Ian Sinclair

Programmering med Amstrad

IAN SINCLAIR

PROGRAMMERING MED AMSTRAD

Oversat og bearbejdet af Peter Wiwe

teknisk forlag as

PROGRAMMERING MED AMSTRAD
Originaltitel: Amstrad Computing with the cpc 464

Oversættelse og bearbejdning: Peter Wiwe

Forlagsredaktion: Henrik E. Hansen
Omslag: John Ovesen
Sats: Satsform, Århus
Tryk: Teknisk Forlag A/S

ISBN 87-571-0909-5

Indhold

Forord

Amstrad CPC464 er en computer med langt flere muligheder for anvendelse både privat og i erhverv, end andre computere i samme prisklasse, – ja, flere end mange dyrere maskiner. Derfor er der sikkert mange af dens købere, der aldrig har brugt en computer før – eller i hvert fald ikke en så alsidig maskine som denne. Den manual, der følger med computeren, er en af de bedste hjemmecomputermanualer, der er lavet, men alligevel er det umuligt at tilgodese alle behov. Og ét er at lære at betjene en computer, – noget andet er at lære at programmere den, eller at lære at udforme sine egne programmer. Denne bog er netop lagt an på at lære begynderen disse ting.

Der vil uundgåeligt være nogle anvendelser af Amstrad CPC464, som ikke kan dækkes ind med færdigkøbte programmer. Når du er kommet til det punkt, hvor du har brug for at lave dine egne programmer, så er det noget, der skal læres. Og programmering er trods alt en af de ting en computer er beregnet til. At have en computer uden selv at programmere den, svarer til at købe en Mercedes, og så lade andre styre den. Hvis du ikke har prøvet at programmere før, kan denne bog lære dig det. Hvis du har prøvet at arbejde med en af de lidt ældre hjemmecomputere, så vil denne bog vise dig en helt ny måde at programmere på. Amstrad har nemlig en langt mere avanceret BASIC, end de tidligere generationer af hjemmecomputere – så det svarer næsten til at starte helt forfra, hvis du kun kender til »stenalder-BASIC«.

Jeg vil gerne understrege at denne bog er skrevet, mens jeg brugte en Amstrad CPC464, som jeg fik hjem hos mig selv, og alle programudskrifterne i denne bog stammer direkte fra min printer, som var koblet til min Amstrad. Det kan måske lyde som en unødvendig bemærkning, men i mange bøger er programmerne ofte sat sammen med den øvrige tekst, og det giver unødvendigt mange fejl. Hvert eneste program i denne bog og hver eneste programordre, er afprøvet på den maskine jeg har foran mig. Jeg har ikke taget noget fra manualen uden først at afprøve det, og i de tilfælde hvor en ordre virker på en måde, som ikke umiddelbart kan forstås ud fra manualen, har jeg påpeget forskellen. Jeg har hele tiden arbejdet med en Amstrad far-

vemonitor, og derfor er det billedet på en farveskærm, jeg beskriver i bogen. Jeg valgte farveskærmen, fordi den er meget populær p.g.a. den lave pris.

Efter mit skøn er Amstrad CPC464 en af de bedste hjemmecomputere, jeg har prøvet, og jeg vil gerne ønske Amstrad tillykke med deres succesfulde debut på computermarkedet. Det var noget nyt for mig at arbejde med en helt ny computer, som fungerede så overbevisende.

Som altid skylder jeg tak til alle de mennesker, der har gjort denne bog mulig.

Ian Sinclair

Oversætterens forord

Denne bog er en bearbejdet oversættelse af den originale engelske udgave. Alle programmer i bogen er afprøvet på en Amstrad CPC464 og er udskrevet direkte på en skrivehjulsprinter. I appendix E kan du finde et lille program, der giver din Amstrad danske tegn. Manuskriptet til den danske udgave er i øvrigt skrevet på Amstrad'en med tekstbehandlingsprogrammet Amsword II/Tasword.

Peter Wiwe

Kapitel 1

Opsætning og afprøvning

Amstrad CPC464 leveres i to papkartoner. Den ene indeholder selve computeren, et demonstrationsbånd og en stor manual på engelsk *). Den anden indeholder enten en grøn monitor (dataskærm), eller en farvemonitor, – strømforsyningen til computeren er indbygget i monitoren. **). Tilkoblingen skulle være uden problemer: De to stik sættes i bøsningerne bag på computeren mærket »MONITOR« og »5V DC«. Når du sætter skærmens 220V stik i stikkontakten, (pas på med ledningsføringen, så der ikke kommer nogen, som falder over ledningen og river det hele på gulvet), skal du inden sikre dig, at både skærm og computer af afbrudt (»OFF«). Derefter tænder du først på stikkontakten, derefter på monitoren og endelig på computeren. Og så skulle den røde lampe på computeren gerne være tændt. I manualen kan du se en illustration af tilkoblingen på Page F1.2. Er du i tvivl om noget, så se i manualen eller spørg din forhandler – begynd ikke under nogen omstændigheder at skille computer eller monitor ad. Du kan få livsfarlig højspænding igennem dig, du kan let komme til at ødelægge noget, og garantien kan bortfalde!

Når du sætter monitoren til, skal du være forsigtig med det seks-polede stik, det passer i bøsningen (monitor), og det er ikke nødvendigt at bruge kræfter, for at få det på plads. Spiralledningerne mellem monitor og computer er desværre temmelig korte, så det kan godt knibe med at de kan nå, med mindre de to enheder står ret tæt på hinanden i samme plan. Jeg har min monitor stående på en hylde over computeren, og jeg måtte strække kabler-

---

*) Du kan købe en dansk manual. Henvisningerne her i bogen er til den engelske udgave, der følger med computeren.

**) Det er også muligt at koble computeren til et almindeligt TV, det sker ved hjælp af en kombineret modulator og strømforsyning, som skal købes separat. Billedkvaliteten er dog noget dårligere, end den er på en monitor. Via modulatoren sker tilkoblingen til TV'ets antenneindgang, og det indstilles på UHF-kanal 36. Manualen fortæller om tilkobling både med og uden monitor, idet Amstrad kan købes både med og uden monitor i England. Da Amstrad sælges med monitor i Danmark, vil vi stort set holde os til opsætningen med monitor. O.A.

ne ud, for at sikre at de kunne nå og at stikkene blev siddende. Hvis der ikke er noget billede på skærmen, når maskinen er tændt, kan du prøve at vrikke forsigtigt med stikkene bag på computeren, for at se om der er ordentlig forbindelse. Der kan ikke ske nogen skade, hvis stikket skulle falde ud, mens du bruger maskinen. Men hvis det lille stik, som leverer strømmen til computeren falder ud, går programmer mv. tabt – hvis du da ikke har dem på bånd. Derfor er det meget vigtigt, at du straks lærer at bruge den indbyggede kassettebåndoptager – eller *datarecorder*, og det kommer vi også ind på i dette kapitel.

Når du har det nødvendige udstyr, skal du have et sted at stille det, og der skal egentlig bruges en del plads. Og senere skal du måske have plads til en printer og en diskettestation, så du ikke blot har en computer, men et helt computersystem. Jeg købte et specielt computerbord (fig. 1.1), som giver en god arbejdsstilling og god plads til forskelligt ekstraudstyr. Så fint behøver det ikke være i begyndelsen, et stort skrivebord og en passende stol kan gøre det. Men det er med datagrej som med Hi-Fi – der er altid et eller andet man kan købe!

Når det hele er sat op og er klar, skal du lære nogle få grundregler for brugen af computeren. Amstrad bruger, som næsten alle andre hjemmecomputere, en kassettebåndoptager til at lagre informationer på. De informationer, der kommer ind i computerens hukommelse, går nemlig tabt når computeren bliver slukket. Kassettebåndoptageren kan optage signaler på bånd, og disse kan gemmes, når maskinen bliver slukket. Når signalerne afspilles igen, sendes de tilbage til hukommelsen, hvor computeren kan arbejde med dem. I modsætning til de fleste andre computere har Amstrad kassettebåndoptageren indbygget. Det giver mange fordele. For det første er du helt fri for besværet med at få båndoptager og computer til at passe sammen, – det er noget der ellers tit giver problemer. For det andet er du helt fri for at koble ledninger til. Og for det tredje er den indbyggede båndoptager en datarecorder, der er lavet specielt til at arbejde med computersignaler, og det giver en mere pålidelig opbevaring af programmer og data, i forhold til at bruge en almindelig kassettebåndoptager. Men selvom datarecorderen giver en god sikkerhed, så er det alligevel en god ide, at have to kopier af de programmer du gerne vil gemme – den anden kopi kaldes en *sikkerhedskopi*.

Ellers skulle du gerne have et billede på din monitor, som svarer til det i manualen på Page Fl.4. Hvis billedet ruller, kan du justere det på »v-hold« (lodret hold). Derudover kan du justere lys og kontrast, så det passer til lysforholdene i det rum du sidder i.

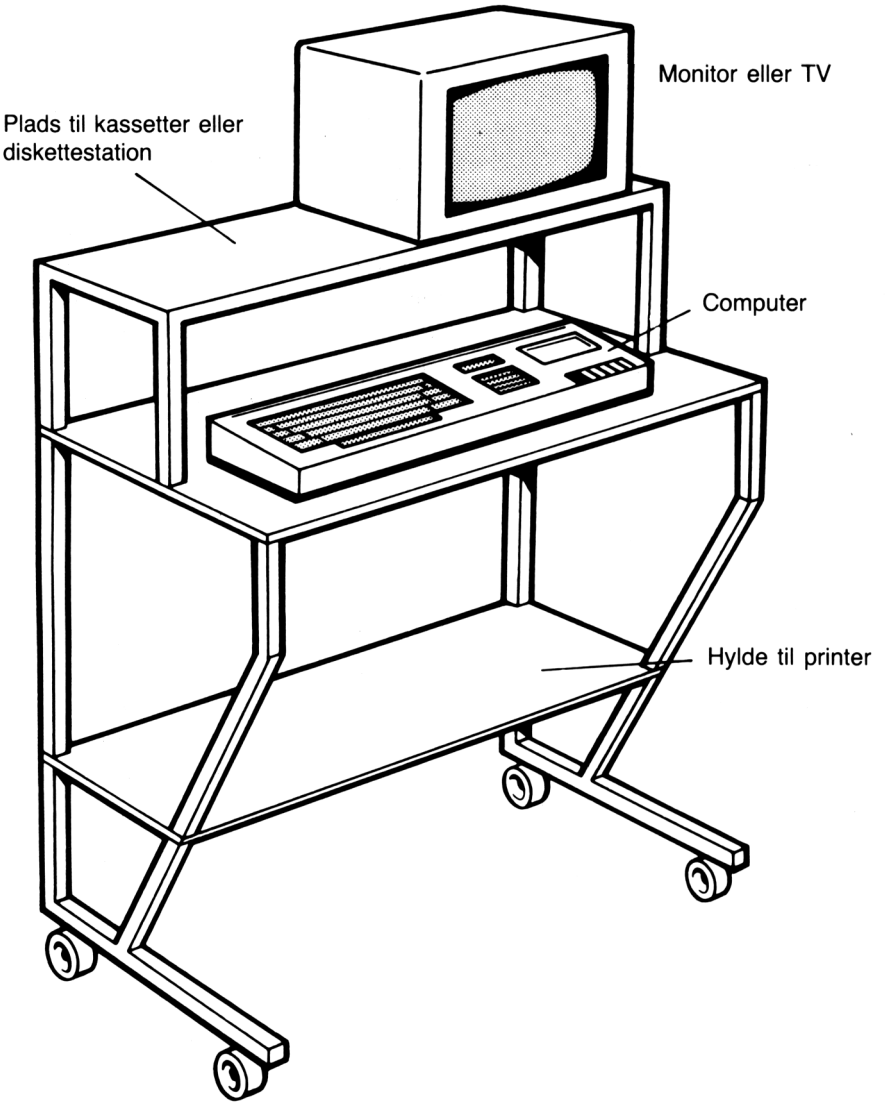

Fig.1.1 Sådan kan et computerbord se ud med plads til et helt computersystem.

Så går vi i gang

Når du har fået billede på skærmen, kan vi tage fat på at lære maskinen at kende. Under meldingen om copyright ser du en klar (guld) firkant lige under ordet »Ready« (klar). Denne firkant kaldes en *markør*. Når du trykker en

tast ned vil det bogstav, tal eller hvad der nu er på tasten, komme frem dér hvor markøren er. Markøren vil så flytte videre til den næste plads på linien. Det er vigtigt at bemærke, at du ikke blot ved at betjene tastaturet kan komme til at ødelægge noget på computeren, det værste der kan ske er, at du sletter hukommelsen og dermed eventuelt mister et program. Derimod kan du komme til at skade den ved at spilde kaffe, cola eller lignende over den, ved at tabe den eller koble den til noget, mens den er tændt. Du kan også komme til at ødelægge nogle signaler på dine bånd, hvis du prøver at tage dem ud af maskinen, mens de kører, eller hvis du enten tænder eller slukker maskinen, mens båndene sidder i. Sluk altid computeren, og det der er koblet til, hvis du kobler stik til eller fra på maskinens bagside. Tænd og sluk *aldrig* maskinen mens der sidder bånd i den. Opbevar dine bånd tørt og på god afstand af varmekilder og skærm/strømforsyning.

Vi sætter os til tastaturet

Nu er det på tide at se på tastaturet, for det er via tastaturet, du styrer computeren. Hvis vi ser bort fra tasterne ude til højre, ligner Amstrad-tastaturet stort set tastaturet på en skrivemaskine. Tal og bogstaver sidder på samme måde som på en skrivemaskine, og hvis du før har skrevet på en skrivemaskine (især en elektrisk), skulle det være meget let at bruge tastaturet. Når du trykker på en af bogstav-tasterne, kommer bogstavet frem på skærmen. Det er et *lille* bogstav du ser, ikke et *stort*. Ligesom på en skrivemaskine, får du de små bogstaver, med mindre du trykker en af de to SHIFT-taster ned samtidig.

Du kan give maskinen ordrer med både små og store bogstaver (eller blandet). Hvis du vil have store bogstaver på skærmen hele tiden, skal du trykke CAPS LOCK-tasten ned. For at vende tilbage til de små bogstaver, skal du taste CAPS LOCK endnu en gang. Desværre er maskinen ikke forsynet med en indikator, som fortæller at CAPS LOCK er i brug, – så du må prøve dig frem! Men uanset om CAPS LOCK er i funktion eller ej, så skal du bruge SHIFT for at få de øverste tegn på tasterne med to tegn. Når du bruger disse taster alene, får du altid det nederste tegn. Med SHIFT får du de øverste tegn frem. Fx får du 8 når du trykker på 8-tasten, men du får (når du samtidig trykker SHIFT ned *).

Udover »skrivemaskinetasterne« er der en række taster, som ikke findes på

---

*) Du kan faktisk godt »låse« SHIFT, – det sker ved at trykke CTRL-tasten ned samtidig med SHIFT. Gentager du operationen kommer du tilbage til den normale funktion. O.A.

en skrivemaskine. Øverst til venstre er der fx en tast mærket ESC (escape = »flygte«, »undslippe«), og til højre for mellemrumstangenten finder du tasten CTRL (control). Disse taster bruges som manualen angiver. ESC-tasten er specielt nyttig, fordi den kan afbryde programkørsler osv. Hvis du derefter trykker på en anden tast (ligegyldigt hvilken), sættes det afbrudte program i gang igen, men trykker du derimod to gange på ESC stopper programmet helt. Når et program stoppes på denne måde kan du altid starte det igen, for det slettes ikke af computerens hukommelse. En anden måde at stoppe en programkørsel på er, at trykke CTRL, SHIFT *og* ESC ned samtidig, men det betyder at hukommelsen slettes samtidig. Vi skal bruge dette senere, fordi det nogle gange er den nemmeste måde at få computeren stillet tilbage til normal funktionsmåde. Før du bruger CTRL-SHIFT-ESC må du sikre dig, at du har optaget det program, du arbejdede med. CTRL kan bruges til nogle specielle funktioner, som vi skal se på senere. En af de taster, du bør lære at kende med det samme er DEL. Skriv er ord og tryk på DEL, som er tasten oppe til højre. Du vil så se at det sidste bogstav, du har skrevet forsvinder (DELete = »slette«). Hvis du holder DEL nede vil du se at de andre bogstaver også slettes, samtidig med at markøren hurtigt bevæger sig til venstre. Når der ikke er flere bogstaver at slette, hører du den indbyggede højttaler give et »bip«. Skru op for lyden (ved siden af afbryderen på kabinettets højre side), hvis du ikke kan høre tonen. Med DEL kan du rette de fejl, du laver, når du skriver.

De fire taster med pilene er markørtaster. Prøv at trykke en af dem ned og du vil se markøren flytte sig i pilens retning. Disse taster bruges, sammen med COPY-tasten i midten, til én slags redigering. Redigeringsmuligheder er beskrevet i appendix A, som du kan lære når du får brug for at redigere. Endelig er der de 12 taster under markørtasterne, som til sammen kaldes for »talblokken«. For dem, der er højre-håndede, tror jeg de er lettere at bruge ved indtastninger af tal. Jeg er venstre-håndet, og jeg bruger dem aldrig!

Den vigtigste af alle taster – lige i øjeblikket – er den store blå ENTER-tast. Den sidder samme sted, som vognretur/linieskift sidder på en elektrisk skrivemaskine, men dens funktion er ikke helt den samme. Når ENTER trykkes ned, er det et signal til computeren om, at du har indtastet en ordre, og at du nu vil have computeren til at udføre den (enter = »gå ind«, »indlæs«). Hvis du er vant til at bruge en elektrisk skrivemaskine, kommer du til at vænne dig til at denne tast bruges lidt anderledes. På en elektrisk skrivemaskine, trykker du på vognretur, hver gang du vil starte på en ny linie. ENTER-tasten virker på en anden måde. Hvis du skriver mere end der kan stå på en skærmlinie, skifter maskinen automatisk linie. ENTER må *ikke* bruges til dette formål. ENTER bruges kun når du vil have maskinen til at

udføre en ordre eller indlæse en ordre i hukommelsen, den bruges *ikke* til at lave linieskift med. Men *når* du taster ENTER, så bliver der samtidig skiftet linie, markøren flyttes ud til venstre, og det næste du skriver kommer frem der.

Hvis du holder en tast nede, vil du se at tegnet på skærmen gentages – tasten »repeterer«, og det sker ret hurtigt. Hvis du skriver en række meningsløse bogstaver og taster ENTER, vil computeren normalt svare med meldingen:

Syntax error

(syntaks fejl) efterfulgt nedenunder »Ready« og markøren. Syntaks fejlmeldingen kommer, fordi en computer på en måde er en ret primitiv maskine, der kun kan forstå nogle få ord. Disse få ord er de *reserverede ord*. Hvis du skriver noget, hvor disse ord ikke bruges, eller hvis du bruger dem forkert, så betragter computeren det som en syntaks fejl. Det kan godt være at du forstår det, men det gør computeren altså ikke! Ordet »syntaks« betyder den måde ord bruges sammen på i alle former for sprog, og for computeren udgør de reserverede ord et sprog.

Afprøvning af datarecorderen

Amstrad lagrer, ligesom alle andre computere, ordrer og data i sin hukommelse – men kun så længe maskinen er tændt. Når du slukker din computer, bliver hukommelsen omgående slettet – også selvom du tænder den igen med det samme. Derfor er det nødvendigt at kunne opbevare programmer og data, som bruges i programmerne, på en anden måde. De bedste lagringsformer er magnetbånd og disketter. Til Amstrad CPC464 kan du købe en diskettestation *). Lagring på disketter er mere sikker og langt hurtigere.

I computeren findes der elektroniske kredsløb, som kan omsætte et program til signaler, der kan indspilles på bånd. Når disse signaler afspilles igen, findes der er andet kredsløb, som omsætter signalerne til et program igen. På den måde kan du med Amstrad's datarecorder optage programmer og afspille dem igen. Før du går videre med denne bog, er det vigtigt, at du prøver at bruge datarecorderen til at optage og afspille programmer.

Det nemmeste er først at prøve at overføre et program fra bånd til computerens hukommelse. Dette kaldes på dansk at *indlæse* et program (engelsk:

---

*) Mens denne bog har været under udarbejdelse er der kommet en Amstrad med indbygget diskettestation. Den hedder CPC664 og er fuldt kompatibel med Amstrad CPC464. O.A.

»loading«), og da der følger et bånd med Amstrad'en, som indeholder forskellige programmer, kan vi ligeså godt prøve at indlæse det først. Som du kan se af den vejledning, der følger med båndet, kan du ikke uden videre lave kopier af programmerne på det. Det skyldes at programmerne er »beskyttet« mod kopiering. Normalt er det ikke nogen god idé at købe »beskyttede« programmer, for det betyder at du ikke kan lave sikkerhedskopier. Og det er yderst vigtigt at have sikkerhedskopier af de programmer, du bruger. Det skyldes ikke, at datarecorderen er specielt upålidelig, – det er simpelt hen blot en almindelig sikkerhedsforanstaltning. Alle kopieringssystemer kan svigte nu og da, og alle der arbejder med computere, har kopier af alt, som skal bruges igen. Det er i virkeligheden ret dumt at betale en masse penge for et program, der ikke kan kopieres. Hvis program-forhandleren stolt fortæller dig at programmet ikke kan kopieres, så *lad være med at købe det!* Men hvis du ikke kan undgå at købe beskyttede programmer, så kan du – hvis du har lidt erfaringer med kassettebåndoptagere – lave kopier ved overspilning mellem to kassettebåndoptagere.

Læg nu »Welcome«-båndet i datarecorderen og indlæs det. Først skal du trykke CTRL og den *lille* ENTER-tast på talblokken ned samtidig. Derefter får du denne udskrift på skærmen:

```
RUN"
Press PLAY then any key:
```

– altså tryk PLAY (på datarecorderen) ned og tryk derefter på en tilfældig tast – fx mellemrumstangenten. Se i øvrigt manualens anvisninger. Nu begynder båndet at køre, og når det efter nogle minutter stopper igen, trykker på på STOP-knappen på datarecorderen. Og det er vigtigt. For selvom computeren stopper motoren, så er »gummitrykrullen« i båndoptageren stadig trykket ind mod »capstanakslen«. Hvis gummitrykrullen er trykket mod capstanakslen i længere tid, så giver det mærker i gummiet. Det betyder at båndet vil blive ført mere uregelmæssigt frem når du optager og afspiller, og derved bliver din kopieringssikkerhed dårligere. Hvis datarecorderen går i stykker, så skal du indlevere hele computeren til reparation – så pas godt på den. Når du har stoppet båndoptageren er programmet gået i gang. Det er mest de programmer du køber, der skal indlæses på denne måde. Det vigtigste nu er imidlertid, at du lærer at optage dine egne programmer.

Klar til optagelse

Før du kan optage noget for at afprøve datarecorderen, skal du have et program at optage, og derfor skal du til at skrive nu. Det er let nok, hvis du lige

har tændt maskinen, men hvis du har lavet nogle tilfældige indtastninger, er det bedst at slette hukommelsen først. Tryk derfor på CTRL-SHIFT-ESC, eller sluk maskinen og tænd den igen (uden der er bånd i den).

Skriv nu tallet 10 (1 og 0), og derefter ordet rem. Det er ligegyldigt om du skriver rem eller REM. Det er et af de reserverede ord, og det er ligegyldigt om du bruger små eller store bogstaver, computeren vil (senere) lave det om til store bogstaver. Hvis det ser rigtigt ud så tast ENTER. Nu bliver linien »10 REM« sendt ind i computerens hukommelse. Når du skriver det første ciffer vil det komme frem på skærmen der hvor markøren står. Når du taster ENTER, vil markøren springe ned i næste linie. Din ordre bliver samtidig stående, hvor den blev skrevet på skærmen. Hvis du brugte små bogstaver står der:

```
10 rem
```

Hvis du laver fejl mens du skriver linien, så skal du blot slette med DEL og skrive det en gang til. Hvis du har skrevet noget forkert, som fx REN eller RWM og taster ENTER, vil maskinen ikke give en fejlmelding – den accepterer det, hvis det står efter et tal. For at rette en linie som fx:

```
20 Ren
```

kan du, i stedet for at bruge DEL, blot skrive den rigtige linie:

```
20 rem
```

og taste ENTER. Indtast nu resten af linierne som vist i fig. 1.2. Og husk at taste ENTER efter hver linie. Tallene kaldes *linienumre*, og de har to funktioner. For det første fortæller de computeren, at det er et program, og for det andet fortæller de computeren i hvilken rækkefølge programmet skal udføres. Du kan undersøge om programmet er rigtigt, ved at få det skrevet ud på skærmen. Ved sådan en programudskrift eller »listning«, udskriver computeren det, der findes i hukommelsen. Når linienumrene bruges sikrer vi, at ordrerne kommer ind i hukommelsen, og hvis du skriver »list« og taster ENTER, får du programmet at se. Du skal ikke blive overrasket over, at de små bogstaver (som rem) er blevet lavet om til store (som REM), for det er en del af computerens behandling af programmer, – den sætter også linierne i rækkefølge, og den laver et mellemrum mellem det sidste ciffer i linienummeret og det første bogstav i ordren. Se efter om »listningen« svarer til den trykte version i fig. 1.2.

Nu skal du have et kassettebånd parat. Tag et nyt bånd helst et C 12 eller C 15, som er specielt beregnet til databrug. Du kan købe dem, hvor du købte din computer, eller hos de fleste radioforhandlere. Indsæt båndet i datarecorderen, og tryk på REW-knappen, for at sikre dig at båndet er spolet helt

```
10 REM
20 REM
30 REM
40 REM
```

Fig. 1.2 Et testprogram til afprøvning af optagelse og afspilning på datarecorderen

tilbage. Når du har gjort det, skal du trykke på den lille knap ved båndtælleren, så den viser 000. Tryk nu på FF-knappen, så båndet spoles frem til 002 på tælleren. På den måde sikrer du at du kommer forbi »indløbsbåndet«, som er uden magnetisk belægning. Det er selvfølgelig yderst vigtigt, at der er noget at indspille signalerne på! Tilmed bliver båndene først slidt lige i begyndelsen, så det er – også af den grund – en god idé at begynde lidt inde på båndet.

Nu skal vi lave optagelsen. Først skriver du:

```
SAVE "TEST"
```

og så taster du ENTER. Ordet SAVE (»gemme«) er en ordre til computeren om at du vil *udlæse* et program på bånd *). TEST er et *filnavn*, programmet er en *fil* (engelsk: file = kartotek, arkiv) og TEST er filens navn. Computeren bruger filnavnet, hvis den skal lede efter programmet på båndet. Et program kan godt udlæses på bånd uden et filnavn, hvis du blot bruger »save« og det første sæt gåseøjne (SAVE"), men det er en god vane at give sine filer navne. Normalt har du flere programmer på et bånd, og ved at bruge filnavnet kan du sikre, at computeren indlæser det rigtige program. Når du har tastet ENTER, får du meldingen:

```
Press PLAY and REC then any key:
```

altså PLAY og REC ned (på datarecorderen) og tryk derefter på en tilfældig tast. – Helt tilfældigt kan tasten nu ikke vælges, idet SHIFT, CAPS LOCK og CTRL ikke kan bruges her, og ESC vil afbryde funktionen helt. Men mellemrumstangenten, ENTER, bogstav- og taltaster kan bruges. Nu vil motoren starte, båndet vil begynde at køre, og lidt efter kommer der en melding på skærmen. I dette tilfælde står der:

```
Saving TEST block 1
```

---

*) Her oversættes »save« ved at »udlæse«, ligesom »load« oversættes ved »indlæse«. Huskereglen er at ind- og udlæsning skal ses i forhold til hukommelsen. Således sker *ud*læsningen fra computerens hukommelse *ud* på båndet, mens *ind*læsningen sker fra båndet *ind* i hukommelsen. O.A.

og hvis du skruer op for højttaleren, kan du svagt høre de signaler, som programmet bliver omsat til. Når programmet er optaget, stopper motoren, og meldingen »Ready« kommer frem på skærmen sammen med markøren. Nu er programmet udlæst på båndet, og programmets størrelse angives af tallet der står efter »block«. De korte programmer, som den bog fx indeholder, fylder normalt ikke mere end 1 »block«. Selvom motoren stopper automatisk skal du huske at trykke STOP-knappen ned, for at skåne datarecorderens mekanik. Og mere skal der ikke til for at lave en optagelse.

Nu kommer det spændende. Du skal sikre dig, at programmet faktisk er lagret på båndet. Skriv NEW og tast ENTER. Så skulle hukommelsen være slettet. For at få en ren skærm at se på, så skriv CLS og tast ENTER. Skriv nu LIST og tast ENTER. Nu skal der ikke komme nogen programudskrift – LIST giver en programudskrift (eller listning), men der er jo ikke noget program. En anden måde at slette hukommelsen på, er at bruge CTRL-SHIFT-ESC.

Nu kan vi indlæse programmet fra båndet. Spol båndet tilbage, skriv LOAD "test" og tast ENTER. Nu får du samme melding, som da du indlæste demonstrationsbåndet, og du trykker PLAY ned (*ikke* REC) og trykker derefter på fx ENTER. Nu starter motoren igen og meldingen:

Loading TEST block 1

kommer frem på skærmen kort efter. Hvis du skruer lyden op, kan du høre de samme signaler som før. Når programmet er indlæst stopper datarecorderen, og som sædvanlig trykker du på STOP. Nu skriver du LIST og taster ENTER og programmet kommer frem på skærmen. Når du har lært at lagre dine programmer på bånd, og har lært at indlæse dem igen, kan du rigtigt gå i gang med at bruge din computer. Hvis du har brugt en time eller mere på at taste et program ind, er det rart at vide, at du på et par minutter kan lagre programmet på bånd, så du er fri for at skulle indtaste det igen. Amstrad's datarecorder er ikke den hurtigste på markedet, men den er meget pålidelig, og det er trods alt det vigtigste. Du skal passe virkelig godt på dine databånd. I fig. 1.3 er der en række regler, som du bør følge, hvis du vil have dine bånd til at holde længe.

Når du senere kommer til at arbejde med meget længere programmer, vil du sikkert synes at kassettebåndene er for langsomme at bruge. Løsningen på det problem er at gå over til at bruge et diskettesystem, – men du *kan* faktisk få datarecorderen til at arbejde hurtigere! Hvis du skriver:

SPEED WRITE 1 (tast ENTER)

bliver datarecorderen sat til at arbejde dobbelt så hurtigt. Båndet kører ikke

1. Opbevar kassetterne tørt og ikke for varmt.
2. Hold altid kassetterne i god afstand fra magneter. Dvs. højttalere, TV, radioer og elektromotorer.
3. Rør aldrig båndet med fingrene.
4. Hvis båndet sætter sig fast, tag det ud og slå det mod din hånd og spol det frem og tilbage nogle gange. Smid dårlige bånd ud!
5. Brug aldrig den første del af båndet.
6. Lad ikke kassetterne stå for længe uden at bruge dem, spol dem evt. igennem ind imellem.

Fig. 1.3 sådan passer du på dine bånd.

hurtigere, men der overføres dobbelt så mange signaler pr. sekund. Du behøver ikke bruge denne ordre når du indlæser – computeren finder automatisk frem til den rigtige overførselshastighed. Lagringen er ikke helt så sikker som den langsomme hastighed, men hvis du passer godt på dine bånd, skulle der ikke være nogen nævneværdig forskel. Det er en god idé at lave sikkerhedskopier på den lave hastighed, og så have arbejdskopier, som kører på den høje hastighed. For at omstille maskinen til den normale (lave) hastighed skriver du:

SPEED WRITE 0 (ENTER)

eller tryk på CTRL-SHIFT-ESC, så hele computerens hukommelse slettes.

LOAD og RUN

Når du senere begynder at skrive dine egne programmer, vil du sikkert gerne indlæse dem på samme måde som Welcome-båndet. Metoden er enkel. Du skriver SAVE efterfulgt af programnavnet på normal vis. Derefter skriver du ,P – P betyder »protected« (beskyttet). Det kan fx se således ud:

SAVE "PProg",P

udlæsningen sker som normalt, men der kommer ikke helt det samme på båndet. For at indlæse båndet skal du enten:

1. Trykke på CTRL og den *lille* ENTER-tast, og derefter trykke på PLAY og derefter en »tilfældig« tast, eller

2. Skriv RUN" og tast ENTER, og gør ellers som beskrevet under 1.

Programmet vil på begge måder blive automatisk indlæst og kørt. Men du kan ikke få programmet LISTet, og du kan ikke køre det mere end den ene

gang. Så du skal aldrig optage et beskyttet program, med mindre du har en kopi af det i forvejen, for hvis programmet er beskyttet er det helt tabt, hvis der går noget galt, og du kan ikke lave kopier af det. Beskyttelsen af programmerne begrænser nok omfanget af piratkopier, men samtidig straffes den almindelige bruger, fordi der ikke kan laves sikkerhedsprogrammer af sådanne programmer. Du kan dog finde måder at omgå beskyttelsen i forskellige datablade, når du bliver dygtigere til at programmere.

Mere om ind- og udlæsning

Hvis det er første gang du læser denne bog, kan du godt springe de næste linier over, for du vil først få brug for det, når du har lidt mere programmeringserfaring. Bortset fra LOAD og RUN er der to måder mere, du kan indlæse programmer på. Du kan bruge CHAIN (»kæde sammen«) efterfulgt af filnavnet. På den måde indlæses programmet og køres med det samme. En anden type indlæsning sker med MERGE (»blande sammen«). Normalt når du indlæser et program slettes hukommelsen automatisk, når et nyt program indlæses. Med MERGE slettes hukommelsen ikke først, så du kan samle mange små programmer til et større. Men programmerne må ikke have sammenfaldende linienumre. Du kan endelig bruge CHAIN og MERGE sammen (som fx CHAIN MERGE"STYKKER") og tilføje et program til et andet og køre det med det samme. En anden funktion er at lade maskinen lave et »katalog« over hvad der er på båndet med CAT. CAT er især anvendelig, hvis du har korte bånd, der er udlæst med SPEED WRITE 1.

Kapitel 2

Vi går i gang

Som du har set i kapitel 1, giver du ordrer til Amstrad via tastaturet, ligesom på næsten alle andre computere. Du har også set at ordrene bliver udført, når ENTER tasten trykkes ned. Indtil nu har du brugt LIST-ordren, som skriver programmet ud på skærmen. Før vi går videre, må du lære CLS-ordren at kende, den visker skærmen ren (»Clear Screen«), – når du husker at taste ENTER bagefter. Når du har lært computeren lidt bedre at kende, vil det ofte være lettere at bruge redigeringsordrene, – de er forklaret i appendix A.

Computeren kan bruges på to måder. Enten direkte eller via et program. Når du bruger *direct mode*, indtaster du en ordre, trykker på ENTER, og ordren bliver udført straks. Det kan nogle gange være ret nyttigt, men langt den vigtigste måde at bruge en computer på, er at lade den køre programmer (*program mode*). Her får computeren et sæt ordrer, som den skal udføre i en bestemt rækkefølge. Sådan et sæt ordrer hedder et *program*. Forskellen er vigtig, fordi programmet kan køres igen og igen uden en masse indtastningsarbejde. Bruger du derimod ordren direkte, skal du indtaste den hver gang, du har brugt den. Ordrene og reglerne for brugen af dem udgør det, som hedder *programmeringssproget*. Amstrad er forsynet med en ny, avanceret version af det mest almindelige programmeringssprog til hjemmecomputere: BASIC. BASIC er en forkortelse for »Beginners All-purpose Symbolic Instruction Code« og det blev oprindeligt lavet til undervisningsformål. Siden hen er det blevet udviklet til en bredere anvendelse. Amstrads BASIC-version er yderligere forsynet med en række ekstra ordrer.

Når man giver computeren ordrer, er det vigtigt at de har en ganske bestemt form, og det gælder både for *program mode* og *direct mode*. Ordrerne skal fx være stavet rigtigt, ellers kan computeren ikke udføre dem. De skal også *bruges* på den rigtige måde. Du kan fx ikke skrive SAVE og så taste ENTER, og så forvente at computeren udfører ordren. Maskinen forventer nemlig gåseøjne efter ordet SAVE, og den kan ikke udføre en ordre, der ikke er i orden. I nogle ordrer er der mellemrum mellem to ord, og hvis det mangler, kan ordren ikke udføres. I andre tilfælde kan man imidlertid godt udela-

de mellemrum. Det må du lære efterhånden, for der findes ingen sikker regel for, hvornår et mellemrum er nødvendigt.

På Amstrad'en kan du, som du ved, skrive ordrerne med både store og små bogstaver, men når programmet bliver LISTet på skærmen, bliver det altid skrevet med store bogstaver. For at undgå forvirring på dette punkt er *alle* ordrer i teksten og i programudskrifterne i denne bog skrevet med store bogstaver. Du kan altså godt bruge små bogstaver, når du skriver ordrer. I bogen bruger jeg store bogstaver til ordrene, for at gøre det nemmere at skelne dem fra den øvrige tekst. Som før nævnt skal alle ordrer efterfølges af et tryk på ENTER-tasten, så det er ikke nødvendigt at skrive det hver gang her i bogen. Derfor vil jeg ikke minde om det hver gang ved fx at skrive LIST (tryk på ENTER).

Lad os nu se nærmere på forskellen mellem *direct mode* og *program mode.* Hvis du blot vil have computeren til at lægge to tal sammen 1.6 og 3.2 fx så skal du skrive:

PRINT 1.6 + 3.2 (og husk så ENTER)

Du skal starte med at skrive PRINT (eller print), for maskinen forstår kun ret få, bestemte ordrer. Hvis ordren PRINT ikke bliver brugt her, kan maskinen ikke vide hvad det er, du vil have den til at gøre. Den kan ikke genkende ordrer som fx HVAD ER eller WHAT IS, da de ikke findes i computerens BASIC-sprog. Men PRINT er et af de ord, maskinen forstår. De ord, som computeren forstår kaldes *reserverede ord.* Husk at der *skal* være et mellemrum efter T i PRINT. Det er derimod ikke nødvendigt at have mellemrum mellem 6-tallet og + eller mellem + og 3-tallet.

Når du trykker på ENTER efter at have skrevet PRINT 1.6+3.2, så viser skærmen svaret 4.8. Svaret kommer imidlertid ikke samme sted, hvor ordren blev skrevet. Det kommer i den næste linie, et mellemrum inde. Hvis du har brugt CLS, før du skrev ordren, kommer svaret oppe i venstre hjørne. Hvis du bruger en farve-skærm er baggrundsfarven blå efter CLS. Der er en særlig finesse ved PRINT-ordren, idet du kan bruge et spørgsmålstegn (?) i stedet for PRINT. Og her behøver du ikke engang at lave mellemrum frem til det første tal Du kan skrive:

?1.6+3.2

og taste ENTER og få svaret, uden at være bange for at få »syntax error«, som du måske troede. Da det er hurtigere at taste ? end at skrive PRINT, er det en god måde at spare sig selv for noget skrivearbejde på, – især når der er en masse linier, der starter med PRINT.

Når en ordre i *direct mode* først er udført, så er den slettet. Og her er det,

at et program virker helt anderledes. Når et program er skrevet bliver det ikke udført, når du taster ENTER. Det bliver i stedet lagret i computerens hukommelse, og så kan det kaldes frem, når du har brug for det. Computeren skal kunne kende forskel på direkte ordrer og programordrer, derfor er BASIC-sproget bygget sådan op, at hver programinstruktion starter med et *linienummer.* Dette skal være et helt, positivt tal – et såkaldt naturligt tal. Det er derfor at maskinen ikke kan forstå instruktionen 5.6+3=, for den regner 5 for at være linienummeret, og så giver resten jo ingen mening.

```
10 PRINT 5.6+6.8
20 PRINT 9.2-4.7
30 PRINT 3.3*3.9
40 PRINT 7.6/1.4
```

Fig. 2.1 Et regneprogram på fire linier.

Lad os gå i gang med at programmere. Først med nogle simple regneoperationer: addere, subtrahere, multiplicere og dividere. Når jeg tager dem først, er det fordi de er enkle at arbejde med, for computere bruges faktisk ikke til den slags udregninger i særligt stort omfang. Men derfor er det jo meget godt at kunne alligevel. Fig. 2.1 viser et program på fire linier, som vil give resultaterne på de fire små regnestykker.

Der kan læres meget ud af dette lille eksempel. For det første har linierne numrene 10, 20, 30, 40 frem for 1, 2, 3, 4. Det gør det nemlig nemmere at lave ændringer i programmet senere hen. Hvis du fx vil have en instruktion ind mellem linie 10 og 20, skal du blot skrive en linie med et nummer mellem 10 og 20, det kan være 11, 12, 15 eller et andet helt tal mellem 10 og 20, som så indeholder den nye instruktion. Det betyder ikke noget at linierne ikke kommer i den rigtige rækkefølge – det finder maskinen selv ud af. Hvis du havde brugt numrene 1, 2, 3, 4, havde der ikke været plads til at indføje nye linier. Så skulle programmet først have været omredigeret ved hjælp af de forskellige redigeringsordrer (som er beskrevet i appendix A).

For det andet kan du se at nullerne ligner store Ø'er. Grunden til at de skrives sådan er, at de ikke skal kunne forveksles med bogstavet O. Computeren kan nemlig ikke forstå hvad bogstavet O betyder, hvis den forventer et tal som fx nul. Og fejlen kan jo let opstå, fordi man på en skrivemaskine ofte bruger bogstavet O som et nul. Når nul derfor skrives Ø, undgås denne forveksling *). Prøv at skrive nogle O'er og nogle Ø'er på skærmen, så du kan

*) På engelsk findes bogstavet Ø jo ikke, – så pas på du ikke kommer til at bruge bogstavet Ø i stedet for tallet 0. Pas i øvrigt også på at du ikke bruger bogstavet l i stedet for tallet 1. O.A.

vænne dig til forskellen. I bogens tekst er nullerne skrevet på normal vis, men læg mærke til, at der er en lille forskel: Nullerne er lidt mere ovale end de store O'er. Det gælder også for programudskrifterne.

Men der er flere ting at bemærke. Stjernen (eller asterixen) i linie 30 bruger Amstrad'en i stedet for et normalt multiplikationstegn. Der er heller ikke noget almindeligt divisionstegn, her bruges den lille skråstreg under spørgsmålstegnet i stedet for. Du kan huske det på at den minder om en lille brøkstreg.

Nu skal du bare skrive programmet, som du ser det. Det er ikke nødvendigt at lave mellemrum mellem linienummeret og P'et i PRINT – det finder computeren selv ud af, når den skriver programmet ud på skærmen. Derimod *skal* du huske at lave et mellemrum *efter* PRINT, og det er vigtigt at understrege, for det er ikke alle computere der er så nøjeregnende. Hvis du bruger ? i stedet for PRINT, behøver du ikke at tænke på mellemrum. Hver gang du har skrevet en linie i programmet, skal du taste ENTER, og derefter skrive den næste linie. Når du har gjort det, skal dit program gerne se ud ligesom fig. 2.1. Når programmet er skrevet på skærmen, og du har tastet ENTER efter linie 40, er programmet sendt til computerens hukommelse (eller lager), hvor det findes som et sæt tal-koder. Nu kan du så enten checke om programmet findes i lageret, eller du kan straks lade maskinen køre programmet.

Det første gør du som bekendt ved at bruge ordren LIST. Hvis du vil, kan du først bruge CLS, for at få slettet det der står på skærmen, og derefter indtaste LIST og endelig trykke på ENTER.

Først i det øjeblik ENTER trykkes ned vil programmet blive skrevet ud på skærmen. Så vil du se at computeren har skrevet programmet med mellemrum mellem linienumrene og PRINT. Hvis du har brugt små bogstaver, er de blevet lavet om til store, og har du brugt ?, er det blevet lavet om til PRINT med mellemrum både før og efter. For at få programmet *kørt* skal du bruge ordren RUN. Skriv RUN og tryk på ENTER, og du kan se at programmets instruktioner er blevet udført. På skærmen står der nemlig nu:

```
12.4
4.5
12.87
5.42857143
```

I den sidste linie kan du se hvor nøjagtigt Amstrad kan lave udregninger. Hvis du har LISTet programmet før du kørte det, kan du for øvrigt se, at resultaterne bliver udskrevet under programmet. Hvis du vil have resultater-

ne til at stå alene på skærmen, skal du bruge CLS (tryk på ENTER), inden du bruger RUN.

Når PRINT efterfølges af en udregning som fx 2.8 * 4.4, så er det kun *resultatet* af udregningen, der bliver skrevet på skærmen. Programmet skriver *ikke* 2.8 * 4.4, det skriver kun resultatet af udregningen 2.8 * 4.4.

Det er praktisk nok kun at få resultatet skrevet ud. Men nogle gange er det rart også at få spørgsmålet med – man kan jo fx glemme, hvad det var der skulle udregnes. Og på Amstrad'en kan man skrive hvad som helst på skærmen, *hvis* det bliver skrevet som det, der hedder en *streng*.

```
10 PRINT"2+2="2+2
20 PRINT"2.5*3.5="2.5*3.5
30 PRINT"9.4-2.2="9.4-2.2
40 PRINT"27.6/2.2="27.6/2.2
```

Fig. 2.2 Brugen af gåseøjne. I dette eksempel blev ? brugt i stedet for PRINT, men i udskriften er det blevet »oversat« til PRINT.

Fig. 2.2 viser brugen af *strenge*. I hver programlinie er en del af udskriften sat i gåseøjne. Prøv at indtaste programmet, og kør det derefter. Så kan du se at computeren har udført ordrerne helt anderledes. Det der var *indenfor* gåseøjnene bliver skrevet præcis, som du skrev det. Det der ikke var i gåseøjne bliver udregnet, således at linie 10 fx giver os resultatet:

```
2+2= 4
```

Og det kan vel ikke overraske nogen. Og dog: hvis du skriver en ny linie:

```
15 PRINT "2+2= "5*1.5
```

så får du følgende besked, når du har skrevet RUN:

```
2+2= 7.5
```

Computeren gør simpelt hen det, den har fået besked på – og du har lige bedt den om at skrive, at resultatet af 2+2 er det samme som 5*1.5. Så der er vist ikke grund til at frygte, at computeren vil erobre verdensherredømmet!

Her er der grund til at lægge mærke til noget andet. Den nye linie 15 er blevet sat ind imellem linie 10 og 20 – prøv LIST, hvis du ikke tror på det. Det er nemlig ligegyldigt i hvilken rækkefølge, du skriver linierne, computeren vil altid ordne efter stigende numre. Læg også mærke til at der er blevet indsat et mellemrum mellem = og det første ciffer i svaret. Det sker for at give plads til et eventuelt – eller +. Hvis du vil have mere plads efter =, skal du blot lave et mellemrum mellem = og ".

Med al den viden vi har nu, kan vi begynde at se på andre typer af udskrifter. Når PRINT bruges alene, betyder det altid en udskrift på skærmen. Hvis vi skal have en udskrift på papir via en *printer*, (sådan en papirudskrift kaldes en *hard copy*), skal vi bruge en udvidet PRINT-ordre: PRINT#8. # er et »nummertegn« og #8 er altid en printer. PRINT#8 vil altså altid få printerne til at lave en udskrift, *hvis du har tilsluttet en printer!* Hvis der ikke er tilsluttet en printer, skal du undgå denne ordre, idet den får computeren til at gå i *baglås*, dvs. at den ikke reagerer på tastaturet og i øvrigt virker helt død. Skulle det ske, kan du enten tilslutte en printer eller trykke to gange på den røde ESC-tast, for at »lås op« for computeren igen. I appendix B er det beskrevet hvordan en parallel printer som fx Epson RX80, kan tilsluttes Amstrad'en.

```
10 PRINT"Dette er"
20 PRINT"den glimrende"
30 PRINT"Amstrad CPC464 computer"
```

Fig. 2.3 Sådan kan PRINT-ordren bruges til at få tekst på skærmen.

Prøv nu programmet i fig. 2.3. Du kan prøve at skrive linierne i en anden rækkefølge for at se, at de bliver ordnet efter stigende numre, når programmet bliver LISTet. Når du sletter det, der står på skærmen, og derefter kører programmet, kommer teksten frem på skærmen i tre linier. Det skyldes, at PRINT ikke bare betyder, at teksten skal skrives på skærmen. Det betyder samtidig, at der skal startes på en ny linie – helt ude til venstre! Når du fylder skærmen helt med tekst, vil du lægge mærke til at den øverste linie forsvinder, og en tom linie kommer frem nederst. Denne funktion kaldes *rulning*, og det er sådan, der bliver plads til en masse linier på en skærm, hvor der kun er plads til 25 linier i alt.

Nu er det jo ikke altid, at vi vil have en ny linie for hver PRINT-ordre, ved hjælp af *skilletegnene* ; og , kan vi ændre på PRINT-funktionen. Denne gang skal du starte med at slette hukommelsen. Det sker ved at skrive NEW (og derefter ENTER.) Nu er det gamle program slettet fra lageret, vil du have slettet skærmen også, må du bruge CLS. Hvis du ikke bruger NEW, kan det ske, at linier fra det gamle program dukker op i det nye program. Ganske vist sletter du de gamle linier, med de nye du indtaster, men hvis der er et linienummer i det gamle program, som du ikke bruger i det nye, vil den gamle linie blive i hukommelse. I fig. 2.2 ville linie 15 ikke blive slettet, selvom du skrev en ny linie 10 og en ny linie 20.

Prøv nu programmet i fig. 2.4. Når du kører programmet, vil du se, at der er en vigtig forskel mellem fig. 2.3 og fig. 2.4. Semikolonet efter de sidste

```
10 PRINT"Dette er ";
20 PRINT"den glimrende ";
30 PRINT"Amstrad CPC464"
```

Fig. 2.4 Sådan virker semikolon.

gåseøjne i en linie bevirker nemlig, at den nye PRINT-ordre i næste linie bliver udskrevet *uden* linieskift. Når du har kørt programmet vil du derfor se, at hele teksten bliver skrevet i en linie. I dette tilfælde ville det have været meget nemmere, kun at have en linie:

10 PRINT"Dette er den glimrende Amstrad CPC464"

men i visse situationer er det nødvendigt at bruge semikoloner for at få flere PRINT-ordrer til at blive skrevet ud på samme linie. Det vender vi tilbage til i senere programeksempler. Prøv at ændre linie 30 så der står:

30 PRINT"AMSTRAD CPC464 computer"

og kør programmet igen. Nu bliver de to første programlinier skrevet ud på samme linie, mens teksten i linie 30 bliver skrevet ud nedenunder. Det skyldes at et af ordene i linie 30 var blevet delt, fordi der simpelt hen ikke er plads på linien. Det forhindrer Amstrad'en ved i stedet at tage en ny linie i brug, den finesse – der gør det nemmere at få pæne udskrifter – er der ikke mange computere der har!

Rækker og kolonner

For at få pæne udskrifter, må ord og tal ordnes i rækker og kolonner, så det vil vi se lidt nærmere på nu. Vi ved allerede at PRINT vil medføre et linieskift, så en kørsel af programmet i fig. 2.5 skulle ikke overraske nogen. I linie 10 og 20 er der alligevel noget vi ikke har set før, nemlig to ordrer i samme linie.

```
10 CLS:PRINT"Dette er CPC464"
20 PRINT:PRINT
30 PRINT"- parat til at arbejde."
```

Fig. 2.5 Skærmen slettes først med CLS, og der bruges flere ordrer i en linie.

Ordrerne er adskilt med et kolon (:), og man kan i princippet have alle de ordrer man vil efter et linienummer. Programlinien må gerne fylde flere linier på skærmen, problemet er bare at programmet bliver sværere at læse. Når der er flere ordrer i en linie, bliver de udført i rækkefølge – dvs fra venstre mod højre. CLS-ordren i linie 10 virker på samme måde, som når den bliver

brugt direkte, den sletter skærmen, og teksten i programmet starter oppe til venstre.

I linie 20 er der faktisk også noget andet nyt, idet de to PRINT-ordrer giver to tomme linier, inden teksten i linie 30 bliver skrevet ud. Det kan også gøres på andre måder, som vi senere skal se, men det her er en enkel og hurtig måde at få større linieafstand på.

På fig. 2.6 opstiller vi tal i kolonner. I linie 10 bruges PRINT-ordren i forbindelse med tallene 1 og 2. Når de bliver skrevet ud på skærmen, er der mellemrum imellem dem, ligesom de var opstillet i to kolonner. Det er kommaet, der får det til at ske – helt automatisk. Kommaet findes lige ved siden af M'et, og hvis du i stedet bruger apostroffen på tasten med 7-tallet, sker der noget helt andet. Så selvom de ser ret ens ud på tastaturet, bruges de altså vidt forskelligt. Linie 20 demonstrerer, at der kun kan laves 3 kolonner på denne måde, derfor havner 4-tallet i kolonne 1 i næste linie i stedet for. I linie 30 og 40 kan du se, at ord kan stilles i kolonner på samme måde. Når ord skal stilles i kolonner, skal du huske at kommaerne skal sættes *udenfor* gåseøjnene. Hvis et komma kommer indenfor et sæt gåseøjne, vil det bare blive skrevet ud sammen med det, der ellers står indenfor gåseøjnene, og kommaet vil ikke skabe noget mellemrum. Der er plads til 14 anslag (eller *karakterer*) i hver kolonne, og linie 50 viser, hvad der sker, hvis vi prøver at skrive noget, der er for langt til at være der. Den lange sætning i linie 50 går helt ind i kolonne 2, og det der skulle have stået i kolonne 2 kommer hen i kolonne 3 i stedet for. I linie 60 kan du se hvad der sker, når der bruges mange kommaer – kolonnerne fortsætter blot linie efter linie.

```
10 PRINT 1,2
20 PRINT 1,2,3,4
30 PRINT"EN","TO"
40 PRINT"EN","TO","TRE","FIRE"
50 PRINT"DENNE HER ER LÆNGERE","TO"
60 PRINT 1,2,3,4,5,6
```

Fig. 2.6 Kommaet bevirker at tal og ord opstilles i kolonner.

Den funktion kommaerne har her går igen på næsten alle hjemmecomputere, men Amstrad'en giver også andre muligheder. Prøv at skrive ZONE 6, tryk på ENTER og kør programmet igen. Nu passer det hele bedre, det er ligesom der er flere kolonner nu – og det er der faktisk også. Det tal der kommer efter ZONE bestemmer hvor mange *karakterer*, der skal være mellem hver kolonne, så på den måde kan du få det antal kolonner du har brug for. Du kan godt ændre ZONE-værdier flere gange i et program, sådan at du

får de enkelte kolonneudskrifter til at stå, som det nu ser pænest ud. Det kan specielt være nyttigt, hvis du skal bruge computeren til regnskaber eller andre erhvervsformål.

Brugen af kommaet er en enkel måde at få lavet kolonner med samme mellemrum. Men der findes andre og mere avancerede måder at få placeret udskrifterne på. Det sker ved at bruge ordren TAB (ikke tasten!), som skal komme efter PRINT-ordren. TAB er en forkortelse for »tabulator« og bevirker, at der kan laves udskrifter hvor som helst på linien. Når vi bruger TAB skal vi huske at skærmen – når vi tænder for maskinen – er opdelt i 40 kolonner (dvs med plads til 40 karakterer) på tværs af skærmen. Fig. 2.7 viser TAB-numrene på tværs af skærmen.

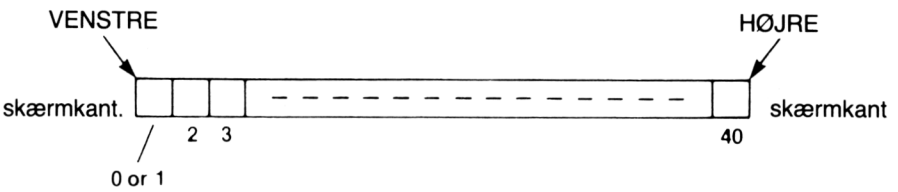

Fig. 2.7 Sammenhængen mellem TAB-numre og placeringen på skærmen.

Placeringerne har numrene fra 1 til 40, men du kan også bruge TAB(0), som er det samme som TAB(1) – helt ude til venstre. TAB(1) er altså helt ude til venstre, og TAB(40) er helt ude til højre. TAB skal stå efter PRINT og skal efterfølges af TAB-nummeret i parentes. Hvis parentesen ikke kommer med, bliver der udskrevet et nul sammen med det ord eller det tal, som du ville have skrevet ud – du vil senere forstå hvorfor.

```
10 PRINT TAB(18)"Midten"
20 PRINT TAB(5)"Start her"
30 PRINT TAB(45)"45 er her"
```

Fig. 2.8 Brugen af TAB til placering af markøren i et progam.

Prøv nu at bruge TAB som vist i fig. 2.8. Det første ord bliver skrevet ud midt på skærmen, det andet nær ved venstre side. Det tredje ord bliver imidlertid skrevet på samme placering som det andet. Det skyldes at numrene fra 1 til 40 bruges sammen med TAB-ordren, og hvis du bruger 41, svarer det til at starte forfra med 1. Linien bliver altså *ikke* forkastet, hvis du bruger et TAB-nummer, der er større end 40, sådan som det sker på nogle maskiner. Der er også en anden ting at lægge mærke til her. Ordet »midten« i dette

eksempel er skrevet ud midt på skærmen ved hjælp af TAB(18). Fig 2.9 viser en fremgangsmåde, der kan bruges til at finde et TAB-nummer, der vil få et ord eller en sætning til at stå midt på skærmen (*centrering*). Husk imidlertid at Amstrad'ens billedfelt ikke altid står helt midt på skærmen – på min monitor er den venstre margin fx lidt bredere end den højre.

1. Tæl antallet af tegn i overskrift – mellemrum medregnet.
2. Træk tallet fra 40 hvis det er lige, fra 41 hvis det er ulige.
3. Divider tallet med 2.
4. Brug resultatet som TAB-nummer.

Fig. 2.9 Sådan »centreres« en overskrift.

Fig. 2.10 viser en anden måde at lave mellemrum mellem skærmudskrifterne på. Her bruges SPC-ordren (*space*=mellemrum), og hvis du tror, at den minder om TAB-ordren, tager du fejl! Fig. 2.10 viser forskellen. Når du bruger TAB, kommer det første bogstav af hvert ord på den bestemte TAB-placering. Læg nu først mærke til, at du kan have mere end en TAB i en linie efter en PRINT-ordre. Til forskel fra de fleste andre computere skal Amstrad'en ikke have semikolon til at adskille de forskellige dele af denne ordre. I linie 20 er SPC brugt. I stedet for faste placeringer på skærmen laver SPC mellemrum – i dette tilfælde er der således 16 mellemrum mellem CPC464 og COMPUTER. Det giver en stor forskel i forhold til at få C i COMPUTER til at stå på TAB(16). Normalt bruges TAB når man vil have pæne kolonner på skærmen, mens SPC bruges når man vil have faste mellemrum mellem ord eller tal, selvom de har forskellig længde. Men der er endnu en måde at placere sine udskrifter på, og den er endnu mere flexibel end TAB. Ordren hedder LOCATE (placér), og ved hjælp af den kan du placere ord hvor som helst på skærmen – i den rækkefølge du ønsker. Når du ellers bruger PRINT, så får PRINT computeren til at starte på en ny linie. Men ved at bruge LOCATE kan du placere ord og tal overalt på skærmen, og du kan endda få en linie til at erstatte en anden, hvis du vil have det.

```
10 PRINT TAB(2)"CPC464"TAB(16)"COMPUTER"
20 PRINT TAB(2)"CPC464"SPC(16)"COMPUTER"
```

Fig. 2.10 SPC bruges til at lave faste mellemrum i udskrifter.

På Amstrad *skal* LOCATE-ordren bruges *før* den PRINT-ordre, som den bruges sammen med. Det kan være i linien lige før PRINT-ordren, eller det kan være i samme linie, adskilt med et kolon.

LOCATE skal efterfølges af to tal, Det første af disse er den *vandrette* placering. Du kan skrive 40 tegn (bogstaver, tal. punktummer) på tværs af skærmen. Numrene går fra 1 (til venstre) til 40 (til højre) – præcis som ved TAB-ordren. Det andet er den *lodrette* placering – *skærmlinienummeret.* Skærmens linier starter med 1 (øverst) og går til 25 (nederst).

```
10 CLS
20 LOCATE 1,1:PRINT"ØVERST TIL VENSTRE"
30 FOR N=1 TO 1000:NEXT
40 LOCATE 24,25:PRINT"NEDERST TIL HØJRE";
50 FOR N=1 TO 1000:NEXT
60 LOCATE 1,25:PRINT"NEDERST TIL VENSTRE"
70 FOR N=1 TO 1000:NEXT
80 LOCATE 25,1:PRINT"ØVERST TIL HØJRE"
90 FOR N=1 TO 1000:NEXT
100 LOCATE 17,12:PRINT"MIDTEN"
```

Fig. 2.11 LOCATE kan bruges til at lave udskrifter overalt på skærmen.

Se nu på eksemplet i fig. 2.11, hvor LOCATE demonstreres. Linie 10 sletter skærmen, og linie 20 starter med LOCATE 1,1. Det betyder at den efterfølgende udskrift kommer helt oppe til venstre. Der *skal* være mellemrum mellem E i LOCATE og det første tal. Det første bogstav i udskriften kommer derfor i øverste venstre hjørne, hvor det i alle tilfælde alligevel var blevet udskrevet, når det kommer lige efter CLS. Den næste linie laver en pause. Den slags ordre har vi endnu ikke set på – men det kommer.

Det næste LOCATE er efterfulgt af 24,25, således at sætningen NEDERST TIL HØJRE får E'et i HØJRE placeret helt nede i højre hjørne af skærmen. Hvordan blev det regnet ud? – Ved at tælle E'et som kolonne 40, og derefter tælle baglæns 38, 38, 37 osv. hen til N, som bliver 24. Det skal stå på den nederste linie – nummer 25. For at forhindre skærmen i at rulle på dette sted i programmet, skal der sættes et semikolon i slutningen af linien. Normalt er det ikke nødvendigt med et semikolon, men E'et i HØJRE udskrives på den sidste placering på skærmen, hvilket jo normalt får skærmbilledet til at rulle en linie op. Nu skulle resten af programmet være til at forstå umiddelbart. Pauserne er indsat, så du kan se at ordrerne udføres i rækkefølge efter linienumrene, og ikke i den normale rækkefølge: fra venstre-til-højre og fra top-til-bund.

Fig. 2.12 giver et godt overblik over kolonne- og linienumre på skærmen, når du skal bruge LOCATE-ordren. Figuren viser flere forskellige sæt tal, som måske ikke lige er til at gennemskue. De forskellige tal-sæt henviser til

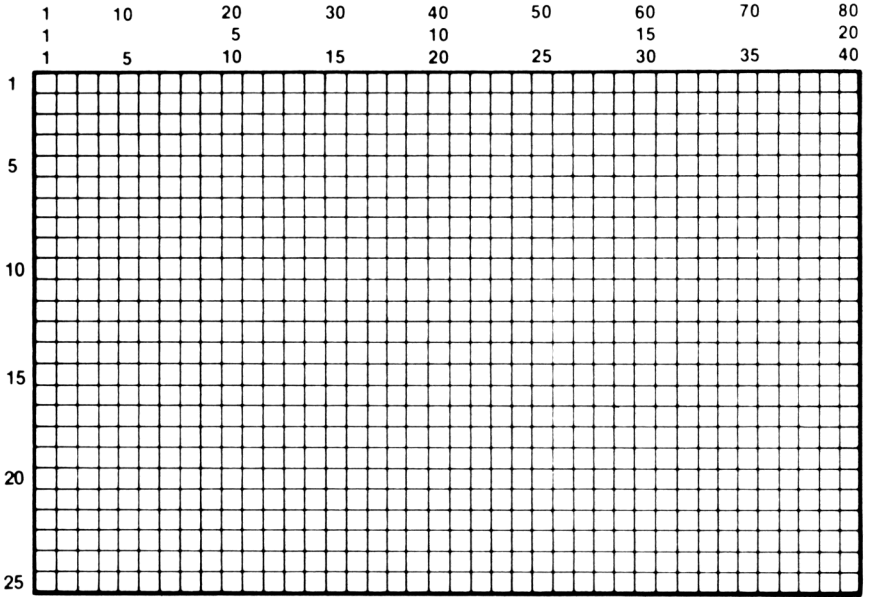

Fig. 2.12 Kolonne- og linienumre på skærmen som bruges i forbindelse med LOCATE-ordren.

tre forskellige størrelser af tegnene på skærmen. Prøv at give ordren MODE 0 – uden først at have slettet programmet i fig. 2.11 (husk ENTER). Der *skal* være mellemrum mellem E'et i MODE og nullet. Hvis du LISTer programmet nu vil du se, at bogstaverne nu pludselig er blevet dobbelt så brede. Når du derefter kører programmet vil du se, at ordene til venstre stadig står rigtigt, mens de andre står forkert. Det skyldes at LOCATE-numrene i MODE 0 går fra 1 til 20 tværs over skærmen, mens der stadig er 25 linier. Prøv nu at skrive MODE 2 (ENTER). Nu vil du se at tegnene er blevet meget mindre – 80 i en linie. De er også sværere at læse, især på en farveskærm eller på et TV. LOCATE-numrene vandret går nu fra 1 til 80. Når Amstrad'en tændes er den altid i MODE 1, med 40 tegn pr. linie. Det er et udmærket kompromis mellem at have tegn der er lette at læse, og samtidig at have et rimeligt antal tegn pr. linie. Hvis du bruger den grønne Amstrad-skærm kan du godt bruge MODE 2, ellers er det bedst at undgå de små MODE 2 bogstaver.

Til sidst et par finesser. Amstrad'en har to ordrer som mange maskiner mangler. Den ene er AUTO. Hvis du skriver AUTO (ENTER), vil maskinen automatisk skrive linienumrene for dig! Du skal bare skrive selve indholdet i linien. Og hver gang du taster ENTER, kommer der et nyt linienummer frem! Hvis du skal afbryde den automatiske linienummerering, skal du blot taste ESC to gange. Du kan starte AUTO-funktionen hvor du vil (prøv fx

AUTO 100), og du behøver ikke at have linienumrene til at stige med ti hver gang – prøv AUTO 10,5. Det er en meget nyttig funktion, hvis du fx er ved at kopiere fra en trykt program-listning.

RENUM er den anden ordre. Hvis du skriver RENUM, vil dit program blive om-nummereret med linie 10 som den første linie, og med linienumrene stigende med 10 ad gangen. Du kan også lade om-nummereringen starte i en bestemt linie, fx linie 240, og du kan bestemme hvor meget nummereringen skal stige med. Det er utroligt, men der er faktisk mange maskiner, der mangler disse smarte detaljer.

Kapitel 3

Nogle variationer

Indtil videre har vi indskrænket os til at bruge computeren til at skrive tal og ord på skærmen. Det er naturligvis også vigtigt nok, – men nu skal vi se på nogle af de operationer, der foregår, inden der bliver lavet udskrifter på skærmen. En af disse hedder en *tildeling*. Se engang på programmet i fig. 3.1.

```
10 CLS
20 x=23
30 PRINT"2 gange"x" er";2*x
40 x=5
50 PRINT"x er nu"x
60 PRINT"og 2 gange "x" er";2*x
```

Fig. 3.1 Eksempel på tildelinger. Bogstavet x bliver brugt i stedet for et tal.

Indtast det, kør det, og sammenlign det, der kommer på skærmen, med det der står i programmet. Den første linie, der giver udskrift på skærmen, er linie 30. På skærmen står der:

2 gange 23 er 46

men tallene 23 og 46 forekommer slet ikke i linie 30! Det der sker er, at bogstavet x bruges som en slags kode for tallet 23. Sådan en kode kaldes for en *variabel.*

Linie 20 tildeler variablen x værdien 23. »Tildeler« betyder at når vi bruger x, *uden* gåseøjne omkring, vil computeren bruge værdien 23 i stedet for. Da x kun består af et tegn og 23 består af to, sparer vi altså plads. Havde x fået tildelt værdien x = 1675.3345, så havde vi sparet endnu mere plads. Linie 30 viser at x faktisk har fået værdien 23, for når x ikke står mellem gåseøjne, bliver der udskrevet 23 i stedet for, og udtrykket 2*x bliver tilsvarende til 46. Men x behøver ikke være 23 hele tiden. I linie 40 får x tildelt værdien 5, og linie 50 og 60 viser os at x nu har fået en ny værdi.

Det er derfor vi kalder x for en »variabel« – vi kan ændre (variere) den, når

vi vil have den til at antage en ny værdi. Men indtil vi ændrer den, forbliver x's værdi den samme. Selv efter programmet i fig. 3.1 er kørt kan du – hvis du ikke har slettet programmet eller tilføjet nye linier – indtaste PRINT x (ENTER) og få værdien oplyst på skærmen. Måske har du lagt mærke til en lille detalje i programmet. – Man kan godt sætte et variabelnavn efter en sætning i gåseøjne, som i linie 30: PRINT"2 gange"x. Men man kan *ikke* gøre det, hvis sætningen efterfølges af en variabel, der indgår i et udtryk, som fx 2*x eller x−7. Det er derfor en god vane, altid at bruge et semikolon når en variabel eller et udtryk kommer efter et sæt gåseøjne.

Men tilbage til variablerne. Du behøver ikke at bruge et enkelt bogstav som variabelnavn, du kan også bruge »rigtige« navne. Et variabelnavn skal starte med enten et lille eller et stort bogstav, og derefter kan du sætte flere bogstaver eller tal. Men du kan *ikke* bruge mellemrum, eller regnetegn (+, −, *, /) eller skilletegn *). Du kan altså bruge navne som fx SUM, regnskab, DETERDETHELE og R2D2 som variabelnavne, og hver af dem kan blive tildelt hver sin værdi. Her er det vigtigt at lægge mærke til at Amstrad'en *ikke* skelner mellem små og store bogstaver. Hvis du fx tildeler variabelnavnet hest værdien 45 og variabelnavnet HEST værdien 88, vil du opdage at både hest og HEST har fået tildelt samme værdi – nemlig 88, som er den værdi, der blev tildelt sidst. Der er også noget andet du skal være opmærksom på – *reserverede ord.* De reserverede ord er Amstrad'ens ordre-ord som fx PRINT, LIST, RUN og NEW. Du kan *ikke* bruge disse som variabelnavne, og skulle du komme til det alligevel, vil du få meldingen »Syntax error«. På nogle computere kan du ikke engang bruge variabelnavne, der indeholder ordre-ord, så der må du ikke bruge ord som fx nummerliste, der indeholder ordet LIST. Amstrad'en er mere flexibel, her må du bruge alle ord, der ikke er magen til ordrene. Hvis du spørger maskinen om værdien af en variabel, der ikke er tildelt nogen værdi, vil du få svaret 0 (nul), – nogle maskiner giver her en fejlmelding.

Men variabler kan bruges til andre ting også. Du kan således på samme måde tildele »navne« til ord og sætninger. Forskellen er blot, at du skal tilføje et dollartegn ($) til variabelnavnet. Hvis n er en variabel for et tal (*talvariabel*), så er n$ (udtales n-streng eller n-dollar) en variabel for et ord eller en sætning (*strengvariabel*). Computeren behandler disse to, n og n$, vidt forskelligt og bruger dem vidt forskelligt. Tildelingen sker også på forskellige måder. Når du tildeler et tal til en talvariabel, bruger du = tegnet, og du skal ikke bruge gåseøjne omkring tallet. Når du bruger = tegnet ved tildeling af

*) Hvis du bruger danske tegn på din Amstrad, skal du være opmærksom på at du *ikke* kan bruge æ, ø, å i variabelnavne. O.A.

en strengvariabel, skal du bruge gåseøjne. Vi skal senere se på andre måder at lave tildelinger på. I det næste eksempel skal vi bruge »lange« variabelnavne, dvs. variabelnavne på mere end to bogstaver. Selvom sådanne navne fylder lidt mere i computerens hukommelse, så er de alligevel gode at bruge, fordi man ved at vælge de rigtige navne, kan blive mindet om hvad variablen står for. Og der er jo masser af plads i Amstrad'ens hukommelse, så sådan en luksus er der råd til.

Strengeleg

Fig. 3.2 viser brugen af *streng-variabler*, dvs. brugen af variabelnavne for ord og sætninger. I linie 10 og 20 sker tildelingen, og i linie 30 til 50 vises det hvordan disse variabelnavne kan anvendes. Bemærk at du kan bruge en variabel, som ikke skal sættes i gåseøjne, sammen med almindelig tekst der *skal* i gåseøjne. Når du sætter almindelig tekst og strengvariabler sammen, skal du passe på, at ordene ikke går direkte over i hinanden. I linie 30 og 50 kan du se, at der er indsat mellemrum for at undgå dette. Når du skriver flere variabler efter hinanden, indsætter du mellemrum ved at sætte gåseøjne – trykke på mellemrumstasten – og sætte gåseøjne igen " " – sådan. For at lave mellemrum mellem tekst i gåseøjne og en variabel, skal du blot lave et mellemrum, der hvor du har brug for det – som fx i linie 40. Det er ikke strengt nødvendigt at bruge semikolonner, når du sætter tekst sammen på denne måde. Hvis du undlader at bruge semikolon i disse tilfælde, vil du få det samme på skærmen, forudsat at der ikke indgår talvariabler i matematiske udtryk.

```
10 CLS:navn$="CPC464"
20 foran$="Den glimrende":bagved$="hjemm
ecomputer"
30 PRINT foran$;" ";navn$;" ";bagved$
40 PRINT"Jeg bruger ";navn$
50 PRINT foran$;" ";navn$;" i funktion!"
```

Fig. 3.2 Brugen af strengvariabler, – de kan kendes på dollartegnet.

Fig. 3.3 viser et andet eksempel, – her bruger vi variabelnavnene paf$ og puf$ til nogle længere sætninger. Der er selvfølgelig ikke nogen idé i at lave udskrifter på denne måde, hvis de kun skal bruges en enkelt gang. Men hvis du skal bruge en hel sætning mange gange i et program, så kan du på denne måde undgå at skulle skrive det igen og igen. I linie 40 skal du i øvrigt lægge mærke til, at de to strengvariabler er adskilt af semikolon. Det er ikke nød-

```
10 CLS:paf$="Den nye computer "
20 puf$="som giver dig meget mere for pe
ngene"
30 PRINT"CPC464- "
40 PRINT paf$;puf$
50 PRINT paf$+puf$
```

Fig. 3.3 Her har hele sætninger fået variabelnavne.

vendigt at bruge det her, men det betyder at programmet bliver lettere at læse.

Hvad enten du bruger semikolon i dette tilfælde, eller blot skriver de to variabler efter hinanden, så vil maskinen prøve at undgå at ord bliver delt. Hvis »puf$« var blevet skrevet lige efter »paf$« på samme linie, så var ordet »mere« blevet delt, som linie 40 viser. Hvis du laver en tekstlinie, der er længere end en linie på skærmen, kan det selvfølgelig ikke undgås at ord bliver delt. I linie 50 kan du se, at de to sætninger er blevet »tvunget« sammen ved at bruge et +. Vi vender senere tilbage til denne måde at bruge + på.

Vi spiller på flere strenge

Da en strengvariabel altid er markeret med et $ tegn, kan der ikke ske forvekslinger med en talvariabel, fx kan a$ ikke forveksles med a. Vi kan udmærket bruge begge udtryk i samme program, uden at det generer maskinen. Fig. 3.4 viser et eksempel på hvor stor forskellen egentlig er.

```
10 a=2:b=3
20 a$="2":b$="3"
30 CLS
40 PRINT a,b
50 PRINT a$,b$
60 PRINT a" gange"b" er";a*b
70 PRINT a$" gange "b$" er umuligt!"
```

Fig. 3.4 Streng- og talvariabler ser måske ret ens ud, men de er meget forskellige!

I linie 10 og 20 sker der en tildeling af talvariablerne a og b, og strengvariablerne a$ og b$. Når variablerne er blevet skrevet ud i linie 40 og 50, er det ikke til at se forskel på a og a$, eller på b og b$. Det eneste der adskiller dem er at talvariablerne udskrives med et mellemrum foran, og derfor står en kolonne længere til højre end strengvariablerne. Men forskellen bliver tyde-

lig, når computeren skal til at regne med dem. Den kan godt gange to talvariabler med hinanden, men den kan ikke gange strengvariabler med hinanden, heller ikke når strengvariablerne er tal. Du kan godt gange "2" med "3", men du kan ikke gange "Vesterbrogade 2" med "Allégade 3". Derfor vil computeren ikke udføre nogen former for udregninger med strengvariabler. Hvis du prøver at lave den udregning, der står i linie 70, vil du få en fejlmelding, og fejlmeldingen vil altid standse videre kørsel af programmet. Fejlmeldingen er »Type mismatch in 70«, og den betyder at du har forsøgt at lave en operation med strenge, som kun kan lave med talvariabler. Vi skal senere se, at der også er operationer, der kun kan laves med strenge og ikke med talvariabler, og forkert brug vil give den samme fejlmelding her. Forskellen er meget vigtig. Computeren lagrer tal på en helt anden måde end den lagrer strenge. De forskellige lagringsmetoder gør det lettere for maskinen at lave udregninger med talvariabler, og lettere at lave andre operationer med strenge. Det er jo trods alt kun en maskine!

Der er en operation, der ligner en udregning, der kan laves med strenge, men ikke med tal. Det er en bestemt brug af +, men det er ikke addition, som vi kender det med tal. Fig. 3.5 viser denne »sammenlægning« af strenge, det kaldes *sammenkædning*.

```
10 a$="Møller"
20 b$="Jensen"
30 CLS
40 PRINT"Kald mig bare "a$+"-"+b$" , sag
de han."
50 PRINT:a$="123":b$="456"
60 PRINT"Samlet til en streng er det ";a
$+b$
70 PRINT"En addition ville give"; 579
```

Fig. 3.5 Sammenkædning af strenge. Det er ikke det samme som addition!

Det har ikke noget med addition at gøre, som du kan se ud af linie 40 og 60. I linie 60 bruges tal i gåseøjne i stedet for ord. Og for at gøre forskellen tydelig, viser linie 70 hvad der var sket, hvis vi havde haft talvariabler her. Sammenkædning er en meget god måde at undgå en masse skrivearbejde på – du har allerede i fig. 3.5 set hvordan strenge kan sættes sammen. Se nu på fig. 3.6. Her defineres a$ og b$ som tegn, der kan bruges som »ramme« omkring en overskrift. Overskriften står i linie 20 »Den nye CPC464«. Linie 40 udskriver en sammenkædet streng.

```
10 a$="***":b$="+++"
20 s$="DEN NYE CPC464"
30 CLS
40 PRINT TAB(8) a$+b$+s$+b$+a$
```

Fig. 3.6 Sammenkædning til at fremhæve en overskrift.

I forbindelse med dette sammenkædningseksempel, er der en meget nyttig ordre, som kan lave lange rækker af ens tegn for dig. Ordren hedder STRING$, og den skal efterfølges af to poster i parentes. Den anden post fortæller hvilket tegn du vil bruge, og den første post fortæller, hvor mange gange du vil have tegnet i kæden. Hvis du fx skriver g$=STRING$(20,"$"), vil du få 20 dollartegn på række. Fig. 3.7 viser STRING$ ordren i funktion.

```
10 CLS
20 a$=STRING$(10,"*")
30 b$=STRING$(5,"+")
40 PRINT:PRINT a$+b$+"TITEL"+b$+a$
```

Fig. 3.7 STRING$ bruges til at lave en række ens tegn. I fig. 8.10 er en anden anvendelse af STRING$ vist.

INPUT

Indtil videre er det, der er blevet skrevet ud på skærmen af de forskellige programmer, noget der har stået i programmet, inden det er blevet kørt. Men det er skam ikke en nødvendighed, for vi kan godt *indlæse* information – navne eller tal – i programmet, mens det kører. Den slags kaldes for et *input* og BASIC-ordren, der bruges i denne sammenhæng hedder – meget passende – INPUT.

```
10 CLS
20 PRINT"Hvad hedder du?"
30 INPUT navn$
40 CLS:PRINT:PRINT
50 PRINT navn$;" - det glæder mig at træ
ffe dig!"
```

Fig. 3.8 Her bruges INPUT-ordren. Dit navn kommer ind i sætningen i linie 50.

Fig. 3.8 viser dette med det program, der skriver dit navn. Jeg ved jo ikke hvad du hedder, så jeg kan ikke skrive det i programmet på forhånd. Det der

sker, når programmet køres, er at spørgsmålet:

Hvad hedder du?

bliver skrevet ud på skærmen. På linien nedenunder vil du se (endnu) et spørgsmålstegn, efterfulgt af markøren. Maskinen venter nu på, at du skal skrive noget og så taste ENTER. Indtil ENTER bliver trykket ned, vil programmet vente på svar i linie 30. Hvis du er ærlig, giver du maskinen dit eget navn og taster ENTER. Du behøver *ikke* sætte gåseøjne omkring dit navn – du skal bare skrive dét, du vil se på skærmen. Når du taster ENTER bliver dit navn tildelt variablen navn$. Det er en af de måder at lave tildelinger på, hvor du ikke skal bruge gåseøjne. Nu kan programmet køre videre, – i linie 40 bliver skærmen slettet og der bliver lavet to gange linieafstand. Du kunne selvfølgelig have svaret Batman eller Bedstemor And, eller hvad du ellers kunne have fundet på. Computeren har jo ingen muligheder for at kontrollere, om det rigtige navn bliver oplyst. Hvis du ikke skriver noget, men blot taster ENTER, så vil programmet fortsætte uden et navn. Så tro endelig ikke at computere kan alt!

```
10 CLS:PRINT"Indtast venligst et tal"
20 INPUT n
30 CLS:PRINT
40 PRINT"to gange"n" er";2*n
```

Fig. 3.9 INPUT i en talvariabel. Dit INPUT skal være et tal.

Det er ikke kun strengvariabler, der bruges sammen med INPUT. Fig. 3.9 viser en indlæsning i talvariablen n. Fremgangsmåden er ellers den samme. Når programmet venter med et spørgsmålstegn og markøren, skal du skrive et tal og taste ENTER. ENTER vil så tildele n dit tal, og programmet vil kunne køre videre. Linie 40 viser, at dit tal bliver brugt i programmet. Når du bruger en talvariabel i et INPUT, så skal det være et tal du indlæser. Hvis du prøver at indlæse en streng, vil du straks få en fejlmelding – »redo from start« – dvs. »start forfra«. Til forskel fra de fleste andre fejlmeldinger får denne *ikke* programkørslen til at stoppe, – den giver dig i stedet en mulighed for at lave dit INPUT om. Hvis dit INPUT gælder en strengvariabel, kan indlæses hvad som helst, men der kommer straks en fejlmelding, hvis du prøver at lave udregninger med en streng.

Du kan bruge INPUT i et program på en måde, der får det til at se ud som om computeren kan tale med dig. I fig. 3.10 er der et eksempel på det.

```
10 CLS
20 INPUT"Skriv venligst dit navn ";navn$
30 PRINT
40 PRINT"Hej med dig, ";navn$
```

Fig. 3.10 INPUT bruges her til at skrive en sætning, som svaret indgår i.

Denne gang kommer der nemlig en sætning før dit navn. Sætningen står mellem gåseøjnene, og derefter kommer et semikolon og variabelnavnet navn$. Linie 20 har samme funktion som disse to linier ville have haft:

```
15 PRINT "Skriv venligst dit navn";
20 PRINT navn$
```

og denne gang kommer spørgsmålstegnet i *samme* linie som spørgsmålet. Dit svar kommer også i samme linie – med mindre dit navn er så langt, at det må fortsætte i næste linie. Du kan også bruge et komma i denne linie i stedet for et semikolon. Prøv ... Kommaet sletter spørgsmålstegnet, men markøren bliver stående.

Brugen af INPUT er ikke begrænset til et enkelt navn eller tal, vi kan bruge INPUT sammen med flere variable, og vi kan have flere slags variabler i en INPUT-linie. Fig. 3.11 viser et eksempel, hvor der bruges to variabler efter et INPUT. Den ene er en strengvariabel navn$, og den anden er en talvariabel nr. Når computeren når frem til linie 20, vil den udskrive forespørgslen, og vente på at *begge* indlæsninger sker – først et navn og så et tal.

```
10 CLS
20 INPUT"Navn og nummer ";navn$,nr
30 PRINT:PRINT
40 PRINT"Navnet er ";navn$
50 PRINT"Nummeret er ";nr
```

Fig. 3.11 To variabler i et INPUT trin.

Det skal gøres rigtigt: Først et navn, så et komma, så et tal og endelig ENTER. Hvis du taster ENTER før du har fået indlæst tallet, vil du få besked på at starte forfra (»redo from start«). Du må så starte med navnet igen og derefter komma, tallet og ENTER. Navn og tal vil så blive udskrevet i linie 40 og 50. Ud af dette kan du se, at du *aldrig* må indlæse noget, der indeholder et komma i et INPUT. Hvis du fx har et INPUT navn$ trin, må du ikke skrive »Hansen, Søren«. Det ville blive behandlet som to indlæsninger, og maskinen ville kun acceptere »Hansen«. Så hvis du prøver at indlæse »Hansen, Søren« vil du få en »redo from start« melding. Når vi bruger computere, skal disse

detaljer være i orden – og der er forskel fra mærke til mærke! Der findes dog en ordre, der gør det muligt at indlæse poster, der indeholder kommaer. Den hedder LINE INPUT, og bruges nøjagtigt som du bruger INPUT.

```
10 CLS
20 INPUT"Indtast fire tal (husk komma me
llem dem) ";a,b,c,d
30 PRINT
40 PRINT"Summen af tallene er ";a+b+c+d
```

Fig. 3.12 Et INPUT trin, der kræver fire tal indlæst.

Vi kan gå endnu videre. I fig. 3.12 skal der indlæses fire tal. De skal også her indlæses på en gang, og her minder programmet selv om, at der skal kommaer mellem hver post. Der må først tastes ENTER, når alle fire tal er indlæst. Tallene bliver tildelt variabelnavne, og programmet lægger dem sammen i linie 40.

READ ... DATA

Der er også en anden måde at indlæse data i et program på, mens det kører. Her sker indlæsningen fra en liste, og der bruges to ordrer, READ og DATA. READ får programmet til at indlæse en post fra listen. Listen starter med ordet DATA. De enkelte poster i listen skal være adskilt med kommaer. Hver gang en post bliver indlæst fra listen, bliver »adressen« ændret, så den næste post i listen vælges næste gang, der skal ske en indlæsning.

Vi skal se nærmere på det her i kap. 5, nu skal vi blot lære READ ... DATA at kende. Fig. 3.13 viser en meget enkel brug af de to ordrer.

```
10 CLS
20 READ j
30 PRINT"Post ";j;
40 PRINT" er en ";
50 READ navn$
60 PRINT navn$
70 DATA 5, diskette station
```

Fig. 3.13 READ ... DATA bruges til at indlæse data i et program.

I linie 20 indlæses den første post på listen, og tildeler den variabelnavnet j. Posten bliver så udskrevet i linie 30, og semikolonnet sørger for, at linie 40 udskrives på samme skærmlinie. Semikolonnet sidst i linie 40 har samme funktion, og i linie 50 indlæses den anden post i listen. Denne post får varia-

belnavnet navn$, og den bliver udskrevet i linie 60. Listen findes i linie 70, og den indeholder to DATA – et tal og en sætning. Tallet kan jo skrives som det er, men sætningen kan også skrives *uden* gåseøjne omkring. Tildelinger til strengvariabler i en DATA-linie kan således ske *uden* gåseøjne. Hvis du alligevel bruger gåseøjne i en DATA-linie, vil programmet blot se bort fra dem, – og der kommer ingen fejlmelding.

Du skal være omhyggelig med at få READ og DATA til at passe sammen. Hvis du bruger en talvariabel i en READ-linie, som fx READ a, så *skal* det der indlæses fra DATA-linien være et tal. Hvis det ikke er det, kommer der en fejlmelding. Den vil vise at der er fejl i DATA-linien, og give dig mulighed for at rette fejlen. Måske skulle du lige holde en pause nu og læse appendix A om redigering! Hvis du bruger en strengvariabel, som fx READ a$, betyder det ikke noget, om der står et tal eller streng i DATA-linien. Husk at hvis du indlæser et tal ved hjælp af READ a$, så er det ikke muligt at bruge dette tal i udregninger. Og husk at der *skal* være et mellemrum efter ordet DATA, og at posterne skal være adskilt af kommaer. Du kan ikke indlæse poster, der indeholder kommaer.

READ ... DATA ordrerne er for alvor effektive, når der skal indlæses poster fra en lang liste ved hjælp af et READ-trin. Det kan fx være poster, der skal bruges hver gang programmet køres – og det er jo nemmere, end hvis du skulle indlæse dem som svar hver gang. Men det er vi ikke helt klar til at lære endnu, så det vender vi tilbage til senere.

Tryllerier med tal

Det du har lært om computere indtil nu, har sikkert overbevist dig om, at computere ikke kun kan arbejde med tal. Men i mange sammenhænge er computerens færdigheder i talbehandling meget centrale. Hvis du fx vil have maskinen til at arbejde med matematik eller andre beregninger, så kan den virkelig udnytte disse egenskaber i langt højere grad end ved fx spil, tekstbehandling eller regnskaber. Så lad os se lidt på talbehandleren Amstrad. Vi kommer ikke til at gå i dybden med dette – for der er slet ikke plads til at forklare alt det, den kan, på dette område. Hvis du har beskæftiget dig med matematik før, og kender udtryk som sinus, tangens og exponentialfunktioner, så vil du let kunne bruge disse matematiske funktioner i dine programmer. Ved du ikke hvad disse udtryk betyder, kan du bare springe de dele, der handler om dem, over.

Den enkleste og mest grundliggende taloperation er at tælle. Man kan enten tælle ved at lægge til eller ved at trække fra. Lægger man 1 til det tal, man har i forvejen, *opskriver* man, trækker man 1 fra, det tal man har, *nedskriver*

man. Disse enkle operationer kan godt se ret forvirrende ud i BASIC, som det ses i fig. 3.14.

```
10 CLS
20 x=5:PRINT"x er"x
30 PRINT
40 x=x+1
50 PRINT"Der er nu sket en ændring - "
60 PRINT"x er nu"x
```

Fig. 3.14 Her opskrives x's værdi med 1, lighedstegnet betyder »bliver til«.

I linie 20 får variablen x værdien 5. Det bliver skrevet ud i linie 20, men linie 40 »opskriver x«. Det sker med den underlige »ligning« x = x+1, som betyder at x får en ny værdi, som er 1 højere end den tidligere værdi. Resten af programmet viser at x nu har fået en ny værdi – nemlig 6.

Brugen af = tegnet betyder »bliver til« eller »ændres til«, og den må du vænne dig til. Når den samme variabel optræder på begge sider af lighedstegnet, betyder det altså, at vi giver variablen en ny værdi. Vi kunne lige så godt have en linie som:

x = x−1

den ville betyde at x nu fik en værdi, der var *»nedskrevet«* (med 1). Vi kunne også have x = 2*x, – her ville x få en ny værdi, der er dobbelt så stor som den tidligere værdi, eller x = x/3, hvor x får en ny værdi, der er en tredjedel af den tidligere værdi. Fig. 3.15 viser en tildeling af denne type, hvor der indgår både addition og division i ændringen af x's værdi.

```
10 CLS
20 x=5:PRINT"x er"x
30 PRINT
40 x=2*x+4
50 PRINT"Der er nu sket en ændring - "
60 PRINT"x er nu"x
```

Fig. 3.15 Her indgår et *udtryk* i tildelingen af en ny værdi.

Fig. 3.16 viser nogle forskellige tal-funktioner. Her forstår vi en tal-funktion, som en funktion, hvor vi indsætter et tal, og får et andet tal ud. Linie 10 giver x værdien 2.5. Linie 20 udskriver kvadratet på x, dvs. x*x. Det bliver programmeret ved at skrive x ↑ 2, og du bruger det tegn der sidder på samme tast som £-tegnet. For at få kvadratroden af variablen x, skal vi bruge ordren SQR. Du kan også bruge x ↑ .5 i stedet, men SQR(x) er vist nemmere at

```
10 CLS:x=2.5
20 PRINT"x i anden er ";x^2
30 PRINT
40 PRINT"Kvardratroden er";SQR(x)
50 PRINT
60 PRINT"Den naturlige logaritme er";LOG
(x)
70 PRINT"og den almindelige logaritme er
";LOG10(x)
```

Fig. 3.16 Nogle tal-funktioner.

skrive – og huske. Skal du bruge andre rødder, som fx kubikrod, kan du bruge udtryk som x ↑ (1/3) osv. LOG(X) giver den *naturlige* logaritme af x. Hvis du vil have den almindelige 10-tals logaritme skal du bruge LOG10(x) *). I fig. 3.17 er der vist de funktioner der kan bruges, med en kort forklaring til hver. Nogle af funktionerne har kun interesse, hvis du skal lave programmer med et matematisk, statistisk eller teknisk indhold. Men en del af funktionerne kan faktisk bruges i andre programtyper, – fx i grafikprogrammer. Fig. 3.17 viser også hvordan maskinen prioriterer de forskellige regneoperationer. Prioriteringen skal sikre at man får et rimeligt resultat, når man fx skriver 3+4*5. I dette tilfælde er svaret 23, fordi multiplikation *altid* udføres før addition.

ABS(x) Ændrer eventuelt negativt fortegn til positivt.
ATN(x) Den inverse tangens-funktion.
BIN$(x,y) Ændrer x til et binært tal, og udfylder med nuller til længden y.
CINT(x) Afrunder x til et heltal.
COS(x) Giver cosinus af vinklen x (i radianer).
CREAL(x) Omformer x til et reelt tal.
DEG Omstiller computeren til at regne i grader.
EXP(x) Exponentialfunktionen, opløfter e til potensen x.
FIX(x) Fjerner brøker eller decimaler fra et tal.
FRE(x) Oplyser hvor meget af maskinens hukommelse, der ikke er i brug eller reserveret.
HEX$ Ændrer et tal til hex (grundtal 16).
INT(x) Runder ned til nærmeste mindre tal.
LOG(x) Giver den naturlige logaritme af x (ln).
LOG10(x) Giver den almindelige 10-tals logaritme af x.

---

*) Bemærk at LOG svarer til den funktion, vi normalt kalder ln, mens den vi plejer at kalde log her hedder LOG(10). O.A.

MAX(x,y,z, ...) Giver de største tal i listen.
MIN(x,y,z, ...) Giver det mindste tal i listen.
PI Giver pi (π) – forholdet mellem omkredsen af en cirkel og dens diameter.
RAD Omstiller computeren til at regne i radianer.
RANDOMIZE(x) Giver en ny række af »tilfældige« tal.
RND(x) Giver et tilfældigt (decimal)tal mellem 0 og 1.
ROUND(x,y) Runder x af til det antal decimaler y angiver.
SGN(x) Giver x's fortegn. + hvis x er positivt, − hvis x er negativt og 0 hvis x er nul.
SIN(x) Giver sinus af vinklen x (i radianer).
SQR(x) Giver kvadratroden af x.
TAN(x) Giver tangens af vinklen x (i radianer).
UNT(x) x skal være mellem 0 og 65536. UNT omsætter x til et tal mellem −32768 og +32767.

Prioritering
For almindelig regning er rækkefølgen MDAS – dvs. Multiplikation, Division, Addition og Subtraktion. Hel prioriteringslisten er:
1. Potensopløftning med ↑.
2. Multiplikation og division.
3. Addition og subtraktion.
4. Sammenligning med = > <.
5. AND
6. OR
7. NOT

Fig. 3.17 Talfunktionerne, med korte forklaringer. (Hvis du ikke kender dem alle sammen, gør det nok ikke så meget, – du får næppe brug for dem!)

Hvor nøjagtig?

Noget af det der tit er et problem ved hjemmecomputere, er nøjagtigheden ved udregninger. Du ved sikkert at brøken ⅓ ikke kan udtrykkes helt nøjagtigt i decimaltal. Jo flere decimaler vi kan tage med, jo nøjagtigere bliver tilnærmelsen, således er 0.33 nærmere på den virkelige værdi, og 0.333 er endnu nærmere. Computeren omsætter de fleste af de tal, den arbejder med til *binære brøker*, og denne omsætning er sjældent helt præcis. Omsætningen kan blive særligt besynderlig for tal som 1, 10, 100 og .1, .01, .001, – altså alle titalspotenserne. For at undgå at få udskrifter som fx 3−2 = .9999999, runder computeren sådanne tal op eller ned, inden den udskriver dem. Det er ikke alle computere der gør det – så vær glad for at du købte en Amstrad! Fig. 3.18 viser hvordan Amstrad'en klarer brøker som ⅓, 1/11 og 10/11. På skærmen ser du tallene afrundet til ni decimaler, men i computerens lager opereres der med betydeligt flere decimaler. Afrundingen bevirker at additionen af brøkerne giver de rigtige resultater.

```
10 CLS
20 PRINT 1/3,2/3
30 PRINT 1/11,10/11
40 PRINT 1/3+2/3,1/11+10/11
```

Fig. 3.18 Sådan klarer computeren »skæve« tal. Meget store og meget små tal udskrives på *standard form.*

Når du kører programmet, kan du se at computeren ikke skriver brøken 1/11 på normal decimalform. Hvis du arbejder med meget små og meget store tal (fx i fysik og kemi), kender du allerede alt til denne måde at skrive tallene på. Det kaldes udskrift på *standard form.* Et tal på standard form består af et tal mellem 1 og 10, men ikke lig med nogen af dem, ganget med en potens af 10. Tag fx et tal som 132000. Flytter vi 5 decimaler til venstre (det svarer til at dividere med 10 i 5'te potens), så får vi 1.32. For stadig at få samme værdi må vi derfor gange 1.32 med 10 i 5'te – eller som vi skriver her E5, (E'et står for *exponent*). Tallet 132000 kan derfor skrives 1.32E5.

Lad os nu tage et mindre tal som fx 0.00036. Det er det samme som 3.6 gange ti i minus fjerde, eller 3.6E−4, minusset betyder at vi denne gang skal flytte fire decimaler til *højre* for at få 3.6. Man kan, som på andre hjemmecomputere, godt indtaste tal på standardform. Omsætningen sker automatisk, og maskinen vil kun udskrive tal på denne form, når det er nødvendigt – dvs. når det tal der skal udskrives er på mere end ni cifre.

De fleste computere kan lagre tal på langt mere nøjagtig måde inden for et vist område – kaldet *heltal variable.* Et heltal er for Amstrad'en et helt tal, der ligger mellem −32768 og +32767. Et *heltal variabelnavn* er et variabelnavn efterfulgt af % tegnet. Hvis du tildeler et tal til en heltal variabel, kan du kun bruge tal inden for det nævnte område, og alle decimaler bliver automatisk strøget. Du vil således få fejlmeldingen »Overflow« hvis du fx skriver:

```
a% = 32800
```

Fig. 3.19 viser hvordan decimalerne bliver strøget. Variablen x, der har værdien 3.7, bliver i linie 40 tildelt x%, og i linie 50 udskrives x% som 4. Decimalen .7 bliver rundet op til 1. Dette er usædvanligt, idet mange andre computere i stedet ville smide .7 væk og skrive et 3-tal ud.

Der er to fordele ved at anvende heltal variabler. Den ene er at enhver udregning vi laver – bortset fra division – kan udføres med hele tal *uden* at afrundinger er nødvendige. Undtagelsen er division, fordi decimalerne bliver strøget, som linie 70 viser. Den anden fordel er at heltal variabler fylder mindre i hukommelsen. Og et program, der bruger heltal variabler kører hurtigere end programmer, der bruger almindelige variabler. Man kan imid-

```
10 CLS:x=3.7
20 PRINT"x er et tal lig med"x
30 PRINT"x% er et helt tal."
40 x%=x
50 PRINT"Værdien af x% er"x%
60 y%=7/5
70 PRINT"7/5 er"y%"i hele tal!"
```

Fig. 3.19 Brugen af heltal. De tager ikke så meget plads op i hukommelsen og de kan være mere nøjagtige – men ikke i divisioner!

lertid godt udføre divisioner med heltal, hvis man bruger \ (\ sidder på tasten lige over CTRL) og MOD funktionerne. \-tegnet betyder en division af heltal op *i* heltal. Hvis du fx skriver:

PRINT 7\3

vil du få resultatet 2. Det er fordi 7/3 er 2 plus nogle decimaler, og heltal division stryger jo decimalerne. MOD bruges så til at finde *resten* efter heltalsdivisionen. Skriv:

PRINT 7 MOD 3

og du får resultatet 1. Det er *resten* efter at 7 er divideret med 3. \ og MOD kan især bruges i mere avanceret programmering – men det falder udenfor denne bogs rammer.

Ud over at bruge de indbyggede funktioner, kan du også *definere* dine egne *funktioner*. Dvs. at du kan gøre brug af en bestemt matematisk funktion flere gange i et program uden at skulle skrive den mere end en gang. Fig. 3.20 viser én måde at bruge en *defineret funktion,* på. Det er en simpel funktion, der her er brugt som eksempel, og du vil næppe bruge DEF FN til noget så let, men eksemplet er nemt at forstå. Linie 30 og 40 beder om INPUT af tre tal. Linie 50 udskriver så et tal, der kaldes FNsum(a,b,c). Men det giver kun mening, fordi udtrykket er blevet defineret *tidligere i programmet*, nemlig i linie 10, der starter med DEF FN. Når du har skrevet ordren i denne linie, skal du derefter skrive det navn, du har valgt til funktionen. Og sidste del af

```
10 DEF FN sum(a,b,c)=a+b+c
20 CLS
30 PRINT"Indtast venligst tre tal"
40 INPUT a,b,c
50 PRINT"Summen er ";FNsum(a,b,c)
```

Fig. 3.20 Brug af en meget simpel defineret funktion.

linien skal angive funktionen og de *parametre*, der indgår i den. I dette tilfælde skal de tal, der bliver tildelt variablerne a, b og c, lægges sammen. Det hele skal stå i samme linie. DEF FN er altså en slags formel, som maskinen vil bruge, hver gang den finder et FN i et program. I dette eksempel er DEF FN placeret i programmets første linie, for at sikre at maskinen kender funktionen, når den skal bruge den. Hvis FN kommer før DEF FN i et program, vil der komme en »Syntax error« melding, når maskinen kommer til det første FN.

```
10 DEF FNhypot(a,b)=SQR(a^2+b^2)
20 CLS
30 INPUT"Opgiv kateterne i en retvinklet
 trekant ";x,y
40 PRINT"Den tredie side er ";FNhypot(x,
y)
```

Fig. 3.21 Et andet eksempel på en defineret funktion. Her udregnes længden af den længste side i en retvinklet trekant (hypotenusen).

Fig. 3.21 viser et andet eksempel på denne funktion. Her bliver definitionen også giver tidligt i programmet. FNhypot udregner kvadratroden af a ↑ 2+b ↑ 2, men i linie 30 og 40 bruges variablerne x og y som siderne i en retvinklet trekant (kateterne)! Forklaringen er at DEF FN fortæller computeren, hvad den skal gøre med to tal, *og så er det ligegyldigt, hvad de hedder.* Når du bliver lidt skrappere til at programmere, vil du opdage at det gør tingene nemmere at have med at gøre.

Slutbemærkning

Hvis du engang sidder med et program, hvor der er en masse variabler af samme slags, kan du spare en masse skrivearbejde ved at bruge en anden type DEF ordrer. Hvis du fx skriver DEFINT a-z, så bliver alle variable, der starter med bogstaverne a-z automatisk lavet om til heltal variable, og du behøver ikke skrive a%, b% c% osv i programmet – blot a, b, c osv. Tilsvarende kan du bruge DEFSTR til at få bogstaverne til at betyde strenge, og DEFREAL til at betyde reelle tal. Du kan også blande dem ved fx at skrive DEFSTR a-i, o-z:DEFINT j-n, – så bliver alle variabler med bogstaverne j-n heltal variable, mens de andre bliver strenge. Det kan spare indtastning af en masse $ og % tegn!

Kapitel 4.

Igen og igen og igen

Noget af det en computer er allerbedst til, er at køre et sæt ordrer igen og igen, og alle computere har et ordresæt, der kan lave gentagelser. Amstrad'en har også en del af disse »gentagelses«-ordrer – ja, faktisk flere end de fleste hjemmecomputere har. Vi starter med en af de mest enkle gentagelsesordrer – GOTO.

GOTO (»gå til«) betyder præsis det man skulle forvente – gå til et andet linienummer. Normalt køres et program efter stigende linienumre. Dvs at der startes i den linie, der har det laveste nummer, og at programmet køres linie for linie frem til den sidste linie, der altså har det højeste nummer. Når man bruger GOTO bliver denne rækkefølge ændret, sådan at linierne køres i en »forkert« rækkefølge, eller at linierne køres igen og igen. GOTO-ordren skal efterfølges af et mellemrum og et linienummer, – og der kommer en fejlmelding, hvis mellemrummet udelades.

Fig. 4.1 viser et eksempel på en meget enkel gentagelsesordre eller en *løkke,* som det kaldes. Linie 10 indeholder en almindelig PRINT ordre. Når linie 10 er udført, fortsætter programmet til linie 20, som sender det tilbage til linie 10 igen. Det er en *uendelig løkke,* og den vil fylde hele skærmen med sætningen:

Amstrad CPC464 fylder skærmen helt ud!!

indtil du taster ESC for at »bryde løkken«. Enhver løkke, som kører i det uendelige kan stoppes med ESC tasten. Den stopper (break = »stop«) programmet – men dog ikke helt. Hvis du derefter trykker en anden tast ned, vil programmet køre videre. Senere skal vi se, hvordan dette kan bruges når man skal finde fejl i et program. Hvis du vil stoppe programmet helt, så du fx kan lagre det eller ændre det, skal du taste ESC en gang til.

```
10 PRINT"Amstrad CPC464 fylder skærmen h
elt ud!!"
20 GOTO 10
```

Fig. 4.1 En meget enkel løkke. Den stoppes ved at taste ESC to gange.

Prøv nu løkken i fig. 4.2, hvor der sker lidt mere. Her udskrives der et nyt tal hver gang computeren »har været gennem løkken« en gang. I linie 10 får variablen n værdien 10. Værdien udskrives derefter i linie 20 og linie 30 opskriver derefter n. I linie 40 etableres løkken, således at maskinen lynhurtigt tæller fra 10 og op – på skærmen. Her skal ESC også bruges til at stoppe programmet, og du kan prøve at sætte det i gang igen ved at trykke en anden tast ned. Som før stoppes programmet helt med to gange ESC.

```
10 CLS:n=10
20 PRINT n
30 n=n+1
40 GOTO 20
50 REM brug ECS ECS til at afbryde med
```

Fig. 4.2 Denne løkke tæller fra 10 – meget hurtigt. Her stoppes programmet også helt med to gange ESC.

Nu er det jo ikke særligt smart at have sådan nogle løkker, der bare kører og kører, og faktisk prøver vi at undgå GOTO-løkker! I visse tilfælde er der ikke andre muligheder, men Amstrad'en har to andre – den ene er FOR ... NEXT løkken. Denne type løkke bruger to nye ordrer FOR og NEXT (»først ... derefter«). De ordrer, der bliver gentaget, er dem der står mellem FOR og NEXT. Fig. 4.3 viser et elementært eksempel på en FOR ... NEXT løkke. Linien med FOR skal også indeholde en talvariabel, som bruges til at tælle, og to tal, der angiver hvor der tælles fra og til,(en FOR ... NEXT løkke kaldes ofte en *tæller*). I eksemplet bruges n som tæller-variabel, og den tæller fra 1 til 10. NEXT står i linie 40, og alt mellem linie 20 og 40 vil derfor blive gentaget.

```
10 CLS
20 FOR n=1 TO 10
30 PRINT"Amstrad slår alle de andre!"
40 NEXT
```

Fig. 4.3 FOR ... NEXT løkken laver et bestemt antal gentagelser.

I dette tilfælde står der en PRINT ordre, og programmet vil derfor skrive 'Amstrad slår alle de andre!' ti gange på skærmen. Første gang løkken køres igennem er tællervariablen (n) 1, og sætningen udskrives. Når programmet når frem til NEXT opskrives n's værdi – fra 1 til 2 i dette tilfælde. Derefter undersøges det om grænsen på 10 er nået. Hvis den ikke er det, bliver linie 30 gentaget, og sådan vil det fortsætte indtil n er større end 10, – vi ser nærmere på det senere. Resultatet bliver 10 gentagelser.

Man kan godt have flere løkker i et program. Fig. 4.4 viser et eksempel på én løkke *inden i en anden*, – dvs. at den ene løkke skal være helt omsluttet af den anden. Vi kan altså her tale om en indre og ydre løkke i dette program. Den ydre løkke starter i linie 20 med variablen n, der går fra 1 til 10. Linie 30 er en del af den ydre løkke, og den udskriver n's aktuelle værdi. I linie 40 har vi den indre løkke. Den er nødt til at bruge et andet variabelnavn, og den skal både starte og slutte, før den ydre løkke slutter. Vi har brugt variablen j, og vi har ikke sat noget ind mellem FOR-delen og NEXT-delen. Så alt hvad der sker her er, at tiden spildes – programkørslen forsinkes – og det betyder, at der går lidt mere tid mellem hver gennemkørsel af den ydre løkke. Inden en ny løkke starter, slettes skærmen i linie 50. Resultatet af alt dette, er et program, der udskriver tallene fra 1 til 10, sletter skærmen før et nyt tal udskrives og gør det hele så langsomt, at vi kan følge med i hvad der sker. I eksemplet har vi brugt NEXT ved slutningen af hver løkke. Vi kunne også have brugt NEXT j i linie 40 og NEXT n i linie 50, hvis vi ville, men det er ikke afgørende. Det betyder at programmet kører lidt langsommere, – selvom det jo er ligegyldigt her. Når du bruger NEXT j og NEXT n, skal du være helt sikker på at få dem i den rigtige rækkefølge. Får du dem byttet om, kommer der en »NEXT missing« (»NEXT mangler«) fejlmelding, fordi NEXT'erne ikke passer sammen med FOR'erne i dette tilfælde. Du får i øvrigt samme fejlmelding hvis du udelader et NEXT.

```
10 CLS
20 FOR n=1 TO 10
30 PRINT"Vi er nået til ";n
40 FOR j=1 TO 1000:NEXT
50 CLS:NEXT
```

Fig. 4.4 Et program med en løkke inden i en anden

Allerede nu kan du se hvor anvendelig FOR ... NEXT er, men vi er slet ikke færdige. De løkker, vi har set indtil nu, har talt op ved at opskrive talvariablen. Det er ikke altid det vi ønsker, og vi kan lave en tilføjelse med ordren STEP (»trin«) til FOR-linien, for at lave om på ændringen af variablens værdi. Vi kan fx skrive:

```
FOR n=1 TO 9 STEP 2
```

som ville få n's værdi til at stige således: 1, 3, 5, 7. Når vi *ikke* skriver STEP bruger maskinen altid *standardværdien* +1.

I fig. 4.5 har den ydre løkke et trin på −1, således at der tælles ned. N starter med værdien 10, og den nedskrives ved hver gennemkørsel af løkken.

```
10 CLS
20 FOR n=10 TO 1 STEP -1
30 PRINT n;" sekunder - nedtælling"
40 FOR j=1 TO 800:NEXT
50 CLS:NEXT
60 PRINT"FYR!!!"
```

Fig. 4.5 Et ned-tællingsproram, ved brug af STEP.

I linie 40 er der igen indlagt en forsinkelse, så nedtællingen sker passende langsomt. Hvis vi vil have nedtællingen til at foregå hurtigere, er det nemmeste at bruge en heltal variabel, som fx j% i stedet for j. Men gør vi det er vi dog afskåret fra at bruge trin, der indeholder decimaler som fx STEP .1.

Når vi arbejder med løkker får vi af og til brug for tællerens værdi, fx n eller j, når en gennemkørsel af løkker er afsluttet. Hvis vi vil vide, hvad tællerens aktuelle værdi er, kan vi fx bruge programmet i fig. 4.6. Det indeholder to løkker – en der tæller op, og en der tæller ned. Efter hver gennemkørsel udskrives tællervariablens værdi. Vi ser at n's værdi er 6 i linie 50, efter FOR n=1 TO 5-løkken er afsluttet, og 0 efter FOR N=5 TO 1 STEP −1-løkken er slut. Hvis du skal bruge n's værdi, eller hvad din tæller nu måtte hedde, så må du huske, at den er *ændret med et trin mere* i slutningen af løkken. Du kan naturligvis også bruge negative værdier af n i løkker. Hvis du bruger heltal variable som fx n% eller j%, for at få programmet til at køre hurtigere, må du huske at der er grænser for værdierne. Du må ikke bruge heltal, der er større end 32767 eller mindre end −32768. Hvis du overskrider dette område, får du en »Overflow« melding.

```
10 CLS
20 FOR n=1 TO 5
30 PRINT n
40 NEXT
50 PRINT"n er nu ";n
60 FOR n=5 TO 1 STEP -1
70 PRINT n
80 NEXT
90 PRINT"n er nu ";n
```

Fig. 4.6 Her får vi løkkens tæller oplyst, efter løkken er gennemkørt.

Noget af det allerbedste ved FOR ... NEXT løkkerne er imidlertid, at de ikke bare kan bruges med tal, men også sammen med talvariable. Fig. 4.7 viser et lille eksempel på dette. Bogstaverne a, b og c bliver tildelt hver sit tal

i linie 20, og derefter bruges de i FOR ... NEXT-løkken i linie 30. Grænserne sættes af a og b, mens trinnet fremgår af udtrykket b/c. Generelt gælder det, at du kan bruge alt, hvad der repræsenterer tal eller som kan resultere i et tal i en løkke – som her.

```
10 CLS
20 a=2:b=5:c=10
30 FOR n=a TO b STEP b/c
40 PRINT n
50 NEXT
```

Fig. 4.7 En løkke, der arbejder med talvariable.

Løkker og beslutninger

Nu er det på tide at se nogle løkker i funktion, fremfor blot at se dem demonstreret. En simpel anvendelse er summering af tal. Vi vil have programmet til at tage imod de tal, vi indlæser, lægge dem sammen og endelig udskrive resultatet. På baggrund af hvad vi har lavet indtil nu, skulle det være ret let at se, hvad vi skal gøre, for at få et bestemt antal tal lagt sammen, som fx 10 ad gangen. Og det er præcis det, programmet i fig. 4.8 gør.

```
10 sum=0:CLS
20 PRINT TAB(2)"ADDITIONSPROGRAM TIL UDR
EGNING AF SUM"
30 PRINT:PRINT"Indtast tallene"
40 PRINT"Programmet vil give summen."
50 FOR n=1 TO 10
60 PRINT"Tal nr.";n;" er";
70 INPUT j:sum=sum + j
80 NEXT
90 PRINT:PRINT"Summen er ";sum
```

Fig. 4.8 Et program der adderer 10 tal.

Programmet starter med at sætte en talvariabel »sum« lig nul. Det er denne talvariabel, der skal bruges til at holde styr på summen, og den skal naturligvis have værdien nul fra start. Nu gør Amstrad'en faktisk det automatisk, men det er en god vane at sikre sig, at værdierne fra start er sat til det, vi ønsker. Hvis vi skulle bruge ordet TO til at være vores tal variabel, så ville det være umuligt, fordi TO er et reserveret ord i Amstrad'ens BASIC. Hvis vi kommer til at bruge et reserveret ord som variabelnavn, får vi en »Syntax error« melding, når programmet køres.

Linie 20 til 40 giver brugeren instruktioner, og selve programmet starter i linie 50. Her sørger FOR ... NEXT løkken for at linie 60 og 70 køres ti gange. Linie 60 fortæller dig hvor mange tal du har indlæst, ved at udskrive n's aktuelle værdi, og linie 70 lader dig foretage INPUT af et tal, som bliver tildelt talvariablen j. I den anden halvdel af linie 70 bliver j lagt til sum, og løkken gentages. Når programmet er slut indeholder variablen sum resultatet af hele udregningen.

Det er selvfølgelig meget godt, – men hvor tit skal du lægge præcis ti tal sammen? Det ville være smartere, hvis vi kunne stoppe sammenlægningen ved at give maskinen et signal af en slags, fx ved at skrive 0 eller 999. En sådan værdi kan vi kalde et *slutsignal*, – dvs en indlæsning som tydeligvis ikke er et almindeligt INPUT, men blot et signal. I et sammenlægningsprogram er 0 jo et udmærket slutsignal, for det gør ikke noget hvis det skulle blive lagt til de øvrige tal. Men vi får brug for en ny ordre her, for at kunne give maskinen bedsked om vores slutsignal, og denne nye ordre er IF (»hvis«).

IF skal efterfølges af en *betingelse*. Her kan du bruge betingelser som fx IF n=20, eller IF nm$="slut". Efter betingelsen kan du bruge ordet THEN (»så«), og derefter skal der stå, hvad der så skal ske – altsammen i samme linie. Og her kan du simpelt hen stoppe løkken, hvis betingelsen er opfyldt. Det kan gøres i programmet ved at skrive et linienummer efter THEN. Hvis denne linie er den sidste i programmet, vil programmet stoppe når betingelsen er opfyldt.

Nu lyder det måske ret kompliceret, men prøv en gang at se på fig. 4.9. Vi kan ikke bruge en FOR ... NEXT løkke her, fordi vi netop ikke ved hvor mange gange vi vil køre løkken igennem, så derfor bruges IF ... THEN til at styre løkken. Først kommer instruktionerne til brugeren, og variablen nulstilles i linie 40. Hver gang du indlæser et tal, bliver tallet lagt til summen i linie 60, og linie 70 udskriver den foreløbige sum. I linie 80 styres løkken. Her testes (undersøges) det om betingelsen er opfyldt – i dette tilfælde testes det om n *ikke* er nul. Hvis det ikke er tilfældet, starter løkken forfra i linie 50. Det nye tegn (< >) laves ved først at taste < (mindre end) og derefter > (større end), – sammen betyder de »forskellig fra«. Du kan skrive GOTO efter THEN, men det er ikke nødvendigt. Da det er nemmere at lade være med at skrive det, har jeg udeladt det.

Virkningen af det hele bliver, at hvis det tal du skrev i linie 50 ikke er et nul, så sender linie 80 programmet tilbage i linie 50 efter et nyt INPUT. Sådan vil det blive ved indtil du indlæser et nul. Når det sker, er testen i linie 80 negativ, (dvs. at det ikke er opfyldt, at n er forskellig fra nul), og programmet går videre til linie 90. Her meddeles det, at programmet er slut, og da der ikke er flere linier, standser programmet.

Hvis du taster ENTER uden først at have skrevet et tal, opfatter programmet det som det samme som nul, og programmet stopper. Det er ikke alle maskiner, der gør det.

Dette program har kun en enkelt test i forbindelse med IF, men der kan godt være flere. Fx kunne du bestemme dig for at programmet skulle slutte, hvis der enten blev indlæst 0 eller 999. I det tilfælde skulle IF linien se sådan ud:

```
IF n<>0 AND n<>999 then 50
```

– og så var der to betingelser, knyttet sammen af AND (og). Du kan også bruge ordet OR (eller) i en test, fx:

```
IF x=0 OR X=999 THEN
```

men du skulle passe på at 999 ikke blev lagt til summen i dit additionsprogram! Man skal være ret omhyggelig med, hvor man placerer nogle af disse tests!

```
10 CLS:PRINT TAB(8)"ET ANDET ADDITIONSPR
OGRAM"
20 PRINT:PRINT"Programmet vil lægge tal
sammen for dig"
30 PRINT"indtil du indtaster et nul."
40 sum=0
50 INPUT"Indtast tal ";n
60 sum=sum+n
70 PRINT"Summen er nu";sum
80 IF n<>0 THEN 50
90 PRINT"Slut på udregningen"
```

Fig. 4.9 Et additionsprogram, der ikke kan bruge FOR ... NEXT. I linie 80 undersøges det om løkken skal køres endnu en gang.

Fig. 4.10 viser de forskellige tests du kan lave ved hjælp af IF. De matematiske tegn bruges for en nemheds skyld, men husk at samtlige tests også kan bruges i forbindelse med strenge. For øjeblikket er det nemt at se hvad = og <> betyder, men vi skal senere se på hvordan < og > også kan bruges.

Tegn	Betydning
=	Lig med
>	Venstre side er større end højre side
<	Venstre side er mindre end højre side

Tegnene kan sættes sammen på følgende måder:

<>	Forskellig fra, ikke lig med
>=	Venstre side er større end eller lig med højre side
<=	Venstre side er mindre end eller lig med højre side

Bemærk: Når < eller > tegnet sættes sammen med =, så *skal* < eller > tegnet komme først. En kombination som => vil altid give en fejlmelding

Fig. 4.10 De matematiske tegn bruges til at sammenligne tal og talvariable.

IF ... THEN ... ELSE

IF ... THEN udgør en test, som kan bruges i mange programmer, og IF ... THEN kan tilmed udbygges. Du kan nemlig bruge ordet ELSE (»ellers«) til at lave endnu en test, eller til at få programmet til at gøre noget andet. Et eksempel vil nok gøre det mere forståeligt, så se en gang på fig. 4.11.

```
10 PRINT TAB(12)"PLAT ELLER KRONE"
20 PRINT:PRINT TAB(5)"(Tast E for at sto
ppe spillet)":PRINT
30 n=1+INT(2*RND(2))
40 IF n=1 THEN PRINT"PLAT", ELSE PRINT"K
RONE",
50 PRINT"(Stop: Tast E)"
60 INPUT a$:IF a$="E" THEN END ELSE 30
```

Fig. 4.11 Et simpelt plat og krone program, hvor ELSE kommer i funktion når n ikke er 1.

Det er et helt enkelt plat og krone spil. Linie 10 og 20 giver som sædvanlig instruktioner til brugeren, og i linie 30 starter løkken. Og det er her »spillet« foregår! RND betyder »vælg tilfældigt« (engelsk: RaNDom = tilfældig), og når RND efterfølges af et positivt tal i parentes, vil maskinen vælge et tilfældigt tal mellem 0 og 1. Dette decimaltal bliver aldrig helt lig nul og aldrig helt lig 1. Når dette tal ganges med 2, får vi et tal der ligger mellem 0 og 2, tallet kan ligge meget tæt på enten 0 eller 2, fx kan det godt være 1.999999. Når vi så bruger INT, og tager den hele del af det reelle tal, får vi et tal der enten er 1 eller 0. Lægger vi derefter 1 til dette tal får vi dermed enten tallet 1 eller 2, og det kan vi netop bruge i vores plat og krone spil. Testen bliver lavet i linie 40, – hvis n er 1 bliver PLAT skrevet ud, hvis n ikke er 1 bliver KRONE skrevet ud. I dette eksempel bliver ELSE altså brugt til at vælge den anden mulighed. I linie 60 bruger vi ELSE igen, denne gang til at afgøre, om spillet skal stoppe eller fortsætte. Nu kan du jo prøve at udbygge dette spil, fx til at spørge brugeren om at gætte næste »kast«, og så lade maskinen føre regnskabet automatisk!

Det vigtige ved ELSE er, at der kan være en valgmulighed. Hvis testen er positiv kan en bestemt ordre blive udført (en melding udskrevet, måske), og hvis testen er negativ kan ELSE så bruges til at få en anden ordre udført (en anden melding udskrevet, måske). Senere skal vi vende tilbage til programmer med flere tests. Nu skal vi først se på en ny slags løkke, der giver mange flere muligheder.

WHILE ... WEND

Det er kun de færreste hjemmecomputere, der har andre løkker end FOR ... NEXT og IF ... THEN, ja, der er mange, som ikke engang har ELSE. Men AMSTRAD har også en mere avanceret type løkke – WHILE ... WEND. Princippet er, at du starter din løkke med en betingelse, og derefter kan du bruge det antal linier, du ønsker, til at få udført et eller andet, og så slutter løkken ved ordet WEND (WHILE betyder »mens« eller »så længe som«, WEND er dannet af WHILE og END (»slut«).

```
10 CLS:PRINT TAB(7)"ET TREDIE ADDITIONSP
ROGRAM":sum=0:n=1
20 PRINT:PRINT"Indtast tal til addition,
 indtast 0 for":PRINT"at slutte"
30 WHILE n<>0
40 INPUT n
50 sum=sum+n
60 PRINT"summen er indtil nu";sum
70 WEND
80 PRINT"programmet er slut"
```

Fig. 4.12 Et additionsprogram, hvor WHILE ... WEND løkken anvendes.

Igen er et eksempel en god hjælp, så se engang på fig. 4.12. Her har vi endnu en version af vores additionsprogram. Vi sætter som sædvanlig sum = 0, men vi giver også n en værdi, her allerførst i programmet. Vi sætter n = 1 af hensyn til WHILE ... WEND løkken. Løkken starter nemlig i linie 30 med betingelsen WHILE n <> 0, og det betyder at løkken gentages som n ikke er nul. Men når programmet starter sættes alle variabler jo til nul, og derfor ville programmet stoppe med det samme, hvis vi ikke havde lavet det lille kunstgreb at sætte n = 1. Læg mærke til at 1-tallet ikke kommer med i additionen!

De forskellige trin i løkken er velkendte, så dem går vi let hen over. I linie 70 dukker WEND op. Her afsluttes løkken og sendes tilbage til WHILE

testen. Den eneste hage er, at testen laves i begyndelsen af løkken, og derfor skal der være en værdi at teste, ellers kan løkken ikke køres.

Lad os tage et andet eksempel. I fig. 4.13 har vi en READ ... DATA ordre indeni løkken. Programmet indlæser simpelt hen et antal poster fra DATA listen, indtil det støder på et »x«. I linie 30 defineres a$ som et mellemrum, – det fås ved at bruge mellemrumstasten én gang mellem det to sæt gåseøjne. Løkken starter i linie 30, med betingelsen at løkken skal gentages indtil a$ tilskrives indholdet »x«. I løkken skrives værdien af a$ ud (som altså er tom første gang), og derefter tilskrives a$ en ny værdi fra DATA linien. I linie 60 »vender« løkken tilbage til linie 30 til en test af den nye værdi af a$. Programmet skriver alt i alt en nydelig lille liste på skærmen.

```
10 CLS
20 a$=" "
30 WHILE a$<>"x"
40 PRINT a$
50 READ a$
60 WEND
70 DATA Andersen,Hansen,Jensen
80 DATA Olsen,Petersen,x
```

Fig. 4.13 Et andet eksempel på en WHILE ... WEND løkke, her brugt i forbindelse med READ ... DATA.

Programmet i fig. 4.14 bruger WHILE ... WEND løkken på en tredje måde. Her bruges den som en *fælde*. En fælde bruges til at fange fejl med, dvs. til at vurdere om indlæsninger, der kommer til programmet er brugbare. Hvis du giver programmet en forkert indlæsning, som fx et bogstav, eller et tal som ikke ligger mellem 1 og 5, så vil fælden fange fejlen, give en fejlmelding på skærmen, og bede om en ny indlæsning. For en ekspert (som dig, når du er færdig med bogen!), kan problemet let løses med et GOTO, som vil få programmet sat i gang igen. Men for den mere uerfarne bruger kan en fejlmelding ødelægge det hele, og en masse indlæsninger kan gå tabt!

```
10 INPUT"Vælg et tal mellem 1 og 5";n
20 WHILE n<1 OR n>5
30 PRINT"Fejltastning - kun tal mellem 1
 og 5"
40 INPUT n
50 WEND
60 PRINT"Du valgte"n
```

Fig. 4.14 WHILE ... WEND løkke brugt som en fælde.

I dette eksempel bliver du bedt om at indlæse et tal mellem 1 og 5. Hvis det tal, der indlæses, ligger indenfor dette område, er alt i orden, men hvis ikke (prøv det!) træder WHILE ... WEND-løkken i funktion. Den udskriver en fejlmelding, og giver dig en chance (eller evt. flere) til at lave en korrekt indlæsning. Det er princippet i en fælde, og WHILE ... WEND er ideel til formålet. Læg mærke til at WHILE ... WEND udmærket kan bruges sammen med strenge, selv om det mest har handlet om tal her. Du kan fx have en linie som:

```
WHILE navn$<>"x"
```

som kan bruges ved indlæsning af navne til en liste, eller:

```
WHILE svar$<>"J" AND svar$<>"N"
```

som fælde for »J« eller »N« svar.

En enkelt tast som svar

Indtil videre har vi prøvet at skulle svare J eller N i forbindelse med en INPUT sætning, hvor vi til sidst skulle taste ENTER. Det har den fordel, at der er en mulighed for at lave svaret om, fordi det kan slettes igen, og et nyt kan indtastes. Hvis det skal gå lidt kvikkere, kan ordren INKEY$ bruges. INKEY$ er en funktion, der undersøger om en tast bliver trykket ned. Det kan den gøre meget hurtigt, og normalt må vi derfor placere INKEY$ i løkke som kan gentages indtil en tast bliver trykket ned. Fig. 4.15 viser en løkke, der har sådan en »ventefunktion«. WHILE ... WEND-løkken er meget enkel – den vil blive ved med at køre så længe a$ er *tom*. En *tom streng* laves ved at sætte to gåseøjne uden noget imellem. Hver gang INKEY$ bruges, undersøger computeren om tastaturet bliver betjent. Hvis der ikke sker noget, er INKEY$ tom, og WHILE ... WEND-løkken kører videre. Men når en tast bliver trykket ned, bliver løkken brudt, og programmet kan køre videre. Det er en meget god lille programstump at benytte, når der skal gives instruktioner på skærmen. Brugeren kan så bruge den nødvendige tid på at læse anvisningerne, og så trykke en tilfældig tast ned, for at komme videre. Som i tidligere eksempler betyder en »tilfældig tast« en *karakter*-tast, for nogle ta-

```
10 CLS
20 PRINT"Tryk på en tast..."
30 WHILE INKEY$="":WEND
40 PRINT"Slut"
```

Fig. 4.15 INKEY$ bruges i en WHILE ... WEND løkke, for at finde ud af om en (tilfældig) tast bliver trykket ned.

ster har ingen effekt her (fx CTRL og SHIFT), mens ESC vil afbryde programmet.

Når INKEY$ forbindes med en karakter, kan vi tildele INKEY$ en strengvariabel, k$. På den måde kan vi, når en tilfældig tast trykkes ned, tildele den karakter, der derved fremkommer, til k$. Og dermed kan vi lave en test med den streg vi får. Vi illustrerer det igen med et eksempel – fig. 4.16 viser hvordan INKEY$ kan bruges til at få et »J« eller »N« svar, – med en indbygget fælde. I linie 20 startes en løkke, hvor INKEY$ løbende testes. Først i det øjeblik, hvor en tast trykkes ned er k$ ikke tom, og løkken brydes derved. Så bliver k$ igen testet, for at se om svaret kan bruges. Hvis svaret ikke duer, oplyses det på skærmen, og det fortælles der ud over hvad, der blev svaret.

```
10 PRINT"Tast J eller N"
20 k$=INKEY$:IF k$=""THEN 20
30 IF k$<>"N" AND k$<>"J"THEN PRINT"Fork
ert svar - kun J eller N"
40 PRINT"Dit svar er ";k$
```

Fig. 4.16 INKEY$ bruges til at få et »J« eller »N« svar.

Der findes også en anden måde at teste en bestemt tast på. Her bruges INKEY-funktionen, og dens virkemåde baserer sig på, at hver tast har hver sin talkode. Talkoden kan du finde i manualen i appendix III page 16. INKEY bruges *ikke* til at undersøge om en tilfældig tast aktiveres, men derimod til at undersøge om en *bestemt* tast trykkes ned. Fig. 4.17 viser dette med INKEY(18), som undersøger om ENTER trykkes ned. Du kan bruge INKEY(18) = 0 for at teste om ENTER trykkes ned, eller INKEY(18) = −1 til at teste om ENTER *ikke* trykkes ned. Du kan også bruge bestemte tal til at undersøge om SHIFT eller CTRL (eller begge) trykkes ned *samtidig* med den taste du tester. Fx tester INKEY(18) = 32 om ENTER og SHIFT trykkes ned samtidig, INKEY(18) = 128 tester om CTRL og ENTER trykkes ned samtidig, og INKEY(18) = 160 tester om alle tre taster er trykket ned samtidig.

```
10 PRINT"Tast ENTER, hvis du vil fortsæt
te
20 WHILE INKEY(18)<>0:WEND
30 PRINT"Fortsæt her.."
```

Fig. 4.17 INKEY funktionen, som kan undersøge om en bestemt tast trykkes ned.

INKEY funktionen er meget anvendelig fordi den kan teste alle taster, – også markørpilene (spil!!), eller COPY eller DEL. Ja, selv ESC kan testes på den måde!

Kapitel 5

Mere strengeleg

Streng-funktioner

I kapitel 3 så vi lidt på talfunktioner. Hvis du godt kan lide at arbejde med tal, er det jo også spændende, men streng-funktioner er på mange måder mere interessante. Grunden til det er, at nogle af de mest bemærkelsesværdige og fascinerende operationer, der kan laves med en computer, er *streng-funktioner.* Hvad er en streng-funktion da? Her i bogen er en streng-funktion, enhver operation computeren kan lave med strenge. Det bliver du jo ikke meget klogere af, så lad os se nærmere på dem.

Når vi bruger Amstrad'en er streng en samling karakterer, som er tildelt en strengvariabel, – et navn der ender med et dollartegn. I praksis kan du have så mange karakterer, du vil, i en streng, på Amstrad er grænsen 255 karakterer pr. streng, og det er flere end vi normalt får brug for. Som på de fleste andre computere lagres strengene ved hjælp af det, der hedder ASCII (udtales Askey) koden. Bogstaverne er en forkortelse for »American Standard Code for Information Interchange« (altså en amerikansk standard for informationsudveksling). ASCII-koden har ikke noget at gøre med de tal, der bruges sammen med INKEY-funktionen, så pas på at du ikke blander dem sammen. Fig. 5.1a viser et lille program, du kan indtaste og få ASCII-tegnsættet *) frem på skærmen med, CHR$ kommer vi til senere. På fig. 5.1b vises ASCII-tegnsættet, som det tager sig ud, når det bliver skrevet ud på en almindelig dansk skrivehjulsprinter.

```
10 FOR n=32 TO 127
20 PRINT n;
30 PRINT CHR$(n),
40 NEXT
```

Fig. 5.1a Et lille program, der kalder ASCII koden fra 32 til 127 frem på skærmen. Hvis du bruger MODE 2, kan det hele være der på en gang.

---

*) På dansk bruges »karakterer« og »tegn« lidt i flæng. Her bruges udtrykket »karakterer« mest, fordi vi i almindelig tale kun forbinder »tegn« med bogstaver og tal. O.A.

```
32              33 !            34 "
35 £            36 $            37 %
38 &            39 '            40 (
41 )            42 *            43 +
44 ,            45 -            46 .
47 /            48 0            49 1
50 2            51 3            52 4
53 5            54 6            55 7
56 8            57 9            58 :
59 ;            60 <            61 =
62 >            63 ?            64 §
65 A            66 B            67 C
68 D            69 E            70 F
71 G            72 H            73 I
74 J            75 K            76 L
77 M            78 N            79 O
80 P            81 Q            82 R
83 S            84 T            85 U
86 V            87 W            88 X
89 Y            90 Z            91 Æ
92 Ø            93 Å            94 ^
95 _            96 `            97 a
98 b            99 c            100 d
101 e           102 f           103 g
104 h           105 i           106 j
107 k           108 l           109 m
110 n           111 o           112 p
113 q           114 r           115 s
116 t           117 u           118 v
119 w           120 x           121 y
122 z           123 æ           124 ø
125 å           126 ¨           127
```

Fig. 5.1b ASCII-koderne udskrevet på en skrivehjulsprinter med dansk tegnsæt.

Hver karakter er repræsenteret ved et tal, og tallene går fra 32 (repræsenterer mellemrum) til 127, der på skærmen viser sig som et lille ternet mønster, men som på printeren også giver et mellemrum. Du kan tildele karakterer til en streng ved at bruge lighedstegnet. Når tildelingen sker på denne måde, skal du bruge gåseøjne omkring karaktererne. Du kan også lave tildelinger ved at bruge INPUT i forbindelse med READ ... DATA, og så behøver du ikke gåseøjnene.

Talvariablerne bruger ikke den samme slags kode som strengvariablerne, idet der bruges et fast antal koder til tal – ligegyldigt om de er store eller små. Der bruges en slags koder til heltal og en anden slags til almindelige tal (kaldet *reelle* tal). Men da strengene består af en række koder (en for hver karakter) i computerens lager, kan vi lave operationer med strenge, som ikke er mulige med tal. Vi kan fx nemt finde ud af hvor mange karakterer, der er i en streng. Eller vi kan udvælge nogle karakterer fra strengen og erstatte dem med andre. Det er operationer som disse, vi kalder »streng-funktioner«.

LEN tæller op

En af strengfunktionerne kan, som jeg var inde på, finde ud af hvor mange karakterer, der er i en streng. Og da der kan være op til 255 karakterer i en streng, er det meget rart, at en sådan optælling kan ske automatisk, og det klarer LEN. LEN (dannet af engelsk: LENght = »længde«) skal efterfølges af navnet på strengvariablen i parentes, og resultatet af LEN-funktionen er altid et tal, og det kan derfor både udskrives og tildeles en talvariabel. Det er ikke nødvendigt med mellemrum mellem N i LEN og parentesen.

```
10 CLS
20 a$="Du kan udvide din Amstrad"
30 PRINT"Der er";LEN(a$);" karakterer i
den sætning."
40 INPUT"Prøv selv en sætning ";b$
50 PRINT b$;"indeholder";LEN(b$);" karak
terer
```

Fig. 5.2 Her er LEN, medlem af streng-funktion familien.

Fig. 5.2 viser et lille eksempel på LEN's anvendelse. I linie 20 tildeles en variabel til en sætning, og linie 30 fortæller dig så hvor mange karakterer der er i sætningen. Bemærk at ordet »karakterer« ikke er de samme som »bogstaver«. Hvert mellemrum, punktum, komma osv. tæller som en karakter i strengen, fordi de alle er tilknyttet et tal i ASCII-koden. I linie 40 og 50 vises det, hvordan du selv kan indlæse en sætning og finde længden af den. Det kan være meget nyttigt, hvis du fx vil sikre dig at en sætning, som du har indtastet, ikke er for lang for computeren. Det kan bl.a. være aktuelt ved indtastninger til et skema, hvor der kun er plads til fx 20 karakterer i hver kolonne. Hvis du har et langt navn eller en lang adresse, har du sikkert bemærket, at navn eller adresse ofte er forkortet, når brevet er adresseret på EDB. Nu ved du hvorfor!

LEN kan også bruges til andre praktiske opgaver, som vist i fig. 5.3. Dette program bruger LEN i forbindelse med centrering af variablen t$ på en linie. En sådan lille programstump, der kan bruges som en del af større programmer kaldes ofte en *rutine*, og denne rutine kan du netop bruge, når du skriver programmer, fordi det er nemmere end at skulle tælle og beregne hele tiden.

```
10 CLS
20 t$="Amstrad - Årets computer 1984"
30 PRINT TAB((41-LEN(t$))/2);t$
```

Fig. 5.3 Her bruges LEN til at centrere en overskrift.

Princippet i programmet er at LEN finder ud af hvor meget t$ fylder. Længden trækkes så fra 41, og resultatet divideres med 2. Hvis denne division ikke går op, så får vi et TAB-tal, der indeholder .5, men det betyder intet, TAB bruger kun det hele tal. Du kan for øvrigt bruge 41 eller 42 som du vil – er der et ulige antal karakterer i strengen, bliver det pænest med 41 og er der et lige antal, bliver det pænest med 42. Prøv at forbedre programmet med et par IF ... THEN ... ELSE trin! Vi kan bruge denne rutine til alle strenge, der hedder t$. I næste kapitel skal vi se på noget der kaldes *subrutiner* (»underrutiner«), som gør det muligt at indsætte en rutine (en centrering fx) i et program én gang, hvor efter den umiddelbart kan bruges på alle strenge.

STR$ og VAL

Du ved nu at der er nogle operationer, der kan udføres med tal men ikke med strenge, og omvendt er der nogle operationer, der kan laves med strenge, men ikke med tal. Det kan godt være upraktisk, men det er ikke noget problem, for der findes funktioner, der kan ændre strenge til tal – og omvendt. Det gør det muligt at lave beregninger med tal, der har været strenge, og lave strengfunktioner med tal, der førhen har været talvariable. Se på fig. 5.4.

```
10 n$="22.5":v=2
20 CLS:PRINT
30 PRINT n$;" gange";v;" er";v*VAL(n$)
40 PRINT
50 v$=STR$(v)
60 PRINT"Der er";LEN(v$);" karakterer i"
;v" !"
70 PRINT
80 PRINT n$;" adderet med";v$;" giver ";
n$+v$;" ?"
```

Fig. 5.4 Tal kan ændres fra at være talvariable til at være strenge, – og tilbage igen – med STR$ og VAL. Læg mærke til at STR$ indsætter et mellemrum.

Først gør vi n$ til et tal i en strengform, og v til et tal i talform. Linie 30 viser at vi kan lave udregninger med n$. Når vi skriver VAL(n$) (VALue = »værdi«) i stedet for blot n$, kan værdien af n$ bruges som et tal, og udregningen giver det rigtige resultat. I linie 30 bliver tallet v lavet om til en streng, v$. Her skal vi lige være opmærksomme på noget, som er vist i linie 50. Tallet v stod for et 2-tal, som er et en-cifret tal, men ved hjælp af STR$-funktionen bliver tallet til en streng på *to* karakterer! Det skyldes det usynlige mellemrum foran tallene, der giver plads til + og – tegn. STR$ rutinen medregner altid dette mellemrum, så husk at længden af en streng, der har været et tal, altid har en karakter for meget, – med mindre den tomme plads har været optaget af et minus. Linie 80 demonstrerer hvad der kan ske, hvis man glemmer at bruge VAL, og vil lægge to strenge sammen!

Skær af fra venstre

Den næste type operationer, vi skal se på er nogle funktioner, der kan uddrage dele af strenge. På den måde kan vi få en ny streng ud af en større streng. Ved hjælp af disse funktioner kan vi finde ud af hvilke karakterer, der står hvor i en streng.

Hvis det ikke lyder alt for spændende, så prøv en gang at se på fig. 5.5. Her tildeles a$ en streng i linie 20, og i linie 30 tildeles en anden streng. I linie 40 udskrives en sætning som bruger de to første bogstaver i »Amstrad«. Hvordan gik det så til? LEFT$ (»venstre«) funktionen betyder »lav en kopi af strengen fra venstre«. LEFT$ skal efterfølges af to størrelser i parentes, adskilt af et komma. Den første størrelse skal være navnet på den strengvariabel vi skal »skære« i. Den anden størrelse skal være et tal, der angiver hvor mange karakterer vi skal kopiere fra venstre. LEFT$(a$,2) betyder derfor at vi skal kopiere de to første karakterer fra venstre af strengen a$ – dvs. »Am«. Den anden streng i linie 30 bliver så sammenkædet med dette, og vi får den sætning som linie 40 udskriver.

```
10 CLS
20 a$="Amstrad"
30 b$="atør programmering med en profess
ionel maskine."
40 PRINT LEFT$(a$,2)+b$
```

Fig. 5.5 Brug af LEFT $.

For nu at bruge funktionen til noget lidt mere seriøst, så se en gang på fig. 5.6. Programmet bruges til at finde dit navns forbogstaver, og det gøres med LEFT$ og lidt sammenkædning. INPUT-trinene i linie 20 og 30 får indlæst

```
10 CLS:PRINT:PRINT
20 INPUT"Efternavn ";EN$
30 PRINT:INPUT"Fornavn ";FM$
40 PRINT:PRINT
50 PRINT"Her bruger du initialerne ";LEF
T$(FM$,1)+"."+LEFT$(EN$,1)+". når du spi
ller."
```

Fig. 5.6 Vi finder forbogstaverne med LEFT$.

efternavn og fornavn og tildeler dem til variabelnavnene EN$ og FM$. Det ville være mere logisk at bruge FN$ i stedet for FM$, men FN er – som du måske husker – et reserveret ord i BASIC. Så hvis vi havde brugt det havde vi fået en fejlmelding. I linie 50 udskrives dine forbogstaver, ved hjælp af LEFT$, der kopierer første bogstav i hver streng. Bogstaverne sættes sammen med punktummer imellem ved hjælp af sammenkædning. Hvis der er to deltagere i et spil, kan det ofte være en fordel kun at bruge forbogstaverne på point-tavlen, mens hele navnet måske kan anvendes andre steder i spillet.

Og så fra højre

Vi behøver ikke nøjes med at kopiere strenge fra venstre side, – vi kan også gøre det fra højre side. Denne funktion bruges knap så meget som LEFT$, men den er alligevel meget anvendelig. Fig. 5.7 viser, hvordan man kan bruge funktionen til at bruge et ord igen, uden at skulle skrive det en gang til. Men funktionen RIGHT$ (»højre«) kan også bruges til noget mere »fornuftigt«. Fx kan du uddrage de sidste fire cifre i en tal-streng som fx 010-242-7016. Jeg brugte med vilje udtrykket *tal-streng*, for tallene skal være på streng-form, for at kunne bruges sammen med disse funktioner. Hvis du prøver at tildele den ovennævnte tal-streng til en tal-variabel, vil du få et underligt resultat. Hvorfor? – Fordi n = 010-242-7016 vil blive opfattet som en subtraktion, og n vil derfor blive −7248, og det var jo ikke tanken. Hvis du i stedet skriver n$ = "010-242-7016", så vil det kunne fungere.

```
10 CLS
20 a$="Amstrad magi"
30 PRINT:PRINT
40 PRINT"Det hele er ";RIGHT$(a$,4);" fo
r mig."
```

Fig. 5.7 RIGHT$ bruges til at kopiere bogstaverne fra højre side af strengen.

Vi kan få nogle sjove effekter ud af at bruge RIGHT$ og LEFT$. Se fx på fig. 5.8, hvor der sker mærkelige ting med dit navn. Du bliver bedt om at skrive dit navn i linie 20, og det tildeles variablen a$. I linie 30 bruger vi LEN, således at n bliver lig antallet af karakterer i navnet. Tallet nedregner mellemrum og bindestreger – der er vel ingen, der bruger stjerner og spørgsmålstegn! I linie 40 starter en løkke, som kører n gange. Det er linie 50, der laver udskriften. Når j er 1 skrives det første bogstav i dit navn i venstre side af skærmen, og det sidste bogstav i højre side. Når løkken kører næste gang, kommer der en ny linie, hvor der udskrives to bogstaver mere, og sådan fortsætter det, indtil hele navnet er skrevet. Hvis du bruger LEFT$ eller RIGHT$ sammen med et tal, der er større end antallet af karakterer i strengen, får du simpelt hen hele strengen.

```
10 CLS
20 INPUT"Navn ";a$
30 n=LEN(a$)
40 FOR j=1 TO n
50 PRINT LEFT$(a$,j);TAB(21)RIGHT$(a$,j)
60 NEXT
```

Fig. 5.8 RIGHT$ og LEFT$ i funktion sammen.

Og så fra midten

Kopifunktionen MID$ (engelsk: middle = »midte«) kan bruges til meget mere end RIGHT$ og LEFT$. Efter MID$ skal der stå tre poster i parentes – adskilt af kommaer. Post 1 er navnet på den streng, funktionen skal bruges på. Den anden post angiver *hvor* i strengen funktionen skal starte, – her tælles der fra venstre, hvor den første karakter er nr. 1. Den tredje post er også et tal, – det skal angive antallet af karakterer, der skal kopieres. Her startes der fra den karakter, der blev angivet i post 2, – og der regnes fra venstre mod højre.

Her kan der vist være god brug for et illustrerende eksempel, så prøv programmet i fig. 5.9. I linie 20 bliver a$ tildelt Amstrad CPC464, og linie 30 finder L – antallet af karakterer i a$. Løkken starter i linie 40 og udskriver bogstaver fra strengen. Da n er lig 1 bliver det første bogstav A, fordi det står på plads nr. 1 i strengen, og vi kopierer et bogstav ad gangen. Hvis vi havde haft MID (a$, 1,2), ville vi have fået Am, og med MID$(a$,3,2) havde vi fået st. Men her uddrages der et bogstav ad gangen, og derefter laves der et mellemrum. Semikolonnet i linie 50 sikrer at den næste udskrift kommer i samme linie. Resultatet bliver alt i alt at bogstaverne bliver skrevet ud med mel-

```
10 CLS
20 A$="Amstrad CPC464"
30 L=LEN(a$)
40 FOR n=1 TO L
50 PRINT MID$(a$,n,1);" ";:NEXT
60 PRINT:PRINT
70 FOR n=1 TO l
80 PRINT MID$(a$,n,1)+"+";:NEXT
```

Fig. 5.9 MID$ kan bruges til at uddrage de dele af en streng, der ønskes, og MID$ kan styres af variable, ligesom LEFT$ og RIGHT$

lemrum. Den anden løkke i linie 70 og 80 giver samme effekt, men her indsættes et + i stedet for et mellemrum.

Som før nævnt kan vi bruge variabler eller udtryk i stedet for tal i disse funktioner. Fig. 5.10 viser et lidt mere avanceret program, hvor der bruges et udtryk sammen med MID$. Det starter meget normalt med indlæsning af dit navn i linie 20. Din indlæsning tildeles variablen nm$, og i linie 30 laves der lidt matematisk hokus-pokus. For at forklare hvad der sker kan vi tage et eksempel. Lad os sige at du har skrevet JESPER. Her er der seks bogstaver, så linie 30 bliver L 6, og c bliver den hele del af L delt med 2 (dvs. 3), plus 1 – i alt 4. Linie 40 lader løkken køre 4 gange. I første gennemløb kommer udskriften ved TAB(20) (fordi n = 1 og 21−n er 20), MID$ af de seks bogstaver er c−n+1, dvs. 4−1+1 = 4, og n*2−1 er også 1. Det der udskrives er derfor MID$ (nm$,4,1), som er et P i vores eksempel. I næste gennemløb af løkken er n 2, c−n+1 er 3 og n*2−1 er også 3. Vi får altså MID$(nm$,3,3), som er SPE. Sådan fortsætter løkken og på skærmen får du dit navn skrevet ud i pyramideform. Det ser ret flot ud, især med et langt navn! Hvis du har et kort navn, så prøv at skrive et længere!

```
10 CLS
20 INPUT"Navn ";nm$
30 L=LEN(nm$):c=INT(L/2)+1
40 FOR n=1 TO c
50 PRINT TAB(21-n)MID$(nm$,c-n+1,n*2-1)
60 NEXT
```

Fig. 5.10 Her laves en bogstav-pyramide ved hjælp af MID$ og et udtryk.

Flere karakterer

Vi skal nu til at se på nogle andre streng-funktioner. I begyndelsen af dette kapitel så vi lidt på ASCII. Det er den talkode, der er knyttet til hver af de

karakterer, som vi kan få frem på skærmen. Vi kan finde koden for erhvert tegn ved hjælp af ASC-funktionen. ASC efterfølges af en parentes med enten en karakter i gåseøjne, eller med navnet på en streng-variabel (uden gåseøjne). ASC giver så ASCII-koden for den pågældende karakter. Hvis du fx skriver ASC("CPC464"), får du kun koden for C, fordi ASC-funktionen automatisk ser bort fra de efterfølgende tegn. Fig. 5.11 viser funktionen i et lille program. Strengvariablen a$ tildeles i linie 10 og i linie 30 starter en løkke, der kører alle karaktererne i a$ igennem. Bogstaverne tages et for et med MID$(a$,n,1), og ASCII-koden findes med ASC. Læg mærke til hvordan parenteserne er brugt! Mellemrummet mellem gåseøjnene betyder sammen med semikolonnerne i linie 40, at alle koderne udskrives med mellemrum mellem tallene og uden at der skiftes linie hver gang. Ret enkelt, ikke!

```
10 a$="CPC464 computeren"
20 CLS:PRINT
30 FOR n=1 TO LEN(a$)
40 PRINT ASC(MID$(a$,n,1));" ";
50 NEXT
```

Fig. 5.11 ASC bruges til at finde ASCII-koden for en række karakterer.

Den modsatte funktion af ASC hedder CHR$. Det der kommer efter CHR$ – i parentes – skal være en ASCII-kode, og funktionen finder så den tilsvarende karakter, (det var CHR$ vi brugte i fig. 5.1a). PRINT CHR$(65) vil således fx udskrive et A, fordi 65 er ASCII-koden for A. Vi kan bruge koden til at skjule svar eller lignende i programmer. Nogle gange er det meget praktisk at skjule en meddelelse i et program, så det ikke er alle og enhver, der læser programlistningen, der kan forstå den. Men ASCII-koden er selvfølgelig ikke svær at gennemskue for folk, der selv programmerer, men for begyndere er den god nok. Fig. 5.12 er et eksempel på sådan en skjult kode. I linie 40 er der en WHILE ... INKEY$ ... WEND-løkke, som bevirker at programmet venter på dig. Når du trykker på en tast starter løkken i linie 50 med at skrive seks tegn på skærmen. Hvert af disse indlæses fra en liste ved hjælp af READ ... DATA-ordren i løkken. PRINT CHR$(d%) i linie 60 omsætter ASCII-koden til karakterer og skriver dem på skærmen, semikolonnet i slutningen af linien bevirker, at det hele bliver på samme linie. Prøv selv! Hvis du gerne vil sløre budskabet lidt bedre, kan du prøve fx at gange alle koderne med 5, eller trække 20 fra hver af dem eller lignende. Disse ændrede koder kan du skrive i DATA linien, og så omsætte dem tilbage til ASCII-koder i programmet. Det skulle være nok til at forvirre de fleste! I dette eksempel bruger vi READ ... DATA i en løkke – og det vil du se mange

```
10 CLS:PRINT
20 PRINT"Hvad er koden for succesfuld":P
RINT"programmering?"
30 PRINT"Tryk på en tast for at få svare
t!"
40 WHILE INKEY$="":WEND
50 FOR j%=1 TO 6
60 READ d%:PRINT CHR$(d%);
70 NEXT j%
100 DATA 67,80,67,52,54,52
```

Fig. 5.12 ASCII-koden bruges til at skjule en meddelelse i programmet, og CHR$ bruges til at tyde den.

gange i det følgende. Læg også mærke til at vi bruger heltal variabler, som j% og d%. De fylder mindre i hukommelsen og programmet kører hurtigere.

Der er også en anden ordre, der hører sammen med READ ... DATA. Det er RESTORE (»genoprette«). Når RESTORE bruges alene »nulstiller« den *DATA-pointeren* (»DATA-pegepinden«). DATA-pointeren virker som en slags »bogmærke« for computeren, idet den viser hvor langt den er kommet i DATA indlæsningen. Hvis du fx lige har fået indlæst seks poster fra en DATA-linie, som kun indeholder de seks, så kan du ikke bruge READ igen fordi der ikke er mere at indlæse. Men hvis du bruger RESTORE før du på ny bruger READ, så bliver den oprindelige situation genoprettet – DATA-

```
10 CLS
20 PRINT"Hvilken liste vil du have?"
30 INPUT"Vælg mellem 1,2 og 3";n
40 IF n<1 OR n>3 THEN PRINT"Fejltastning
, - prøv igen!":GOTO 30
50 IF n=1 THEN RESTORE 140
60 IF n=2 THEN RESTORE 150
70 IF n=3 THEN RESTORE 160
80 a$=""
90 WHILE q$<>"x"
100 PRINT q$;" ";
110 READ q$
120 WEND
130 END
140 DATA Opel, Mercedes, Porsche,x
150 DATA Renault, Peugeot, Citroen,x
160 DATA Fiat, Alfa-Romeo, Lancia,x
```

Fig. 5.13 Her bruges RESTORE til at vælge mellem forskellige DATA-linier.

pointeren flyttes tilbage til DATA-liniens start. RESTORE kan også bruges sammen med et linienummer, – det betyder så at indlæsningen starter forfra i pågældende linie. Du skal selvfølgelig være sikker på at det *er* en DATA linie RESTORE bruges på! Se engang på fig. 5.13. Her er der tre forskellige DATA linier, der kan indlæses fra, og her anvendes RESTORE sammen med linienumre. Når du har valgt et nummer, bruges det til at udføre ordren RESTORE i linie 50, 60 eller 70. Desværre kan RESTORE ikke bruges sammen med et udtryk. Hvis det var muligt kunne vi lave programmet lidt pænere med fx RESTORE 1000*n. Hver DATA-linie indeholder tre poster og ender med et x, og løkken, der indlæser fra DATA-listen, er indrettet sådan at den afbrydes når den kommer til et x. Vi kunne godt have brugt en FOR ... NEXT løkke her, for hver linie indeholder samme antal poster. Men ved brug af denne fremgangsmåde kan vi have det antal poster, vi ønsker, i hver af linierne. RESTORE er altså, brugt på denne måde, en meget enkel metode at vælge mellem forskellige DATA-linier på.

Alfabetisk orden

I fig. 4.10 så vi hvordan tal kunne sammenlignes. Vi kan også sammenligne strenge ved at bruge ASCII-koden som sammenligningsgrundlag. Hvis to bogstaver er ens har de samme ASCII-kode, så det er ikke svært at se hvad lighedstegnet (=) betyder, når vi bruger det på strenge. Hvis to strenge er ens må de indeholde de samme bogstaver i den samme rækkefølge. Det er måske ikke så let at se hvad < og > betyder, men tænk igen på ASCII-koden. ASCII-koden for A er 65 og for B er den 66. Så i den forstand er A »mindre end« B. Hvis vi vil have nogle bogstaver i alfabetisk orden, skal vi blot ordne dem efter stigende ASCII-koder.

Det system kan vi gå et trin videre med, idet vi kan sammenligne hele ord – bogstav for bogstav. Fig. 5.14 viser sammenligning ved hjælp af = og > teg-

```
10 CLS
20 a$="QWERTY"
30 PRINT:INPUT"Skriv et ord med store bo
gstaver ";b$
40 IF a$=b$ THEN PRINT"MAGEN TIL MIT!":E
ND
50 IF a$>b$ THEN q$=a$:a$=b$:b$=q$
60 PRINT"Den rigtige rækkefølge er "a$;"
 og derefter ";b$
70 END
```

Fig. 5.14 Sammenligning af ord for at stille dem i alfabetisk orden.

nene. Linie 20 tildeler a$ et tilfældigt ord – her er det simpelt hen de øverste seks bogstaver på tastaturet. I linie 30 skal du indlæse et ord, og sammenligningen foretages i linie 40 og 50. Hvis det ord du har skrevet, og som tildeles b$, er *magen til* QWERTY, udskrives beskeden i linie 40, og programmet er slut. Hvis QWERTY skal stå efter dit ord i en alfabetisk liste, bliver linie 50 udført. Hvis du fx har skrevet PEGEPIND, så kommer Q efter P i alfabetet, og Q har derfor en større ASCII-kode. Dvs at dit ord, b$ er »mindre« end a$, og linie 50 bytter dem om. Det sker ved at tildele en ny streng q$ til a$ (så q$ = "QWERTY"), og derefter tildeles a$ til b$ (så a$ = "PEGEPIND"). Linie 60 vil så udskrive ordene i rækkefølgen a$ – b$, som netop er den rette alfabetiske orden. Hvis det ord du har skrevet kommer efter QWERTY – fx SLUT – så er a$ ikke »større end« b$, og testen i linie 50 er derfor negativ. Så sker der ingen ombytning, og rækkefølgen a$ – b$ er stadig rigtig. Bemærk at programmet ikke kun ser på det første bogstav, – QWERTX vil fx kommer før QWERTY og QWERTZ vil komme efter.

Vi laver tabeller

De variabelnavne vi har brugt indtil videre, har kunnet bruges til mange formål, men de har deres begrænsninger. Hvis du fx har et program, hvor du skal udskrive et stort antal tal – hvordan vil du så tildele dem hver sit variabelnavn? Fig. 5.15 viser hvordan det kan gøres. Linie 10 til 40 laver en tilfældig række point. Vi lader programmet lave disse point, fordi det er lettere end selv at skulle indlæse dem. Variablen i linie 30 er en »nyhed«. Den kaldes en *indiceret variabel,* og »indexet« (flertal af index = indices) er det tal, der her kaldes n. Det navn vi bruger her har ikke noget at gøre med programmering, det har været brugt meget længere. Du laver sikkert af og til en tabel, hvor de enkelte poster er nummererede 1, 2, 3 osv. Her bruges index-numrene på samme måde, for at vi kan skelne de enkelte poster fra hinanden. Så

```
10 CLS
20 FOR n=1 TO 10
30 a(n)=1+INT(RND(1)*100)
40 NEXT
50 PRINT
60 PRINT TAB(9)"POINT LISTE"
70 PRINT:FOR n=1 TO 10
80 PRINT TAB(2)"Nummer ";n;" fik"a(n);"
point."
90 NEXT
```

Fig. 5.15 En tabel med indicerede tal-variabler. Det er ikke så indviklet, som det lyder.

ved at bruge variabelnavnene a(1), a(2), a(3) osv. kan vi skelne de enkelte poster, med det fælles variabelnavn a, fra hinanden.

Det, der gør denne metode så anvendelig er, at vi kan bruge et enkelt variabelnavn for en hel tabel, og blot give de enkelte poster et index-nummer. (Tabeller, skemaer eller lister, der laves med indicerede variabler kaldes ofte for *arrays.)*. Fig. 5.15 viser en tabel, der bliver lavet af FOR ... NEXT-løkken i linie 20 til 40. Hver post bliver lavet ud fra et tilfældigt tal mellem 1 og 100, og bliver derefter tildelt a(n). Der bliver lavet ti point på denne måde, og linie 60 til 90 udskriver point-listen. Skulle vi lave et program, hvor hver post havde sin specielle variabel, ville det have været en lidt større opgave.

Nå, det er jo godt nok, – men der er ting vi ikke har nævnt endnu. Prøv at ændre løkken til FOR n = 1 to 11, og kør programmet! Nu kommer der pludselig en fejlmelding: »Subscript out of range in 30« (Index for stort i linie 30). Computeren er parat til at arbejde med index-numre op til 10 – og ikke større. Hvis vi skal bruge større index-numre, skal vi først give computeren besked om det, så den kan reservere plads til formålet i sit lager. Pladsreservationen sker ved hjælp af DIM (af ordet dimensionere). En linie som DIM a(11) vil faktisk give plads til op til tolv poster i vores array, fordi vi også kan bruge a(0), men vi kan ikke bruge a(12) eller højere tal. Hvis du prøver, får du igen en fejlmelding.

DIM er en meget vigtig ordre, der bruges ved at skrive navnet på den variabel, du vil bruge til tabellen. Det efterfølges af det *maximale* antal poster i tabellen, i parentes, som du vil få brug for. Du behøver ikke bruge dem alle sammen, men du må ikke bruge flere. Hvis du overskrider dimensione-

```
10 CLS:DIM a(12),n$(12)
20 PRINT TAB(2)"Indtast navne og point."
30 FOR n=1 TO 12
40 INPUT"Navn ";n$(n)
50 INPUT"Point ";a(n)
60 NEXT
70 CLS:Sum=0
80 PRINT TAB(14)"POINT LISTE":PRINT
90 FOR n=1 TO 12
100 PRINT TAB(2);n$(n);TAB(22);a(n)
110 Sum=Sum+a(n)
120 NEXT
130 PRINT
140 PRINT"Gennemsnittet er ";Sum/12
```

Fig. 5.16 Her bruges strenge i det ene *array* og tal i det andet. De to arrays dimensioneres med DIM.

ringen, og programmet stopper med en fejlmelding, så må du ændre din DIM-ordre og starte forfra – og det er jo ret ærgerligt, hvis du fx har siddet og skrevet en tabel med 100 navne! Bemærk at du godt kan dimensionere mere end én variabel i en DIM linie, som fig. 5.16 viser. Selvom du ikke behøver at bruge DIM, når du har under 10 numre i et array, så er det en god vane at gøre det alligevel. Med DIM sparer du nemlig på pladsen i lageret.

Fig. 5.16 udvider brugen af indicerede variabler. Her skal du skrive navn og point for hver af de tolv poster, der er lavet plads til. Når indlæsningen er slut slettes skærmen, og variablen Sum sættes til nul i linie 70. Pointtabellen bliver skrevet ud, og pointene lægges sammen efterhånden som løkken gentages (linie 110), således at gennemsnittet kan udregnes i linie 130. Vi behøver altså ikke at nøjes med at have tal i et array. I fig. 5.16 har vi både et streng-array (navne) og et tal-array (point).

Rækker og kolonner

Et array er en liste hvor posterne står efter hinanden, men der er en anden slags array, som giver mulighed for at lave skemaer – den slags array kalder vi for en *matrix*. En matrix er en liste, hvor posterne står i forskellige grupper. Vi kan fx tænke os en matrix, som består af rækker og kolonner, – hver gruppe står i samme række og med de forskellige poster i gruppen i hver sin kolonne. Prøv programmet i fig. 5.17, hvor der opbygges en matrix. Vi bruger variablen n$ som har to index-numre. Det første nummer er rækken og det andet er kolonnen. Vi skal bruge to FOR ... NEXT løkker for at kunne indlæse data i denne matrix. Det sker i linie 20 til 50. Posterne bliver så skrevet ud i kolonner af løkken i linie 60 til 80. I denne løkke bruges variablen n som rækkenummer og kolonnenumrene er 1 og 2. Rækkerne indeholder dyrenavne, og kolonnerne indeholder navnene på henholdsvis voksne dyr og deres unger. I dette eksempel har vi brugt en *streng-matrix*, men vi kan også lave tal-matricer. Hvis du er nogenlunde fortrolig med matematik, ved du sikkert, at matricer kan bruges til mange forskellige formål.

```
10 CLS:DIM n$(3,2)
20 FOR n=1 TO 3
30 FOR j=1 TO 2
40 READ n$(n,j)
50 NEXT j:NEXT n
60 FOR n=1 TO 3
70 PRINT TAB(5);n$(n,1);TAB(20);n$(n,2)
80 NEXT
100 DATA Ko,Kalv,Hund,Hvalp,Kat,Killing
```

Fig. 5.17 En matrix med rækker og kolonner

```
10 CLS:DIM A$(50,2)
20 FOR n=1 TO 50
30 LOCATE 2,5:INPUT "Navn ";a$(n,1)
40 LOCATE 2,7:INPUT "Telf.nr. ";a$(n,2)
50 CLS:NEXT
60 CLS:PRINT"Det var 50 numre - indlæsni
ngen er slut!"
70 LOCATE 10,4:INPUT"Vælg et forbogstav
";j$
80 FOR n=1 TO 50
90 IF j$=LEFT$(a$(n,1),1)THEN PRINT"Navn
 ";a$(n,1);TAB(20)"Nummer ";a$(n,2)
100 NEXT
```

Fig. 5.18 En matrix med navne og numre til en lille telefonliste.

Fig. 5.18 viser et noget mere udbygget matrix program. Ideen i programmet er at lagre navne og telefonnumre, som du indlæser ved hjælp af løkken i linie 20 til 50. Når matrixen er fyldt, kan du indtaste et begyndelsesbogstav, og få computeren til at udskrive det pågældende navn og telefonnummer. Jeg har udeladt fælder i programmet, for at det ikke skal blive for langt. Men skal du bruge listen får du brug for nogle fælder, i det mindste en som denne her:

PRINT" Desværre er der ikke nogen navne med";j$;" som begyndelsesbogstav."

Hvis du laver sådan en liste, vil du sikkert gerne bruge den mere end en gang. Derfor må der indsættes en løkke, der giver mulighed for at vende tilbage til linie 70 for at finde et nyt nummer. Måske kan du også finde ud af, hvad programmet skal gøre, når navnet ikke kan findes. For i det tilfælde vil programmet jo springe forbi IF ... i linie 90. Vi vender tilbage til den slags problemer senere.

En ret nyttig og ret usædvanlig streng-funktion er INSTR. Den bruges til at finde ud af om en streng er indeholdt i en anden. Den kan bruges således:

x=INSTR(a$,b$)

for at undersøge om b$ er indeholdt i a$. Hvis det er tilfældet, er x positionen af det første bogstav i b$, der bliver fundet i a$. Hvis b$ ikke er en del af a$ bliver x nul. Hvis b$ er længere end a$ er x også nul. Hvis du vil se positionen kan du skrive:

PRINT INSTR(a$,b$)

```
10 CLS
20 a$="Jens Hansen"
30 b$="Jens Johansen"
40 c$="Jens Jensen"
50 PRINT"I a$ står Hans på plads nr.";IN
STR(a$,"Hans")
60 PRINT"I b$ står hans på plads nr.";IN
STR(b$,"hans")
70 PRINT"I c$ står HANS på plads nr.";IN
STR(c$,"HANS")
```

Fig. 5.19 Sådan kan INSTR bruges.

Fig. 5.19 viser et lille eksempel på hvordan denne ordre virker. Linie 20 til 40 tildeler en strengvariabel til hvert af navnene, og i linie 50 til 70 laves tre tests, så du kan se hvordan det virker. Bemærk at strengene skal være helt ens for at fungere – det duer ikke at lede efter »hans« hvis der står »Hansen«. Til sidst kan du lige se lidt på denne her: a$ = JAjaJEPSjepsSIKKERTsikkertOKok". Hvis du skulle have et ja/nej svar, kunne du få INSTR til at se a$ igennem. Hvis resultatet af x = INSTR(a$,svar$) er nul, så har der ikke været svaret »Ja« på nogen måde!

Kapitel 6

Menuer, subrutiner og programmer

I programmet i fig. 4.17 så vi at man kunne vælge ved blot at trykke på en tast. I eksemplet var valget begrænset til J og N. Nu er et valg mellem to muligheder jo ikke særlig imponerende, og vi kan da også udvide vores valgmuligheder, ved hjælp af det der hedder en *menu*. En menu er en liste af valgmuligheder, der normalt kan sætte forskellige programafsnit i funktion. Ved at vælge en af mulighederne kan du altså sætte programmet til at køre bestemte program-dele. En af måderne at vælge på er at nummerere de forskellige poster på menuen, og så indtaste det nummer man ønsker. Fig. 6.1 viser en typisk menu af den slags, som den vil se ud på skærmen.

```
                    MENU
1. Opret en ny navneliste.
2. Tilføj nye navne til listen.
3. Find navne i listen.
4. Slet navne fra listen.
5. Vælg et navn.
6. Afslut programmet.
          VÆLG VED AT TASTE ET NUMMER
```

Fig. 6.1 En typisk menu som den vil se ud på skærmen.

Med en ældre computer med »stenalder BASIC« skulle vi bruge linier som:

```
IF K=1 THEN 1000
IF K=2 THEN 2000
```

osv. Her kan vi bruge en noget enklere metode, fordi Amstrad'en har en ny ordre, ON ... GOTO eller ON ... GOSUB. Disse udgør to måder at få udført den samme opgave på, og vi skal se nøje på dem begge, for ordren findes ikke i ret mange BASIC-versioner.

Lad os sige at vi ønsker at vælge mellem fire poster ved at indtaste et nummer mellem 1 og 4. Først må vi sikre os at det kun er tallene 1 til 4, der

accepteres af programmet, ikke tal som 0, 5, −10 eller lignende – dvs vi skal først lave en fælde. En anden ting er at det kan være lidt tungt at bruge INPUT, fordi vi skal bruge ENTER hver gang. Vi bruger derfor INKEY$, til at vælge med (med tal-tasterne), og derefter laver vi fælden, der kan checke vores valg. Endelig bruger vi det tal, der bliver indtastet, til at finde et linienummer og køre den ønskede del af programmet. I fig. 6.2 udskriver linie 10 til 60 en overskrift og menuens poster, og vi bruger TAB for at få det til at stå pænt. Derefter er der en lille brugervejledning (det er måske indlysende for dig, men ikke for alle andre brugere!). I linie 70 og 80 starter selve programmet. I linie 70 er der en INKEY$-løkke, som kører så længe der ikke sker nogen indtastning. Men *når* en tast bliver trykket ned, vil k$ få en værdi på streng-form, og ELSE ... laver indholdet om til et tal. Linie 80 tester tallet, for at sikre at det ligger mellem 1 og 4. Hvis tallet ligger uden for dette område, får du besked om hvilke tal du kan bruge, og så kan du prøve pånyt. Hvis det skulle ske at et tal bliver indtastet, så omsætter k = VAL(k$) trinnet bogstavet til 0, og linie 80 træder derefter i funktion. Så husk aldrig at bruge et nul som en valgmulighed i en menu, for det gør det langt vanskeligere at skelne mellem tal og bogstaver!

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT TAB(2)"1. Aktier."
30 PRINT TAB(2)"2. Obligationer."
40 PRINT TAB(2)"3. Valuta."
50 PRINT TAB(2)"4. Oversigt."
60 PRINT:PRINT TAB(4)"Vælg et nummer 1-4
."
70 k$=INKEY$:IF k$="" THEN 70 ELSE k=VAL
(k$)
80 IF k>4 OR k<1 THEN PRINT"Fejltastning
 - kun tal mellem 1 og 4";" - prøv igen!
":GOTO 70
90 ON k GOTO 110,130,150,170
100 END
110 PRINT"Dette er rutine nr. 1."
120 GOTO 100
130 PRINT"Dette er rutine nr. 2."
140 GOTO 100
150 PRINT"Dette er rutine nr. 3."
160 GOTO 100
170 PRINT"Dette er rutine nr. 4."
```

Fig. 6.2 En menu hvor vi vælger ved hjælp af ON k GOTO.

I linie 90 ligger k indenfor det rigtige område, og valget sker med ordren ON k GOTO 110, 130, 150, 170. Rækkefølgen her *skal* være den samme som i menuen, – dvs. at rutinen der starter i linie 110 skal svare til nr. 1 i menuen, linie 130 skal være til menuens nr. 2 osv. Linierne behøver naturligvis ikke hedde 110, 130, 150, 170 – de kunne ligeså godt hedde 7000, 600, 150, 2010 – *forudsat at disse var de rette startlinier* for rutinerne. I vores lille eksempel er rutinerne blot en udskrift efterfulgt af GOTO 100. Dette sikrer at programmet slutter efter hver rutine. Havde vi i stedet valgt GOTO 10, havde vi fået menuen serveret igen. Og det er ofte det der er brug for. Men når et program altid vender tilbage til menuen, er det en god ting at lade en af valgmulighederne være »Afslut programmet«, så man kan stoppe uden at skulle afbryde maskinen.

Programmering i sektioner

Mange programmer består af en overskrift, en brugervejledning og en menu. Afhængigt af hvilket punkt du vælger på menuen, vil visse dele af programmet køre og derefter afsluttes programmet – eller osse vender det tilbage til menuen. ON k GOTO er en god måde at lave en menu på, men endnu bedre er det at bruge *subrutiner* (»under-rutiner«). En subrutine er en sektion (en del) af programmet, der kan indsættes, hvor du ønsker det, i større programmer. En subrutine indsættes ved at skrive GOSUB (»gå under«) efterfulgt af det linienummer, hvor subrutinen starter. Når programmet kommer til denne ordre, vil det springe hen til det linienummer, der står efter GOSUB – ligesom ved GOTO. Men til forskel fra GOTO har GOSUB *automatisk retur.* Ordet RETURN (»vend tilbage«) bruges til at afslutte en subrutine, og vil få programmet til at vende tilbage til hvor det kom fra – lige *efter* GOSUB. Fig. 6.3 illustrerer dette. Når programmet kører vil linie 20 udskrive en lille sætning, med et semikolon i slutningen af linien for at forhindre linieskift. GOSUB 1000 i linie 30 får ordet »Gul« udskrevet, og RETURN i linie 1010 sen-

```
10 CLS
20 PRINT"Dette er en ";
30 GOSUB 1000
40 PRINT" subrutine":PRINT:PRINT
50 PRINT"Rødt lys og Grønt lys giver ";:
GOSUB 1000:PRINT"t lys."
60 PRINT:PRINT:END
1000 PRINT"Gul";
1010 RETURN
```

Fig. 6.3 Brug af en subrutine – vejen til mere avanceret programmering.

der programmet tilbage til lige efter GOSUB 1000 – dvs til linie 40. Det samme sker også når GOSUB står i en linie med flere forskellige ordrer, som linie 50 viser. GOSUB 1000 bevirker endnu en gang at »Gul« udskrives, men RETURN får programmet til at vende tilbage til samme linie – til PRINT ordren lige efter GOSUB 1000 – ikke til linie 60.

Programmet i fig. 6.4 viser nogle subrutiner i funktion i en del af et større program (som vi må tænke os til). Linie 10 til 80 viser valgmulighederne, og beder os om at vælge. Derefter følger INKEY$ og en fælde – som vi jo efterhånden kender og i linie 120 udføres valget. Denne gang vender programmet tilbage til det, der kommer lige efter det, du har valgt. Dvs at hvis du valgte subrutine 1000 checker programmet om du ønsker at køre subrutine 2000, 3000, 4000 eller 5000. Men da værdien af k stadig er 1 går programmet videre til linie 130, hvor du får mulighed for at få menuen frem igen eller at afbryde programmet. Hvis linie 1000 havde ændret k's værdi til 2, 3, 4 eller 5 ville programmet straks have valgt den respektive subrutine. Hvis der er mulighed for, at den slags kan ske (det er vist ikke smart at bruge k i en subrutine, vel!), så skal subrutinen afsluttes med k = 0 eller lignende, sådan at programmet ikke »løber løbsk«.

```
10 CLS:PRINT
20 PRINT TAB(10)"Vælg en hovedperson."
30 PRINT
40 PRINT TAB(2)"1. Nuser."
50 PRINT TAB(2)"2. Lucky Luke."
60 PRINT TAB(2)"3. Superman."
70 PRINT TAB(2)"4. Tarzan."
80 PRINT TAB(2)"5. Garfield."
90 PRINT:PRINT"Vælg et nummer.":PRINT:PR
INT
100 GOSUB 10000: REM inkey$ rutine.
110 IF k<1 OR k>5 THEN PRINT"Fejltastnin
g - kun tal mellem 1 og 5.":PRINT"Prøv i
gen.":GOTO 100
120 ON k GOSUB 1000,2000,3000,4000,5000
130 PRINT:PRINT"Nyt forsøg? tast J eller
 N."
140 GOSUB 10000:IF k$="J" THEN 10
150 END
1000 PRINT"Den Røde Baron var igen på sp
il...":RETURN
2000 PRINT"Jolly Jumper stod udenfor bar
en i...":RETURN
```

```
3000 PRINT"Clark Kents venner troede at.
..":RETURN
4000 PRINT"Abernes konge var ensom, Jane
...":RETURN
5000 PRINT"Lasagnen stod i ovnen og...":
RETURN
10000 k$=INKEY$:IF k$=""THEN 10000 ELSE
k=VAL(k$)
10010 RETURN
```

Fig. 6.4 Menuens poster vælges ved brug af subrutiner.

En subrutine er meget god at bruge i menuer, men den er nærmest uundværlig, når vi har små programafsnit, der skal bruges flere gange i løbet af et program. I fig. 6.4 har vi et eksempel på det. Her er INKEY$-rutinen lavet som en subrutine, fordi vi skal bruge den flere steder i vores hovedprogram. Når INKEY$ står som en subrutine, kan du nøjes med at skrive disse programlinier én gang. Hver gang du skal bruge INKEY$ kan du blot skrive GOSUB 1000, og programmet vil finde rutinen frem når den skal bruge den. Bemærk at subrutinerne er placeret i linier, der normalt ikke kan køres (med RUN). Selvom du fjernede GOSUB-ordrerne kunne det stadig ikke køre, for der er indsat END- eller STOP-linier, inden subrutinerne kommer. Det er vigtigt, for hvis programmet tilfældigt kommer ind i en subrutine (uden at GOSUB har været brugt), så vil programmet stoppe med en fejlmelding »Unexpected RETURN in line 1010« (Uventet RETURN i linie 1010), (eller hvor RETURN nu står). Når man kommer ind i en subrutine ved et tilfælde kaldes det et »crash« eller et »sammenbrud«, og du kan undgå det ved at bruge END i slutningen af programmet – før subrutinerne begynder.

Fig. 6.5 viser en udbygning af INKEY$ subrutinen. Problemet ved at bruge INKEY$ er at du ikke får nogen melding om at den er i funktion – der er ikke noget spørgsmålstegn, som når du bruger INPUT. Subrutinen i linie 1000 til 1040 får en stjerne til at blinke, mens maskinen venter på svar. Vi får stjernen til at blinke ved først at udskrive en stjerne og derefter slette den igen. CHR$(8) får markøren til at gå et trin tilbage og CHR$(16) sletter karakteren, der står på markørens plads. For at få stjernen til at blinke tilpas langsomt, har jeg indføjet endnu en subrutine, en forsinkelse i linie 2000. Prøv at bruge disse subrutiner i dine programmer og se hvilken forskel, det gør. Prøv i det hele taget at bruge subrutinerne i forskellige små programmer, så du bliver fortrolig med dem. De er ikke alene mægtig gode at bruge i programmer, for at gøre dem nemmere at bruge, de er også en helt uundværlig hjælp, når man planlægger programmer – og det er det vi skal i gang med nu.

```
10 CLS
20 PRINT"Vælg 1 eller 2"
30 GOSUB 1000
40 PRINT"Du valgte ";k$
50 END
1000 k$=INKEY$
1010 IF k$<>"" THEN RETURN
1020 PRINT"*";:GOSUB 2000
1030 PRINT CHR$(8);CHR$(16);:GOSUB 2000
1040 GOTO 1000
2000 FOR j=1 TO 200:NEXT:RETURN
```

Fig. 6.5 En blinkende stjerne-subrutine. Stjernen blinker indtil du trykker en tast ned.

Lav dine egne programmer

Du kan få stor fornøjelse ud af din Amstrad ved at bruge den til de programmer, du kan købe på kassette, eller på indstiksmoduler. Det kan være endnu mere fornøjeligt at bruge de programmer, du kan indtaste fra computerbladenes listninger. Hvis du prøver at ændre på disse programmer, vil du opdage at det ofte er endnu sjovere at lave om på dem, så de kører lidt anderledes, – og det kan man samtidig lære meget af. Men det allermest tilfredsstillende – når det drejer sig om computere – er at lave sine egne programmer. Det behøver slet ikke at være små mesterværker. Bare det at du finder ud af hvad du vil have programmet til, får det skrevet ud som et program, får det indlæst og får det til at køre. Det er helt dit eget værk, og derfor vil du holde lidt mere af det. Trods alt er det at købe en computer uden selv at programmere den, det samme som at købe en sportsvogn, og så lade en anden køre den for dig.

Nu ved jeg jo ikke hvilken type programmer, du specielt er interesseret i. Nogle af læserne er måske specielt interesserede i at holde styr på en frimærkesamling, en pladesamling, madopskrifter eller på specifikationer på veteranbiler. Den slags programmer kaldes for *database* programmer. Andre er måske interesserede i spil, farvegrafik, tegninger, lydeffekter eller bevægelig grafik. Den slags programmer skal vi se nærmere på i kapitel 8, 9 og 10. Her skal vi til at se på database programmer, fordi de er lavet på en måde, der kan bruges i alle typer af programmer. Når du først har lært at lave et lille program af denne slags, kan du gå videre og bruge samme metode til at lave grafik- og lydprogrammer. Men du må huske på, at meget af den flotte, hurtige grafik, som du ser i action-spil ikke er skrevet i BASIC. Grunden til det er, at BASIC er for langsomt til at lave hurtig grafik, og til at styre mange bevæ-

gelser på samme tid. De spil du kan købe er næsten alle skrevet i *maskinkode*, en række ordrer i talkoder, der går direkte til den mikroprocessor, der er selve hjertet i computeren. Maskinkoden springer helt BASIC over, og er langt vanskeligere at bruge. Mange maskinkodeprogrammer er faktisk skrevet ved hjælp af andre programmer på store datamater – så de kan bruges på »små« hjemmecomputere. Men hvis du lærer at programmere i BASIC, så vil du senere kunne gå videre til at lære maskinkode. Du skal bare have en masse erfaring med programmering først.

Der er to ting der er vigtige her. For det første gælder det om at starte i det små, – det lærer du mest af. Det skyldes simpelt hen, at du så har større chancer for at lave et program, der virker. Du lærer mere af at lave et enkelt program, som du kan få til at virke efter nogle få ændringer og rettelser, end du lærer af at starte på et stort program, som aldrig rigtigt kommer til at kunne køre. For det andet starter det, at lave et program, med at du slukker computeren og sætter dig hen til skrivebordet! Bedst er det hvis du sidder i et helt andet rum! Grunden til det er at program-design kræver planlægning, og du kan ikke lave en ordentlig planlægning, hvis du har tastaturet lige foran dig. Det er for stor en fristelse – så gå et andet sted hen!

Start på papiret

Vi starter altså med en blok papir. Jeg bruger selv en almindelig A4-blok med huller til ringbind, så jeg kan holde orden på dem. Når jeg bruger sådan et løsblad-system, er det også lettere at smide alle de ark væk, som jeg ikke får brug for – og det er lige så vigtigt. Ja, jeg skrev *alle*! Selv et meget enkelt program, vil ofte fylde mere end en enkelt side, når det skal planlægges. Når du senere går i gang med at lave større programmer, kan du nemt komme til at bruge en 20-30 sider på planlægning og listninger, *før* du sætter dig til tastaturet. For at gøre træningen i at lave programmer lidt mere spændende, vil jeg bruge et program som eksempel, og så vise planlægningen trin for trin. Det bliver ikke noget indviklet program, men det er velegnet til at vise de ting, der er at lære.

Først skal vi starte med at skrive ned hvad vi vil have programmet til at gøre. Det kan godt være, at du ikke mener, at det er nødvendigt, fordi du godt ved, hvad programmet skal kunne, men bare vent og se. Der er et gammelt ordsprog, der siger, at man ikke kan se skoven for bar' træer, og det passer faktisk fint på at programmere. Hvis du ikke skriver ned hvad programmet skal kunne, så er der store chancer for at det heller aldrig kommer til det! Forklaringen er, at når du først kommer til at sidde og arbejde med små detaljer i programmet, så er det utroligt nemt at komme til at glemme,

hvad det egentlig var, du ville. Hvis du har skrevet det ned, så har du hele tiden et mål at arbejde hen imod, og det er lige så vigtigt i programmering, som det er i livet. Og skriv nu ikke kun et par linier. Brug den tid der skal til, og find ud af hvad det er programmet skal kunne. Hvis du ikke ved det, kan du heller ikke programmere det! Hvad der om muligt er endnu vigtigere ved at skrive ned, hvad programmet skal kunne, er, at det giver mulighed for at få programmet *struktureret* ordentligt. Struktureret betyder her, at programmet bliver sat sammen på en logisk og overskuelig måde, så det bliver nemmere at udbygge, ændre eller lave om. Hvis du lærer programmering på denne måde, bliver dine programmer lette at forstå, de kommer til at fungere hurtigere, og de vil være lettere at bygge videre på.

Nu tager vi fat på eksemplet, så se på fig. 6.6. Det viser en oversigt over en lille engelsk-prøve. Det er lavet som et spil og formålet er at få lært, hvad nogle engelske ord betyder. Planen viser, hvad det er jeg vil have spillet til. Det skal tilfældigt finde et engelsk ord frem, skrive det på skærmen og spørge om oversættelsen. Efter at have tænkt lidt mere, har jeg fundet ud af, at svarene skal være rigtigt stavet. Og så skal svarene sløres lidt, så brugeren (søn, datter, bror, far, mor ...) ikke blot kan taste LIST, og på den måde finde frem til de rigtige oversættelser i DATA-linierne. Ved alle spil skal der være en form for pointgivning, så vi giver et point for rigtigt svar. Da det er vigtigt at stave rigtigt, vil vi tillade to forsøg hver gang. Og endelig skal vi holde rede på antallet af forsøg og på antallet af rigtige svar. Det er stort set hvad vi har brug for lige nu, med mindre vi vil lave spillet mere udbygget. Da det er det første forsøg, så er det her mere end nok. Hvordan kommer vi så videre med planlægningen herfra?

Svaret er, at vi skal lave programmet på samme måde som maleren maler et billede, eller arkitekten tegner et hus. Dvs. at vi først skal have de store

---

Mål:

1. Skrive et engelsk ord på skærmen.
2. Spørge hvad oversættelsen er.
3. Svaret skal være stavet rigtigt.
4. Brugeren må ikke uden videre kunne finde svaret i listningen.
5. Der gives 1 point for rigtigt svar.
6. Der skal være to chancer for at svare rigtigt.
7. Antallet af forsøg skal opregnes.
8. Ved pointgivningen skal det opregnes hvor mange forsøg, der har været, og hvor mange rigtige svar der har været.
9. Spørgsmålene skal vælges tilfældigt.

Fig. 6.6 En oversigt over det planlagte program. Den skal du først lave.

i programmet klar, før vi arbejder med detaljerne. De store linier i programmet er de forskellige trin, det skal bygges op i. Vi skal fx have navnet på spillet i en overskrift. Brugeren skal have tid til at læse den, og derefter skal der være en vejledning. Og vi skal uden tvivl også tildele variabelnavne, dimensionere arrays, og lave andre ting af samme skuffe. Så skal vi have selve spillet. Og vi skal have styr på pointgivningen, og vi skal spørge brugeren om spillet skal fortsætte. Ja, det må skrives ned, ellers bliver det helt uoverskueligt! Fig. 6.7 viser hvordan det kunne se ud, når vi er nået så langt.

1. Overskrift og instruktioner.
2. Skrive et engelsk ord på skærmen.
3. Bede om en oversættelse.
4. Bruge INPUT til svaret.
5. Sammenligne svaret med den rigtige oversættelse.
6. Hvis svaret er rigtigt gives der point, og der spørges om spillet skal fortsætte.
7. Hvis svaret er forkert gives et nyt forsøg.
8. Hvis det andet svar også er forkert, vælges et andet ord.
9. Spillet skal slutte når brugeren taster N på spørgsmålet om, om spillet skal fortsætte.

Fig. 6.7 Næste fase i planlægningen.

Hjørnestenene

Nu kan vi endelig gå i gang med at skrive en bid af programmet. I den nuværende fase må du for enhver pris undgå at fylde papiret med en masse BASIC-linier. Enhver bygherre kan fortælle dig, at det er vigtigt først at have et godt fundament. Det er kvaliteten af fundamentet, der bestemmer hvor godt resten kan blive. Det er vigtigt, at du ikke begynder at bygge væggene op, før du har fundamentet klar.

Fig. 6.8 viser hvad du skal sigte efter at nå i denne fase. Der er kun fjorten programlinier her, og det er alt hvad der er brug for nu. Dette er et *fundament* og *ikke* Empire State Building. Det er også en arbejdsplads, det her, så vi har hængt nogle »advarselsskilte« op. Det er dem du finder i linierne med REM. REM (af engelsk: REMinder = påmindelse) betyder at du bliver husket på et eller andet, og linier der begynder med REM tager computeren sig ikke af. Dvs at du kan skrive hvad du vil efter REM, og formålet med det hele er, at du på den måde kan putte små notitser ind i programmet. De bliver ikke skrevet ud på skærmen når du kører programmet, og du vil kun se dem, når du LISTer programmet. I fig. 6.8 har jeg sat REM-bemærkningerne ind på linier der kun er én højere end programmets hovedlinier. På den måde kan jeg nemt fjerne dem senere, når programmet er *helt* færdigt og kører

```
10 CLS:GOSUB 1000
11 REM titel
20 GOSUB 1200
21 REM instruktioner
30 GOSUB 1400
31 REM opstilling
40 GOSUB 2000
41 REM spil
50 GOSUB 3000
51 REM scoring
60 GOSUB 4000
61 REM en gang til?
70 IF INSTR("JAja",k$)<>0 THEN 40
100 END
```

Fig. 6.8 Her er *hjørnestenene* eller *fundamentet* i programmet.

perfekt. REMerne er meget nyttige at have, men de får programmet til at fylde lidt mere og køre lidt langsommere. Jeg har altid mine programmer i to kopier – en hvor REMerne stadig er med og en som jeg bruger som »arbejdskopi«. På den måde har jeg et program, der kører effektivt, når jeg bruger det til hverdag, og en mere overskuelig version, som jeg kan bruge, hvis jeg vil ændre programmet.

Lad os nu vende tilbage til selve programmet. Som du kan se består det af en række GOSUB-ordrer, der henviser til linier, som vi endnu ikke har skrevet. Det er med fuldt overlæg. Husk at vi er i gang med at lave fundamentet. Vi arbejder helt efter planen i fig. 6.7, og det eneste der ikke står i forbindelse med GOSUB er IF i linie 70. Vi skal nemlig have en subrutine, der bruger INKEY$ til at undersøge om der svares j eller J, og linie 70 tager sig af svaret. Hvad er spørgsmålet da? – Jo, det er naturligvis det trin i programmet, der spørger om spillet skal fortsætte.

Ja, du har ret – linie 70 ser lidt mystisk ud! Ved at taste INSTR("JAja",k$) vil vi få 1 hvis J bliver tastet og 3 hvis j tastes. Vi ville også få disse svar, hvis der blev tastet Ja eller ja (men det kan det jo i øvrigt ikke med INKEY$). Hvis k$ hverken er j eller J, så giver INSTR 0, fordi strengen vi undersøgte ikke indeholder JAja. Enkelt og effektivt.

Se nu disse fjorten programlinier grundigt igennem – de er vigtige. Vores brug af subrutiner betyder, at vi nemt kan checke de forskellige programsektioner – og der er ikke så meget der kan gå galt. Vi kan nu beslutte i hvilken orden vi vil skrive de enkelte subrutiner. Normalt er det uhensigtsmæssigt at skrive dem i den rækkefølge, de står. Fx skal du altid skrive titel og instruk-

tioner til sidst, fordi de er ret ligegyldige for dig nu. Og i alle tilfælde – hvis du skriver dem for tidligt, så får du garanteret en eller anden god idé til at forbedre spillet, sådan at du skal til at skrive instruktionerne om igen. Det er derfor en god idé at tilføje denne linie nu:

```
9 GOTO 30
```

den vil få programmet til at springe titel og instruktioner over. Det vil spare en del tid, når du skal afprøve programmet, fordi du ikke behøver at lade dig forsinke af udskriften af disse programdele, hver gang du kører programmet.

Nu kan du endelig sætte dig til tastaturet og indtaste vores »fundament« for resten af programmet. Hvis du bruger GOTO-trinnet for at komme uden om titel og instruktioner, så kan du indsætte PRINT linier ved hvert GOSUB-linienummer. Det var det vi gjorde i fig. 6.2, så du kender fremgangsmåden. På den måde kan du teste hele denne program-del og forvisse dig om, at den fungerer, inden du fortsætter.

Det næste du skal gøre er at udlæse denne programsektion på bånd, og derefter kan du så bygge nye programsektioner til. Når grundprogrammet er lagret på bånd, så kan du indlæse det tilbage i Amstrad'en, tilføje en ny subrutine og afprøve programmet. (Her kan du bruge MERGE – se sidst i kap. 1, hvis du ikke kan huske det). Når det så fungerer, kan du udlæse det på et nyt bånd. Når du igen skal tilføje en ny subrutine, bygger du videre på den seneste version, og sådan fortsætter du. På den måde kan du have de forskellige faser i programmet lagret på bånd – og du kan være sikker på at de virker. Og det er ret vigtigt. Afprøvningen af programmer tager ofte meget længere tid end man tror, og det er en stor lettelse kun at skulle teste én subrutine ad gangen.

Subrutine rutine

Det næste vi skal i gang med er at konstruere subrutinerne. Nogle af dem er der ikke mange ben i. Fx kan vi hurtigt lave subrutinen i linie 4000. Det er den velkendte INKEY$-rutine, sammen med et par PRINT-linier, så den kan vi ligeså godt lave med det samme. Fig. 6.9 viser hvordan den kan se ud. Den er helt enkel – indtast den og prøv den sammen med grund-programmet.

```
4000 PRINT"Vil du have et spil mere?"
4010 PRINT"Svar J eller N."
4020 k$=INKEY$:IF k$="" THEN 4020
4030 RETURN
```

Fig. 6.9 Subrutinen for linie 4000.

Nu kommer vi til det, som du måske tror, er det sværeste – nemlig den subrutine, som skal rumme »spillet«. Men det er faktisk ikke så svært – du skulle allerede have forudsætningerne for at lave det. Spille-subrutinen planlægges ligesom vi planlagde grundprogrammet. Dvs at vi skal have papir og blyant frem igen og skrive ned hvad programmet skal kunne, – og vi skal skrive ned, trin for trin, hvordan det skal gøres. Hvis der er noget der virker for vanskeligt, kan vi sende det ned i en subrutine, som vi laver senere.

Se nu på fig. 6.10, her er et eksempel på hvordan planen for subrutinen kunne se ud. Punkt 1 krævede lidt tænkearbejde. Vi skulle jo finde en måde at skjule svarene på, så en bruger af vores program ikke umiddelbart skulle kunne læse svarene i DATA-linierne. En rimeligt enkel og rimeligt effektiv løsning er at bruge ASCII-koder. Det vil ikke afskrække en BASIC-kyndig, men den er god nok til de fleste. Jeg har besluttet at putte et svar i hver

1. Svarene skjules i et array af ASCII-koder i strenge.
2. Spørgsmålene (de engelske ord) skal stå i en anden streng.
3. Det tal der bruges til at vælge spørgsmålet, bruges også til svaret.
4. Variablen sp bruges til antal spørgsmål, og sc bruges til score.
5. Variablen fs bruges til antallet af forsøg på hvert spørgsmål.

Fig. 6.10 Planen for spille-subrutinen.

DATA-linie – formet som en streng af ASCII-koder, hver af dem skrevet som et tre-cifret tal. Hvorfor tre cifre? Ja, de store bogstaver har to-cifrede ASCII-koder og de små bogstaver har tre, så hvis vi giver alle sammen tre-cifrede, så bliver det lidt nemmere. Du skal senere få at se hvorfor – det vi gør, er at skrive et tal som 86 som 086 osv. Det var det første punkt i denne subrutine.

I punkt 2 skal vi have de engelske ord i et array. Det giver os mange fordele. En af dem er at det er dejlig nemt at vælge tilfældigt, hvis vi gør det på den måde. En anden er at det gør det let at sammenligne spørgsmål og svar. Hvis spørgsmålene er poster i et array med indices 1 til 10, så kan vi placere svarene i DATA-linier (et sæt ASCII-koder i hver linie), og læse dem som streng-arrays også.

De næste punkter i planen er at bestemme de navne vi vil give vores variabler. Her er det altid en god idé, hvis vi kan bruge navne, der minder os om, hvad variablerne står for. Vi har brugt sc for score, sp for antallet af spørgsmål og fs for antallet af forsøg på hvert spørgsmål. Endelig vil vi give det array, der skal indeholde de engelske ord (spørgsmålene) navnet q$.

Spilleprogrammet

Fig. 6.11 viser den programsektion, jeg har lavet på grundlag af planen i fig. 6.10. Trinene er først at vælge et tilfældigt tal, derefter at bruge tallet til at få udskrevet et engelsk ord og endelig at finde svaret. Og det er alt – for undersøgelsen af om svaret er rigtigt og pointregnskabet klares af en anden subrutine. Prøv altid at få splittet programmet op så meget som muligt, så du slipper for at skulle skrive alt for store programsektioner. Her har jeg jo fx lagt en subrutine ind i en anden, for at få det hele til at være overskueligt.

```
2000 fs=0:v=INT(10*RND(1))+1
2010 CLS:PRINT"Ordet er - ";q$(v)
2020 PRINT:PRINT"Det betyder - ";
2030 INPUT x$:sp=sp+1
2040 GOSUB 5000
2041 REM find det rigtige svar
2050 RETURN
```

Fig. 6.11 Programlinierne i spille-subrutinen.

Vi starter subrutinen i linie 2000 med at »slette en variabel«. Værdien af fs sættes til 0, for at vi kan være sikre på at denne variabel har den rigtige værdi, hver gang subrutinen startes. I en anden del af linie 2000 vælges et tilfældigt tal mellem 1 og 10. Linie 1010 til 2030 er meget ligetil. Vi vælger det engelske ord, der svarer til det tilfældige tal, og beder om oversættelsen. Den sidste del af linie 2030 holder regnskab med antallet af spørgsmål. Det er jo et logisk sted at placere dette trin. Og så sniger jeg mig uden om det med at læse ASCII-koderne – det overlader jeg til en subrutine, som skal starte i linie 5000, så vil jeg skrive den senere. REM i linie 2041 minder mig om hvad den nye subrutine skal gøre, og spillesubrutinen slutter med det sædvanlige RETURN.

Ned i de små detaljer

Når du har lagret spillesubrutinen på bånd, kan vi begynde at tænke på detaljerne. Det første vi skal se på er pointregnskabet – som kommer lige efter spilletrinnet. Som sædvanlig skal der laves lidt planlægning, og fig. 6.12 viser planen. Hver gang der er et rigtigt svar, bliver talvariablen sc opskrevet, og vi kan vende tilbage til hovedprogrammet. Hvis svaret ikke er rigtigt, er vi derimod ikke færdige. Så skal vi udskrive en melding om at svaret ikke var rigtigt, og at der gives endnu et forsøg. Hvis dette forsøg heller ikke er rigtigt, så er vi færdige og kan vende tilbage til næste spørgsmål. Senere, når du har

læst kapitel 11, kan det være du får lyst til at tilføje noget lyd til programmet. Vi kunne fx lave et kort bip ved forkert svar og et langt bip ved rigtigt svar. Skriv det ned!

1. Ved rigtigt svar, opskriv sc.
2. Ved forkert svar gives der et forsøg mere, hvis fs = 0, derefter sættes fs = 1.
3. Ved endnu et forkert svar (fs = 1), fortsættes der til næste spørgsmål og fs nulstilles.

Fig. 6.12 Planen for score-subrutinen.

Fig. 6.13 viser den program-subrutine, der er kommet ud af planen. Linie 3000 tager sig af et rigtigt svar. Vi skal have udskrevet en melding her, så vi skaffer lidt plads ved at skrive GOTO 3200. GOTO 3040 i linie 3210 sikrer, at hvis svaret er rigtigt, så springes resten af denne subrutine over, og der vendes tilbage til hovedprogrammet. Men hvis svaret ikke er rigtigt, træder linie 3010 i funktion. Den tester om fs er nul, og hvis det er tilfældet, springes der til linie 3300, hvor der gives yderligere instruktioner. Linie 3320 fører hen til subrutinen i linie 2010, så brugeren kan få et forsøg mere. GOTO 3000 i linie 3320 tester svaret en gang til.

```
3000 PRINT:IF x$=a$ THEN sc=sc+1:GOTO 32
00
3010 IF fs=0 THEN GOTO 3300
3020 fs=0:PRINT"Desværre forkert - prøv
det næste ord."
3030 FOR q=1 TO 1000:NEXT
3040 RETURN
3200 PRINT"Rigtigt - din score er nu";sc
3210 PRINT"ud af"sp;" mulige. ":GOSUB 70
00:GOTO 3040
3300 PRINT"Ikke rigtigt - men det kan væ
re en"
3310 PRINT"stavefejl! Du får et forsøg m
ere.":sp=sp-1
3320 GOSUB 7000:fs=1:GOSUB 2010:GOTO 300
0
```

Fig. 6.13 Programmet for score-subrutinen.

Der er brugt lidt snilde her. Talvariablen fs starter med værdien 0. Når der kommer et forkert svar og fs stadig er 0, så træder linie 3010 i funktion. I linie

3320 sker der bl.a. det at fs sættes til 1. Når du svarer igen – og fs er 1 – bliver linie 3000 brugt og hvis dit andet svar også er forkert, kan linie 3010 ikke bruges, fordi fs ikke er nul. Den næste linie bliver derfor 3020. Den nulstiller fs til næste runde, udskriver en melding om at svaret var forkert og spørger om du vil prøve igen. Så holdes der en lille pause, og i linie 3040 vendes der tilbage til hovedprogrammet.

Nu har vi fået lavet en god del af programmet, og vi kan tage fat på de sidste subrutiner. Fig. 6.14 viser den lille subrutine, der tager sig af dimensionering og arrays. Linie 1400 sætter alle variablerne i pointsystemet til nul. Linie 1410 dimensionerer de to arrays der bruges til de engelske ord – q$ – og til svarene – a$. Linie 1420 læser ordene fra en DATA-liste ind i q$, og linie 1430 indlæser ASCII-koderne ind i a$. Og det er det hele – for DATA-linierne venter vi med – som sædvanlig.

```
1400 sp=0:sc=0:fs=0
1410 DIM q$(10),a$(10)
1420 FOR j=1 TO 10:READ q$(j):NEXT
1430 FOR j=1 TO 10:READ a$(j):NEXT
1440 RETURN
```

Fig. 6.14 Subrutine med dimensionering og arrays.

Nu kommer vi så til at skulle finde svarene. Vi planlagde det jo, så det ikke skulle være alt for besværligt. Fig. 6.15 viser programlinierne.

```
5000 a$="":FOR j=1 TO LEN(a$(v)) STEP 3
5010 a$=a$+CHR$(VAL(MID$(a$(v),j,3))):NE
XT
5020 RETURN
```

Fig. 6.15 Svarene kontrolleres.

Variablen v er det tilfældige tal, og det bruges til at vælge en af strengene med ASCII-koder, a$(v). Da hver kode består af tre cifre, kan vi blot skære tre cifre af ad gangen, og det er derfor vi bruger STEP 3 i FOR ... NEXT-løkken i linie 5000. Linie 5010 laver så svar-strengen a$ til os. Husk at a$ ikke kan forveksles med a$(v) arrayet. I den første del af linie 5000 sørges der for, at a$ er tom, sådan at a$ ikke indeholder det tidligere svar – som jo også skulle hedde a$. Vores a$-streng opbygges så trin for trin, med tre cifre ad gangen, – de ændres til at være tal med VAL og omsættes til bogstaver med CHR$. Bogstaverne bliver et for et lagt til a$, indtil vi har »oversat« alle ASCII-koderne i den streng, vi havde fat i. Nu er det værste overstået, og vi kan begynde at lave instruktioner – fig. 6.17 – og titel – fig. 6.18. Begge er

```
1200 CLS:PRINT TAB(13)"INSTRUKTIONER"
1210 PRINT:PRINT TAB(4)"Computeren vil g
ive dig et engelsk"
1220 PRINT"navneord. Du skal så indtaste
 den rig-"
1230 PRINT"tige oversættelse af ordet. -
 Du skal"
1240 PRINT"være sikker på at du har stav
et rig-"
1250 PRINT"tigt, og at du starter med st
ort begyn-";"delsesbogstav."
1260 PRINT:PRINT"Computeren regner din s
core ud."
1270 PRINT"Du får to forsøg til at finde
 den rig-"
1280 PRINT"tige oversættelse."
1290 PRINT:PRINT"Tryk på Mellemrumstange
nten når du er   klar."
1300 IF INKEY(47)=-1 THEN 1300 ELSE RETU
RN
```

Fig. 6.16 Instruktionerne – vent altid med dem til du er næsten færdig.

```
1000 MODE 0
1010 PRINT TAB(5)"Engelske ord"
1020 GOSUB 7000:MODE 1:RETURN
```

Fig. 6.17 Programlinierne, der giver titlen.

```
6000 DATA Emperor,King,Entrance,Ant,Aunt
,Treaty,Exit,Cotton,Wool,Snake
6001 DATA 075101106115101114
6002 DATA 075111110103101
6003 DATA 073110100103097110103
6004 DATA 077121114101
6005 DATA 084097110116101
6006 DATA 084114097107116097116
6007 DATA 085100103097110103
6008 DATA 066111109117108100
6009 DATA 085108100
6010 DATA 083108097110103101
7000 FOR q=1 TO 3000:NEXT:RETURN
```

Fig. 6.18 DATA-linierne sammen med en lille pause-rutine.

naturligvis subrutiner. Titlen skrives med MODE 0 bogstaver, og derefter er der en lille pause. På den måde får vi store bogstaver – og metoden fungerer fint hvis ikke vores titel er for lang (over 20 karakterer). Endelig har vi DATA-linierne og en pause i den sidste subrutine – fig. 6.18.

Nu kan vi samle hele programmet og prøve det. Husk som nævnt at du kan bruge MERGE, til at samle de enkelte programsektioner med. Da vi har lavet vores program efter modul-princippet, er det en ret enkel sag at gå ind og ændre programmet. Du kan fx bruge andre DATA. Du kan gøre spillet mere afvekslende ved at bruge mange flere spørgsmål – men husk endelig at rette DIM i linie 1410. Du kan lave spillet om, så det handler om noget helt andet – hvad fx med en spørgeleg med de forskellige BASIC-ord! Alt hvad du skal gøre er at ændre instruktionerne, DIM og DATA *). Du kan også give programmet nogle flotte lydeffekter, og du kan tilføje noget spændende grafik. En svaghed ved programmet er at spørgsmål, der har været brugt en gang kan dukke op igen, – sådan fungerer vor RND-funktion. Du kan klare problemet ved at bytte rundt på spørgsmålene, således at når et spørgsmål har været brugt, så byttes det om med det sidste spørgsmål, (med mindre det *var* det sidste spørgsmål), og så samtidig nedskrive antallet af valgmuligheder. Fx: hvis du vælger nr. 5 så byttes nr. 5 rundt med nr. 10 – og vælg derefter mellem de 9 første spørgsmål. Dvs at 10*RND(1)+1 trinnet vil blive til d*RND(1)+1, hvor d starter som 10, og nedskrives en gang for hvert rigtigt svar.

Der er faktisk masser af ting du kan gøre med dette program, for at gøre det meget mere interessant. Det er grunden til, at jeg har brugt det som et eksempel på, hvad du selv kan konstruere dig frem til, med den viden du har nu. Se på det som et »BASIC byggesæt«, som du kan lave om på, som det passer dig. Det vil give dig en fornemmelse af hvor langt du kan nå, når du først rigtigt har lært at håndtere din Amstrad. Efterhånden som du får mere erfaring med programmering, kan du lave programmer, der er meget længere og mere udbyggede end dette. Når du når så langt, vil du sikkert begynde at tænke på at få dig en printer og en diskettestation til din Amstrad. Men også før du kommer dertil, vil du opdage, at det er en mægtig god idé at gemme båndkopier af forskellige subrutiner, som du ved hjælp af MERGE, kan få ind i dine programmer.

Og så lige én ting til. Hvad nu hvis programmet ikke vil køre, som du vil have det til? Hvad kan du gøre for at finde fejl mv.? Disse ting kan du læse noget om i appendix A, som handler om redigering og fejlfinding.

---

*) Husk at det kan give problemer at bruge de danske tegn i DATA-linier, – prøv dig frem. O.A.

Kapitel 7

Brug af datarecorderen

Amstrad har, i modsætning til langt de fleste hjemmecomputere, et indbygget lagringssystem for programmer og data. Hvor de fleste andre computere bruger en almindelig båndoptager som lager, har Amstrad en indbygget båndoptager, der er lavet specielt til formålet – en *datarecorder*. Da brugen af et sådant datalagringssystem formentlig er ny for de fleste Amstrad-ejere – også selv om de har haft en anden computer før – så vil hele dette kapitel omhandle brugen af datarecorderen. I kapitel 1 har vi allerede set på brugen i forbindelse med udlæsning og indlæsning af programmer. De mest gådefulde nye begreber vi skal se på her er *udstyr* (engelsk: devise), *datastrøm* og *mellemlager*. Når du har fået fat i hvad disse begreber dækker og i hvilke sammenhænge de bruges, vil du opleve at brugen af datarecorderen bliver meget mere spændende, og du vil være i stand til at lave meget mere med din Amstrad. Så lad os starte med at forklare disse nye begreber.

Udstyr er noget, der udsender eller modtager data. Dit tastatur er udstyr, for hver gang du rører en tast, udsendes der et elektrisk signal til computeren. Skærmen er også udstyr, idet den jo modtager signaler, som bliver til udskrifter. Tastaturet er udsendelses-udstyr, fordi det udsender signaler. Skærmen er modtagelses-udstyr, fordi den modtager signaler. Amstrad-skærmen er faktisk adskillige stykker udstyr, fordi den kan opdeles, og bruges til flere ting samtidig. I kapitel 8 skal vi se på, hvordan vi kan opdele skærmen.

Noget udstyr kan bruges til begge dele samtidig. En *konsol* er en kombination af tastatur og skærm, som både kan udsende og modtage data. Datarecorderen er også udstyr, der kan bruges i begge retninger. En diskettestation er udstyr af samme type.

Vi har her sagt at elektriske signaler går fra noget udstyr til andet udstyr, – og det er faktisk osse hvad der sker. Men det giver en bedre forståelse at tænke på det, som disse signaler repræsenterer. Hvert sæt signaler svarer til en data-enhed, der kaldes en *byte,* og *data* er det som datamater er konstrueret til at behandle. En byte er den plads i computerens hukommelse, der skal bruges til at rumme en karakter i ASCII-kode, eller en ordre i BASIC. Data

kan enten være tal eller ord, – simpelt hen det som computeren arbejder med. Hvis du har et program, der giver en oversigt over navnene på dine venner, ordnet efter fødselsdage, så skal det program bruge data. Og data er i dette eksempel navne og datoer. Hvis du har et program, der kan bruges til madopskrifter, så er data i det tilfælde anvisninger på hvordan maden laves, navnene på madvarerne og tal for mængderne af de forskellige madvarer. Enhver computer, der kan bruges til andet end nogle simple spil, skal kunne både lagre og indlæse disse typer data – adskilt fra det program, der bruger eller producerer dem.

Ved at adskille data og programmer kan der opnås store fordele. Amstrad'ens lager bruges til mange formål, som du ikke har nogen indflydelse på. Et meget langt program, som opsamler data, er der måske ikke plads til i computeren. Så er det mere hensigtsmæssigt at have et kort program, der opsamler data ved hjælp af INPUT-linier, og som lagrer de data på bånd. På denne måde er de data i sikkerhed, hvis der skulle ske noget med computeren, og samtidig kan man også bruge de opsamlede data sammen med andre programmer. Ved at holde data og program adskilt, kan du behandle meget større datamængder, end det ville være muligt, hvis du skulle have det hele i computerens hukommelse på én gang.

Hvad har det så med mellemlagre, datastrømme og udstyr at gøre? Ja, udstyret er de dele af computeren, der udsender eller modtager data. *Datastrømmene* er de veje, som dataerne må følge. Tænk bare på hvad der sker, når du bruger din Amstrad. Når du rører en tast, kommer der noget frem på skærmen. Tastaturet er udstyr, skærmen er udstyr og de er forbundet af en datastrøm. Det er blot en anden måde at udtrykke det forhold på, at der er en vej, som fører data-signaler fra tastatur til skærm. Men det vigtige er imidlertid at disse datastrømme kan styres. Vi kan styre dem i den forstand at vi kan ændre vejene, afbryde nogle og oprette andre som vi vil. Det er selvfølgelig ikke nogen god idé at afbryde nogle veje, med mindre vi har et bestemt formål med det. Normalt vil vi jo gerne have det, vi skriver, frem på skærmen. Men hvis du fx skal skrive en bestemt adgangskode, som andre, der kan se skærmen, ikke må se, ville det jo være en god ide at afbryde datastrømmen mellem tastatur og skærm. Fig. 7.1 viser hvordan det kan ske. Linie 10 beder om en adgangskode på fire bogstaver. I linie 20 afbryder PRINT CHR$(21) datastrømmen, der forbinder tastatur og skærm. Nu kan du så skrive din adgangskode, via INPUT i linie 30, men der vil ikke komme noget på skærmen! I linie 40 kobles datastrømmen til igen, således at meddelelserne i linie 50, 60 og 70 kan udskrives. Vores indlæsning bliver testet på sædvanlig vis, og du får en ny chance, hvis du ikke har indtastet fire bogstaver. Hvis du vil se det, du har indtastet, så kan du få det frem med PRINT b$.

```
10 PRINT"Indtast din adgangskode (4 bogs
taver)"
20 PRINT CHR$(21)
30 INPUT b$
40 PRINT CHR$(6)
50 IF LEN(b$)<>4 THEN PRINT"Forkert - pr
øv igen":GOTO 10
60 PRINT"Tak - adgangskoden godkendt."
70 PRINT"Skriv PRINT b$ for at se den!"
```

Fig. 7.1 Sådan kan en adgangskode indtastes, uden at komme frem på skærmen.

Du har jo nok gennemskuet, at nogle datastrømme er koblet til i det øjeblik, du tænder for computeren. Det er fx indlysende, at der er en datastrøm mellem tastatur og skærm. Denne datastrøm har normalt nummer 0, og vi kan få en PRINT-ordre til at bruge denne datastrøm, ved at skrive #0 efter PRINT. »Krydset«(0) er det amerikanske tegn for »nummer«, og det bruges ligesom vi ville bruge »Nr.«. Amstrad er opbygget sådan, at der er mulighed for at bruge i alt ti datastrømme. Af disse er tre af dem (0, 8, 9) på forhånd bestemt til at udføre bestemte opgaver, og du skal prøve at undgå at ændre på disse datastrømme. Det forskellige udstyr har hver sit nummer – som bruges med krydset (#) foran.

Som så ofte i EDB, er nummereringen ikke fra 1 til 10 men fra 0 til 9. Af disse datastrømme anvendes 0, 8 og 9 til computerens eget brug. #0 styrer skærmen, #8 styrer printeren og #9 styrer datarecorderen. De sidste syv datastrømme kan du disponere over, og vi skal i dette og næste kapitel se på, hvordan de kan bruges. Dette kapitel handler om at lagre data på bånd, og for at kunne gøre det skal du vide to ting. Det ene er at kunne vælge et mellemlager, og det andet er at kunne forbinde det med datarecorderen.

Når du skal vælge et *mellemlager* og koble det til datarecorderen, skal du bruge en af OPEN-ordrene. Dem er der to af: OPENOUT og OPENIN. OPENOUT (»åben ud«) skal efterfølges af et filnavn, ligesom SAVE. Når det kører, kobler det datarecorderen til computeren, så den er parat til at modtage data. Det er blot en forberedelse – der overføres ikke data når OPENOUT bruges – ordren sørger blot for, at der gøres klar til dataoverførsel. OPENIN (»åben ind«) skal også efterfølges af et filnavn, ligesom LOAD. Da der bruges et filnavn, kan kun en fil med det rigtige navn kunne vælges, og datarecorderen starter med at udsende data, i det øjeblik den finder en fil med det rigtige navn på båndet.

En fil – lagring af data

Hvad er en fil?

Ordet *fil* (engelsk: file = »arkiv«, »kartotek«) bruges ofte i denne bog. En fil er enhver samling af sammenhørende karakterer. Karaktererne i et BASIC-program udgør fx en fil, fordi programmet ikke kan køres, hvis der mangler karakterer. Et sæt navne og adresser i ASCII-kode er en fil, fordi de udgør en samling informationer, om fx venner, leverandører eller debitorer. Et sæt bytes i maskinkode er en fil, og en samling tal, der anvendes i et økonomistyringsprogram, er en fil. I dette kapitel vil jeg dog kun bruge fil i betydningen: en samling informationer, der kan lagres på bånd – adskilt fra programmet. Hvis du fx har et program, der kan lægge et husholdningsbudget, så skal du have en fil med de forskellige poster på budgettet og med de forskellige beløb. Eller lad os tage et andet eksempel, hvis du ville lave et program, der kunne holde styr på din pladesamling. Programmet ville i det tilfælde kræve, at der blev indlæst en masse oplysninger om pladerne. Denne samling af oplysninger er en fil, og på et vist trin i programmet, skal du indlæse denne fil. Hvorfor? Fordi hvis programmet virkelig skal kunne bruges, er der ikke plads nok i lageret til alle oplysningerne på én gang, – selv ikke på en Amstrad. Og dertil kommer at du skulle ændre dit program, hver gang du skulle indlæse nye oplysninger. Det er det kapitlet her handler om – lagring af de informationer, som et program bruger.

Inden vi går videre skal vi lige have et fagudtryk mere præsenteret. Ordet er *felt*, – felter er de enkelte punkter i en *post*, – som igen er de enkelte punkter i en fil. Hvis du fx har en fil om LNER damplokomotiver, kan du have en post for hver lokomotiv-type. Inden for hver post kan du så have navnet på konstruktøren, kedelens rumfang, arbejds-damptryk, trækkraft

FIL OVER VENNER	
POST 1	
FELT 1	Navn 1
FELT 2	Adresse 1
FELT 3	Telefon nr. 1
FELT 4	Fødselsdag 1
POST 2	
FELT 1	Navn 2
FELT 2	Adresse 2
FELT 3	Telefon nr. 2
FELT 4	Fødselsdag 2
POST 3	
osv.	

Fig. 7.2 En fil med *poster* og *felter.*

osv. Hver af disse underpunkter er et felt, og et sæt felter er en post. Din post kan fx være SCOTT 4-4-0 lokomotivet. Hver eneste oplysning om SCOTT lokomotiverne er et felt, og alle felterne om SCOTT er igen en post. Og SCOTT er kun en af posterne i filen, der vil også være andre som fx Gresley Pacifics 4-6-0 osv. Eller lad os tage endnu et eksempel, – en fil med »Engelske motorcykler« I denne fil er BSA en post, AJS er en anden og Norton er en tredje. I hver af disse poster er der felter. Felterne kan være motorstørrelse, slaglængde, antal cylindre, acceleration, tophastighed osv. Hvis du kan lide at sætte ting i system, så kan du få meget fornøjelse ud af at opbygge filer.

Filer på bånd

I denne bog vil vi – da vi arbejder med Amstrads datarecorder system – undlade at behandle de filer, der kan opbygges i DATA-linier i BASIC-programmer. Det skyldes at vi med Amstrad-systemet nemt kan lagre data adskilt fra programmet, og det er en langt bedre metode. Hvis du kender til det

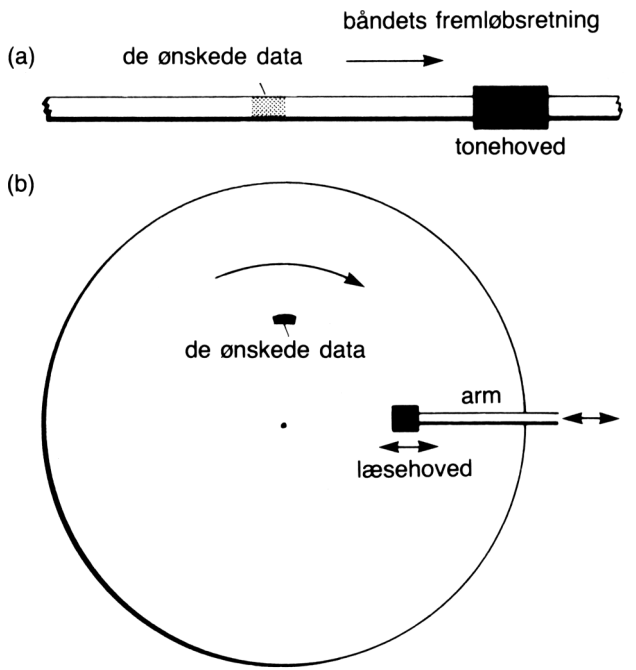

Fig. 7.3 Skematisk fremstilling af forskellen mellem (a) sekventiel tilgang på bånd og (b) direkte tilgang på diskette. Ethvert punkt på disketten kan aflæses af hovedet, ved at placere det med den rigtige radius, mens disketten roterer. For at aflæse et bestemt sted på båndet, må du lade alt båndet foran køre forbi først.

i forvejen, må du have tålmodighed indtil jeg kommer til noget, du ikke kender til endnu.

For det første er der to slags filer, og det er kun den ene vi kan bruge, når vi lagrer på bånd. Den ene er filer med *direkte tilgang*, og den anden er filer med *sekventiel tilgang*. Forskellen er enkel, men vigtig. En sekventiel (eller seriel) fil lagrer alle filerne i rækkefølge på bånd. Hvis du vil have fat i en enkelt post må du indlæse hele filen i computeren, og derefter kan du udvælge den enkelte post. Der findes ingen genveje til kun at indlæse den post (eller det felt) du skal bruge. En fil med direkte tilgang giver mulighed for, som navnet siger, at vælge direkte fra det *eksterne lager* (både bånd og disketter er eksterne lagre), uden først at skulle indlæse hele filen i computeren. Forskellen på sekventiel og direkte tilgang er illustreret i fig. 7.3. Direkte tilgang kræver brug af et diskettesystem, men vi kan lave programmer, hvor vi får noget af den samme effekt, når vi bruger sekventielle filer på bånd. Vi starter med at se på de sekventielle filer, som er den type filer, vi kan få med vores datarecorder.

Opbygning af en fil

Når du starter på noget nyt, er det altid en god ide at begynde med begyndelsen, og gøre tingene så enkle som muligt. Først skal vi se på hvordan vi får forbindelse med datarecorderen. Fig. 7.4 viser et lille eksempel på, hvordan en post, værdien af variablen A, kan blive lagret på datarecorderen, der altid har datastrøm nr. 9. Da trinene i processen er meget vigtige, vil vi gennemgå dem meget nøje. I linie 10 *åbner vi en fil.* Dertil skal vi bruge ordren OPENOUT (»åben ud«), og vi skal give filen et navn. Linie 10 bruger OPENOUT "TEST", for at åbne en ny fil med navnet TEST. Du skal være omhyggelig med kassettebåndet, for du skal bruge et bånd (eller et sted på båndet), hvor der ikke er lagret noget i forvejen. Der er nemlig ikke noget, der ellers kan forhindre, at du kommer til at slette en fil ved et tilfælde – hvad enten den har samme navn eller et andet navn. Med en diskette station kan du gardere dig mod den slags fejltagelser, men når du bruger kassettebånd, har du kun båndtælleren til at vejlede dig.

```
10 OPENOUT "TEST"
20 A=5
30 PRINT#9,A
40 CLOSEOUT
```

Fig. 7.4 Lagring af værdien af en enkelt variabel. Det er værdien der lagres, ikke variabelnavnet.

Næste trin er at tildele en variabel, A, i linie 20. Linie 30 er vigtig. Ordren PRINT#9,A betyder: »send værdien af A over datastrøm 9. Datastrøm 9 er imidlertid altid forbundet med datarecorderen, så linie 30 vil »udskrive« værdien af A på datarecorderen. Du vil få den samme melding at se, som når du bruger SAVE, og værdien vil blive lagret under navnet TEST, så den let kan findes frem igen. Linie 30 får selve optagelsen til at foregå, men det sker ikke helt så enkelt, som det ser ud til. Datarecorderen optager grupper af karakterer, og operativsystemet er indrettet sådan, at computeren vil opsamle data i hukommelsen indtil den har nok. Denne del af hukommelsen hedder et *mellemlager*, og når du åbner en datastrøm, vil datastrømmen automatisk blive sendt til et mellemlager. De data, der skal lagres, bliver først overført til mellemlageret og derefter optaget på båndet. Hvis der er flere data end mellemlageret kan rumme (2000 karakterer), så skal mellemlageret fyldes og tømmes mere end én gang. Efter denne procedure skal du sikre dig at mellemlageret bliver tømt for indhold, og det sker med CLOSEOUT-ordren (»luk ud«) i linie 40.

Hvad sker der så når dette her udføres? Fra dit synspunkt sker der blot det, at datarecorderen afleverer den sædvanlige melding på skærmen, kører forholdsvis længe, og stopper.

Nu skal vi så vise at vores data faktisk er blevet lagret på båndet, og vi skal se hvordan vi kan indlæse dem i computeren igen. Se på programmet i fig. 7.5. Her bruger linie 10 OPENIN og ikke OPENOUT. Vi har allerede en fil på vores bånd, og vi vil gerne indlæse den igen – ikke åbne en ny fil. Datastrømmen hedder igen #9, og vi skal huske filens navn »TEST«. Når vi »åbner filen« betyder det, at mellemlageret nu vil opsamle de signaler, der kommer fra datarecorderen, når den har fundet filnavnet på båndet, – hvis det er der. Hvis filnavnet ikke findes, vil systemet fortsætte med at søge hele båndet igennem! Den slags opgaver klarer en diskettestation meget bedre, idet den

```
10 OPENIN "TEST"
20 INPUT#9,X
30 PRINT"X er";X
40 CLOSEIN
```

Fig. 7.5 Indlæsning af variabelværdien fra båndet

straks kan finde ud af, om filnavnet findes på disketten. Når filen er fundet, kan vi indlæse vores data. Det klarer linie 20 med INPUT #9,X. INPUT gælder altid tastaturet, men når vi sætter #9 efter INPUT, vil det ske fra datarecorderen i stedet. Da vi har sat vores fil i OPENIN-linien, så kan der kun komme data fra en fil, der hedder TEST. Linie 30 indlæser vores data, og

linie 40 lukker for datastrømmen igen. Det hele ser jo ret enkelt og ligetil ud, men se alligevel grundigt på disse to små programmer, for de indeholder faktisk en masse, som du er nødt til at have check på, når du skal lagre filer. Bemærk fx, at vi kan tildele det, der indlæses fra båndet, et hvilket som helst variabelnavn, der måtte passe os. Vi brugte A, da vi lagrede og X da vi indlæste igen. For computeren gør det ingen forskel om INPUT kommer fra et kassettebånd eller fra tastaturet.

```
10 OPENOUT "Lige tal"
20 CLS:FOR n=0 TO 50 STEP 2
30 PRINT#9,n
40 PRINT#0,n;" ";
50 NEXT
60 CLOSEOUT
```

Fig. 7.6 Oprettelse af en fil med tal og lagring af tallene.

Prøv nu noget mere ambitiøst – nemlig at oprette en fil med tal. Fig. 7.6 viser et program, der laver de lige tal fra 0 til 50. Derefter lagres de på bånd og udskrives samtidig på skærmen. Der er kun seks linier i programmet, men tre af dem indeholder ordrer, som det er vigtigt at du forstår. Vi starter med linie 10. Her finder vi en OPENOUT ordre, der forbinder datastrømmen med et mellemlager. Det er datastrøm #9 – den til datarecorderen – og filnavnet er »Lige tal«. Det næste der sker er, at vi skal lave selve filen, og det sker med løkken, der starter i linie 20. Her antager variablen n værdien af de lige tal, – NEXT står i linie 50. Hvordan får vi disse tal ind i mellemlageret? Det klarer linie 30 med PRINT#9,n – og da det er en hel streng tal, der skal optages, opsamler mellemlageret tallene *inden* datarecorderen starter. Computeren er langt hurtigere end datarecorderen. FOR ... NEXT-løkken kan nemt nå at køre færdig inden motoren overhovedet når at starte! Hver gang linie 30 køres bliver værdien af n midlertidigt lagt ind i mellemlageret. Mellemlagerets funktion er at forbinde computerens lager med datarecorderen, sådan at dataoverførslen kan ske effektivt. I dette lille eksempel bliver alle de tal, filen indeholder, overført til mellemlageret. Endnu er intet blevet optaget, – og datarecorderen kører endnu ikke. Optagelsen sker først når alle data er blevet opsamlet i mellemlageret. CLOSEOUT er en ordre til at afbryde datastrømmen, og når datastrømmen afbrydes skal det mellemlager, som datastrømmen var forbundet til, tømmes. Samtidig sker der det, at der indsættes et *slut-på-filen* signal, sådan at datarecorderen kan finde ud af at filen er slut – selv efter optagelse af mange blokke. Og det er det! Hvis du kører programmet, vil du se tallene komme frem på skærmen, således at du

kan se at løkken kører. Du får først meldingen om at trykke på REC og PLAY, når løkken er færdig.

Vi skal lige se på én ting mere, før vi for alvor begynder at jonglere med data. Nogle gange er det ikke hensigtsmæssigt at få meldingerne om at gøre datarecorderen klar, og om optagelse af de forskellige blokke frem på skærmen. Du kan hindre disse udskrifter ved at lade den første karakter i filnavnet være et udråbstegn. Fx kan du ændre linie 10, så der står OPENOUT "!Lige tal", når programmet kører vil du kun se tallene, og optagelsen vil starte uden yderligere udskrifter. Det er nyttigt, men husk nu at trykke REC og PLAY ned! Denne detalje giver dig imidlertid også mulighed for at styre tingene selv, idet du kan få dine egne meldinger frem i stedet for – det ser vi på straks.

Programmet her har vist dig mellemlageret i funktion, og mellemlageret er stort nok til en masse data. Prøv nu at lave programmet om, så det svarer til fig. 7.7. I dette program bruger vi udråbstegn i filnavnet, for at udelade meldingerne. Vi lader derfor linie 20 og 30 give instruktioner om at gøre klar til optagelse. Denne gang er der langt flere data i filen – adskillige tusinde bytes – og når du kører programmet vil du opdage at tallene frem til 598 vil blive udskrevet. Derefter starter datarecorderen, og den første blok data bliver optaget. Så udskrives tallene frem til 1160, og en ny blok optages, og det samme sker ved 1672, og det fortsætter for hver 512 tal. Det der sker er altså at mellemlageret fyldes op af løkken og det tømmes derefter af datarecorderen. Grunden til at løkken kører så længe hver gang er, at der hver gang skal laves en udskrift på skærmen.

```
10 OPENOUT "!Flere lige tal"
20 PRINT"Tryk REC og PLAY ned nu."
30 PRINT"Tryk på mellemrumstangenten for
 at";"starte."
40 WHILE INKEY(47)=-1:WEND
50 CLS:FOR n=0 TO 5000 STEP 2
60 PRINT#9,n
70 PRINT n
80 NEXT
90 CLOSEOUT
100 PRINT"Indlæsning til lageret slut.";
"Stop Datarecorderen."
```

Fig. 7.7 En meget længere talfil, der demonstrerer mellemlageret i funktion. De sædvanlige meldinger udelades ved at bruge ! tegnet i filnavnet.

Disse små programmer har fyldt data ind i en fil, men indtil videre har du kun mit ord for, at der faktisk er kommet noget på båndet. Fig. 7.8 viser et program, der kan læse filen på båndet tilbage i computerens hukommelse.

```
10 PRINT"Tryk PLAY ned, og tryk derefter
 på";"mellemrumstangenten."
20 WHILE INKEY(47)=-1:WEND
30 OPENIN "!Lige tal"
40 FOR N=0 TO 50 STEP 2
50 INPUT#9, A
60 PRINT A;" ";
70 NEXT
80 CLOSEIN
90 PRINT:PRINT"STOP datarecorderen."
```

Fig. 7.8 Indlæsning af Lige tal-filen, med de normale meldinger udeladt. Vi må derfor lave vores egne instruktioner.

Programmet er lavet til »Lige tal«-filen, men denne gang vil vi udelade meldingerne, så vi bruger »!Lige tal«, som filnavn her. Vi må derfor lave vores egne instruktioner, men du skal være opmærksom på, hvor du placerer dem. Hvis du starter programmet med OPENIN "!Lige tal", vil du opdage at programmet ikke kører, som det skal. Det skyldes at maskinen, når den skal udføre OPENIN "!Lige tal", skal finde filnavnet "Lige tal" (selvom det ikke blev optaget som !Lige tal), før der kan ske mere. Du skal derfor have din »PRESS PLAY« ind i begyndelsen af programmet, så OPENIN ordren kan blive udført. Som sædvanlig åbnes filen i linie 30 med OPENIN og filnavnet !Lige tal. Vi indlæser filen fra båndet med INPUT#9 i en løkke med udskrift af tallene. Når programmet kører vil du høre motoren køre, maskinen finder filen og tallene frem på skærmen.

Lad os nu prøve med den lange fil, !Flere lige tal, i fig. 7.9. Denne gang vil vi have tallene til at stå lidt pænere på skærmen, – det gør vi ved at afbryde udskriften. Det er linie 70 der udfører dette, hvis betingelsen IF n/20=INT(n/20) er opfyldt. Dvs. at betingelsen er at 20 skal gå op i n. Hvis der er en rest af divisionen n/20, så er n/20 ikke lig med INT(n/20), som jo kun er den hele del af n/20. Hver gang n antager en værdi, som 20 går op i, så udføres anden del af linie 70. Her laves der en pause, så slettes skærmen og løkken kører videre. Når du kører programmet vil du høre motoren starte, finde filen og indlæse den. Så kommer tallet 0 frem på skærmen under det, der står der i forvejen, og endelig vil tallene blive udskrevet på skærmen i grupper på 20. Motoren vil starte og stoppe med mellemrum, for at få mellemlageret fyldt og

```
10 PRINT"Tryk PLAY ned."
20 PRINT"Tryk mellemrumstangenten ned fo
r at";"starte."
30 WHILE INKEY(47)=-1:WEND
40 OPENIN "!Flere lige tal"
50 FOR n=0 TO 5000
60 INPUT#9,k
70 PRINT k
80 IF n/20=INT(n/20) THEN FOR j=1 TO 200
0:NEXT:CLS
90 NEXT
100 PRINT"STOP datarecorderen!"
```

Fig. 7.9 Indlæsning af „flere lige tal"-filen, hvor du får tid til at se tallene på skærmen.

tømt. Du hører motoren køre hver gang, der læses data fra !Flere lige tal filen på båndet ind i mellemlageret. Denne proces tager sin tid, og det går lidt kvikkere, hvis data-lagringen er sket med den høje hastighed – SPEED WRITE 1. Men er udlæsningen på båndet sket med den langsomme hastighed, så sker indlæsningen tilbage i maskinen også med den lave hastighed. Du har måske lagt mærke til at CLOSEIN er udeladt i dette eksempel. Det kan du gøre, hvis du, når programmet er kørt, sletter maskinens hukommelse med NEW. Hvis du imidlertid skal indlæse flere data, kan du ikke bruge OPENIN igen, med mindre du først har brugt CLOSEIN.

Flere sekventielle filer

Hvad nu hvis det ikke er tal, vi vil have optaget, men nogle navne, som vi selv vil indtaste? Ja, selve proceduren er den samme – for datarecorderen gør det ingen forskel hvad slags data, det drejer sig om. Hver gang du taster ENTER i et INPUT-trin, bliver dataerne ført over i mellemlageret, og der bliver de indtil mellemlageret er fyldt op, eller indtil indtastningen er slut og filen bliver lukket. Her kan du igen se hvor vigtigt mellemlageret er – for du vil vel ikke have, at maskinen skal lagre et bogstav ad gangen, vel?

Fig. 7.10 viser et kort program af denne type. Normalt ville du, hvis du skulle indsamle den slags informationer, lagre navnene som et array. Men det giver jo bl.a. besvær med at dimensionere arrayet. Med mindre du af og til vil se de navne du *har* indlæst, så er der ikke grund til at bruge et array til den slags opgaver. Det kan være noget andet, når du indlæser dem fra båndet – men det ser vi på om lidt.

```
10 SPEED WRITE 1
20 OPENOUT "!Navne"
30 CLS:PRINT TAB(18)"Navne"
40 PRINT:PRINT"Dette program lagrer en f
il med navne";"for dig på kassette. Sørg
 for at data-";"recorderen er klar."
50 PRINT"Tast x efter endt indlæsning."
60 PRINT"Tryk PLAY og REC ned på datarec
orderen,";"og tryk på mellemrumstangente
n når du";"er klar."
70 WHILE INKEY(47)=-1:WEND
80 WHILE navn$<>"x" AND navn$<>"x"
90 INPUT "navn ";navn$
100 PRINT#9,navn$
110 WEND
120 CLOSEOUT
130 PRINT"STOP datarecorderen."
140 SPEED WRITE 0
```

Fig. 7.10 Indlæsning af navne i en fil. Den høje datalagringshastighed er valgt, for at få programmet til at være mere effektivt.

Lad os nu se nærmere på programmet. I linie 10 vælges SPEED WRITE 1, som er den hurtige overføringshastighed. Det betyder *ikke* at båndet kører hurtigere, men blot at informationerne bliver »pakket« tættere på båndet. Vi skal slette ordren igen når programmet er slut, – ellers vil den vedblive at være i funktion. Hvis du ønsker at bruge den lave hastighed til en anden opgave senere, så er det ikke muligt, når SPEED WRITE 1 har været brugt. Linie 20 åbner filen, og meldingerne udelades som sædvanligt. Linie 30 til 60 giver nogle korte instruktioner og linie 70 er »Tryk på mellemrumstangenten når du er klar« pause.

Indlæsnings WHILE ... WEND-løkken starter i linie 80, og ideen er at der kan indlæses navne, indtil der tastes X eller x. Hvis du vil lave en mere anvendelig navnefil, skal du sikkert lave nogle mere fyldestgørende instruktioner. Linie 90 tager imod det navn du indtaster (husk – ingen kommaer i INPUT-trin). Navnet bliver sendt via mellemlageret til datarecorderen i linie 100. Og hvis der ikke er tastet x, sender linie 110 programmet tilbage efter et nyt navn. Hvis du indlæser mange navne, vil du opdage at motoren kører af og til for at tømme mellemlageret. Du kan ikke indtaste noget mens det sker.

Mere om indlæsning fra båndet

Nu har du både nogle tal – og navnefiler lagret på datarecorderen, og det er på tide at gå lidt mere i dybden med indlæsningen fra datarecorderen og med brugen af informationerne. Lad os sige, at de tal vi har i Lige tal-filen, var indlæst af et regnskabsprogram. Det kunne fx være det daglige salg i en lille butik. Vi vil derfor gerne indlæse tallene fra vores fil, lægge dem sammen, og have vist summen på skærmen. Det kan klares med et enkelt program, det er ikke vanskeligt, og du kan sikkert bruge det til mange formål. Fig. 7.11 viser programmet. I linie 10 slettes skærmen, og linie 20 udskriver en titel, og i linie 30 og 40 følger nogle instruktioner. Linie 50 får maskinen til at vente, indtil mellemrumstangenten trykkes ned, og selve programmet starter i linie 60, hvor !Lige tal åbnes. Da vi optog filen "Lige tal", valgte vi alle lige tal mellem 0 og 50 – i alt 26 tal. For at få tallene læst ind igen bruger vi en FOR ... NEXT-løkke, der kører 26 gange. I linie 70 sættes summen til nul. Tallene indlæses et for et i linie 90, hvor de tildeles variablen j, og linie 100 summerer. Når løkken er færdig med at køre, udskrives summen på skærmen, og filen lukkes endelig i linie 130.

```
10 CLS
20 PRINT TAB(9)"Additionsprogram"
30 PRINT:PRINT"Tryk PLAY ned, når båndet
 er parat."
40 PRINT"Start: Tryk på mellemrumstangen
ten."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!Lige tal"
70 total=0
80 FOR n=1 TO 26
90 INPUT#9,j
100 total=total+j
110 NEXT
120 PRINT:PRINT"Summen er";total
130 CLOSEIN
140 SPEED WRITE 0
```

Fig. 7.11 Lige tal-filen indlæses igen med en FOR ... NEXT løkke.

Hvad nu hvis du ikke vidste hvor mange tal, der var i din fil? Så bliver det umuligt at bruge en FOR ... NEXT løkke, fordi vi ikke ville vide hvor mange gange løkken skulle køre. Men det kan vi let klare. Amstrad'ens datarecordersystem sætter altid et signal ind efter den sidste blok data der optages. Vi

kan spore dette slutsignal ved hjælp af funktionen EOF (End Of File). Hvis EOF=0 så er »enden af filen« ikke nået. Men er EOF= −1, så er filen slut. Programmet, som skal indlæse vores fil fra datarecorderen, kan derfor skrives som du ser det i fig. 7.12. Her bruger vi en WHILE ... WEND-løkke i stedet for FOR ... NEXT. WHILE-betingelsen er at EOF=0, for hvis EOF=0, så er vi endnu ikke nået til slutningen af filen.

```
10 CLS
20 PRINT TAB(9)"Additionsprogram"
30 PRINT:PRINT"Tryk PLAY ned når båndet
er parat."
40 PRINT"Start: Tryk mellemrumstangenten
 ned."
50 WHILE INKEY(47)=-1:WEND
60 OPENIN "!Lige tal"
70 total=0
80 WHILE EOF=0
90 INPUT#9,j
100 total=total+j
110 WEND
120 PRINT:PRINT"Summen er";total
130 CLOSEIN
140 PRINT"STOP datarecorderen."
```

Fig. 7.12 En bedre metode til at genindlæse en fil på. Her behøver du ikke at vide hvor mange poster, der er i filen.

Tilbage til navnene

Nu har vi prøvet at læse tallene ind i maskinen igen, og vi har brugt dem i et program. Så nu må det være på tide, at se på hvordan vi kan læse den navnefil, som vi oprettede tidligere, tilbage i maskinen. Da vi lavede filen, blev hvert navn optaget, og det sædvanlige slutsignal blev til sidst indlagt på båndet. Som nævnt ville vi normalt have brugt et array til navnene, således at computeren kan arbejde med dem. Fx kan vi med arrays sætte navne i alfabetisk orden. Men det er ikke altid, der er brug for et array. Lad os sige at du gerne vil finde navne, der begynder med bogstavet J. Det kan du gøre med programmet i fig. 7.13.

De første linier virker velkendte. Når programmet er kørt kan du finde et navn i filen, blot ved at indtaste det første bogstav i navnet. Der skal skiftes til SPEED WRITE 1 i linie 50, fordi navnefilen blev optaget med den høje hastighed. Når den høje hastighed bruges, skal du forvisse dig om at båndet

```
10 CLS:PRINT TAB(16)"Navnefinder":PRINT
20 PRINT"Dette program kan finde et navn
 for dig";"fra 'navne' filen."
30 PRINT"Tryk PLAY og mellemrumstangente
n ned";"når du er klar."
40 WHILE INKEY(47)=-1:WEND
50 SPEED WRITE 1
60 INPUT "Det første bogstav i navnet er
 ";q$
70 OPENIN "!Navne"
80 WHILE EOF=0 AND q$<>LEFT$(n$,1)
90 INPUT#9, n$
100 WEND
110 PRINT:PRINT"Navnet er ";n$
120 CLOSEIN:SPEED WRITE 0
130 PRINT"STOP datarecorderen."
```

Fig. 7.13 Her gennemsøges filen efter en bestemt post – i dette tilfælde et navn, der begynder med et bestemt bogstav.

er spolet helt tilbage til et punkt før starten af filen. Det hurtige system er nemlig mere følsomt overfor at starte efter det punkt, hvor de første toner er optaget. Linie 60 beder om første bogstav i et navn (husk stort bogstav!). Linie 70 åbner filen, så den kan indlæses fra båndet af linie 80, der starter en løkke, som tager det første navn fra mellemlageret og checker først om det er EOF-signalet, og derefter om det første bogstav svarer til det du søger efter. Hvis der ikke findes et navn med det rigtige forbogstav i filen, vil linie 80 til sidst støde på EOF-signalet, og dermed afslutte programmet. Men hvis navnet findes, så vil linie 80 udskrive det, og dermed slutter programmet. WHILE ... WEND er en meget effektiv ordre at bruge i programmer som dette.

Programmet er meget godt til små mængder data, men kun hvis alle data er forskellige. Hvis der er en Mogens og en Marianne i filen, vil programmet give dig det navn, det først finder. Du må derfor ændre de tests, der er i løkken, hvis du vil have programmet til at udskrive alle navne, der begynder med et givet bogstav i filen. Programmer af denne type kan bedre laves ved brug af et array, der kan rumme alle data, når de er i computerens hukommelse. Så bliver båndet kun brugt som lager, og søgningen foregår så i computerens lager. Nu tænker du måske på, om man ikke ligeså godt kan opbevare sine data i DATA-linier. Men der er fordele ved dette system, fordi vi kan lave data med ét program, og bruge dem i adskillige andre. På den måde kan du lave de programmer, der henholdsvis indlæser og bruger dine data ret korte, så der er endnu bedre plads i Amstrad'ens store hukommelse. Datare-

cordersystemet giver dig altså mulighed for at bruge små, korte programmer og masser af data.

```
10 CLS:PRINT TAB(16) "Navnefinder"
20 PRINT:PRINT TAB(2)"Indlæs navnefilen
efter instruktionen."
30 PRINT"Når filen er helt indlæst, så i
ndtast";"det første bogstav i det navn d
u ønsker";"at se."
40 PRINT"Tast 0 for at afbryde programme
t."
50 PRINT:PRINT"Spol kassetten tilbage ti
l start.";"Nulstil tælleren."
60 PRINT"Tryk PLAY og mellemrumstangente
n ned";"når du er klar."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"Vent..."
90 SPEED WRITE 1
100 OPENIN "!Navne"
110 J=0:WHILE EOF=0
120 INPUT#9,n$
130 j=j+1
140 WEND:CLOSEIN
150 CLS:PRINT:PRINT"Tryk på STOP."
160 PRINT:PRINT"Spol nu båndet tilbage t
il start igen,";"tryk på PLAY og mellemr
umstangenten ned"
170 PRINT"igen - når du er klar."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT TAB(10)"Vent...."
200 OPENIN"!Navne"
210 DIM navn$(j)
220 FOR n=1 TO J
230 INPUT#9,navn$(n)
240 NEXT:CLOSEIN:SPEED WRITE 0
250 PRINT:PRINT"Tryk på STOP nu."
260 FOR x=1 TO 2000:NEXT
270 CLS:PRINT"Skriv det første bogstav a
f navnet.":q$="x":m=0
280 WHILE q$<>"0"
290 INPUT q$:q$=LEFT$(q$,1):m=0
300 FOR x=1 TO j
```

```
310 IF q$=LEFT$(navn$(x),1) THEN PRINT n
avn$(x):m=1
320 NEXT
330 IF m=0 AND q$<>"0" THEN PRINT"Navnet
 kan ikke findes - prøv med an-";"det na
vn."
340 WEND
350 PRINT"Programmet er slut."
```

Fig. 7.14 Data indlæses fra båndet ind i et array. Filen indlæses to gange: én gang for at maskinen kan dimensionere arrayet, og én gang for at indlæse data.

Fig. 7.14 viser hvordan data fra et program, som har været lagret på datarecorderen, kan indlæses i et array i computerens hukommelse. Hvis vi skal indlæse data i et array, får vi et problem. Vi skal nemlig dimensionere arrayet, så der er plads til alle posterne. For at kunne gøre det skal vi vide hvor mange poster, der er. Det er jo let nok, hvis vi brugte et array til at rumme posterne, når vi optog dem. Lad os fx sige, at vi først havde brugt INPUT til at indlæse vores data i et array Navn$(j), i stedet for at optage dem direkte. Vi kunne så åbne filen, optage værdien af j med PRINT#9,j, og derefter lave en løkke til at optage posterne i arrayet. Men hvad nu hvis vi ikke brugte et array og ikke talte vores poster, inden vi optog dem? En mulighed er, at vi noterer os hvor mange poster, der er i filen, og indlæser antallet i programmet når det »spørger« os. Det kunne gøres sådan her:

```
INPUT"Hvor mange poster ";n:DIM Navn$(n)
```

hvorefter programmet ville blive dimensioneret til de foreliggende data. En anden mulighed ville være at indsætte en tæller i det program vi brugte til optagelsen af filen:

```
INPUT Navn$:n=n+1
```

og optage tallet med et filnavn, der henviste til hovedprogrammet et andet sted på båndet. Hovedprogrammet kunne fx hedde NAVNEVENNER, og det andet kunne hedde TALVENNER. Det kan måske se lidt besværligt ud, men det er sliddet værd, for at opnå en præcis dimensionering. Når dimensioneringen er præcis, får du aldrig problemer med fejlmeldinger, fordi du har dimensioneret for lidt plads til det antal poster, du har. Det betyder også at du udnytter Amstrad'ens hukommelse effektivt, så du kan ændre dimensioneringen sådan, at du altid har den plads, du skal bruge – og ikke mere.

Men hvis du nu engang har en fil som vores Navne fil, hvor vi ikke kender antallet af navne, måske fordi filen løbende bliver suppleret, så må vi finde på

noget andet. Dette er et af de steder hvor diskette-systemet er langt bedre end et bånd-system, fordi et antal kan optages på disketten før eller efter vores data, og antallet kan så indlæses først, hvis vi har brug for det. Fig. 7.14 viser en måde at tackle problemet på, som ikke tager for lang tid – selv ikke ved længere filer. I programmet bliver posterne først indlæst, en for en, og derefter talt, indtil slutsignalet for filen dukker op. Derefter bliver du bedt om at spole tilbage til start. Nu ved programmet hvor mange poster, der er i filen, og en korrekt dimensionering kan derfor foretages. Så kan filen indlæses igen, og denne gang kan posterne placeres i et array. Når navnene skal vælges kan vi bruge en løkke fordi vi ikke skal indlæse filen igen.

Hvis vi går programmet igennem trin for trin, ser vi instruktionerne i linie 10 til 60, og linie 70 venter indtil mellemrumstangenten trykkes ned. I linie 80 udskrives et »Vent ...«, for at give tid til at indlæsningen kan ske. SPEED WRITE 1 vælges, fordi den blev brugt, da filen blev optaget, og variablen j nulstilles. I løkken indlæses en post hver gang, og j opskrives hver gang løkken kører. I linie 140 lukker vi filen, fordi vi senere har brug for at åbne den igen. Så skal vi spole båndet tilbage til nul igen, og der udskrives en påmindelse herom. Værdien af j bruges til at dimensionere arrayet navn$ i linie 210, lige før navnene indlæses fra filen. Da vi nu kender antallet af poster kan vi bruge en FOR ... NEXT-løkke i linie 220 til 240 til at indlæse navnene i arrayet. Når først navnene er indlæst kan vi bruge vores »Navnefinder« i linie 280 til 350. Bemærk at vi giver q$ en værdi forskellig fra 0, før løkken starter. Hvis der bliver fundet et navn, bliver variablen m sat til 1. Hvis der ikke kan findes et navn med det ønskede forbogstav er m = 0 og meldingen i linie 330 udskrives. Når du taster 0 for at afslutte søgningen, bliver m =0, og meldingen linie 330 udelades med betingelsen IF m = 0 AND q$<>"0". Hvis q$ = "0", så bliver meldingen *ikke* udskrevet. Det ser pænere ud på den måde!

Nogle forbedringer

Lad mig med det samme slå fast, at der ikke kan ændres i en fil, der er optaget på bånd. Det er derimod muligt at indlæse filen i computeren, lave nogle ændringer, og optage den på bånd på ny. Du kan derfor optage en ny fil, *med samme navn* som den gamle, på et andet bånd, eller et andet sted på det samme bånd. Fordelen ved at bruge det samme navn til en opdateret fil er, at du kan bruge samme program til at læse filen med. Det er noget vanskeligere at gøre det samme med et diskette-system. Teknikken er vist i fig. 7.15 som følger fig. 7.14 ret nøje.

Linie 10 til 260 svarer praktisk talt til linierne i fig. 7.14 bortset fra instruktionerne. Linie 140 indeholder dog nu j = j−1. Det forhindrer at filens sidste

```
10 CLS:PRINT TAB(12) "Opdatering af fil"
20 PRINT:PRINT TAB(2)"Indlæs navnefilen
efter instruktionen."
30 PRINT"Når filen er helt indlæst, så f
ølg";"følg instruktionerne angående ny o
ptag-";"else."
40 PRINT"Du kan nu tilføje nye poster ti
l filen,";"- skriv 0 efter sidste indtas
tning."
50 PRINT:PRINT"Spol kassetten tilbage ti
l start.";"Nulstil tælleren."
60 PRINT"Tryk PLAY og mellemrumstangente
n ned";"når du er klar."
70 WHILE INKEY(47)=-1:WEND
80 PRINT:PRINT TAB(10)"Vent..."
90 SPEED WRITE 1
100 OPENIN "!Navne"
110 J=0:WHILE EOF=0
120 INPUT#9,n$
130 j=j+1
140 WEND:CLOSEIN:j=j-1
150 CLS:PRINT:PRINT"Tryk på STOP."
160 PRINT:PRINT"Spol nu båndet tilbage t
il start igen,";"tryk på PLAY og mellemr
umstangenten ned"
170 PRINT"igen når - du er klar."
180 WHILE INKEY(47)=-1:WEND
190 PRINT:PRINT TAB(10)"Vent...."
200 OPENIN"!Navne"
210 DIM navn$(j)
220 FOR n=1 TO J
230 INPUT#9,navn$(n)
240 NEXT:CLOSEIN:SPEED WRITE 0
250 PRINT:PRINT"Tryk på STOP nu."
260 FOR x=1 TO 2000:NEXT
270 CLS:PRINT"Gør klar til at optage den
ne fil igen."
280 PRINT:PRINT"Find et nyt sted på kass
etten, hvor der";"er plads til en ny opt
agelse":PRINT"Eller brug en anden kasset
te."
290 PRINT"Nulstil tælleren igen og følg
instruk-";"tionerne."
```

```
300 PRINT:PRINT"Tryk PLAY og REC ned og
derefter mellem-";"rumstangenten - når d
u er klar."
310 WHILE INKEY(47)=-1:WEND
320 OPENOUT "!Navne"
330 FOR n=1 TO j
340 PRINT#9,navn$(n):NEXT
350 CLS:PRINT:PRINT TAB(12)"Ny indlæsnin
g."
360 PRINT:PRINT"Skriv de navne du vil ti
lføje til filen";"nu.":PRINT"Tast 0 for
at afslutte indtastningen."
370 navn$="x":WHILE navn$<>"0"
380 INPUT navn$
390 PRINT#9,navn$
400 WEND
410 SPEED WRITE 0
420 CLOSEOUT
430 PRINT"Programmet er slut."
```

Fig. 7.15 Opdatering af en fil. Det kan gøres ved at indlæse filen i computeren, lave nogle ændringer, og optage den et andet sted på båndet.

post bruges, det er jo den karakter, x eller 0 eller hvad der nu stod i programmet da filen blev oprettet, som afslutter indlæsningen. På den måde har vi en fortløbende fil med et x eller et 0 i slutningen, fremfor en fil med disse karakterer spredt over det hele – hvor filen tidligere er sluttet.

Linie 270 til 340 gør klar til en ny optagelse af filen. Jeg siger »gør klar« for med mindre filens navn er meget langt, så sker der tilsyneladende ikke noget i linie 320 til 340. Det skyldes at filen blot overføres til mellemlageret, og ikke bliver optaget på dette trin. Du bliver så bedt om at indlæse nye navne. Mens du gør det bliver mellemlageret måske fyldt op, og optagelsen starter. Mens det sker kan du ikke indtaste nye navne. Hvis du imidlertid kun indtaster nogle få navne, vil lageret ikke blive fyldt, og du vil ikke høre motoren starte før du har tastet det afsluttende nul. Dermed er den nye fil færdig. Hvis du vil checke den så kør (RUN) igen, og når du bliver bedt om at indtaste nye navne, så tast ESC to gange. Nu kan du indtaste denne linie:

```
FOR z=1 TO j:?navn$(z);" ";:NEXT
```

og ENTER. Nu vil du se navnene blive udskrevet på skærmen. Hvis du vil have navnene udskrevet efter et system, fx alfabetisk, må du selv skrive programmet!

Kapitel 8

Vinduer og andre skærmeffekter

Det er aldrig kedeligt at arbejde med en computer, men der er forskellige effekter vi kan bruge, for at gøre vores programmer lidt mere spændende. En af de effekter, som nogle få af de nyeste hjemmecomputere kan lave, er *vinduer*. I dette kapitel skal vi se på vinduer og på nogle andre effekter, der kan bruges for at gøre almindelige tekster og figurer mere overskuelige og mere interessante at se på. Lad os starte med vinduerne, for dem har du sikkert kun *set* i funktion før, for som sagt er Amstrad en af de få computere, hvor man kan lave vinduer uden en masse programmering.

Et »vindue« (engelsk: window) er simpelt hen et udsnit af skærmen, der kan bruges uafhængigt af andre skærm-udsnit eller endda uafhængigt af resten af skærmen. Et vindue kan vise udskrifter, vindues-tekster kan rulle og vinduet kan slettes helt uden hensyn til hvad der sker på skærmen i øvrigt. Men prøv selv. Skriv et eller andet på skærmen og skriv derefter denne ordre:

WINDOW 15,25,2,6 (og så ENTER)

Brug så CLS. Nu vil du se at kun en lille del af skærmen – nemlig vinduet – bliver slettet. Prøv nu at skrive et kort program og du vil se, at det kommer frem i vinduet – en lille selvstændig skærm på skærmen. Vinduet laver automatisk linieskift, det ruller osv. – præcis som den »almindelige« skærm. Og resten af skærmen er helt upåvirket. CLS sletter kun vinduet og ikke hele skærmen. For at vende tilbage til hele skærmen kan du bruge ordren MODE 1 (ENTER).

Det var en lille forsmag – lad os nu se nærmere på hvordan vi kan styre sådan nogle vinduer. Som du har set hedder ordren WINDOW og den skal efterfølges af fire tal. De svarer helt til LOCATE-tallene og kontrollerer – i rækkefølge – venstre side, højre side, top og bund i vinduet (se Appendix VI i manualen). Når du bruger denne type WINDOW-ordrer kommer alle udskrifter i dette vindue. Det er datastrøm #0, der bruges, og det er den datastrøm der normalt bruges til PRINT osv. Men du kan bruge WINDOW i

forbindelse med nummeret på en anden datastrøm, som fx #2. Når du gør det kommer vinduet kun i brug, når du bruger en ordre som fx PRINT#2,"WINDOW".

Fig. 8.1 viser et eksempel. Først bliver hele skærmen slettet og derefter bliver et vindue koblet til datastrøm #2 med ordren:

WINDOW#2,10,30,1,5

Det betyder at datastrøm nummer 2 bliver forbundet med et skærmvindue. De tal der kommer efter #2 og kommaet angiver både vinduets størrelse og placering. Venstre side er kolonne 10 og højre side er kolonne 30. Overkanten er linie 1 og underkanten er linie 5. Det betyder at ordrer som fx CLS#2 og PRINT#2 nu henholdsvis sletter vinduet og laver udskrift i det. Denne operation er selektiv. Med mindre du lukker datastrømmen eller forbinder den til noget andet, så vil vinduet blive kontrolleret af datastrøm #2. En anden væsentlig forskel er at nu kan vi bruge hele skærmen på normal vis. Vi kan lave udskrifter og vi kan slette dem, – og her vil der blive skrevet hen over vores vindue, og CLS sletter hele skærmen. Men selvom hele skærmen er fyldt med ene udskrifter, kan vi slette vinduet med CLS#2.

```
10 CLS
20 WINDOW#2,10,30,1,5
30 WINDOW#3,12,28,10,15
40 PRINT#2,"Her er window #2";"i brug"
50 GOSUB 150
60 PRINT#3,"Se på teksten i  window #2 -
";"her er window #3"
70 GOSUB 150
80 CLS #2
90 GOSUB 150
100 CLS #3
110 GOSUB 150
120 PRINT#2,"window #2"
130 PRINT#3,"window #3"
140 END
150 FOR n=1 TO 2000:NEXT:RETURN
```

Fig. 8.1 Her laves et vindue, som forbindes til datastrøm #2, og et andet, som forbindes med datastrøm #3.

I eksemplet laves et vindue i linie 20 og et andet – længere nede på skærmen – i linie 30. For at vise, at vinduerne faktisk er der, laver linie 40 en udskrift i vindue #2. Her kunne du godt skrive LIST#2 i programmet, men

maskinen kan ikke udføre de programlinier, som kommer efter en LIST-ordre i et program. Så kommer der en pause, der laves af subrutinen i linie 1000. Efter pausen bliver der udskrevet noget tekst i vindue #3. Resten af programmet viser hvordan vinduerne kan modtage udskrifter og slettes uafhængigt af hinanden. Bemærk dog, at når programmet er slut kommer »Ready« markøren frem i øverste venstre hjørne. Hovedskærmen, der har datastrøm #0, vil altid »køre ind over« de enkelte vinduer.

Vi kan lave nogle andre tricks med en anden WINDOW-ordre som hedder WINDOW SWAP (swap = »bytte«). Lad os fx sige at du har et program, hvor du skal indtaste nogle data og optage dem på bånd, ligesom det vi så på i kapitel 7. Her kunne du have et vindue midt på skærmen, hvor indtastningen kom frem, og et andet vindue, hvor forskellige meldinger kunne blive udskrevet. Det kunne fx være meldingerne vedrørende datarecorderen, som kunne blive udskrevet et bestemt sted. Hvis vi laver et vindue nede i bunden af skærmen, som fx kan forbindes med datastrøm #3, så kan du skrive:

```
WINDOW SWAP #0,#3
```

For at få det der normalt kommer frem på hovedskærmen til at komme frem i vindue #3 i stedet. Fig. 8.2 viser hvordan det kan ske. Programmet er forkortet noget, idet jeg ikke har medtaget alle datarecorderrutinerne. Det vil vi ikke bruge plads på her. Du kan sætte en kassette ind i datarecorderen, og nøjes med at trykke PLAY ned, når du bliver bedt om at optage – på den måde slipper du for at bruge bånd på en optagelse, du alligevel ikke vil bruge. Der er i øvrigt flere måder at gøre den slags på *). Hvis du forsigtigt trykker den lille føler, der sidder under kassettelågen inde bagved til venstre, er det muligt at trykke REC ned selvom der ikke er en kassette i datarecorderen. På den måde kan du »snyde« mekanikken til at tro, at der er en kassette i maskinen. Og det kan være meget nyttigt, hvis du vil experimentere med programmer, der skal optages på kassette.

Vi vil vende tilbage til vores program: de to vinduer kaldes #2 og #3. Vindue #3 er blot en enkelt linie, helt nede i bunden af skærmen. På den måde kan vi sikre at der kun udskrives én melding ad gangen, således at vi

---

*) Prøv at se i manualen Chapter 2 Page 3, der kan du se et billede af en kassette. Bag på kassetten er der to små tapper, – hvis de mangler kan der ikke optages på båndet. Det skyldes, at der i datarecorderen sidder en lille føler, der checker om der er tapper på kassetten, – og er der ikke det, kan REC ikke trykkes ned! Hvis du vil sikre dine bånd mod at blive slettet ved et tilfælde, skal du altså knække tapperne af på kassetten. Og hvis du senere vil optage på båndet, kan du blot sætte et stykke tape over hullet. Bemærk at »hullerne« sikrer én side hver. Er der ingen kassette i båndoptageren, svarer det til at der sidder en »slettesikker« kassette, idet føleren jo ikke bliver trykket ind. Dette princip gælder i øvrigt for alle kassettebåndoptagere. O.A.

```
10 CLS
20 WINDOW#2,2,39,5,10
30 WINDOW#3,2,39,25,25
40 PRINT#3,"Indtast navne. Tast X for at
 slutte."
50 WHILE Navn$<>"X" AND Navn$<>"x"
60 INPUT Navn$
70 WEND
80 WINDOW SWAP 0,3
90 OPENOUT "Navne"
100 PRINT#9,"Navn$"
110 CLOSEOUT
120 REM Indtast MODE 1 for at få alminde
lig skærm igen!
```

Fig. 8.2 Med WINDOW SWAP kan vi holde meldinger væk fra hovedskærmen.

ikke forveksler nye meldinger med gamle. Du skal selvfølgelig være sikker på at der ikke kommer meldinger, der fylder mere end en enkelt linie! Programmet er helt uden problemer frem til linie 80. Her byttes der rundt på vindue 0 og 3 – bemærk at der *ikke* bruges # her! – hvis du bruger # sammen med WINDOW SWAP får du en fejlmelding! Efter linie 80 kommer alle meldinger, der ellers ville komme normalt frem på skærmen i dette vindue i stedet. Og det kan faktisk være meget nyttigt at holde meldingerne adskilt fra den øvrige tekst.

Brug farver

Nu skal vi til at se på Amstrads farve-ordrer *). Der er fire af dem, der er specielt vigtige: BORDER, PAPER, PEN og INK. Vi starter med den simpleste af dem, BORDER. BORDER (»ramme«) bruges med et eller to tal. Tallene er farvekoder og hvis du kun bruger én kode, vil du få en ramme i én bestemt farve udenom selve skærmfeltet. Farvernes koder kan du finde i fig. 8.3. Hvis du bruger *to* farvekoder sammen med BORDER, vil du få en blinkende ramme, der hele tiden skifter mellem de to farver.

Fig. 8.4 demonstrerer BORDER. Først laves der et vindue oppe i højre hjørne, hvor vi kan få udskrevet farvekoderne. Derefter startes en løkke,

*) Selvom du har en grøn monitor, skal du alligevel læse dette afsnit, for langt det meste giver fine effekter også uden farveskærm. O.A.

Kode	Farve
0	Sort
1	Blå
2	Højblå
3	Rød
4	Magenta (rødt+blåt lys)
5	Grålilla
6	Højrød
7	Violet
8	Højmagenta
9	Grøn
10	Cyan (blåt+grønt lys)
11	Himmelblå
12	Gul
13	Hvid
14	Pastelblå
15	Orange
16	Lyserød
17	Pastelmagenta
18	Højgrøn
19	Søgrøn
20	Højcyan
21	Lysegrøn
22	Pastelgrøn
23	Pastelcyan
24	Højgul (guld)
25	Pastelgul
26	Højhvid

Fig. 8.3 Farvekoderne. Koderne er ordnet efter hvor lyse de er, – som de ville fremtræde på en sort/hvid monitor (eller s/h TV).

som gennemløber alle farvekoder for BORDER. Hver farve holdes på skærmen i en vis periode ved hjælp af en forsinkelse. I modsætning til de tidligere forsinkelser, som vi har brugt, laves denne forsinkelse med Amstrads indbyggede *timer.* TIME (»tid«) er et tal der starter på nul i det øjeblik maskinen tændes, og som opskrives 300 gange i sekundet. I subrutinen i linie 150 til 170 sættes en variabel lig med TIME's værdi på et bestemt tidspunkt. For hvert sekund der går, er TIME blevet 300 større, så en WHILE ... WEND-løkke venter indtil TIME er blevet 600 større end variablen (start) – dvs 2 sekunder. Det er enkelt, elegant og langt mere præcist end en FOR ... NEXT løkke. Når programmet er færdigt med de 27 farver, køres de blinkende farver igennem, men blink-frekvensen er her bestemt af en anden variabel, og den kan du selv styre. Rammen fortsætter med at blinke efter programmet er slut, og

```
10 REM LIST PROGRAMMET FØR DET KØRES
20 WINDOW#2,35,40,5,5
30 FOR n=0 TO 26
40 PRINT#2,n
50 BORDER n
60 GOSUB 150
70 NEXT
80 GOSUB 150
90 FOR n=1 TO 26
100 PRINT#2,n;",";27-n
110 BORDER n,27-n
120 GOSUB 150
130 NEXT
140 END
150 start=TIME
160 WHILE TIME<start+600:WEND
170 RETURN
```

Fig. 8.4 Demonstration af BORDER, hvor TIME bliver brugt til at fastholde farverne i en bestemt periode.

du kan prøve at ændre frekvensen ved at skrive SPEED INK 10,10. Når du taster ENTER, vil du se at blinkene kommer hurtigere. Skriver du SPEED INK 20,100 får du en anden effekt. Det første tal udmåler tiden for den ene farve, og det andet tal udmåler tiden for den anden farve. Her er tiderne 1/50-dele af et sekund, så tallet 50 giver et sekund for en farve. Du skal være lidt forsigtig med denne ordre – *nogle blinkfrekvenser på omkring 8 blink i sekundet kan udløse anfald hos personer, der lider af epilepsi!*

Den blinkende ramme kan være ubehagelig på flere måder. Du kan se at når rammen blinker, så skifter billedet på skærmen størrelse. Det skyldes at skærmen ikke er konstrueret til at klare disse blink. Du kan godt få en monitor der kan klare blinkene – men den koster nok omkring kr. 15.000, så den er nok for dyr for de fleste! For at få stoppet blinkeriet skal du skrive BORDER 1 (ENTER). MODE kan ikke bruges her.

De næste ordrer er PAPER, PEN og INK (»papir, »pen« og »blæk«). Navnene siger noget om funktionen, men ikke alt. Hvis du har brugt en Spectrum før du fik en Amstrad, vil det være lidt forvirrende for dig. Hvis du ikke kender udtrykkene, kan det også være forvirrende – så læs her! Vi starter med INK.

INK (»blæk«) står for farve, og for hver *mode* har du en begrænset mængde INKs, som du kan bruge. I MODE 0 kan du have op til 16 forskellige

farver INK på skærmen. *Disse 16 farver kan frit vælges mellem 27 mulige farver i fig. 8.3.* I MODE 1, den normale mode til tekst, kan der bruges op til 4 farver INK. Og her kan du også vælge mellem de 27 farver. I MODE 2, hvor vi har den høje opløsning med 80 karakterer pr. linie, kan vi kun have 2 INK farver ud af de 27. Hvis du forestiller dig at INK er nogle »inkpots« *), så har du kun et begrænset antal af dem til rådighed. I MODE 1 fx, har du kun fire inkpots til rådighed. Du kan putte i alt 27 forskellige farver i dine inkpots, men du kan altså kun bruge fire ad gangen (i MODE 1). Hvis du vil bruge andre farver, må du først fylde nye farver på dine inkpots!

Her er det, at Amstrad begynder at skille sig ud fra andre farvecomputere. I MODE 1 er INK-tallene 0, 1, 2 og 3. Og jeg skriver *tal* ikke *farver,* for disse tal er *ikke* farvekoderne! Hvis vi vender tilbage til vores inkpots, så er disse tal blot numrene på disse inkpots. Vi kan bruge INK-ordren til at bestemme, hvilke farver vi vil putte i de forskellige inkpots. Fx betyder INK 2,7 at vi hælder farve 7 (violet) på inkpot 2. Når du starter i MODE 1, så er de fire inkpots fyldt op på forhånd. Inkpot 0 er fyldt med blå, det er den inkpot, der bestemmer baggrundsfarven. Inkpot 1 er fyldt med guld-farve – det er tekstens farve. Inkpot 2 og 3 med henholdsvis højcyan og højrød. På fig. 8.5 kan du se standard-farverne og deres koder.

MODE 0 og MODE 1

Inkpot 0	farvekode	1	blå
Inkpot 1	farvekode	24	højgul (guld)
Inkpot 2	farvekode	20	højcyan
Inkpot 3	farvekode	6	højrød

(I MODE 2 bruges farverne 1 og 24 fra start).

Fig. 8.5 Sådan er standard-farver i MODE 0, 1 og 2

Hvordan kan disse farver så ændres? – Ved brug af INK-ordren! I MODE 1 kan du bruge INK-farverne 0 til 3 (faktisk kan du godt bruge større tal, men det er blot de fire samme farver, der gentages). Hvis du vil have inkpot 1 fyldt med farve 15, så skal du blot bruge ordren INK 1,15. Husk mellemrummet mellem »K« i INK og tallet. Hvis du skriver *to* farvekoder med et komma imellem, så får du en inkpot med skiftende farve! Farven vil da hele tiden skifte mellem de to farver, du har valgt. Fx vil INK 1, 3, 7 give en farve der skifter mellem farve 3 og 7. Du kan som før ændre blinkfrekvensen med SPEED INK. Det er næsten ligesom de gamle vittigheder om stribet maling!

*) Inkpot betyder blækhus. Her bruges det engelske ord, idet det gør det lettere at huske sammenhængen med INK. O.A.

Når du tænder computeren er den altid automatisk i MODE 1, og »standardfarverne« er som beskrevet ovenfor. Det er også fastlagt hvilken farve baggrunden skal være og hvilken farve teksten skal være. Baggrunden – PAPER – får farven i inkpot 0, og teksten får farven i inkpot 1. Jeg vil her igen understrege – især hvis du før har brugt Spectrum – at INK-tallene *ikke er farvekoder*, men blot er inkpot-numre. De farver der kommer frem på skærmen, er bestemt af hvilke inkpots, der bruges og af hvilke farver, der er fyldt i dem. Ved at bruge PAPER-ordren kan du ændre nummeret på den inkpot, som bruges til baggrund. Med PEN-ordren kan du tilsvarende ændre nummeret på den inkpot, som bruges til teksten.

Fig. 8.6 viser PAPER og PEN i funktion. En ordre som PAPER 2 får ikke farven til at skifte uden videre. Hvis vi skriver på skærmen efter ordren PAPER 2, så vil baggrunden for teksten være højcyan, men kun der hvor vi skriver. Hvis vi skal have hele skærmen til at få den nye farve, skal vi efterfølge PAPER-ordren med en CLS-ordre. Det kan du se i linie 30. Den anden halvdel af programmet bruger PAPER 0, som er blå, for det er nemlig farven i inkpot 0, og der skrives med forskellige PEN-farver. Du kan ud af programmet se, at hvis du vil have din tekst til at stå klart, så skal du bruge farver med stor indbyrdes kontrast. Farver, der er næsten lige lyse giver ikke ordentlig

```
10 BORDER 0
20 FOR n=0 TO 3
30 PAPER n:CLS
40 GOSUB 150
50 NEXT
60 PAPER 0:CLS
70 FOR n=0 TO 3
80 PEN n
90 GOSUB 180
100 GOSUB 150
110 NEXT
120 GOSUB 150
130 PEN 1
140 END
150 start=TIME
160 WHILE TIME < start+600:WEND
170 RETURN
180 PRINT"Her er lidt tekst til at vise
PEN's";"forskellige farver";n
190 RETURN
```

Fig. 8.6 Brug af PAPER og PEN ordrene.

kontrast, og giver derfor ikke en tydelig skrift. Selv foretrækker jeg at have en mørk baggrund og en lys skrift, det modsatte – mørke bogstaver på en lys baggrund – har tendens til at flimre. Du skal ikke forvente at få bogstaverne til at stå særlig klart på TV-apparatet, for farve-TV'er er ikke særlig gode til at gengive små detaljer. Dertil kommer at op mod 90% af den mandlige befolkning faktisk er delvis farveblinde, og at de flotteste farver fås, hvor der bruges store flader med kraftige farver – og helst på en rigtig monitor.

Når du kører programmet, vil du opdage at teksten for PEN 0 ikke kommer frem på skærmen. Forklaringen er den enkle, at inkpot 0 netop er den farve, der bruges til baggrund. Hvis du ændrer baggrundsfarven, inden du kører programmet vil PEN 0 teksten komme frem, men en af de andre vil mangle i stedet for. Jeg har brugt MODE 1 til demonstrationen her, fordi det er meget passende at have fire inkpots til rådighed. Husk imidlertid, at du har to inkpots at vælge imellem i MODE 2 og hele 16 at vælge imellem i MODE 0.

PEN og PAPER ordrene kan bruges til at bestemme hvilke farver, der skal bruges i vinduerne. Fig. 8.7 viser et eksempel på det, – og viser samtidig, at det ikke er alle farvekombinationer, der er lige gode. Når du bruger forskellige PEN- og PAPER-farver i vinduerne, så se lige grundigt på farverne først, og find ud af om de passer sammen. Nogle gange kan to nuancer af den samme farve passe godt sammen, men to lyse farver eller to mørke farver

```
10 BORDER 0
20 WINDOW#1,1,40,3,5
30 WINDOW#2,1,40,7,10
40 WINDOW#3,1,19,11,24
50 WINDOW#4,20,40,11,24
60 a$="Denne sætning viser vinduerne."
70 PAPER#1,0
80 PAPER#2,1
90 PAPER#3,2
100 PAPER#4,3
110 PEN#1,3
120 PEN#2,2
130 PEN#3,1
140 PEN#4,0
150 CLS
160 FOR n=1 TO 4
170 PRINT#n,a$:NEXT
```

Fig. 8.7 PAPER og PEN brugt i forskellige vinduer.

virker aldrig tilfredsstillende, især ikke på et almindeligt farve-TV. Og mange kombinationer kan ikke engang bruges på en farvemonitor.

Pænere udskrifter

Nu skal vi se lidt på de forskellige effekter, der kan laves med teksterne på skærmen, og vi kan lige så godt fortsætte med INK. Se på programmet i fig. 8.8, hvor en blinkende tekst vises. Blinkene fås ved at bruge ink, der skifter farve i inkpot 2 og 3 i linie 20 og 30. Når vi »dypper« vores PEN i disse inkpots, bliver resultatet at vi skriver med en blinkende skrift! – det kan du se når linie 50 og 80 køres. I linie 90 skifter vi tilbage til PEN 1, som er den normale inkpot til tekst, så undgår vi at »Ready« ikke også blinker. Bemærk at bogstaverne bliver ved med at blinke så længe de er på skærmen. Du kan ændre de blinkende farver straks, eller skifte til en enkelt farve ved at bruge INK. I fig. 8.9 har vi et eksempel, hvor titlen blinker i nogle sekunder, og derefter ændres til en enkelt farve. Det er en god måde at sikre sig at opmærksomheden rettes mod noget bestemt, uden at det hele tiden vil distrahere brugeren.

```
10 CLS
20 INK 2,6,8
30 INK 3,12,19
40 PEN 2
50 PRINT:PRINT"PAS PÅ! PAS PÅ!"
60 PRINT:PRINT:PRINT
70 PEN 3
80 PRINT"ADVARSEL - FARE!!"
90 PEN 1
```

Fig. 8.8 Teksten blinker fordi vi bruger en inkpot, hvor farven skifter.

```
10 INK 2,24,6
20 CLS
30 PRINT:PEN 2
40 PRINT TAB(12)"BLINKENDE TITEL"
50 FOR N=1 TO 3000:NEXT
60 INK 2,20
70 PEN 1
80 PRINT:PRINT:PRINT"Nu kan vi udskrive
en tekst på skærmen,";"- uden at vi bliv
er distraheret!"
```

Fig. 8.9 Her skiftes fra blinkende to-farvet skrift til en enkelt farve.

Amstrad kan også lave effekter som understregning, lave accenter osv. Vi kan lave disse effekter (og en masse andre, som vi vender tilbage til senere), ved hjælp af ASCII-koderne 1-31. De kan ikke give udskrifter men de kan styre udskrifterne, vi kalder dem for *kontrol-karakterer*. Vi har set lidt på dem før, – kode 21 afbryder datastrømmen til skærmen, kode 6 tilslutter skærmen igen, og kode 8 rykker markøren et trin baglæns. For at kunne lave understregninger, som vist i fig. 8.10, må vi ændre den måde, som udskrifterne laves på. Når vi normalt skriver, erstatter den nye udskrift helt den gamle.

```
10 CLS
20 PRINT:PRINT:PRINT
30 PRINT"DETTE ER VIGTIGT";
40 S$=STRING$(18,8)
50 PRINT S$
60 PRINT CHR$(22)+CHR$(1);
70 PRINT"_____ __ _______"
80 PRINT:PRINT
90 PRINT CHR$(22)+CHR$(0)
```

Fig. 8.10 Understregning, hvor CHR$(22) bruges til at forhindre at de nye karakterer sletter de gamle. Se også hvordan STRING$ kan bruges sammen med en ASCII-kode i stedet for en karakter i gåseøjne.

Ved at bruge CHR$(22)+CHR$(1), kan vi ændre dette, således at den nye tekst »lægges ovenpå« den gamle. I programmet udskrives en sætning i linie 30, og strengen s$ defineres som 18 trin baglæns. Dette vil flytte markøren tilbage til starten af sætningen. Så »beordrer« vi, at det næste skal lægges ovenpå teksten ved hjælp af CHR$(22)+CHR$(1), og vi får understregningen udskrevet. Nu bliver den gamle tekst ikke slettet – og vi får en nydelig understregning. Men hvis vi ikke husker at ophæve denne udskriftsmåde ville vi få nogle sære ting at se senere, så derfor bruges CHR$(22)+CHR$(0), for at få den normale udskriftsmåde tilbage igen. De fleste af denne slags ordrer kræver to (eller flere) CHR$-funktioner – som fx her. Og de kædes sammen som vist. Mange af disse kontrolkarakterer udfører det samme som BASIC-ordrene, og årsagen til at de bruges er, at de kan udskrives som tal. Hvis du har prøvet en BBC-computer, kan du måske se at dette svarer til BBC's »VDU« ordrer.

Forskellige detaljer

Der findes en masse ordrer i Amstrad's BASIC, som ikke rigtig hører hjemme sammen med andre ordrer, men som alligevel er yderst nyttige. Vi vil her

```
10 CLS:PRINT:PRINT TAB(10)"UPPER-LOWER"
20 PRINT:PRINT
30 FOR n=1 TO 5
40 INPUT "navn - ";a$
50 PRINT UPPER$(a$), LOWER$(a$)
60 NEXT
```

Fig. 8.11 Tegnskrift med UPPER$ and LOWER$.

til sidst se på nogle af dem. Tag fx LOWER$ og UPPER$ *) (fig. 8.11). Hver af disse skal efterfølges med navnet på en streng-variabel i parentes. Når du bruger UPPER$ (»øvre$«) bliver alle bogstaverne i strengen lavet om til store bogstaver. Når du bruger LOWER$ (»nedre$«) bliver alle bogstaver lavet om til små bogstaver. Det er en meget nyttig funktion at bruge, når du har indlæsninger med store og små bogstaver, blandet imellem hinanden. Og hvad skal det så bruges til? – Ja, fx når du skal have stillet ord i alfabetisk rækkefølge. Som du ved sker den slags ved hjælp af ASCII-koderne. ASCII-koderne for de små bogstaver er større end koderne for store bogstaver, – det betyder at computeren fx vil sætte ordene AND, abe, BID og bad i rækkefølgen AND, BID, abe, bad. Her vil du jo nok foretrække rækkefølgen abe, AND, bad, BID i stedet for. Og ved at skrive alle ordene med enten store eller små bogstaver, kan vi få computeren til at lave den rigtige alfabetiske orden. Når vi nu er i gang med streng-funktionerne, så er der en meget nyttig ordre, der hedder FRE. Hvis du skriver PRINT FRE(0) – eller har ordren i et program – så fortæller computeren hvor meget plads, der er tilbage i dens hukommelse. Jeg ville ønske at min anden computer kunne det samme! Hvis du bruger FRE(""), vil computeren »rydde op« i lagringen af strenge i hukommelsen. Det er vigtigt, for når et program, der arbejder med en masse strenge, kører bliver hukommelsen fyldt med ubrugte pladser, hvor der har været strenge, som er blevet byttet rundt. FRE("") kan frigøre denne plads, og bruges ordren inde i et langt sorteringsprogram, kan programmet køre hurtigere og mere effektivt.

Hvis du vil have udskrevet mellemrum, er der en funktion, der hedder SPACE$, som kan bruges til det. Fx vil s$ = SPACE$ (40) lave en linie med 40 mellemrum når du skriver PRINT s$. Det er lettere end at lave en løkke, som skal fylde en streng med mellemrum. Ordren WRITE er også god at bruge af og til. Hvis du skriver WRITE "Papir", så bliver *gåseøjnene skrevet ud sammen med ordet!* Det samme sker hvis du bruger a$="Papir" og så

*) »Upper case« svarer til de tegn man får frem når SHIFT er trykket ned, »lower case« svarer til de tegn man får frem når SHIFT ikke bruges – deraf navnet på disse ordrer. O.A.

bruger WRITE a$. Den kan blandt andet bruges hvis du vil have gåseøjne til at optræde i en sætning, for det kan du jo ikke gøre når du bruger PRINT. Og lige en mere. Tasterne på computeren repeterer (gentager) når du holder dem trykket ned. Til de fleste formål er standard repetitions-hastigheden meget passende, men til nogle formål kan en anden hastighed være bedre. Fx kan hastigheden være for langsom til visse spil, og måske for hurtig til visse dataprogrammer. I nogle tilfælde er det muligvis at foretrække, at maskinen har en så lav repetitionstakt, at den i realiteten slet ikke repeterer. Hvis du fx har to INKEY$ trin lige efter hinanden, så kan du komme til at vælge i begge trin på én gang, hvis du holder tasten trykket ned for længe, og på den måde få en forkert del af menuen. Som du nok har gættet så kan du selv bestemme repetitionshastigheden – det gøres med SPEED KEY, efterfulgt at to tal. Det første tal bestemmer hvor lang tid en tast kan holdes nede, før den begynder at repetere. Det andet tal bestemmer selve repetitionshastigheden. Du kan bruge tallene mellem 0 og 255, og hvis du bruger et lavt tal, kan du få mærkelige udskrifter, som fx mange dobbelte eller tredobbelte bogstaver. Prøv først at skrive med SPEED KEY 2,2, og prøv derefter at skrive med SPEED KEY 255,255 (hvis du kan!!). I det første tilfælde er det næsten umuligt at skrive normalt, i det andet repeteres der praktisk talt ikke.

Kapitel 9

Vi starter på grafikken

Det forventes at enhver moderne computer, at den har en flot farvegrafik og en masse effekter. Det har Amstrad naturligvis også, og i dette kapitel vil vi se på noget af den grafik Amstrad kan lave. Først må vi lige præsentere, hvad vi forstår ved de udtryk, der bruges her. Det første er *grafik*, og grafik er de billeder vi kan tegne på skærmen, og alle computere har nogle ordrer, der gør det muligt at tegne forskellige figurer og mønstre. I forbindelse med disse mønstre vil du se ordene »lav opløsning« og »høj opløsning«, »Opløsning« er ikke let at forklare. Forestil dig at du laver et billede på papir som er ca. 25 cm i bredden og 20 cm i højden – det svarer nogenlunde til størrelsen af en 14-tommer TV-skærm (der er ca. 14 tommer i diagonal).

Hvis du skal lave billedet ved at bruge små firkanter af farvet papir, så svarer det meget til den måde, computeren laver billeder på. Hvis du nu kun måtte bruge 936 stykker papir, (af en sådan størrelse at de præcis kunne dække skærmen), så ville du ikke kunne lave et særligt detaljeret billede med så store papirstykker, – og det er det der menes med lav opløsning. De fleste computere kan klare en bedre opløsning end dette. Hvis du på den anden side havde 32.000 små papirstykker som til sammen kunne dække skærmen, så ville du kunne lave et langt mere detaljeret billede. Det er det der menes med høj opløsning. Amstrad har høj-opløsnings grafik-ordrer til sin rådighed, og de 32.000 stykker papir, som jeg brugte som sammenligning, svarer til Amstrads billede i dens *laveste opløsnings-mode*, mode 0. Den højeste opløsning Amstrad har, er mode 2 med 128.000 små stykker til at bygge skærmbilledet op med. For at lave grafik skal computeren bruge en del hukommelse. En stor del af alle hjemmecomputere bruger den samme hukommelse til dette formål, som brugeren anvender til programmerne. Derfor har mange computere, der siges at have 64K eller 32K hukommelse (RAM), i virkeligheden ikke meget plads tilbage til brugeren – måske helt ned til 6K! Det er fordi så stor en del af hukommelsen bruges til at styre grafikken og de andre operativ systemer. Amstrad »stjæler« ikke hukommelsesplads på den måde. I stedet er en bestemt del af hukommelsen reserveret til grafik, og der

er derfor 43533 bytes (42.5K) programhukommelse tilbage, som ikke påvirkes af grafikken.

Indtil nu har vi arbejdet med det der hedder »tekstskærmen«, som er den normale måde at bruge skærmen på. Tekstskærmen kan kun bruges til tekst – dvs karakterer vi kan udskrive med PRINT. Som vi skal se bruges der andre ordrer til »grafikskærmen«, og dem ser vi på senere.

Tastaturgrafikken

Vi kan få nogle grafiske figurer, som du kan se i fig. 9.1, frem direkte via tastaturet. Du skal bare trykke på CTRL samtidig. Disse figurer kan imidlertid *ikke* udskrives på samme måde som vi udskriver ord, altså ved hjælp af PRINT efterfulgt af gåseøjne, den grafiske figur, og derefter endnu et sæt gåseøjne. Hvis du vil bruge disse figurer i dine programmer, som understregninger, symboler ved menuer eller lignende, så skal du bruge en anden metode i stedet for. Denne anden metode er at bruge ASCII-koderne.

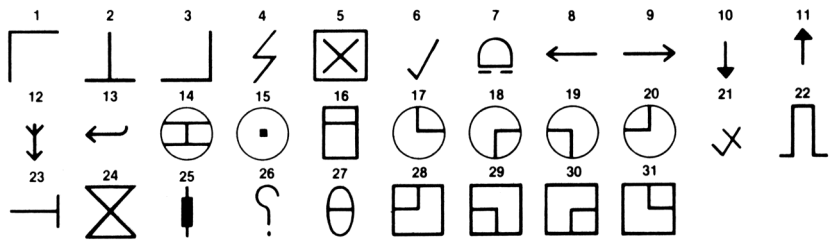

Fig. 9.1 Disse grafiske figurer kan fås frem via tastaturet. Tallene er deres ASCII-koder.

Karakter-koderne

Vi skal altså bruge figurernes karakter-koder, for at få adgang til dem i vores programmer. Nu findes der to sæt ASCII-koder, som kan lave grafiske figurer. Det ene sæt anvender ASCII-koderne 128 til 255. De er vist i Appendix III i manualen, men du kan også se dem ved at bruge programmet i fig. 9.2. Der er bl.a. en del bogstaver fra det græske alfabet i dette sæt, som kan være

```
10 CLS:n=127
20 FOR j=1 TO 128
30 PRINT CHR$(n+j);" ";
40 IF j/20=INT(j/20) THEN PRINT
50 NEXT
```

Fig. 9.2 Et lille program, der viser de grafiske karakterer for koderne 128 til 255.

nyttige til brug i matematik eller teknik. Programmet er bygget op sådan at figurerne bliver udskrevet adskilt fra hinanden, prøv om du kan finde ud af hvordan. Det andet sæt grafiske figurer fås med ASCII-koderne 1 til 31. Problemet er at disse koder også bruges til andre formål, og du må vide hvordan vi skifter fra den ene brug til den anden. Hemmeligheden er CHR$(1). Hvis du sætter CHR$(1) foran en af disse koder vil du få en figur udskrevet *uden* andre effekter. Fx ved du at markøren rykkes én plads tilbage når du benytter PRINT CHR$(8), men PRINT CHR$(1)+CHR$(8) bevirker i stedet at vi blot får en figur udskrevet. Programmet i fig. 9.3 viser disse figurer, – mange af dem er gode at bruge i undervisningsprogrammer og spil mm.

```
10 CLS
20 FOR n=1 TO 31
30 PRINT n;" ";CHR$(1)+CHR$(n);" ";
40 IF n/5=INT(n/5) THEN PRINT:PRINT
50 NEXT
```

Fig. 9.3 Et program der viser de grafiske karakterer for koderne 1 til 31. Bemærk at PRINT CHR$(x) – hvor x ligger mellem 1 og 31 – *ikke* kan kalde disse grafiske figurer frem.

Du kan lave nogle flotte mønstre med disse CHR$ figurer, hvis du bruger en af planerne over kolonne og linienumre i manualen. Planen for MODE 1 (Appendix VI, Page 2 i manualen), viser 40 felter på tværs af skærmen og 25 felter nedad, for det er antallet af placeringer for karaktererne på mode 1 skærmen. Hvis du bruger mode 0 skærmen, fylder karaktererne mere, og der er kun plads til 20 karakterer pr. linie i mode 0. I mode 2 er der plads til 80 karakterer pr. linie. I alle tre tilfælde er antallet af linier det samme – nemlig 25. Dvs. at karaktererne altid har samme højde, mens bredden varierer. Figurerne ser temmelig forskellige ud i de tre modes. Mode 0 gengiver karaktererne i »Cinemascope«, og mode 2 viser nogle ret afmagrede karakterer! Hver firkant i planerne svarer til placeringen af en karakter, og hvis du lægger et stykke kalkerpapir over en af planerne, kan du tegne noget du senere kan overføre til skærmen. Der er forskellige måder at programmere den slags på. En mulighed er at udskrive én linie med de grafiske karakterer ad gangen. En anden mulighed er at lave udskriften med en løkke, hvor ASCII-koderne står i en DATA linie. En tredje mulighed er at indsætte alle karaktererne i en streng, ligesom du kan indtaste bogstaver i en streng.

Ja, lad os se nogle eksempler på det her. I fig. 9.4 kan du se hvordan et »logo« for fx et firma kunne udformes. Det består af bogstavet »S« med linier der »stråler« ud til alle sider. Bogstavet »S« er let nok, det er CHR$(83) (se fig. 5.1b), – men de andre karakterer findes også i ASCII-koden – se oversig-

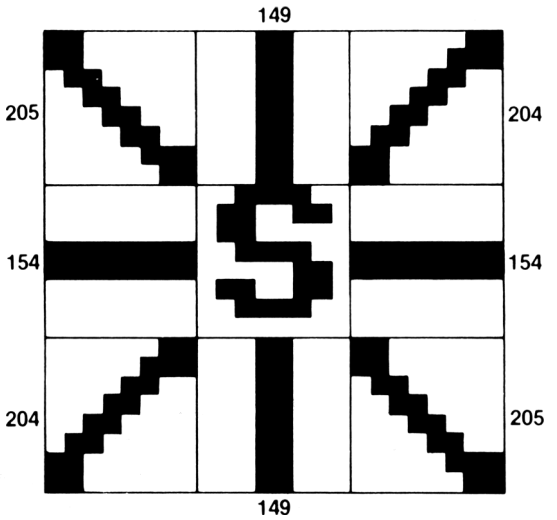

Fig. 9.4 Her er de grafiske karakterer brugt til at udforme et »logo«.

```
10 PAPER 3:PEN 2
20 CLS:PRINT:PRINT
30 PRINT TAB(5) CHR$(205)+CHR$(149)+CHR$
(204)
40 PRINT TAB(5) CHR$(154)+CHR$(83)+CHR$(
154)
50 PRINT TAB(5) CHR$(204)+CHR$(149)+CHR$
(205)
60 PRINT:PRINT
```

Fig. 9.5 Et meget enkelt program, der udskriver logoet i fig. 9.4

ten i manualen (Appendix III). Logoet er blevet lavet ved at kalkere de enkelte karakterer af fra manualen. Nu kan du simpelt hen skrive et program, der udskriver CHR$ værdierne på de rigtige steder – som det er gjort i programmet i fig. 9.5. Det fungerer, men det er en ret klodset måde at programmere på. Se nu på fig. 9.6. Det ser måske ikke mere elegant ud – der bruges jo flere programlinier – men det er alligevel bedre. Der er kun én PRINT CHR$(d%) ordre, mens det foregående program havde hele tre linier af den slags. Der bruges to løkker, en for hver linie karakterer, og en anden for hver kolonne. Alle variabler er heltalvariabler, således at programmet kører hurtigt. Derudover bliver logoets placering på skærmen bestemt af en LOCATE-ordre, der også står i en løkke. Fordelen ved denne metode er,

```
10 PAPER 3:PEN 2:CLS
20 FOR j%=1 TO 3:LOCATE 18,j%+10
30 FOR n%=1 TO 3
40 READ d%:PRINT CHR$(d%);
50 NEXT:PRINT:NEXT
60 WINDOW#2,1,40,24,25
70 WINDOW SWAP 0,2
100 DATA 205,149,204
110 DATA 154,83,154
120 DATA 204,149,205
```

Fig. 9.6 Her bruges løkker til at læse og udskrive de grafiske karakterer.

at det er let at se de data, der anvendes, og det er dermed også meget let at ændre data, uden at skulle ændre programmet. Bemærk at jeg har brugt et vindue her, som byttes rundt med #0, for at få »Ready« og markøren til at komme frem nederst på skærmen – fremfor lige under logoet. Du skal bruge WINDOW SWAP 0,2, hvis du vil anvende hele skærmen igen, når programmet er slut.

Fig. 9.7 viser en endnu bedre metode. Programmet starter med at definere strengen b$. Den består af tre karakterer med koden 8 (markøren til venstre), efterfulgt af kode 10. Hvis du ser efter i manualen, kan du se at 10 er koden for »markøren ned«. Når vi kører b$, sker der derfor det, at markøren rykker tre pladser til venstre og en plads ned. I linie 20 sættes strengen gr$ lig en tom karakter. Det næste der sker er, at der startes to løkker, en til linierne og en til kolonnerne. Efter data er blevet læst og kædet sammen med gr$ i linie 40, tilføjes b$. Det får markøren til at rykke en linie ned og tre pladser til venstre. Alt i alt bliver effekten, at udskrive tre karakterer, og markøren rykker tilbage til den rigtige plads i den næste linie. Hver linie føjes til strengen gr$, og den udskrives så i linie 60.

Den store fordel ved denne metode kan ses i fig. 9.7, hvor logoet kan udskrives hvor som helst på skærmen, uden at der skal ske nogle ændringer i programmet. PRINT gr$ vil udskrive logoet, der hvor markøren befinder sig.

```
10 PAPER 2:PEN 3:CLS
20 gr$="":b$=STRING$(3,8)+CHR$(10)
30 FOR j%=1 TO 3:FOR n%=1 TO 3
40 READ d%:gr$=gr$+CHR$(d%):NEXT
50 gr$=gr$+b$:NEXT
60 LOCATE 18,12:PRINT gr$
70 DATA 205,149,204,154,83,154,204,149,2
```

Fig. 9.7 De grafiske karakterer samles i en enkelt streng, som kan udskrives med én ordre.

Du skal selvfølgelig være sikker på at markøren ikke er for tæt på højre side eller for tæt på underkanten. Når vi nu har lært at lave mønstre, kan vi gå i gang med at lave animation (bevægelse). Det er vist i fig. 9.8. Metoden er simpelt hen at udskrive vores logo på en placering, bestemt af LOCATE. Logoet står til venstre et øjeblik og slettes så. Efter endnu et kort øjeblik er LOCATE placeringen flyttet en kolonne til højre, og proceduren gentages. Hvis den lille pause er meget kort, som i dette eksempel, og bevægelsen er kort, som den *ikke* er her, så kan dette se ud som en bevægelse. Men tekstskærmen er ikke særlig velegnet til at lave animation – dertil er afstandene mellem karaktererne for store.

```
10 PAPER 2:PEN 3:CLS
20 gr$="":b$=STRING$(3,8)+CHR$(10)
30 FOR j%=1 TO 3:FOR n%=1 TO 3
40 READ d%:gr$=gr$+CHR$(d%):NEXT
50 gr$=gr$+b$:NEXT
60 a$=STRING$(3,32):sp$="":sp$=a$+b$+a$+
b$+a$
70 FOR n=1 TO 18
80 LOCATE n,12
90 PRINT gr$:GOSUB 1000
100 LOCATE n,12
110 PRINT sp$:GOSUB 1000
120 NEXT:LOCATE n,12:PRINT gr$
130 DATA 205,149,204,154,83,154,204,149,
205
140 END
1000 FOR k=1 TO 20:NEXT:RETURN
```

Fig. 9.8 Simpel animation på tekstskærmen.

Lav dine egne karakterer!

På Amstrad kan der laves grafik på tekstskærmen, på en meget spændende måde, hvor vi ved brug af PRINT kan udskrive vores egne små »hjemmelavede« figurer. Man kunne godt kalde figurerne for »lav-opløsnings grafik«, for der er ganske vist kun et begrænset antal PRINT-placeringer på skærmen, men de giver alligevel muligheder for ret flotte effekter. Som du kan forstå af overskriften, kan vi selv udforme vores egne karakterer. Der er to led i denne proces – dels at udforme karaktererne og dels at få dem frem på skærmen. Lad os se på det første.

Vi starter logisk nok med planlægningen. Vi kan lave figurer af samme størrelse som de øvrige skærmkarakterer, – altså som markøren fx. Markøren og alle de andre karakterer er dannet (af op til) 8×8 punkter – i alt 64 punkter. Fig. 9.9 viser det »symbolnet«, karaktererne udformes ud fra. Du kan tegne det af på et stykke kvadreret papir, når du skal bruge det. Alle de små firkanter repræsenterer skærmpunkter, der enten kan være PEN eller PAPER farvede, alt afhængigt af hvilke talkoder vi giver computeren. Når en karakter bliver udformet i symbolnettet bruger vi normalt kun 7 felter på hver led, sådan at den venstre kolonne og den nederste linie ikke anvendes. På den måde kan vi skabe mellemrum mellem karaktererne mellem tekstlinierne. Men når du laver dine egne karakterer, kan du godt bruge 64 felter, sådan at de kan hænge sammen på skærmen.

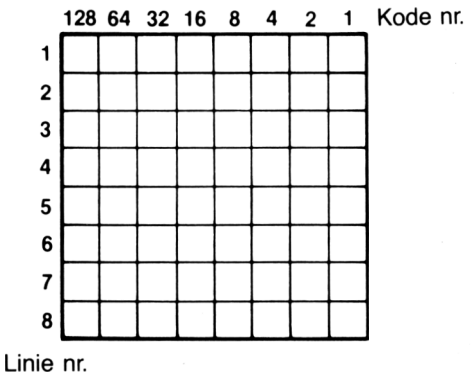

Fig. 9.9 Symbolnettet. En plan over de 8×8 punkter, der danner alle karaktererne. Ved hjælp af tallene kan du selv lave nye karakterer.

Manualen går meget let hen over brugen af symbolnettet, så hvis du ikke har prøvet den slags før, kan det godt være svært at finde ud af. Det er de tal der står over hver kolonne i fig. 9.9, du skal bruge. Hvert tal er en kode for hvert af de 8 felter i kolonnen nedenunder. Hvis tallet bruges bliver feltet PEN farvet, hvis du bruger 0 bliver det PAPER farvet – altså usynligt. Her kan et eksempel gøre det mere forståeligt. Se på fig. 9.10, hvor jeg har lavet en »rummand« for at vise princippet.

I den første linie er der fire sorte felter. Jeg lægger normalt kalkerpapir over symbolnettet, men i dette eksempel har jeg tegnet nettet med. De mørke felter i den øverste linie er dem, der skal have PEN farve, og de står under talkoderne 32, 16, 8 og 4 (i fig. 9.9). Der er ikke andre mørke felter i denne linie, så vi skal bare lægge de fire tal sammen, det giver i alt 60, som vi noterer ude til højre. Vi bruger samme fremgangsmåde i anden linie, hvor

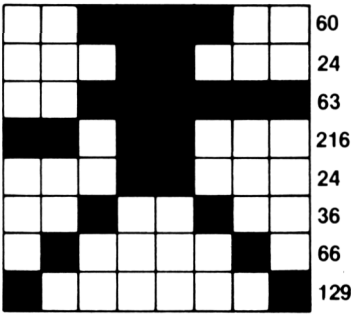

Fig. 9.10 Sådan bruges symbolnettet – til at lave en lille »rummand«.

felterne under 16 og 8 er mørke, og her bliver summen 24. Og sådan fortsætter vi hele vejen ned.

Der er lige to bemærkninger til dette. Den første er at en linie uden mørke felter får koden nul. Den anden er at du kan spare en masse regnearbejde, ved at huske på at en helt fyldt linie har en samlet kode på 255. Det er det største tal, du kan bruge som kode i en tekstkarakter. Du ender altid med at have otte tal ligegyldigt hvilken figur, du prøver at lave.

Så skal vi se på hvordan vi kan få computeren til at udskrive vores nye karakter. Vi skal have lagret talkoderne i maskinens hukommelse sammen med den ASCII-kode, som vi skal bruge til at kalde karakteren frem på skærmen med. Her bruger vi en ny ordre, der hedder SYMBOL. SYMBOL skal efterfølges af *ni* tal. Det første tal er den ASCII-kode, som vi skal bruge. På den måde kan vi, ved at trykke på en tast eller ved at bruge ASCII-koden, kalde vores figur frem på skærmen. De næste otte tal i SYMBOL er blot de koder vi allerede har udregnet. Alle tallene skal adskilles af kommaer, og der skal være mellemrum mellem »L« i SYMBOL og det første tal. Talkoderne læses ovenfra og ned. Med SYMBOL indlæser vi alle ni talkoder i hukommelsen.

Det ser jo godt nok ud – men hvilke ASCII-koder skal vi anvende? Svaret er at ASCII-koderne fra 240 til 255, i alt 16 koder, kan bruges. Tolv af disse koder benyttes normalt af markørpilene, oppe til højre på tastaturet – der er tolv fordi de også kan bruges med SHIFT og CTRL. Men selvom vi bruger disse koder til egne formål, *så påvirker det ikke tasternes funktion*, så du får *ikke* en rummand frem på skærmen, fordi du trykker på en markørtaste! Lad os sige at vi vil have vores lille rummand til at komme på skærmen, når vi bruger CHR$(244). Fig. 9.11 viser det lille program vi skal bruge til formålet. ASCII-koden er 244, og derefter kommer de otte koder fra fig. 9.10. Vi kan når som helst kalde rummanden frem på skærmen med PRINT CHR$(244).

```
10 CLS
20 SYMBOL 244,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12)CHR$(244)
```

Fig. 9.11 Sådan kan vi få rummanden på skærmen.

Vi behøver ikke at begrænse os til at benytte ASCII-koderne 240 til 255. Vi kan starte helt nede fra 32 og frem. Det vil sige at vi kan omdefinere *alle* tasternes koder, og på den måde få figuren frem, blot ved at trykke en almindelig tast ned. I fig. 9.12 kan du se hvordan det gøres. I linie 10 har vi nu indsat SYMBOL AFTER 90. Det betyder at vi kan omdefinere alle karakterer, der har en ASCII-kode større end 90. Vi har her valgt 120, som er ASCII-koden for »x«. Når du kører programmet, vil du se at rummanden bliver udskrevet af ordren PRINT TAB(12);"x". Du kan selvfølgelig også bruge CHR$(120) hvis du vil. Eller – mere vigtigt – du kan få rummanden frem ved at trykke på x-tasten. For at få maskinen til at fungere normalt igen skal du trykke CTRL-SHIFT-ESC. Hvis du vil gemme programmet, så husk at optage det inden!

```
10 CLS:SYMBOL AFTER 90
20 SYMBOL 120,60,24,63,216,24,36,66,129
30 PRINT:PRINT TAB(12);"x"
```

Fig. 9.12 Koden for en tast omdefineres, således at den udskriver en rummand.

Her er lige en vigtig ting at bemærke. Ingen af de andre koder større end 90 bliver påvirket af dette, – kun den du har omdefineret. Men selvfølgelig kan du omdefinere alle de andre koder også, hvis du vil. Men det er jo et stort arbejde. Prøv for sjov at se på fig. 9.13, som omdefinerer alle taster til at lave rummænd. Når programmet er kørt bliver alt på skærmen til rummænd – bortset fra markøren. Selv meldinger som »Ready« og »Syntax error« bliver til små rækker rummænd! Det giver en vis sikkerhed for at der ikke er nogen der roder med din computer, hvis du forlader den et øjeblik! Du skal igen bruge CTRL-SHIFT-ESC til at »rense« maskinen – og husk at programmet også bliver slettet.

```
10 CLS:SYMBOL AFTER 32
20 FOR n=32 TO 127
30 SYMBOL n,60,24,63,216,24,36,66,129
40 NEXT
```

Fig. 9.13 Sådan kan alle taster omdefineres – det gør det meget vanskeligt at bruge computeren. Brug CTRL-SHIFT-ESC for at »rense« maskinen, men optag programmet først.

Du behøver ikke at nøjes med at lave en enkelt karakter på denne måde, – du kan lave flere, der i sammenhæng udgør større figurer! Fig. 9.14 viser et stort symbolnet, hvor der er plads til at lave en figur, der består af ni sammenhængende karakterer. Lad os fx tænke os at vi laver nye karakterer til koderne 240 til 242. Vi kunne så sætte karaktererne sammen ind i en streng gr$, således at ordren PRINT gr$ kunne udskrive hele figuren. Fig. 9.15 viser en slange, der er tegnet på tre sammenhængende symbolnet. Fig. 9.16 viser et program, der kan få slangen til at bevæge sig. Ellers behøver jeg ikke at forklare så meget. Jeg har brugt de tre karakterkoder 240 til 242, og hver af dem tildeles en del af slangen. Da hele symbolnettet på 8*8 er blevet brugt, hænger slangens tre dele sammen på skærmen. Billedet på skærmen fås ved at sætte karaktererne ind i en streng og derefter udskrive strengen. Det er ret enkelt, det kræver lidt arbejde og det ser ret godt ud!

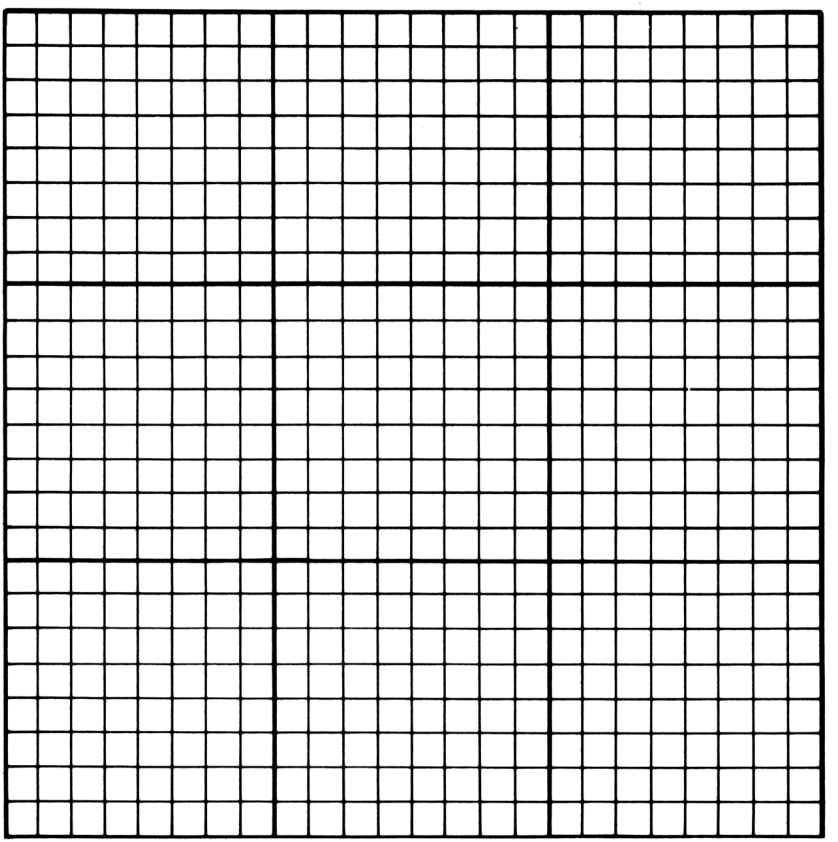

Fig. 9.14 Et stort symbolnet, hvor der kan tegnes flere sammenhængende karakterer.

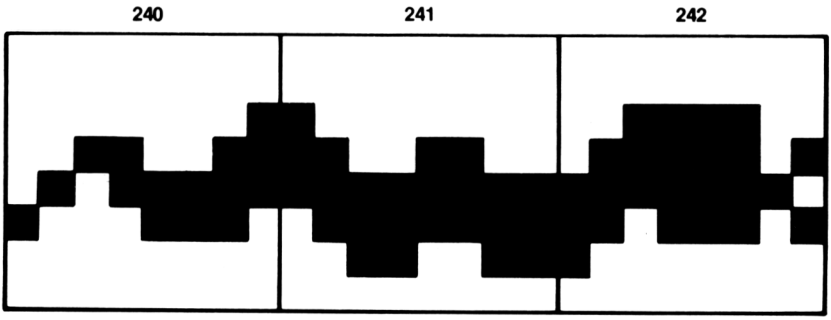

Fig. 9.15 En slange tegnet på tre sammenhængende symbolnet.

```
10 CLS
20 SYMBOL 240,0,0,1,51,95,142,0,0
30 SYMBOL 241,0,0,128,204,255,127,51,0
40 SYMBOL 242,0,0,60,125,254,221,128,0
50 gr$=CHR$(240)+CHR$(241)+CHR$(242)
60 x=2:y=12
70 EVERY 10 GOSUB 1000
80 FOR n=1 TO 25:LOCATE 5,n:PRINT"ANIMER
ET SLANGE"
90 FOR j=1 TO 1000:NEXT:NEXT
100 GOTO 100
110 END
1000 LOCATE x-1,y:PRINT" ":LOCATE x,y
1010 PRINT gr$;
1020 x=x+1
1030 IF x>38 THEN LOCATE x-1,y: PRINT CH
R$(18);:x=2
1040 RETURN
```

Fig. 9.16 Slangen animeres med EVERY.

Der er imidlertid også en helt ny ordre i programmet, – EVERY. Det er en ordre, der er meget vigtig når vi laver animation, og vi vil se grundigt på den, og andre af samme slags, senere. EVERY gør det muligt at bruge en af Amstrad'ens timere – der er fire i alt. Ideen med EVERY er, at vi kan få noget til at ske meget hyppigt. Hvor hyppigt afhænger af det tal, der kommer efter EVERY. Med tallet 50 sker det én gang i sekundet. Med dette faste interval kan vi fx få en subrutine til at starte. Det vigtige er her, at subrutinen vil køre med samme interval igen og igen, – *også selv om computeren er i gang med andre opgaver!* Du kan se det i eksemplet, at løkken kører og

udskriver ordene »ANIMERET SLANGE«, samtidig med at slangen bevæger sig hen over skærmen. Når løkken stopper, kører den endeløse løkke i linie 100 videre. Og derfor kører programmet, og dermed timeren videre. Timeren arbejder uafhængigt af programmets andre dele, og den kan afbryde programmet for at udføre subrutinen med det interval, der er fastsat. I eksemplet er intervallet 10 – en femtedel sekund. En form, der kan bevæges uafhængigt af programmets andre dele kaldes en *sprite*. De fleste nyere maskiner i dag har sprites, men Amstrad er en af de maskiner, der giver den enkleste og mest effektive kontrol af dem.

Da der er fire timere, kan du have mindst fire forskellige genstande i uafhængig bevægelse – du kan selvfølgelig lade hver timer kontrollere så mange sprites du ønsker. For at fastlægge hvilken timer, du vil bruge, skal timerens nummer, 0 til 3, sættes efter interval-tallet med et komma imellem. Du kan fx have ordrer som EVERY 10,1 GOSUB 1000 eller EVERY 20,3 GOSUB 2000. Og nu da vi er ved bevægelser på tekstskærmen, så har Amstrad en måde at finde markørens placering på. Her bruges funktionerne POS og VPOS, og hver af dem *skal* knyttes til et vindue-nummer. Når det er hele skærmen er det jo #0, men det må ikke udelades. Fig. 9.17 viser et lille eksempel på, hvordan POS og VPOS virker. POS angiver markørens horisontale POSition, og VPOS angiver markørens Vertikale POSition. Tallene der bruges er de samme som kendes fra LOCATE. POS og VPOS er specielt anvendelige, når vi arbejder med vinduer, fordi det ikke altid er til at hitte ud af hvor markøren er henne. Du kan bruge tallene fra POS og VPOS i alle de tests du kan finde på.

```
10 CLS
20 FOR y=1 TO 25
30 PRINT TAB(RND(1)*35);POS(#0);",";VPOS
(#0)
40 FOR n=1 TO 1000:NEXT
50 NEXT
```

Fig. 9.17 POS og VPOS bruges til at finde markørens position.

Kapitel 10

Rigtig grafik

Amstrad har en betydelig mere avanceret grafik end de fleste andre hjemmecomputere. I dette kapitel skal vi se på ordrer, der specielt er lavet til grafik, dvs at vi kan få figurer frem på skærmen uden at bruge ordrer som fx PRINT og CHR$. Det første vi skal lægge mærke til er, at Amstrad's grafik har en meget høj opløsningsgrad. Hvis du ser nærmere på skærmen på et farve-TV, kan du se, at billedet består af en række linier eller punkter. Disse linier eller punkter udsender lys, og man kan ikke få noget frem på skærmen, der er mindre end tykkelsen af en linie eller et punkt. Faktisk er det kun en farvemonitor, der kan gengive så små detaljer. De »punkter« computeren arbejder med kaldes *pixels (pic*-ture *el*-ement-*s* (dvs. »billed-bestanddele«)). Den størrelse pixels, computeren kan styre, svarer til størrelsen af adskillige punkter på TV-skærmen. I mode 0 er den grafiske opløsning 160 pixels vandret og 200 lodret. For de to andre modes er opløsningen lodret den samme, men mode 1 har 320 pixels vandret og mode 2 har 640. Den højeste opløsning bliver ikke særlig pæn på et farve-TV, her skal du bruge en monitor, for at få en virkelig pæn grafik. Men opløsningen i mode 2 er altså meget høj, og prisen for det er, at der kun kan bruges to farver.

Muligheden for at give hver af de 64.000 pixels farver, ville ikke være meget bevendt, hvis vi skulle lave billeder ved at definere placering og farve for hver enkelt pixel for sig. På nogle hjemmecomputere er det den eneste måde at lave højopløsningsgrafik på, men her er Amstrad mere alsidig. Ja, faktisk er der en hel række grafiske ordrer, der gør det noget lettere at »tegne« på skærmen. Alle disse ordrer benytter sig af x-y koordinatsystemer, som vi allerede har set på i forbindelse med LOCATE. Forskellen er at der skal bruges andre tal. I alle tre grafiske modes, skal x ligge mellem 0 og 399. Det er meget bekvemt, idet du så kan lave et grafisk program i én mode og omsætte det til en anden mode, uden at skulle gå programmet igennem for at ændre alle tallene. Fig. 10.1 viser hvordan du kan tegne et koordinatsystem til grafik ved hjælp af milimeterpapir.

Der er nogle væsentlige forskelle mellem at arbejde med tekst og med

1. Tag et ark A4 milimeterpapir. Du kan få det hos boghandleren.
2. Sæt enheder på akserne – 0, 25 ... 600 på langs og 0, 25 ... 400 på tværs.
3. Brug kalkerpapir, med dette koordinatsystem under, når du skal lave skitser og finde koordinater.

Fig. 10.1 Sådan kan du lave et koordinatsystem til grafikken.

skærmgrafik. Når du bruger grafiske ordrer, så er det de enkelte pixels du styrer, således at afstanden mellem enkelte x-y koordinater er meget mindre. Og dertil kommer at grafik-systemet har et andet begyndelsespunkt. På tekstskærmen kan nul ikke bruges – og LOCATE 1,1 er øverste venstre hjørne af skærmen. Begyndelsespunktet i grafikken er det punkt hvor x = 0 og y =0 (det skrives normalt (0,0)), og det er *nederst* i venstre side. Y-afstandene måles nedefra og op og x-afstandene måles fra venstre mod højre. Hvis du er vant til at arbejde med grafer og kurver, vil du have let ved at lære de grafiske ordrer. Og nu vi er ved grafer, så lad os starte med en ordre vi bruger til at tegne grafer med, PLOT.

Vi plotter

Hvis en computer skal kunne bruges til uddannelses- eller erhvervsprogrammer, skal den kunne tegne grafer. En graf er et sæt punkter, som forbindes og derved danner en figur. Amstrad kan plotte (»plotte« betyder at afsætte punkter i et koordinatsystem) grafer for dig med ordrerne PLOT og PLOTR. Punktets farve bliver den aktuelle INK-farve. Forskellen mellem de to ordrer er at PLOT bruger de *absolutte* koordinater, mens PLOTR bruger koordinaterne *relativt i forhold til markørens placering*. Fx er PLOT 0,0 punktet 0,0 – nederste venstre hjørne på skærmen. PLOTR 0,0 er derimod det sted hvor markøren befinder sig. Og grafik-markøren er ikke den samme som tekst-markøren. Grafik-markøren er usynlig, så du kan ikke se hvor den befinder sig, før du bruger en ordre, der giver nogle pixels en anden farve end baggrundsfarven. Når et grafikprogram er slut vender tekst-markøren tilbage igen.

Fig. 10.2 viser Amstrad'en plotte tre grafer samtidig – i et meget rimeligt tempo. X sættes i linie 30 således at skærmen bruges i fuld bredde, og for hver x-værdi udregnes de tre tilsvarende y-værdier. Den første fås af sinus af vinklen x/639, i radianer. Den anden fås af kvadratet af sinus af den samme vinkel, og den tredje fås af sinus af vinklen i tredje. Hvert punkt bliver plottet ind med PLOT-ordrerne i linie 40, 60 og 80 med tre forskellige INK-farver til de tre grafer. Grunden til at vi har valgt de viste værdier er, at vi vil

```
10 MODE 0
20 INK 0,0
30 FOR x=1 TO 639
40 y=200+200*SIN(2*PI*x/639)
50 PLOT x,y,1
60 y=200+200*(SIN(2*PI*x/639)^2)
70 PLOT x,y,2
80 y=200+200*(SIN(2*PI*x/639)^3)
90 PLOT x,y,3
100 NEXT
110 GOTO 110
```

Fig. 10.2 Programmet tegner tre grafer i forskellige farver i høj opløsning.

have graferne til at fylde skærmen ud. Sinus af en vinkel ligger mellem −1 og +1. Og da værdien 1 på y-aksen ikke ser ud af meget, ganger vi værdien med 200. Det får værdien til at svinge mellem −200 og +200. Men vi kan ikke plotte −200 på skærmen, så vi lægger 200 til alle y-værdierne, således at værdierne i stedet ligger mellem 0 og 400 – dvs. hele y-aksens område. Vi bruger 2*PI*x/639 som vinklen. På den måde passer værdierne med radiantallene. Disse vinkel-funktioner går gennem en række tal samtidig med at vinklen går fra nul radianer til 2*PI radianer. Når x = 0 er 2*PI*x/639 også nul, og vi får nul radianer til venstre på x-aksen. Når x = 639, til højre på x-aksen, så er vinklen 2*PI*639/639, som netop er de 2*PI radianer vi ønsker. Hvis du ikke kan lide at arbejde med vinkler i radianer, så skal du blot stille funktionerne om til grader med DEG – i en af linierne før vinkelfunktionerne bliver brugt, på den måde kan du bruge sinus med grader i stedet for. For at stille maskinen tilbage til radianer bruger du RAD, eller CTRL-SHIFT-ESC.

Se på graferne på skærmen. Du kan tydeligt se størrelsen af de enkelte pixels i mode 0 her. Du kan også se at Amstrad's »lave opløsningsgrad« er højere end visse computeres højeste opløsningsgrad! Prøv at ændre linie 10 til MODE 1 og senere til MODE 2, så du kan se hvor små pixels der bruges i de to andre modes. Det er sværere at se farverne på disse fine grafer, med mindre du bruger en monitor, men grafen ser mere jævn ud end i mode 0. I mode 2 er der kun to grafer. Hvorfor? Fordi vi kun kan have to farver i denne mode. Til de fleste formål er mode 1 et godt valg, her får du fire farver og en pæn opløsning.

Vi er endnu ikke færdige med at tegne grafer. Prøv den anden ændrede udgave af programmet i fig. 10.3. Her startes der med ORIGIN 0,200 (»begyndelsespunkt«). Det betyder at alle graferne nu starter i punktet 0,200 – dvs. midt på skærmen ude til venstre. Men begyndelsespunktet (»origin«) er

```
10 MODE 0:ORIGIN 0,200
20 INK 0,0
30 FOR x=1 TO 639
40 y=200*SIN(2*PI*x/639)
50 PLOT x,y,1
60 y=200*(SIN(2*PI*x/639)^2)
70 PLOT x,y,2
80 y=200*(SIN(2*PI*x/639)^3)
90 PLOT x,y,3
100 NEXT
110 GOTO 110
```

Fig. 10.3 ORIGIN bruges til at bestemme grafernes begyndelsespunkt. Det er den enkleste brug af ORIGIN.

altid det punkt du får med 0,0, så PLOT 0,0 vil nu være et punkt midt på skærmen i venstre side. Ved at kunne vælge begyndelsespunktet på denne måde, kan vi gøre selve programlinierne lidt enklere, idet vi ikke skal bruge »200+« mere. Nu kan vi fx skrive PLOT 50, – 50, fordi –50 betyder 50 pixels under midten, ligesom +50 betyder 50 pixels over midten. ORIGIN kan også bruges til at definere vindue-størrelser, – det er vist i manualen.

Vi laver linier

At plotte grafer er en meget vigtig del af højopløsningsgrafik, men der er grafiske ordrer, der kan meget mere. To af disse er MOVE (»flyt«) og DRAW (»tegn«). MOVE-ordren flytter – som navnet siger – den grafiske markør. Da grafikmarkøren er usynlig, kan du ikke se noget ske når du bruger MOVE. Det svarer til at bevæge en blyant – men uden at den rører papiret. DRAW er en ordre, der tegner en linie – med blyanten på papiret. Begge ordrer benytter sig af samme koordinater (x og y-tal) som PLOT. Med DRAW kan du bruge et tredje tal – »inkpot«-nummeret.

Men lad os se på et eksempel, fig. 10.4. Det er ikke særlig kompliceret, programmet tegner en firkant. Farven bliver »guld« på blå baggrund, hvis du lige har tændt maskinen, men sandsynligvis bliver farverne rød på sort, hvis du lige har lavet grafprogrammet. Årsagen er at hvis du ikke definerer inkpot-farver efter DRAW x,y, så bliver farven den samme, som blev brugt sidst med PLOT eller DRAW. De glemmer ikke noget, disse maskiner! Du vil se at maskinen tegner firkanten meget hurtigt, hurtigere end øjet kan registrere. Prøv at sammenligne det med den tid visse andre maskiner er om at lave en simpel tegning.

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAW 150,300
50 DRAW 450,300
60 DRAW 450,100
70 DRAW 150,100
80 GOTO 80
100 NEXT
110 GOTO 110
1000 MODE 1
1010 LIST
```

Fig. 10.4 Et tegneprogram, der laver en firkant ved hjælp af DRAW.

Nå ja, der er jo en ny ordre mere med. I linie 10 bruges ON BREAK GOSUB 1000. Det betyder at når du taster ESC to gange (som giver Break), så udføres subrutinen i linie 1000. Grunden til at jeg har taget den med her er, at programlistninger ikke er særlig rare at se på i mode 0. Subrutinen i linie 100 sætter maskinen tilbage til mode 1 og LISTer programmet. Da LIST *altid* efterfølges af et »Ready«, er det ikke nødvendigt at skrive RETURN til sidst – men det er nu ellers en god vane at gøre det. Når du taster ESC to gange for at bryde den endeløse løkke i linie 80, bliver skærmen stillet om til mode 1, og programmet bliver LISTet. Denne subrutine er god at bruge, når du er ved at få et program til at køre.

Nu vi har set på DRAW må vi hellere også se på DRAWR. Denne ordre er DRAW med *relative* koordinater. DRAWR 10,100 betyder fx »DRAW« en linie 10 pixels til højre og 100 pixels op. Det er *ikke* det samme som at tegne en linie til punktet 10,100, som DRAW ville gøre. Fig. 10.5 viser vores firkant tegnet med DRAWR. Hvis du vil flytte til højre eller op, så er afstandene positive, hvis du vil flytte ned eller til venstre er afstandene negative. Hvis du vil have en af koordinaterne, x eller y, til at være uændret, så skal tallet 0 bruges.

For lige at slå forskellen mellem DRAW og DRAWR fast, så viser programmet i fig. 10.6 nogle linier, der tegnes tilfældigt fra skærmens midte med DRAW (linie 60). Efter en kort pause ser vi en helt anden effekt, når vi bruger DRAWR. Denne gang bliver hver ny tilfældig linie forbundet med den foregående. I linie 1030 får a en værdi, som er enten −1 eller +1, hver linies retning er tilfældig, dens længde er også tilfældig, og den bestemmes i linie 1040. Der fremkommer et tilfældigt mønster af linier, som kan minde om

```
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAWR 0,200,2
50 DRAWR 300,0,2
60 DRAWR 0,-200
70 DRAWR -300,0
80 GOTO 80
100 NEXT
110 GOTO 110
1000 MODE 1
1010 LIST
```

Fig. 10.5 Her tegnes firkanten med DRAWR, bemærk at der bruges andre tal.

```
10 MODE 1
20 CLG
30 FOR n=1 TO 50
40 MOVE 320,200
60 DRAW RND(1)*639,RND(1)*339,RND(1)*4
70 NEXT
80 AFTER 200 GOSUB 1000
90 GOTO 90
1000 CLG
1010 MOVE 320,200
1020 FOR n=1 TO 50
1030 a=RND(1): IF a>0.5 THEN a=1 ELSE a=
-1
1040 DRAWR a*INT(RND(1)*100),a*INT(RND(1
)*100),RND(1)*4
1050 NEXT
1060 RETURN
```

Fig. 10.6 Tilfældige liniemønstre illustrerer forskellen mellem DRAW og DRAWR.

en »løbsk« flyver el. lign. Pausen laves med en anden ny ordre. AFTER 200 GOSUB 1000 betyder at programmet skal fortsætte til linie 1000 når timeren har »talt til« 200. Som før tæller timeren 50 gange i sekundet, således at 200 giver en forsinkelse på fire sekunder. Efter pausen udfører GOSUB 1000 DRAWR rutinen, og vender tilbage til den endeløse løkke i linie 90.

Planlæg grafikken

Man kan lave en masse højopløsningsgrafik, men det skal planlægges, hvis det skal se ordentligt ud. Det er en god hjælp at bruge rigtigt millimeterpapir. Hvis du lader fire centimeter svare til 100 pixels, så kan du lade det være 0 til 400 på den ene led og 0 til 650 på den anden led. Du kan lave dine tegninger direkte på millimeterpapiret, eller du kan lægge et stykke kalkerpapir oven på millimeterpapiret og tegne på det. Når din tegning er færdig (du kan indtil videre kun bruge lige linier), skal du skrive x-y-koordinaterne ned alle de steder hvor linier møder hinanden. Når det er gjort kan du begynde at skrive programmet. Fig. 10.7 viser et eksempel på en tegning, der er planlagt på denne måde. Huset er tegnet på kalkerpapir, som var lagt over millimeterpapiret. Først blev der lavet en skitse, som så blev rentegnet, så linierne fulgte linierne på millimeterpapiret, og koordinaterne til hvert punkt blev aflæst på millimeterpapiret.

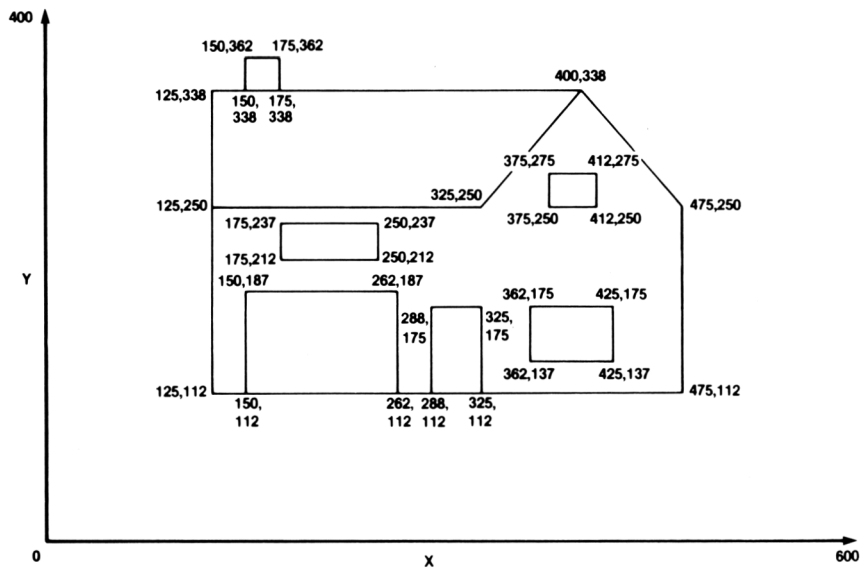

Fig. 10.7 En tegning af et hus, planlagt på kalkerpapir, der var lagt over millimeterpapir.

Det fører frem til fig. 10.8. Her er brugt DATA linier i det omfang, det er muligt, således at det er nemt at foretage rettelser, hvis det er nødvendigt. Så behøver vi nemlig kun at ændre i datalinierne og ikke i selve programmet. Hver del af tegningen starter med MOVE. Jeg starter altid i nederste venstre

```
10 MODE 0
20 MOVE 125,112
30 PEN 2
40 FOR n=1 TO 7
50 READ x,y
60 DRAW x,y
70 NEXT
80 MOVE 150,112: FOR n=1 TO 3
90 READ x,y:DRAW x,y:NEXT
100 MOVE 288,112
110 FOR n=1 TO 3
120 READ x,y:DRAW x,y:NEXT
130 MOVE 362,137
140 FOR n=1 TO 4:READ x,y
150 DRAW x,y:NEXT
160 MOVE 375,250
170 FOR n=1 TO 4: READ x,y
180 DRAW x,y:NEXT
190 MOVE 175,212
200 FOR n=1 TO 4:READ x,y
210 DRAW x,y:NEXT
220 MOVE 150,338
230 FOR n= 1 TO 3:READ x,y
240 DRAW x,y:NEXT
250 MOVE 125,338:DRAW 400,338
1000 DATA 475,112,475,250,400,338,325,25
0,125,250,125,338,125,112
1010 DATA 150,187,262,187,262,112
1020 DATA 288,175,325,175,325,112
1030 DATA 362,175,425,175,425,137,362,13
7
1040 DATA 375,275,412,275,412,250,375,25
0
1050 DATA 175,237,250,237,250,212,175,21
2
1060 DATA 150,362,175,362,175,338
```

Fig. 10.8 Programmet der tegner huset i fig. 10.7

hjørne med de ting, jeg tegner. Efter MOVE bruges DRAW i en løkke for at tegne de enkelte detaljer. Det er et meget hurtigt og effektivt program, – uden planlægning ville det tage en frygtelig tid at lave.

Vi tegner cirkler

Tegninger består ofte ikke blot af lige linier, men også af cirkler, ellipser og cirkelbuer. Her har Amstrad en af sine få svagheder, fordi den ikke har nogen CIRCLE-ordre, og heller ikke nogen ordrer, der kan farvelægge afgrænsede flader. Det må vi jo indrette os på, så vi må lave nogle subrutiner til at udføre disse opgaver. Her er der god hjælp af finde i manualen. Fig. 10.9 viser et program, der kan tegne cirkler. Linie 10 bestemmer hvilken mode, der skal bruges, hvis den tilfældigvis skulle være blevet ændret, og samtidig slettes skærmen. Linie 20 definerer cirklens radius, r, og dens centrum. Linie 30 sender programmet ned til cirkel-subrutinen, derefter sættes ORIGIN igen til det normale, og programmet kører derefter i en uendelig løkke indtil du taster ESC ESC. Linie 1010 starter en løkke, som plotter punkterne hele vejen rundt langs kanten. Hvis du har haft geometri i skolen (og kan huske det), kan du sikkert forstå hvad der sker, hvis ikke – så stol på at det fungerer, det er nemlig lidt indviklet at forklare.

```
10 MODE 1
20 ORIGIN 320,200:r=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR n=1 TO 360
1020 PLOT r*SIN(n),r*COS(n),2
1030 NEXT
1040 RETURN
```

Fig. 10.9 Programmet tegner en cirkel – ret langsomt.

Det er ikke den eneste måde at tegne en cirkel på, og det går temmelig langsomt, men det er i hvert fald et ret enkelt program. Fig. 10.10 viser en meget hurtigere måde at gøre det på. Det tegner ikke en rigtig cirkel, i stedet tegnes en *polygon* – en mangekant. Ved at vælge et højt antal sider (36), bliver polygonen rund at se på, og den er langt hurtigere at tegne! Denne subrutine er god at bruge, hvis du ikke vil vente på at få tegnet en »rigtig« cirkel.

Nu vi ser på cirkler, så prøv programmet i fig. 10.11, der udfylder en cirkel med farve. Linie 10 og 20 bestemmer også her radius og centrum, og subrutinen, der starter i linie 1000, laver en farvet cirkelflade. Det fungerer ved at tegne linierne så tæt sammen, at vi ender med at få en cirkelform frem. Da linierne er farvede, får vi til sidst en farvet cirkel. MOVE i linie 1020 flytter

```
10 MODE 1
20 ORIGIN 320,200:r=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1005 MOVER 0,r
1010 FOR n=0 TO 360 STEP 10
1020 DRAW r*SIN(n),r*COS(n)
1030 NEXT
1040 RETURN
```

Fig. 10.10 En hurtigere måde at tegne en cirkel på.

```
10 MODE 1
20 ORIGIN 320,200:r=100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR n=0 TO 360
1020 MOVE -r*SIN(n),r*COS(n)
1030 DRAW r*SIN(n),r*COS(n)
1040 NEXT
1050 RETURN
```

Fig. 10.11 Sådan fyldes en cirkel med farve. En firkant er meget nemmere at farve.

hen til et punkt på den venstre side af cirklen, og DRAW i linie 1030 laver en linie til den modsatte side. Prøv det – det går næsten ligeså hurtigt som med den FILL-ordre, som nogle få computere har.

Detektiv-arbejde

Vi kan få en masse spændende effekter frem, hvis vi kan lave et program, der automatisk viser hvor markøren er henne. Husk at den grafiske markør er usynlig, så vi må bruge en eller anden ordre, der kan fortælle os hvor markøre er henne, eller hvad den rammer. Der er faktisk flere ordrer, der kan bruges her. To af dem er XPOS og YPOS (»X- og YPOSition«). Hver af dem giver os et tal. Hvis vi sætter j = XPOS, så bliver j x-koordinaten, på det tidspunkt vi laver testen. Og hvis vi sætter k = YPOS, får vi tilsvarende y-koordinater. Disse ordrer svarer til POS og VPOS for tekstmarkøren. For-

skellen er at XPOS og YPOS ikke skal efterfølges af et vindue-nummer. Vi skal se på, hvad vi kan bruge det til.

For det første, kan vi sikre os, at den grafiske markør ikke går udenfor skærmens rammer. Fx kan vi sørge for, at når markøren forlader skærmen i den ene side, så dukker den op i den modsatte side – det kaldes »wrap-around« (»pakke-rundt-om«). Fig. 10.12 viser det med et program, der tegner nogle zig-zag linier. Linie 20 definerer en position midt på skærmen, og linie 30 placerer den grafiske markør der. Linie 30 indeholder også den nye ordre RANDOMIZE TIME. Grunden til at vi bruger den er at RND ikke giver ægte tilfældige tal, – ikke tilfældige på samme måde som fx fås med et lykkehjul. RND-tallene bliver udregnet af computeren, og som alt andet, der bliver udregnet, er disse tal ikke helt tilfældige. Resultatet er, at hvis vi laver tilstrækkeligt mange tilfældige tal, så begynder de på et tidspunkt forfra. Det kan stoppes med RANDOMIZE, og effekten bliver endnu bedre hvis tallet, der kommer efter RANDOMIZE i sig selv er et tal der ikke gentages. TIME er jo sådan et tal, så med RANDOMIZE TIME skulle vi være sikre på, at dette program ikke giver samme mønster på skærmen flere gange.

```
10 CLS
20 x=320:y=200
30 MOVE x,y:RANDOMIZE TIME
40 WHILE x<>0
50 j=RND(1):IF j>0.5 THEN x=20*RND(1) EL
SE y=20*RND(1)
60 DRAWR x,y,1
70 IF XPOS>630 THEN MOVE 10,YPOS
80 IF XPOS<10 THEN MOVE 630,YPOS
90 IF YPOS>390 THEN MOVE XPOS,10
100 IF YPOS<10 THEN MOVE XPOS,390
110 WEND
```

Fig. 10.12 Med XPOS og YPOS laves »wrap-around«.

Linie 40 starter en endeløs løkke, fordi x ikke kan være 0. Linie 50 giver så x og y nye værdier. En af dem får en ny tilfældig værdi – j. I linie 60 tegnes der med DRAW en linie med de nye værdier af x og y, dvs. med relative værdier. I linie 70 til 100 testes markørens position. Testen sikrer at markøren kommer til syne i skærmens modsatte side, hvis den når en af kanterne. Bemærk brugen af XPOS og YPOS sammen med MOVE i disse linier. Vi kan ikke bruge x og y her, fordi x og y er *relative* positionstal, ikke absolutte.

Der findes også en måde at checke den usynlige grafiske markørs position

på. Ordren TEST (x,y) giver et INK-nummer, som er INK-nummeret for positionen x,y. Du kan derfor bruge TEST (x,y) i en IF ... ordre, hvis du skal finde ud af om markøren befinder sig et sted, der har baggrundsfarven, eller om farven er en anden – fx den farve som »forhindringer« på skærmen har. Du kan også bruge TESTR, som fungerer i forhold til relative koordinater. Med TESTR (1,0) tester du fx den pixel, der er lige til højre for markøren i mode 2. Tilsvarende kan du bruge TESTR (2,0) i mode 1 og TESTR (4,0) i mode 0.

TAG

Vores højopløsningsgrafik har indtil nu mest bestået i at tegne mønstre og figurer. Der er imidlertid også muligheder for at lave bevægelser med ordren TAG (»hægte på«). TAG forbinder den grafiske markør med tekstmarkøren. Den grafiske markør er lille – kun en pixel, og tekstmarkøren fylder 8×8 pixels. De to markører hægtes sammen i tekstmarkørens øverste venstre hjørne. Hvis du PRINTer en karakter på grafikmarkørens position, så vil karakteren dække et område på 7 pixels til højre og 7 pixels ned. Her forudsætter jeg at det er en almindelig tekstkarakter på 7×7 pixels – du kan naturligvis godt bruge en »defineret« karakter på 8×8 pixels. Resultatet bliver at vi kan udskrive alle de former og figurer, vi kan få med CHR$, på grafikmarkørens plads ved hjælp af TAG. Skulle det så være en fordel? Ja, prøv at tænke på at du fx i MODE 1 har 40 pladser for en karakter på en skærmlinie. Hvis vi flytter en karakter henad denne linie, har vi kun 40 trin til rådighed og bevægelsen bliver ikke særlig jævn. Men grafikmarkøren kan bevæges i meget finere trin. I MODE 1 kan grafikmarkøren bevæges i hele 320 trin, hvis x går fra 0 til 639. Trinnene bliver altså meget mindre og bevægelsen bliver meget jævnere og animeringen ser meget bedre ud.

Se på eksemplet i fig. 10.13. Med TAG ordren i linie 30 hægtes de to markører sammen. De der har prøvet at arbejde med en BBC computer, kan genkende ordren – den svarer til BBC's VDU5 ordre. Nu kan vi lave udskrifter på grafikmarkørens plads.

Linie 40 starter en lille løkke, som bevæger grafikmarkøren tværs hen over skærmen. I linie 50 flyttes grafikmarkøren til den position, der bestemmes af x og y. Læg mærke til at vi bruger STEP 2 i løkken. Det skyldes at løkken kører gennem værdierne 0 til 639, mens der kun er 320 trin på en linie i MODE 1. Hver af grafikmarkørens pladser svarer til to x-værdier. Således er x = 0 og x = 1 samme plads, x = 2 og x = 3 er samme plads osv. I MODE 2 er der en x-værdi for hver af tallene fra 0 til 639. Prøv programmet i MODE 2. I MODE 0 er der kun 160 pladser på en linie, så hver af dem svarer til hele fire

```
10 MODE 1
20 y=200
30 TAG
40 FOR x=0 TO 639 STEP 2
50 MOVE x,y
60 PRINT " ";
70 MOVER 2,0
80 PRINT CHR$(243);
90 CALL &BD19
100 NEXT
110 TAGOFF
120 END
```

Fig. 10.13 Med TAB kan vi få tekstmarkøren hægtet sammen med grafikmarkøren.

x-værdier – x = 0, 1, 2 eller 3 er det første, x = 4, 5, 6 eller 7 er det næste osv. I en løkke skal vi derfor bruge STEP 4. Det er vigtigt at bruge STEP fordi de forhindrer bevægelsen i at blive ujævn. Hvis figuren (i MODE 0) står stille mens løkken kører fire gange, inden den flyttes, så bliver bevægelsen mere ujævn, end hvis den flyttes hver gang løkken køres igennem.

På markørens plads udskriver vi først et mellemrum for at slette det, der står på pladsen, og det har ingen betydning første gang. Derefter flytter linie 70 markøren to trin frem og linie 70 udskriver CHR$(243) – en pil. Og her skal vi så være forsigtige. Prøv at køre programmet uden semikolonnet, og se det besynderlige resultat! Når vi bruger den grafiske skærm bliver *alt* udskrevet – også koderne for linieskift og »vognretur«. Når vi bruger TAG kan vi udelade disse koder ved at bruge semikolon. Det næste mærkelige dukker op i linie 90. Her bruges der lidt »maskinkode«. Den bedste måde at finde ud af hvorfor, er at prøve at køre programmet uden linie 90. Du vil så se pilen bevæge sig hurtigere, med nogle mærkelige forstyrrelser af og til. Årsagen til disse forstyrrelser er, at skærmbilledet dannes ved at der tegnes linier på skærmen, som lyser op, der hvor der skal være noget. Når en genstand på skærmen også bevæger sig, så sker det af og til at de to bevægelser »støder sammen« (det kaldes »interferens«). For at undgå den slags problemer må vi bevæge pilen med en hastighed, der er synkroniseret med skærmens billeddannelse. Billedet dannes 50 gange i sekundet – 50 Hz. I maskinens ROM findes der en rutine, der kan lave denne synkronisering, og det er den vi kalder frem med CALL &BD19. »&BD19« er faktisk et tal i en form for kode der kaldes *hexadecimal.* Hvis du bliver nysgerrig efter at vide mere om ma-

skinkode, så har jeg skrevet en bog om Amstrad maskinkode, som du kan læse *).

Hvis du synes at bevægelsen er for langsom, kan du gøre den hurtigere på flere måder. Dels kan du gøre x og y til heltal variabler, x% og y%. Og dels kan du bruge STEP 4 eller STEP 8. Som du så gik det også hurtigere uden CALL &BD19, men den manglende synkronisering er endnu tydeligere, når bevægelsen er hurtig. Bemærk at brugen af MOVER i linie 70, MOVER 2,0 sikrer at en del af pilen hele tiden er på skærmen, og det medvirker til at give indtrykket af en jævn bevægelse. Ved andre figurer er det måske en fordel at udelade denne linie, eller at bruge andre tal.

Animation med INK

En helt anderledes og meget snild måde at lave animation på, er at bruge INK ordren. Hvis du ikke vil lave animation med en CHR$-figur (som du jo godt selv kan definere), men vil lave animation med en hel tegning, der er lavet ved hjælp af MOVE og DRAW, hvad så? Her kunne du jo lave tegningen, slette den og lave den et nyt sted osv. Men det giver en meget langsom bevægelse, og animationen er ujævn. En meget bedre metode er at bruge de utroligt effektive INK ordrer. Forestil dig at du laver et helt sæt tegninger lidt forskudt for hinanden, og hver med farve fra hver sin inkpot. Og forestil dig, at farven i hver inkpot er den samme som baggrundsfarven. Resultatet bliver så at alle dine tegninger er usynlige!

Nu skal du så forestille dig en løkke i dit program. Hver gang løkken køres igennem ændrer du en af inkpot farverne til en farve *forskellig fra* baggrundsfarven. Resultatet bliver at en af tegningerne bliver synlig. Vent et øjeblik og giv igen denne inkpot baggrundsfarve. Nu forsvinder tegningen, og du kan nu lade farven skifte i en anden inkpot, så den bliver synlig, og derefter kan du lade den forsvinde igen. På den måde kan du lade en række tegninger, *som i forvejen findes på skærmen,* komme frem og blive usynlige igen – og du har en flot, hurtig animation! Dette fungerer naturligvis bedst i MODE 0, for der er flest inkpots til rådighed.

Fig. 10.14 viser et eksempel på denne type animation. Første del af programmet giver inkpots 2 til 15 baggrundsfarven – som er farvekode 1. I linie 40 til 90 tegnes der fire firkanter på skærmen. Den ene firkant er på højkant, den næste hælder 45 grader osv. Alt i alt ser vi en firkant dreje rundt om sig selv. Hvis du vil undersøge hvad der sker i denne del af programmet, så lav INK-tallet i linie 30 om til 24 (i stedet for 1), og indsæt en linie 95 STOP.

---

*) Bogen hedder »Introducing Amstrad Machine Code« og er udgivet af det engelske forlag Granada. Den findes ikke på dansk. O.A.

```
10 MODE 0
20 FOR cl=2 TO 15
30 INK cl,1:NEXT
40 FOR nr=1 TO 4
50 READ x,y:MOVE x,y
60 FOR ln=1 TO 4
70 READ x,y
80 DRAW x,y,nr+1
90 NEXT:NEXT
100 WHILE INKEY(47)=-1
110 FOR nr=2 TO 5
115 CALL &BD19
120 INK nr,24
130 FOR x=1 TO 30:NEXT
140 INK nr,1
150 NEXT
160 WEND
170 DATA 300,300,350,300,350,100,300,100
,300,300
180 DATA 385,288,415,252,270,113,235,148
,385,288
190 DATA 225,225,425,225,425,175,225,175
,225,225
200 DATA 230,255,275,293,410,148,377,110
,230,255
210 MODE 1:END
```

Fig. 10.14 Animation ved hjælp af inkpots med »usynlig farve«.

Når tegningerne først er lavet, uden de kan ses på skærmen, kan du få dem gjort synlige ved at skifte INK-farverne. Løkken der starter i linie 100 vil køre indtil mellemrumstangenten trykkes ned. Den FOR ... NEXT løkke, ændrer INK farven et kort øjeblik, og vender derefter tilbage til baggrundsfarven. CALL &BD19 bruges for at gøre animationen mere jævn – prøv at udelade linien og se, hvad det betyder.

Vil du lave grafik, så kræver det en arbejdsindsats. Koordinaterne for de forskellige tegninger skal findes, og de skal indlæses i DATA-linierne. Det kan godt tænkes at nogen laver et program, der kan finde koordinaterne for dig. Du skal så blot lave den første figur, og programmet vil give dig koordinaterne for hver drejning af figuren. Den slags geometriske opgaver kan maskinen løse meget hurtigere, end du kan. Denne animationsmetode bruges for øvrigt også af BBC computeren, men ikke på en helt så enkel måde.

Kapitel 11

Vi sætter lyd på

En vigtig del af enhver moderne computers egenskaber er, at den kan lave forskellige lyde. Amstrad'ens lyd kommer fra en indbygget højttaler, og styrkekontrollen sidder på kabinettets højre side. Dertil kommer, at Amstrad'en kan kobles til en forstærker via en bøsning (I/0) på kabinettets bagside, derved kan du få en bedre og kraftigere lyd. Forskellen er ret imponerende, hvis du ellers kun har hørt lyden gengivet på den indbyggede højttaler. Lyden fra bøsningen kan tilmed gengives i stereo. Hvis du har en hovedtelefon fra en »Walkman«, kan du koble den direkte til bøsningen. Lydstyrken er dog meget svag, og med en stereoforstærker får du det bedste resultat.

Hvis du kun har brug for et lille bip i dit program, fx for at gøre opmærksom på et eller andet, så skal du blot skrive PRINT CHR$(7). Så får du et bip, og fig. 1.11 viser et eksempel på anvendelsen af dette i et program. Programmet er tænkt som starten på et program, der kan demonstrere forskellige lydeffekter, og den del af det vi er interesserede i, er subrutinen i linie 1000. Her bruger vi k$ = INKEY$ til at teste, om en af tasterne trykkes ned, og når det sker, tilskriver j = VAL(k$) værdien af det indtastede til j. Det interessante er, hvad der sker, når der ikke trykkes en tast ned. Linie 1010 laver et lille bip ved hjælp af CHR$(7). Du skal sætte et semikolon efter CHR$(7) for at undgå at skærmen ruller. Selvom PRINT CHR$(7) ikke giver nogen udskrift på skærmen, vil markøren alligevel skifte linie efter hvert PRINT, og hvis du venter et stykke tid uden at lave nogen indtastning, vil du se at skærmen begynder at rulle. Det ønsker vi ikke, og derfor bruger vi semikolonnet til at forhindre det. Hvis du prøver at fjerne semikolonnet, så kan du se, hvad jeg mener. Efter bip'et er der en lille pause og GOTO 1000 laver en ny INKEY$ test. Resultatet er, at maskinen bip'er indtil du trykker en tast ned – et udmærket alternativ til en blinkende stjerne el. lign. Du kan selvfølgelig kombinere lyden med en blinkende stjerne, hvis du vil, så du er sikker på, at alle vil lægge mærke til menuvalget!

Hvis du skal lave andet end små bip'er, så må du vide lidt mere om lyd. Det vi kalder lyd er hurtige ændringer i lufttrykket ved vores ører. Alt, der frem-

```
10 CLS:PRINT TAB(19)"MENU"
20 PRINT:PRINT TAB(3)"1. Musik"
30 PRINT TAB(3)"2. Eksplosioner"
40 PRINT TAB(3)"3. Bølger"
50 PRINT TAB(3)"4. Maskiner"
60 PRINT:PRINT"Vælg et nummer!"
70 GOSUB 1000
80 IF j<1 OR j>4 THEN PRINT"Forkert - pr
øv igen!":GOTO 10
90 ON j GOSUB 500,500,500,500
100 END
500 PRINT"Senere kan du indsætte et prog
ram her!"
510 RETURN
1000 K$=INKEY$:IF k$<>""THEN j=VAL(k$):
RETURN
1010 PRINT CHR$(7);:FOR n=1 TO 500:NEXT
1020 GOTO 1000
```

Fig. 11.1 Der skabes opmærksomhed på menuen med PRINT CHR$(7) – et bip.

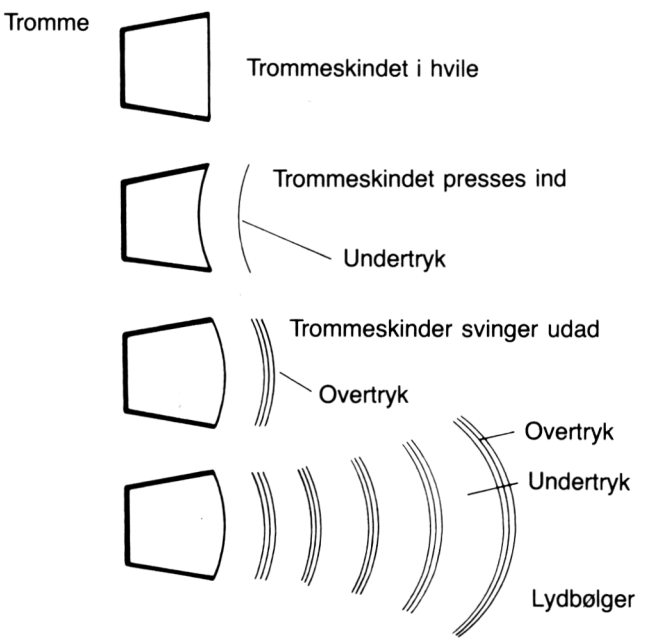

Fig. 11.2 Et trommeskind frembringer lyd ved hele tiden af skabe overtryk og undertryk.

bringer lyde, gør det ved at ændre lufttrykket, og fig. 11.2 viser hvordan lyden frembringes af en tromme. Lufttrykket er usynligt, og vi lægger ikke mærke til disse ændringer i lufttrykket, med mindre det sker ret hyppigt. Vi måler hyppigheden af disse ændringer i *svingninger pr. sekund* eller *Hertz* (forkortes *Hz*). Svingningerne ændrer først lufttrykket den ene vej og så den anden vej osv., indtil de holder op. Se hvordan det kan fremstilles grafisk på fig. 11.3. Lydsvingninger kaldes også lydbølger – og du kan se at grafen faktisk ligner bølger i vand. Grafen viser hvor meget lufttrykket ændres pr. tidsenhed.

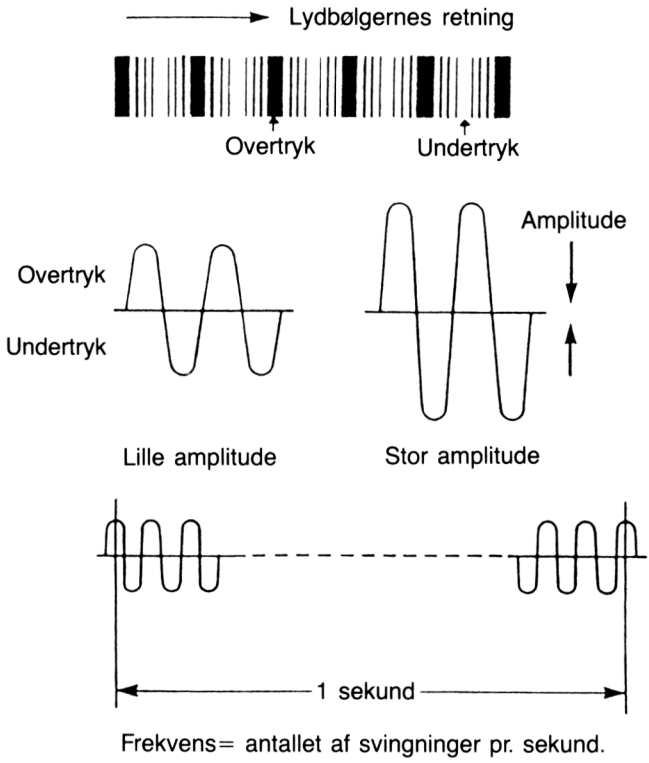

Fig. 11.3 Frekvens og amplitude (lydstyrke) for en lydbølge.

Lydens *frekvens* er antallet af svingninger pr. sekund (Hz). Hvis frekvensen er under ca. 20 Hz kan vi simpelt hen ikke høre svingningerne, (disse lavfrekvente svingninger kan i øvrigt godt virke generende).Vi kan høre frekvenser fra ca. 20 Hz og op til ca. 15.000 Hz. Frekvensen svarer til det vi

opfatter som»tonehøjden«. En lav frekvens på omkring 100 Hz svarer til en dyb tone (bas). En frekvens på 400 Hz og derover svarer til en høj tone (diskant).

Det er trykkets størrelse, der er bestemmende for en tones lydstyrke. Det kaldes *amplituden,* og står altså for, hvor store udsvingene i lufttrykket er i forhold til det normale. For at kunne styre frembringelsen af en tone skal vi kunne kontrollere frekvensen, lydbølgernes form og amplituden, og vi skal også kunne kontrollere ændringerne i amplituden.

Når der skal skrives musik, skal komponisten bestemme hver eneste tones højde, styrke og længde. Når musik bliver nedskrevet bliver styrken angivet med bogstaver som f (kraftigt) og p (svagt), og der kan anvendes flere bogstaver, hvis det er nødvendigt. Fx betyder fff meget kraftigt og ppp meget svagt.

Tonernes længde angives ved nodetegnets udformning. Nu er denne længde selvfølgelig relativ, og derfor kan man benytte en metronom til at angive præcis hvor lang tid noden skal vare. En metronom lyder som et tikkende ur – men man kan vel og mærke variere hastigheden af tikkene. Hvis man skriver således: ♩ = 100 betyder det at hver fjerdedelsnode skal vare en hundredel minut.

Man opdeler nodesystemet i takter, og er der tale om en firefjerdedelstakt varer en *helnode* en hel takt (eller fire taktslag), en *halvnode* en halv takt (eller to taktslag). En fjerdedelsnode varer et taktslag osv. – se fig. 11.4. Der er tilsvarende angivelser for pauser i musikken – se fig. 11.5. Det er ikke alle partiturer, der bruger metronomangivelser, i stedet anvendes italienske ord som fx lento, moderato eller allegro til at angive tempoet.

Tonehøjden angives i musik i *nodesystemer*. Klavermusik har to nodesystemer, som hver består af fem linier og fire mellemrum. De to nodesystemer er mærket med G-nøgle(𝄞), og en bas-nøgle (𝄢).

G-nøgle-systemet bruges til de høje toner, og bas-nøgle-systemet bruges til de dybe toner. Musik der ikke skrives for klaviaturinstrumenter, bruger kun

Symbol	*Længde*	*Navn*
𝅘𝅥𝅯	1/8	32-delsnode
𝅘𝅥𝅰	1/4	16-delsnode
𝅘𝅥𝅮	1/2	Ottendedelsnode
𝅘𝅥	1	Kvartnode
𝅗𝅥	2	Halvnode
𝅝	4	Helnode

Fig. 11.4 Nodetegnene og deres længde.

Pause	*Længde*
𝄿	¼
𝄾	½
𝄽	1
𝄼	2
𝄻	4

Fig. 11.5 Pausetegn

èt nodesystem. Klaver- og orgelmusik har altid to nodesystemer, med plads til noder mellem dem. Den node der ligger lige midt i mellem hedder et *enstreget* C, og på et klaver ligger dette C midt på klaviaturet – (dette C kaldes nogle gange for et 'nøglehuls-C'). Fig. 11.6 viser nodesystemet med C'et i midten.

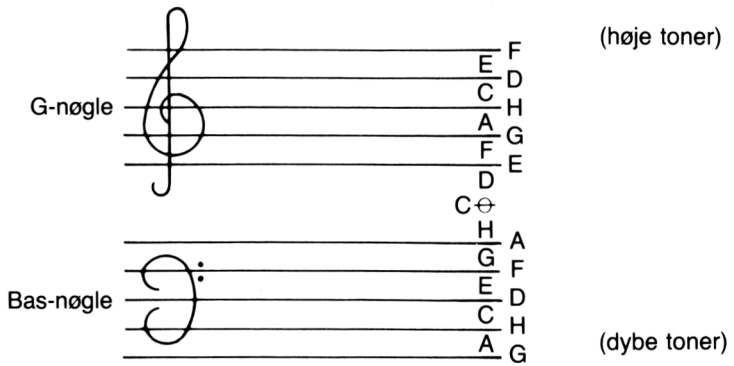

Fig. 11.6 De to nodesystemer, med tonernes navne skrevet ind. (Læg mærke til det enstregede C midt mellem systemerne).

De noder, der er vist i fig. 11.6, er ordnet i grupper af otte. Sådan en gruppe hedder en *oktav*. Der er kun en halv tone mellem H og C og mellem E og F, mellem de andre er der en hel tone. Der, hvor der er et spring på en hel tone mellem de hvide tangenter (se fig. 11.7), er der indsat sorte tangenter som halvtoner. Således er der (i vestlig musik) 12 toner i en oktav – hel- og halvtoner. I nodesystemet er der ikke plads til halvtonerne, så de mærkes i stedet med ♯ og ♭. ♯ ('kryds for') betyder en halv tone ned. På klaveret kan du se at 'kryds for' en tone er det samme som 'b for' den næste tone – ♯C er altså det samme som ♭D. Det gælder for langt de fleste instrumenter.

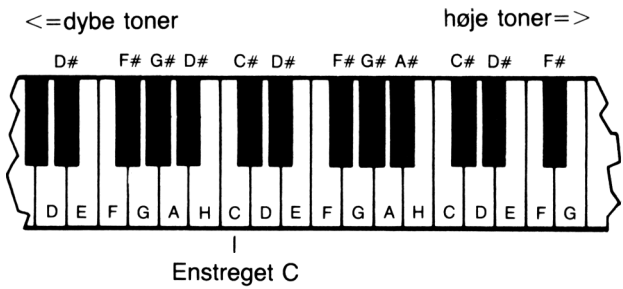

Fig. 11.7 Udsnit af klaviatur med det enstregede C. Der er kun en halv tone mellem H og C og mellem E og F.

Amstrad SOUND

Så kan vi gå i gang. Amstrad har en SOUND-(»lyd«) ordre, som enten kan bruges på en meget enkel måde, eller på en ret kompliceret måde. Hvis du blot skal lave en advarselstone eller spille en lille melodi, er den ret enkel at anvende. Men når du har lært SOUND bedre at kende, giver den en masse spændende muligheder. Af alle de computere, jeg i tidens løb har brugt, har Amstrad – specielt på lydområdet – langt det bedste BASIC. Vi starter med at lave toner.

Her skal vi bruge SOUND ordren, som altid skal efterfølges af mindst to tal. Det første tal er en kode der bestemmer valget af »kanal«. Amstrad kan lave tre toner på en gang, og de kan styres individuelt. Det sker ved at give dem hver sin »kanal«. Kanalerne hedder A, B og C, og vi kan vælge at bruge en eller flere af dem med kanal-koden. Fig. 11.8 viser hvordan tallene fra 1 til 7 kan bruges til at vælge kanal(er). Amstrad giver bedre muligheder for at gengive i stereo end andre hjemmecomputere. Kanal A går til venstre kanal, og kanal C går til højre kanal. Kanal B går til begge kanaler, og alt i kanal B vil lyde lige kraftigt i begge stereokanaler. Hvis der bruges tal større end 7 får vi mere komplicerede effekter, og dem ser vi på senere.

Kanal-kode	Kanal(er)
1	A
2	B
3	A og B
4	C
5	A og C
6	B og C
7	A og B og C

Fig. 11.8 De koder der bruges til at vælge de tre kanaler.

Tal nr. 2 efter SOUND-ordren bestemmer tonehøjden. Den kaldes »perioden«. En periode er den tid en hel svingning tager, og jo kortere tid en svingning varer, jo flere er der inden for en bestemt tid, og jo højere bliver frekvensen *). Små perioder har derfor de højeste toner. Periodetallene går fra 1 (den højeste tone) til 4095 (den dybeste tone). I praksis bruges tallene fra 10 til 1000, fordi perioder over 1000 mest lyder som en række »klik« – med mindre du har nogle virkelig gode højttalere. Tallet nul kan også bruges, men det er til andre formål, som vi ser på senere. Toner med perioder mindre end 10 er så høje, at de ikke rigtig kan høres. I manualens Appendix VII kan du se en oversigt over sammenhængen mellem tonernes navne her og i musikken **). På Page 2 står de mest almindelige toner – svarende til de tre midterste oktaver på klaveret. Et klaver er nok et af de musikinstrumenter flest kender, og dets klaviatur er lavet på en måde, der gør det meget nemt at spille en bestemt række toner – C-dur skalaen. Denne skala starter med det enstregede C og fortsætter med den næste 7 hvide tangenter, således at der sluttes på det C, der ligger en oktav over.

Vi skal senere spille en C-dur skala, men vi starter med at spille en enkelt tone. I fig. 11.9 ser du SOUND ordren i linie 10. Her vælges kanal A med et 1-tal, og perioden sættes til 478, som er meget tæt på et enstreget C. Linie 10 starter tonen, og i dette tilfælde varer den et halvt sekund. Nu er det meget vigtigt at forstå, hvad der sker. Når computeren udfører SOUND-ordren tager den alle tallene og sender dem til et helt separat system – lydsystemet.

```
10 SOUND 1,478,50
20 FOR n=1 TO 2000:NEXT
30 SOUND 1,478,100
40 FOR n=1 TO 2000:NEXT
50 SOUND 1,478,200
```

Fig. 11.9 En enkelt tone.

Denne del af computeren arbejder adskilt fra computerens øvrige funktioner, og frembringer lyden. I mellemtiden fortsætter computeren i programmet. Og her kan computeren altså også gøre to ting samtidigt. Hvis der er mange SOUND-ordrer, vil computeren lynhurtigt køre dem igennem, og sende alle tallene til lydsystemet. Hver tone lyder i en vis tid, computeren kan let have sendt tallene for de næste fire toner frem til lydsystemet, når den første tone er slut! Det er det manualen kalder en »lydkø« (»sound queue«), og hvis du

*) Se på figuren i manualen Chapter 6 Page 2. O.A.

**) Der er fire kolonner i manualens Appendix VII: tone (note) – frekvens – tonehøjde (period) – og procentvis afvigelse (relative error). O.A.

ikke har forstået dette princip, så vil du få problemer med at få de mere komplicerede SOUND-ordrer til at give de lydeffekter, du forventer. I dette eksempel er der indsat en pause efter hver lyd i programmet, således at de enkelte toner når at slutte, før computeren når frem til næste SOUND. Tonelængden bestemmes af det tal, der følger efter perioden –dvs det tredje tal i SOUND-ordren. I linie 10 er tallet 50 og tiden er 50/100 sekund, altså et halvt sekund. Tonelængder på nul og negative værdier bruges til andre formål, og dem vender vi tilbage til.

Fig. 11.10 viser en variation af det foregående program, idet der her er lige lange pauser mellem tonerne. Det klares med funktionen SQ (Sound Queue = 'lydkø'), som giver oplysninger om lydkøen på samme måde, som XPOS oplyser om markørens position. Når der venter noget i køen er SQ mindst 128. Ved at teste denne værdi med en WHILE ... WEND løkke kan vi få computeren til at vente med at gå videre i programmet, til tonen er færdig. På den måde kan vi også her få adskilt tonerne. Vi får her yderligere den fordel, at vi med samme FOR ... NEXT-løkke får det samme interval mellem tonerne. Hvis du bruger en simpel FOR ... NEXT-løkke starter pausen i det øjeblik en lyd bliver sendt hen i lydkøen. Men med WHILE ... WEND og SQ kan du lave en pause, fra den ene lyd ender, til den næste starter. SQ skal efterfølges af kanalkoden i parentes. Der er andre måder at bruge SQ på, men det er der ikke plads til at behandle her. Amstrad's lydsystem har så mange muligheder, at det skulle have sin egen manual!

```
10 SOUND 1,478
20 WHILE SQ(1)>128:WEND
30 FOR n=1 TO 100:NEXT
40 SOUND 1,478,50
50 WHILE SQ(1)>128:WEND
60 FOR n=1 TO 100:NEXT
70 SOUND 1,478,200
```

Fig. 11.10 Her er der en bestemt pause mellem tonerne. SQ bruges til at finde ud af hvornår en tone ender.

Næste trin er at lave C-dur skalaen. Vi finder frem til de rigtige toner ved hjælp af tabellen i manualen. Ved hjælp af denne tabel er programmet i fig. 11.11 lavet. Vi starter med det enstregede C (»Middle C«) og fortsætter op til det C, der ligger en oktav over. Læg mærke til at frekvensen fordobles fra det ene C til det andet. Det samme gælder for alle andre toner i skalaen – at fordoble frekvensen svarer til at stige en oktav. Og tilsvarende svarer en halvering til at gå en oktav ned. For perioden er det naturligvis lige omvendt.

```
10 CLS:PRINT TAB(13)"C-DUR SKALA":PRINT:
PRINT
20 FOR n=1 TO 8
30 READ note
40 SOUND 2,note,100
50 NEXT
60 PRINT"Nu er alle tonerne i køen!"
100 DATA 478,426,379,358,319,284,253,239
```

Fig. 11.11 Amstrad's C-dur skala.

Her svarer en halvering af perioden til at stige en oktav. Du kan se i fig. 11.11 at det enstregede C svarer til 478 og C'et en oktav højere svarer til det halve – 239. Det er ikke altid muligt at lave dette helt præcist, når vi kun kan bruge hele tal, men vi kan holde os så tæt som muligt til dette forhold.

I fig. 11.11 kan du også se lydkøen i funktion. Amstrad lydsystemet kan have op til 5 toner i en kø. Mens en af tonerne lyder, venter de andre fire i køen. Når du kører programmet, ser du programmets titel og du hører skalaen starte. Når der er kommet fem toner ind i køen, er der ikke plads til flere og computeren må derfor vente. Så snart de sidste toner er kommet ind i køen, kan computeren komme videre i programmet, og meddelelsen bliver skrevet ud.

Nu skal vi til at bruge flere lydkanaler på en gang. Fig. 11.12 viser en akkord, med tre toner – en i hver kanal. Vi har ikke set på styrkekontrollen endnu, så vi bruger for nemheds skyld samme lydstyrke til alle tre toner, men ofte giver det en bedre musikalsk effekt at bruge forskellige lydstyrker i de tre kanaler. Det skyldes at det menneskelige øre er mindre følsomt overfor dybe og høje toner, end overfor toner i mellemområdet. Akkorden er måske ikke helt så »ren« som vi kunne ønske, det skyldes at tallene, der bestemmer tonehøjden, kun er tilnærmede. Hvis du er vant til at lytte til musik, kan du jo prøve at lave små ændringer i tallene. Jeg synes fx, at det lyder bedre med 380 i stedet for 379 (linie 20).

```
10 SOUND 1,478,300
20 SOUND 2,379,300
30 SOUND 4,239,300
```

Fig. 11.12. En akkord hvor tre kanaler benyttes.

Det fjerde tal, vi kan bruge i SOUND ordren, bestemmer tonernes relative lydstyrke. Relativ lydstyrke betyder, at du hvis du bruger forskellige tal på denne plads, får du forskellige lydstyrker – med samme indstilling af styrke-

kontrollen på computeren (eller stereoforstærkeren). Du kan styre den *absolutte* lydstyrke med styrkekontrollen, som du plejer. Du kan bruge tallene fra 0 (ingen lyd) til 7 (fuld lydstyrke) til at styre den relative lydstyrke med. Tallene fra 0 til 15 kan bruges, når lydstyrken styres af en *envelope* – det er også noget af det vi vender tilbage til. Hvis du prøver at bruge tal større end 15 til at styre lydstyrken, får du fejlmeldingen »Improper argument«.

Fig. 11.13 viser hvordan vi kan bruge lydstyrke-tallene. Programmet starter med en løkke hvor lydstyrken stiger med tallene fra 0 til 7. I anden del af programmet falder lydstyrken tilsvarende mens løkken kører med STEP −1. Resultatet bliver at du hører en tone blive trinvis kraftigere, og derefter aftage trinvis igen.

```
10 FOR n=0 TO 7
20 SOUND 2,339,50,n
30 NEXT
40 FOR n=7 TO 0 STEP -1
50 SOUND 2,339,50,n
60 NEXT
```

Fig. 11.13. Brugen af lydstyrketallene.

De toner der frembringes af musikinstrumenter ændrer lydstyrke på lignende måde. Mange instrumenter starter med en kraftig lydstyrke, som så efterhånden bliver svagere. Klaveret er et typisk eksempel på dette, – lyden er kraftigst når hammeren rammer strengen, så holdes en »dæmper« ind mod den, og lyden dør ud igen. Amstrad kan efterligne denne »envelope« på andre måder, så vi behøver ikke bruge løkken til at styre amplituden med.

Musik, musik, musik

Amstrad's tre lydkanaler gør det muligt at lave musik med akkorder (dog højst 3 toner i hver akkord – det kaldes treklange). Med mindre du er en dygtig komponist, eller gerne vil være det, er det bedst at gå ud fra noget, der er komponeret i forvejen. Det mest velegnede, hvis du vil bruge tre kanaler, er musik for klaver og violin, eller sopran og klaver eller fløjte og klaver. For denne musik ligger tonerne indenfor det område, som den lille højttaler er bedst til. Musik for kontrabas lyder ikke så godt på en lille højttaler!

Den bedste teknik er at bruge en løkke, som indlæser data for tonehøjde og tonelængde for hver tone. Vi skal senere se på de andre data, som du også kan bruge, men det er bedst at starte med noget enkelt. Når du prøver dette første gang, er det lettest at bruge en DATA linie for hver tone. Hvis du vil have en tone til at fortsætte i den ene kanal, mens de andre ændres, skal du

bruge noget synkronisering, som vi straks skal se på. Du vil opdage at klavermusik gengives bedst, hvis der er et kort ophold mellem tonerne, mens orgelmusik virker fint uden ophold. Når du først har lært at programmere lyden, så er det vigtigste at få nogle erfaringer.

Se på eksemplet i fig. 11.14, som byder på nogle akkorder. Jeg har lavet det uden at have et partitur som forlæg, og jeg skrev en kanal ad gangen. Det er ret lige til, for hovedprogrammet består af en løkke, hvor c1, c2 og c3 er tonerne i kanal A, B og C, og p er perioden. Ved at lave det på den måde er det nemt at lave om på melodien, ved at ændre DATA-linierne. Jeg fandt først tonerne til kanal A, og imens brugte jeg nul i de andre kanaler, og jeg fandt nogle passende tonehøjder. Da det var gjort, lavede jeg lyden til kanal B, og endelig kom lyden i kanal C på. Det ville nok have lydt bedre, hvis jeg havde haft et partitur at kigge i. Beethoven skrev partiturer, – jeg programmerede bare en computer.

```
10 FOR n=1 TO 9
20 READ c1,c2,c3,p
30 SOUND 1,c1,p,7
40 SOUND 2,c2,p,5
50 SOUND 4,c3,p,5
60 NEXT
70 END
500 DATA 478,319,956,100
510 DATA 506,319,851,70
520 DATA 478,319,956,210
530 DATA 426,319,851,70
540 DATA 379,243,758,140
550 DATA 319,268,638,70
560 DATA 358,284,716,70
570 DATA 379,319,758,100
580 DATA 426,319,851,140
```

Fig. 11.14 Treklange i tre kanaler.

Lydeffekter

SOUND-ordren kan, selv i denne enkle form, lave en lang række lydeffekter. Først skal vi se på en tone hvor tonehøjden stiger. Når den gentages nogle gange i træk, er den velegnet til »advarselslyd« – det har politiet opdaget! Programmet kan du se i fig. 11.15. I linie 10 gennemløber j værdierne 1 til 10, og tonen gentages derved ti gange. I den næste løkke gennemløber n værdierne fra 200 til 1 med STEP −10, herved dækkes en god del af det hørbare

```
10 FOR j=1 TO 25
20 FOR n=200 TO 1 STEP - 10
30 SOUND 2,n,1,7
40 NEXT
50 NEXT
```

Fig. 11.15 Programmering af stigende tonehøjde, ved hjælp af en løkke.

område. Disse tal indsættes i SOUND-ordren i linie 30. Her bruges en meget kort tonelængde og fuld lydstyrke. Prøv og lyt godt efter. Et af problemerne ved disse SOUND-ordrer er, at det er svært at beskrive resultaterne! I grafikprogrammerne er det ofte muligt at danne sig et indtryk af, hvad der vil ske, men i et lydprogram er det ofte en overraskelse – indtil man kender det virkelig godt. Det er derfor ekstra vigtigt at du prøver alle programmerne her af, så du kan høste nogle erfaringer. Det kan være en god idé at indspille de forskellige lyde på en kassettebåndoptager.

Fig. 11.16 viser et program, hvor to toner hurtigt skifter. Sådan en lyd er også god til at tiltrække opmærksomheden. Det har telefonselskaberne opdaget! I programmet kører løkken to toner igennem 50 gange, tonerne varer hver 5/100 sekund. De valgte perioder er her 239 og 244. Højere toner er mere effektive – prøv fx 90 og 95 – det signal lægger man da mærke til!

```
10 FOR j=1 TO 50
20 SOUND 2,239,5,7
30 SOUND 2,244,5,7
40 NEXT
```

Fig. 11.16 Et to-tonet signal. Telefonen ringer!

Vi tilsætter støj

Du kan også lave en anden slags lyd på en eller flere af de tre kanaler, det er virkelig en anden slags lyd – nemlig støj. Det vi kalder støj, er en tilfældig blanding frekvenser, men ofte er bestemte frekvensområder dominerende. En støjkilde er meget nyttig at have til at lave bølgeslag, trommelyde, pistolskud og andre lyde, der er umulige at lave med en almindelig SOUND-ordre. For at lave støj skal vi lukke af for tonegeneratoren, ved at bruge et nul som periode. Støjens »periode« kan ligge mellem 0 og 15, og den skal stå som det syvende tal i SOUND-ordren. Jeg ved godt, at vi har sprunget det femte og sjette tal over, det er der gode grunde til, og vi vender tilbage til dem senere. Indtil videre skriver vi nul på deres pladser.

```
10 CLS
20 FOR j=0 TO 15
30 PRINT"Støjinterval";j
40 SOUND 1,0,20,7,0,0,j
50 FOR n=1 TO 2000:NEXT
60 NEXT
```

Fig. 11.17 Støj i én kanal.

Fig. 11.17 viser et program der sender støj i kanal A. Der er kun én støjgenerator, så du kan ikke have forskellig støj i de tre kanaler, men du kan selvfølgelig lægge støjen i den eller de kanaler, der passer dig. Med en stereoforstærker – og højttalerne foran dig – kan du skabe en rigtig rodeo-stemning, med pistolskud til alle sider! Som sagt kan du bruge tallene mellem 0 og 15 til at lave støjeffekter med. Men princippet er lidt forskelligt fra de andre effekter, for her gentages tallene blot hvis du bruger tal der er højere end 15. Der er ikke stor forskel mellem støjen, som laves med tallene fra 0 til 5, men prøv programmet i fig. 11.17, så kan du høre dem alle 16. I det næste program i fig.11.18 demonstreres to glimrende støjeffekter. De laves ved at programmere støjen ind i en løkke, og lade forskellige ting ændres i de to tilfælde. I det første eksempel hører vi ti bølgeslag. Det gøres ved at ændre støjens

```
10 CLS:PRINT:PRINT"Bølgeslag...."
20 FOR n=1 TO 10
30 FOR j=15 TO 1 STEP -1
40 SOUND 2,0,20,7,0,0,j
50 NEXT:NEXT
60 PRINT:PRINT"Smedens hammer...."
70 FOR n=1 TO 10
80 FOR j=15 TO 0 STEP-1
90 SOUND 2,0,5,j,0,0,1
100 NEXT:NEXT
```

Fig. 11.18 Støjgeneratoren kan bruges til at lave lydeffekter.

periode med variablen j i en løkke. Først er de dybe støjfrekvenser dominerende – og derefter dominerer de høje. I den anden del af programmet kan du se hvordan vi kan lave »hammerlyde«. I dette tilfælde ændrer løkken lydstyrken, mens støjens periode er uændret. Med et lavt periodetal får vi en »lille« hammer, med højere tal giver det »dybere« støj, og vi får andre effekter.

Vi udvider SOUND

Nu er det på tide at se på de muligheder vi har for at udvide SOUND-ordren, så vi kan kontrollere lydens »envelope«. Denne effekt er ret kompliceret, og fordi det er så svært at beskrive hvordan en lyd lyder, bliver du nødt til at prøve programmerne og lytte! Men først må jeg forklare hvad en »envelope« er. De lyde vi kan lave med den enkle version af SOUND, har konstant amplitude og konstant frekvens. Eller sagt på en anden måde, lydstyrke og tonehøjde er uændret, mens tonen lyder. Med musikinstrumenter er det anderledes, for her ændrer de enkelte toner lydstyrke. En tone på et klaver er fx kraftigst lige når hammeren har ramt strengen. – dvs i tonens start. Derefter bliver tonen hurtigt svagere, og det er netop måden tonerne dør ud på, der gør klaverlyden så karakteristisk. Andre instrumenter frembringer lyde, der er helt anderledes, og alle instrumenter frembringer toner, der er en blanding af forskellige frekvenser. Det er derfor du kan høre forskel på lyden fra et klaver, en violin, en fløjte, en obo osv. – selvom de spiller den samme tone. En kurve over en tones amplitude, som strækker sig over det tidsrum tonen lyder, kaldes for en tones »volume-envelope« (»lydstyrke-envelope«).

Dertil kommer at en tones frekvens heller ikke er konstant. Hvis du har set en violinist spille, har du sikkert lagt mærke til at den hånd, der trykker

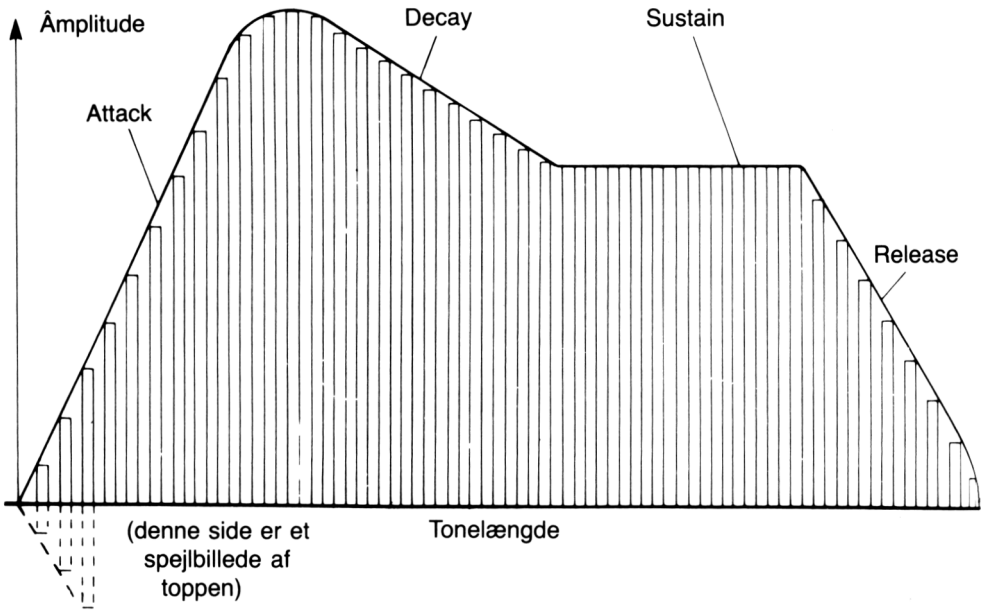

Fig. 11.19 En kurve over en typisk »volume-envelope«, som viser hvordan lydstyrken ændres, mens tonen lyder.

strengene ned, hele tiden bevæges en lille smule. Det giver den effekt at tonen får »vibrato« – dens tonehøjde »vibrerer«. Når der bruges vibrato, bliver lyden lidt mere spændende at lytte til. En tone med vibrato eller andre variationer i tonehøjden, har »tone-envelope«.

Se nu på fig. 11.19, her ser du en styrke-envelope, som er typisk for mange musikinstrumenter. Amplituden stiger hurtigt i starten – det kalder vi »attack« (»angreb«). Herefter kommer »decay« (»henfald«), hvor amplituden falder. Så kommer »sustain« (»holde i gang«) hvor amplituden er næsten konstant. Og endelig »release« (»udløse«) hvor amplituden falder til nul – tonen dør ud. Med Amstrads uvidede SOUND-ordre kan vi efterligne sådan en envelope-kurve. Princippet er, at vi kan bruge et tal, som kan stå for forskellige envelope-kurver. Hver af disse kurver har et nummer, og nummeret skal stå som det femte tal i SOUND-ordren. Når vi bruger envelope, skal den kunne kontrollere både lydstyrke og tonelængde. Hvis tonelængden er sat til nul, træder envelopen i funktion. Tallet for lydstyrken bestemmer udgangsnuveauet for styrken, men derefter bestemmer envelopen forløbet.

Vi prøver envelope

Når vi skal lave en volume-envelope, skal vi bruge ENV-ordren, og den skal efterfølges af mindst 4 og højst 16 tal. Det kan godt se lidt skræmmende ud, men ligesom resten af Amstrad's fremragende lydsystem, er det virkelig værd at sætte sig ind i. Lad os se på det med nogle eksempler. Først skal vi have et planlægningsark, og fig. 11.20 viser, hvordan det kan laves. Lydstyrken har trinnene fra 0 til 15, og det er det område, der kan bruges med denne ordre.

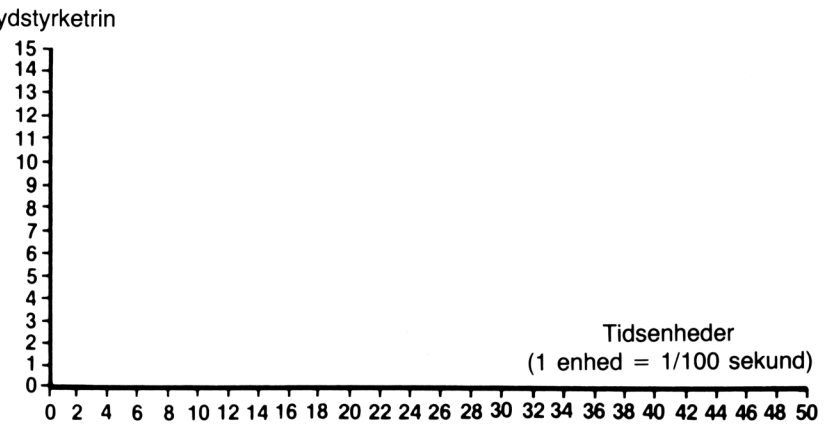

Fig. 11.20 Planlægningsark til ENV-ordren. Du kan selv tegne et på millimeterpapir.

Enhederne på tidsaksen er 1/100 sekund, og her er der lavet plads til 50 enheder, svarende til et halvt sekund. Hvis du vil bruge længere toner, må du lave om på denne akse. Princippet er så at du indtegner din envelope-kurve i dette koordinatsystem, idet du følger enhederne på akserne. Du kan ikke bruge halve enheder, så hvis du har brug for det, må du lave tilnærmelser. Du skal finde ud af hvor mange lydstyrketrin du vil bruge – husk at maximum er 15.

Fig. 11.21 viser den enkleste envelope – en tone der falder fra fuld lydstyrke til nul i en lige linie. Nu skal vi se, hvor mange trin vi skal bruge. For at gøre det enkelt nøjes vi med at lade lydstyrken falde i 5 trin på hver −3 lydstyrkeenheder, og hvert trin varer 4 tidsenheder. Tiden måles i de intervaller, hvor lydstyrke ikke falder – dvs. der hvor trinnene er parallelle med tidsaksen – i manualen kaldes tiden for »pause time«. Lad os kalde denne envelope nummer 1. Nu kan vi programmere denne envelope med ENV, derefter nummeret, som er 1, så antallet af trin, trinstørrelsen og trintiden. Det bliver til ENV 1,5,−3,4. Nu kan vi indsætte ordren i et program, og høre hvordan den lyder (se fig. 11.22).

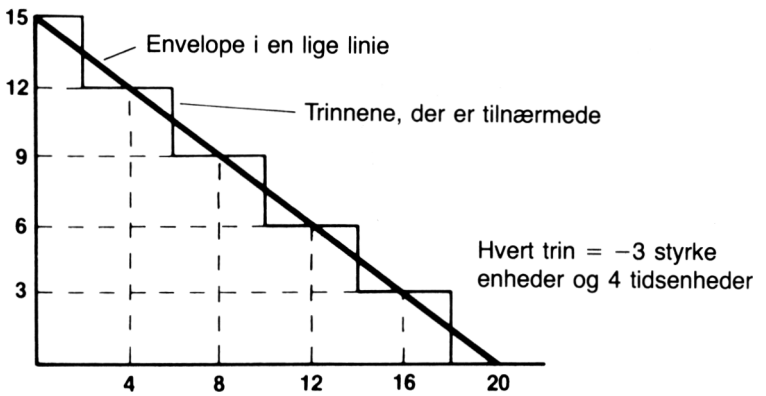

Fig. 11.21 En meget enkel envelope inddelt i trin.

```
10 CLS
20 ENV 1,5,-3,4
30 PRINT"Tonelængde: 0"
40 SOUND 2,239,0,15,1,0,0
50 FOR n=1 TO 2000:NEXT
60 PRINT"Negativ tonelængde: 10"
70 SOUND 2,239,-10,15,1,0,0
```

Fig. 11.22 Envelopen i et program – hør den!

Dette program definerer først vores envelope, som derefter bruges i en SOUND-ordre. SOUND bruger kanal 2, og tonens periode er 239. Tonelængden er 0, således at envelopen kan bestemme varigheden. Envelopen varer 18 tidsenheder – 18/100 sekund. Vi starter med fuld lydstyrke – 15. Når vi bruger envelope kan vi bruge lydstyrker fra 0 til 15. Efter at have fastsat lydens startniveau, skal vi angive envelopens nummer – 1. Periodeenvelopen er 0, den bruger vi jo ikke endnu, og støj-perioden er også 0 – den bruger vi heller ikke her. Alt dette kommer ind i linie 40, og du kan høre envelopen, når du kører programmet. I linie 70 demonstreres det, hvad negativ tonelængde betyder. Det får envelopen til at gentage det antal, der angives – minusset ses der bort fra. En tonelængde på −10 får således envelopen til at gentages ti gange.

Selv med en så enkel envelope, hvor vi kun bruger fire tal i ENV-ordren, er der rige muligheder for at eksperimentere. Prøv fx at sætte perioden til 1 og støjen til 12 – så får du et pistolskud. Og der er mange muligheder for at lave variationer ved at blande toner med støj. Skal vi have mere udbyggede effekter, må vi til at bruge hele ENV-ordren. Her kan envelopen opdeles i *hele fem afsnit,* og der kan bruges meget finere trin, end de fem vi brugte i det første eksempel – det bedste er at prøve at bruge så mange lydstyrketrin som muligt i hvert afsnit, og dvs. op til 15. Nu skal vi prøve at lave en »rigtig« envelope, som består af fire afsnit, attack, decay, sustain og release.

I fig. 11.23 er envelopen indtegnet. Også her viser den tykke linie den lyd, vi gerne vil have frem, og trinnene viser den tilnærmelse vi må lave. Her skal vi bruge en ENV-ordre med fire afsnit, og de tal der bruges er vist ude til højre for kurven. Nu skal vi prøve den med programmet i fig. 11.24. ENV starter med at give envelopen et nummer, så kommer antallet af trin, trinstørrelsen og trin-tiden for hvert afsnit. Når det er gjort laver vi lyden med en tonelængde på −10 for at få lyden gentaget ti gange. I linie 40 til 70 prøver vi

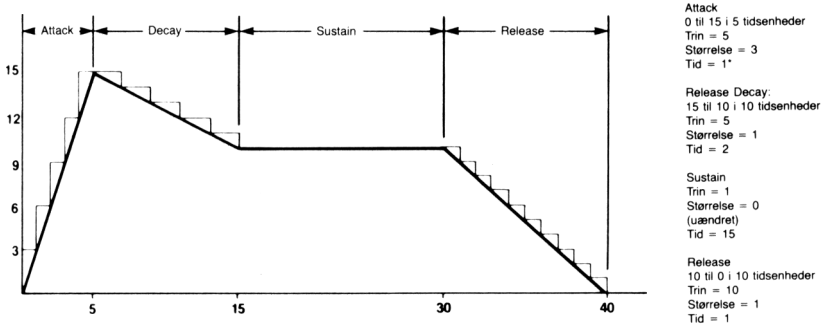

Fig. 11.23 Kurve og plan for en envelope i fire afsnit.

```
10 ENV 1,5,3,1,5,-1,2,1,0,15,10,-1,1
20 SOUND 2,478,-10,0,1,0,0
30 FOR n=1 TO 2000:NEXT
40 FOR j=1 TO 8
50 READ tone%
60 SOUND 2,tone%,0,0,1,0,0
70 NEXT
100 DATA 478,379,426,638,638,426,379,478
```

Fig. 11.24 SOUND-programmet, der bruger envelopen ovenfor.

et lille »klokkespil« med vores envelope. Er den lidt for hurtig? – så sæt bare et femte afsnit med en pause til envelopen. Tilføj tallene 1,0,40 efter ENV-ordren og hør forskellen!

Det kræver planlægning, et »godt øre« og masser af øvelse at lave envelopes. Selvom reglerne for at lave en envelope egentlig er ret enkle – meget enklere end på andre maskiner – så er der allligevel nogle tips, der kan være gode at kende. Lav trinnene i en rimelig størrelse og brug et antal trin. Hvis du laver din envelope for kort er der ikke meget at lytte til, med mindre du laver lyde som fx pistolskud. Når du bruger tonerne i musik, virker korte envelopes ikke særlig godt. Her skal tonerne gerne vare mindst et sekund. Et andet godt råd er at lade ændringerne i amplituden være indtegnet som diagonaler i kvadraterne på millimeterpapiret. På den måde kan du sikre dig at have hele tal til trinstørrelser og antallet af trin.

Toner med vibrato

Vi så tidligere at vi også kan bruge tone-envelope. Tone-envelopen programmeres i stor udstrækning på samme måde som volume-envelopen. På fig. 11.25 kan du se et koordinatsystem, der kan bruges til at planlægge en tone-envelope. Her skal vi også tilnærme den kurve vi ønsker til en række trin. Men her er trinnene periodetal. Er ændringen positiv bliver tonen dybere, er den negativ bliver tonen højere. Vi skal som sædvanlig se på et eksempel, og fig. 11.26 viser en enkel tone-envelope kurve. Denne envelope får tonen til at stige i 8/100 sekund, og derefter falder den i de næste 8/100 sekund. Som sædvanlig tegnes linien gennem diagonalerne på millimeterpapiret, for at gøre tilpasningen nemmere. Fig. 11.27 viser et program, der vil give os denne lyd. Den er god at bruge som lydeffekt i mange sammenhænge!

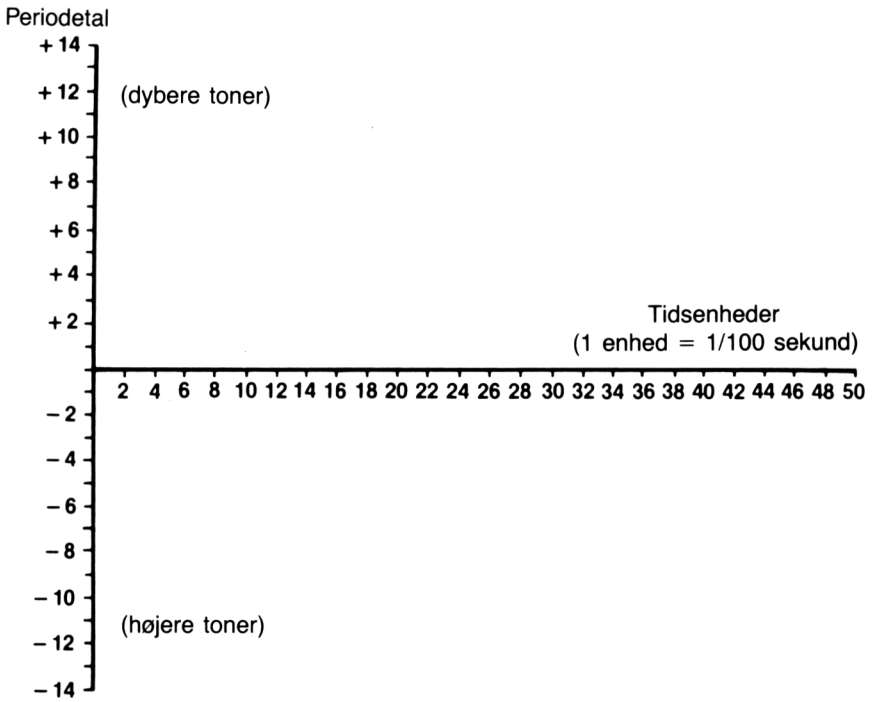

Fig. 11.25 Planlægningsark til tone-envelope.

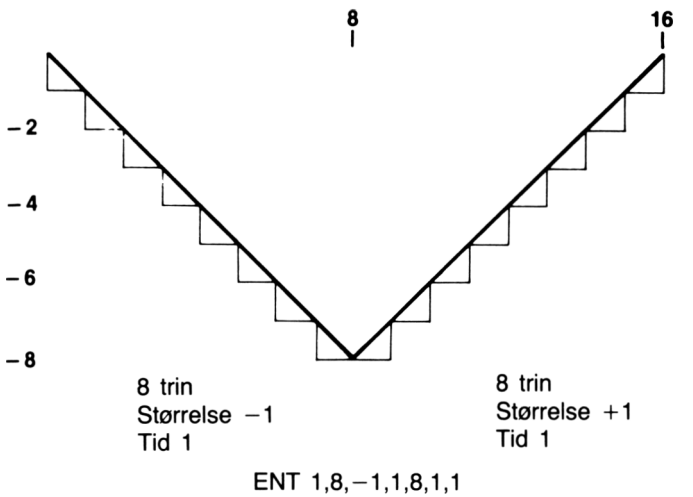

Fig. 11.26 Kurve og plan for en enkelt tone-envelope.

```
10 ENT 1,8,-1,1,8,1,1
20 SOUND 2,478,50,7,0,1,0
30 FOR n=1 TO 2000:NEXT
40 FOR j=1 TO 8
50 READ tone%
60 SOUND 2,tone%,40,7,0,1,0
70 NEXT
100 DATA 478,379,426,638,638,426,379,478
```

Fig. 11.27 Sådan lyder envelopen ovenfor.

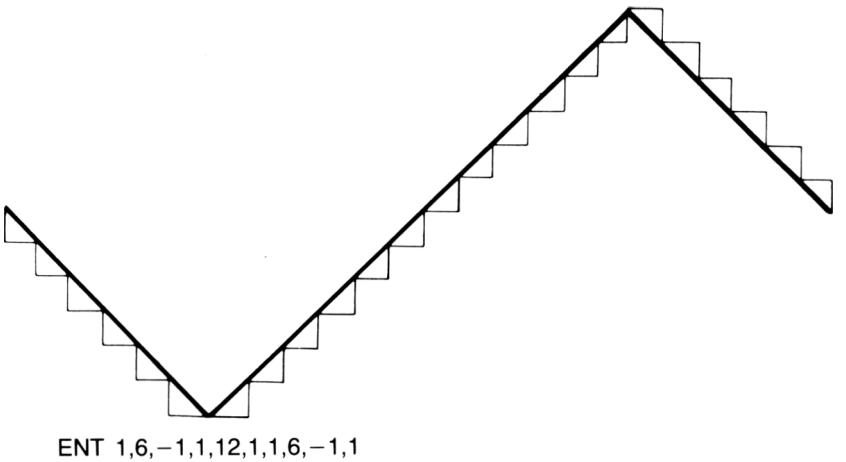

Fig. 11.28 Et vibrato-mønster til tone-envelope.

For at få vibrato-effekter i musik, skal den enkelte tone stige og falde, som fig. 11.28 viser. Det er en ret langsom vibrato, og til mange formål ville det lyde bedre med en hurtigere variation i tonen. ENT er naturligvis den ordre, der bruges til »ENvelope/Tone«, (ligesom ENV bruges til »ENvelope/Volume«). I ENT skrives envelopenummeret som −1, selvom der *ikke* bruges minus i selve SOUND-ordren. Minusset bevirker at envelopen gentages. Hvis vi havde brugt 1, ville vi kun høre vibrato i tonens start. Med ENT −1 får vi vibratoen til at vare tonen ud.

Kombinationen af støj sammen med tone-envelope giver nogle spændende lydeffekter. Et flot eksempel kan du lave på følgende måde. Brug programmet ovenfor med disse ændringer i SOUND: lad perioden være 1 og lad støjperioden være 7, og sæt tonelængden til 1200. Kør så programmet og hør et damplokomotiv arbejde sig op ad en kraftig stigning! Du kan naturligvis også bruge volume-envelopes og tone-envelopes sammen, og få næsten alle tæn-

```
10 ENT -1,6,-1,1,12,1,1,6,-1,1
20 SOUND 2,478,200,7,0,1,0
```

Fig. 11.29 En vibrato tone-envelope. Læg mærke til brugen af ENT –1 for at få envelopen til at gentages.

kelige lyde frem. Når du udforsker lydsystemet er det vigtigt, at du går systematisk frem. Optag de lyde du laver, og gem de kurver du tegner. På den måde kan du efterhånden oparbejde et helt »lyd-bibliotek«, som du kan bruge når du laver dine egne programmer. Eller måske kan de bruges i andres programmer – husk at nogle af databladene gerne vil have tilsendt programmer, – nogle gange kan man endda få betaling for dem.

Synkronisering

Vi er ved at være færdige. Hvis Amstrad's lydsystem virkelig skulle beskrives grundigt, kunne der skrives en hel bog på denne størrelse om det. Der er nogle ting, som vi simpelt hen ikke kan nå at behandle her – specielt mulighederne i SQ-ordren. Men inden vi slutter skal vi se på *synkronisering*. Når der kommer toner i lydkøerne i de forskellige kanaler, er det ret vanskeligt at sikre sig, at det hele kommer til at lyde, som det skal. Det er praktisk taget umuligt på nogle computere. Det gælder især hvis du vil have to toner til at lyde samtidigt, og derefter vil have stilhed i den ene kanal, mens den anden kanal fortsætter med en ny tone osv ... Amstrad har et synkroniseringssystem, der kan få dette til at ske. Synkroniseringen sker ved at bruge kanalkoderne, og fig. 11.30 viser mulighederne. Ligesom der er koder, der sender tonerne til de tre kanaler, er der tre koder, som laver synkronisering, en kode, som får en tone til at vente i køen og en kode, der »sletter« køen – en hurtig måde at få stilhed på. De forskellige koder kan bruges samtidig, ved at koderne lægges sammen.

Kode	Virkning
1	Send til kanal A
2	Send til kanal B
4	Send til kanal C
8	Synkronisér med kanal A
16	Synkronisér med kanal B
32	Synkronisér med kanal C
64	Vent
132	»Slet« lydkø

Fig. 11.30 Kanalkoderne – også synkronisering.

Fig. 11.31 viser princippet i dette *). Linie 10 er en SOUND-ordre. Normalt ville den straks lyde, men her er kanalkoden 33, og det betyder, at den holdes tilbage. Kanalkoden består af et 1 (send lyden til kanal A) +32 (synkronisér med kanal C), og resultatet er at køen holdes tilbage indtil kanal C lyder. Efter pausen i linie 20, kommer vi til SOUND i linie 30, som netop gælder kanal C. Her er koden 12, som er dannet af 4 (kanal C) og 8 (synkronisér med kanal A). Bemærk at *begge kanalkoder skal indeholde synkroniseringskoder*, hvis de to kanaler skal synkroniseres med hinanden. Programmet virker altså sådan, at begge toner lyder samtidigt efter pausen.

```
10 SOUND 33,478,300
20 FOR n=1 TO 2000:NEXT
30 SOUND 12,379,300
```

Fig. 11.31 Et lille eksempel på synkronisering.

Hvis det ser lidt skræmmende ud, så husk at der kan laves masser af effekter, hvor du ikke har brug for synkronisering. Du kan let lave et sæt kanalkoder og perioder fra en data-linie, hvor du bruger periode 0 som stilhed. Med mindre du virkelig vil dyrke mulighederne for at lave musik, så kan du godt klare dig uden synkroniseringen. Ikke desto mindre synes jeg det er imponerende, at en computer, der sælges så relativt billigt, har den slags muligheder. Måske vil du bruge dem senere, – de forsvinder ikke, fordi du ikke bruger dem.

*) Hvis du ser på fig. 11.8 kan du se at koderne 3, 5, 6 og 7 er fremkommet ved addition. O.A.

Appendix A:

Redigering

At *redigere* vil sige at ændre i noget, der allerede står på skærmen. Du kan selvfølgelig slette en karakter med DEL-tasten, eller du kan skrive en forkert linie om. Du kan også slette en hel række linier med ordren DELETE (slet), som fx DELETE 10-100. Men at redigere vil sige, at ændre en enkel ting i en linie, uden at skulle ændre alt det andet. Enhver del af en linie – også linienummeret – kan ændres ved redigering. Redigeringen kan foretages på tre måder:

(a) Mens en linie skrives, før ENTER bruges.
(b) Senere, efter linien er indlæst, men før programmet har været kørt.
(c) Når der kommer en fejlmelding under kørslen.

Vi tager dem i rækkefølge:

(a) Før en linie er indlæst kan alle linie-redigeringsordrerne nedenfor bruges. Redigeringen afsluttes ved at taste ENTER.
(b) Når en linie er indlæst uden at programmet har været kørt, kan alle redigeringsordrerne nedenfor bruges.
(c) Når programmet stoppes med en fejlmelding, kommer den linie, hvor fejlen er fundet, frem på skærmen med markøren ved linienummeret. Fejlen kan rettes ved linie-redigering.

Redigeringsmetoder

Amstrad har et ret enestående redigeringssystem, idet der altid kan redigeres på to måder. De fleste computere har enten linie-redigering eller skærm-redigering. Linie-redigering vil sige at du kan kalde den pågældende linie frem på skærmen, med en ordre som fx EDIT 200 (»rediger linie 200«). Markøren kan så flyttes hen over linien, og fejlen kan rettes. Ved skærmredigering kan enhver linie på skærmen rettes ved, at du flytter markøren hen til den og retter fejlen. Mange sværger til én af metoderne og vil ikke bruge andre. Amstrad har begge systemer, og alle kan få det, som de vil have det!

Redigeringsordrerne

1. Linie-redigering

Hvis du ikke allerede har en linie på skærmen, med markøren klar ved den (som ved en fejlmelding), så brug EDIT til at få en linie frem. Husk at der skal være mellemrum mellem »T« i EDIT og første ciffer i linienummeret. Du vil så få linien frem øverst på skærmen, med markøren over første ciffer i linienummeret. Du kan nu flytte markøren med markørtasterne (med pilene). Du kan slette en karakter ved at flytte markøren hen over den og taste CLR. Ved at holde CLR nede, kan du slette alt det, der står til *højre* for markøren. På den anden side kan du slette alt det, der står til *venstre* med DEL-tasten. Du kan indsætte en karakter på markørens plads ved blot at taste den ind. Brug ENTER når redigeringen er slut. Husk at du kan redigere en linie på denne måde, mens du skriver den. Hvis du har skrevet

PRINT DETTE ER SLUTNINGEN"

og du pludselig opdager, at du har glemt de første gåseøjne, så kan du bruge venstre-pilen til at flytte markøren hen over »D« i DETTE og skrive de manglende gåseøjne ind, og så taste ENTER. Det er vigtigt at bemærke at du ved linie-redigering *kan taste ENTER så snart fejlen er rettet,* du behøver ikke at flytte markøren hen til enden af linien *).

2. Kopi-redigering

Kopi-redigering er Amstrad's udgave af skærm-redigering. Med denne metode laves en kopi af linien, men du kan udelade og erstatte de dele du vil. Enhver linie på skærmen kan kopi-redigeres – selv direkte ordrer. Tryk SHIFT ned og hold den nede. Tryk nu på en af markørtasterne, – nu deler markøren sig, således at den ene styres med SHIFT + markørtasterne og den anden styres på normal vis. Nu flytter du markøren op til den linie, du vil redigere (med SHIFT nede). Nu skal du slippe markørtasten og trykke på COPY i stedet for (her betyder det ikke noget, om du holder SHIFT nede).

---

*) Der er sikkert nogen, der har undret sig over, at Amstrad ikke har den mulighed, at man kan skrive ny tekst hen over gammel. Det fortæller hverken manualen eller Sinclair om. Men det kan man nu godt, så det må være en forglemmelse. Man kan få denne redigeringsmåde frem ved at taste CTRL-TAB. Funktionen slettes ved endnu et tryk på CTRL-TAB.
En anden finesse, som heller ikke er omtalt i manualen, er den måde CTRL og markørtasten kan bruges sammen på. Hvis du har en programlinie, der fylder fx tre skærmlinier, kan du få markøren til at springe rundt i disse tre linier på følgende måde: Højre-pilen får markøren ud til højre i den linie den står i, venstre-pilen springer tilsvarende til venstre. Pilen, der peger op, får markøren til at springe til første karakter i programlinien, – pilen, der peger ned, får tilsvarende markøren til at springe til sidste karakter i programlinien. Prøv! O.A.

Nu kan du lave en kopi af linien – kopien kommer frem der hvor »den anden markør« står. Kopieringen starter der hvor du placerede markøren. Hvis du vil springe en del af en linie over, skal du blot bruge markørtasten (og SHIFT!) i stedet for COPY. Tryk på ENTER når du har kopieret det, du ønsker. Hvis du vil kopiere hele linien, *skal du kopiere det hele inden du taster ENTER.* Hvis du taster ENTER midt i en linie, så bliver kun den del af linien kopieret. Det er en meget vigtig forskel mellem kopi-redigering og linie-redigering.

Det er en meget alsidig redigeringsmetode. Du kan fx ændre et linienummer. Hvis du har linie 200 på skærmen og du gerne vil have lavet den om til fx linie 1000. Skriv da først 1000 og tryk SHIFT ned og flyt markøren op i linie 200 – lige efter linienummeret. Nu kan du kopiere linien med COPY-tasten. Tryk derefter på ENTER og du har en linie 1000, som er magen til linie 200. Skriv nu 200 og tryk ENTER og den gamle linie 200 slettes fra hukommelsen.

Du kan også kopiere noget, der aldrig har haft et linienummer. Hvis du er midt i et program og fx skriver FOR N=1 TO 200:NEXT og glemmer linienummeret, så behøver du ikke skrive det om igen. Du kan blot skrive linienummeret, og placere kopimarkøren ovenpå »F« i FOR og kopiere linien. Når du er nået til enden af linien taster du ENTER og problemet er løst! En anden stor fordel ved denne metode, som også bruges af BBC-computeren er, at du kan kopiere en del af en linie ind i en anden. Hvis du fx finder ud af, at den forsinkelse du brugte i linie 400 også skal bruges i linie 700, så er det let at klare. Du skal bare sørge for, at begge linier er på skærmen. Du kopierer så linie 700, indtil du når til det sted hvor du vil have forsinkelsen, så flytter du markøren op i linie 400 og indkopierer den del du skal bruge, derefter vender du tilbage til linie 700 og laver kopien færdig, (hvis der mangler noget). Tryk på ENTER og du er færdig.

Aflusning

I EDB-sprog kaldes fejl i programmer for *lus,* og den der laver fejlene kaldes naturligvis for – en programmør. Dine programmer kan indeholde mange forskellige slags lus, og disse dukker op med fejlmeldinger, når du kører programmet. Nogle meldinger er ret nemme at gennemskue. »Line does not exist« (»linien findes ikke«), betyder fx at du har brugt en ordre som GOTO 1000 eller GOSUB 1000, og har glemt at skrive linien. Det kan også ske, hvis du prøver DELETE 1000, hvis linie 1000 ikke findes, eller har THEN 1000 ELSE 2000 efter et IF et sted.

Den mest almindelige fejlmelding er »Syntax error«. Det betyder at du har brugt nogle af de reserverede BASIC-ord forkert. Måske har du stavet for-

kert, som PRIBT i stedet for PRINT. Du kan have glemt et komma eller en parentes, et semikolon, eller have brugt semikolon i stedet for et kolon. Maskinerne kan ikke vide, hvad du har tænkt på, de kan kun gøre præcis det, du fortæller dem, de skal. Hvis du ikke har brugt BASIC på den måde, som maskinen forventer, får du også en »Syntax error«. En anden hyppig fejlmelding er »Improper argument« (»fejlagtigt argument« – dvs. parameter- eller variabel-værdi). Det er normalt en eller anden dum fejl, hvor et forkert tal er brugt. Fx har du måske brugt TAB (300). Selvfølgelig vil du ikke finde på at skrive TAB (300) i et program, når du har læst denne bog, men måske har der stået TAB (n), og n er så blevet til 300 ved et uheld. Alt hvad der bruges tal til, som fx MID$, LEFT$, RIGHT$, INSTR, STRING$ osv. kan få forkerte tal, og det vil give »Improper argument«. Hvis du bruger et negativt tal i SQR(n), eller et negativt tal eller nul i LOG(n), eller lignende matematiske umuligheder, får du samme fejlmelding. Det skulle ikke være svært at klare disse fejl, idet maskinen jo fortæller i hvilken linie fejlen står.

Der kan komme mange fejl i programmerne, også selvom du tager dem direkte fra en program-listning i et blad. Stort set er programmerne i databladene pålidelige, men nogle gange er udskrifterne lavet på en måde, der kan give misforståelser. De mest typiske fejl opstår, når der bruges I (stort i) eller l (lille L) som variabelnavn, eller når printeren ikke kan skrive nullet med en streg igennem. Selvfølgelig er det forvekslingen af O og 0, der giver de fleste problemer. En linie som:

```
IFM=10ORJ=4ORD=2
```

kan også give en masse problemer, og et af de engelske datablade har nærmest specialiseret sig i den slags listninger. Hvis det blev skrevet sådan:

```
IF M=10 OR J=4 OR D=2
```

ville det udelukke en del misforståelser. Nogle gange kan det være nødvendigt at skrive meget tæt, for at få plads til programmet i computerens hukommelse, men det er også den eneste acceptable undskyldning.

Selv om du har rettet alle »Syntax error« og »Improper argument« meldinger, kan det godt ske at programmet alligevel ikke kører, som det skal. Her har Amstrad det, som enhver moderne computer burde have – den har et væld af muligheder for at opspore fejl. En af de mest effektive er ESC-tasten. Som du ved kan du stoppe maskinen i det den er i gang med, ved at taste ESC én gang, og starte den igen, ved at trykke på en anden tast. Som vi skal se senere, er det en vældig god måde at finde lus i grafik-programmer på, i andre programmer er det mere effektivt at taste ESC to gange. På den måde stopper programmet, og det udskrives i hvilken linie kørslen var nået til. Men du

ved måske ikke at du kan kalde variabelværdierne frem (med ? eller PRINT), og endda ændre variabelværdierne, mens programmet er stoppet, og at du derefter kan få programmet til at køre videre med GOTO. Lad os fx sige, at du kører et program, der tæller ret langsomt, og at du ret tidligt taster 2 gange ESC. Så stopper programmet og melder »Break in 40«. Det linienummer, 40, er vigtigt, for du kan starte programmet igen i den linie, *forudsat* at du ikke har redigeret, slettet eller tilføjet noget i programmet. Hvis tællervariablen er n, så skriv ?n og ENTER. Så får du n's aktuelle værdi oplyst. Skriv nu (fx) n=998 og ENTER. Skriv så GOTO 40 (eller hvor du nu stoppede) og ENTER. Nu vil programmet tælle videre, – men fra 998! Det er en glimrende måde at finde ud af, hvad der sker ved slutningen af en lang løkke. Ellers kan du komme til at bruge ret lang tid, hvis du skulle vente på at programmet selv nåede frem til enden af løkken. Ja, du kan endda få dette til at ske automatisk! Vi har allerede set på ordren ON BREAK GOSUB. Med den kan du få en subrutine til at køre, hver gang ESC tastes to gange. I dette tilfælde kunne subrutinen oplyse den aktuelle værdi af variablerne og give dig mulighed for at indtaste nye, og så køre videre!

I grafikprogrammer kan du bruge ESC (én gang), for at finde ud af, hvad der går galt. Når du taster ESC stopper programmet (bortset fra blinkende karakterer), og på den måde kan du se i hvilken rækkefølge tingene sker på skærmen. Ved at trykke på en anden tast, kan du sætte programmet i gang igen, – og sådan kan du blive ved med at stoppe programmet med ESC og sætte det i gang igen, og på den måde følge med i, hvordan billedet ændrer sig. Hvis det ikke er nok, kan du midlertidig indsætte en pauseløkke i dit grafikprogram, så det kører i »slow motion«. Og så bruge ESC i særligt vanskelige dele af programmet.

TRON – TROFF

En af de mest drilske fejl er, at programmet kører forkert – uden at lave fejlmeldinger. Det kan tyde på dårlig planlægning, men det kan også skyldes en forglemmelse, og her er ON BREAK GOSUB-metoden god at bruge, fordi du hele tiden kan få variabelværdierne oplyst, og så køre videre. Men nogle gange er der brug for at spore fejlene på på en anden måde. Hvis dit program indeholder en masse IF ... THEN ... ELSE linier, kan det ofte ske at de ikke kører, som du ville have dem til. I den slags tilfælde har Amstrad to ordrer, der virkelig giver en god hjælp, TRON og TROFF.

TRON betyder TRACE ON (»trace« betyder »spor«), og ordren virker på den måde at linienumrene skrives ud på skærmen, efterhånden som de køres. Linienumrene sættes i skarpe parenteser, og de skrives ud øverst på skærmen

foran programmets udskrifter. Prøv at taste TRON (ENTER) og kør et langsomtkørende program. TRON kan være specielt nyttig, hvis du ikke rigtigt ved, hvad programmet foretager sig, og det er en stor hjælp, når du skal finde ud af, hvad der kører forkert i en løkke. Husk at du kan bruge TRON sammen med andre aflusnings-ordrer. Du kan fx stoppe programmet, ændre variablerne, og fortsætte, mens TRON viser dig hvilke linier der køres. Når du er færdig med at bruge TRON skriver du TROFF (»trace off«) (ENTER).

Appendix B:

Tilslutning af printer

Amstrad er nem at tilslutte til alle de printere, der har et Centronics parallelt interface. Du skal bruge et passende kabel, – det kan du købe der hvor du købte din Amstrad. Den ene ende af kablet passer på computeren, og den anden ende passer i printerens standard-stik. Fordelen ved at maskinen er »Centronics-kompatibel« er, at du kan bruge næsten alle de printere, der er på markedet. Du kan fx bruge den meget anvendte Epson RX-50, eller du kan lave skønskrift med en skrivehjulsmaskine, eller du kan plotte grafik med den lille effektive Tandy CGP-115 printer/plotter. Du er altså ikke tvunget til at skulle bruge Amstrads printer – mange andre hjemmecomputere kan jo kun bruge printere af samme mærke som computeren. Du kan få printere i alle prislag med Centronics interface.

Ifølge manualen udsender maskinen to koder ved hvert linieskift. Den ene er linieskift ASCII 10, og den anden er vognretur ASCII 13. De fleste printere kan indstilles sådan, at de laver linieskift, *enten* når linieskift *eller* vognretur-karakteren udsendes. Omstillingen sker med en omskifter på printeren. Hvis du opdager at din printer skriver det hele i en linie, eller at der kommer dobbelte linieskift hver gang, så plejer det at være nok at omstille omskifteren. Men på alle mine printere fik jeg dobbelt linieafstand, uanset om omskifteren stod i den ene eller den anden stilling, – hvilket må betyde at Amstrad udsender to linieskiftkarakterer. Hvis du bruger en Epson, kan problemet løses ved at sætte linieafstanden lavere end normalt (7/72 tomme). Det kan gøres med ordren

```
PRINT#8, CHR$(27);"A";CHR$(7)
```

når printer er tilsluttet og tændt. Dermed er problemet løst, *forudsat at Amstrad og printeren bruger samme antal karakterer pr. linie.* Hvis du indstiller printeren til 40 karakterer pr. linie, skal du indstille computeren til det samme med WIDTH 40. Heldigvis kan man justere de fleste printere så de passer, – du må selv se efter i printermanualen.

Appendix C:

Tryllerier med tasterne

En af de vigtigste egenskaber ved de nyere computere er, at de har programmerbare taster. Det vil sige, at nogle at tasterne kan programmeres til at udføre bestemte funktioner eller ordrer, blot ved at trykke dem ned. I stedet for at skrive LIST (ENTER) hver gang du vil have en programudskrift, kan du nøjes med at trykke en tast ned. Eller du kan programmere en tast til at skrive PRINT TAB(2), hver gang du skal bruge det i en tekst.

Selvom der er mange computere, der har programmerbare taster, så er det kun få af dem, der kan programmeres relativt let. Der er nok nogle brugere af nogle af de andre maskiner, der har en mistanke om, at disse taster mest er til pynt. På Amstrad kan du programmere op til 32 taster med almindelige BASIC-ordrer. Der er dog visse begrænsninger. Der må ikke være knyttet mere end 32 karakterer til en tast, og summen af karakterer må ikke overstige 120 i alt. I praksis betyder disse begrænsninger vist ikke noget.

Lad os sige at du vil omdefinere en tast. Det er det smarteste at omdefinere en, som du normalt ikke bruger. Du kan bruge ASCII-koderne fra 128 til 159 til dette formål, og de første tretten er allerede knyttet til nogle taster, nemlig tasterne på tal-blokken til højre for tastaturet. I Appendix III Page 15 i manualen, kan du se koderne for disse taster, og der kan du se at tast 0 har koden 128. For at få den til at udskrive LIST(ENTER), skal du programmere den således:

```
KEY 128,"LIST"+CHR$(13)
```

og taste ENTER. CHR$(13) sørger for at LIST automatisk efterfølges af ENTER, når tasten bruges. Tast 0 på talblokken – og du vil få en programlistning!

Hvad nu hvis du vælger en ASCII-kode, som fx 156, der ikke er tilknyttet en tast? Prøv det – skriv:

```
KEY 156,"PRINT :PRINT"+CHR$(13)
```

Nu kan du få *en anden tast* til at give ASCII 156. Tag fx tasten med den skarpe parentes, øverst ved siden af ENTER. Skriv:

```
KEY DEF 17,1,156
```

og tast ENTER. Når du trykker denne tast ned, udskrives PRINT:PRINT og udføres derefter. »17« i denne ordre er tastens INKEY-kode, og alle taster kan omdefineres på denne måde. Hvis maskinen slukkes, eller hvis CTRL+SHIFT+ESC bruges får tasterne deres normale funktioner tilbage, men du kan godt bruge NEW uden at få slettet de nye definitioner. Hvis du ikke vil have ordren til at blive udført med det samme, skal du blot udelade CHR$(13). Når du fx skal bruge PRINT TAB(, skal du jo først have indsat et tal, lukket parentesen og skrevet noget tekst før indlæsningen *).

*) Se endvidere Appendix E her i bogen. O.A.

Appendix D:

Fejl-subrutiner

Vi har tidligere i denne bog set på *fælder.* Vi bruger fælder til at sikre os, at de data, der indtastes, giver mening. En fælde kan fx se sådan ud:

```
60 IF LEN(a$)=0 THEN GOTO 1000:GOTO 50
```

og vi skal have specielle fælder til de forskellige typer fejl. Det kan være et ret stort og kedeligt arbejde, og der vil ofte være fejl, du alligevel ikke får fanget. Her er Amstrad en af de meget få maskiner, der har en »fejl-ordre«, ON ERROR GOTO (»ved fejl så gå til«). Fig. D.1 giver et noget søgt eksempel – et mere realistisk eksempel ville fylde meget mere. I dette eksempel måles længden af et ord, og derefter tages længdens reciprokke værdi (længden divideres op i 1). Det kan ikke gøres, hvis længden er nul, og ON ERROR GOTO bruges her som fælde, hvis dette skulle blive forsøgt. Det kunne fx ske, hvis ENTER blev trykket ned, uden at der var blevet indtastet noget inden. Normalt ville der komme en fejlmelding i dette tilfælde, og programmet ville stoppe. Den store fordel ved ON ERROR GOTO er, at programmet ikke stopper, når der findes en fejl, – det går ned i en subrutine i stedet. I dette eksempel udskriver subrutinen en meddelelse, og vender derefter tilbage til linie 20. Det er en dejlig enkel løsning på problemet, men den kræver noget programmeringserfaring. For hvis programmet stadig skulle indeholde syntax fejl eller lignende, så vil de også aktivere subrutinen, – og det vil jo

```
10 ON ERROR GOTO 1000
20 PRINT"Skriv et ord!"
30 INPUT a$
40 L=LEN(a$)
50 PRINT 1/L
60 END
1000 PRINT"Ordet har ingen bogstaver!"
1010 RESUME 20
```

Fig. D.1 Brugen af ordren ON ERROR GOTO.

virke ret forvirrende, hvis programmet pludselig springer hen i en anden linie.

Fig. D.2 viser et andet eksempel. Linie 20 sikrer, at programmet vil springe til linie 1000, hvis der skulle være fejl i programmet. I programmet læses en række tal og kvadratroden af disse tal sættes på strengform. Fejlen er, at et af tallene er negativt. Computeren kan ikke udregne kvadratroden af et negativt tal, fordi det er et »imaginært« tal. Kvadratet af et positivt eller negativt tal er nemlig altid positivt, så der kan ikke være reelle tal med en negativ kvadratrod. I mange sammenhænge kan vi imidlertid godt tillægge et sådant tal en betydning, fx kan det stå for længden af en linie, der står vinkelret på en anden linie. Når -2 læses, får linie 20 programmet til at springe til linie 1000. I denne linie tages den absolutte værdi af -2, og kvadratroden af dette tal findes. Bogstavet »j« tilføjes for at markere at dette var et negativt tal. Linie 1010 sætter programmet i gang igen, og den nye værdi af y$ udskrives sammen med et »j«.

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR n=1 TO 5
40 READ x:y$=STR$(SQR(x))
50 PRINT"Tal";x;"  Kvadratrod";y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 y$=STR$(SQR(ABS(x)))+"j"
1010 RESUME 50
```

Fig. D.2 Et andet eksempel på brugen af ON ERROR GOTO. Programmet går automatisk tilbage til hovedprogrammet.

```
10 CLS
20 ON ERROR GOTO 1000
30 FOR n=1 TO 5
40 READ x:y$=STR$(SQR(x))
50 PRINT"Tal";x;"  Kvadratrod";y$
60 NEXT
70 DATA 5,4,3,-2,2
80 END
1000 y$=STR$(SQR(ABS(x)))+"j"
1005 PRINT"Linie ";ERL;" -- error ";ERR
1010 RESUME 50
```

Fig. D.3 Udskrift af fejlkode og linienummer i en ON ERROR GOTO subrutine.

Fig. D.3 viser en variation af dette, hvor du får at vide hvilken fejltype programmet er stødt på og i hvilken linie. I linie 1005, giver ERL linienummeret, og ERR giver fejlkoden. Fejlkoderne finder du i manualen. Det kan være en god linie at bruge, når du tester et program, for hvis der dukker fejl op du ikke har tænkt på – så får du en melding om det på skærmen.

PRINT USING

Det sidste vi skal se på er PRINT USING (»PRINT brugende«), den er lidt svær, så derfor tager vi den først nu. PRINT USING bruges til at få udskrifter til at følge et bestemt mønster. Efter USING skal der være en streng, som kan defineres på forhånd. Strengen skal tilpasses det du ønsker at udskrive.

Det lyder sikkert uforståeligt, så se på eksemplet i fig. D.4. I dette eksempel bliver a$ defineret som ###.##. Det betyder at ethvert tal, der udskrives med USING a$ vil blive udskrevet med tre cifre før punktummet og to efter. I datasprog er a$ en »formatterings-streng». Det er langt den vigtigste anvendelse af PRINT USING, men manualen angiver også andre anvendelser.

```
10 CLS
20 PRINT TAB(15)"PRINT USING"
30 PRINT:PRINT
40 n=140.2716
50 PRINT:PRINT"n er ";n
60 a$="###.##"
70 PRINT:PRINT"- med PRINT USING, er det
: ";USING a$;n
80 PRINT:PRINT"Momsen på ";n;" er: ";22*
n/100
90 PRINT"Det ser pænere ud sådan her: ";
USING a$;n*22/100:PRINT
```

Fig. D4 Et simpelt PRINT USING eksempel.

Appendix E:

Danske tegn m.m.

AMSTRAD CPC 464 leveres i Danmark i en udgave uden danske tegn. Hvis du ønsker det, kan du få din Amstrad ombygget for et tillægsbeløb, men det er på bekostning af bl.a. de skarpe parenteser, og dem *skal* du bruge, hvis du senere hen vil programmere i PASCAL. Derfor er det måske værd at overveje en anden løsning i stedet for.

I kapitel 9 så du, hvordan du ved hjælp af et symbolnet kunne definere dine egne karakterer. I fig. E.1 ser du et lille program, som giver Amstrad de danske tegn. De er placeret på nogle bestemte taster, der også skulle gøre det muligt umiddelbart at få de danske tegn frem på en printer. Som du kan se af programmet bruges ASCII-koderne 91 (Æ), 92 (Ø), 93 (Å), 123 (æ), 124 (ø) og 125 (å) til de anførte bogstaver. Måske skal de lægges anderledes, når de bruges sammen med nogle printere. Hvis du ikke skal tilslutte en printer, men blot gerne vil have danske tegn på skærmen, kan du jo let lægge tegnene andre steder. Det kan også være, at du vil lave lidt om på tegnenes udseende – prøv!

```
10 SYMBOL AFTER 91
20 SYMBOL 91,62,122,216,220,248,218,222
30 SYMBOL 92,124,206,222,254,246,230,124
40 SYMBOL 93,56,0,124,198,254,198,198
50 SYMBOL 123,0,0,118,26,126,216,110
60 SYMBOL 124,0,0,118,204,214,102,220
70 SYMBOL 125,48,0,120,12,124,204,118
```

Fig. E.1 Med dette program kan du få danske tegn på din Amstrad.

I appendix C så du, hvordan du kunne programmere tasterne på Amstrad'en. Du kan lægge forskellige ordrer på talblokkens taster – det er ret nemt at gøre, men det kan godt være svært at huske, hvilke taster du bruger til hvad. Der er imidlertid en anden mulighed, som giver nogle fordele. Du kan nemlig programmere tasterne på selve tastaturet, således at de, når de bruges

sammen med CTRL-tasten, kan kalde dine ordrer frem. Det har to fordele, dels kan du bruge talblokken normalt, og dels kan du lægge de indprogrammerede ordrer på nogle taster, der kan minde dig om hvilken ordrer, der er knyttet til dem. I fig. E.2 kan du se et program som lægger en række ordrer på bestemte taster. REM-linierne i programmet angiver hvilke taster de forskellige ordrer ligger på. Fx ligger EDIT selvfølgelig på E, LIST ligger på L og CLS ligger på C. De tre modes er naturligvis lagt på talblokkens 0, 1 og 2. Prøv at programmere de ordrer ind, som du synes er mest hensigtsmæssige. Programmets linienumre passer sammen med programmet i fig. E.1, så du kan indtaste hele programmet på én gang. Programmet slutter med at slette det hele med NEW, men programmet slettes *efter* tasterne er blevet omdefinerede og programmerede.

```
80 REM A
90 KEY DEF 69,1,97,65,141
100 KEY 141,"AUTO "
110 REM C
120 KEY DEF 62,1,99,67,142
130 KEY 142,"CLS"+CHR$(13)
140 REM D
150 KEY DEF 61,1,100,68,143
160 KEY 143,"DATA "
170 REM E
180 KEY DEF 58,1,101,69,144
190 KEY 144,"EDIT "
200 REM L
210 KEY DEF 36,1,108,76,147
220 KEY 147,"LIST"+CHR$(13)
230 REM 0
240 KEY DEF 15,1,128,128,151
250 KEY 151,"MODE 0"+CHR$(13)
260 REM 1
270 KEY DEF 13,1,129,129,152
280 KEY 152,"MODE 1"+CHR$(13)
290 REM 2
300 KEY DEF 14,1,130,130,153
310 KEY 153,"MODE 2"+CHR$(13)
320 NEW
```

Fig. E.2 Et eksempel på programmering af tasterne, hvor du skal bruge CTRL-tasten samtidig, for at få ordrerne frem.

Du kan optage programmet på bånd, og indlæse det hver gang du tænder for computeren, – med SPEED WRITE 1 tager det ikke mange sekunder. Husk at hvis du vil rette i programmet, skal du bruge LOAD" og ikke RUN", når du indlæser det fra båndet. God fornøjelse!

Oversætteren.

STIKORDSREGISTER

- Hvis du ikke har prøvet at programmere før, kan denne bog lære dig det.

- Hvis du har prøvet at arbejde med en af de lidt ældre hjemmedatamater, så vil denne bog vise dig en helt ny måde at programmere på.

- AMSTRAD har en langt mere avanceret BASIC, end de lidt tidligere generationer af hjemmedatamater.

- Bogen indeholder en mængde programeksempler som alle er afprøvet og trykt direkte fra printudskrift – en garanti for, at de virker efter hensigten.

ISBN 87-571-0909-5