøbing

FØRSTE-BOGEN

EN DATA BECKER BOG

SZCZEPANOWSKI

AMSTRAD

664/ 6128

FØRSTE-BOGEN

En letfattelig indføring i 664/6128

DANSK / NORSK

udgave

POSTBOX 105 · DK-6950 RINGKØBING
1986

Postbox 105
Østerport
DK-6950 Ringkøbing

Distribution i Norge:
Computer Equipment A/S
Gyldenløves gt. 42
N-4600 Kristiansand S.
Tlf. (042) 70 294

Sats & tryk: Tarm Bogtryk & Offset A/S

ISBN 87-7283-004-2

VIGTIGT

Programmer, programeksempler mv. er omfattet af lov om copyright. Disse må kun anvendes til personlige- eller undervisningsformål og må ikke anvendes kommercielt.

Alle programeksempler, tekniske anvisninger osv. i denne bog er omhyggeligt gennemgået for fejl. Trods dette kan der optræde fejl i reproduktionsfasen.
Skulle nogle læsere opdage fejl, beder vi Dem rette henvendelse til os, så vi har mulighed for at foretage rettelse.

INDHOLDSREGISTER

Kapitel 1: TASTATURET

I dette kapitel ser vi nærmere på de 74 taster på din CPC6128. Ved hjælp af disse taster kan du bruge alle de forskellige muligheder der er indbygget i din computer. Emner som indsætning af tekst, fremstilling af billeder (grafik), beregning af matamatiske problemer og reaktion på fejlmeldinger bliver gennemgået i dette kapitel. Når du har lært at beherske alle taster på tastaturet, vil du kunne se at CPC6128 bliver en uvurderlig hjælp til løsningen af mange dagligdags problemer.

Kendskab til tastaturets opbygning er ikke kun af interesse for deciderede "computer-freaks", som finder fornøjelse i at kende hver enkelt bit i maskinen. Også den bruger, der udelukkende anvender kommercielle programmer vil have stor glæde af et nærmere kendskab til de enkelte tasters funktion. Brugere af kommercielle programmer vil ofte komme ud for en skærmmelding som f.eks. "TAST RETURN FOR KOPI AF SKÆRM" og et godt kendskab til tastaturet sparer mange opslag i instruktionsbogen.

Kort sagt bør alle der engang imellem betjener sig af en computer gøre sig fortrolig med tastaturet. Hvis man f.eks. har computerinteresserede børn i huset, kan det have sin værdi at vide hvordan man sætter "favoritspillet" igang når ungerne er lagt i seng — det er jo ærgeligt ikke at kunne skyde rumskibe ned, bare fordi man ikke kan finde den rigtige tast, der sætter gang i det hele.

Det er ikke nødvendigt at deltage i et aftenskolekursus i maskinskrivning for at betjene en computer. De fleste — også erfarne — programmører benytter "2-fingersystemet". Efter et stykke tids arbejde med tastaturet vil man selv undres over hvor hurtigt fingrene suser hen over tasterne.

I dette kapitel bliver de forskellige hjælpetaster udførligt beskrevet. Disse taster er meget vigtige ved eksempelvis bearbejdning af tekst, programafbrydelse og forskellige kommandoer til computeren.

Generelt om tastaturet

Ved første øjekast ligner CPC6128's tastatur et ganske almindeligt skrivemaskinetastatur. Ser du nøjere efter afslører der sig imidlertid nogle forskelle.

— der findes ikke danske bogstaver (æ, ø og å) på tasterne
— der er forskellige hjælpetaster (som vi senere ser nærmere på)

En lille advarsel: Lad være med at lege med tasterne før du har et elementært kendskab til deres virkemåde. Der kan ikke ske nogen som helst skade på computeren ved en evt. fejlbetjening, men der kan komme nogen ret overraskende ting ud af blot et enkelt forkert tastetryk. Vent altså, til vi kommer til de praktiske øvelser på næste side.

Så ta'r vi fat.

Nu er det på tide at vække den slumrende teknik i CPC6128 til live. Du starter med at tænde for computeren. Der findes 2 afbryderknapper, en på monitoren og en bag på selve computeren. Begge skal tændes. Hver gang computeren tændes kommer der en besked på skærmen der fortæller at computeren er parat til at modtage instruktioner fra dig. Den første linie i opstartmeddelelsen (se ill. 1) siger at du arbejder med en AMSTRAD computer der har 128 Kbyte (131070 tegn) lagerplads i hukommelsen. Af disse 131070 tegn er der umiddelbart ca. 40000 tegn til rådighed for dine egne BASIC programmer. Området fra 40000 op til 64000 anvendes af computeren til dens eget operativsystem (dvs. det regelsæt den anvender internt til alle aktiviteter) samt programmeringssproget BASIC. De sidste 64000 tegn kan bruges til at opbevare forskellige data i.

Den næste linie indeholder navnet på fabrikanten, AMSTRAD CONSUMER ELECTRONICS plc. i England. Fra firmaer LOCOMOTIVE SOFTWARE stammer den indbyggede BASIC som, hvad du senere vil erfare, er ganske fremragende.

Denne BASIC hedder version 1.1. Hvis der senere sker ændringer eller forbedringer, vil dette nummer ændres tilsvarende. I den sidste linie fortæller ordet "READY" at operativsystemet forventer en kommando.

Nu til den lille firkant under ordet READY. Denne kaldes i fagsproget for en "CURSOR" (udtales "kørsa"). Cursorens funktion er at markere på skærmen hvor den næste tekst vil blive skrevet. Det er altså en markør der hjælper dig med at placere tekst på skærmen på rette sted.

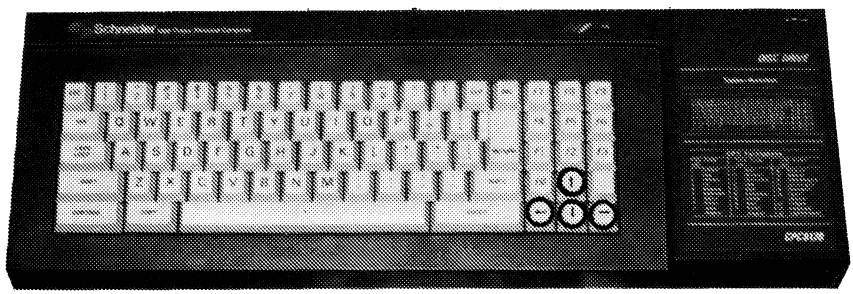
Tre-finger-tricket

Det er ofte nødvendigt at bringe computeren tilbage til opstart tilstand. Ved mange computere kan dette kun opnåes ved at slukke og derefter tænde for computeren. Det giver en overflødig belastning af eletronikken, som forkorter levetiden for de dyre komponenter. CPC6128 har indbygget en såkaldt "RESET" (også kaldet "koldstart") funktion, der bringer computeren tilbage til opstarttilstand uden at det er nødvendig at tænde og slukke. Ved at trykke på SHIFT-CLRL-ESC samtidig opnåes det ønskede. Vær opmærksom på, at ESC skal nedtrykkes til sidst, medens SHIFT-CLRL stadig er nedtrykket. Dette kaldes populært for tre-finger-tricket.

Prøv at trykke på de tre taster nu! Du vil bemærke, at efter et kort øjeblik får du det samme skærmbillede som du får i ill. 2.

Inden du bruger RESET af computeren skal du tænke dig godt om. Hvis du har et BASIC program liggende i computerens hukommelse mister du nemlig dette ved RESET!!

Cursor-tasterne



Disse taster befinder sig til højre på tastaturet, lige under tasterne F0-F9. Pilene på tasterne indikerer retningen som CURSOR'en vil bevæge sig. Prøv at trykke på HØJRE-pilen 20 gange. Cursoren flytter sig 20 tegn mod højre på skærmen og markerer at evt. tekst vil blive skrevet her.

Hvad sker der, hvis cursoren rammer højre kant af skærmbilledet? Prøv selv! Tryk 20 gange mere på højre-pilen. Ved det 20. tryk på piletasten flytter cursoren sig ned i første spalte på næste linie.

Det er imidlertid ret omstændeligt at trykke 20 gange på en tast for at nå midten af skærmen. Vi kan forenkle det hele lidt. Hvis du holder piltasten nedtrykket, vil cursoren automatisk bevæge sig mod højre. Dette gælder for alle taster på tastaturet. Prøv at holde højre-pilen nedtrykket og læg mærke til hvad der sker, når cursoren når højre skærmkant.

Hvis du vil have cursoren til venstre, trykker du blot på venste-pil-tasten. Prøv også denne tast. Cursoren bevæger sig mod venstre.

Det vil naturligvis også ske automatisk, såfremt tasten holdes nedtrykket.

Hvordan får vi nu cursoren tilbage til udgangspositionen (under "READY")? Du kunne f.eks. holde venstre-pilen nedtrykket længe nok. Du skal nok komme tilbage, men det tager tid. For at komme hurtigt op og ned pæ skærmen, bruger man op-pil-tasten eller ned-pil-tasten. Disse muliggør hurtige op/ned bevægelser.

Prøv at placere cursoren ved venste skærmkant. Derefter trykker du på nedpilen 3 gange og — bravo, bravo — cursoren har nu flyttet sig 3 linier længere ned.

Prøv nu at flytte cursoren op i 3. linie. Her bruger du op-pil-tasten. Også her foregår bevægelsen automatisk når du holder tasten trykket ned.

For at opnå en fornemmelse af cursor-bevægelserne, kan du prøve at lave en "usynlig" firkant på skærmen. Du flytter cursoren mod højre/ned/venstre/op — og modsat vej venstre/ned/højre/op. Det er nødvendigt at kunne betjene cursortasterne godt, for senere hurtigt at kunne rette i programlinier. Her gælder det gamle ordsprog: Øvelse gør mester!!

Editering er et fagudtryk der sikkert ikke siger begynderen ret meget. Det betyder kort sagt "bearbejdelse af tekst", f.eks. fremstilling af tekst og ændringer i tekst. En af de enkleste former for editering er at skrive ord på skærmen.

På mange andre computere kan man frit flytte rundt med cursoren på skærmen, skrive den tekst man har lyst til — endda oven i eksisterende tekst — og så igen flytte med cursoren. Denne måde at flytte rundt med cursoren på kaldes "FULL-SCREEN" editering. På CPC6128 kan man også frit bevæge cursoren rundt på skærmen indtil man skriver et tegn. Herefter adskiller CPC6128's editor sig fra de fleste andre. Når du har skrevet tegn på skærmen, kan cursoren kun bevæges henover de tegn der allerede er skrevet. Denne måde at editere på lad os prøve med en praktisk øvelse: Reset computeren (SHIFT-CLT-ESC). Skriv nu ordet "computerven" på skærmen. Når du har skrevet ordet, står cursoren lige bag det sidste bogstav og den kan nu flyttes over mod venstre. Prøv blot op/ned og til højre. Den lader sig ikke tvinge i andre retninger end til venstre. Cursoren kan nu kun anvendes til at bearbejde den skrevne tekst med og derfor kan den ikke flyttes ud over det beskrevne felt.

Prøv nu at flytte cursoren helt ud til venstre skærmkant. Når cursoren kommer til kanten, stopper den som var den stødt mod en mur. Det samme gør sig gældende hvis du går mod højre og prøver at passere det sidste bogstav. Hvorfor nu disse indskrænkninger i cursorbevægelserne? Der findes flere grunde, f.eks. programfremstilling. Når man skriver et program, skrives de enkelte kommandodele ind i linier. Hver programlinie afsluttes ved tryk på tasten mrkt. RETURN. Når denne aktiveres gemmes programlinien i computerens hukommelse. Det er ulogisk at fortsætte en påbegyndt programlinie et andet sted på skærmen, derfor "låses" programmøren fast til linien indtil han/hun trykker på RETURN. Herefter er cursoren givet fri, og kan igen bevæges over det hele.

Lad os komme tilbage til ordet "computerven". På et hvilket som helst sted i ordet kan der indsættes nye bogstaver. For at gøre dette, skal cursoren positioneres umiddelbart før det sted man ønsker det nye bogstav eller tegn placeret. Den såkaldte "tilføje" modus er hele tiden virksom. Vi skal nu prøve at ændre ordet "computerven" til "hjemmecomputer-ven".

Start med at flytte cursoren ud til venstre skærmkant.

Cursoren befinder sig nu over bogstavet "c". Nu skriver du bogstaverne "h", "j", "e", "m", "m", "e".

Hver gang du indtaster et bogstav, bliver teksten til højre for cursoren flyttet en plads mod højre, cursoren flytter med og er stadig over bogstav "c". Nu skal bindestregen på plads. Du gætter nok allerede hvordan dette skal ske. Placer cursoren over bogstav "v" og tryk på tasten med bindestregen (til højre for "0" i øverste række).

Herefter køres cursoren helt ud til højre (lige efter "n") og du kan fortsætte med at skrive mere tekst. Som vi før har nævnt, overgives skærmteksten først til operativsystemet i det øjeblik du trykker på "RETURN". Når du trykker på denne tast, vil du få en reaktion fra computeren. Prøv selv!

Ordene "Syntax error" betyder (oversat til menneskesprog): "Det du vil bede mig om at gøre, forstår jeg overhovedet ikke". Det er selvfølgelig klart, da det eneste sprog computeren forstår er BASIC, og det må du først lære for at arbejde med din CPC6128.

Du har nu lært at indsætte tekst. Hvis du øver dig med andre ord, vil du snart beherske det perfekt.

Sletning af skærmbillede

Prøv engang at forestille dig, at det der står på skærmen kun er ubrugeligt kaudervælsk og det skal vi have visket ud. STOP!! Ikke med tavlekluden. Med den kan du måske tørre støvet af monitoren, men du kan ikke gøre noget ved det, der allerede er skrevet, og som nu opbevares i computerens hukommelse. Heldigvis er det for en computer, der kan udføre ca. 1 million operationer pr. sekund en ren børneleg at slette de "kun" 1000 tegn der findes på skærmen.

Mange andre computere har en speciel tast beregnet til at slette skærmen med, men helt så komfortabel er CPC6128 desværre ikke. Du skal igennem 4 tastetryk for at udføre operationen. Der er i CPC6128's BASIC indbygget en kommando specielt beregnet til at slette skærmen. Denne hedder "CLS" (Clear Screen). Som du allerede har lært skal alle kommandoer til computeren afsluttes med "RETURN". Du skriver altså "CLS" efterfulgt af "RETURN". Computeren udfører kommandoen og afslutter med ordet READY, hvilket betyder at den er klar til at modtage en ny kommando. "CLS" kommandoen skal du huske, du vi ofte refererer til den i bogen.

Bogstavtasterne

Som du ser på ill. ligner tastaturet på mange måder et skrivemaskine-tastatur, ihvertfald for bogstav-tasternes vedkommende. Lad os nu koncentrere os om disse. Du starter med at slette skærmen med en "CLS" kommando. Vi har nu en "ren" skærm at arbejde på.

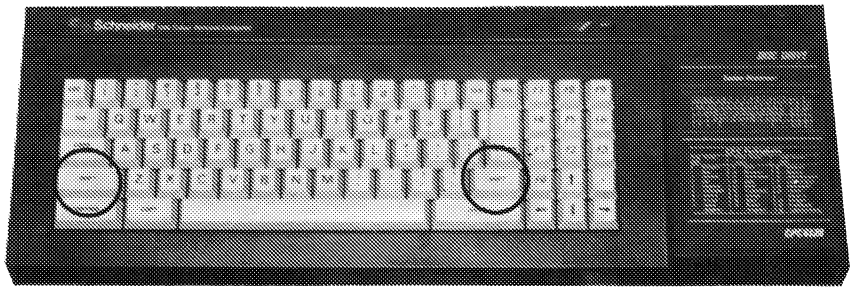
Når du trykker på en bogstav-tast, bliver det pågældende bogstav skrevet som "lille" bogstav ved cursorpositionen. Det, du skriver på skærmen bliver gemt i et lille lageområde i computeren der kaldes skærmhukommelse. Først når du trykker på "RETURN" bliver din skærmtekst videregivet til det egentlige hukommelsesområde. Det kommer vi til at beskæftige os nærmere med under afsnittet om BASIC programmering.

Prøv nu at skrive hele alfabetet på skærmen. Hvis du aldrig har skrevet på maskine, vil du nu opdage, hvor svært det er at finde de rigtige bogstaver i en fart her i begyndelsen.

Nu skulle du gerne have hele alfabetet på skærmen.

Slet nu skærmen og skriv alle bogstaverne igen — denne gang ikke i alfabetisk rækkefølge, men som de faktisk ligger på tastaturet. Altså først øverste række (Q til P), Det er sikkert langt lettere end at indtaste dem alfabetisk.

Store/små bogstaver



Du har allerede set at bogstaverne skrives på skærmen som "små" bogstaver. Ofte, og frem for alt indenfor tekstbehandling, er det ønskværdigt også at benytte "store" bogstaver. På CPC6128 hedder skiftetasten mellem store og små bogstaver "SHIFT". Denne tast er større en de fleste andre og findes to steder på tastaturet. Du trykker altså på "SHIFT" hvis du vil skrive "store" bogstaver.

Mellemrumstangenten

Mellemrumstangenten anvendes, når du har brug for at lave mellemrum mellem ord eller bogstaver. Hvis der allerede befinder sig et bogstav, der hvor du placerer et mellemrum, vil bogstavet blive slettet.

Prøv at slette skærmen og skriv den følgende sætning:

Øvelse gør mester

Brug mellemrumstangenten imellem de enkelte ord.

DEL-tasten

DEL er en forkortelse af det engelske ord DELETE, som på dansk betyder at slette. Med DEL-tasten kan du slette tegn til venstre for cursoren.

Trykfejl er ikke til at undgp

Som du sikkert har bemærket har der indsneget sig en fejl i ordet "undgå". Med et tryk på DEL-tasten, kan du nu slette det forkerte bogstav. Herefter skriver du blot det rigtige, og ingen kan se, at de nogensinde har stået andet. Dette eksempel illustrerer dog ikke den helt store fordel ved DEL-tasten.

Derfor skal vi nu se på et andet eksempel. Prøv at skrive:

Trykkfejl er ikke til at undgå

Her er der kommet 2 "k"er i trykfejl. Nu behøver du ikke at slette alt efter dit fejlslag. Du placerer blot cursoren til venstre for der forkerte bogstav og trykker på DEL. Hvis du ser godt efter, vil du opdage, at alt hvad der ligger til højre for cursoren nu har flyttet sig en plads mod venstre. Herefter er rettelsen færdig, og du kan flytte cursoren hen hvor den var, da du opdagede fejlen og fortsåtte skrivningen derfra.

CLR-tasten

Ud over DEL tasten er endnu en editeringstast på din CPC6128. Den hedder CLR og befinder sig i øverste række på tastaturet. Med CLR kan du slette det tegn, der befinder sig direkte under cursoren. Også her rykker tegnet på højre side af cursoren mod venstre. Prøv at skrive denne sætning:

CPC6128 er en rimelig god computer

Nu finder du måske ud af, at en Amstrad ikke blot er en rimelig god computer, men faktisk en udmærket computer. Vi skal derfor have fjernet ordet "rimelig" og erstatte det med "udmærket". Du kan her vælge imellem at bruge DEL og CLR. Vi bruger CLR i eksemplet. Da CLR sletter alt hvad der ligger under cursoren, skal denne placeres oven over "r" i ordet "rimelig". Du trykker nu i alt 8 gange på CLR-tasten, og ordet er slettet. Derefter — med curso-

ren på samme sted — skriver du blot ordet "udmærket" og rettelsen er foretaget.

3 forskellige bogstavsstørrelser

Som du har set i de første øvelser, er der plads til 40 bogstaver på een linie, og da der er plads til ialt 25 linier, kan der i alt rummes 1000 karakterer på skærmen af gangen. Andre computere, som f.eks. Commodore 64 kan ikke placere flere en de 40 tegn pr. linie uden kostbare udvidelsesmoduler.

Udviklingsteamet bag Amstrad computerne har valgt at gøre antallet af tegn pr. linie fleksibelt, så brugeren selv kan skifte mellem 3 forskellige skriftstørrelser, 20, 40 og 80 tegn pr. linie. Dette skift sker med en BASIC kommando, der hedder MODE. Da der er 3 tegnsæt at væge mellem, skal der ud over ordet MODE angives et tal der fortæller computeren, hvilken skriftstørrelse vi vil arbejde med.

De 3 MODE kommandoer hedder som følger:

— MODE 0 giver 20 tegn pr. linie
— MODE 1 giver 40 tegn pr. linie
— MODE 2 giver 80 tegn pr. linie

Du kan nu skrive MODE 0 og afslutte kommandoen med at trykke på RETURN. Du vil sikkert forbavses over det overdimensionerede READY, der nu kommer til syne på skærmen. Denne MODE bruges især til spil, da man i MODE 0 foruden den store skrift også kan have mange forskellige farver på skærmen af gangen. Desværre kan man ikke bruge MODE 0 til f.eks. overskrifter og så skrive selve teksten i MODE 1. Der kan kun bruges 1 skriftstørrelse på skærmen af gangen, da computeren selv udfører en CLS inden den skifter MODE.

Du kan også prøve af bemærke forskellen, når du nu skriver MODE 2. Den er virkelig stor.

Måske vil det overraske, at de 80 tegn pr. linie trods alt står så skarpt. Det gælder især for de 6128-ejere der anvender grøn (også kaldt monocrom) monitor. Skarpheden — og det faktum at den grønne farve ikke er så hård ved øjnene — er den største fordel ved at bruge monocrom monitor. Man kan, hvis man vil udnytte CPC6128's farvepagt evt. senere investere i en modulator. På denne måde kan man bruge farvefjernsynet som skærm.

Kopi-cursoren

Alle computere kan gemme tekst, programmer og data, og disse kan selvfølgelig hentes frem igen og viderebearbejdes. Programmer bliver næsten aldrig skrevet på een gang, men bliver opbygget i småbidder af flere gange. For at du kan få de bedste forudsætninger for dit første programmeringsforsøg, skal vi nu se nærmere på, hvordan man retter i en tekst, der allerede er skrevet.

Lad os antage, at du vil skrive kommandoen CLS efterfulgt af RETURN. Du kommer til at lave en fejl og skrive CLD i stedet. Det resulterer i en fejlmelding fra computeren (Syntax error).

Vi skal nu prøve at rette kommandoen, uden at taste det hele ind igen. Mange computere har som før nævnt en "Full-screen editor". I en sådan kan man direkte flytte cursoren op til det forkerte tegn, og skrive der rigtige i stedet, og derefter programmere videre. På en Amstrad er det ikke helt så nemt. Prøv bare — placer cursoren over "D"et, skriv "S" i stedet og tryk på RETURN.

Selvom der nu står den rigtige kommando på skærmen, får du stadig fejlmeldingen "Syntax error".

Grunden hertil er, at computeren opfatter den første kommando (CLD) som afsluttet, når du har trykket på RETURN. Derfor bliver din rettelse pånyopfattet som en kommando, og da computeren ikke kender kommandoen "S" reagerer den med en fejlmelding.

Hvis man arbejder med lange kommandosætninger og i disse opdager fejl, ville det være rart, om man kunne slippe for at skrive hele linien om. Det er også muligt. CPC6128 har en indbygget funktion, netop til dette formål. Funktionen, som kaldes KOPI-CURSOR, kopierer simpelthen den "gamle" linie til et nyt sted på skærmen — og her kan den rettes.

Beherskelse af denne teknik kræver en smule øvelse. Slet derfor skærmen og skriv igen "fejlkommandoen" CLD.

1. trin — CURSOR til begyndelsen af den "nye" linie.

Nu skal du bestemme hvortil den "gamle" kommandoliste skal kopieres og cursoren skal placeres på dette sted. I vort eksempel er det linie 5.

2. trin — KOPI-CURSOR til begyndelsen af "gammel" linie.

Den førnævnte cursor kender du allerede, man hvad i alverden er en KOPI-CURSOR, og hvordan bevæger man sådan en svend?

KOPI-CURSOREN er en ekstracursor med nøgagtig samme udseende som den normale cursor. Den bevæges med piltasterne og SHIFT tasten nedtrykket samtidigt. Prøv nu at trykke på SHIFT/pil-op for at flytte kopicursoren op i den "gamle" linie.

3. trin — kopier de ønskede tegn med COPY-tasten.

COPY tasten flytter kopicursoren, ligesom en almindelig cursor, mod højre og kopierer derved de tegn den passerer hen over. Dette lyder mere kompliceret end det er — prøv at trykke 2 gange på COPY-tasten.

Nu har du fået kopieret de første 2 tegn i kommandoen CLS. Det 3. bogstav skal ikke kopieres! I stedet skriver du et "S". Den gamle kommando er nu kopieret og ændret. Nu kan du trykke på RETURN og den korrekte kommando udføres. Når du trykker på RETURN forsvinder kopicursoren.

Foruden denne rettemetode findes den endnu en. Denne metode retter fejlen på det sted den er opstået, og det skal vi se nærmere på i det følgende:

1. trin — CURSOR flyttes til begyndelsen af den "gamle" linie.

Vi starter med at placere cursoren ved begyndelsen af den linie, der skal ændres.

2. trin — COPY-tasten indtil det forkerte bogstav nåes.

Nu trykker du på COPY-tasten 2 gange. Derved kopieres de to første bogstaver på samme sted hvor de i forvejen stod. Husk at du denne gang skal trykke på COPY-tasten, og ikke på højre-pilen.

Nu kan du skrive det rigtige bogstav (et "S") i stedet for det forkerte ("D") og slutte af med RETURN.

Du har nu lært de to forskellige rettemetoder at kende. Den ene er fuldt ud lige så god som den anden, og det er op til dig selv, hvilken du vil benytte fremover.

Hvis du stadig har problemer med at få KOPI-CURSOR'en til at makke ret, kan du eventuelt have glæde af det følgende eksempel:

Du begynder med at skrive følgende sætning: Kopiering sjovt er.

Vi skal nu prøve at bytte om på rækkefølgen af de tre ord ved hjælp af kopicursoren. Da sætningen ikke er en kommando computeren kan forstå, må vi tage fejlmeldingen "Syntax error" med i købet.

Den korrekte sætning vil vi gerne have på 6. linie, så du placerer cursoren ved begyndelsen af denne.

Vi kan nu kopiere det første ord. Tryk på COPY-tasten indtil kopicursoren står oveni bogstavet "s" i "sjovt". Mellemrummet skal også kopieres.

Nu er du klar til at kopiere ord nr. 2. Det skal være ordet "er". Du flytter derfor kopicursoren hen til begyndelsen af dette ord med SHIFT og højrepil-tasten. Du kopierer ordet med 2 tryk på COPY. Når det er gjort, skal du trykke en ekstra gang på COPY for at få et mellemrum i den nye linie.

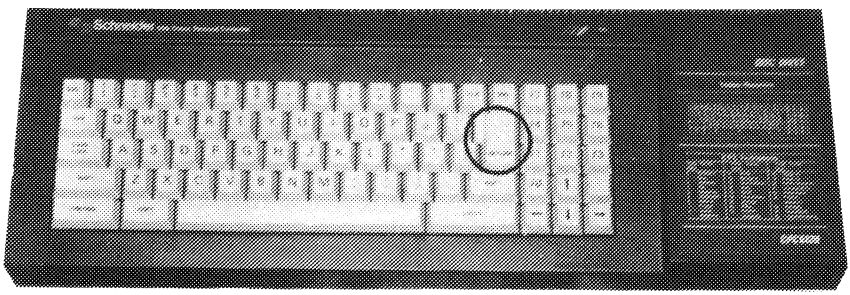
Prøv selv at flytte det sidste ord med SHIFT, piltaster og COPY.

Klarede du det. Tillykke! Med lidt mere øvelse, vil du snart beherske denne teknik fuldkomment. Pas især på i begyndelsen, ikke at forveksle de to cursorer med hinnanden, når de skal flyttes.

Kapitel 2: DEN FØRSTE KOMMANDO

Nu skal vi et skridt videre end blot at skrive bogstaver på skærmen. Det er jo nok de færreste, det har anskaffet en dyr computer kun til dette formål. For de læsere, der måske fandt kapitel 1 lidt langtrukkent, bliver dette kapitel noget mere interessant. Du skal nu bruge computeren til det den egentlig er beregnet til: nemlig udførsel af brugerens (dine) kommandoer.

RETURN-tasten



RETURN er den vigtigste tast på din computer. Når du trykker på denne, overgives skærmteksten til computeren, som først nu oversætter teksten til kommandoer (maskinsprog). Når oversættelsen er sket og kommandoen er udført, melder computeren sig igen klar til at modtage tekst, ved at skrive "READY" — nøjagtig ligesom da den først blev tændt.

Prøv at hilse på din CPC6128 ved at skrive GODDAG og tryk på RETURN. Hvad tror du computeren svarer på dette? Ja, prøv selv at se... Den fejlmelding, der kom på skærmen hedder SYNTAX ERROR (kan bedst oversættes ved GRAMMATIK-FEJL). Syntaksen, altså sammensætningen af tegn, kan computeren ikke forstå. Når du har skrevet en kommando på skærmen og afslutter denne med RETURN undersøger BASIC — fortolkeren den for fejl, og kun hvis kommandoen er fejlfri bliver den bragt til udførelse.

Din CPC6128 har fra starten indbygget sproget BASIC og forstår således kun kommandoer givet i dette sprog. BASIC er i øjeblikket det mest udbredte programmeringssprog, da det giver begynderen optimale muligheder, for at komme igang med at lave programmer, på en let og hurtig måde, BASIC kommandoer er hovedsageligt enten engelske ord eller forkortelser af disse.

Et af de spørgsmål man oftest hører fra begynderen er: "Hvor hurtigt kan jeg lære at programmere?". Svaret er ikke let. Hvor hurtig brugeren selv er til at lære i det hele taget, en anden faktor er, hvor megen tid man vil/kan ofre på sagen ad gangen. Der findes (mange) mennesker, der tilbringer det meste af deres fritid foran skærmen, og det er klart, at der så må kommer ret hurtige fremskridt. Grundlæggende kan dog siges, at rimelige daglige "doser" giver det bedste resultat for en nogenlunde hurtig indlæring. Hvis du har et basalt kendskab til matematik vil en ugentlig indsats på omkring 10 timer give dig et godt resultat i løbet af ca. 3 måneder. De fleste udvikler i løbet af indlæringsperioden den logiske og abstrakte tænkemåde, der er en forudsætning for at udvikle velstrukturerede programmer. Jo mere du finpudser dine programmer, jo mere vil denne tænkemåde ligge "i rygmarven".

PRINT-kommandoen

Uden denne kommando kan det ikke lade sig gøre at lave et program, PRINT styrer simpelthen hvor tekst og resultater skal sendes hen og hvordan disse oplysninger skal præsenteres. Hvis du f.eks. skal have resultatet af en beregning skrevet på skærmen, eller adressen på en af dine venner gemt på disc'-en, bruges PRINT-kommandoen. Også tekst, der skal sendes til printeren, skal sendes derover med en PRINT.

I de første eksempler arbejder vi i DIRECT-MODE, dsv. du skriver kommandoen direkte til computeren, som så udfører den øjeblikkeligt. Vi er altså endnu ikke begyndt at programmere.

Skriv nu kommandoen "print 10" og tryk på RETURN.

Som du ser dirrigerer PRINT alt hvad der står, som parameter til skærmen (hvis det skal andre steder hen, benyttes en anden form af PRINT). Parametre er dele af en kommando, der i detaljer beskriver, hvilken virkning en given kommando helt nøjagtig skal have.

I vort eksempel hed parameteren 10. "PRINT 10" betyder altså "vis tallet 10 på skærmen". Naturligvis kan ikke blot tal, men også alle bogstaver og en del specialtegn udskrives med PRINT. Hvor fleksibel denne kommando er, vil du se senere i dette kapitel.

Matematik med PRINT

Du skal udregne hvor meget restskat du skal betale, men du kan ikke finde den f.... lommeregner. Bare rolig — hjælpen er nær! Du tænder bare for din

6128 og laver dine beregninger. Jeg bruger ofte selv computeren til småberegninger, og flere bekendte, der kommer på besøg spørger ironisk: "Hvad, kan den også regne ?". Selvfølgelig kan den det. Det manglede da også bare, hvis ikke en computer der koster over 20 gange mere end en lommeregner, skulle kunne det.

Nå, men til sagen. Vi skal se nærmere på de 4 grundlæggende regnearter. Til disse findes på tastaturet 4 symboler, nemlig:

\+ for addition
\- for subtraktion
* for multiplikation
/ for division

Indtastning på en computer svarer ikke helt til indtastninger på en lommeregner. På en computer kan du f.eks. ikke blot skrive "12 * 2 =", da der jo som bekendt mangler en kommando. Da vi ved, at du alligevel lige skal prøve, gennemgår vi nu eksemplet. Du skriver altså "12*2=" og afslutter denne "kommando" med RETURN.

Blev du forvirret over resultatet??

For at forklare, at cursoren blot hoppede ned på næste linie, uden at skrive hverken resultatet eller READY. Computeren må altså have udført et eller andet stykke arbejde, da vi ellers havde fået en fejlmelding.

Her må vi foregribe begivenhedernes gang en smule. Når man programmerer (altså skriver et program), består det færdige program af et antal linier, der udføres i en bestemt rækkefølge. Hver af disse linier begynder med et tal så linien kan identificeres. Det der skete, da du gav computeren "regnestykket", var faktisk at du oprettede en linie med nummeret 12 og liniens indhold blev "*2=". Dette er ikke en korrekt kommando, men det bliver ikke check'et af computeren, når man programmerer i linieform. Her sker kontrollen først ved selve programkørslen.

For at få et resultat ud af beregningen, skal der altså en kommando til. I dette tilfælde den tidligere nævnte PRINT. Du skriver altså "print 12*2" og afslutter med RETURN. Husk, at du altid skal afslutte en kommando med RETURN.

Det hele munder måske ud i dit første "AHA". Computeren har for første gang udført et stykke arbejde for dig.

Du skal bemærke, at et programmeringssprog (kommandostrukturen) kan virke lidt kompliceret, fordi man i vidt omfang benytter sig af forkortelser. En sætning som f.eks. "UDREGN 12 * 2 OG VIS RESULTATET PÅ SKÆRMEN" vil du aldrig kunne benytte i programmering. Helt så let er det desværre ikke.

Modsat mange andre varianter af BASIC, skal du i Locomotive BASIC huske, at lave et mellemrum efter PRINT kommandoen, da du ellers får en "Syntax error" melding. Dette gælder i øvrigt stort set alle andre kommandoer også. Du opnår — hvis du adskiller samtlige kommandoer med mellemrum — desuden også et program, der er langt lettere at rette for fejl. Dette vil du erfare senere. Men vi skal videre med regneeksemplerne. Beregn nu følgende: "12+2", "12—2", "12*2" og "12/2".

Er det OK? Godt, så vil vi kravle et trin længere op ad programmeringsstigen.

Med ovenstående simple metode kan du også lave ret lange beregninger (op til 225 tegn). Du kunne f.eks. prøve at få resultatet af

"1+2+3+4+5+6+7+8+9".

Parentesregning

Der er også mulighed for at foretage forskellige kædeberegninger. Et praktisk eksempel: Vi vil gerne kende den samlede pris på tre tæppestykker.

Kvadratmeterprisen er kr. 123,80. Størrelse på det ene stykke er 2.45 m *2.80 m, det næste er 4.50 m * 3.85 m og det tredie stykke er 2.75 m * 4.80 m. Derforuden skal der beregnes moms, 22%. Hvor stor er nu den samlede pris for alle 3 stykker? Hvis du sætter paranteserne rigtigt, kan du nøjes med en enkelt PRINT kommando:

PRINT (2.45 * 2.80 + 4.50 * 3.85 + 2.75 * 4.80) * 123.80 * 1.22

Det ser mere kompliceret ud, end det egentlig er. Indenfor paranteserne bliver det samlede kvadratmeterantal beregnet. Derefter ganges kvadratmeterantallet med kvadratmeterprisen og momsprocenten beregnes. Og det hele med en kommando. Hvis du har skrevet det hele korrekt, skulle resultatet gerne blive 5646.48086, altså en samlet pris på i alt kr. 5646,48.

Som du allerede har set bruges punktum i stedet for komma i decimalberegninger. Det er egentlig en amerikansk norm, som også computere på det danske marked benytter. Hvis du prøver at bruge komma i stedet for punktum, f.eks. PRINT (9 + 23,8) * 2, får du fejlmeldingen "Syntax error".

Exponentialregning

I eksemplet tidligere viste computeren resultatet på skærmen med 5 decimaler. Den regneform der benyttes hedder flydende kommaregning. Grundlæggende bliver ni talpositioner beregnet. Arbejder du f.eks. med 3 decimaler før kommaet, så udregnes automatisk 6 decimaler efter kommaet. Skulle tallet overskride værdien 999999999 bliver resultatet kun beregnet ud fra totalt ni talpositioner. Et større tal vil få et tegn tilføjet, der viser at tallet er større end 999999999. Prøv f.eks. at lave følgende udregning på computeren:" 999999999 + 1". Resultatet vil være et ticifret tal, som ikke vises på den måde du er vant til:

1E+09

Nu skal man ikke fortvivle over computerens måde at skrive resultatet på. Det betyder ganske enkelt, at tallet er lig med 1 i 9'ende, altså et ettal efterfulgt af 9 nuller. "E + 09" betyder Basis 10, Eksponent + 9, Eksponenten kan imidletid også være negativ. Regnestykket "PRINT 9/30000000000" viser dette.

Selvom resultatet egentlig skulle være 0.0000000003 benytter computeren eksponentialformen. "3E-10" betyder nøjagtig det samme som "3 * 10 i -10'ende potens". Tallet 3 befinder sig altså på tiende cifferplads efter kommaet. Dette er foreløbig alt, hvad du behøver at vide om de 4 grundregnearter. Vil du vide noget om andre matematiske funktioner, så prøv at se i instruktionsbogen.

Tekst med PRINT

Ud over tal kan også tekst, såkaldte "strenge", kaldes frem med PRINT. Du har måske allerede opdaget, at forskellige forsøg med PRINT HALLO mislykkedes for dig.

```
PRINT HALLO
0
READY
```

Selvom kommandoen ikke førte til det ønskede resultat, bemærkede du måske at det ikke gav en fejlmelding. Computeren har altså udført kommandoen. Men hvad har den egentlig lavet? Og hvorfor nullet?
Det der sker er, at computeren viser dig indholdet af en såkaldt VARIABEL. En variabel er betegnelsen for et sted i lagerområdet computeren bruger internt. Navnet på dette område er HALLO. Da vi ikke har lagt nogen information ind i variablen HALLO, er værdien altså 0. Hvis du ikke helt forstår dette på nuværende tidspunkt, kan vi trøste dig med, at det er noget, der kommer mere om senere.

Men hvordan kan man nu kalde strenge (i vort tilfælde tekst) frem på skærmen med PRINT? Løsningen er ganske enkel: Strenge skrives altid indrammet af anførselstegn. Hvis du derfor gerne vil have en frisk hilsen fra din computer skriver du blot:

PRINT "GODDAG"

Som du ser, gør computeren nøjagtigt det, du bad den om.

I øvrigt: Det skal indskærpes at uanset hvor mange fejl du laver, ved at skrive til computeren, kan du ikke ødelægge noget! Uanset hvor fejlbehæftet en kommando skrives, er det værste der kan ske, at den overhovedet ikke udføres. Der findes nok ingen hobby der i den grad lever op til mundheldet "Man lærer af sine fejl". En radioamatør der slutter strømmen forkert til et 2000 kr.'s byggesæt drager en rimelig kostbar lære af dette. Det kan ikke ske for dig! Når du har lavet fejl i et program på 23K tager det kun ganske kort tid at rette det. Den eneste (og værste) ulykke der kan ske er, at du, efter at have siddet og tastet programlistninger ind i 8 timer, kommer til at slukke for computeren, uden at have gemt resultatet på disc. først. Så kan det godt være at der kommer nogle udsøgte bemærkninger fra dig!!

Forenkling af PRINT kommandoen

I stedet for kommandoen PRINT kan man bruge et spørgsmålstegn. Vænner du dig til at bruge dette i stedet for at skrive PRINT, sparer du dig selv for en del arbejde, da PRINT er en af de hyppigst anvendte kommandoer. Det er altså fuldstændig underordnet, om du skriver "PRINT 3*7" eller "? 3*7". Prøv det selv engang. Du behøver ikke at lave et mellemrum mellem spørgsmålstegnet og tallene.

PI og potensregning

Vi skal lige have en smule matematik mere. Konstanten PI burde være kendt af enhver. Den bruges mest ved cirkel- og kugleberegninger. Men hvilken værdi har PI. Er det 3.1412 eller 3.1214 eller ....? Slap bare af. For det meste kan 3.14 bruges, altså 2 decimaler efter kommaet. CPC6128 har allerede denne konstant klar til brug. Prøv at skrive "PRINT PI" og se hvad der sker:

PI med 8 decimalers nøjagtighed! Det må da vist være mere end tilstrækkeligt, Prøv nu at beregne omkredsen af en cirkel med diameteren 12 cm. Formelen er Omkreds = Tværsnit * PI.

Omkredsen er altså ca. 37.7 cm.

Nu til potensregning. At opløfte et tal betyder, at tallet skal ganges med sig selv. Exponenten ("opløfteren") bestemmer hvor mange gange Basis (grundtallet) skal ganges med sig selv.

2 i 3. betyder 2*2*2, altså 8
10 i 4. betyder 10*10*10*10, altså 10000

I det første eksempel er "2" Basis og "3" exponent. Men det er vist på tide at slutte matamatikundervisningen. Blot skal det nævnes, at på CPC6128 er selve potenstegnet mellem Basis og exponent, vist som en pil der peger opad. Prøv nu til slut selv at udregne et par eksempler. Hvad er 2 i 8.?

```
PRINT 2↑8
```

Det var sikkert ikke så svært. Beregn så arealet af en cirkel med diameteren 12 cm. Formlen lyder: Areal = Radius i 2. * PI (radius er det halve af diameteren).

```
PRINT 6↑2*PI
```

Overfladen udgør altså 113 kvadratcm. Lad os så slutte med denne tørre matematik og kigge på noget lidt mere interessant.

Kombinering af strenge og tal

Tasterne ";" og "," spiller en stor rolle, når du skal bruge strenge og tal sammen. Hvis du f.eks. vil have en linie som "2.5 gange 2.5 = 6.25" på skærmen, så er det nødvendigt med både en streng og en beregning. Løsningen ser ud som på dette billede:

PRINT "2.5 * 2.5 ="; 2.5 * 2.5

Semikolon'et adskiller altså strengen fra tallene. Men ikke blot det. Strenge og tal, f.eks. i beregninger, kan placeres under hinanden. Prøv nu at opløfte tallet "2" med exponenterne fra "1" til "8" og vis resultaterne under hinanden.

PRINT 2↑1; 2↑2; 2↑3; ...

En PRINT kommando, der afsluttes med semikolon bevirker at en efterfølgende PRINT kommando bliver placeret på samme linie i stedet for på linien nedenunder. Det ser vi nærmere på om lidt.

Prøv nu at skrive den sidste opgave igen, men denne gang erstatter du alle semikolon med kommaer.

Kommaet placerer også udskriften på samme linie, men med større afstand, Hvis du tæller efter, vil du se at afstanden svarer til 13 tegn. På tegn. På denne måde opnår man let en pæn opstilling af talkolonner. Ved strengbehandling benyttes kommaet derimod kun sjældent.

Kommando-struktur

En kommandolinie kan maximalt bestå af 255 tegn. BASIC kommandoer er kun yderst sjældent så lange. Ved hjælp af KOLON kan man adskille forskellige kommandoer på en linie og således bruge mange kommandoer i samme linie. Det kan også bruges i såkaldt "direkte mode" (hvor du skriver kommandoerne direkte til computeren). Vi prøver med et eksempel: Du vil udføre 2 beregninger med kun et tryk på RETURN. Yderligere ønskes resultaterne placeret under hinanden. I vort eksempel kan vi benytte udregninger af 30 i 2. * PI og 30 * PI. Prøv nu at anbringe to PRINT kommandoer på samme linie. adskilt af kolon.

PRINT 30↑2×PI: PRINT 30×PI

Du kan naturligvis også skrive 3 eller 4 kommandoer på linien, adskilt af kolon. Det eneste du skal tænke på er, at det hele ikke må overskride 255 tegn.

Her kan vi iøvrigt illustrere semikolons virkning ganske effektivt. Slet skærmen og skriv begge PRINT kommandoer igen, men skriv efter den første også et semikolon.

Semikolon efter den første PRINT kommando bevirker at linien ikke betragtes som afbrudt af computeren. Det andet PRINT følger efter det første. Mellemrummet mellem tallene fremkommer ved at computeren gør plads for et fortegn (+ eller -). Bemærk i øvrigt, at om du har et, to, fem eller flere mellemrum efter kolon'et er totalt ligegyldigt. Computeren er ligeglad og det har absolut ingen indflydelse på den efterfølgende kommando.

Kapitel 3: DET FØRSTE PROGRAM

Når du nu skal til at lave dit første program, skal du ikke forvente, at dette vil berettige dig til et videnskabeligt diplom. Vi starter stille og roligt, så du har mulighed for hele tiden, ganske nøjagtigt at vide både hvad der sker, og hvorfor det sker, Disse kapitler kan du simpelthen betragte som en grundig indføring i programmering i BASIC.

Hvad er et program?

I fagsproget defineres et program som "en række kommandoer, der løser et givet problem". Når denne række kommandoer sammenkædes i et logisk forløb, har man et program. Inden du sidder med et færdigt program, skal der altså indgå 2 faktorer: Opgavebeskrivelse og en logisk kommandorækkefølge. Vendingen "logisk kommandorækkefølge" indebærer at programmøren ikke blot skal kende de forskellige BASIC kommandoer, men han skal også være i stand til at kombinere disse på en måde, så sammensætningen fører frem til målet. Det er f.eks. ret håbløst at kende en masse franske ord, hvis man ikke kan sætte disse sammen til en sætning der giver mening for en franskmand.

Linienummereringen

Et program består altså af en råkke af kommandoer. Men hvordan bestemmes rækkefølgen? I BASIC forsynes hver eneste kommandolinie (kaldes også Statements) med et nummer. Det er dette nummer, der bestemmer i hvilken rækkefølge de forskellige statements skal udføres. Prøv f.eks. at tænke dig til hvordan man løser opgaven "48 divideret med 12" på en lommeregner. Også her er der en ganske bestemt rækkefølge, der skal overholdes for at nå frem til resultatet:

1. Find lommeregner
2. Tænd for den
3. Hvis batteriet er "fladt" så punkt 12
4. Tryk på tast 4
5. Tryk på tast 8
6. Tryk på operatortegnet for division
7. Tryk på tast 1
8. Tryk på tast 2
9. tryk på tast =

10. Aflæs resultatet
11. Nedskriv resultatet
12. Sluk lommeregneren

Det er faktisk utroligt, så mange enkeltdele en såre simpel handling består af. Som du ser, er alle handlinger nummereret for at fastlægge rækkefølgen. Udover at fastlægge rækkefølgen indebærer nummereringen en vigtig ting mere. Handling nr. 3 indfører en vigtig detalje i programlogikken: Springkommandoen. Når en bestemt forudsætning er opfyldt (batteriet er fladt) går programmet direkte til punkt 12. Dette tal er den såkaldte kommandoadresse. Uden linienumre var det umuligt at udføre et spring til en bestemt linie.

Vi lærte i forrige afsnit PRINT kommandoen at kende. Denne kommando arbejdede vi kun med i direkte mode, dsv. kommandoen blev udført med det samme, når du trykkede på RETURN. Vi skal nu prøve at lave 3 PRINT kommandoer i programform. For at gøre dette, må vi naturligvis benytte linienumre og her er det vigtigt at vide at:

LINIENUMRENE SKAL PÅ CPC6128 LIGGE I OMRÅDET MELLEM 0 OG 65535. AFSTANDEN MELLEM TALLENE (LINIESPRING) ER UDEN BETYDNING!

Liniespringet bestemmer hvor meget større den følgende linies nummer skal være, og her er du helt frit stillet. Et program kan f.eks. bestå af linierne 1, 8, 10, 20 osv. — men du kan også senere i programmet have linier, der springer 100 hver gang, altså 200, 300, 400. De må gerne variere indenfor det enkelte program.

Liniespringet har stor betydning for den videre udvikling af dit program. Hvis du f.eks. brugte et spring på 1 mellem linierne, ville du låse dig selv fast, så du ikke senere kunne "skubbe" en ekstra kommando ind mellem 2 eksisterende linier. I praksis arbejder man mest med et liniespring på 10. Dette giver altid mulighed for 8 ekstra kommandolinier à hver 255 tegn. Så er der plads til udvidelser.

Men nu til selve opgaven:

1. Slet skærmen
2. Skriv teksten "18 divideret med 6 bliver:" på skærmen
3. Skriv resultatet umiddelbart efter teksten.

For at løse opgaven skal vi bruge 3 BASIC linier. Vi begynder med at finde et nummer til den første linie, f.eks. 100. Du skriver altså tallet 100 efterfulgt af

kommandoen for at slette skærmen. Linien kommer så til at se således ud:

100 CLS

OBS! LINIENUMMER SKAL ALTID HAVE ET MELLEMRUM EFTER SIG.

Vi har nu lavet den første linie i vort program. Den næste linie skal udskrive en tekst, og det sker som bekendt med kommandoen PRINT. Vi bruger et liniespring på 10, så linien kommer til at hedde 110. Skriv nu denne linie:

110 PRINT "18 div. med 6 =";

Efter strengen indsættes et semikolon, så den følgende PRINT kommando skriver på samme linie i stedet for at rykke en linie ned.

Den sidste linie skulle du nu kunne klare selv. Den skal beregne 18/6 og skrive resultatet.

PRINT 18/6

Nu er programmet færdigt og skal blot startes.

Programstart

Computeren har ikke blot skrevet programlinierne på skærmen, men også lagret dem i sin hukommelse. Hvis f.eks. skærmen slettes, mister du altså ikke dit program. Det kan genstartes til enhver tid (indtil du slukker eller "resetter" computeren).

KOMMANDOEN "RUN" STARTER ET BASIC PROGRAM

Skriv nu kommandoen "RUN" og læg mærke til, hvad der sker på skærmen.

Du har nu set hvordan dit første program blev til. Det kan du bruge igen og igen ved at skrive RUN. Prøv selv et par gange mere.

Programændringer

Det kan tit forekomme, at du vil ind og ændre i et program, du har skrevet. Vi kunne forestille os, at du ville ændre regnestykket i eksempel fra "18/6" til

"24/6". For at gøre det, behøver du ikke at skrive alle tre linier igen — du skal kun rette den linie der indeholder beregningen.

For at gøre det må vi have programmet frem på skærmen igen.

KOMMANDOEN "LIST" VISER PROGRAMLINIERNE PÅ SKÆRMEN IGEN.

Skriv nu kommandoen LIST efterfulgt af RETURN og se hvad der sker. Dine programlinier kommer igen til syne på skærmen. Du har sikkert allerede set, at computeren har lavet dine kommandoord om til "store bogstaver". Dette er for at gøre det lettere for dig at gennemgå listningen. "Listning" er betegnelsen for at udskrive et program på enten skærm eller printer.

Her skal du bemærke: Når større listninger skal udskrives "scroller" skærmbilledet. At et billed scroller betyder ganske enkelt at computerskærmen kan betragtes som et vindue, hvor teksten kører frem. Er der mere tekst end der er plads til i vinduet på een gang, forsvinder den overskydende tekst ovenud. For at stoppe denne "scrolling" kan du trykke en enkelt gang på "ESC" tasten. Når du har læst færdig, trykker du på en af de andre taster på tastaturet, og teksten kører videre. Hvis du trykker 2 gange efter hinnanden på ESC afbrydes listningen.

Nu tilbage til programændringen. For at ændre beregningen til 24/6 har du 2 muligheder:

1. Metoden med COPY/cursor
2. EDIT kommando metoden

COPY/cursor metoden gennemgik vi i starten af bogen, så den benytter vi os også af nu. Flyt COPY cursoren til begyndelsen af linie 110 ved hjælp af SHIFT tasten.

Tryk på COPY tasten, indtil du når hen til det første tal, der skal ændres:

På dette sted skal den første ændring placeres. Skriv nu tallet "24". Derefter flytter du COPY cursoren (SHIFT/COPY) hen til mellemrummet efter 18 og kopierer den resterende linie med tryk på COPY alene. Tryk derefter RETURN og linieændringen er afsluttet. For at resultatet skal stemme med det der står på skærmen, skal vi ændre selve udregningen i linie 120. Ændringen foretages på samme måde som i linie 110. Det er ikke helt let at editere med COPY cursor metoden, men også her gør øvelse mester. Jeg havde selv problemer med den ret usædvanlige editeringsmetode i begyndelsen, men nu sidder det faktisk i fingerspidserne.

Vi skal prøve at se på den anden form for programændring. Kommandoen "EDIT" kalder en bestemt linie frem til ændring på skærmen. Skriv nu kommandoen EDIT 110 (husk RETURN).

Linie 110 vises på skærmen og cursoren befinder sig ved begyndelsen af linien. Flyt cursoren hen til det første tal, der skal rettes. I det eksempel vil vi igen skrive 18 i stedet for 24. Når du skriver tallet 18, vil du se at det ikke overskrider 24, men i stedet for bliver "skubbet ind" umiddelbart før 24.

EDIT kommandoen arbejder i "indsæt-mode". Vi mangler altså at slette tallet 24. Til dette bruger du tasten "CLR", der jo som bekendt sletter det der befinder sig under cursoren. Tryk på CLR to gange.

Når alle ændringer er afsluttet trykker du på RETURN. Du kan nu sikre dig at linien virkelig er rettet, ved at skrive LIST.

Du har her lært 2 metoder, til at rette i programlinier, at kende. Det er så op til dig selv, hvilken du vil bruge fremover.

Forgrening

Det er kun sjældent at et program kører den lige vej fra det laveste linienummer til det højeste. Ofte springes der rundt i programmet, når bestemte betingelser er blevet opfyldt.

KOMMANDOEN "GOTO" SPRINGER TIL ET ANGIVET LINIENUMMER

Lad os straks omsætte dette i praksis. Som du så, endte vort program med linie 120. Hvad ville der ske, hvis vi lavede en linie 130, der bad computeren returnere til linie 110? Ja, prøv engang at skrive: 130 GOTO 110 og start programmet med RUN.

Lille årsag — stor virkning.

Det du har lavet er en såkaldet "endeløs løkke", der tvinger programmet til at køre i ring mellem linierne 110 og 130. Hvis du ikke selv griber ind, vil programmet fortsætte med det sålænge elværket leverer strøm til dig!

"ESC" TASTEN AFBRYDER PROGRAMFORLØBET

Tryk på ESC tasten og programmet stoppes. Hvis du efter ESC trykker på en anden tast (lige meget hvilken), vil programmet fortsætte. Hvis du vil afbryde helt, skal du trykke 2 gange efter hinanden på ESC. Computeren giver dig en besked: "BREAK IN ..." og viser således hvilken linie programkørslen blev afbrudt i. I vort eksempel er det helt tilfældigt, hvor du får stoppet programmet.

Skal programmet fortsætte efter en "BREAK IN ...", findes der også en kommando til det. Fortsæt hedder på engelsk "continue", og da BASIC kommandoerne læner sig kraftigt op ad engelsk, hedder fortsættelseskommandoen "CONT".

KOMMANDOEN "CONT" STARTER ET PROGRAM, EFTER AT DER HAR VÆRET AFBRUDT MED "ESC".

Skriv nu denne kommando. Programmet befinder sig igen i den uendelige løkke og kan afbrydes med ESC.

At GEMME og HENTE programmer!

Når du slukker din computer forsvinder programmet. Da du sikkert ikke ønsker at genindtaste alle dine programmer hver gang du slukker computeren, er det på tide at lære, hvordan du gemmer mesterværkerne via den indbyggede diskettestation. Du skal nu prøve at gemme programeksemplet (FORMATTERING af diskette beskrives i kapitel 8). Du vælger først et navn til programmet, der maximalt må bestå af 8 tegn. Du kan her bruge navnet "TEST". Kommandoen der gemmer et program eller data hedder "SAVE". Skriv nu sådan:

SAVE "TEST"

Hvis du vil hente og starte programmet igen, må du bruge kommandoen "LOAD".

LOAD "TEST"

Det kan jo ske, at du glemmer navnet på dit program. Kommandoen "CAT" vil vise hvilke programmer, der er gemt på disketten.

Programmet startes med RUN. Efter selve kommandoen RUN kan du skrive et linienummer. Her er et eksempel:

Kommando	Funktion
RUN	starter et program, der allerede ligger i hukommelsen.
RUN 100	starter et program, der ligger i hukommelsen fra linie 100.

Programmet startes i sidste tilfælde ikke fra begyndelsen, men fra linien der er specificeret efter RUN.

Det er faktisk alt hvad du behøver at vide om at gemme og hente programmer fra diskette. Kapitlet "DISKETTESTATIONEN" dækker senere flere aspekter af de mange muligheder diskbehandlingen tilbyder.

Sletning af programmer

Som vi tidligere har set, slettes et program hvis du slukker for computeren, men det er ikke den eneste mulighed.

KOMMANDOEN "NEW" SLETTER ET PROGRAM DER BEFINDER SIG I HUKOMMELSEN.

Skriver du nu kommandoen NEW, så slettes dit program. Man bør naturligvis altid gemme sit program, inden man bruger denne kommando.

Prøv at skrive NEW (og RETURN). Hvis du derefter skriver LIST eller RUN, vil du blive overbevist om effekten af denne kommando.

Du har nu lært de første skridt af vejen til at fremstille et program. Hvis der på et tidspunkt er nogle af de ting, vi har gennemgået, der stadig er uklare, så prøv at gennemgå kapitlet en gang til.

Kapitel 4:
PROGRAMMERINGSHJÆLP

Den omfangsrige BASIC i din CPC6128 ligger til grund for dette kapitel. Det er ikke mange computere, der i standardversionen tilbyder hjælpefunktioner til fremstilling og bearbejdelse af programmer. På Amstrad computeren findes kommandoer til automatisk linienummerering, sletning af dele af et program, renummerering osv. osv. Vi skal i det følgende se på de mest anvendte hjælpemidler.

Automatisk linienummerering

Når man skriver et program er linieafstanden (liniespringet) som regel 10. Når man er dybt involveret i et programmeringsproblem, glemmer man ofte at skrive linienummeret inden man giver sig i kast med selve problemløsningen. Dette har man hos Amstrad erkendt, og derfor er der i din computer indbygget en kommando, der automatisk laver nummerering af linierne. Her får du en oversigt over kommandoen:

Funktion: Automatisk linienummerering
Kommando: AUTO start, afstand
Parametre: start — begyndelseslinie
afstand — afstand mellem linienumrene

Eksempel: AUTO 100,10
Begyndende med linie 100 øges linietallet med 10 for hver ny linie (100, 110, 120, 130, 140 ...)

Kommentar: Den automatiske linienummerering kan afbrydes med tasten "ESC". Er et linienummer allerede i brug, vises en stjerne (*) efter linienummeret. Med RETURN beholdes denne linie.

Her ser du for første gang en nøjagtig beskrivelse af en kommando. En sådan vil du fremover se hver gang vi introducerer en ny kommando. Der er måske grund til at forklare ordet "parametre" lidt nærmere. Parametre er betegnelsen for alt hvad man tilføjer kommandoen — altså alt, hvad der står bag ved selve kommandoen. Til en så almindelig kommando som "LOAD" f.eks. findes der også et parameter, nemlig programnavnet.

Men tilbage til AUTO. Lad os antage, at du vil lave et program, der begynder med linie 10 og har et spring i numrene på 5. Dette kan AUTO lave for dig.

Prøv kommandoen ved at skrive:

AUTO 10,5

Straks kommer det første linienummer til syne (10). Computeren venter nu på at du skriver noget på denne linie, f.eks. "PRINT "Linie 10"". Afslut linien med RETURN. Nu bliver næste linie automatisk mummereret 15 og computeren venter igen på tekst. Du kan skrive "PRINT "Linie 15"" — men denne gang afslutter du linien ved at trykke på "ESC". Meldingen "READY" kommer nu på skærmen og viser at den automatiske linienummerering ikke mere er aktiv.

Prøv igen at skrive "AUTO 10,5", for at se en specialitet ved kommandoen. Computeren reagerer på følgende måde:

10*

Hvad betyder stjernen? Hvis du har læst beskrivelsen grundigt, kender du sikkert allerede løsningen. Stjernen signalerer, at linien allerede er i brug. Dette er vigtigt at vide, så man ikke uforvarende kommer til at slette noget af det man allerede har skrevet. Hvis linien skal forblive uforandret trykker du blot på RETURN. Hvis du derimod vil ændre linien skriver du den nye tekst og afslutter med RETURN. Prøv at beholde linie 10 ved blot at trykke på RETURN og skriv så i linie 15 "PRINT "Ny linie 15"". Derefter afslutter du kommandoen AUTO med "ESC" tasten og du kan se ændringen ved at skrive "LIST".

Omnummerering (RENUMBER)

Som du allerede ved, har afstanden imellem de enkelte linienumre ingen inflydelse på selve programforløbet. Om du bruger spring på 1, 5 eller 10 er helt op til dine egne ønsker. Du skal dog være opmærksom på, at en for lille afstand mellem linierne gør det svært at føje noget til et program.

Skriv nu følgende lille programstump:

```
1 CLS
2 PRINT "Vi multipliceret nu to tal:"
3 PRINT "Det første tal er 15"
4 PRINT "Det andet tal er 12"
5 PRINT "og resultatet bliver";15*12
```

Som du ser, har vi brugt et liniespring på 1. Programmet skriver følgende på skærmen:

Vi multiplicerer nu to tal:

Det første tal er 15
Det andet tal er 12
og resultatet er 180

For at opnå dette skal der indsættes en linie efter linie 2. D.v.s. de linienumre, der ligger efter linie 2, skal forhøjes med 1 for at få plads til en ny linie 3. Det lyder mere indviklet end det er. Programmet kommer nu til at se således ud:

```
1 CLS
2 PRINT "Vi multiplicerer nu to tal:"
3 PRINT "--------------------------"
4 PRINT "Det første tal er 15"
5 PRINT "Det andet tal er 12"
6 PRINT "og resultatet bliver";15*12
```

Hvordan løser vi dette problem? Let — du behøver nemlig ikke at skrive linierne 3 til 5 om — computeren har en kommando netop til dette brug, og her får du beskrivelsen:

Funktion: Renummerering af linier.
Kommando: RENUM ny, gammel, spring
Parametre: ny — ny startlinie
gammel — gammel startlinie
spring — nyt liniespring

Eksempel: RENUM 100,10,5
Programmet renummereres så gl. linie 10 kommer til at hedde 100 og resten af linierne i programmet springer 5 numre.

Kommentar: Den sidste programlinie må efter renummerering ikke overstige 65535. Er dette tilfældet bliver kommandoen RENUM ikke udført og fejlmeldingen "Improper argument" vises på skærmen.

Denne kommando er noget mere kompliceret end dem vi tidligere har gennemgået. Som du ser, skal der til kommandoen føjes 3 parametre. Det første tal er det nye laveste linienummer (i vort eksempel vil den nye nummerering starte med 100), det næste er nummeret på linien, omnummereringen skal starte fra (linie 10 i eksemplet), og det sidste er det evt. nye liniespring. Du behøver altså ikke renummerere hele dit program.

Hvordan skal nu kommandoen RENUM se ud, for at vi opnår at linierne 3 til 5 kommer til at hedde 4 til 6? Vi prøver at bygge kommandoen op, skridt for skridt. Først skal vi fastsætte det nye linienummer. Det er 4. Det første gamle linienummer der skal ændres er 3 — det bliver så andet parameter. Den nye linieafstand er 1 — tredie parameter. Den samlede kommando hedder så:

RENUM 4,3,1,

Denne kommando fører til følgende nye listning:

```
1 CLS
2 PRINT "Vi multiplicerer nu to tal:"
4 PRINT "Det første tal er 15"
5 PRINT "Det andet tal er 12"
6 PRINT "og resultatet bliver";15*12
```

Med en enkelt kommando har du skaffet plads til linie 3, som nu kan skrives:

```
3 PRINT "----------------------------"
```

For at undgå disse problemer i fremtiden, arbejder vi fremover med et liniespring på 10. Hvis vi ønsker at vort eksempelprogram skal begynde med linie 100 og have et liniespring på 10, ser kommandoen således ud:

RENUM 100,1,10

Kommandoen RENUM renummerer ikke blot linierne, men den har en yderligere vigtig funktion: Den ændrer også nummereringen i "GOTO" og "GOSUB" kommandoer. Og hvad betyder så det? Prøv at tilføje en linie til dit program (der nu går fra linie 100 til 150). Den nye linie skal hedde:

160 GOTO 110

Vi har med linie 160 lagt en endeløs sløjfe ind i programmet. Hvis vi renummerer programmet vil linie 110 få et ukendt nummer. GOTO kommandoen skal derfor ikke mere springe tilbage til linie 110, da denne linie måske ikke engang er til stede efter renummereringen. RENUM sørger automatisk for at GOTO kommandoen finder det rigtige nye linienummer, hvilket du kan overbevise dig om ved at skrive "RENUM 10,100,5". Når du LIST'er programmet vil du se, at "GOTO" kommandoen nu indeholder et nyt linienummer.

Begrænset sletning af linier

Du har tidligere lært om kommandoen NEW, der bruges til at slette et program i hukommelsen. Men hvad hvis man kun ønsker at slette dele af et program — evt. kun en enkelt linie? Sletning af enkeltelinier er en ren barneleg, efter at du har lært RENUM kommandoen. Du skriver nemlig blot linienummeret, uden at tilføje noget — og linien er slettet. Hvis du ønsker flere linier slettet på en gang, skal du bruge en ny kommando, som vi introducerer her:

Funktion:	Udvalgt sletning af linienumre
Kommando:	DELETE start-slut
Parametre:	start — første linie der skal slettes
	slut — sidste linie der skal slettes
Eksempel:	DELETE 40-130
	sletter linierne fra 40 til 130

Der findes flere muligheder for at angive parametrene. Prøv at se på følgende tabel:

DELETE 10	sletter linie 10
DELETE 100-150	sletter linierne 100 til 150
DELETE 150-	sletter alt hvad der ligger over linie 150
DELETE -100	sletter alt hvad der ligger under linie 100

Nøjagtig samme parametre kan i øvrigt bruges sammen med kommandoen LIST. Også her tydeliggør nedenstående tabel mulighederne:

LIST 10	lister linie 10
LIST 100-150	lister linierne 100 til 150
LIST 150-	lister alle linier over 150
LIST -100	lister alle linier under linie 100

Funktionstaster

Vi skal her se på et af de mest praktiske programmeringshjælpemidler på din computer, nemlig defineringen af funktionstaster.

Det er ofte ærgeligt, at hyppigt anvendte kommandoer skal skrives fuldt ud hver eneste gang. På CPC6128 har du mulighed for at "programmere" hyppigt anvendte kommandoer ind på det numeriske tastatur, så du kan bruge disse med et enkelt tastetryk.

På hver af de ovenstående taster kan du maximalt indlægge 32 tegn. Det samlede antal tegn må dog ikke overstige 120.

Funktion: Programmering af Funktionstaster
Kommando: KEY n, streng
Parametre: n — funktionstastens nummer
streng — diverse tegn (max 32)

Eksempel: KEY 0,"LIST"

Kommentar: Det samlede antal tegn må ikke overstige 120.

På det numeriske tastatur findes i alt 13 taster, nummeret fra 0 til 12. Du undrer dig måske over, hvordan man får 13 funktionstaster ud af kun 12 almindelige taster. Det er muligt fordi "ENTER" tasten har to vigtige funktioner. En funktion ved normal anvendelse, og en anden hvis den bruges sammen med "CTRL" tasten. Det følgende diagram viser hvilket nummer den enkelte tast har:

7	8	9
4	5	6
1	2	3
0	10	11/12

Tasten "." har altså nummer 10, "ENTER" tasten numrene 11 og 12. Lad os prøve at lægge nogle kommandoer ud på tasterne. LIST kommandoen benyttes ofte, så den vil vi have på funktionstast 0. Det klarer følgende kommando:

KEY 0,"LIST"

Glem ikke mellemrummet mellem kommandoen og tastens nummer! Trykker du nu på funktionstast 0 (kan forkortes til F0), skrives kommandoen "LIST" på skærmen. Kommandoen bliver dog ikke udført før du trykker på "RETURN". Du kan også få kommandoen afsluttet med "RETURN" automatisk. For at gøre dette, må du tilføje computerens interne kode for "RETURN" automatisk. For at gøre dette, må du tilføje computerens interne kode for "RETURN" med en speciel kommando, som du senere vil lære mere om. Denne kode er 13. Kommandoen skal se således ud:

KEY 0,"LIST"+CHR$(13)

Udvidelsen CHR$(13) skal du ikke hæfte dig ved lige nu — det der er vigtigt, er hvad kommandoen udfører. Tryk på F0 og du vil se, at kommandoen straks udføres.

Der findes også kommandoer hvor man skal passe på med at tilføje "RETURN". En af disse er "NEW". Prøv at forestille dig hvad der ville ske, hvis du havde denne kommando liggende på en funktionstast, efterfulgt af RETURN. Et forkert tryk — og dit program ville være slettet for tid og evighed. Ikke særlig smart !!

Brug altså kun "RETURN" sammen med kommandoer, der ikke kan gøre uoprettelig skade.

Du har allerede lært koden for "RETURN". I forbindelse med funktionstasterne er der yderligere en vigtig kode, nemlig anførselstegnet. Prøv lige at programmere funktionstast 1 med den kendte kommando "RUN". Det kan nemt give dig grå hår i hovedet! Kommandoen "KEY 1, "RUN"" giver fejlmeldingen "Syntax error". Computeren opfatter anførselstegn nr. 2 som afslutning på kommandoen og ved ikke hvad den skal stille op med det tredie anførselstegn. Vi bliver her nødt til at bruge koden for anførselstegn (34) i stedet. Den korrekte kommando lyder:

KEY 1, "RUN" + CHR$ (34)

Hvis der nu skal tilføjes et yderligere "RETURN" bliver tingene først rigtig komplicerede:

KEY 1, "RUN" + CHR$ (34) + CHR$ (13)

Med tiden vil du sikkert udvikle dine egne standardkommandoer. Du kunne jo samle dine KEY-kommandoer i et program, og gemme dette på disc'en. Programmet kunne f.eks. se således ud:

```
10 KEY 0,"LIST" + CHR$ (13)
20 KEY 1,"PRINT"
30 KEY 2,"NEW"
40 KEY 3,"RENUM"
50 KEY 4,"AUTO"
60 KEY 5,"DELETE"
```

Du kan selv tilføje flere kommandoer, eller lave dine helt egne programmeringer.

En sidste vigtig ting: RESET (SHIFT/CTRL/ESC) sletter kommandoerne på funktionstasterne! Hvis du har programmeret noget ind du vil bruge igen, skal du derfor gemme det på disc'en inden du resetter computeren.

Kapitel 5: INDFØRING I BASIC

Dette kapitel er tænkt som en indføring i programmeringssproget BASIC. I stedet for f.eks. at gennemgå de enkelte kommandoer i alfabetisk rækkefølge vil vi prøve at opbygge et lille kartoteksprogram, og på den måde få et overblik over disse, efterhånden som vi støder på dem.

Du er ikke, efter at have læst dette kapitel, den perfekte programmør — men du får et grundigt indblik i den praktiske programmering. Det vil give dig de bedste forudsætninger for at dykke ned i den righoldige litteratur, der findes på markedet om programmering i BASIC.

Problembeskrivelse

Et kartoteksprogram er et af de nyttigste programmer, du kan lave til din computer. Det brede anvendelsesområde gør, at stort set alle, kan have fornøjelse af et sådant program. Har du imidlertid kun få oplysninger, der skal opbevares, er det ingen ubetinget fordel at gemme disse i et sådant program. Her er de traditionelle kort og små sedler stadig hurtigere og nemmere at bruge — selvom det ser imponerende ud at hente sine oplysninger fra computeren. Ved store datamængder er computeren derimod det ideelle hjælpemiddel.

Hvilke oplysninger skal et kartoteksprogram så indeholde?

I alle tilfælde skal der være mulighed for at lægge oplysninger ind i programmet, og senere hente dem ud igen.

Før vi stiller yderligere krav til programmet, må vi gøre os en enkelt ting klart: Et program der bruger data (oplysninger) består af 2 grundlæggende dele.

1. Selve programmet og
2. Programmets data

De forskellige data er ikke del af programmet. Det er muligt at gøre dem til en integreret del af programmet, men dette ville indebære at kun programmøren (og ikke brugeren) havde mulighed for at ændre i oplysningerne — og det er jo ikke særlig smart.

For at programmet kan hente og gemme data, må der oprettes en såkaldt datafil. En fil er en samling data på et eksternt lagermedie (f.eks. en diskette). Der findes flere forskellige former for filer, man vi vil her holde os

til den enkleste — den sekventielle fil. At en fil er sekventiel betyder, at de enkelte data ligger i rækkefølge efter hinnanden.

Dataorganisering

Hvis du nu blev spurgt om hvad der først skulle organiseres, ville du sikkert svare: "Programmet". Det er imidlertid forkert. Det vi allerførst skal overveje er: Hvad skal gemmes — og hvordan? Svarene er nærliggende: Vore data skal gemmes på disketten — og der skal de ligge i sekventiel form. Og hvad skal så gemmes? Vi kunne f.eks. gemme adresser på gode venner. Ud over deres adresser er der imidlertid også andre oplysninger, det ville være rart at have på skærmen, når vi ledte efter den pågældende. Et forslag til et "kartotekskort" kunne se således ud:

Titel	Postnummer
Fornavn	By
Efternavn	Telefonnummer
Gade	Bemærkninger

De enkelte bestanddele i et sådant kartotekskort, kalder man DATAFELTER. Man taler f.eks. om et TITELFELT, BYFELT osv. osv. Alle datafelterne, der indgår i kartotekskortet, kaldes for et DATASÆT. Hvert datasæt i eksemplet består altså af 8 felter. Der er plads til et antal datasæt i computeren og samlingen af datasæt kaldes for en FIL. Prøv lige at kigge på denne opdeling, der illustrerer hvad vi mener:

DATABASE

FELT 1	FELT 2	FELT 3 ..... FELT n	DATASÆT 1	
FELT 1	FELT 2	FELT 3 ..... FELT n	DATASÆT 2	
FELT 1	FELT 2	FELT 3 ..... FELT n	DATASÆT 3	osv.

Hirakiet hedder altså FELT — DATAFELT — DATASÆT.

Opbevaring af data i computeren

En sekventiel fil er let at arbejde med (hvilket du vil erfare..), men den har nogle ulemper i forhold til andre organisationsformer. Forestil dig at filen — med masser af navne og adresser — ligger på en diskette. Du skal nu bruge en ganske bestemt adresse i filen. Og nu kommer problemet: Diskettestationen er ikke i stand til at finde en enkelt adresse i en sekventiel fil — den må have

hele filen ind i hukommelsen, før den kan begynde at søge i den. En sekventiel fil kan derfor ikke være større end den plads, du har til rådighed i computerens hukommelse. Det indebærer i øvrigt en fordel i sig selv. Al søgning i filen foregår lynhurtigt, da computeren jo allerede har oplysningerne liggende. For arbejde med sekventielle filer, gælder altså følgende regler:

1. LOAD FIL
2. LÆS DATASÆT, ÆNDRE, SLETTE .....
3. SAVE FIL

Efter at programmet er startet op, må du som første trin indlæse datafilen. Derefter kan datasættene bearbejdes. Før du afslutter programmet, skal filen igen gemmes på disketten, såfremt du har ændret i den. Hvis du blot har kigget i filen og ikke ændret, tilføjet eller slettet i den, skal den ikke gemmes igen. Så kan du blot afslutte programmet uden videre.

Variabler

Som bekendt gemmes datasættene inde i computerens hukommelse. Men hvordan — og hvor gemmes de?

DATA GEMMES INDE I COMPUTEREN SOM VARIABLER.

Variabler er altså nøgleordet. En variabel er et område i computerens hukommelse der navngives af brugeren. Ved at kalde dette navn, fåes adgang til indholdet af det pågældende hukommelsesområde. Vi har allerede lært to forskellige datatyper at kende: Numeriske data (tal) og alfanumeriske data (strenge, tekst). Strengvariabler kan altid kendes på, at de afsluttes med et "$"-tegn. Et variabelnavn må i Amstrad BASIC højst omfatte 40 tegn. Det første tegn SKAL være et bogstav, resten kan frit være enten tal eller bogstaver. Der findes forskellige navne, som du skal undgå, når du navngiver variabler, nemlig BASIC kommandoer eller funktioner — f.eks.:

PRINT, PI, KEY osv.

Her er nogle eksempler på "lovlige" nummeriske variabelnavne:

TÆLLER
PERSONNR.
TELEFON
KONTO

Og nogle eksempler på strengvariabler:

SPØRGSMÅL$
VARE$
KONTONAVN$
X$

Når du vælger variabelnavne, kan det betale sig, at du tænker dig lidt om. Hvis programmet f.eks. skal spørge om et navn, så kald variablen for "NAVN$", en indkøbspris kunne hedde "INDPRIS" osv. osv. Meningsfulde variablnavne letter overblikket ganske væsentligt, når du senere kigger programmet igennem.

Variabelbearbejdning

Lad os efter denne lange pasus vende tilbage til computeren. Vi skal prøve at lave variabler, bearbejde dem og udlæse dem igen.

At gemme numeriske data i variabler er egentlig en meget let sag. Hvis vi vil lægge en indkøbspris på kr. 45.50 ind i variablen INDPRIS skriver du ganske enkelt:

INDPRIS = 45.50

Vil du igen se indholdet af denne variabel hedder kommandoen:

PRINT INDPRIS

Prøv begge kommandoer på computeren. Hvis du ønsker at nulstille variablen INDPRIS igen, skriver du blot:

INDPRIS = 0

Variabler kan imidlertid også bruges sammen med beregninger.

Vi vil nu gerne tjene lige så meget som varen kostede. Efter at du har nulstillet INDPRIS skriver du igen INDPRIS = 45.50. Da varens udsalgspris er lig med 45.50 * 2 skriver du nu:

PRINT INDPRIS * 2

Computeren fordobler indkøbsprisen og viser resultatet (91.00) på skærmen.

Selve variablen har stadig værdien 45.50.

Ønsker du derimod at fordoble selve variablens værdi, skal du gøre det på en lidt anden måde. Her hedder kommandoen:

```
INDPRIS = INDPRIS * 2
```

Computeren vil altid først beregne variablen til højre for lighedstegnet, og derefter tillægge værdien for variablen til venstre for lighedstegnet.

Strengvariabler behandles lidt anderledes. Lad os gemme strengen "Ringkøbing" i strengvariablen BY$. Det gør vi med følgende sætning:

```
BY$ = "Ringkøbing"
```

Også indholdet af strengvariabler kan vises på skærmen med PRINT kommandoen.

Prøv selv:

```
PRINT BY$
```

Det der skete var, at du fik vist indholdet af variablen BY$. Skal indholdet af BY$ slettes igen må du gemme en såkaldt "tom" streng i variablen BY$. Det gøres på følgende måde:

```
BY$ = ""
```

En tom streng består af 2 hold anførselstegn umiddelbart efter hinanden. Man kan naturligvis ikke udføre beregninger på strengvariabler. Heller ikke hvis man lægger tal ind i en sådan. Det indebærer, at hvis du f.eks. skriver X$ = "123", så kan du ikke bruge variablen X$ til videre beregninger.

Strengvariabler kan imidlertid sammenkædes, som du skal se i det følgende:

```
BY1$ = "Silkeborg"
BY2$ = "Ringkøbing"
BY$  = BY1$ + BY2$

PRINT BY$
```

Det ville være naturligt, at lave afstand mellem de to byer:

```
BY$ = BY1$ + " " + BY$2
```

Tabeller

Når man arbejder med data, og specielt når det drejer sig om flere data af samme type, så anvender man ofte det, der kaldes tabeller. Det er en meget omstændelig sag at bruge et variabelnavn for hvert dataelement, man har i computerens hukommelse. Man giver hellere den samme type data de samme variabelnavne, men med hvert sit kendetegn. Hvis vi som eksempel bruger fem forskellige bynavne:

BONN
PARIS
LONDON
ROM
MADRID

så ville vi med den kendte metode få brug for fem forskellige variabelnavne. Benytter vi i stedet en tabel (også kaldet en array), kan vi nøjes med et fælles variabelnavn. Da dataelementerne stadigvæk skal kunne identificeres enkeltvis, må vi give hvert element et kendetegn. Dette kendetegn består af et tal i en parentes efter variabelnavnet. Tallet kaldes et INDEX. Det første dataelement vil få indexet 1, det andet indexet 2, o.s.v. Men det første index(tal) er 0 (nul) og ikke 1. Vi vælger af oversigtsmæssige grunde tallet 1 som det første index, selvom det medfører at vi smider kostbar hukommelsesplads ud ad vinduet.

En tabel- eller arrayvariabel kunne f.eks. være:

A$(1)

En numerisk arrayvariabel, der også kan indexeres efter behov er:

XY(212)

Et index kan sættes til en hvilken som helst værdi, når blot disse to regler overholdes:

1. Overstiger index'et tallet 10, må man med en særlig kommando reservere plads til tabellen.

2. En tabel må/kan ikke dimensioneres så højt, at den maksimale hukommelse overskrides.

Tabeller med et index indtil 10 forvaltes uden videre af computeren selv. Men hvorledes defineres større arrays? Hertil må vi anvende den særlige BASIC-kommando:

DIM

OBJEKT:	Dimensionering af Arrays (strenge)
KOMMANDO:	DIM variabel (n1,n2,n3...)
PARAMETRE:	variabel — Arraynavnet n1,n2,n3 — Maks. index pr. dimension.
EKSEMPEL:	DIM D$(20) En Array (streng) kaldet D$ dimensioneres i een dimension med index 20.
BEMÆRKNING:	En Array (streng) må kun dimensioneres EEN gang i et program, ellers fremkommer fejlmeddelelsen: "ARRAY ALREADY DIMENSIONED"

Hvis det maksimale index eksempelvis skal være 5, så er det ikke nødvendigt at dimensionere strengen med kommandoen: DIM.

Dette er i og for sig meget smart, men vær opmærksom på, at computeren automatisk HAR dimensioneret ALLE strenge til INDEX 10. Har man kun brug for et index på 5-6, så er resten op til de 10 spild af kostbar hukommelse! Hvis man bruger mange strenge, i et program, og har brug for at spare plads, så kan man alligevel bruge DIM-kommandoen. Hvis man er sikker på kun at få brug for et index på f.eks. 5, kan man nøjes med at reservere de 5 dataelementer med DIM.

Men lad os vende tilbage til de fem bynavne, der skal lægges ind i en TABEL. Hvis vi bestemmer os for at bruge variablen D$, vil bynavnene blive gemt med de følgende linier:

```
10 D$(1)="BONN"
20 D$(2)="PARIS"
30 D$(3)="LONDON"
40 D$(4)="ROM"
50 D$(5)="MADRID"
```

MED KOMMANDOEN DIM D$(5) KAN MAN FJERNE RESERVATIONEN AF INDEX 6-10

BEMÆRK: Overskrides en streng's index, vil man få fejlmeddelelsen:
BAD SUBSCRIPT ERROR

Lad os udvide eksemplet ved at tilføje de fem hovedstæders lande. Disse placeres også i en tabel. Lad os kalde variablen for L$. Vi får følgende tilskrivning:

```
 60 L$(1)="BRD"
 70 L$(2)="FRANKRIG"
 80 L$(3)="ENGLAND"
 90 L$(4)="ITALIEN"
100 L$(5)="SPANIEN"
```

Nu har vi fået konstrueret to tabeller, hvis index står i en umiddelbar sammenhæng, hvilket vil sige at f.eks. D$(1) indeholder hovedstaden i landet L$(1).

Det specielle ved tabeller er, at deres index ikke nødvendigvis skal være nøgne tal, men udmærket kan være givet via variabler eller aritmetiske udtryk.

En array (streng) betegnet ved:

```
DS$(X-2*I+13)
```

er helt lovlig. Fordelene ved dette, skal vi straks se på.

Begge de viste tabeller, er af den slags, der kaldes ENDIMENSIONAL. Da tabeller kan være indekseret i flere dimensioner, kan vi lave de to tabeller om til een todimensional tabel:

```
 10 DIM D$(1,5)
 20 D$(0,1)="BONN"
 30 D$(0,2)="PARIS"
 40 D$(0,3)="LONDON"
 50 D$(0,4)="ROM"
 60 D$(0,5)="MADRID"
 70 D$(1,1)="BRD"
 80 D$(1,2)="FRANKRIG"
```

```
 90 D$(1,3)="ENGLAND"
100 D$(1,4)="ITALIEN"
110 D$(1,5)="SPANIEN"
```

DIM-kommandoen i linie 10 er ikke nødvendig, idet computeren automatisk DIMensionerer tabellen:

```
DIM D$(10,10)
```

Hvilket svarer til 11*11 (121) pladser i tabellen. Imidlertid, har vi kun brug for 10 pladser. Husk derfor nøje at overveje, hvor meget plads man har brug for, inden man dimensionerer sine variabler.

Vi har nu en tabel i to dimensioner. Det første index angiver gruppen (by eller land), og det andet tabelpladsen. Vi kan vise tabellen i et diagram:

	1	2	3	4	5
0	BONN	PARIS	LONDON	ROM	MADRID
1	BRD	FRANKRIG	ENGLAND	ITALIEN	SPANIEN

En tabel i to dimensioner er ideel i sammenhæng med et adressekartotek ordnet sekventielt. Det første index numererer datasættene, og det andet numererer felterne i hvert datasæt. Vi har allerede fundet frem til, at vi har brug for 8 felter pr. datasæt; og dermed har vi også fundet, at det andet index skal være højst 8. Det første index, det der angiver antallet af datasæt, er valgfrit. Lad os vedtage at vores kartotek ikke skal kunne indeholde mere end 200 adresser. Tabellens navn skal være D$. DIM-kommandoen for denne dimensionering bliver således:

```
DIM D$(200,8)
```

Hvordan tabellen forvaltes, kommer vi til senere.

Indtil nu, har vi kun forarbejdet og udskrevet data. Det er langt mere interessant at vide, hvordan programmet egentlig i første omgang modtager disse data over tastaturet. Kommandoen, vi skal bruge hedder:

INPUT

Men INPUT alene har ikke nogen virkning. Kommandoen retter sig efter to parametre: En streng, der udskrives før det egentlige INPUT sker, og en variabel, hvori de indtastede data skal gemmes.

BEMÆRK: KOMMANDOEN INPUT SKAL GIVES I ET PROGRAM. DEN KAN IKKE GIVES I DIREKTE MODE.

OBJEKT:	Input af data via tastaturet.
KOMMANDO:	INPUT "streng";variabel
PARAMETRE:	"streng" — ønsket tekst variabel — den variabel, hvori data gemmes.
EKSEMPEL:	INPUT "ET TAL";X Teksten: ET TAL udskrives på skærmen, og et tal (numerisk værdi) afventes indtastet over tastaturet. Tallet gemmes i variablen X.
BEMÆRKNING:	Kommandoen kan ikke indtastes i direkte mode.

Indtast nu kommandoen NEW , der sletter et allerede eksisterende program i hukommelsen. De følgende programlinier skal vise, hvordan kommandoen INPUT anvendes:

```
10 PRINT"CIRKELBEREGNING"
20 PRINT"----------"
30 INPUT"DIAMETER ";D
40 PRINT"AREAL : ";(D/2)↑2*3.14
```

Start programmet med kommandoen RUN. I linie 30 spørges der efter diameteren for den cirkel, arealet ønskes udregnet af. En indtastning skal altid afsluttes med et tryk på RETURN-tasten. Efter strengen skal der stå et semikolon før variablen, men strengen kan også udelades:

```
10 PRINT"CIRKELBEREGNING"
20 PRINT"----------"
30 INPUT D
40 PRINT"AREAL :";(D/2)↑2*3.14
```

Det er selvfølgelig ikke nødvendigt at ændre hele programmet. Man kan

nøjes med at ændre i linie 30, idet det er det eneste sted, hvor der er forskel mellem de to programeksempler.

Bemærk at strengen før INPUT-kommandoen ene og alene er en LEDETEKST. Denne tekst kan også ligge i linien før:

```
10 PRINT"CIRKELBEREGNING"
20 PRINT"----------"
25 PRINT"DIAMETER ";
30 INPUT D
40 PRINT"AREAL : ";(D/2)↑2*3.14
```

Linie 25 tilføjes og programmet startes. Der er tilsyneladende ikke nogen forskel. De to programmer forløber fuldstændigt ens. Semikolon'et i linie 25 er meget vigtigt. Fjernes det, vil INPUT-kommandoen blive udført i linien under! Prøv selv.

Strenge kan også indlæses over tastaturet via INPUT-kommandoen. Inden vi viser dette, må vi hellere slette det program, vi allerede har i computeren. Tast NEW.

```
10 PRINT"HVAD HEDDER DU ?"
20 INPUT N$
30 PRINT"GODDAG ";N$
```

Linierne 10 og 20 kan sættes sammen til een linie:

```
10 INPUT"HVAD HEDDER DU ?";N$
20 PRINT"GODDAG ";N$
```

Disse eksempler illustrerer alle INPUT-kommandoens funktioner. Bemærk forøvrigt at programmet "kører videre" selvom vi taster RETURN, uden at have skrevet noget. En numerisk variabel vil få værdien 0 (nul) og en strengvariabel vil være tom.

Sløjfer

Vi skal nu se på en øvelse, der senere skal føre til styring af sløjfer. Der skal indlæses 5 navne via tastaturet. Navnene skal gemmes i strengen N$. Med de for hånden værende oplysninger ville programmet komme til at se således ud:

```
10 PRINT"INDTASTNING AF NAVNE"
20 PRINT"-------------------"
30 INPUT"NAVN ";N$(1)
40 INPUT"NAVN ";N$(2)
50 INPUT"NAVN ";N$(3)
60 INPUT"NAVN ";N$(4)
70 INPUT"NAVN ";N$(5)
```

I ovenstående program vil de 5 navne blive indlæst efter tur, men er fremgangsmåden ikke en smule omstændelig? Tænk engang, hvis det drejede sig om 100 navne!!!

I praksis, vil man aldrig finde en sådan opbygning af et program. Man benytter hellere en såkaldt SLØJFE. Men hvordan? Lad os se en sådan syntaks:

OBJEKT:	Sløjfestyring.
KOMMANDO:	FOR v1 = n1 to n2 STEP n3
PARAMETRE:	v1 — numerisk variabel n1 — startværdi for v1 n2 — slutværdi for v1 n3 — forhøjelse. Den værdi, hvormed v1 for hvert gennemløb forhøjes med.
EKSEMPEL:	FOR A=1 TO 20 STEP 1 Variablen A bliver for hvert gennemløb forhøjet med værdien 1, indtil værdien 20 nåes. (Efter 20 gennemløb).
BEMÆRKNING:	Er forhøjelsen kun på 1, kan parametret STEP udelades. Eksemplet ville se således ud: FOR A=1 TO 20

Alle sløjfer begynder med FOR-anvisningen. De efterfølgende kommandoer, hører til denne anvisning, og udføres tilsvarende det antal gange anvisningen foreskriver. Men hvordan kan man vide, hvorhenne sløjfen slutter?

Dertil findes der en tilhørende anvisning, hvis opgave udelukkende består i at afprøve om sløjfens slutværdi er nået. Er denne værdi endnu ikke nået, forhøjes sløjfevariablen og sløjfen gennemløbes endnu en gang. Svarer de to værdier til hinanden (n1 og n2), fortsættes programmet efter kommandoen:

OBJEKT:	Sløjfens slutning.
KOMMANDO:	NEXT v1
PARAMETER:	v1 — Sløjfevariabel
EKSEMPEL:	NEXT A Viser slutningen på den tidligere beskrevne sløjfe defineret med: FOR A=1 TO 20
BEMÆRKNING:	Er variablen v1 ikke angivet, vil kommandoen NEXT rette sig efter den sidst definerede sløjfe.

Med de to kommandoer kan vi nu programmere alle slags sløjfer. Lad os derfor vende tilbage til de 5 navne, der skulle indlæses via tastaturet og INPUT-kommandoen. Hvis vi nu lægger dette program ind i en sløjfe, kan vi nøjes med een eneste INPUT-kommando. Vi benytter index som variabel i sløjfen. Sløjfen skal køre fra 1 til 5. Vi får derfor følgende linie:

```
FOR I=1 TO 5
```

Hele det ændrede program:

```
10 PRINT"INDTAST NAVNE"
20 PRINT"----------"
30 FOR I=1 TO 5
40 INPUT"NAVN ";N$(I)
50 NEXT I
```

En særdeles elegant måde at indlæse de 5 navne på, ikke sandt? Det forudbestemte repeterende forløb er især facinerende. Først udføres de to PRINT-kommandoer.

Dernæst sættes variablen I til værdien 1.
I linie 40 spørges efter en streng, der lægges ind i strengvariablen N$(1) (Indexet retter sig jo efter variablen I). I linie 50 testes variablen I. Svarer den til værdien 5, er sløjfen bragt til slutning, og der kan fortsættes i linie 60... Men er I mindre end 5, forhøjes I med 1 og sløjfen gennemløbes igen.

Indtil nu, eksisterer der ikke nogen linie 60, men lad os udvide vort program til at skrive de indtastede navne ud på skærmen:

```
60 FOR I=1 TO 5
70 PRINT "NAVN : ";N$(I)
80 NEXT I
```

Som det ses, er det slet ikke så uoverkommelig en sag, som det så ud til at være i starten af dette afsnit!

Hvad kan man mere bruge en sløjfe til?
Vi kan f.eks. lave en forsinkelsessløjfe i et program! Her får vi kun brug for selve sløjfen og dens afslutning. Der skal ikke udføres noget egentligt, bortset fra at sløjfen skal gennemløbes. Antallet af gange, dette skal ske er vigtig for, hvorlænge PAUSEN skal vare. Vores sløjfe skal afsluttes efter 1 sekund, hvilket omtrent svarer til 1000 gennemløb!

```
FOR X=1 TO 1000:NEXT
```

Bemærk at man kan indsætte flere kommandoer i samme linie, adskilt af et kolon. I vores "NAVNEPROGRAM", vil vi indsætte en pause på 2 sekunder. Pausen skal ligge mellem indtastningen af de forskellige navne og udlæsningen af disse. D.v.s. i linie 55.

Hvordan ser denne linie ud?

```
55 FOR X=1 TO 2000:NEXT
```

Kig en gang på programmet med kommandoen LIST. Læg mærke til at linierne står pænt i nummerorden uanset, i hvilken rækkefølge, de er indtastet. Start programmet op med RUN og læg mærke til effekten af linie 55. Man kan evt. lege lidt med forskellige værdier i denne sløjfe.

Indtil nu, har vi kun arbejdet med en forhøjelse på 1 for hvert gennemløb. Skal navnene i eksemplet ikke indlæses i ordenen 1-5, men i stedet 5-1, så kan det også lade sig gøre. Sløjfens startværdi bliver så 5, og slutværdien 1. Forhøjelsen vil blive -1 d.v.s. STEP-parameter -1. Hvordan skal linierne 30 og 60 så se ud?

De fleste har vel allerede løsningen klar:

```
30 FOR I=5 TO 1 STEP -1
60 FOR I=5 TO 1 STEP -1
```

Lad os lige til sidst kigge på endnu et par øvelser i sløjfer. Hvordan kommer de følgende sløjfer til at se ud ?

	SLØJFEVARIABEL	STARTVÆRDI	SLUTVÆRDI	FORHØJELSE
A)	I	10	18	1
B)	XX	30	15.5	−0.5
C)	Y	590	1800	0.25
D)	B	1	10	0.30

Her følger løsningerne:

A) FOR I=10 TO 18 STEP 1
eller FOR I=10 TO 18

B) FOR XX=30 TO 15.5 STEP -0.5

C) FOR Y=590 TO 1800 STEP 0.25

D) FOR B=1 TO 10 STEP 0.30

Endnu en bemærkning til sløjfer:

Hvis man definerer sløjfer, der er logisk falske, f.eks. FOR I=10 TO 0, så vil en sådan sløjfe ALTID blive gennemløbet een gang.

Programmets første reaktioner

Nu, hvor vi er nået så langt i BASIC, er det blevet tid for at begynde med adressekartoteket for alvor.

Hvordan skal et sådant program starte?

Hvis man kender den type programmer, ved man også at de oftest starter med en art velkomst, der fortæller noget om, hvad det er for et program, man har startet op, og hvem, der har lavet det o.s.v.

```
 99 GOTO 1000
100 REM ====================
110 REM Program overskrift
120 REM ====================
130 CLS
140 PRINT STRING$ (40, "=")
150 LOCATE 12,2
160 PRINT "Adresse kartotek"
170 PRINT STRING$ (40, "=")
```

Efter at have indtastet og RUN'net programstykket, er det rart at vide, hvad det er som foregår!

Vi begynder med 100-120.

Her er anført, hvilken del af hele programmet, der udføres umiddelbart efter. For at computeren ikke fejlagtigt skal prøve at oversætte disse linier som værende kommandoer, må vi fortælle den, at linierne ikke er en del af programmet. Linierne betegnes som REM-sætninger, og REM er en forkortelse for det engelske ord REMARK, der betyder BEMÆRKNING. Sådanne REM-sætninger kan frit placeres hvor som helst i et program; de springes ganske enkelt over af computeren.

Den næste nye kommando, er ingen rigtig kommando, men en streng-funktion. En tegnkæde er i fagsproget en streng. Med funktionen STRING$ kan et tegn mangfoldiggøres op til 255 gange. Lad os se på beskrivelsen:

```
OBJEKT     : Fremstilling af tegnkæder
FUNKTION   : STRING$ (n,"z")
PARAMETER: n — antal tegn (0-255)
           "z" — mangfoldiggørelse af tegn
EKSEMPEL   : PRINT STRING$ (80,":")
           80 DOBBELTPUNKTER bliver fremstillet
```

Denne funktion kan vi prøve i forbindelse med PRINT-kommandoen. Idet denne kommando er ret så simpel, går vi straks videre til den næste.

Kommandoen PRINT skriver data på skærmen. Hvilket sted på skærmen, afhænger af sidste PRINT — henholdsvis CLS-kommando. For at begynde skrivningen på et bestemt (ønsket) sted på skærmen, skal BASIC indeholde følgende kommando:

OBJEKT : Bestem cursorens position
KOMMANDO : LOCATE sp, z
PARAMETER: sp — SPALTE
z — LINIE
EKSEMPEL : LOCATE 5, 10
sæt cursoren på spalte 5, linie 10

Efter enhver PRINT-kommando bliver cursorens position ændret, men det ses ikke på skærmen, idet udskrivning styres af et program.

Vil man selv bestemme hvor på skærmen udskrivningen skal begynde, så må kommandoen LOCATE sættes før PRINT-kommandoen. Hermed gives som ovenfor beskrevet besked om i hvilken spalte og i hvilken linie man vil begynde.

Et eksempel som illustration. Teksten HALLO skal skrives midt på billedskærmen. Kommandoen bliver herefter således:

```
LOCATE 18, 13: PRINT "HALLO"
```

Som vi husker kan en kommando-linie indeholde flere kommandoer, når de blot bliver adskilt med : (kolon).

OBJEKT:	Fra ASCII-kode til tegn.
FUNKTION:	CHR$(n)
PARAMETER:	n — ASCII-kode 0-255
EKSEMPEL:	FOR I=32 TO 127:PRINT I,CHR$(I):NEXT I viser alle de tegn som kan fremstilles med CPC 6128.
BEMÆRKNING:	Angives en ASCII-kode med værdien over 255, udskrives fejlmeddelelsen: "IMPROPER ARGUMENT"

OBJEKT:	Fra tegn til ASCII-kode.
FUNKTION:	ASC("z")
PARAMETER:	"z" er et tegn eller en strengvariabel, der indeholder et tegn.
EKSEMPEL:	PRINT ASC("A") viser ASCII-koden for bogstavet "A" (65) på skærmen.
BEMÆRKNING:	Er strengvariablen tom, udskrives fejlmeddelelsen: "IMPROPER ARGUMENT" Er strengvariablen længere end et tegn, er det koden for det første tegn, der udskrives.

Det er en temmelig stor mængde informationer, der først skal fordøjes. Men som sagt, arbejder computeren internt kun med de forskellige tegns ASCII-koder. Derfor forstår den også, hvis man bruger koderne i stedet for tegnene.

Hvorfor har vi så ikke brugt det sædvanlige styretegn til sletning af skærmbilledet?

Hvis man lader blikket løbe nedover et af de programmer, der er vist i et datablad, vil man se, at der er fantastisk mange styretegn i en sådan udlistning. Hvis man vil indtaste et af disse tegn, skal man først vide, hvilket styretegn det drejer sig om.

F.eks. dukker der pludseligt et reverse "S" op i et program, man er ved at indtaste. Hvad er det for et tegn? Og de 16 forskellige styretegn for farverne er bestemt ikke mindre forvirrende.

Det er derfor, vi i det følgende kun anvender ASCII-koderne.

Vil man vide, hvilken funktion et bestemt styretegn har, så fåes det med PRINT ASC("z"), hvor "z" er det tegn, man ønsker at få værdien af.

Sløjferne i linierne 140 og 160 burde ikke volde problemer. Her udskrives lighedstegnet 40 gange, hvilket svarer til een linie på skærmen.

Underprogrammer

I adressekartoteksprogrammet vil vi ofte få brug for at vise den samme tekst flere gange. Hvis vi skriver en sådan tekst lige så mange gange, som den skal bruges, vil vores program blive alt for stort og uoverskueligt. Vi gør i stedet denne tekst til en rutine; et såkaldt underprogram. Dette underprogram kan så hentes frem af programmet og udføres så ofte, det er nødvendigt. Men hvordan laver man et underprogram, og hvad er kendetegnende for et sådant? Ja det vil vi opklare med det samme.

I forrige kapitel stiftede vi bekendtskab med kommandoen GOTO, der i et program giver mulighed for springe i den rækkefølge de enkelte linier udføres. Da man imidlertid efter udførelse af et underprogram har brug for at vende tilbage til det sted i programmet, hvor maskinen fik besked på at gå til underprogrammet, så kan vi altså ikke bruge GOTO. I stedet for, har vi en lignende sammensat kommando, som ser således ud:

OBJEKT:	Kald af underprogrammer (rutiner).
KOMMANDO:	GOSUB zn
PARAMETER:	zn — linienummer hvor rutinen starter.
EKSEMPEL:	GOSUB 100 kalder et underprogram, der starter i linie 100
BEMÆRKNING:	Et underprogram startet med GOSUB SKAL afsluttes med kommandoen RETURN. Møder et program under udførelse kommandoen RETURN uden, at det er startet med GOSUB standses programafviklingen og fejlmeddelelsen UNEXPECTED RETURN vises.

Dette ser ikke så slemt ud, og har man prøvet det et par gange, er det ikke noget at tale om. Vi laver linierne 100-170 om til et selvstændigt underprogram, der nu kan kaldes efter behov fra hovedprogrammet. Som det ses af kommandobeskrivelsen så skal rutinen afsluttes med kommandoen RETURN. Derfor må vi tilføje en linie 180:

```
180 RETURN
```

Nu må vi ikke længere bare starte vores program med kommandoen RUN. Gør man det, vil man straks få udskrevet en fejlmeddelelse lydende RETURN WITHOUT GOSUB ERROR. Dette er ganske klart idet vi jo er hoppet direkte ind i et underprogram, der ikke er blevet kaldet med GOSUB. Hvordan kommer vi videre?

Betragt et øjeblik hvordan vi har organiseret vores fremtidige program's linienumre:

10-99: Her sker initialiseringen af vores program. D.v.s. alle de informationer, der er nødvendige for, at vi overhovedet kan opbygge et program skal angives. F.eks. dimensionering af tabeller og tilskrivning af variabler.

100-999: I disse linier ligger alle vore underprogrammer.

1000-????: Fra linie 1000 har vi hovedprogrammet, d.v.s. det egentlige program.

I vort tilfælde, begynder programmet i linie 10 og må således springe over underprogrammerne. Derfor må vi gøre brug af en linie 99:

```
99 GOTO 1000
```

I linie 1000 starter hovedprogrammet, og vi skriver:

```
1000 GOSUB 100
```

Så kan programmet minsandten køre igen! Det første, der sker, er at der hoppes fra linie 10 til linie 1000, hvor et underprogram kaldes; nemlig det, som laver en overskrift til hele programmet.

```
1000 REM
1010 REM Hovedprogram
1020 REM
1030 GOSUB 100
```

Menuen

Hvilke fremskridt er der mon sket i adressekartoteket? Lad os se engang:

```
 99 GOTO 1000
100 REM ====================
110 REM Program overskrift
120 REM ====================
130 CLS
140 PRINT STRING$ (40, "=")
150 LOCATE 12,2
160 PRINT "Adresse kartotek"
170 PRINT STRING$ (40, "=")
180 RETURN
```

Efter opstart af programmet, melder det sig med sit navn. Men det er ikke nok. Vi har nu kun en ganske lille del af hele programmet. Da vi med dette program åbner mulighed for flere måder at behandle data på, så må vi også vise det. Lad os overveje, hvad programmet skal kunne gøre. Der er brug for at gemme data på et eksternt lagermedium, og også at hente disse data igen. Det giver følgende valgmuligheder:

-1- HENT DATA

-2- GEM DATA

Yderligere skal vi kunne indtaste nogle adresser. Det bliver den tredie funktion:

-3- NYE ADRESSER

Der skal også være mulighed for at ændre/rette i adresserne:

-4- RETTELSER

Når vi ikke vil vide af onkel Peter mere, så skal der naturligvis også åbnes mulighed for at slette ham og hans adresse!:

-5- SLETNING

Nu kommer vi til det vigtigste: Alle disse data er til ingen nytte, hvis ikke man på en eller anden måde kan finde rundt i dem, derfor:

-6- SØG ADRESSE

Der findes ikke det program, der skal afsluttes ved at stikket skal trækkes ud af stikkontakten! Derfor har vi et sidste punkt:

-7- AFSLUT

Nu, hvor vi er klar over, hvilke muligheder, der er i programmet, så vil vi vise dem som en MENU:

```
1040 LOCATE 1,7
1050 PRINT" PROGRAMFUNKTION:"
1060 PRINT" ---------------"
1070 PRINT
1080 PRINT" -1- Hent data      "
1090 PRINT" -2- Gem data       "
1100 PRINT" -3- Opret adresse  "
1110 PRINT" -4- Ændre adresse"
1120 PRINT" -5- Slet adresse   "
1130 PRINT" -6- Print adresse  "
1140 PRINT" -7- Slut program   "
```

Når vi igen starter programmet, opdager vi, at det så langsomt FÅR PROFESSIONEL KARAKTER.

Brugeren kan selv bestemme hvilken funktion i programmet han/hun vil gøre brug af. Derfor har vi brug for en numerisk variabel, der skal kunne antage en af heltals-værdierne 1 til 7, og deraf bestemme den valgte funktion. Inden skal vi dog bruge en PRINT-sætning med indholdet: TAST FOR VALG, i sort skrift.

```
1150 LOCATE 10,18: PRINT "VALG"
1160 INPUT FUNKTION
1170 IF FUNKTION = 0 or FUNKTION › 7 then fejl$=
     "Ugyldig værdi":GOSUB200:GOTO1150
1175 GOSUB 300
1180 ON FUNKTION GOTO 5000,10000,15000,20000,25000,
     30000,35000
```

Test med IF

Hvad nu, hvis brugeren indtaster et ugyldigt tal, f.eks. 9? Det må der tages højde for i programmet, men spørgsmålet er bare, hvordan?

Vi må teste den indtastede værdi, der befinder sig i variablen F, for at se om det er et ugyldigt tal. Hvis det er ugyldigt så....:

PROBLEM:	Test af betingelser.
KOMMANDO:	IF betingelse THEN anvisning.
PARAMETER:	betingelse # Sammenligning f.eks. A=10 anvisning # Kommando, der skal udføres, hvis betingelse opfyldes. (Er sandt).
EKSEMPEL:	IF F=0 THEN GOTO 1000 Hvis variablen F har værdien 0, skal programmet fortsættes i linie 1000. Hvis ikke, fortsættes der på normal vis.
BEMÆRKNING:	Skal der efter THEN udføres en GOTO, kan GOTO ignoreres. Det sidste eksempel kunne altså også se således ud: IF F=0 THEN 1000

En betingelse kan også testes med andet end et lighedstegn. Også › (større end) og ‹ (mindre end). Kombinationer af alle tre er ligeledes tilladt. Der gives følgende muligheder:

SYMBOL	BETYDNING
=	lig
›	større end
‹	mindre end
›= , =›	større end eller lig med
‹= , =‹	mindre end eller lig med
‹› , ›‹	forskellig fra

IF-kommandoen tester først om betingelsem er sand. Er den det, udføres anvisningen efter THEN. Opfyldes betingelsen ikke (er falsk), så udføres linien umiddelbart efter test-linien.

Men IF-kommandoen kan flere ting. Flere betingelser kan kobles sammen i logisk sammenhæng, hvilket lyder mere kompliceret end, det er. Lad os se på et eksempel som hjælp:

IF A=1 OR B›12 THEN 1000

Her testes der to betingelser i logisk sammenhæng. OR betyder ELLER, og linien kan oversættes til:

Hvis A er lig med 1 ELLER hvis B er større end 12, SÅ spring til linie 1000 og fortsæt der.

Der springes altså til linie 1000 hvis enten:

A=1

ELLER

B›12

Anvisningen efter THEN ignoreres kun hvis BEGGE betingelser er falske. Endnu et eksempel:

IF A=1 AND B›12 THEN 1000

Her opfyldes betingelsen kun, hvis BÅDE A og B er sande. D.v.s. A SKAL være lig med 1 og B SKAL være større end 12.

I forbindelse med IF-kommandoen er kommandoerne: ELSE, BEGIN og BEND af betydning. Men det vil vi overlade til viderekomne.

Lad os vende tilbage til det oprindelige problem. Vi ville teste om brugeren har indtastet et tal udenfor det gyldige område: 1-7. Hvordan skal den korrekte IF-instruktion se ud?

Således:

IF F=0 OR F›7 THEN ..... etc.

Ja, hvad så? Lad os udskrive en fejlmeddelelse på den sidste linie på skærmen og springe tilbage til linie 1150.

Fejlmeddelelsen lægger vi ind i en underrutine, det kunne jo være, at vi fik brug for den senere hen.

Meddelelsen skal fremstå REVERSE i linie 24. Lad os se på underprogrammet:

```
200 REM ============
210 REM Fejlmelding
220 REM ============
230 LOCATE 1,24
240 PRINT fejl$
250 PRINT CHR$ (7)
260 FOR I=1 to 1000: NEXT I
270 LOCATE 1,24
280 PRINT STRING$ (39, " ")
290 RETURN
```

Derefter vil cursoren blive flyttet fra linie 230 til sidste linie. Linie 240 meddeler streng-fejlen i det kaldte program. Altså før denne rutine bliver kaldt fra hovedprogrammet, må fejlmeldingen lagres i strengvariablen "fejl$".

ASCII-koder

Nu en forklaring på hvad der sker i linie 250. Som vi husker, har vi hægtet kommandoen RETURN "CHR$ (13)" på, så vi kan få en automatisk funktion. Lad os kigge nærmere på det. Alle tegn bliver kodet ind i computeren med værdier fra 0 til 255. Kodningen følger en amerikansk standard, der kaldes ASCII og CPC 6128 afviger meget lidt fra denne kode. Også specielle funktioner, såsom den korte tone (lyd) i vores program kan udløses med en ASCII-kode. Til at kode et tegn efter ASCII-koden, er der to funktioner, som beskrives i det efterfølgende.

Betinget spring

For at kunne vælge den rigtige gren i programmet på baggrund af værdien af F, findes der en kommando, der kan klare dette på en simpel og overskuelig måde:

PROBLEM:	Betinget spring.
KOMMANDO:	ON v1 GOTO ln1, ln2, ln3,.....
PARAMETER:	v1 — numerisk variabel. ln1, ln2,... — linienumre, der springes til afhængig af værdien af variablen v1.
EKSEMPEL:	ON F GOTO 100,200,300,400 Er F=1 så springes der til linie 100, er F=2 så springes der til linie 200, o.s.v. Er F større end 4, springes der ikke, men fortsættes i næste linie.
BEMÆRKNING:	Er F ikke et helt tal, f.eks. 3.45, så vil funktionen rette sig efter heltalsdelen; i dette tilfælde 3.

Denne kommando er således den helt ideelle for os. Først beslutter vi os til de linienumre, hvori de enkelte programdele skal starte. Den næste linie, vil se således ud:

1180 ON FUNKTION GOTO 5000,10000,15000,20000,25000,30000,35000

Det følgende skema viser, hvilke linienumre funktionerne knytter sig til:

F	LINIE	PROGRAMDEL
1	5000	HENT DATA
2	10000	GEM DATA
3	15000	NYE ADRESSER
4	20000	RETTELSE
5	25000	SLETNING
6	30000	SØG ADRESSE
7	35000	AFSLUT

Vores opgave bliver nu at færdiggøre programdelene, også kaldet funktioner, een for een. Vi vil dog gå frem efter en mere logisk rettesnor. Vi starter med funktion 3: NYE ADRESSER.

Inden vi kan starte, må vi dimensionere en tabel til vore data. Det sker i en linie 10, da dette er en del af initialiseringen:

```
10 DIM D$(200,7)
```

Vi gør plads til 200 adresser, hvilket bør være nok; hvem har vel flere venner og bekendte?

Til mere kommerciel brug er programmet nok ikke egnet. Er man et firma med et par tusinde kunder, vil det nok være på sin plads at anskaffe sig et bedre egnet program.

Disse navne skriver vi i en tabel, hvis index modsvarer variabel-funktionen. I oversigten på forrige side, kan vi se at programdelen "NYE ADRESSER" har nr. 3. Dette nummer er således index for pladsen i tabellen. Tabellen giver vi navnet "funktion$" og dermed bliver variablen "funktion$ (3)" lig med teksten "NYE ADRESSER". Oprettelsen af tabellen hører med til forberedelserne og hører hjemme i linieområdet 0-99.

```
20 funktion$ (1) = "Hent data"
21 funktion$ (2) = "Gem data      "
22 funktion$ (3) = "Nye adresser  "
23 funktion$ (4) = "Ændre adresser"
24 funktion$ (5) = "Slette adresser "
25 funktion$ (6) = "Søg adresse   "
26 funktion$ (7) = "Afslut        "
```

Bemærk at alle strenge skal have den samme længde, hvilket vil sige at andet anførselstegn i linierne 20-80 skal stå under hinanden som vist ovenfor.

```
300 REM ================
310 REM Funktionstitel
320 REM ================
330 GOSUB 100
340 LOCATE 10,5: PRINT STRING$ (19, "*")
350 LOCATE 10,6: PRINT "*"+ Funktion$ (funktion)+ "*"
360 LOCATE 10,7: PRINT STRING$ (19, "*")
370 RETURN
```

Her kan vi se, at et underprogram også kan kalde et andet underprogram. Først søger vi den normale overskrift for programmet, og derefter fremstilles den første linie af kassen, hvori navnet på funktionen skal stå. Nu bliver cur-

soren anbragt i spalte 10, linie 6, og her kommer en stjerne og navnet på funktionen. Dette navn tages fra tabellens index. Til denne streng bliver yderligere koblet en stjerne. Linie 360 angiver rammens nederste linie.

Selfølgelig kan rutinen først anvendes, når brugeren har udvalgt en funktion, hvis nummer findes i variablen. Altså efter spørgsmålet IF. Lad os tilføje linien:

```
1175 GOSUB 300
```

Nye adresser

Nu er det alvor. Den første af de vigtige programdele skal skrives, men lad os begynde med det nemmeste:

```
15000 REM ======================
15010 REM NYE ADRESSER
15020 REM ======================
```

Så langt, så godt. Som det første udskrives det faste programhovede, derefter selve funktionens navn. Nu skal vi forhøje den tæller, der overvåger antallet af datasæt, med een. Tælleren kalder vi pointer.

```
15030 Pointer = pointer + 1
```

Hvis pointer ikke er 0, så betyder det, at der allerede befinder sig et eller flere datasæt i vort program. Tildelingen foregår automatisk.

Egentlig, så skulle vi nu bruge 7 input-linier, en for hvert datafelt. Men det kan faktisk gøres meget nemmere. Fra linie 30 gemmer vi alle felternes navne i en array: F$(1) til F$(7).

```
30 F$(1)="STILLING"
31 F$(2)="FORNAVN"
32 F$(3)="EFTERNAVN"
33 F$(4)="GADE/VEJ"
34 F$(5)="POSTNR./BY"
35 F$(6)="TELEFON NR."
36 F$(7)="EVENTUEL"
```

Feltnavnene lægges i en array i initialiseringsdelen, og kan bruges hvor som helst i hele resten af programmet. Hvor vi gør dette, fremgår af det følgende.

```
15040 PRINT
15050 FOR index = 1 TO 7
15060 PRINT felt$ (index);
15070 INPUT adresse$ (pointer, index)
15080 NEXT index
```

Kør programmet efter at disse linier er blevet tastet ind, og vælg funktion nummer 3. Det, som nu sker skyldes udelukkende den hastigt sammenflikkede sløjfe. En hel post indlæses med 4 kommandoer. Inde i sløjfen udskrives navnet på det enkelte felt. Variablen I anvendes som index for tabellens feltnavne. Umiddelbart efter feltnavnet (efter semikolon!) udskrives feltets indhold. Her bliver der indekseret to gange, idet adresserne jo som bekendt gemmes i en to-dimensional tabel (array). Det første index opnås med variablen pointer, der forhøjes med 1 for hver ny adresse. Det andet index gives af variablen I, der kan antage 7 værdier svarende til de enkelte datafelter. De 7 felter indtastes fra tastaturet en efter en.

Efter at felterne er blevet læst ind, må vi give brugeren mulighed for at gentage indtastningen. Et eller andet sted kan der jo være opstået en fejl, f.eks. kan man have taget fejl af det aktuelle felt.

```
15090 LOCATE 1,20
15100 PRINT "Data OK (j/n)";
15110 INPUT svar$
15120 IF svar$ = "j" THEN 15150
15130 IF svar$ = "n" THEN pointer = pointer -1:
      GOSUB 300: GOTO 15000
15140 Fejl$ = "Tryk »j eller »n!":
      GOSUB 200: GOTO 15090
```

Hvis data er indlæst på den rette vis, fortsættes programmet. Er alt ikke i orden, vil programafsnittet blive gennemført en gang til. Dog bliver post-tælleren formindsket med 1, idet vi ellers i linie 15130 ville få endnu en post. Får vi ikke det rigtige svar, så bliver fejlmeldingen indlæst pånyt.

Programmet fortsætter med endnu et spørgsmål.

```
15150 LOCATE 1,20
15160 PRINT "Flere adresser (j/n)"
15170 INPUT svar$
15180 IF svar$ = "j" THEN GOSUB 300: GOTO 15000
```

Her skal brugeren afgøre om der skal indtastes flere adresser eller ej. Hvis der svares ja, bliver programdelen kørt igen; men svares der nej, vender programmet tilbage til menuen.

Dermed har vi afsluttet første del af selve programmet. Man er i stand til at indlæse et antal af adresser, men de kan endnu ikke bruges til ret meget. Til slut følger hele afsnittet:

```
15000 REM ===============
15010 REM Opret adresse
15020 REM ===============
15030 Pointer = Pointer + 1
15040 PRINT
15050 FOR index = 1 TO 7
15060 PRINT felt$ (index);
15070 INPUT adresse$ (pointer, index)
15080 NEXT index
15090 LOCATE 1,20
15100 PRINT "Data OK (j/n)";
15110 INPUT svar$
15120 IF svar$ = "j" THEN 15150
15130 IF svar$ = "n" THEN pointer = pointer -1:
      GOSUB 300: GOTO 15000
15140 Fejl$ = "Tryk »j eller »n!":
      GOSUB 200: GOTO 15090
15150 LOCATE 1,20
15160 PRINT "Flere adresser (j/n)"
15170 INPUT svar$
15180 IF svar$ = "j" THEN GOSUB 300: GOTO 15000
15190 IF svar$ = "n" THEN 1000
15200 Fejl$ = "Tryk »j eller »n"
```

Rettelser

Nu vil vi give brugeren mulighed for at rette eller ændre i de enkelte datafelter i de enkelte poster. Vi vil dertil benytte funktionstasterne. Med tasterne "v" og "r" skal man henholdsvis kunne "bladre" frem og tilbage. Skal et felt ændres, må man trykke "a". RETURN bringer brugeren tilbage til menuen; dvs. at rettefunktionen er forladt.

Det lyder jo ganske smart, men vi står over for et væsentligt problem; det hele skal nemlig først programmeres! Lad os begynde med de første nemme linier:

```
20000 REM ======================
20010 REM RETTELSE
20020 REM ======================
```

De 3 første linier behøver ingen kommentarer, men derefter bliver det kritisk. Hvordan griber vi det an? Lad os først bestemme en variabel, der skal fremstå som det første index i adresse-tabellen. Vi kalder den tæller og giver den startværdien 1:

```
20030 Tæller=1
```

Nu kan vi udskrive overskriften og felternes navne:

```
20040 LOCATE 1,10
20050 FOR index = 1 TO 7
```

I de følgende linier kan vi udskrive den første adresse (tæller=1). Den består som bekendt af 7 felter, hvis navne ligger gemt i en tabel. Derfor laves endnu en sløjfe for at kunne udskrive navnene og indholdet:

```
20060 PRINT index; felt$ (index); adresse$ (tæller, index)
20070 NEXT index
```

Der bliver altså udskrevet 7 linier, som indeholder både nummeret på feltet og dels betegnelse samt indhold. Dette indhold bestemmes af index TÆLLER, der som bekendt peger på data-sætningen og det egentlige feltindex (INDEX).

Efter den første adresse er udskrevet, må brugeren igen træffe en beslutning og han/hun har 4 muligheder.

Nu bliver det nødvendigt at kunne aflæse (overvåge) tastaturet for, om der bliver trykket nogen taster. Kommandoen vi skal bruge hedder INKEY$:

PROBLEM:	Aflæsning af tastaturet.
KOMMANDO:	s = INKEY$
PARAMETER:	s — Strengvariabel, i hvilken den aktuelle tasts tegn skal lagres.
EKSEMPEL:	X$ = INKEY$ Tester for om en tast berøres. Hvis en tast trykkes ned, gemmes tegnet i variablen X$. Programmet kører derefter videre.
BEMÆRKNING:	Programmet kan ikke fortsætte førend en tast trykkes ned.

Ønsker man at opholde et program indtil en tast berøres, så er denne linie nødvendig:

```
10 X$ = INKEY$ : IF X$ = " " THEN 10
```

Denne ovenstående linie skal endelig ikke tastes ind i programmet. Den er kun vist som et eksempel. Linien bliver udført igen og igen indtil en tast berøres. Vi har brug for en tilsvarende instruktion i vores program:

```
20080 taste$ = INKEY$ : IF taste$ = " " THEN 20080
```

Når denne linie vil køre igennem, så har brugeren trykket en på taste. Vi må nu spørge om det er en af tasterne v, r, a eller om det er RETURN, der er trykket, og reagere herefter. Først prøver vi tasten RETURN, der som bekendt har ASCII-coden 13.

```
20090 IF taste$ = CHR$ (13) THEN 1000
```

Vi går altså til menuen, når RETURN trykkes. Bliver tasten "v" aktiveret, skal næste data-sætning fremkommme. Hertil forhøjer vi index-tælleren med 1, men kun hvis tælleren ikke allerede har nået sidste sætning (tæller = peger). Følgende linier klarer opgaven:

```
20100 IF taste$ = "v" AND tæller < pointer THEN
      tæller = tæller + 1 : GOSUB 300 : GOTO 20040
```

Kun når tasten "v" trykkes og den sidste adresse endnu ikke er nået, bliver tælleren sat på næste adresse som udskrives.

Ved tasten "r" er det omvendt, idet tælleren her nedskrives med 1, hvis tælleren da ikke allerede er på 1.

```
20110 IF taste$ = "r" AND tæller › 1 THEN tæller = tæller -1:
      GOSUB 300 GOTO 20040
```

Ved linie 20040 bliver adressen udskrevet med index tælleren, idet denne blev nedskrevet med 1. Sidst må vi se om taste "a" er trykket. Dermed indlæses nemlig en ændring.

```
20120 IF taste$ = "a" THEN 20140
20130 fejl$ = "fejl!": GOSUB 200 GOTO 20080
```

Linie 20130 kommer man i, hvis man er ved at komme udenfor kartoteket, eller hvis en taste uden "gyldighed" bliver trykket. Så bliver tasterne simpelthen ignoreret, der kommer en fejlmelding og der bliver henvist til linie 20080. Herfra bliver tastaturet påny udspurgt.

```
20140 LOCATE 1,18
20150 INPUT "feltnummer (1-7)"; nummer
20160 IF nummer › 0 AND nummer ‹ 8 THEN 20180
20170 Fejl$ = "Opret nummer 1-7!": GOSUB 200: GOTO 20140
20180 INPUT "Nyt indhold: "; ADRESSE$(Tæller, Nummer)
20190 GOSUB 300:GOTO 20040
```

Det var så det sidste i denne del af programmet. Først læses nummeret på det felt, der skal ændres, ind i variablen X. Derefter testes der på X for dets gyldighed (validitet). Indeholder X ikke et tal, der kan godkendes, flyttes cursoren en linie op, så der kan spørges påny. Er værdien i X iorden, indlæses feltets nye indhold og fra linie 20040 vises den nye posts indhold. Programafsnittet kan kun forlades med F7. Her følger hele udlistningen af afsnittets rettelse:

```
20000 REM ==============
20010 REM Ændre adresse
20020 REM ==============
20025 IF pointer = 0 THEN GOSUB 500: GOTO 1000
20030 Tæller = 1
20040 LOCATE 1,10
```

```
20050 FOR index = 1 TO 7
20060 PRINT index; felt$ (index); adresse$ (tæller, index)
20070 NEXT index
20080 taste$ = INKEY$: IF taste$ = "" THEN 20080
20090 IF taste$ = CHR$ (13) THEN 1000
20100 IF taste$ = "v" AND tæller ‹ pointer THEN
      tæller = tæller + 1: GOSUB 300: GOTO 20040
20110 IF taste$ = "r" AND tæller › 1 THEN tæller = tæller -1:
      GOSUB 300: GOTO 20040
20120 IF taste$ = "a" THEN 20140
20130 fejl$ = "fejl!": GOSUB 200: GOTO 20080
20140 LOCATE 1,18
20150 INPUT "feltnummer (1-7)"; nummer
20160 IF nummer › 0 AND nummer ‹ 8 THEN 20180
20170 Fejl$ = "Opret nummer 1-7!": GOSUB 200: GOTO 20140
20180 INPUT "Nyt indhold: "; ADRESSE$(Tæller, Nummer)
20190 GOSUB 300:GOTO 20040
```

Sletning

Når vi skal slette indholdet af et eller flere felter, anvender vi en lidt anden teknik end i afsnittet om rettelser. Det gælder om at være så alsidig som muligt når man skal programmere. En adresse, der skal slettes, skal angives med for- og efternavn. Findes adressen ikke ved denne metode, kan man bladre sig igennem de eksisterende adresser i funktionen RETTELSE.

Men før funktionen SLETNING kan begynde, skal det undersøges om der overhovedet eksisterer nogen data i computerens lager. Til dette job, bygger vi en rutine op, som skal ligge fra linie 500. Lad os studere den et øjeblik:

```
500 REM ============
510 REM Ingen data!
520 REM ============
530 Fejl$ ="Ingen data i maskinen!"
540 GOSUB 200
550 RETURN
```

Selvom ovenstående rutine er kort, er den alligevel temmelig meningsfyldt. Vi får brug for den i andre programafsnit senere. I linie 530 testes om der ligger data i computerens hukommelse. Variablen pointer indeholder altid numme-

ret på den sidste post og derfor også det totale antal af poster i lageret. Er variablens værdi 0 (nul), så er der ikke indlæst data. Derfor skal vi i dette tilfælde udskrive fejlmeddelelsen: INGEN DATA I LAGER! Det gør vi med rutinen fra linie 200.

Lad os begynde på dette afsnits programdel. De første linier, ved vi af erfaring, er de nemmeste.

```
25000 REM ================
25010 REM Slette adresse
25020 REM ================
25030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Fornavn: ";Fornavn$
25060 INPUT "Efternavn: ";Efternavn$
25070 Sløjfe = 1
```

Vi henter igen programhovedet i linie 25030. Derefter kalder vi rutinen fra linie 500; det er som bekendt den, der undersøger om vi overhovedet har nogen data at arbejde med. Så indlæses for- og efternavn på den adresse, der skal slettes. Vi skal nu lede efter disse oplysninger gennem hele adressetabellen. Det er ikke tilrådeligt at gøre brug af en FOR...NEXT-sløjfe her, idet vi næsten med 100% sikkerhed tvinges til at forlade sløjfen før den er afsluttet; og det kan man kun gøre i BASIC-fortolkere, der er fuldstændig fri for fejl. Det er sikkert, at dersom man tilegner sig denne form for sløjfe-programmering, så vil man på et eller andet tidspunkt uvægerligt løbe ind i en BASIC-fortolker, der ikke kan klare dette.

Vi laver vores egen selvstyrende sløjfe:

```
25080 IF Adresse$ (sløjfe,1) = fornavn$ AND
      adresse$ (sløjfe,2) = efternavn$ THEN 25110
25090 IF sløjfe < pointer THEN sløjfe = sløjfe + 1: GOTO 25080
25100 Fejl$ = "adressen ikke fundet!": GOSUB 200: GOTO 1000
25110 GOSUB 300: LOCATE 1,10
25120 FOR index = 1 TO 7
```

Startværdien af sløjfevariablen sløjfe sættes til 1. Inde i sløjfen testes det om den med sløjfe indexerede adresse stemmer med den søgte. Det andet index 2 og 3 svarer til for- og efternavn, der sammenlignes med D1$ og D2$. Stemmer navnet, fortsættes programmet i linie 25130. I linie 25100 forhøjes sløjfevariablen sløjfe med 1, hvis ikke den sidste adresse i tabellen overskrides.

Bliver sløjfen fuldendt, uden at navnet er fundet, så eksisterer navnet ikke blandt de data (adresser), der ligger i hukommelsen. Vi udskriver en meddelelse derom og vender tilbage til hovedmenuen.

Programmet skal fortsætte der, hvor der springes til, efter at den rigtige adresse (dvs. navn) er fundet. Her er det linie 25130. Hele adressen (posten) skal nu udskrives, således at brugeren kan beslutte, om den nu også virkelig skal slettes.

```
25100 Fejl$ = "adressen ikke fundet!": GOSUB 200: GOTO 1000
25110 GOSUB 300: LOCATE 1,10
25120 FOR index = 1 TO 7
25130 PRINT felt$ (index); adresse$ (sløjfe, index)
25140 NEXT index
25150 LOCATE 1,18
25160 INPUT "Slette adresse (j/n) "; svar$
25170 IF svar$ = "j" THEN 25200
25180 IF svar$ = "n" THEN 1000
25190 fejl$ = "Tryk »j eller »n!": GOSUB 200: GOTO 25150
```

Her opbygges der først et helt nyt skærmbillede (linie 25130). Derefter udskrives adresse-posten med sløjfen i linie 25135 til 25150. Her spørger vi om det er sikkert at adressen skal slettes. Skal adressen ikke slettes (der findes ubeslutsomme mennesker overalt), så springes der tilbage til menuen. Skal adressen alligevel slettes, så sker dette fra linie 25200. Lad os kaste et blik på denne sidste del af programafsnittet.

```
25200 FOR i1 = sløjfe TO pointer -1
25210 FOR i2 = 1 TO 7
25220 Adresse$ (i1,i2) = adresse$ (i1 + 1,i2)
25230 NEXT i2
25240 NEXT i1
25250 Pointer = pointer -1
25260 GOTO 1000
```

For at slette en adresse, er det ikke nok at slette det aktuelle tabel-sted. Nøjedes vi med det, ville tabellen stadig have den samme størrelse, som før vi slettede adressen. Det kan ikke være meningen. Alle adresser, der befinder sig efter den adresse, der blev slettes skal rykkes een plads tilbage. Hvis f.eks. adressen med index 5 slettes, rykkes adresse 6 til den 5. adresses plads. Hvis der befinder sig syv adresser i hukommelsen, så skal den 7. (den sidste adresse) rykkes til den 6. adresses plads.

Formindsker vi nu antallet af adresser med 1 (til 6), så har vi slettet adresse 5 på den rigtige måde.

Ved brug af denne programmeringsteknik dukker for første gang udtrykket "Nested loop" op. D.v.s. at en sløjfe befinder sig inde i en anden sløjfe. Her er det sløjfen med variablen Y, der indeholder en sløjfe med variablen I. Sløjfen Y gennemløber alle adresse-poster fra den slettede. Alle felter rykkes en plads frem. Er alle felter rykket, kan næste post bearbejdes. Den "ydre" sløjfe kører kun til Z-1, idet den sidste post i linie 25220 indexeres med Y+1. Hvis man betragter de omtalte linier, vil man forstå princippet i denne "oprykning".

Nu har vi afsluttet denne del af adressekartoteket. Det er allerede blevet et omfangsrigt program. Her følger hele udlistningen af programdelen SLETNING.

```
25000 REM ================
25010 REM Slette adresse
25020 REM ================
25030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Fornavn: ";Fornavn$
25060 INPUT "Efternavn: ";Efternavn$
25070 Sløjfe = 1
25080 IF Adresse$ (sløjfe,1) = fornavn$ AND
      adresse$ (sløjfe,2) = efternavn$ THEN 25110
25090 IF sløjfe < pointer THEN sløjfe = sløjfe + 1: GOTO 25080
25100 Fejl$ = "adressen ikke fundet!": GOSUB 200: GOTO 1000
25110 GOSUB 300: LOCATE 1,10
25120 FOR index = 1 TO 7
25130 PRINT felt$ (index); adresse$ (sløjfe, index)
25140 NEXT index
25150 LOCATE 1,18
25160 INPUT "Slette adresse (j/n) "; svar$
25170 IF svar$ = "j" THEN 25200
25180 IF svar$ = "n" THEN 1000
25190 fejl$ = "Tryk »j eller »n!": GOSUB 200: GOTO 25150
25200 FOR i1 = sløjfe TO pointer -1
25210 FOR i2 = 1 TO 7
25220 Adresse$ (i1,i2) = adresse$ (i1 + 1,i2)
25230 NEXT i2
```

```
25240 NEXT i1
25250 Pointer = pointer -1
25260 GOTO 1000
```

Udskrivning af adresser

Dette programafsnit hører ikke til de letteste. For første gang vil vi skrive noget på printeren. Men der skal også være mulighed for udskrift på skærmen. Efter at man har valgt mellem enten skærm- eller printerudskrift, skal søgeordene (nøgleordene) indtastes. Men inden da, skriver vi de første linier:

```
30000 REM ========================
30010 REM Print adressekartotek
30020 REM ========================
30030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
30040 LOCATE 1,10
30050 INPUT "printer eller skærm (p/s) "; svar$
30060 IF svar$ = "p" THEN enhed = 8: GOTO 30090
30070 IF svar$ = "s" THEN enhed = 0: GOTO 30090
30080 fejl$ = "Tryk »p eller »s!": GOSUB 200: GOTO 30040
```

Her testes det igen om, der eksisterer data i computerens hukommelse (linie 30030). Derefter spørges der om data skal udskrives på printer eller skærm. Svaret (P eller S), gemmes i strengvariablen svar$, idet denne skal bruges senere i programmet.

I programmets videre forløb skal søgeemnerne indlæses.

```
30090 GOSUB 300: LOCATE 1,10
30100 PRINT "Søgenøgle:"
30110 PRINT "----------"
30120 PRINT
30130 FOR index = 1 TO 7
30140 søg$ (index) = ""
30150 PRINT felt$ (index);
30160 INPUT søg$ (index)
30170 NEXT index
```

Linie 300 giver os et nyt skærmbillede, før man i linierne 30090 — 30100 opfordres til at indtaste søgeemnerne. Indtastningerne sker igen i en sløjfe,

men først skal de søgeemner, der evt. befinder sig i tabellen søg$ slettes. Det skyldes, at de kan indeholde søgeemner fra den forrige søgning. Emnerne indlæses i linierne 30130 til 30160. Hvis man f.eks. ville søge på alle adresser i Ringkøbing, skal der ved de første 4 felter kun tastes RETURN. Det 5. felt, der indeholder postnummer og by, udfyldes med "6950 RINGKØBING" og afsluttes med RETURN. De to sidste felter ignoreres med RETURN. Ønsker man derimod at få en oversigt over alle skorstensfejere, skal man samtidigt udfylde det 1. felt med denne stillingsbetegnelse. Flere kriterier kan således angives på samme tid. Efter indtastningen kan selve søgningen begynde.

```
30170 NEXT index
30180 FOR i1=1 TO pointer
30190 Fundet=0
30200 FOR i2 = 1 TO 7
30210 IF søg$ (i2) = "" OR søg$ (i2) = adresse$ (i1,i2) THEN
      fundet = fundet + 1
30220 NEXT i2
```

Her har vi igen en sløjfe i en sløjfe. Den yderste sløjfe (i1) gennemløber alle poster. Før den inderste sløjfe begynder at sammenligne de enkelte felter med de indtastede søgeemner, sættes en tæller til 0 (nul). Hvorfor vil man erfare senere. I sløjfe (i2), testes det om der er er indtastet et søgeord til det aktuelle felt. Hvis nej, forhøjes tælleren fundet med 1 og der springes til NEXT i linie 30220. Først i det øjeblik, hvor de to felter stemmer overens forhøjes tælleren fundet med 1.

```
30230 IF fundet ‹› 7 THEN 30300
30240 IF enhed = 0 THEN GOSUB 300
```

Her forberedes udskriften af den fundne adressepost. Kun når en post ikke stemmer overens med søgeemnerne, springes hele udskriften over. Hvis det er printeren, der er valgt, sendes der først en "tom" linie via PRINT#1. Denne kommando adskiller sig fra den normale PRINT, idet den kun retter sig mod en forudbestemt datakanal (fil). Denne kanal bestemmes ved tallet efter PRINT-kommandoen.

Blev skærmen valgt, så forberedes denne i linie 30245.

Her følger resten af program-linierne i dette afsnit.

```
30250 PRINT #enhed
30260 FOR index = 1 TO 7
30270 PRINT #enhed, felt$ (index); adresse$ (i1, index)
30280 NEXT index
30290 PRINT #enhed
30295 IF enhed = 0 THEN INPUT "Tryk RETURN"; svar$
30300 NEXT i1
30310 GOSUB 300: LOCATE 1,18
30320 PRINT "** data slut **"
30330 INPUT "Tryk RETURN"; svar$
30340 GOTO 1000
```

Disse linier drejer sig næsten kun om selve udskriften af adresseposten. Linierne 30250 til 30280 danner en sløjfe, hvori de 7 adressefelter udskrives sammen med deres respektive betegnelser. Afhængig af indholdet i variablen svar$ udskrives adressen på enten skærm eller printer.

Vælges udskrift på skærm, fortsættes søgningen først når der tastes RETURN. Er det udskrift på printer, der er aktuel, så springes der over linien hvori, der forventes et tryk på RETURN-tasten. Således skrives adresserne ud på printeren en efter en.

Efter sløjfen i1 er søgningen afsluttet. Dette oplyses med meddelelsen "SLUT PÅ DATA". Hvis brugeren nu taster RETURN, afsluttes programafsnittet og der springes til menuen igen. Hvis udskriften foregik på printer, lukkes dennes datakanal inden.

Dette var endnu en brik til opbygningen af vores kartotek. Vi mangler kun afsnittene for skrivning og læsning på diskette.

Her følger dette afsnits udlistning:

```
30000 REM ========================
30010 REM Print adressekartotek
30020 REM ========================
30030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
30040 LOCATE 1,10
30050 INPUT "printer eller skærm (p/s) "; svar$
30060 IF svar$ = "p" THEN enhed = 8: GOTO 30090
30070 IF svar$ = "s" THEN enhed = 0: GOTO 30090
30080 fejl$ = "Tryk »p eller »s!": GOSUB 200: GOTO 30040
```

```
30090 GOSUB 300: LOCATE 1,10
30100 PRINT "Søgenøgle:"
30110 PRINT "----------"
30120 PRINT
30130 FOR index = 1 TO 7
30140 søg$ (index) = ""
30150 PRINT felt$ (index);
30160 INPUT søg$ (index)
30170 NEXT index
30180 FOR i1=1 TO 7 pointer
30190 Fundet=0
30200 FOR i2 = 1 TO 7
30210 IF søg$ (i2) = "" OR søg$ (i2) = adresse$ (i1,i2) THEN
      fundet = fundet + 1
30220 NEXT i2
30230 IF fundet <> 7 THEN 30300
30240 IF enhed = 0 THEN GOSUB 300
30250 PRINT #enhed
30260 FOR index = 1 TO 7
30270 PRINT #enhed, felt$ (index); adresse$ (i1, index)
30280 NEXT index
30290 PRINT #enhed
30295 IF enhed = 0 THEN INPUT "Tryk RETURN"; svar$
30300 NEXT i1
30310 GOSUB 300: LOCATE 1,18
30320 PRINT "** data slut **"
30330 INPUT "Tryk RETURN"; svar$
30340 GOTO 1000
```

At gemme sine data

Vi nærmer os slutningen på adressekartoteket. De indtastede data går selvfølgelig tabt, når computeren afbrydes. Men vi har jo netop en diskettestation af samme grund. Men spørgsmålet er, hvordan vi griber sagen an. Som sædvanligt, kan vi starte med den nemme afdeling. Vi skal have udskrevet en overskrift og have undersøgt om der overhovedet ligger nogen data i hukommelsen.

```
10000 REM =========
10020 REM GEM data
10030 REM =========
10040 IF pointer = 0 THEN GOSUB 500: GOTO 1000
```

Vi kender allerede ovenstående linier fra tidligere. Inden vi kan gemme vore data på disketten, skal denne først lægges i diskettestationen. De følgende linier skal opfordre brugeren til at sørge for dette.

```
10050 LOCATE 1,12
10060 PRINT "læg datadisketten ind!"
10070 PRINT
10080 INPUT "Tryk derefter RETURN"; svar$
```

Disse linier skulle ikke volde nogen vanskeligheder. På nuværende tidspunkt burde et par PRINT- og INPUT-sætninger være den rene barnemad!

Men nu går det løs. Da adresserne sikkert skal lagres mere end een gang på den samme diskette, skal den gamle datafil først slettes inden vi kan skrive en ny på disketten. Det gør vi med kommandoen ERA"filnavn"-kommandoen.

Dataene skal skrives efter, at en datakanal er blevet åbnet til diskettestationen.

```
10090 a$ = "adresse": *ERA,@a$ :REM * = SHIFT @
10100 OPENOUT "adresse"
10110 PRINT #9, pointer
10120 FOR i1 = 1 TO pointer
10130 FOR i2 = 1 TO 7
10140 PRINT #9, adresse$ (i1,i2)
10150 NEXT i2
10160 NEXT i1
10170 CLOSEOUT
```

Vi åbner en ny fil med navnet "ADRESSER". Som det fremgår, åbnes en datafil. Det er faktisk nemmere end på printeren.

Det første, vi skriver på disketten er antallet af adresseposter; dette antal ligger i variablen pointer. Derefter sender vi de enkelte felter i adressetabellen ved hjælp af en "nested loop" (en sløjfe i en sløjfe), hvilket ikke længere skulle

være et ukendt begreb. Først felterne 1 til 7 i første post, derefter felterne 1 til 7 i den anden post, o.s.v.

Når alle poster er lagt over, skal dette meddeles:

```
10180 GOSUB 300
10190 PRINT "Adresserne er gemt!"
10200 FOR i=1 TO 2000: NEXT i
10210 GOTO 1000
```

Så enkelt foregår lagringen af data via diskettestationen. Men skal der være mening i tingene, så må vi også kunne hente de samme data igen!

At læse data

At indlæse data fra diskettestationen foregår næsten på samme vis, som da vi skrev dataene på disketten. Her skal der dog ikke først undersøges om der ligger data i computerens hukommelse. Brugeren skal altså først gemme sine nyindtastede data inden en adressefil indlæses.

```
5000 REM ==========
5010 REM Hent data
5020 REM ==========
5030 LOCATE 1,12
5040 PRINT "Læg venligst disketten ind!"
5050 PRINT
5060 INPUT "Tryk derefter RETURN"; svar$
```

Her udskrives der en meddelelse om at disketten skal lægges i diskettestationen.

```
5070 PRINT
5080 OPENIN "Adresse"
5090 INPUT #9, pointer
5100 FOR i1=1 TO pointer
5110 FOR i2=1 TO 7
5120 INPUT #9, adresse$ (i1,i2)
5130 NEXT i2
5140 NEXT i1
5150 CLOSEIN
```

Her åbnes filen "adresser" for læsning. Man kan se, at der læses ved at W ikke er tilføjet, hvor filen åbnes. Antallet af poster læses først, og denne værdi bruges som slutværdi i en sløjfe i1. Derved sikres det, at der ikke gøres forsøg på at læse ud over filen. Datafilen lukkes igen i linie 5160.

```
5160 GOSUB 300: PRINT "Data hentet"
5170 FOR i=1 TO 2000: NEXT i
5180 GOTO 1000
```

Dette afsnits sidste linier udskriver igen en meddelelse om, at funktionen er udført. Efter en pausesløjfe vender programmet tilbage til menuen.

Programmets afslutning

Som vi allerede nævnte i begyndelsen af dette kapitel, så bør et program ikke afsluttes ved at stikket tages ud af stikkontakten. Det er heller ikke nogen god ide at bruge RUN/STOP. Hvordan et program bør afsluttes vil vi demonstrere her.

```
35000 REM ==============
35010 REM Slut program
35020 REM ==============
35030 IF pointer = 0 THEN 35150
35040 LOCATE 1,12
35050 PRINT "Er alle data gemt (j/n)";
35060 INPUT svar$
35070 IF svar$ = "n" THEN 1000
35080 IF svar$ = "j" THEN 35100
35090 fejl$ = "Tryk »j eller »n!": GOSUB 200: GOTO 35040
```

Meningen med et afsnit kaldet AFSLUTNING begynder at dæmre. Der undersøges om alle data er lagret, og det kunne jo være, at man ved en fejltagelse var kommet til at trykke den forkerte funktionstast; det er rart at kunne fortryde!

Hvis man svarer nej til spørgsmålet, vil man komme tilbage til menuen, men hvis alt er iorden, kan der ikke ske noget ved at afslutte programmet.

```
35100 GOSUB 300: LOCATE 1,12
35110 PRINT "Programmet kan startes"
35120 PRINT "med GOTO 1000"
35130 PRINT "uden tab af data !!"
35140 PRINT
35150 END
```

Programmet afsluttes med en meget vigtig besked om, at det kan startes igen uden at data går tabt, med kommandoen GOTO 1000. Hvis man starter op med RUN vil alle data i variabler gå tabt.

Det var det! Vi har konstrueret et veldokumenteret og gennemtænkt adressekartoteksprogram. Vi håber at læseren har lært en del hen ad vejen, og har fået lyst til selv at gå igang med andre typer kartoteksprogrammer. Man kan f.eks. starte med at ændre dette program, så det passer til ens egne behov. Er der afsnit, man ikke har forstået alt i, kan man gå tilbage og gennemgå det igen.

Har man fået lyst til at vide mere om BASIC-programmering, så er der et righoldigt udvalg på markedet. Har man lyst til at gå i dybden med de fantastiske muligheder, der er med AMSTRAD 6128 (grafik, lyd, spil, o.s.v.), så anbefaler jeg DATA BECKER bøgerne.

Her følger hele adressekartoteksprogrammet:

```
 10 DIMAdresse$(200,8)
 20 Funktion$(1)=" HENT DATA      "
 21 Funktion$(2)=" GEM DATA       "
 22 Funktion$(3)=" OPRET ADRESSE  "
 23 Funktion$(4)=" RET ADRESSE    "
 24 Funktion$(5)=" SLET ADRESSE   "
 25 Funktion$(6)=" PRINT ADRESSE  "
 26 Funktion$(7)=" SLUT PROGRAM   "
 30 felt$(1)="Fornavn   "
 31 felt$(2)="Efternavn "
 32 felt$(3)="Gade      "
 34 felt$(4)="By        "
 35 felt$(5)="Tlf       "
 36 felt$(6)="Bem       "
 37 felt$(7)="Bem       "
 99 GOTO 1000
```

```
100 REM ====================
110 REM Program overskrift
120 REM ====================
130 CLS
140 PRINT STRING$ (40, "=")
150 LOCATE 12,2
160 PRINT "Adresse kartotek"
170 PRINT STRING$ (40, "=")
180 RETURN
200 REM ============
210 REM Fejlmelding
220 REM ============
230 LOCATE 1,24
240 PRINT fejl$
250 PRINT CHR$ (7)
260 FOR I=1 to 1000: NEXT I
270 LOCATE 1,24
280 PRINT STRING$ (39, " ")
290 RETURN
300 REM ================
310 REM Funktionstitel
320 REM ================
330 GOSUB 100
340 LOCATE 10,5: PRINT STRING$ (19, "*")
350 LOCATE 10,6: PRINT "*"+ Funktion$ (funktion)+ "*"
360 LOCATE 10,7: PRINT STRING$ (19, "*")
370 RETURN
500 REM ============
510 REM Ingen data!
520 REM ============
530 Fejl$ ="Ingen data i maskinen!"
540 GOSUB 200
550 RETURN
1000 REM ==============
1010 REM Hoved program
1020 REM ==============
1030 GOSUB 100
1040 LOCATE 1,7
1050 PRINT" PROGRAMFUNKTION:"
1060 PRINT" ----------"
```

```
 1070 PRINT
 1080 PRINT" -1- Hent data      "
 1090 PRINT" -2- Gem data       "
 1100 PRINT" -3- Opret adresse  "
 1110 PRINT" -4- Ændre adresse"
 1120 PRINT" -5- Slet adresse   "
 1130 PRINT" -6- Print adresse  "
 1140 PRINT" -7- Slut program   "
 1150 LOCATE 10,18: PRINT "VALG"
 1160 INPUT FUNKTION
 1170 IF FUNKTION = 0 or FUNKTION › 7 then fejl$=
      "Ugyldig værdi":GOSUB200:GOTO1150
 1175 GOSUB 300
 1180 ON FUNKTION GOTO 5000,10000,15000,20000,25000,
      30000,35000
 5000 REM ==========
 5010 REM Hent data
 5020 REM ==========
 5030 LOCATE 1,12
 5040 PRINT "Læg venligst disketten ind!"
 5050 PRINT
 5060 INPUT "Tryk derefter RETURN"; svar$
 5070 PRINT
 5080 OPENIN "Adresse"
 5090 INPUT #9, pointer
 5100 FOR i1=1 TO pointer
 5110 FOR i2=1 TO 7
 5120 INPUT #9, adresse$ (i1,i2)
 5130 NEXT i2
 5140 NEXT i1
 5150 CLOSEIN
 5160 GOSUB 300: PRINT "Data hentet"
 5170 FOR i=1 TO 2000: NEXT i
 5180 GOTO 1000
10000 REM ==========
10020 REM GEM data
10030 REM ==========
10040 IF pointer = 0 THEN GOSUB 500: GOTO 1000
10050 LOCATE 1,12
10060 PRINT "læg datadisketten ind!"
```

```
10070 PRINT
10080 INPUT "Tryk derefter RETURN"; svar$
10090 a$ = "adresse": *ERA,@a$ :REM * = SHIFT @
10100 OPENOUT "adresse"
10110 PRINT #9, pointer
10120 FOR i1 = 1 TO pointer
10130 FOR i2 = 1 TO 7
10140 PRINT #9, adresse$ (i1,i2)
10150 NEXT i2
10160 NEXT i1
10170 CLOSEOUT
10180 GOSUB 300
10190 PRINT "Adresserne er gemt!"
10200 FOR i=1 TO 2000: NEXT i
10210 GOTO 1000
15000 REM ==============
15010 REM Opret adresse
15020 REM ==============
15030 Pointer = Pointer + 1
15040 PRINT
15050 FOR index = 1 TO 7
15060 PRINT felt$ (index);
15070 INPUT adresse$ (pointer, index)
15080 NEXT index
15090 LOCATE 1,20
15100 PRINT "Data OK (j/n)";
15110 INPUT svar$
15120 IF svar$ = "j" THEN 15150
15130 IF svar$ = "n" THEN pointer = pointer -1:
      GOSUB 300: GOTO 15000
15140 Fejl$ = "Tryk »j eller »n!":
      GOSUB 200: GOTO 15090
15150 LOCATE 1,20
15160 PRINT "Flere adresser (j/n)"
15170 INPUT svar$
15180 IF svar$ = "j" THEN GOSUB 300: GOTO 15000
15190 IF svar$ = "n" THEN 1000
15200 Fejl$ = "Tryk »j eller »n":
      GOSUB 200: GOTO 15150
20000 REM ==============
```

```
20010 REM Ændre adresse
20020 REM ===============
20025 IF pointer = 0 THEN GOSUB 500: GOTO 1000
20030 Tæller = 1
20040 LOCATE 1,10
20050 FOR index = 1 TO 7
20060 PRINT index; felt$ (index); adresse$ (tæller, index)
20070 NEXT index
20080 taste$ = INKEY$: IF taste$ = "" THEN 20080
20090 IF taste$ = CHR$ (13) THEN 1000
20100 IF taste$ = "v" AND tæller ‹ pointer THEN
      tæller = tæller + 1: GOSUB 300: GOTO 20040
20110 IF taste$ = "r" AND tæller › 1 THEN tæller = tæller - 1:
      GOSUB 300: GOTO 20040
20120 IF taste$ = "a" THEN 20140
20130 fejl$ = "oprette fejl!": GOSUB 200: GOTO 20080
20140 LOCATE 1,18
20150 INPUT "feltnummer (1-7)"; nummer
20160 IF nummer › 0 AND nummer ‹ 8 THEN 20180
20170 Fejl$ = "Opret nummer 1-7!": GOSUB 200: GOTO 20140
20180 INPUT "Nyt indhold: "; ADRESSE$(Tæller, Nummer)
20190 GOSUB 300:GOTO 20040
25000 REM ===============
25010 REM Slette adresse
25020 REM ===============
25030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Fornavn: ";Fornavn$
25060 INPUT "Efternavn: ";Efternavn$
25070 Sløjfe = 1
25080 IF Adresse$ (sløjfe,1) = fornavn$ AND
      adresse$ (sløjfe,2) = efternavn$ THEN 25110
25090 IF sløjfe ‹ pointer THEN sløjfe = sløjfe + 1: GOTO 25080
25100 Fejl$ = "adressen ikke fundet!": GOSUB 200: GOTO 1000
25110 GOSUB 300: LOCATE 1,10
25120 FOR index = 1 TO 7
25130 PRINT felt$ (index); adresse$ (sløjfe, index)
25140 NEXT index
25150 LOCATE 1,18
25160 INPUT "Slette adresse (j/n) "; svar$
```

```
25170 IF svar$ = "j" THEN 25200
25180 IF svar$ = "n" THEN 1000
25190 fejl$ = "Tryk »j eller »n!": GOSUB 200: GOTO 25150
25200 FOR i1 = sløjfe TO pointer - 1
25210 FOR i2 = 1 TO 7
25220 Adresse$ (i1,i2) = adresse$ (i1 + 1,i2)
25230 NEXT i2
25240 NEXT i1
25250 Pointer = pointer - 1
25260 GOTO 1000
30000 REM ========================
30010 REM Print adressekartotek
30020 REM ========================
30030 IF pointer = 0 THEN GOSUB 500: GOTO 1000
30040 LOCATE 1,10
30050 INPUT "printer eller skærm (p/s) "; svar$
30060 IF svar$ = "p" THEN enhed = 8: GOTO 30090
30070 IF svar$ = "s" THEN enhed = 0: GOTO 30090
30080 fejl$ = "Tryk »p eller »s!": GOSUB 200: GOTO 30040
30090 GOSUB 300: LOCATE 1,10
30100 PRINT "Søgenøgle:"
30110 PRINT "----------"
30120 PRINT
30130 FOR index = 1 TO 7
30140 søg$ (index) = ""
30150 PRINT felt$ (index);
30160 INPUT søg$ (index)
30170 NEXT index
30180 FOR i1=1 TO pointer
30190 Fundet=0
30200 FOR i2 = 1 TO 7
30210 IF søg$ (i2) = "" OR søg$ (i2) = adresse$ (i1,i2) THEN
      fundet = fundet + 1
30220 NEXT i2
30230 IF fundet ‹› 7 THEN 30300
30240 IF enhed = 0 THEN GOSUB 300
30250 PRINT #enhed
30260 FOR index = 1 TO 7
30270 PRINT #enhed, felt$ (index); adresse$ (i1, index)
30280 NEXT index
```

```
30290 PRINT #enhed
30295 IF enhed = 0 THEN INPUT "Tryk RETURN"; svar$
30300 NEXT i1
30310 GOSUB 300: LOCATE 1,18
30320 PRINT "** data slut **"
30330 INPUT "Tryk RETURN"; svar$
30340 GOTO 1000
35000 REM ==============
35010 REM Slut program
35020 REM ==============
35030 IF pointer = 0 THEN 35150
35040 LOCATE 1,12
35050 PRINT "Er alle data gemt (j/n)";
35060 INPUT svar$
35070 IF svar$ = "n" THEN 1000
35080 IF svar$ = "j" THEN 35100
35090 fejl$ = "Tryk »j eller »n!": GOSUB 200: GOTO 35040
35100 GOSUB 300: LOCATE 1,12
35110 PRINT "Programmet kan startes"
35120 PRINT "med GOTO 1000"
35130 PRINT "uden tab af data !!"
35140 PRINT
35150 END
```

Kapitel 6: FARVE OG GRAFIK

Ofte er det en hjemmecomputers kapacitet m.h.t. farveudvalg og grafik, der er det vigtigste kriterium for en køber. Det er netop disse to ting, der gør CPC'en til et fordelagtigt køb indenfor prislejet. For sine surt tjente penge får en køber mere end 27 farver, af hvilke, afhængigt af grafikkens opløsning 2, 4 eller 16 farver at jonglere med. Opløsningen når op på 640 gange 200 punkter, hvilket ikke er hverdagskost for denne prisklasse.

Grafik modes

CPC 6128 arbejder med 3 forskellige grafikopløsninger, som kan udvælges med den allerede omtalte "MODE" kommando. Der eksisterer således en umiddelbar sammenhæng mellem tegnstørrelse og grafikopløsning. Den følgende tabel skulle lette overblikket:

KOMMANDO	TEGN PR. LINIE	OPLØSNING GRAFIK	OPLØSNING TEGN	ANTAL FARVER
MODE 0	20	160 × 200	32 × 8	16
MODE 1	40	320 × 200	16 × 8	4
MODE 2	80	640 × 200	8 × 8	2

Sammenhængen mellem tegnstørrelse og opløsning ses tydeligt i denne oversigt. Det ses også, at et tegn i MODE 2 består af 8*8 punkter. I MODE 1 fordobles denne matrix til 16*8 punkter. De fleste computere som CPC6128 sammenlignes med, benytter ligeledes en matrix på 8*8 punkter til tegnene, men det er ved 40 tegn pr. linie. Således har man på 6128 en dobbelt så høj opløsning.

Hvert tegns 8 billedpunkter gemmes i en adresse i hukommelsen. Hele skærmen består af 640 * 200 punkter (128.000), der ialt bruger 16.000 adresser. Af hele hukommelsen på 64 K bytes bruges der ca. 16 K bytes til grafikhukommelse. Ved de andre opløsninger bruges den samme mængde hukommelse, idet det frie antal bytes anvendes til at angive de enkelte punkters farver. Almindelig tekst gemmes også i grafikhukommelsen, hvilket ikke er sædvane. Men fordelen er, at tekst og grafik kan kombineres på skærmen.

Skærmfarver

Efter at computeren er blevet tændt, toner de velkendte farver fra MODE 1 frem; blåt og lysegult forudsat at man har koblet en farvemonitor til sit anlæg. Dette er imidlertid kun 2 ud af de ialt 27 mulige farver. I den følgende tabel kan man se en oversigt over farverne:

NUMMER	FARVE	NUMMER	FARVE
0	SORT	13	HVID
1	BLÅ	14	PASTEL BLÅ
2	LYS BLÅ	15	ORANGE
3	RØD	16	ROSA
4	MAGENTA	17	PASTEL MAGENTA
5	LYS VIOLET	18	LYS GRØN
6	LYS RØD	19	HAV GRØN
7	PURPUR	20	LYS TURKIS
8	LYS MAGENTA	21	LIME GRØN
9	GRØN	22	PASTEL GRØN
10	TURKIS	23	PASTEL TURKIS
11	HIMMELBLÅ	24	LYS GUL
12	GUL	25	PASTEL GUL
		26	LYS HVID

Hver farve har fået sin egen kode. Opstartbilledets standardfarver har så koderne 1 for blå og 24 for lys gul.

Kantfarver

Det er ikke kun baggrunds- og tegnfarverne, der kan bestemmes. Også billedets kant kan tilegnes forskellige farver. Kanten er den del af skærmbilledet, der ikke kan nåes af cursoren. I opstartsbilledet kan kanten ikke ses, da den har samme farve som baggrunden. Men ved hjælp af en kommando, kan man ændre kulør på den. Her følger som sædvanlig en beskrivelse af kommandoen:

PROBLEM:	Sætning af kantfarve
KOMMANDO:	BORDER farve1,farve2
PARAMETER:	farve1 — Farvenummer angives kun, hvis farve2 — der løbende skiftes mellem farve1 og farve2
EKSEMPEL:	BORDER 0 ændrer den aktuelle kantfarve til sort. BORDER 0,13 får kanten til at blinke (skift mellem sort og hvidt).
BEMÆRKNING:	De 27 farver kan angives i alle MODEs

Nu kender vi således den første kommando til at gøre brug af de 27 farver. Lad os derfor straks ændre kantfarven til skriftfarven (nummer 24). Kommandoen lyder:

BORDER 24

Nu træder rammen, der før var skjult, tydeligt frem. Vi kan lade kanten antage alle farverne efter tur, men da vi er dovne, så laver vi et program, der kan vise alle farverne efter tur.

Programmet ser således ud:

```
10 FOR farve=0 to 26
20 BORDER farve
30 if INKEY$="" then 30
40 NEXT farve
```

Hvis programmet ikke siger læseren noget, er det sikkert fordi, man er sprunget over afsnittet om indføring i BASIC. Der vil af og til blive vist eksempler på programmer til underbygning af grafikkommandoer.

Men nu tilbage til programmet. De enkelte farvekoder lægges ind i en løkke. Koderne tilegnes kommandoen i linie 20. Som ved alle parametre, kan man angive sine koder ved hjælp af variabler. Linie 30 afventer et tryk på tastaturet inden næste farve bestemmes. Det er vel nok den nemmeste måde at vise CPC 6128's farveregister på. Hvis man i stedet for en farvemonitor har en grøn monitor, så vil man se farvevalget som forskellige intensiteter af samme farve. Der er således ingen grund til ikke at udforske grafikkens forunderlige verden.

Af kommandobeskrivelsen fremgår det, at man kan få billedkanten til at blinke. Der skal anvendes 2 farvekoder adskilt af et komma efter kommandoen BORDER. Kanten vil så hele tiden skifte mellem de to farver. Lad os prøve det, idet vi finder de to farver med den største kontrast, sort og lys hvid. Kommandoen lyder:

BORDER 0,26

Da det sikkert ikke kan anbefales, at man arbejder videre med denne blinkende ramme, så slår vi blinket fra med:

BORDER 1

En måde at anvende funktionen på, er i forbindelse med fejlmeldinger i programmer. Brugeren gøres opmærksom på en opstået fejl på en meget effektiv måde, der ikke er til at overse.

Indstilling af blink-frekvens

CPC 6128 lader ikke meget tilbage at ønske. Således kan man også sætte frekvensen for blink af kant og skærmbillede. Det sørger den følgende kommando for:

PROBLEM:	Sætning af blinkfrekvens
KOMMANDO:	SPEED INK tid1,tid2
PARAMETER:	tid1 — tid for 1. farve. tid2 — tid for 2. farve. Kan sættes i intervaller af 0.02 sek.
EKSEMPEL:	SPEED INK 50,100 Fryser den første farve i 1 sekund og den anden i 2 sekunder.
BEMÆRKNING:	tid1 og tid2 må antage værdier mellem 1 og 255, hvilket svarer til en forsinkelse på fra 0.02 til 5.1 sekunder.

Denne kommando bestemmer altså en blinkfrekvens for rammen. Her kommer program som eksempel på kommandoens brug:

```
10 INPUT"1. Farve (0-26) : ";farve1
20 INPUT"Forsinkelse (1-255) : ";delay1
30 INPUT"2. Farve (0-26) : ";farve2
40 INPUT"Forsinkelse (1-255) : ";delay2
50 BORDER farve1,farve2
60 SPEED INK delay1,delay2
```

Ændring af tegnfarve

Standardfarven for tegnene er som tidligere nævnt lys gul (24). Man er ikke nødvendigvis bundet til at benytte denne farve. Den følgende kommando kan ændre tegnfarven:

PROBLEM:	Ændring af tegnfarven
KOMMANDO:	PEN #n,farvereg
PARAMETER:	#n — skærmnummer kan udelades. farvereg. — 0-15
EKSEMPEL:	PEN 1 Ændrer tegnfarven til den farve, som er indeholdt i farveregister 1
BEMÆRKNING:	Farveregisteret ændres med kommandoen INK

Som bekendt, kan man ikke anvende alle 27 farver på samme tid. Alt efter mode, kan man gøre brug af 2, 4 eller 16 farver af gangen. I 40 tegns mode kan man benytte 4 farver og disse bestemmes af kommandoen INK. Lad os se nærmere på denne kommando:

PROBLEM:	Farvevalg
KOMMANDO:	INK farvereg,farve1,farve2
PARAMETER:	farvereg # Farveregister (0-15) farve1 # Farvekode (0-26) farve2 # angives når der skal skiftes mellem denne og farve1 (BLINK)
EKSEMPEL:	INK 1,3 — Sætter farven RØD i register 1 INK 3,1,9 — Lader farven fra register 3 blinke blåt og grønt.

Da der maksimalt kan vælges mellem 16 farver (MODE 0), findes der også 16 farveregistre 0 til 15. Ønsker man f.eks. at ændre tegnfarven til rød, må man sætte værdien 3 for rød i et farveregister. Vælger vi register 1, lyder kommandoen:

INK 1,3

Nu kan kommandoen PEN ændre tegnfarven i det register, der er valgt. Kommandoen betyder: Skift til den farve, der er angivet i farveregister 1. I dette register satte vi før den røde farve. Farveregistrene er ved computerens opstart allerede belagt med farver. De enkelte farver fremgår af nedenstående tabel:

FARVEREGISTER	MODE 0	MODE 1	MODE 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1,24	20	1
15	16,11	6	24

Da udvalget af farver er ret så begrænset i MODE 1 og MODE 2, vil farverne blive gentaget flere gange. Ved farveregister 0 i MODE 1, berøres f.eks. både farveregister 4, 8 og 12. Der kan ikke vælges mellem flere end 4 farver i MODE 1 og 2 farver i MODE 2.

Kommandoen INK bevirker også, at alle tegn, der er blevet udskrevet med farven, indeholdt i registeret ændres samtidigt. Et eksempel:

Standardtegnfarven efter opstart findes i register 1. Hvis man efter opstart ændrer farven i dette register, vil al tekst, der havde den tidligere farve ændres. Indtast f.eks.:

INK 1,13

Alle tegn på skærmen bliver øjeblikkeligt hvide. Skriver man med en farve fra register 2, så har en ændring i register 1 ingen indflydelse.

Standardfarven til baggrunden hentes i register 0. Hvis man vælger en tegnfarve fra register 0, bliver baggrunds- og tegnfarverne identiske, hvilket resulterer i, at man ikke længere kan skelne eller læse en evt. tekst på skærmen. Prøv dette:

PEN 0

Nu forsvinder cursoren. Vi arbejder baglæns ved at taste:

PEN 1

Farveregister 1 er altså standardregisteret for tegnfarven. Da et register altid er reserveret til baggrunden, kan vi i MODE 1 gøre brug af 3 forskellige skriftfarver. Det følgende program udskriver 3 linier i forskellige farver:

```
10 MODE 1
20 INK 1,12:INK 2,9:INK 3,13
30 PEN 1:PRINT"GUL SKRIFT"
40 PEN 2:PRINT"GRØN SKRIFT"
50 PEN 3:PRINT"HVID SKRIFT"
```

Til de 3 skriftfarver bruger vi registrene 1 til 3. Her sættes i linie 20 farverne gul, grøn og hvid i de 3 registre. Til slut udskrives de 3 linier i hver sin farve.

I et farveregister kan der placeres 2 farver, hvilket som bekendt resulterer i, at der skiftes mellem dem. Reset computeren til opstartstilstand med SHIFT-CTRL-ESC. Så ændrer vi tegnfarve (register 1) til sort/hvid.

INK 1,0,13

Straks begynder alle tegn at blinke på skærmen. Indtaster man nogle tegn, vil disse også blinke. For at forhindre det, må vi ty til:

PEN 2

Nu hentes tegnfarven i register 2. Man kan standse blinkeriet med farven 24 i register 1.

INK 1,24

Her følger igen et lille demonstrationsprogram:

```
10 MODE 1
20 INK 2,1,26
30 INK 3,26,1
40 PEN 2:PRINT"FREM OG TILBAGE"
50 PEN 3:PRINT"FREM OG TILBAGE"
60 PEN 1
```

Ændring af baggrundsfarven

Nu har vi beskæftiget os med kant- og tegnfarver. Der er imidlertid også en kommando for ændring af baggrundsfarven:

PROBLEM:	Ændring af baggrundsfarven
KOMMANDO:	PAPER #n,farvereg.
PARAMETER:	#n — skærmnummer kan udelades. farvereg. — 0-15
EKSEMPEL:	PAPER 2 Ændrer tegnfarven til den farve, som er indeholdt i farveregister 2
BEMÆRKNING:	For at ændre farve på hele baggrunden, skal man taste CLS efter PAPER-kommandoen

Kommandoen PAPER ligner PEN, idet begge kommandoer skal have angivet en farve i et register. Standardregister for baggrundsfarve er register 0. Farveregister 1 må ikke vælges, idet dette jo er tilegnet tegnfarven, og fordi vi igen ville få usynlig skrift.

Vi skal bruge kommandoen PAPER. Baggrunden skal være lys rød. Denne farve er standard i register 3. Følgende kommando skal bruges:

```
PAPER 3
```

Men det er ikke hele baggrunden, der bliver lys rød, men kun baggrunden på de efterfølgende tegn. For at ændre hele baggrunden, skal der tastes CLS for sletning af skærmen.

Man kan også få baggrunden til at blinke, hvis der ved PAPER-kommandoen angives et register, der indeholder 2 farver. Vi vil lade baggrunden blinke sort/hvid efter at computeren er reset'et med SHIFT-CTRL-ESC. Dertil skal den følgende række af kommandoer anvendes:

```
INK 3,0,13
PAPER 3
CLS
```

Frekvensen, hvormed der blinkes, kan ændres med SPEED INK kommandoen. Skal der skiftes farve hvert sekund, sker det med:

```
SPEED INK 50,50
```

Husk at enheden i denne kommando er 0.02 sekund (50 x 0.02 = 1 sekund).

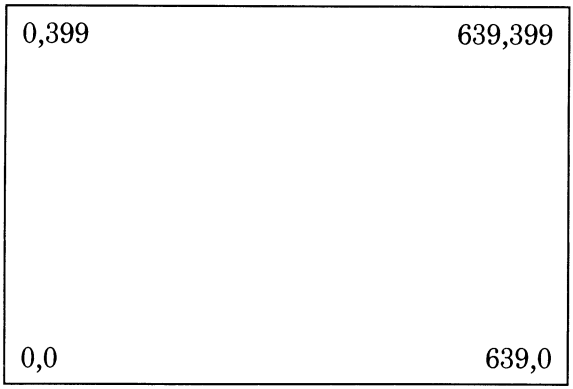
Til sidst endnu et forklarende eksempel:

```
10 MODE 1
20 INK 2,0,13
30 INK 3,13,0
40 SPEED INK 50,50
50 PEN 2
60 PAPER 3
70 CLS
80 PRINT "POSITIV OG NEGATIV"
```

Højopløsning i grafik

Som vi allerede har set, så kan man skifte mellem 3 forskellige opløsninger i grafik; hver med et bestemt antal farver til rådighed. Til at lave figurer med findes der nogle kommandoer, der udelukkende vises med henblik på højopløsning i grafik.

Husk at den højopløste grafik består af 640 x 200 punkter i 2 farver; en baggrund og en punktfarve. For at kunne bestemme de enkelte punkter, må man betragte skærmen som et koordinatsystem. Ligesom den kendte LOCATE kommando, der flytter cursoren til et bestemt sted på skærmen, så bruges metoden med de to værdier også i grafik. Den første værdi bestemmer positionen på X-aksen og den anden positionen på Y-aksen. Opløsningen på 640 × 200 punkter svarer ikke til koordinatsystemets enheder. Y-værdierne går ikke fra 0 til 199, men derimod fra 0 til 399, hvorved Y-værdierne knytter sig parvis til det samme punkt (0 og 1, 2 og 3,...o.s.v.). De fire hjørner har følgende koordinater:



Når man er blevet fortrolig med dette system, kan man gå i gang med de første kommandoer, der styrer grafikken.

Tegning af punkter

PROBLEM:	Sætning af et punkt.
KOMMANDO:	PLOT x,y
PARAMETER:	x — X-akse positioner (0-639) y — Y-akse positioner (0-399)
EKSEMPEL:	PLOT 0,0 Sætter et grafikpunkt i skærmens nederste venstre hjørne

Hvis man behersker koordinatsystemets principper, skulle der ikke være nogen vanskeligheder forbundet med at forstå og anvende grafikkommandoerne. Fremgangsmåden med at sætte de enkelte punkter bliver dog hurtig kedelig og møjsommelig, hvis man ikke tager andre kommandoer i brug. Det er således interessant at sætte punkter ved hjælp af løkker. Det er eksempelvis ingen sag at tegne en ret linie på den måde, men der findes faktisk en speciel kommando til at lave rette linier med. Ønsker man at indøve brugen af PLOT-kommandoen, er det imidlertid en god ide at bruge den sammen med egentlig programmering, når man skal have bestemte figurer eller former frem på skærmen.

Her følger et program, der "tegner" en række af punkter så de tilsammen danner en lodret streg i den yderste venstre billedkant. Der tegnes nedefra og opefter:

```
10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y
40 NEXT Y
```

Programmet skulle ikke volde nogen vanskeligheder. Lav i stedet et program, der tegner en ramme rundt om hele skærmen. Vent med at se vort forslag indtil læseren selv har forsøgt sig.

```
10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y:PLOT 639,Y
40 NEXT Y
```

```
50 FOR X=0 TO 639
60 PLOT X,0:PLOT X,639
70 NEXT X
```

Hvor mange løkker har læseren brugt? De to parallelle linier kan tegnes samtidigt. Sådanne tricks kommer helt af sig selv med tiden. Tegningen af de lodrette linier kan "krympes" lidt. Husk på, at hvert punkt har to værdier på Y-aksen. Vi kan altså halvere løkken, hvis vi samtidig forhøjer step-værdien til 2. Linien kommer dermed til at se således ud:

```
20 FOR Y=0 TO 399 STEP 2
```

Overvej altid i programmering, om der ikke er en nemmere måde at komme om ved tingene på. Det betaler sig både i oversigt, tid og hukommelse.

Tegning af en cirkel

Da CPC 6128 ikke er i besiddelse af en kommando til at tegne cirkler med, må vi selv hjælpe til. Hvert enkelt punkt skal beregnes og sættes. Vi skal benytte de følgende formler:

$X = r * \cos(a)$
$Y = r * \sin(a)$

hvor r er radius og a er vinklen. Kombineres dette med en løkke, der tæller en vinkel fra 1 til 360 grader, kan hvert enkelt punkt i cirkelen sættes på skærmen. Her er selve programmet:

```
10 MODE 2
20 X=320:Y=200:R=100
30 DEG
40 FOR a=1 TO 360
50 PLOT x+r*cos(a),y+r*sin(a)
60 NEXT a
```

I linie 20 bestemmes centrum og radius i den ønskede cirkel. Kommandoen DEG betyder, at de følgende beregninger skal udføres i nygrader. Den alternative kommando hertil er RAD, der regner i almindelige grader. 90 grader = 100 nygrader. Den følgende linie danner starten på løkken, hvori cirkelens 360 grader skal dannes. Ændrer man på start og slutværdier, vil resultatet blive cirkeludsnit. I linie 50 finder vi den egentlige formel. De fundne værdier

skal endelig adderes med centrum (X og Y). Til sidst finder vi afslutningen på løkken.

Ved hjælp af denne rutine kan man lave cirkler, selvom man ikke har forstået den matematiske sammenhæng. Vil man lave elipser, må man finde de støvede skolebøger frem — man kan ikke undvære hverken matematik eller geometri, når det drejer sig om grafikprogrammering.

Tegning af rette linier

PROBLEM:	En ret linie
KOMMANDO:	DRAW x,y
EKSEMPEL:	DRAW 639,0 Tegner en ret linie fra den grafiske cursor til koordinaterne.

Det eneste spørgsmål som denne kommando giver anledning til at stille er: Hvad er en grafik-cursor?

Ved opstart står den "usynlige" grafik-cursor på koordinatpunktet 0,0. Enhver grafikkommando påvirker denne cursor. Hvis man sætter et punkt i 100,200 så vil cursoren også befinde sig der. En efterfølgende DRAW-kommando tager udgangspunkt i samme koordinatsæt. Liniens sidste punkt vil tilsvarende være det aktuelle punkt for cursoren.

Der er dog mulighed for at flytte den grafiske cursor til et vilkårligt punkt. Kommandoen virker således:

PROBLEM:	Placering af grafisk cursor
KOMMANDO:	MOVE x,y
EKSEMPEL:	MOVE 200,250 Flytter den grafiske cursor til punktet 200,250

Lad os vende tilbage til problemet med at lave en lodret linie i venstre side af skærmen. Før løste vi opgaven med PLOT-kommandoen. Vi behøver kun 2 kommandoer (ved opstart kun 1 kommando):

```
10 MODE 2
20 MOVE 0,0
30 DRAW 0,399
```

Umiddelbart efter opstart er det ikke nødvendigt at angive MOVE-kommandoen, idet den grafiske cursor automatisk indtager positionen 0,0.

Nu mangler vi kun at lave den ramme, vi tidligere skabte med PLOT:

```
10 MODE 2
20 MOVE 0,0:REM NEDERSTE VENSTRE HJØRNE
30 DRAW 0,399:REM TIL ØVERSTE VENSTRE HJØRNE
40 DRAW 639,399;REM TIL ØVERSTE HØJRE HJØRNE
50 DRAW 0,639:REM TIL NEDERSTE HØJRE HJØRNE
```

Dette program forklarer sig selv. Forbindelse mellem mange punkter er ikke længere noget problem. Det giver interessante resultater at kombinere tegning af linier med løkker. Se selv:

```
10 MODE 2
20 FOR I=0 TO 399 STEP 100- 30 MOVE 399,I
40 DRAW 0,399-I
50 NEXT I
```

Et simpelt program med sjov effekt. Vi tager eet til!

```
10 MODE 2
20 MOVE 0,0
30 FOR I=0 TO 399 STEP 10
40 DRAW 399,I
50 DRAW 399-I,399
60 DRAW 0,399-I
70 DRAW I,0
80 NEXT I
```

Er det ikke beundringsværdigt, at der kan opstå sådanne kunstværker ved at indtaste et par kommandoer? Prøv at lave noget fantasigrafik. Selvom det måske ikke lykkes de første par gange, så er det utroligt spændende. Det

svære er at omsætte de billedlige ideer, man har i sit hovede, til tørre matematiske tal. Det kan faktisk blive til en hel sport!

Relativ tegning

Dette kapitel skal afsluttes med en variant over de beskrevne kommandoer. Ved PLOT og MOVE er den grafiske cursor ikke relevant, idet vi arbejder med absolutte punkter. Tilføjer vi et R til kommandoerne, så opnår man en anden virkning:

PLOTR x,y
DRAWR x,y
MOVER x,y

Disse kommandoer anvender ikke absolutte koordinater, men relative værdier. Kort sagt så adderes den til kommandoen tilegnede x- eller y-værdi med de tilsvarende værdier for den grafiske cursor. Et eksempel følger:

Den grafiske cursor befinder sig på punktet 200,100. Kommandoen PLOTR 50,50 sætter et punkt på positionen 250,150.

Det følgende eksempel tegner en kvadrat med 10 gange 10 punkter. Øverste venstre hjørne ligger i punktet angivet af variablerne X og Y.

```
10 MODE 2
20 X=200:Y=100
30 MOVE X,Y
40 DRAWR 10,0:REM 10 TIL HØJRE
50 DRAWR 0,10:REM 10 OP
60 DRAWR -10,0:REM 10 TIL VENSTRE
70 DRAWR 0,-10:REM 10 NED
```

Bemærk at man sagtens kan anvende negative tal sålænge den relative værdi er positiv. Det er nødvendigt at bruge negative tal for at bevæge sig "baglæns", d.v.s. flytte punkter ned eller til venstre i koordinatsystemet.

Dette er afslutningen på grafikkapitlet for begyndere. Man kan finde mere grafik i computerens manual eller i andre bøger fra DATA BECKER og NCS.

Kapitel 7: LYD

En computer egner sig fantastisk godt til at lave lyde med. Processoren, den egentlige arbejdshest i en computer, styres af en clock-frekvens. Denne er, alt efter computer, ca. 1-4 MHz (millioner svingninger i sekundet). Det letteste ville være at slutte en højttaler direkte til clockgeneratoren, men mennesker kan ikke høre så høje toner og højttalere kan ikke frembringe dem. Frekvensen skal altså sænkes væsentligt inden der bliver et hørbart resultat. Dette opnår man ved at dele clockfrekvensen op inden den føres til højttaleren.

Dette var så at sige den første generation af lyd på computere. Da tonerne, som kan sammenlignes med lyden af et elektronisk vækkeurs kimen, ikke var tilfredsstillende, så gennemgik computernes lydkredsløb en lang udvikling. Resultatet er at mange hjemmecomputere er udstyret med hele synthesizers, hvis volumen er nedbragt til en enkelt chip. Tonerne, der kan skabes med disse, minder i forbavsende grad om de "ægte" musikinstrumenters lyde.

Lydchippen i 6128 kan ikke betegnes som en egentlig synthesizer. Der mangler nogle væsentlige egenskaber, så som bølgeform (sinus, firkant og savtak etc.) og de klangbestemmende filtre. Det egentlige princip med spændingsafhængig frekvens kan heller ikke understøttes, idet en 6128 uvægerlig må gøre brug af en opdeling (opsplitning) af de faste frekvenser.

CPC 6128's tonegenerator råder over 3 stemmer. Der kan således "lyde" 3 toner samtidigt. Yderligere er der en støjgenerator til rådighed, hvormed der kan skabes utallige effekter.

Al lyden fra CPC får lov til at klinge ud gennem den indbyggede højttaler. Lydstyrken kan reguleres på CPC's højre side.

Har man lyst til at høre sine egne kompositioner i stereo, så er der mulighed for at slutte hjemmecomputeren til et stereoanlæg. Bag på computeren findes der et stik mærket I/O. Udgangen føres til forstærkerens TAPE-, TUNER- eller AUX-indgang.

I det følgende skal vi beskæftige os med de BASIC-kommandoer, der har med lyden at gøre. Der er ikke mulighed for at gennemgå alle de komplexe muligheder i denne bog, men computerens manual giver også tips og oplysninger. Desuden kan man finde bøger fra DATA BECKER om lyd.

SOUND-kommandoen

EMNE:	Lydegenskaber
KOMMANDO:	SOUND a,b,c,d,e,f,g
PARAMETER:	a — kanalstatus (i dette kapitel altid 1) b — dele, bestemmer frekvensen (0-4095) Frekvens = 125000/dele c — Varighed (-32768 til +32767 i enheder af 0.01 sekund). d — Volumen (0-15) e — Envelope for lydstyrke (0-15) f — Envelope for tonen (0-15) g — Støjbillede (0-15)
EKSEMPEL:	SOUND 1,1000,100,7,0,0,0 skaber en tone på 125 Hz af 1 sekunds varighed og lydstyrke 7

Da denne kommando er særdeles omfattende, vil vi ikke anvende alle parametre. Programmering af Envelope vil sprænge denne bogs rammer. Det kendskab man får via dette kapitel, er kun beregnet som et solidt grundlag til at gå i dybden med senere.

Tonehøjde

Vi går i gang med de parametre, der er af interesse for os. Første parameter "a" angiver en kanalstatus, der blandt andet gør det muligt at anvende de 3 stemmer samtidigt. I de følgende eksempler vil vi kun anvende stemme 1.

"b" angiver tonehøjden. Værdien bruges som deler (splitter) til grundfrekvensen på 125000 Hz, der således divideres med "b". Deleren kan antage værdier mellem 0 og 4095. Det giver et frekvensområde på 30 til 125000 Hz. Det følgende program lader en tone på 3000 Hz klinge ud til en tone på 300 Hz. Det tjener til at vise de forskellige frekvenser.

```
10 FOR DELER = 4.1 TO 416 STEP 0.1
20 PRINT INT(125000/DELER)
30 SOUND 1,DELER,1,7,0,0,0
40 NEXT DELER
```

Funktionen INT fjerner decimalerne fra resultatet af divisionen i parantesen, da vi, i dette tilfælde, ikke kan bruge dem til noget. Som det ses, kan alle parametre også udskiftes med variabler ved SOUND-kommandoen.

Ændring af tonehøjde i en løkke giver uanede effekter. En sirene kan på nem vis simuleres med det følgende program:

```
10 FOR DELER = 120 TO 200
20 SOUND 1,DELER,2,7,0,0,0
30 NEXT DELER
40 FOR DELER = 200 TO 120 STEP -1
50 SOUND 1,DELER,2,7,0,0,0
60 NEXT DELER
70 GOTO 10
```

Denne sirene, der minder om en amerikansk politibils kan afbrydes med tasten ESC (trykkes 2 gange). For de,der ikke er blevet træt af sirenen endnu, er der her en variant:

```
 5 INPUT"STARTDELER ";A
 6 INPUT"SLUTDELER ";B
 7 INPUT"TEMPO (1-10)";T
10 FOR DELER = A TO B
20 SOUND 1,DELER,T,7,0,0,0
30 NEXT DELER
40 FOR DELER = B TO A STEP -1
50 SOUND 1,DELER,T,7,0,0,0
60 NEXT DELER
70 GOTO 10
```

Også dette program kan stoppes med ESC.

Frekvenser bliver til musik

Musikinstrumenters toner er ikke glidende i overgangen, men skabes i trin, der er behagelige for et menneskeøre. Lad os lave et grundlag for den videre forståelse. En såkaldt skala består af 12 toner.

Et klaviatur består af flere sådanne afsnit. Disse afsnit kaldes oktaver. Den første oktav indeholder de dybeste toner og den sidste oktav indeholder de højeste toner. I hvilke trin er de enkelte toner så inddelt? Den internationale

beregning af frekvenser for tonerne tager udgangspunkt i kammertonen "A" fra den mellemste oktav. Tonens frekvens er 440 Hz. Lad os høre tonen lidt. Hvilken deler skal vi bruge for at skabe frekvensen?

```
DELER = 125000/FREKVENS
```

440 Hz giver 284.090909. Decimalerne kan vi ikke bruge. Der er ikke nogen hørlig forskel, hvad enten de er der eller ej. Kammertonen kommer til at se således ud:

```
SOUND 1,284,100,7,0,0,0
```

Hvordan beregnes de andre toner ud fra grundtonen? Den nødvendige formel ser således ud:

```
FREKVENS = 440 * (2↑(OKTAV+(N-10)/12)

OKTAVER  # FRA -3 TIL 4
N        # TONENS POSITION I SKALAEN (C=1, A=12)
```

Den matematiske herkomst er uvæsentlig. Det er vigtigere at vide, hvordan formlen skal bruges. Hvis det f.eks. er C, der skal findes i den underste oktav, så findes frekvensen på følgende vis:

```
440 * (2↑(-3+(1-10)/12)) = 32.7031957 Hz
```

Til frembringelse af tonen skal vi bruge en deler:

```
125000/32.7031957 = 3822.25643
```

Deleren 3822 giver "C" i den nederste oktav. På den måde kan vi lave et program, der gennemløber en skala:

```
10 INPUT"OKTAV (-3 TIL 4);OKTAV
20 FOR TONE=1 TO 12
30 FREKVENS=440*(2↑(OKTAV+(TONE-10)/12))
40 DELER=INT(125000/FREKVENS)
50 SOUND 1,DELER,20,7,0,0,0
60 NEXT TONE
```

Man bliver mere og mere bidt af at skabe lyd og musik med en tonegenerator. Kronen på værket med programmet: KLAVER FOR BEGYNDERE, der forvandler CPC'en til et musikinstrument.

```
 10 MODE 0
 20 DIM TONE(255)
 30 TONE$="Q2W3ER5T6Y7UI9O0P"
 40 FOR I=1 TO 17
 50 INDEX=ASC(MID$(TONE$,I,1))
 60 TONE(INDEX)=I
 70 NEXT I
 80 CLS
 90 PRINT" KLAVER FOR"
100 PRINT" BEGYNDERE"
110 LOCATE 1,5 21
120 PRINT" 2 3 567 9 0"
130 PRINT
140 PRINT"Q W E R T Y U I O P"
150 A$=INKEY$:IF A$="" THEN 150
160 IF TONE(ASC(A$))=0 THEN 150
170 TONE-TONE(ASC(A$))
180 FREKVENS = 440*(2↑(TONE/12))
190 DELER=INT(125000/FREKVENS)
200 SOUND 1,DELER,30,7,0,0,0
210 GOTO 150
```

Et lille program med stor virkning! Det er op til læseren at foretage forbedringer når programmet er blevet gennemskuet.

Støj

Ved gennemgangen af SOUND-kommandoen nævnte vi kort muligheden for at bruge parameter til frembringelse af støj. Denne er placeret på sidste position og kan indeholde værdierne fra 1 til 15 når den skal aktiveres. Med SOUND-kommandoen kan man både lave toner og støj alt efter værdien i parameterne "b" og "g". Her er en tabel:

SOUND a, b, c, d, e, f, g			
Parameter		Resultat	
"b"	"g"	Tone	Støj
0	0	nej	nej
1-4095	0	ja	nej
0	1-15	nej	ja
1-4095	1-15	ja	ja

Det fremgår heraf, at SOUND-kommandoen kan lave toner og støj samtidigt. Støjen kan ved hjælp af parameteren gøres lys (1) og dyb (15). Lyt til det følgende program:

```
10 FOR R=1 TO 15
20 SOUND 1,0,20,7,0,0,1
30 NEXT R
```

Støjændringerne er særdeles hørbare. Tonen "slukkes" ved at sætte den anden parameter til 0.

Ved hjælp af støjen kan man opbygge forskellige effekter., bl.a. vind, skud og explosioner. Det følgende program skal efterligne et lokomotiv.

```
10 SOUND 1,0,10,7,0,0,15
20 SOUND 1,0,5,7,0,0,0
30 GOTO 10
```

Lokomotivet kan bringes til standsning med ESC. Støjen (0.1 sek.) efterfølges af en pause (0.05 sek.). Husk at 3. parameter bestemmer tonelængden i intervaller af 0.01 sek.

Det sidste eksempel skal efterligne et skud udløst af et tryk på en tast. Programmet kan afbrydes med ESC:

```
10 A$=INKEY$:IF A$="" THEN 10
20 FOR V=7 TO 1 STEP -1
30 SOUND 1,0,10,V,0,0,15
40 NEXT V
50 GOTO 10
```

Prøv selv evnerne som støjmester. BASIC-kommandoerne skulle ikke længere volde nogen problemer.

Kapitel 8: DISKETTEDREVET

Som man sikkert ikke har overset, så er CPC6128 forsynet med et indbygget diskettedrev. Dette apparat er beregnet til at gemme programmer og data på disketter. Den simple LOADning og SAVEning af programmer volder ingen problemer. I kartoteksprogrammet har vi også gemt data på en diskette. Alt dette skal vi ikke komme nærmere ind på i dette kapitel. I stedet vil vi demonstrere andre specielle egenskaber med eksempler.

Disketterne

AMSTRAD's diskettedrev adskiller sig en del fra de hidtidige drev m.h.t. størrelsen på de tilhørende disketter. Efter at standarden i nogle år har været 5 1/4" disketter er 3 1/2" efterhånden ved at skabe sig en ny standard på markedet (AMIGA, 520ST, MacIntosh er alle udrustet med denne størrelse). Kun AMSTRAD har sit eget disketteformat på 3", der ikke findes på nogen anden maskine.

CPC-disketterne har dog een fordel frem for 3 1/2"-standarden. 3"-disketterne kan vendes og bruges på den anden side også. Man har på den måde hele 2 × 180 KB til rådighed. Disketterne er fysisk indelt i 40 spor. Hvert spor er igen opdelt i 9 sektorer. På hver af disse 360 sektorer kan der lagres 512 bytes. Der er imidlertid ingen grund til bekymring. Operativsystemet, eller rettere DOS'en skal nok klare alt dette arbejde med organisering af data.

Formattering

Inden disketten kan anvendes som lagringsmedie, skal den formatteres. Ved formatteringen lægges sektorerne ind på diskettens spor.

Her følger fremgangsmåden ved formattering af en diskette:

1. Læg den medfølgende (CP/M) systemdiskettes side A i drevet.
2. Kald CP/M-operativsystemet med kommandoen: ICPM
   (Stregen fremkommer ved tryk på SHIFT+@)
3. Load programmet til formattering fra disketten med kommandoen DISCKIT3.
4. Tryk på funktionstast F7 for valg af formattering.
5. Tast F6 for valg af dataformat.
6. Læg disketten, der skal formatteres i drevet.
7. Tryk Y-tasten.
8. Efter endt formattering trykkes en tast.
9. Læg igen systemdisketten i drevet.
10. Forlad hjælpeprogrammet med F0.
11. RESET computeren (SHIFT-CONTROL-ESC).

Dette er en relativ omfangsrig procedure, der dog ikke kan undværes. Efter vellykket gennemløb, er man nu i besiddelse af en formatteret diskette, der kan lagre data.

Filnavne

Ved indtastning af navne til programmer og andre filer på disketten, skal man iagttage følgende:

FILNAVNE må højst bestå af 8 bogstaver.

Der er 3 ekstra bogstaver til rådighed, hvis man adskiller de 8 og de 3 med et punktum. Normalt lagrede BASIC-programmer får automatisk hægtet endelsen .BAS på. Et eksempel:

SAVE "TEST" Programmet (lagres) opføres i CATaloget under navnet TEST.BAS

SAVE"TEST.DEM" Programmet (lagres) opføres i CATaloget under navnet TEST.DEM

Indholdsfortegnelsen

Indtast nogle BASIC-linier og gem dem som program under navnet TEST-CAT.

SAVE "TESTCAT"

Indholdsfortegnelsen kan efter behov hentes frem på skærmen med ordren:

CAT

Prøv at taste CAT. Resultatet skulle gerne se således ud:

DRIVE A: user 0

TESTCAT .BAS 1K

177K free

Man kan se, at programmet ikke fylder mere end 1 Kb, og at der er 177 Kb plads tilbage på disketten. For at loade programmet skal man bruge en af de to viste metoder herunder:

LOAD "TESTCAT" eller LOAD "TESTCAT.BAS"

Programmer som ASCII-filer

Igennem hele bogen er der med mellemrum dukket to udtryk op, der endelig ikke må forveksles:

PROGRAM

og

FIL

Hvori ligger forskellen? Programmer ligger lagret i både computerens hukommelse og på disketten i komprimeret form. Kommandoer har f.eks. deres egne koder, idet en kommando som PRINT ellers ville fylde hele 5 bytes (1 byte for hver bogstav). For at forhindre dette spild, har hver kommando fået tilegnet et kodenr. (0-255), der kun optager plads i en adresse i hukommelsen. Det kan sammenlignes med at computerens bogstaver internt bearbejdes som tal (ASC og CHR$).

CPC'en giver mulighed for at gemme programmer i "udskrevet" form (ASCII-format), d.v.s. i normal tekstformat på diskette. Her følger kommandoens beskrivelse:

EMNE:	LAGRING AF PROGRAMMER I NORMAL TEKSTFORMAT
KOMMANDO:	SAVE"navn",A
PARAMETER:	navn — Programmets navn
EKSEMPEL:	SAVE"adresser",A gemmer programmet "adresser" i ASCII-format
BEMÆRKNING:	Det gemte program kan indlæses som data i et andet program.

Det er ikke sikkert, at læseren kan forestille sig, hvad man kan bruge denne lagringsmetode til. Men senest ved tekstbehandling vil man lære at sætte pris på den. Alle programmer, der er lagret som ASCII-filer, kan indlæses i et tekstbehandlingsprogram og manipuleres på lige fod med forretningsbreve. Det er især en god ting, hvis man skal have lagt en programlistning ind i en tekst, som i denne bog.

Det følgende eksempel skal bevise, at et program kan indlæses som data. Indtast demonstrationsprogrammet herunder:

```
10 REM ==========================
20 REM PROGRAM SOM DATA
30 REM ==========================
40 PRINT"PROGRAMMET ER SLUT!"
```

Gem programmet på disketten med følgende kommando:

SAVE "DEMO.ASC",A

Nu vil vi indlæse programmet som en række data, og vise det på skærmen. Til dette formål har vi et lille program:

```
10 OPENIN "DEMO.ASC"
20 FOR I=1 TO 4
30 INPUT #9,A$
40 PRINT A$
50 NEXT I
60 CLOSEOUT
```

Indtaster man dette program og starter det op, vil man til sin overraskelse se, at det første program kommer frem på skærmen!

Beskyttede programmer

Måske har læseren prøvet at LOADe et købeprogram, for derefter at lade det LISTe på skærmen. Er det ikke lykkedes, så skyldes det at programmet er beskyttet. Man kan selv beskytte sine programmer. Den følgende beskrivelse viser fremgangsmåden:

EMNE:	Beskyttelse af disketteprogrammer
KOMMANDO:	SAVE"navn",P
PARAMETER:	navn — Programmets navn
EKSEMPEL:	SAVE"secret",P gemmer programmet "secret" i en sådan form at det ikke længere kan LISTes.
BEMÆRKNING:	Husk at gemme en kopi, der ikke er beskyttet, idet et beskyttet program ikke kan bringes tilbage til normalt format igen.

Den beskrivelse siger næsten alt, hvad der er brug for til lagring af beskyttede programmer. Lad os prøve kommandoen i praksis. Indtast de følgende BASIC-linier:

```
10 REM ==============
20 REM SECRET
30 REM ==============
40 PRINT"INDTAST ADGANGSKODE";
50 INPUT A$
60 IF A$><"XY12" THEN NEW
70 PRINT"O.K."
```

Gem programmet på diskette med kommandoen:

SAVE"secret.p",P

Hent nu programmet tilbage med:

LOAD"secret.p"

Start programmet med RUN og..... hov vil det ikke køre?

Beskyttede programmer kan ikke startes med LOAD og RUN, men kun med:

RUN "secret.p"

Efter at programmet er hentet fra disketten, startes det automatisk op. Kan koden nu alligevel brydes? Prøv at standse programmet med ESC, husk at trykke 2 gange. Programmet standser minsandten, men der er ingen grund til at glæde sig for tidligt. Prøv at taste LIST...

Der kommer intet frem, fordi der ikke længere er noget program i computerens hukommelse. Det blev slettet i samme øjeblik, der blev trykket på ESC. Der er altså ingen måde at knække programmet på. Og dog, er man erfaren nok og kender sin maskine meget intimt, så er der en lille mulighed. Der findes som bekendt ikke nogen programmer, der ikke kan brydes. Lad os sammenfatte her til sidst:

Et "P" efter SAVE-kommandoen beskytter programmet mod LISTning.

Programmet kan kun LOADes og startes med kommandoen RUN"programnavn".

Standses programmet med ESC, slettes det.

Man kan ikke lave proceduren baglæns, når man har beskyttet sit program.

Lagring af hukommelsesblokke

Hele CPC'ens hukommelse består af 2 gange 64 Kbyte, d.v.s. 2 x 65535 bytes (tegn). Ønsker man at gemme dele af hukommelsen, kan man anvende følgende kommando:

EMNE:	Lagring af hukommelsesblokke
KOMMANDO:	SAVE"navn",B,fra,til,start
PARAMETER:	navn — Programmets navn fra — startadresse (0-65535) til — slutadresse start # startadresse (ved maskinsprog)
EKSEMPEL:	SAVE"rutine.BIN",49152,65535 Gemmer indholdet af skærmen.
BEMÆRKNING:	Hukommelsesudsnittet skal loades med kommandoen RUN"navn"

Denne kommando er frem for alt af betydning for lagring af maskinkodeprogrammer. Dette område er forbeholdt erfarne programmører, der bryder sig om BASIC's hastighed.

Kommandoen kan også anvendes til lagring af skærmgrafik. I så fald lægges hele skærmbilledets lager (49152 — 65535) ned på disketten.

Programsammenlægning (MERGE)

Alle, der programmerer samler i tidens løb en lang række små-rutiner sammen, der kan bruges i mange forskellige programmer. Et omfangsrigt bibliotek af underprogrammer forkorter udviklingen af nye programmer væsentligt. Men hvordan lægges et underprogram til et hovedprogram? 6128'erens store ordforråd har også plads til en kommando for dette:

EMNE:	Sammenlægning af programmer (MERGE)
KOMMANDO:	MERGE"navn"
PARAMETER:	navn — Programmets navn
EKSEMPEL:	MERGE "UPRO1" LOAD'er programmet "UPRO1" og lægger det efter det program, der befinder i computeren i forvejen.
BEMÆRKNING:	Ved sammenfaldende linienumre er det det statiske programs linier, der viger.

Kommandoens funktioner kan ses ud fra denne oversigt:

PROGRAM I HUKOMMELSE	PROGRAM PÅ DISK	RESULTAT
10 REM P1 20 REM P1	5 REM P2 15 REM P2	5 REM P2 10 REM P1 15 REM P2 20 REM P1
10 REM P1 20 REM P1	5 REM P2 10 REM P2	5 REM P2 10 REM P2 20 REM P1

Her ses det tydeligt, at de to programmers linier sorteres på en sådan måde, at det færdige resultat svarer til forskrifterne for et programs linienummerering. Møder to linier med det samme nummer hinnanden, så er det linien, der befinder sig i computerens hukommelse, der må vige. Der kan af gode grunde kun eksistere linier med forskellige numre.

Passer nummereringen ikke for to programmer, der skal flettes sammen, skal de først passes til. En vigtig kommando i den henseende er

RENUMBER

Her til slut følger et lille eksempel på MERGE-kommandoens anvendelse. Indtast det følgende program:

```
10 REM PROGRAM 2
25 REM PROGRAM 2
30 REM PROGRAM 2
45 REM PROGRAM 2
```

Dette program skal gemmes på en diskette med kommandoen:

```
SAVE "PROG2"
```

Slet hukommelsen med NEW og indtast det program, hvori der skal MERGES:

```
10 REM PROGRAM 1
20 REM PROGRAM 1
30 REM PROGRAM 1
40 REM PROGRAM 1
50 REM PROGRAM 1
```

De to programmer flettes sammen med kommandoen:

```
MERGE "PROG2"
```

Computeren forholder sig som ved en normal LOADning, bortset fra at et allerede eksisterende program i hukommelsen ikke slettes. Indtaster vi til sidst kommandoen LIST, får vi det følgende program:

```
10 REM PROGRAM 2
20 REM PROGRAM 1
25 REM PROGRAM 2
30 REM PROGRAM 2
40 REM PROGRAM 1
45 REM PROGRAM 1
50 REM PROGRAM 2
```

De dobbelte linier 10 og 30 blev erstattet af linierne fra programmet på disketten. Alle andre linier blev korrekt indordnet. Kommandoen MERGE er en stor hjælp til opbygning af nye programmer bestående af "gamle" rutiner og underprogrammer.

Sletning af filer

Programmer, der ikke længere anvendes bør ikke optage unødig plads på de relativt kostbare disketter. CPC's DOS, den såkaldte AMSDOS indeholder en kommando for sletning af filer på disketter. *Alle AMSDOS-kommandoer starter med den lille lodrette streg "I". Tegnet fremkommer ved tryk på SHIFT og @.* Men nu til kommandoen:

EMNE:	Sletning af filer
KOMMANDO:	IERA,"navn"
PARAMETER:	navn — filnavn (programmets navn)
EKSEMPEL:	IERA,"secret.p" sletter programmet "secret.p" på disketten.

Prøv kommandoen et par gange og slet til sidst MERGE-programmet fra det forrige eksempel:

IERA,"prog2.bas"

Vær opmærksom på at endelsen .BAS altid skal med ved denne kommando.

RENAME af filer (navneforandring)

Skulle man komme ud for, at et program skal have et nyt navn, er det praktisk at kende AMSDOS-kommandoen dertil:

EMNE	Ændring af filnavn
KOMMANDO:	IREN,"nytnavn","gammeltnavn"
PARAMETER:	nytnavn — Filens nye navn gammeltnavn — Filens tidligere navn
EKSEMPEL:	IREN,"DEMO.BAS","DEMO01.BAS" ændrer filnavnet DEMO.BAS til DEMO01.BAS

Her skal man ligeledes huske altid at tilføje endelsen på en fil.

AMSDOS og operativsystemet CP/M stiller mange flere kommandoer til rådighed, end der er nævnt i dette kapitel. For meget af det gode kan ofte forvirre i stedet for at forklare. Derfor har vi kun medtaget de vigtigste kommandoer til FILBEHANDLING på diskette.

Kapitel 9:
ENDNU FLERE KOMMANDOER

Hvis læseren har nået dette kapitel med succes, er man sikkert parat til at gå i gang med nye udfordringer, og vi skal være de sidste til at stå i vejen. Her konfronteres læseren med kommandoer, hvis omfang er relative. Alle kommandobeskrivelser indeholder uddybende eksempler.

Brug af joystick

Hvis man for en gangs skyld betragter sin CPC bagfra, vil man i højre side finde et stik, der næsten er blevet obligatorisk på alle hjemmecomputere. Det er den såkaldte Joystick-tilslutning, der gør at brugeren kan styre spilleprogrammer optimalt.

Da aflæsningen af et Joysticks stilling er ret simpelt på en CPC vil vi præsentere kommandoen her:

EMNE:	Aflæsning af JOYSTICK
KOMMANDO:	JOY (n)
PARAMETER:	n — nummer på joystick (0 eller 1)
EKSEMPEL:	PRINT JOY(0) Viser koden for Joystickbevægelse på skærmen.
BEMÆRKNING:	Der er tale om følgende koder, som er vist analog med de tilhørende bevægelser:

5	1	9		
4		8	Affyringsknap	1 = 32
6	2	10		2 = 16

Bemærk også at der kan aflæses to Joysticks, selv om der kun er et til stede. Der kan kun tilsluttes to Joysticks, hvis det er de originale AMSTRAD-Joystick eller tilsvarende lavet af andre firmaer til CPC. Er det nok med et, kan man bruge næsten alle eksisterende på markedet (f.eks. ATARI eller COMMODORE 64/128). Dette Joystick skal aflæses med JOY(0).

Lad os skrive et program, der kan vise os brugen af et Joystick. De føromtalte koder kan bruges til at styre visse rutiner i et program.

```
10 PRINT JOY(0)
20 GOTO 10
```

Hvis man starter det korte program, vil koderne, der gælder for de respektive retninger dukke op på skærmen når Joystick'et benyttes.

Med funktionen JOY(n) kan man lave et program, der kan benytte et Joystick til at tegne på skærmen med. Affyringsknappen tjener som "viskelæder".

```
 10 MODE 2
 20 CLS
 30 X=320:Y=200
 40 PLOT X,Y
 50 A=JOY(0)
 60 IF A=4 AND X>0 THEN X=X+1:GOTO 40
 70 IF A=8 AND X<639 THEN X=X-1:GOTO 40
 80 IF A=1 AND Y<399 THEN Y=Y+1:GOTO 40
 90 IF A=2 AND Y>0 THEN Y=Y-1:GOTO 40
100 IF A=16 THEN CLS
110 GOTO 40
```

Programmet er simpelt opbygget og det burde ikke volde de store vanskeligheder at forstå det. Det er op til enhver selv at bygge videre. En passende opgave ville være at lade programmet tegne diagonale linier. Her skal man bruge yderligere fire IF-linier, der skal teste X- og Y-værdierne.

Tilfældige tal

Praktisk taget alle BASIC-begyndere anvender tilfældige tal i deres første "spilleprogrammer". Uden de tilfældige tal, ville der ikke være meget held med i spillet. CPC har en funktion, der genererer tilfældige tal mellem 0.000000001 og 0.999999999.

EMNE:	Tilfældige tal
KOMMANDO:	RND(1)
EKSEMPEL:	PRINT RND(1) Finder et tilfældigt tal og skriver det ud på skærmen.
BEMÆRKNING:	Det fundne tal ligger i intervallet 0.000000001 og 0.999999999

Det er en kommando, der er let at anvende. Det er straks vanskeligere, hvis tallet skal ligge indenfor et andet interval. Så skal det tilfældige tal forberedes. Skal man f.eks. bruge et tal mellem 0 og 999, så skal det gøres på denne måde:

```
10 TAL=RND(1)
20 PRINT INT(TAL*1000)
```

Det tilfældige tal multipliceres altså med 1000. Intervallet for mulige tilfældige tal ligger så mellem 0.000001 og 999.999999. Decimalerne fjernes med INT.

En anden vanskelighed er, hvis det ønskede tal ikke skal starte ved 0, men ved et andet tal. Den følgende formel kan skabe et hvilket som helst interval for tilfældige tal:

```
INT (RND(1)*((B+1)-A))+A
```

A = 1. gyldige tal.
B = 2. gyldige tal.

Skal der eksempelvis bruges tal mellem 1 og 6, opbygges formlen på følgende vis:

```
INT(RND(1)*((6-1)+1))+1
```

GIVER

```
INT(RND(1)*(5+1))+1
```

GIVER

```
INT(RND(1)*6)+1
```

der er den endelige formel.

LEFT$

De følgende funktioner til behandling af strenge, åbner nogle uanede muligheder, for programmering. F.eks. kan man fjerne dele af en streng og indsætte nye ord eller sætninger. Det betaler sig at kunne kommandoerne udenad. Hver funktion er beskrevet for sig.

EMNE:	Find venstre del af streng
KOMMANDO:	LEFT$(streng,antal)
PARAMETER:	streng — strengvariabel eller streng i anførselstegn. antal — antal venstrestillede tegn, der ønskes fundet.
EKSEMPEL:	PRINT LEFT$("ABCDEFG",3) udskriver de tre venstrestillede bogstaver på skærmen (ABC).
BEMÆRKNING:	Er antallet af tegn, der ønskes større end antallet i strengen, er det hele strengen, der vises.

Denne interessante funktion kan så at sige skære et stykke fra venstre i en streng. Det følgende program siger mere end mange ord:

```
10 INPUT"INDTAST DERES NAVN ";NAVN$
20 ANTAL=1
30 DEL$=LEFT$(NAVN$,ANTAL)
40 PRINT DEL$
50 IF DEL$=NAVN$ THEN 80
60 ANTAL=ANTAL+1
70 GOTO 30
80 PRINT"SLUT"
```

Lad os gå videre med den næste kommando:

RIGHT$

EMNE:	Find højre del af streng
KOMMANDO:	RIGHT$(streng,antal)
PARAMETER:	streng — strengvariabel eller streng i anførselstegn. antal — antal venstrestillede tegn, der ønskes fundet.
EKSEMPEL:	PRINT RIGHT$("ABCDEFG",3) udskriver de tre venstrestillede bogstaver på skærmen (EFG).
BEMÆRKNING:	Er antallet af tegn, der ønskes større end antallet i strengen, er det hele strengen, der vises.

Funktionen svarer til LEFT$-kommandoen, og hvis man har lyst, kan man prøve at skifte LEFT$ ud med RIGHT$ i det forrige program.

Nu har vi fundet både den venstre og den højre del af en streng. Nu mangler vi kun en kommando, der kan finde udsnit inde midt i en streng:

MID$

EMNE:	Find streng-udsnit
KOMMANDO:	MID$(streng,position,antal)
PARAMETER:	streng — strengvariabel eller streng i anførselstegn. position — det sted i strengen, hvorfra udsnittet skal begynde. antal — antal venstrestillede tegn, der ønskes fundet.
EKSEMPEL:	PRINT MID$("ABCDEFG",2,4) udskriver 4 tegn fra strengens 2. tegn (BCDE)
BEMÆRKNING:	Er antallet af tegn der ønskes, større end antallet i strengen, er det hele strengen, der vises.

Efter at vi har "klippet" strenge over i venstre og højre side, har vi en funktion, der tillader, at man henter tegnfølger af ønsket længde, et hvilket som helst sted i en streng. Der er kommet en parameter mere til; det er den, der angiver positionen i strengen, hvorfra en tegnfølge skal hentes.

Funktionen bruges ofte til at dele en streng op i enkelte tegn. Et ord kan således skrives ud lodret på skærmen, hvis man ønsker det. Lad os se engang:

```
10 A$="LODRET"
20 FOR P=1 TO 6:REM STRENGENS LÆNGDE
30 PRINT MID$(A$,P,1)
40 NEXT P
```

Hvad nu, hvis vi ikke ved hvor lang strengen er? Ja, der kan den følgende funktion hjælpe os:

EMNE:	Find strenglængde
KOMMANDO:	LEN(streng)
PARAMETER:	streng — strengvariabel eller streng i anførselstegn.
EKSEMPEL:	A$="ABCDEFG":PRINT LEN(A$) udskriver strengen A$'s længde (7) på skærmen.
BEMÆRKNING:	Er strengen tom, udskrives værdien 0

Indsætter vi funktionen i programmet fra før, kan vi nu få udskrevet en vilkårlig streng lodret:

```
10 INPUT"STRENG : ";A$
20 FOR P=1 TO LEN(A$)
30 PRINT MID$(A$,P,1)
40 NEXT P
```

Denne løsning er mere elegant end den forrige, idet vi undgår at kende antallet af tegn i en streng på forhånd.

Efter at de vigtigste strengfunktioner er gennemgået, følger der et lille indtryksfuldt program:

```
 10 REM **************
 20 REM * LYSAVISEN *
 30 REM **************
 40 CLS
 50 INPUT"TEKST : ";TEKST$
 60 INPUT"LÆNGDE : ";LÆNGDE
 70 INPUT"TEMPO (1-7) : ";TEMPO
 80 L=LEN(TEXT$)
 90 TEKST$=STRING$(1,""-"")+TEKST$+STRING$(1,"-")
100 FOR I=1 TO L
110 LOCATE 12,20
120 PRINT MID$(TEKST$,I,LÆNGDE)
130 FOR W=1 TO TEMPO*50:NEXT W
140 NEXT I
150 GOTO 100
```

Teksten, der skal vises i lysavis indlæses i variablen TEKST$. Derefter indlæses længden af udsnittet, der skal vises. Efter at en tidsfaktor er blevet indlæst i variablen TEMPO indeholder variablen I tekstens længde.

Programmet fortsætter indtil det afbrydes med ESC.

INSTR

Ofte er det nødvendigt at gennemsøge en streng for et bestemt indhold. Her følger en beskrivelse af en sådan funktion:

EMNE:	Søg en streng i en anden streng
KOMMANDO:	INSTR(position,streng1,streng2)
PARAMETER:	pos — det sted hvorfra der skal søges streng1 — streng, der søges i streng2 — streng, der søges efter
EKSEMPEL:	PRINT INSTR(3,ABCDEFGHIJ","FGH")
BEMÆRKNING:	Findes den søgte streng ikke, udskrives værdien 0 Parameter position kan udelades, hvis hele strengen skal gennemsøges.

Denne komfortable funktion understøtter søgning af bestemte data i en fil. Indtil nu har vi kun kunnet undersøge en streng på bestemte positioner, men med INSTR kan man undersøge en hel streng for en kendt tegnfølge. Her er et program til forklaring:

```
  5 CLS
 10 INPUT"TEKST : ";TEKST$
 20 INPUT"SØGEORD : ";SOEG$
 30 P=INSTR(TEKST$,SOEG$)
 40 IF P=0 THEN 70
 50 PRINT "DET SØGTE ORD BEFINDER SIG FRA ";P;".
    POSITION I TEKSTEN"
 60 GOTO 100
 70 PRINT"DET SØGTE ORD ER IKKE FUNDET"
100 PRINT"SØGNING AFSLUTTET"
```

Til slut, en udvidelse af programmer, der gør det muligt at søge og erstatte dele af en streng:

```
  5 CLS
 10 INPUT"TEKST : ";TEKST$
 20 INPUT"SØGEORD : ";SOEG$
 30 INPUT"ERSTATTES AF : ";ERSTAT$
 40 P=INSTR(TEKST$,SOEG$)
 50 IF P=0 THEN 90
 60 TEKST$=LEFT$(TEKST$,P-1)+ERSTAT$+MID$
    (TEKST$,P+LEN(SOEG$),LEN(TEKST$)-LEN(SOEG$)-
    (P-1))
 70 PRINT"NY TEKST : ";TEKST$
 80 GOTO 100
 90 PRINT"SØGEORD IKKE FUNDET!"
100 PRINT"SØGNING AFSLUTTET"
```

En tilsvarende rutine til søgning og erstatning findes i næsten alle tekstbehandlingsprogrammer.

TILLÆG

Reserverede ord

Valget af variabelnavne er ikke så frit, som man skulle tro. Der er en række reserverede navne, der er indeholdt i BASIC-sproget. Disse reserverede navne/ord må ikke anvendes. Bruger man et reserveret ord, vil man få udskrevet fejlmeddelelsen SYNTAX ERROR. Her er en liste over de reserverede ord:

ABS
AFTER
AND
ASC
ATN
AUTO

GOSUB

BIN$

CALL
CAT
CHAIN
CHR$
CINT
CLEAR
CLG
CLOSEIN
CLOSEOUT
CLS
CONT
COS
CREAL

JOY

KEY

LEFT$
LEN
LET
LINE
LIST
LOAD

DATA
DEF
DEFINT
DEFREAL
DEFSTR
DEG
DELETE
BORDER
DIM
DRAW
DRAWR

EDIT
EI
ELSE
END
ENT
ENV
EOF
ERASE
ERL
ERR

OUT

PAPER
PEEK
PEN
PI
PLOT
PLOTR
POKE
POS

ERROR
EVERY
EXP

FIX
FRE
FOR
DI
GOTO

HEX$
HIMEM

IF
INK
INKEY
INKEY$
INP
INPUT
INSTR
INT

STR$
STRING$
SWAP
SYMBOL

TAB
TAG
TAGOFF
TAN
TESTR

LOCATE	PRINT	TEST
LOG	RAD	THEN
LOG10	RANDOMIZE	TIME
LOWER$	READ	TO
RELEASE	TROFF	TRON
MAX	REM	
MEMORY	REMAIN	UNT
MERGE	RESTORE	UPPER$
MID$	RESUME	USING
MIN	RENUM	
MOD	RETURN	
MODE	RIGHT$	VAL
MOVE	RND	VPOS
MOVER	ROUND	
NEXT	WAIT	WEND
NEW	SAVE	WHILE
NOT	SGN	WIDTH
SIN	WINDOW	
ON	SOUND	WRITE
ON BREAK	SPACE$	
ON ERROR GOTO	SPC	XOR
ON SQ	SPEED	XPOS
OPENIN	SQ	
OPENOUT	SQR	YPOS
OR	STEP	
ORIGIN	STOP	ZONE

Fejlmeddelelser

Hvor mærkeligt det end lyder, så er fejlmeddelelser lige så vigtige på en computer som kommandoer. Jo større udvalg i fejlmeddelelser, des hurtigere ved brugeren, hvad der er galt. Hvis CPC, ligegyldigt hvilken fejl der opstod, kun skrev ERROR, så kunne det tage forfærdelig lang tid at finde den fejl, som maskinen påstår, der er opstået.

Nummeret, der er anført for hver meddelelse, tjener til indfangning af en fejl med ON ERROR GOTO i et program. Kommandoen er dog ikke gennemgået i denne bog. Nummeret er medtaget, så man på et senere tidspunkt kan drage nytte af dem.

1 UNEXPECTED NEXT

Der er et NEXT for meget. Der mangler en FOR, eller en anført variabel ved FOR passer ikke med den, der er angivet ved NEXT.

2 SYNTAX ERROR

Dette er den hyppigste fejlmeddelelse. Computeren forstår ikke indholdet af en linie, stavefejl eller ukendt kommando.

3 UNEXPECTED RETURN

Programforløbet er blevet afbrudt med et RETURN, hvor ingen GOSUB-kommando er udført.

4 DATA EXHAUSTED

Der skal indlæses flere DATA end anført i DATA-linier.

5 IMPROPER ARGUMENT

Standardmeddelelse når et resultat af en funktion eller en parameter er ugyldig.

6 OVERFLOW

Resultatet af en beregning ligger uden for computerens kapacitet.

7 MEMORY FULL

Programmet eller variablerne har antaget en størrelse, der overskrider antallet af frie bytes i lagerkapaciteten. Resultatet af en beregning ligger uden for computerens kapacitet. Memory-kommandoen udskriver denne besked, når det forsøges at placere starten på BASIC et ulovligt sted i hukommelsen.

8 LINE DOES NOT EXIST

Et angivet linienummer findes ikke i programmet.

9 SUBSCRIPT OUT OF RANGE

En ARRAY's index er for lille, eller større end dimensioneret med DIM.

10 ARRAY ALREADY DIMENSIONED

Forsøg på at dimensionere en ARRAY, der allerede er dimensioneret.

11 DIVISION BY ZERO

Det er ikke lovligt at dividere med 0

12 INVALID DIRECT COMMAND

Forsøg på at lade en kommando udføre direkte (udenfor et program), der ikke tillader dette (f.eks. INPUT).

13 TYPE MISMATCH

Numerisk værdi, hvor en streng er forventet, eller omvendt.

14 STRING SPACE FULL

Der er ikke mere plads til erklæring af variabler.

15 STRING TOO LONG

Grænsen på 255 tegn i en streng er overskredet (f.eks. ved strengaddition).

16 STRING EXPRESSION TOO LONG

Til udredning af lange strengudtryk mellemlagres strenge. Hvis dette lagerområdes kapacitet overskrides, udskrives fejlmeddelelsen.

17 CANNOT CONTINUE

Et program kan ikke fortsættes med CONT.

18 UNKNOWN USER FUNCTION

En funktion (FN) er ikke blevet defineret med DEF FN.

19 RESUME MISSING

Programmet er blevet afbrudt ved en ON ERROR GOTO-rutine.

20 UNEXPECTED RESUME

En RESUME kan ikke udføres, hvis ikke der er lavet en ON ERROR GOTO-kommando.

21 DIRECT COMMAND FOUND

Ved LOAD fra kassette er der fundet en linie uden linienummer.

22 OPERAND MISSING

Der mangler en parameter.

23 LINE TOO LONG

En BASIC-linies maksimale længde er på 255 tegn.

24 EOF MET

Forsøg på at læse en fil ud over dens afslutning.

25 FILE TYPE ERROR

Læsemetode ulovlig. En datafil kan ikke læses med LOAD.

26 NEXT MISSING

Der er lavet en FOR-løkke uden afslutning med NEXT.

27 FILE ALREADY OPEN

En fil der endnu ikke er blevet lukket er forsøgt åbnet.

28 UNKNOWN COMMAND

Der er brugt en ikke-eksisterende kommando.

29 WEND MISSING

En WHILE-løkke er ikke blevet afsluttet med WEND.

30 UNEXPECTED WEND

Programmet er stødt på en WEND-kommando, hvor ingen WHILE-løkke er defineret.

DATAMAT
AMSTRAD

DATAMAT er et komfortabelt dataforvaltningsprogram til AMSTRAD.
DATAMAT kan indeholde op til 512 tegn pr. »kort« og bruger ethvert felt som index (søge) felt. De kan søge, sortere og udvælge data efter alle kriterier og på alle felter, efter Deres ønske.
DATAMAT kan overføre data til TEXTOMAT således at disse bruges til f.eks. personlige serie-breve m.v.
DATAMAT er extrem hurtig da den er skrevet i 100% maskinkode.

DATAMAT i stikord

- ***Fuld menustyrring giver en hurtig og nemmere betjening.***
- ***Fri definerbar indgangsmaske.***
- ***512 tegn pr. datablok, max. 50 felter pr. blok.***
- ***Arbejder sammen med TEXTOMAT.***
- ***Arbejder sammen med 1 eller 2 diskettestationer.***
- ***Udprintning af lister og labels i fri format.***
- ***Data kan sorteres og selekteres.***
- ***Printer kan også tilsluttes gennem RS 232 porten.***

Kun kr. 498,-

BUDGET MANAGER AMSTRAD

Med BUDGET MANAGER behøver De ikke længere tænke på forvaltningen af Deres husholdningskasse. BUDGET MANAGER kontrollerer alle udgifter og indtægter, kredit- og opsparingskonti på øre. Alle data kan læses på skærm eller udskrives på printer.

Da BUDGET MANAGER arbejder terminsorienteret kan alle data udskrives til forfaldsdato. Naturligvis råder BUDGET MANAGER over mange nyttige kalenderfunktioner, som hjælper dem i overvågningen af de vigtige terminer.

Endvidere er der funktioner til beregning af lånetilbud og renteberegning.

BUDGET MANAGER i stikord

- ***Overvågning af indtægt, udgift, kredit, opsparing og variable omkostninger.***
- ***Kalenderfunktioner.***
- ***Terminsovervågning.***
- ***Bedømmelse af lånetilbud.***
- ***Automatisk aktuallisering af konti.***

Kun kr. 498,-

AMSTRAD-bladet...

-dit brugerblad

Med alt det spændende, nye til en af de allerstærkeste computere på markedet.

Amstrad-bladet er det eneste officielle brugerblad i Danmark.
Dette betyder at DU som bruger bliver holdt orienteret om alle de mange spændende projekter der sker omkring Amstrad computeren, både i Danmark og i udlandet, via direkte telexforbindelse til producenten i England samt vort samarbejde med den danske importør, Dinamicro.
Foruden nyheder og reportager bringer Amstrad-bladet også masser af tests (så du undgår at købe »katten i sækken«), oplysende artikler om programmering (så din investering måske kunne give en lille ekstraindtægt), 16 sider med programmer, som er lige til at taste ind, annoncer, tips og tricks, læser til læser – kontakt, oplysninger om brugerklubber og meget, meget mere.
Bladet er trykt i 4-farve offset og udkommer hver anden måned. Hvert nummer indeholder minimum 40 sider.
Amstrads computere har over hele verden vist sig at være virkelige 'top – chart – runner's når det gælder de internationale salgslister. Men selv en så stærk computer som Amstrad'en har brug for opbakning. Dinamicro og Twilights forhandlere er indstillede på at yde dig den bedst mulige hjælp vedr. dit køb og igangsætningen, resten af vejen kan Amstrad-bladet være en god og værdifuld kilde til oplysninger og interessante opgaver.

EN YDERLIGERE FORDEL FOR DIG:

Få vort nye blad «Input», fyldt med programlistninger til din Amstrad - Helt Gratis...

Vi vil gerne have lov at præsentere vort nye programblad for alle nye ejere af Amstrad computere. Derfor får du gratis det første nummer tilsendt, når du tegner abonnement, og så håber vi naturligvis, at du bliver SÅ begejstret at du henter **INPUT** hos din bladhandler, hver gang det udkommer

FORLAGET MICRO'ICON
Gødvad Bakke 4 8600 Silkeborg 06 82 24 55