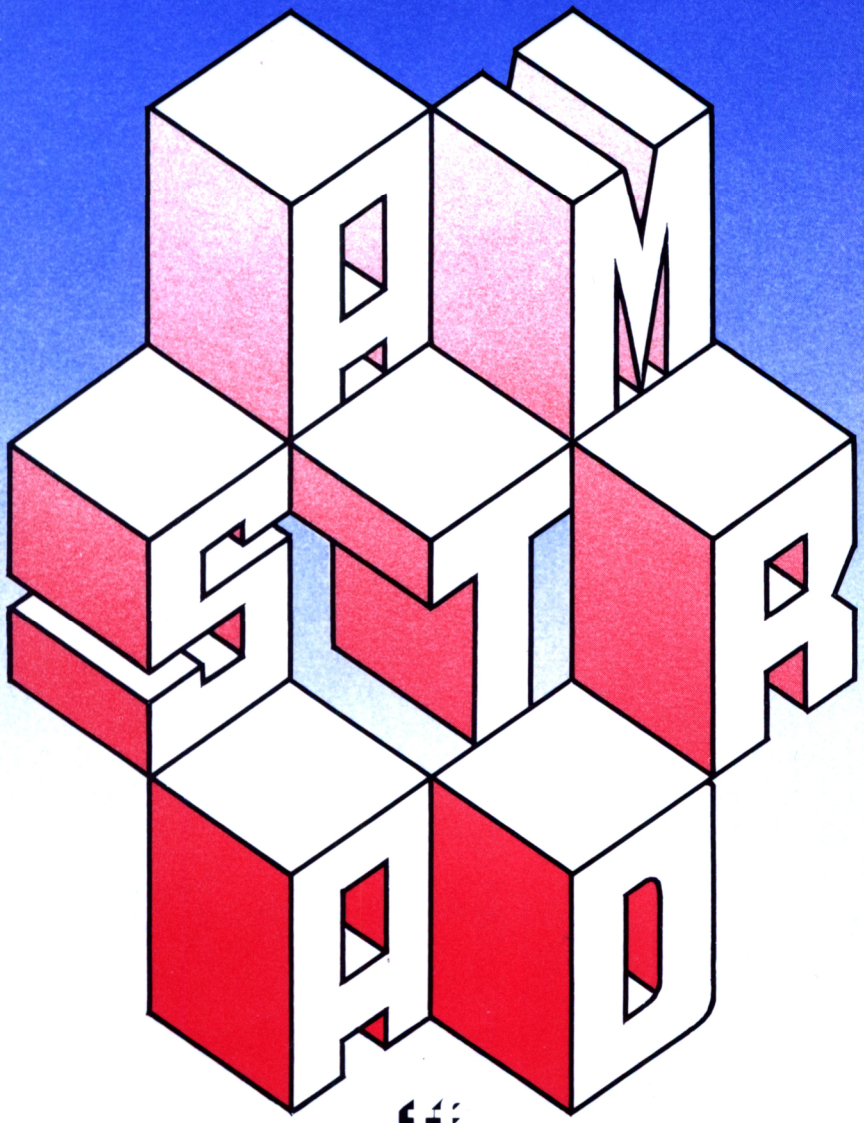
Programudvikling på AMSTRAD

Simon Lane
David Lawrence



PROGRAMUDVIKLING PÅ AMSTRAD

DAVID LAWRENCE
SIMON LANE

PROGRAMUDVIKLING PÅ AMSTRAD

Oversat af Peter Wiwe

teknisk forlag as

Programudvikling på Amstrad
Originaltitel: The Working Amstrad
Udgivet af Sunshine Books, England, 1984

Oversat fra engelsk af Peter Wiwe

Forlagsredaktion: Henrik E. Hansen
Omslag: Anders Holm og Henrik Juul Pedersen
Sats: Fotografik, København
Montage: Montagestuen
Tryk: Teknisk Forlag A/S, København
ISBN 87-571-0913-3

Oversætterens forord

Farver

Programmerne i denne bog er beregnet til en farveskærm. Det betyder at programmerne er beregnet til at køre i *mode 1* (40 karakterer (tegn)) per linie, dels for at have et passende antal farver til rådighed, og dels fordi de helt små bogstaver er svære at læse på farveskærmen. Hvis du bruger en grøn monitor, skal du måske justere nogle af farvekommandoerne, da nogle farver *kan* være svære at se på den grønne skærm.

Dataoptager – diskettestation

Bogens programmer er beregnet til brug sammen med den indbyggede dataoptager. Men det skulle ikke være noget problem for læsere, der bruger diskettestation (enten som »løs« enhed sammen med CPC 464, eller for CPC 664/CPC 6128-brugere), at ændre programmerne så de passer sammen med diskettestationen.

Programlisterne

Programmer udskrevet på en matrixprinter kan nogle gange være svære at læse. Derfor er der i denne bog brugt en skrivehjulsprinter, hvilket giver nogle let læselige programudskrifter. Det betyder, at der er et par tegn, der ser anderledes ud end de gør på skærmen/tastaturet. Således skrives exponentialtegnet '^' (på samme tast som '£') og '\' skrives '|'.

Oversættelsen

Oversættelsen er søgt lagt tæt op ad terminologien i den danske brugervejledning. Dog er der mindre forskelle – f.eks. kaldes 'indicerede variab-

le' for 'arrays'. Navnene på tasterne skrives <ENTER>, <ESC>, <SHIFT> osv.
I den engelske originaludgave er variabelnavnene skrevet med små bogstaver i programmerne og med store bogstaver i teksten, og denne skrivemåde er bibeholdt i oversættelsen. Det er en god idé også at skrive kommandoerne med små bogstaver i programmerne, – så bliver det nemmere at finde fejl i programmet, idet alle kommandoer, som accepteres af maskinens BASIC, jo automatisk laves om til store bogstaver.
I bilag A finder du to programmer, der kan være en god hjælp, når du arbejder med bogens øvrige programmer.

Peter Wiwe

Indhold

Indledning

Denne bog indgår i en af de mest succesfulde serier i hjemmecomputerbøger. I 1982, da den første »Working Micro« bog (denne bog hedder »The working Amstrad« på engelsk) blev udsendt, var der kun få forlæggere som ville tro på, at almindelige ejere af hjemmecomputere gerne ville bruge og forstå deres maskiner, få *kontrol* med dem og lære at programmere – ved at lave programmer. De fleste bøger var enten »Begynderens bog om...« eller osse var de fyldt med spil og småprogrammer. Ideen med denne serie er at samle nogle gode, nyttige programmer, og samtidig give brugerne indsigt i de programmeringsteknikker, der blev anvendt. Og det var kun få, der troede på at den slags kunne sælges.
Men jeg og forlaget »Sunshine Books« var overbevist om at det præcis var, hvad der var behov for. Og vi fik ret.
Siden har »Working Micro«-serien fulgt udbredelsen af hjemmecomputerne til en lang række lande. Serien er indtil nu oversat til 15 forskellige sprog.
Og nu kommer der en maskine fra Amstrad, med en meget stærk BASIC, med en rigelig hukommelse og med en lang række faciliteter til en meget lav pris. Amstrad og ideen bag »Working Micro«-serien passer smukt sammen, og det håber vi, at programmerne i denne bog vil være et bevis på.

Brugen af bogen

Du kan bruge denne bog på flere forskellige måder:

1) Som en samling nyttige programmer, som du kan bruge og udbygge som du nu har brug for.
2) Som en samling subrutiner, der kan bruges i dine egne programmer.
3) Som en vejledning i avanceret BASIC programmering.

Men ligegyldigt hvordan du vil bruge den, så må du være opmærksom på, at den er skrevet *som en bog* og den ikke blot er en samling programmer i tilfældig rækkefølge. Vi hører ofte fra læsere som har problemer med at forstå programmerne, fordi de straks er gået i gang med de sidste programmer i bogen, uden at have læst bogen igennem først. De første programmer i bogen er, selvom de er ganske brugbare i sig selv, ment som en forberedelse til at arbejde med de senere programmer. Dertil kommer at de første programmer er forklaret ret udførligt, sådan at læseren skulle være godt rustet til at gå i lag med programmerne senere i bogen.
God fornøjelse!

KAPITEL 1

Tid

Det er altid svært at vide hvilket niveau en bog som denne skal starte på. Er programmerne for svære, går læserne måske i stå inden de rigtig kommer ind i de programmeringsteknikker, der gør bogens programmer lettere at forstå, når man først kommer i gang. Er programmerne på den anden side alt for enkle, vil mange læsere måske ikke finde de umagen værd at læse videre, og derfor ikke opdage, at de senere programmer er særdeles spændende og brugbare.
Jeg har derfor besluttet, at lade kapitel 1 handle om hvordan Amstrad kan arbejde med tid. Programmerne er ret enkle, men de præsenterer en lang række begreber, som vil blive brugt senere i de mere avancerede programmer. Samtidig indeholder programmerne beregninger, lyd og grafik, som demonstrerer nogle af Armstrads kvaliteter.
Der er fire programmer i dette kapitel:

ANALOG-UR: Viser et almindeligt ur i højopløsningsgrafik.

GRAFIK-UR: Viser hvad klokken er på en helt anden måde.

TIMER: Her kan op til 16 timere bruges samtidig, og de udsender en varselstone når tiden er udløbet. Derudover kan der udsdkrives meddelelser på skærmen.

STOPUR: Amstrad'en bliver her til et avanceret (men dyrt!) stopur.

Program 1.1: Analog-ur

Programmet laver en almindelig urskive med to visere, der flyttes én gang i minuttet. Programmet er udførligt kommenteret, og der fortælles en del om bogens programmeringsmetoder, så det må anbefales at du læser kommentaren grundigt.

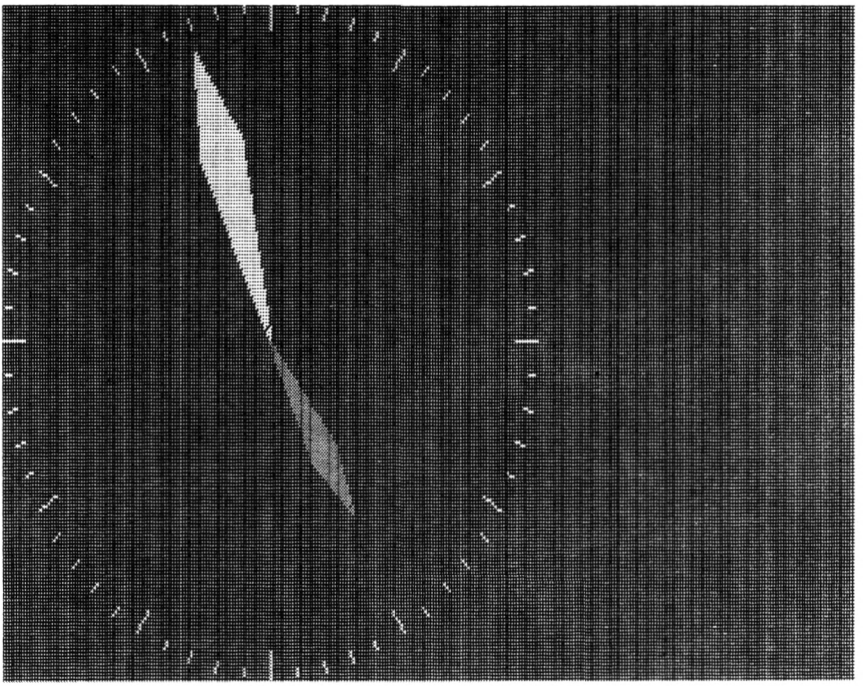


*Fig. 1.1: Skærm-dump fra programmet. Uret ser lidt ovalt ud her, men på skærmen er det rundt. *)*

I dette program præsenteres bl.a.:

1) Gemning af programmet på bånd, mens det opbygges.
2) Initialisering af programmet.
3) Tid med EVERY-kommandoen.
4) Modulvis programmering.
5) Kontrolmodulet.
6) Matematikken og grafikken bag en cirkel.

Modul 1.1.1: Gemning på bånd

Disse nedenstående tre programlinier er måske ikke den mest spændende start, men alle der har prøvet at bruge tidligere bøger i denne serie (til

*) Bogens skærmdumps er lavet med programmet Tascopy. Dette program tillader ikke brugen af kommandoen SYMBOL AFTER, så derfor er der ikke brugt æ, ø og å i de versioner, der er blevet lavet dumps efter.

andre hjemmecomputere) ved, at dette lille modul kan spare en for en masse besvær unden opbygningen af programmer.
De fleste finder af bitter erfaring ud af, at det er nødvendigt at gemme programmerne i de forskellige faser af opbygningen. Før eller senere sker det nemlig for de fleste af os, at flere timers arbejde går tabt, fordi der kommer en lille strømafbrydelse, en sikring springer eller et stik falder ud. Erfarne brugere gemmer deres programmer så ofte, at det højest er et kvarters arbejde, der går tabt.
Formålet med disse tre linier er, at gøre det så nemt som muligt at gemme programmet regelmæssigt. Det sker simpelthen ved at indtaste GOTO 2.
En anden finesse som kan spare tid senere hen er, at et sådant startmodul også giver en fast startlinie til programmet – nemlig '1'. Ofte er det nemlig hensigtsmæssigt at starte et program med GOTO, hvis du ikke vil slette variabler og data med RUN – bruger du dette lille modul behøver du ikke finde ud af hvor programmet starter, idet du blot kan bruge GOTO 1.
Dette modul blev brugt da alle bogens programmer blev udviklet, men da det kun er programmernes navne der ændres, er det udeladt i de øvrige programlistninger.

Modul 1.1.1: Linie 1-3

```
1 GOTO 3
2 SAVE "Analogur":STOP
3 REM
```

Test

Sørg for at der er bånd i dataoptageren. Indtast.

GOTO 2 <ENTER>

Følg instruktionerne på skærmen og start dataoptageren. Efter nogle sekunder kommer udskriften BREAK IN 2. Nu kan du slette de tre linier fra hukommelsen og hente programmet fra båndet efter at have spolet tilbage. Indtast

NEW <ENTER> (sletter programmet i hukommelsen)
LOAD "analogur" <ENTER>

Når programmet er indlæst, kan du få programmet frem på skærmen med LIST.

Modul 1.1.2: Initialisering af programmet

Ethvert program på mere end nogle få linier indeholder variabler og konstanter, hvis værdier kan ændres mens programmet kører, – eller som i hvert fald kan ændres fra program til program. Fordelen ved at bruge variabler er at du kan skrive programlinier, som kan bruges i mere end en situation, – f.eks. kan PRINT 2*A bruges uanset hvilken værdi A har. Det er meget få variable som skal have tildelt en værdi når programmet starter, og folk tilskriver ofte først værdien når variablen bruges. Men det er en fejltagelse at gøre det sådan, for efterhånden som programmet opbygges, bliver det stadigt sværere at finde ud af, hvad værdien af de vigtigste variabler er, når programmet først starter. Det er derfor en god vane at tildele de vigtigste variabler deres startværdier allerførst i programmet, – dette kaldes »initialisering«.

Når det først er slået fast, så er der en undtagelse, der er rimelig nok, når hukommelsen trods alt er begrænset, og det er at udelade de variabler, hvor startværdien *ikke* har nogen betydning, når programmet starter. Bortset fra de værdier, der bestemmer de farver der anvendes, så er de vigtige variabler i dette program bestemt af brugerens angivelse af hvad klokken er. For den slags variabler er det jo formålsløst at give nogen værdi på forhånd, så de udelades ofte i initialiseringsmodulet.

I dette tilfælde skal modulet fastsætte parametrene for den grafiske skærm og omstille maskinen til grader (DEG), som i dette tilfælde er nemmere at regne med end radianer.

Modul 1.1.2: Linie 2000-2110

```
2ØØØ REM ******************************
2Ø1Ø REM Initialisering
2Ø2Ø REM ******************************
2Ø3Ø MODE 1
2Ø4Ø INK Ø,1
2Ø5Ø INK 1,24
2Ø6Ø INK 2,3
2Ø7Ø INK 3,6
2Ø8Ø BORDER 1
2Ø9Ø ORIGIN 2ØØ,2ØØ
21ØØ DEG
211Ø RETURN
```

Hvis du vil, kan du nu lave denne lille test, indtast:

PRINT tim, minut <ENTER>

og det angivne tidspunkt skal blive skrevet på skærmen.

Hvis modulet er rigtigt indlæst, vil skærmen blive slettet og meldingen 'Unexpected RETURN in 2110' vil blive udskrevet. En mere grundig afprøvning kan først ske når de efterfølgende moduler er indlæst.

Modul 1.1.3: Indstil uret

Før vi kan gå i gang med at lave et ur, må vi kunne indstille uret. Formålet med dette modul er, at brugeren skal kunne angive, hvad klokken er i timer og minutter, så det kan blive lagret i computerens hukommelse som to variabler, TIM og MINUT.

Modul 1.1.3: Linie 3000-3080

```
3ØØØ REM ******************************
3Ø1Ø REM Indstilling af uret
3Ø2Ø REM ******************************
3Ø3Ø PRINT "Klokken er:":PRINT
3Ø4Ø INPUT "Timer (1-12):";tim
3Ø5Ø IF tim<1 OR tim>12 THEN PRINT "****
 Udenfor området - prøv igen ****":GOTO
3Ø4Ø
3Ø6Ø INPUT "Minutter (Ø-59):";minut
3Ø7Ø IF minut<Ø OR minut>59 THEN PRINT "
**** Udenfor området - prøv igen ****":G
OTO 3Ø6Ø
3Ø8Ø RETURN
```

Test

Indtast

GOTO 3000 <ENTER>

Og du skal så blive bedt om at angive tiden i timer og minutter. Hvis svaret er 'forkert' skal der komme en fejlmelding, mens et 'rigtigt' svar skal blive accepteret. Endelig skal programmet stoppe med en 'Unexpected RETURN' fejlmelding. I det færdige program bliver modulet en subrutine, som kaldes af programmets kontrolmodul med GOSUB.

Test

En enkel test kan laves ved at indtaste:

GOTO 2000 <ENTER>

Modul 1.1.4: Urskiven

Nu kommer vi til grafikken, – først skal vi lave urskiven og i senere moduler skal vi lave viserne.

Modul 1.1.4: Linie 4000-4090

```
4ØØØ REM ******************************
4Ø1Ø REM Urskiven
4Ø2Ø REM ******************************
4Ø3Ø CLS
4Ø4Ø FOR a=Ø TO 359 STEP 6
4Ø5Ø MOVE 2ØØ*SIN(a),2ØØ*COS(a)
4Ø6Ø r=195: IF a MOD 3Ø=Ø THEN r=185
4Ø7Ø DRAW r*SIN(a),r*COS(a),1
4Ø8Ø NEXT a
4Ø9Ø RETURN
```

Kommentar

Tegning af cirkel

Vi kan tegne en cirkel på grundlag af følgende oplysninger:

a) Cirkelens radius. (RADIUS)
b) Vinklen (med uret) fra 'klokken tre' til den angivne position. (VINKEL)
c) Koordinaterne for cirklens centrum. (CENTRUM X og CENTRUM Y)

Positionen bestemmes nu af to formler:

X koordinat = RADIUS*COSINUS(VINKEL)+CENTRUM X
Y koordinat = RADIUS*SINUS(VINKEL)+CENTRUM Y

Pladsen tillader ikke at vi går ind på hvorfor det er sådan, men enhver god geometribog vil indeholde en udtømmende forklaring. I vores tilfælde er formlen endda lidt simplere, fordi vi i initialiseringsmodulet angav ORIGIN til 200,200. Det betyder at punktet 0,0 på den grafiske skærm nu ligger 200 pixels oppe og 200 pixels til højre fra skærmens nederste

venstre hjørne. Fordelen ved dette er at vi med ORIGIN bestemmer urskivens centrum, og vi derfor kan se bort fra X,Y-angivelserne, idet de begge er nul. Formlen for et vilkårligt punkt på cirklens omkreds bliver derfor:

X koordinat = RADIUS*COSINUS(VINKEL)
Y koordinat = RADIUS*SINUS(VINKEL)

som netop er det udtryk der anvendes i dette modul.

Linie 4040-4080: Denne løkke kører 360 grader rundt i spring på 6 grader. Da der er 60 minutter på en time og 360/60=6, kan det ikke være nogen overraskelse, at udregningen har forbindelse med afmærkningen af minutterne på urskiven, som vil blive tegnet af et senere modul.

Linie 4050: Den grafiske markør bliver flyttet (med MOVE) til et punkt på cirkelens omkreds, ved hjælp af formlen ovenfor. Den første position – mens variablen i løkken er nul – vil være det øverste punkt på cirklen. De efterfølgende positioner vil køre cirklen (med uret) rundt i spring på seks grader.

Linie 4060-4070: Disse to linier tegner linierne fra cirklen og ind mod midten. Det, der faktisk sker er, at der tegnes en linie fra cirklen og ind til en mindre cirkel med samme centrum. Den yderste cirkel har en radius på 200 pixels, mens den mindre cirkels radius varierer alt efter om A i løkken ovenfor svarer til 5 hele minutter (30 grader). Hvis vinklen ikke svarer til 30 grader er afstanden mellem de to cirkler kun 5 pixels, – og linien bliver kun 5 pixels lang.

Brugen af MOD funktionen sikrer at programmet kan kontrollere om en vinkel på 30 grader er nået. 5 MOD 2 giver f.eks. resultatet 1 – dvs. resten når 5 divideres med 2. A MOD 30 – som i programmet – giver resten når løkke-variablen A divideres med 30, – når resten er nul så går 30 op i vinklen. Hver gang det sker bliver den indre cirkel gjort mindre, og den lille linie bliver længere, således at hver femte minut bliver tydeligt markeret.

TEST

Indtast

GOTO 4000 <ENTER>

og skærmen skal blive slettet, hvorefter urskiven tegnes. Derefter skal programmet stoppe med en 'Unexpected RETURN' fejlmelding.

Modul 1.1.5: Justering af minutter og timer

I dette modul når vi ind til kernen af programmet, hvor tiden stilles et minut frem, hver gang modulet køres, og på grundlag heraf udregnes koordinaterne for viserne. Modulet kan ikke fungere ordentligt før det sidste modul er kommet med, idet det kun skal kaldes én gang i minuttet.

Modul 1.1.5: Linie 5000-5130

```
5ØØØ REM ******************************
5Ø1Ø REM Justering af tiden
5Ø2Ø REM ******************************
5Ø3Ø c=Ø
5Ø4Ø vinkel=minut*6:str=18Ø:GOSUB 6ØØØ
5Ø5Ø vinkel=tim*3Ø+minut/2:str=12Ø:GOSUB
 6ØØØ
5Ø65 minut=minut+1
5Ø7Ø IF minut=6Ø THEN minut=Ø:tim=tim+1
5Ø8Ø IF tim=13 THEN tim=1
5Ø9Ø c=2
51ØØ vinkel=minut*6:str=18Ø:GOSUB 6ØØØ
511Ø c=3
512Ø vinkel=tim*3Ø+minut/2:str=12Ø:GOSUB
 6ØØØ
513Ø RETURN
```

Kommentar:

Dette modul kan inddeles i tre afsnit. Det første kalder det efterfølgende modul som sletter viserne. Det næste afsnit lægger et minut til og laver eventuelt timeskift. Det tredie afsnit udregner viservinklerne med de nye værdier for timer og minutter, og kalder det efterfølgende modul, som tegner viserne igen.

Linie 5030-5050: Variablen C bruges i næste modul til at bestemme inkfarven, når viserne tegnes. Først sættes den til 0. Hvis du ser tilbage i initialiseringsmodulet, kan du se at ink 0 baggrundsfarven, var blå. Linie 5040 bestemmer parametrene for den store viser – vinklen og størrelsen. Da minutangivelsen endnu ikke er blevet ændret, kan viseren slettes ved at give den samme farve som baggrunden. Linie 5050 gør det samme med den lille viser.

Linie 5060-5080: Minut- og timeværdierne opskrives ved at lægge 1 til minutterne, og derefter laves eventuelle justeringer, for at sikre at vi får nogle 'fornuftige' værdier.

Linie 5090-5120: De to visere tilskrives nye værdier, og det næste modul kaldes derefter, så viserne kan blive tegnet igen. Ink-farven sættes til 2 for den store viser og 3 for den lille. I initialiseringsmodulet kan du se, at det svarer til farverne rød og højrød.

Test

Skriv:

6000 RETURN <ENTER>

Dette er en midlertidig linie, som tager højde for at modulet kalder en linie som ikke findes endnu.

Skriv nu:

TIM = 0 <ENTER>
MINUT = 59 <ENTER>
GOTO 5000 <ENTER>

og du skal så næsten øjeblikkeligt se at programmet stopper med en 'Unexpected RETURN' fejlmelding. Skriv nu:

PRINT minut, tim <ENTER>

Nu skal denne udskrift vise sig på skærmen:

0 1

som viser at de oprindelige 59 minutter er blevet opskrevet til 60, som igen er blevet til en time.

Modul 1.1.6: Viserne tegnes

Når vi nu har beregnet de tal, der skal til for at finde visernes stillinger, kan vi nu gå igang med at få dem tegnet.

Modul 1.1.6: linie 6000-6110

```
6ØØØ REM ******************************
6Ø1Ø REM Tegning af visere
6Ø2Ø REM ******************************
6Ø3Ø str2=str*2/3
6Ø4Ø PLOT Ø,Ø,c
6Ø5Ø DRAW str2*SIN(vinkel-8),str2*COS(vi
nkel-8)
6Ø6Ø DRAW str*SIN(vinkel),str*COS(vinkel
)
6Ø7Ø DRAW str2*SIN(vinkel+8),str2*COS(vi
nkel+8)
6Ø8Ø DRAW Ø,Ø
6Ø9Ø MOVE str2*SIN(vinkel),str2*COS(vink
el)
61ØØ GOSUB 7ØØØ
611Ø RETURN
```

Kommentar

Linie 6030: Viserne bliver formet som nogle aflange parallelogrammer. STR2 står for afstanden fra centrum til det sted hvor viseren er tykkest.

Linie 6040: Den grafiske markør flyttes til urskivens centrum og farven bestemmes af C, – på den måde kan det foregående modul bestemme hvilken farve, der skal bruges.

Linie 6050-6080: Disse linier tegner de fire linier på skærmen som er visernes omrids. Den første linie er STR2 lang og tegnes 8 grader mod uret i fht. den 'rigtige' vinkel for timer eller minutter. Den næste tegnes fra enden af den første og til et punkt i den 'rigtige' vinkel og i afstanden STR fra centrum. Den tredie linie tegnes til et punkt STR2 pixels fra centrum og 8 grader med uret fra den 'rigtige' vinkel. Endelig afsluttes viseren ved at det sidste linie tegnes tilbage til centrum.

Linie 6090-6100: Disse to linier skal bruges i forbindelse med det næste modul, hvor der fyldes farver i de former, der er blevet tegnet af dette modul. I linie 6090 flyttes den grafiske markør således til et punkt inde i den viser, der skal farves.

Test

Først tilføjes en linie, som gør det ud for det næste modul:

7000 RETURN <ENTER>

Skriv nu:

CLEAR:CLS:DEG:GOTO 5000 <ENTER>

Og du skal se at skærmen slettes og de to visere bliver tegnet i en stilling, der svarer til et minut over tolv. Det er kun omridset af viserne der ses, – de bliver fyldt ud med farver i det næste modul.

Modul 1.1.7: En form udfyldes med farve

En at de få svagheder ved Amstrad CPC 464, i forhold til visse andre hjemmecomputere er, at der mangler en kommando, der kan fylde en form ud med farve. Dette modul er en del langsommere end en indbygget kommando ville være, og det giver nogle begrænsninger, men det virker og det er da værd at tage med.

Den rutine, som står i modulet nedenfor, kan fylde farve i alle polygoner med konkave vinkler – set udefra – dvs. med vinkler der peger udad og ikke indad. Rutinen kan også fungere på *visse* figurer med indadvendte vinkler, men ikke alle. Hvis du vil bruge rutinen på sådanne former, må du fylde dem ud i sektioner. Læg specielt mærke til at hvis rutinen bruges på en form, der ikke er lukket, vil den sandsynligvis 'låse', fordi den søger efter en side, der ikke er der.

Modul 1.1.7: Linie 7000-7300

```
7ØØØ REM ******************************
7Ø1Ø REM Udfyldning med farve
7Ø2Ø REM ******************************
7Ø3Ø IF TESTR(Ø,Ø)=c THEN RETURN
7Ø4Ø s=2
7Ø5Ø MOVE 2*INT(XPOS/2),2*INT(YPOS/2)
7Ø6Ø :
7Ø7Ø :
7Ø8Ø REM check op til højre ***********
7Ø9Ø IF TESTR(Ø,s)<>c THEN GOTO 7Ø9Ø
71ØØ IF TESTR(s,-s)<>c THEN GOTO 7Ø9Ø
711Ø :
```

```
712Ø :
713Ø REM check op til venstre *********
714Ø MOVER -s,Ø
715Ø IF TESTR(Ø,s)<>c THEN GOTO 7Ø9Ø
716Ø IF TESTR(-s,-s)<>c THEN GOTO 715Ø
717Ø :
718Ø :
719Ø REM farve i linier vandret *******
72ØØ x1=XPOS
721Ø MOVER s,Ø
722Ø PLOTR Ø,Ø,c
723Ø IF TESTR(s,Ø)<>c THEN GOTO 722Ø
724Ø x2=XPOS-s
725Ø MOVE x1,YPOS-s
726Ø IF TESTR(s,Ø)<>c THEN GOTO 729Ø
727Ø IF XPOS=x2 THEN RETURN
728Ø GOTO 726Ø
729Ø IF TESTR(-s,Ø)=c THEN GOTO 72ØØ
73ØØ GOTO 729Ø
```

Kommentar

Linie 7030: Denne rutine vil, som de fleste andre farve-rutiner, ikke fungerer hvis farven af den pixel, som rutinen starter ved, har samme farve som den rutinen skal bruge.

Linie 7040: Variablen S tildeles værdien af det mindste vandrette trin, som bruges i modulet. Værdien skal ændres hvis der bruges en anden *grafisk mode.*

Linie 7050: I grafisk mode 1, som vi bruger i dette program, er pixlerne grupperet parvis. Denne linie sikrer at koordinaterne altid er lige tal.

Linie 7080-7100: Nu begynder markøren at søge efter en af figurens grænselinier. Linierne kan måske umiddelbart se underlige ud, idet der tilsyneladende ikke sker andet end at en linie danner en uendelig løkke. Men faktisk flytter både TEST og TESTR grafikmarkøren til den angivne position. Derfor søger den første linie opad i en lige linie, indtil den finder en pixel med den rigtige farve. Når en del af figurens afgrænsning er fundet, flytter den næste linie markøren et trin ned og et trin til højre, for at se om dette punkt har den eftersøgte farve, – er det ikke tilfældet

springes tilbage til den første linie, som undersøger om det er muligt at nå højere op.

Linie 7130-7160: Efter at søgt så langt op til højre som muligt, starter en tilsvarende undersøgelse op til venstre. Når undersøgelsen op til højre og venstre er afsluttet, har programmet fundet frem til det øverste punkt i figuren, – ellers er markøren havnet i en blindgyde (– hvis figuren ikke er regulær).

Linie 7200-7230: Når det øverste punkt er fundet kan udfyldningen med farve begynde. MOVER og PLOTR kommandoerne bruges til at farve en række pixels fra venstre til højre indtil den modsatte side nåes.

Linie 7240-7250: Når en vandret linie er færdig, sættes X2 lig med det punkt yderst til højre, som er farvet, og markøren flyttes tilbage til liniens begyndelse – men en pixel ned.

Linie 7260-7280: Når markøren flyttes tilbage til venstre, undersøges det, om pixelen allerede er farvet. Hvis det er tilfældet søger programmer fra venstre til højre efter det første felt uden farven C. Hvis søgningen fortsætter udover enden af linien ovenfor, som lige er blevet farvet, er bunden af figuren nået.

Linie 7290-7300: Hvis markøren støder på en pixel, der ikke er blevet farvet, når den flyttes ned og til venstre, så søger programmet videre til venstre indtil grænsen bliver nået, før fyldningen med farve fortsætter.

Test

Den enkleste måde at teste dette modul på er at bruge samme test som ved det foregående modul. Den eneste forskel er at nu skal viserne blive fyldt med farve.

Modul 1.1.8: Modulerne sættes sammen

Måske er du efterhånden begyndt at undre dig over at programmet er skrevet som det er. – Kunne alle de funktioner vi har beskrevet ikke have været sat sammen med nogle få GOTO kommandoer? Desværre er mange af de programmer, der findes i databladene, netop opbygget sådan.

I denne bog er alle programmerne opbygget i moduler med klart afgrænsede funktioner. Grunden til at de er lavet på den måde er, at de er nemmere at læse, nemmere at 'afluse', de kan ændres ved at skifte enkelte

moduler ud og de kan udbygges med nye moduler. Der kan læres meget af programmerne i denne bog, men de vigtigste er nok selve programmeringsteknikken – modulopbygningen.

Dette modul er selve nøglen til teknikken, for når alle programmodulerne er indlæst og testet, skal vi bruge et *kontrolmodul* til at styre programmet. På sin vis er dette modul selve programmet – resten er blot udvidelser af det.

Modul 1.1.8: Linie 1000-1100

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
1Ø3Ø GOSUB 2ØØØ
1Ø4Ø GOSUB 3ØØØ
1Ø5Ø GOSUB 4ØØØ
1Ø6Ø GOSUB 5ØØØ
1Ø7Ø EVERY 3ØØØ GOSUB 5ØØØ
1Ø8Ø IF INKEY$<>" " THEN 1Ø8Ø
1Ø9Ø CLS
11ØØ END
```

Kommentar

Linie 1070: Her er selve tidsprogrammets hjerte. Denne ene linie danner 'urværket' ved hjælp af Amstrads effektive EVERY kommando. Kommandoen virker på den måde at brugeren kan fastsætte et tidsinterval som, når denne tid er gået, kan afbryde det computeren ellers er i gang med, og beordre den til at udføre en anden opgave. Når afbrydelsen (engelsk: 'interrupt') er slut, genoptages den oprindelige opgave indtil næste afbrydelse. Det mindste interval er 1/50 sekund, og i denne linie beordres maskinen til at kalde linie 5000, når der er gået 3000 X 1/50 sekund – svarende til ét minut. Subrutinen i linie 5000 sørger så for at lægge et minut til og flytte viserne.

Linie 1080-1100: Resten af tiden kører programmet i den uendelige løkke i linie 1080. Løkken kan afbrydes ved at brugeren trykker mellemrumstangenten ned. Når det sker slettes skærmen og programmet stopper.

Test

Programmet skulle nu kunne fungere. Kør det, indtast tiden, og uret skal komme frem på skærmen med viseren i den rigtige stilling. Hvis du vil checke justeringen af tiden kan du slette to nuller i 3000 i linie 1070. Viserne vil da flytte sig hele tiden.

Program 1.2: Grafik-ur

En af de mest spændende ting ved computere med en så god grafik som Amstrads, er at de giver muligheder for at eksperimentere med at vise forskellige ting på nye og fantasifulde måder. Efter vi har lavet et ganske almindeligt ur i det første program, skal vi her anskue tiden fra en noget anden vinkel. I GRAFIK-UR er timer og minutter repræsenteret af to linier der langsomt glider fra venstre til højre og fra top til bund, således at skærmen inddeles i fire rektangler i forskellige farver. En stor del af programmet svarer til modulerne i analog-uret, så derfor er forklaringerne tilsvarende korte.

I dette program behandles følgende nye emner:

1) Vinduer.
2) Formattering af tal
3) Udsnit af strenge.

Modul 1.2.1: Initialisering

En lille initialiseringsmodul definerer de farver der skal bruges. Vinduet i linie 2100 forklares i et senere modul.

Modul 1.2.1: Linie 2000-2120

```
2000 REM ******************************
2010 REM Initialisering
2020 REM ******************************
2030 MODE 1
2040 INK 0,1
2050 INK 1,24
2060 INK 2,9
2070 INK 3,6
2080 BORDER 1
2090 PAPER 0:PEN 1
2100 WINDOW #1,39,39,9,18
2110 PAPER #1,0:PEN #1,1
2120 RETURN
```

Test

Som i det første program er den eneste test der kan laves indtil videre at skrive:

GOTO 2000 <ENTER>

som skal få programmet til at slutte med en 'Unexpected RETURN' fejlmelding. Andre fejlmeldinger betyder at der er lavet fejl under indlæsningen af modulet.

Modul 1.2.2: Indstilling af tiden

Svarer til modulet i ANALOG-UR

Modul 1.2.3: Linie 3000-3090

```
3ØØØ REM ******************************
3Ø1Ø REM Indstilling af uret
3Ø2Ø REM ******************************
3Ø3Ø PRINT "Klokken er:":PRINT
3Ø4Ø INPUT "Timer (1-12):";tim
3Ø5Ø IF tim<1 OR tim>12 THEN PRINT "****
 Udenfor området - prøv igen ****":GOTO
3Ø4Ø
3Ø6Ø INPUT "Minutter (Ø-59):";minut
3Ø7Ø IF minut<Ø OR minut>59 THEN PRINT "
**** Udenfor området - prøv igen ****":G
OTO 3Ø6Ø
3Ø8Ø sekund=Ø
3Ø9Ø RETURN
```

Modul 1.2.3: Markering af rammen

Dette modul laver en ramme med afmærkning af timer og minutter til venstre og øverst i skærmbilledet.

Modul 1.2.3: Linie 4000-4120

```
4ØØØ REM ******************************
4Ø1Ø REM Markering af rammen
4Ø2Ø REM ******************************
```

```
4Ø3Ø CLS:PRINT"  ";
4Ø4Ø FOR i=5 TO 55 STEP 1Ø
4Ø5Ø PAPER Ø:PEN 1:PRINT USING"###";i;
4Ø6Ø PAPER 1:PEN Ø:PRINT USING"###";i+5;
4Ø7Ø NEXT i
4Ø8Ø PAPER Ø:PEN 1
4Ø9Ø FOR i=1 TO 12
41ØØ LOCATE 1,i*2+1:PRINT USING"##";i
411Ø NEXT i
412Ø RETURN
```

Kommentar

Linie 4040-4070: Løkken skriver minutangivelserne ud langs overkanten af skærmen, baggrundsfarven og skriftfarven byttes om, således at hver anden angivelse er i invers skrift. PRINT USING bruges til at sikre at tallet for hvert femte minut bliver skrevet ved brug af tre karakterer. Da alle tallene er på mindre end tre karakterer, udfylder PRINT USING med en eller to tomme pladser.

Linie 4080-4110: Løkkevariablen I bruges sammen med LOCATE kommandoen til at udskrive timeangivelserne langs venstre side af skærmen.

Test

For at kunne lave en ordentlig afprøvning af dette modul, indsættes en midlertidig linie:

4115 GOTO 4115 <ENTER>

Denne linie hindrer skærmen i at rulle opad med en 'Unexpected RETURN' melding. Indtast nu:

GOTO 4000 <ENTER>

og nu skal afmærkning komme frem langs skærmens kanter. Tast <ESC> to gange, hvis det er i orden – og husk at slette den midlertidige linie.

Modul 1.2.4: Klokkestreng

En af programmets funktioner er at vise kvad klokken er – både som et almindeligt digitalur – og dels rent grafisk, som det blev beskrevet tidlige-

re. Den streng, der skal give den digitale udlæsning, bruger variable som laves i næste modul. Selve 'klokkestrengen' laves i dette modul.

Modul 1.2.4: Linie 7000-7050

```
7ØØØ REM ******************************
7Ø1Ø REM Klokkestreng
7Ø2Ø REM ******************************
7Ø3Ø klok$=STR$(1ØØØØØØ+tim*1ØØØØ+minut*
1ØØ+sekund)
7Ø4Ø klok$=MID$(klok$,3,2)+"  "+MID$(klo
k$,5,2)+"  "+MID$(klok$,7,2)
7Ø5Ø RETURN
```

Kommentar

Linie 7030-7040: En simpel måde at sammensætte en række tal til et tal er at gange hver af dem med faldende 10-talspotenser og lægge dem til en højere 10-talspotens. Hvis vi f.eks. har 1 time, 36 minutter og 2 sekunder, så ville resultatet af denne linie blive 1013602. Hvorfor? Ja, se på time-angivelsen – de 1000000 som blev lagt til giver et 0 foran 1-tallet, ligesom ved 2-tallet i sekundangivelsen.

Nu skal vi lave udsnit af den streng vi har lavet, for at få de tre to-cifrede tal som vi er interesseret i, – vi ved jo at der kommer et nul foran, hvis nogle af tallene er en-cifrede. Strengfunktionerne i den anden linie uddrager de tre to-cifrede tal fra strengen med start fra de angivede pladser. F.eks. uddrager MID$(A$,3,2) to cifre af strengen A$, startende fra plads nummer 3.

Hvis du ser på kommandoerne i linie 7040, virker det umiddelbart underligt, at de to time-cifre tager fra plads tre og frem, når de står på plads to og tre i vores oprindelige streng i linie 7030. Forklaringen er at vi har brugt STR$-funktionen til at omdanne tallet til en streng. Når vi gør det ser tallet ud på samme måde, men det kan nu behandles som en streng – den eneste forskel er at der laves en plads til tallets fortegn – der står blot ikke noget, da tallet er positivt. Det første ciffer i et tal som er blevet lavet om til en streng med STR$ står derfor på plads 2.

Resultatet af modulet bliver at vi får en kort streng, der ser således ud:

TO CIFRE TIMER/MELLEMRUM/TO CIFRE MINUTTER/MELLEMRUM/TO CIFRE SEKUNDER

Test

Skriv:

TIM = 1 <ENTER>
MINUT = 1 <ENTER>
SEKUND = 1 <ENTER>
GOTO 7000 <ENTER>

Når programmet udskriver en 'unexpectet RETURN' fejl, så skriv:

PRINT klok$ <ENTER>

og følgende udskrift skal vise sig:

01 01 01

Modul 1.2.5: Justering af tiden

Dette modul svarer til modulet i det første program.

Modul 1.2.5: Linie 6000-6160

```
6ØØØ REM ******************************
6Ø1Ø REM Justering af tiden
6Ø2Ø REM ******************************
6Ø3Ø sekund=sekund+1
6Ø4Ø WHILE sekund>59
6Ø5Ø sekund=Ø:minut=minut+1
6Ø6Ø WHILE minut>59
6Ø7Ø minut=Ø:tim=tim+1
6Ø8Ø WHILE tim>12
6Ø9Ø tim=1
61ØØ WEND
611Ø WEND
612Ø WEND
613Ø LOCATE #1,1,1
614Ø GOSUB 7ØØØ
615Ø PRINT#1,klok$;
616Ø RETURN
```

Kommentar

Linie 6030-6120: Her er tre løkker indeni hinanden – den ydre løkke

justerer sekunderne og aktiverer den næste, hvis sekund-skiftet skal medføre et minut-skift osv.

Linie 6130-6160: Disse tre linier udskriver strengen fra de forrige modul lodret. Når vi bruger vinduet, som blev defineret i initialiseringsmodulet, sker det meget nemt. Ser du nemlig på det første modul kan du se, at vinduet skal være ude langs højre kant – og det er *kun én karakter bredt.* Det betyder at alt hvad der sendes til dette vindue udskrives i én lodret kolonne.

Test

Skriv:

CLEAR <ENTER>
SEKUND = 59 <ENTER>
MINUT = 59 <ENTER>
TIM = 12 <ENTER>
GOTO 6000 <ENTER>

og der skal komme den sædvanlige 'Unexpected RETURN' melding, samtidig med at '01 00 00' bliver udskrevet vandret ude til højre. Det viser at løkkerne i modulets første halvdel fungerer korrekt.

Modul 1.2.6: Grafisk tidsvisning

Nu kommer vi til det modul, hvor skærmen opdeles i firkanter, som viser hvad klokken er. En linie skal glide hen over skærmen fra venstre mod højre og vise minutterne, mens en anden glider fra top til bund og viser timerne. Dette gøres ved at inddele skærmen i fire rektangler – to røde og to grønne – og grænsefladerne mellem dem er netop de to linier, der viser minutter og timer – se fig. 1.2.

Det er uden tvivl muligt at placere disse firkanter direkte på skærmen, men det kan gøres mere enkelt. Alt hvad vi behøver at gøre, er at tegne nogle linier med farvede 'mellemrum' øverst på skærmen, hvor farven skifter fra rød til grøn på det rigtige sted – og omvendt nederst på skærmen. Selve linierne laves af en løkke i forbindelse med en LOCATE kommando.

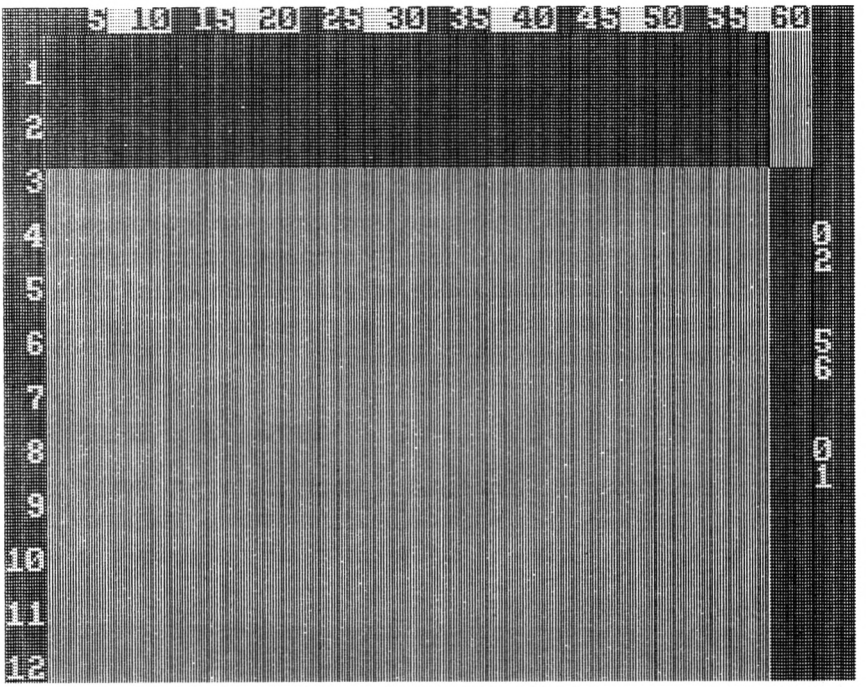


Fig. 1.2: Skærmen opdeles i fire rektangler.

Modul 1.2.6: Linie 5000-5100

```
5ØØØ REM ******************************
5Ø1Ø REM Grafisk tidsvisning
5Ø2Ø REM ******************************
5Ø3Ø PAPER 2:PEN 3
5Ø4Ø FOR i=2 TO 25
5Ø5Ø IF i=INT(tim*2+minut/3Ø+2) THEN PAP
ER 3:PEN 2
5Ø6Ø LOCATE 3,i
5Ø7Ø PRINT STRING$(minut*Ø.6,CHR$(143));
TAB(39)
5Ø8Ø NEXT i
5Ø9Ø RETURN
51ØØ RETURN
```

Kommentar

Linie 5030: Baggrundsfarven bliver grøn og skriftfarven rød.

Linie 5040-5080: Der bruges 24 skærmlinier til tidsvisningen, hvor hver linie svarer til en halv time – den første linie er skærmlinie 2.

Linie 5050: Når løkken når til det sted, hvor time-linien skal være, byttes de to farver om.

Linie 5060-5070: Disse to linier finder og udskriver en af linierne tværs over skærmen. STRING$ funktionen laver en række 'inverse mellemrum' frem til minut-linien. TAB (39) flytter markøren til til skærmens højre side, således at resten af linien beholder skærmfarven. Strengens længde bestemmes af variablen MINUT og ganges med 0.6 fordi skærmbilledet kun er 36 karakterer bredt – ikke 60.

Test

Indtast:

TIM = 6 <ENTER>
MINUT = 30 <ENTER>
GOTO 5000 <ENTER>

Nu skal skærmen være inddelt i fire omtrent lige store firkanter, som fig. 1.2 viser. Programmet slutter med en 'Unexpected RETURN' fejlmelding.

Modul 1.2.7: Modulerne sættes sammen

Nu er programmet indlæst, – og vi mangler blot dette kontrolmodul, som kører modulerne i den rigtige rækkefølge.

Modul 1.2.7: Linie 1000-1090

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
1Ø3Ø ON BREAK GOSUB 1Ø9Ø
1Ø4Ø GOSUB 2ØØØ
1Ø5Ø GOSUB 3ØØØ
1Ø6Ø EVERY 5Ø GOSUB 6ØØØ
1Ø7Ø GOSUB 4ØØØ
1Ø8Ø GOSUB 5ØØØ:GOTO 1Ø8Ø
1Ø9Ø PEN 1 : PAPER Ø : CLS : END
```

Kommentar

Linie 1030: ON BREAK GOSUB kommandoen bruges til at bestemme, hvad der skal ske når <ESC> tastes to gange. I dette tilfælde springes der til linie 1090, som sætter PEN og PAPER normalt, sletter skærmen, sletter hukommelsen og stopper.

Linie 1060: Her kaldes EVERY én gang i sekundet.

Test

Nu kan programmet køres, den rigtige tid indtastes og grafik-uret virker.

Program 1.3: Timer

Programmet giver mulighed for at bruge indtil 16 nedtællings-timere, som hver for sig kan programmeres til at afgive et alarm-signal på givne tidspunkter. Samtidig kan der knyttes en kort besked til hver timer.

Fig. 1.3: Skærmdump fra programmet

I dette program præsenteres bl.a.

1) 'Menu'-modulet.
2) Enkel brug af SOUND
3) Sletning og indføjelse i arrays.
4) Anvendelse af 'venteposition'.

Modul 1.3.1.: Initialisering

Dette modul er større end dem vi har set hidtil, ligesom selve programmet er større end de foregående. Det meste af modulet definerer skærmens farver og laver nogle vinduer, som bliver brugt senere.

Modul 1.3.1: Linie 2000-2150

```
2ØØØ REM ******************************
2Ø1Ø REM Initialisering
2Ø2Ø REM ******************************
2Ø3Ø MODE 1
2Ø4Ø GOSUB 12ØØØ
2Ø5Ø WINDOW 1,4Ø,25,25
2Ø6Ø PAPER Ø:PEN 1:CLS
2Ø7Ø WINDOW #1,2,29,1,1
2Ø8Ø PAPER #1,Ø:PEN #1,1:CLS #1
2Ø9Ø PRINT #1,"Klokken er nu:"
21ØØ WINDOW #2,1,4Ø,3,23
211Ø PAPER #2,3:PEN #2,2:CLS #2:PRINT #2
212Ø SPEED INK 5Ø,25
213Ø DIM ttimer(15),tminut(15),ttid(15),
besked$(15)
214Ø ttid(Ø)=-1
215Ø RETURN
```

Kommentar

Linie 2040: I modsætning til de tidligere programmer indeholder dette modul ikke nogen INK-kommandoer. Det skyldes at INK-farverne vil blive ændret flere gange mens programmet kører. GOSB kalder her en lille rutine, der giver 'startfarverne'.

Linie 2050: WINDOW kommandoen kan bruges på to måder, enten kan et bestemt område af skærmen afsættes til brug sammen med en bestemt

datastrøm – sådan som det f.eks. skete i GRAFIK-UR – eller også kan man simpelthen blot angive vinduets størrelse. Hvis vinduet således defineres *uden* datastrømnummer betyder det, at maskinen bruger dette vindue til alle de meldinger m.v., som ikke er forbundet med en bestemt datastrøm. I dette tilfælde kommer alle de 'normale' udskrifter frem i en linie nederst på skærmen.

Linie 2070-2120: I disse linier defineres yderligere to vinduer, det ene (# 1) en linie øverst på skærmen, det andet (# 2) dækker stort set resten af skærmen. SPEED INK kommandoen anvendes når en INK kommando bruges med to farver, og den bestemmer hvor hurtigt, der skiftes mellem de to farver.

Linie 2130: Disse arrays bruges i programmet, de vil blive forklaret når vi kommer til dem.

Modul 1.3.2: Skærmfarverne ved start

Som forklaret i kommentaren ovenfor bliver skærmfarverne ændret mens programmet kører. Dette lille modul definerer starfarverne.

Modul 1.3.2: Linie 12000-12080

```
12ØØØ REM ****************************
12Ø1Ø REM Skærm
12Ø2Ø REM ****************************
12Ø3Ø INK Ø,1
12Ø4Ø INK 1,24
12Ø5Ø INK 2,26
12Ø6Ø INK 3,2
12Ø7Ø BORDER 1
12Ø8Ø RETURN
```

Modul 1.3.3: Indstilling af uret

Dette svarer til modulet ANALOG-UR, bortset fra at der bruges 24 i stedet for 12 timer.

Modul 1.3.3: Linie 3000-3100

```
3ØØØ REM ******************************
3Ø1Ø REM Indstil uret
3Ø2Ø REM ******************************
3Ø3Ø CLS:PRINT"Klokken er:":PRINT
3Ø4Ø INPUT "Timer (Ø-23):";timer
3Ø5Ø IF timer<Ø OR timer>23 THEN PRINT "
**** Udenfor området - prøv igen ****":G
OTO 3Ø4Ø
3Ø6Ø INPUT "Minutter (Ø-59):";minut
3Ø7Ø IF minut<Ø OR minut>59 THEN PRINT "
**** Udenfor området - prøv igen ****":G
OTO 3Ø6Ø
3Ø8Ø sekund=Ø
3Ø9Ø tid=timer*6Ø+minut
31ØØ RETURN
```

Modul 1.3.4: Justering af tiden

Modulet her svarer til de tidligere tids-justeringsmoduler.

Modul 1.2.4: Linie 14000-14180

```
14ØØØ REM ******************************
14Ø1Ø REM Justering af tiden
14Ø2Ø REM ******************************
14Ø3Ø sekund=sekund+5
14Ø4Ø WHILE sekund>59
14Ø5Ø sekund=Ø:minut=minut+1
14Ø6Ø WHILE minut>59
14Ø7Ø minut=Ø:timer=timer+1
14Ø8Ø WHILE timer>23
14Ø9Ø timer=Ø
141ØØ WEND
1411Ø WEND
1412Ø WEND
1413Ø LOCATE#1,16,1
1414Ø t=timer:m=minut:s=sekund:GOSUB 15Ø
ØØ
1415Ø PRINT#1,klok$;
```

```
14160 tid=timer*60+minut
14170 GOSUB 9000
14180 RETURN
```

Modul 1.3.5: Formattering af tiden

Ligesom i det foregående program virker tidsjusteringsmodulet sammen med dette modul, der oversætter tiden til en form, der umiddelbart kan forstås af brugeren.

Modul 1.3.5: Linie 15000-15050

```
15000 REM ******************************
15010 REM Klokkestreng
15020 REM******************************
15030 klok$=STR$(1000000+t*10000+m*100+s
)
15040 klok$=MID$(klok$,3,2)+":"+MID$(klo
k$,5,2)+":"+MID$(klok$,7,2)
15050 RETURN
```

Modul 1.3.6: Check af timere og alarmsignal

Dette modul kan bryde ind i kørslen af andre programmoduler, og lade dem vente, mens det checkes om tiden er løbet ud for en af timerne – og er det tilfældet, sættes alarmsignal igang. Vi indlæser modulet nu, selvom det ikke kan anvendes fuldt ud, før en del andre moduler er indlæst. Modulet er nemlig vigtigt for menu-subrutinen, som følger umiddelbart efter.

Modul 1.3.6: Linie 9000-9300

```
9000 REM ******************************
9010 REM Check timer
9020 REM ******************************
9030 IF ttid(0)<>tid THEN RETURN
9040 CLS #2:PRINT #2
9050 GOSUB 12000:INK 2,26,2
9060 PRINT #2,TAB(19);"ALARM";TAB(19);"=
====":LOCATE #2,1,11
```

```
9Ø7Ø mlen=LEN(besked$(Ø))
9Ø8Ø PRINT #2,TAB(19-mlen/2);STRING$(mle
n+4,"*")
9Ø9Ø PRINT #2,TAB(19-mlen/2);"*";TAB(22+
mlen/2);"*"
91ØØ PRINT #2,TAB(19-mlen/2);"* ";besked
$(Ø);" *"
911Ø PRINT #2,TAB(19-mlen/2);"*";TAB(22+
mlen/2);"*"
912Ø PRINT #2,TAB(19-mlen/2);STRING$(mle
n+4,"*")
913Ø tone=25Ø:ttt=TIME
914Ø WHILE INKEY$="" AND TIME<(ttt+18ØØØ
)
915Ø WHILE (SQ(1) AND 7)>Ø
916Ø SOUND 1,tone,15,15
917Ø tone=75Ø-tone
918Ø WEND
919Ø WEND
92ØØ alarm=1
921Ø FOR i=1 TO timers-1
922Ø ttimer(i-1)=ttimer(i)
923Ø tminut(i-1)=tminut(i)
924Ø ttid(i-1)=ttid(i)
925Ø besked$(i-1)=besked$(i)
926Ø NEXT i
927Ø timers=timers-1
928Ø IF timers=Ø THEN ttid(Ø)=-1
929Ø INK 2,26
93ØØ RETURN
```

Kommentar

Linie 9030: Her møder vi for første gang arrayet TTID, som bruges til at huske de tider timerne er indstillet til. I de efterfølgende moduler, hvor brugeren kan indstille timerne, sørges der for at de enkelte tider sættes i rækkefølge, således at den timer, der først udløber, står som element 0 i arrayet, den næste bliver nummer 1 osv. I de allerfleste tilfælde er der kun denne linie af modulet, der køres. Kun i det tilfælde hvor den aktuelle tid TID (som blev udregnet i tidsjusteringsmodulet) svarer til første element i TTID, vil kørslen af dette modul fortsætte og alarmen vil lyde.

Linie 9040-9120: Disse linier ser mere indviklede ud end de faktisk er. Formålet med dem er at udskrive ordet 'ALARM' i vindue 2 og samtidig udskrive den besked, der er knyttet til timeren. Beskeden vil blive indsat i en ramme af stjerner, og den vil blive centreret midt på skærmen efter formlen:

MIDTEN AF SKÆRMEN - LÆNGDEN AF DEN STRENG DER SKAL UDSKRIVES/2

Du kan se formlen anvendt i linie 9090-9120, hvor den skrives '19-mlen/2'.

Linie 9130-9200: Disse linier laver et to-tonet alarmsignal i et minut. Alarmen afbrydes hvis tastaturet røres. Den inderste af de to løkker laver selve lyden, og linie 9150 sikrer, at hver tone holdes tilbage indtil SQ funktionen angiver, at den forrige tone er ophørt. Den ydre løkke bruger maskinens indbyggede ur - TIME sættes lig TTT, og når der er gået et minut er TIME 18000 større, og alarmen afbrydes. Variablen ALARM sættes lig 1 for at fortælle menumodulet at alarmen er blevet afbrudt. Se kommentaren til dette modul.

Linie 9210-9280: Når alarmen er færdig sletter disse linier den fra de arrays, der indeholder tid og besked. Det gøres ved at flytte de data, der gælder timer nummer to, et trin frem - og derved slette den første timer. Variablen TIMERS husker hvor mange timere, der er indstillet, og den nedskrives derfor. Hvis der ikke er nogen timere indstillet, sørger linie 9280 for, at der sættes et 'umuligt' tal først i arrayet, således at alarmen ikke pludselig starter på et tilfældigt tidspunkt.

Modul 1.3.7: Programmenuen

Nu kommer vi til en ny teknik, som kommer til at spille en central rolle i resten af bogens programmer - 'menu-styring'. De tidlige programmer har styret sig selv. Når disse programmer først var startet med RUN, har kontrolmodulet overtaget styringen af programmet indtil brugeren stoppede programmet.

Dette program (og mange af dem der kommer efter) adskiller sig fra denne funktionsmåde ved, at det ikke på forhånd er fastlagt, hvad programmet skal gøre. Programmet giver en række forskellige muligheder, som brugeren må vælge imellem, således at det i stor udstrækning er brugeren, der bestemmer hvad programmet foretager sig. Det sker ved hjælp

af det programmodul, der kaldes *menuen,* som giver brugeren en liste af valgmuligheder som der skal vælges imellem. I nogle af de større programmer i bogen er der endda flere forskellige menuer i programmet, som angiver valgmulighederne i programmets forskellige faser. Men foreløbig skal vi se på det program, hvor der kun er brug for én menu.

Modul 1.3.7: Linie 4000-4170

```
4ØØØ REM ******************************
4Ø1Ø REM Menu
4Ø2Ø REM ******************************
4Ø3Ø WHILE x$<>"5"
4Ø4Ø CLS #2:PRINT #2
4Ø5Ø PRINT #2,TAB(19);"MENU";TAB(19);"==
==":PRINT #2
4Ø6Ø PRINT #2," 1) Indstil en timer"
4Ø7Ø PRINT #2," 2) Slet en timer"
4Ø8Ø PRINT #2," 3) Vis timere og vent"
4Ø9Ø PRINT #2," 4) Slet skærmen og vent"
41ØØ PRINT #2," 5) Stop"
411Ø alarm=Ø:x$=""
412Ø WHILE x$="" AND alarm=Ø
413Ø x$=INKEY$
414Ø IF alarm THEN x$=""
415Ø WEND
416Ø ON VAL(x$) GOSUB 5ØØØ,6ØØØ,7ØØØ,8ØØ
Ø
417Ø PRINT
418Ø WEND
419Ø RETURN
```

Kommentar

Linie 4030-4170: Denne løkke vil udskrive menuen hver gang dette modul køres, indtil brugeren vælger punkt 5 – stop. Når det sker vendes der tilbage til kontrolmodulet, som endnu ikke er indlæst.

Linie 4040-4100: Disse linier udskriver menuen – valgmulighederne – i vindue # 2, som er det vindue, der dækker den største del af skærmen.

Linie 4110-4150: Normalt ville der på dette sted i menuen være en INPUT-sætning. Grunden til at der her bruges en mere indviklet og omfat-

tende fremgangsmåde er, at vi hele tiden ønsker at programmet skal kunne checke timerne – også når menuen er på skærmen. Det kan nemlig ikke ske når INPUT bruges – her spærres programmet indtil der tastes <ENTER>, og end ikke EVERY kan afbryde INPUT.

Metoden er her at den variabel, som skal indeholde brugerens svar (X$), sættes til at være en tom streng, og løkken kører så ligeså længe som strengen er tom. Når brugeren indtaster sit svar, bliver det registreret af INKEY$ i linie 4130, og den pågældende karakter bliver indlæst i X$, og derved stoppes løkken.

Mens tastaturet testes af INKEY$ kaldes det foregående modul hele tiden for at checke om en af timerne er udløbet. Brugen af variablen ALARM er helt enkel: Normalt vil der ikke ske noget når det foregående modul kaldes, og menuen forbliver derfor på skærmen. Men i de tilfælde, hvor alarmen sætter i gang, slettes menuen fordi alarm og besked skal udskrives på skærmen – og derfor sættes ALARM lig 1 for at fortælle menumodulet, at menuen er udskrevet på skærmen igen.

Linie 4140: Brugerens svar oversættes – hvis det er muligt – til et tal med VAL. ON...GOSUB kommandoen vælger det linienummer, der svarer til brugerens svar. Hvis der indtastes et tal som ikke passer med GOSUB-linien, dvs. hvis tallet er nul eller større end antallet af linienumre i GOSUB-linien, så reagerer ON...GOSUB ikke og menuen udskrives igen på skærmen.

Test

Før vi laver en test af de moduler vi har indlæst, er det en god ide at indlæse kontrolmodulet også.

Modul 1.3.8: Kontrolmodulet

Dette modul er enklere end i de andre programmer fordi meget af styringen klares med menuen. Også her er programmets 'urværk' EVERY. I dette tilfælde justeres tiden hver femte sekund.

Modul 1.3.8: Linie 1000-1080

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
```

```
1Ø3Ø GOSUB 3ØØØ
1Ø4Ø EVERY 25Ø GOSUB 14ØØØ
1Ø5Ø GOSUB 2ØØØ
1Ø6Ø GOSUB 4ØØØ
1Ø7Ø MODE 1
1Ø8Ø END
```

Test

Hvis programmet køres nu bedes der først om angivelse af hvad klokken er. Derefter kommer programmenuen frem. Prøv at indtaste nogle 'ugyldige' svar for at checke, at de bliver afvist. Hvis tallene 1 - 4 vælges stopper programmet med meldingen 'Line does not exist', mens et 5-tal vil slette skærmen og afslutte programmet.

Modul 1.3.9: Visning af de indstillede timere

Formålet med dette modul er at lave en pæn udskrift på skærmen, som angiver de timere (op til ialt 16), der er indstillet. Da vi endnu ikke har indlæst det modul, der klarer indstilling af timerne, kommer der intet frem på skærmen, når modulet køres. Men når indstillingsmodulet er indlæst, kan vi straks teste dem begge.

Modul 1.3.9: Linie 7000-7170

```
7ØØØ REM ******************************
7Ø1Ø REM Vis timere og vent
7Ø2Ø REM ******************************
7Ø3Ø k$=""
7Ø4Ø WHILE k$=""
7Ø5Ø CLS #2:PRINT #2
7Ø6Ø PRINT #2,TAB(18);"TIMERE";TAB(18);"
======":PRINT #2
7Ø7Ø FOR i=Ø TO timers-1
7Ø8Ø t=ttimer(i):m=tminut(i):s=Ø:GOSUB 1
5ØØØ
7Ø9Ø PRINT #2,USING " ##) &";i+1;klok$+"
 "+besked$(i)
71ØØ NEXT i
711Ø alarm=Ø
712Ø WHILE k$="" AND alarm=Ø
```

```
713Ø k$=INKEY$
714Ø IF alarm=1 THEN k$=""
715Ø WEND
716Ø WEND
717Ø RETURN
```

Kommentar

Linie 7030-7040 og 7160: Listen over timere vil blive på skærmen indtil en tilfældig tast trykkes ned.

Linie 7070-7100: Som tidligere bemærket, bliver antallet af indstillede timere indlæst i variablen TIMERS. Denne løkke udskriver det antal timer-indstillinger, som angives af TIMERS. Modul 1.3.5 bruges til at lave en streng som indeholder tiden, som ligger i de to arrays TTIMERS og TMINUT – og PRINT USING sikrer at udskriften bliver pæn at se på. '&' i PRINT USING kan måske godt virke lidt forvirrende – men det giver os mulighed for at have både et tal og en streng i samme udstryk. Hvis vi ikke havde haft '&' med ville vi have fået en 'Type mismatch'-fejl når programmet skulle til at læse TID$ som et tal i linie 7090.

Linie 7110-7150: Ligesom i menuen, laves her en venteposition, der gør det muligt for programmet at checke om en af timerne udløber. Bemærk, at hvis alarmen starter, mens denne løkke kører, vil en indtastning afbryde alarmsignalet, mens timerlisten forbliver på skærmen. Linie 7140 sikrer, at hvis variablen ALARM viser at alarmen er blevet afbrudt, så skal der tastes en gang til for at få menuen frem.

Test

Skriv:

CLEAR:TIMERS=5:GOTO 7000 <ENTER>

Nu skal tiden for fem timere komme på skærmen, – tiden for dem alle er '00-00-00' og der udskrives ikke nogen besked ud for dem. Hvis en tilfældig tast trykkes ned kommer fejlmeldingen 'Unexpected RETURN'.

Modul 1.3.10: Indstilling af absolut timer

Vi skal straks til at indlæse det modul hvor vi indstiller timerne, men først skal vi indlæse to andre moduler, som giver brugeren mulighed for

at vælge mellem to forskellige måder at indstille timeren på. Når en timer indstilles, kan brugeren således vælge om alarmtidspunktet skal angives i absolut tid (dvs. som et bestemt klokkeslet), eller om tiden skal angives som nedtælling (dvs. hvor lang tid der skal går inden, alarmen skal starte). Dette modul tager sig af den absolutte tid, og det svarer i opbygning til det modul hvor brugeren angiver, hvad klokken er.

Modul 1.3.10: Linie 11000-11070

```
11ØØØ REM ****************************
11Ø1Ø REM Indstil almindelig timer
11Ø2Ø REM ****************************
11Ø3Ø INPUT "Timer (Ø to 23):",ttimer
11Ø4Ø IF ttimer<Ø OR ttimer>23 THEN PRIN
T " **** UDENFOR OMRÅDET - PRØV IGEN ***
*":FOR i=1 TO 2ØØØ:NEXT:GOTO 11Ø3Ø
11Ø5Ø INPUT "Minutter (Ø to 59):",tminut
11Ø6Ø IF tminut<Ø OR tminut>59 THEN PRIN
T " **** UDENFOR OMRÅDET - PRØV IGEN ***
*":FOR i=1 TO 2ØØØ:NEXT:GOTO 11Ø5Ø
11Ø7Ø RETURN
```

Modul 1.3.11: Indstilling af nedtællertimer

Dette andet modul er nødvendigvis lidt mere indviklet end det første, idet det ikke blot angives hvornår timeren skal udløbe. Programmet skal derfor lægge den angivne periode til det aktuelle klokkeslet.

Modul 1.3.11: Linie 10000-10090

```
1ØØØØ REM ****************************
1ØØ1Ø REM Indstil til nedtælling
1ØØ2Ø REM ****************************
1ØØ3Ø INPUT "Antal timer tilbage (Ø to 2
3):",ttimer
1ØØ4Ø IF ttimer<Ø OR ttimer>23 THEN PRIN
T " **** UDENFOR OMRÅDET - PRØV IGEN ***
*":FOR i=1 TO 2ØØØ:NEXT:GOTO 1ØØ3Ø
1ØØ5Ø INPUT "Antal minutter tilbage (Ø t
o 59):",tminut
1ØØ6Ø IF tminut<Ø OR tminut>59 THEN PRIN
```

```
T " **** UDENFOR OMRÅDET - PRØV IGEN ***
*":FOR i=1 TO 2ØØØ:NEXT:GOTO 1ØØ5Ø
1ØØ7Ø tminut=tminut+minut:IF tminut>59 T
HEN tminut=tminut-6Ø:ttimer=ttimer+1
1ØØ8Ø ttimer=ttimer+timer:IF ttimer>23 T
HEN ttimer=ttimer-24
1ØØ9Ø RETURN
```

Kommentar

Linie 10070-10080: Det aktuelle klokkeslet fås fra variablerne TIMER og MINUT, som løbende vil blive justeret af EVERY, når programmet er færdigt. Her lægges nedtællingstiderne TTIMER og TMINUT til hhv. TIMER og MINUT.

Modul 1.3.12: Timerne indstilles

Nu da vi kan indstille den rigtige tid for en timer, kan vi gå videre til hovedmodulet, hvor tiden for op til 16 timere kan indstilles. Bemærk at der her bruges INPUT til indlæsning af de tre nødvendige sæt data, – det betyder at alarmen ikke kan gå igang, mens dette modul er i funktion. Imidlertid husker maskinen en eventuel alarm, som så starter så snart modulet er kørt.

Bemærk også, at du ikke kan lade maskinen vente i al for lang tid på et svar på en INPUT-sætning. For mens der ventes, registrerer maskinen hver gang EVERY-kommandoen skulle have været udført. Til sidst bliver den plads, der er reserveret til dette formål opbrugt, og det betyder i sidste ende, at programmets urværk kommer til at gå forkert.

Modul 1.3.12: Linie 5000-5300

```
5ØØØ REM ******************************
5Ø1Ø REM Indstil timere
5Ø2Ø REM ******************************
5Ø3Ø IF timers=16 THEN PRINT "  ***** IN
GEN TIMERE LEDIGE *****":FOR j=1 TO 2ØØØ
:NEXT:RETURN
5Ø4Ø INPUT "Nedtælling (j/n)";ned$
5Ø5Ø IF LOWER$(ned$)="j" THEN GOSUB 1ØØØ
Ø ELSE GOSUB 11ØØØ
```

```
5Ø6Ø ttid=ttimer*6Ø+tminut
5Ø7Ø rtid1=ttid-tid:IF rtid1<=Ø THEN rti
d1=rtid1+144Ø
5Ø8Ø FOR i=Ø TO timers-1
5Ø9Ø rtid2=ttid(i)-tid:IF rtid2<Ø THEN r
tid2=rtid2+144Ø
51ØØ IF rtid1>rtid2 THEN NEXT i
511Ø IF i<timers AND ttid=ttid(i) THEN P
RINT "   ***** TIMER ER INDSTILLET *****
":FOR j=1 TO 2ØØØ:NEXT:RETURN
512Ø tnum=i
513Ø besked$=""
514Ø WHILE besked$=""
515Ø INPUT "Besked";besked$
516Ø IF LEN(besked$)>25 THEN PRINT "** B
ESKED ER FOR LANG (MAX. 25 TEGN) **":FOR
 i=1 TO 2ØØØ:NEXT:besked$=""
517Ø WEND
518Ø PRINT
519Ø FOR i=timers-1 TO tnum STEP -1
52ØØ ttimer(i+1)=ttimer(i)
521Ø tminut(i+1)=tminut(i)
522Ø ttid(i+1)=ttid(i)
523Ø besked$(i+1)=besked$(i)
524Ø NEXT i
525Ø timers=timers+1
526Ø ttimer(tnum)=ttimer
527Ø tminut(tnum)=tminut
528Ø ttid(tnum)=ttid
529Ø besked$(tnum)=besked$
53ØØ RETURN
```

Kommentar

Linie 5030: Hvis antallet at timere allerede er 16, som er det maximale antal, laves her en fejlmelding og programmet vender tilbage til menuen.

Linie 5040-5050: Brugeren spørges her om det skal være en nedtællingstimer. Linie 5050 oversætter svaret til et lille bogstav, hvis det er nødvendigt, således at både 'J' og 'j' kan bruges. Derefter kaldes et af de foregående moduler.

Linie 5060-5070: Den tid, der angives af et af de to foregående moduler omregnes til minutter. Det aktuelle klokkeslet omregnes ligeledes og trækkes fra. Hvis resultatet er mindre end nul bliver der lagt 1440 minutter til alarmtidspunktet (svarende til én dag). Hvis klokken f.eks. er 1100 (elleve) og timeren indstilles til klokken 1000 (ti), så finder programmet ud af at alarmen skal lyde næste dag klokken ti.

Linie 5080-5120: Programmet lagrer timerne i den rigtige rækkefølge, dvs. med den timer, der først udløber, som den første osv., og denne løkke sørger for at nye timerindstillinger bliver sammenlignet med dem der er i forvejen, så de kan komme til at stå rigtigt i listen.

Hvis den (de) første timer(e) i listen er indstillet til tider tidligere end det aktuelle klokkeslet, så bliver der også lagt 1440 til dem, således at der kan foretages en sammenligning. Hvis der findes en tid som er senere end den nye timerindstilling, så afsluttes løkken af IF-sætningen i linie 5100 i FOR...NEXT løkken. Hvis ingen af timerindstillingerne ligger efter den nye timerindstilling kører løkken videre indtil I er lig TIMERS, og løkkevariablen bliver (som altid) opskrevet (med 1) idet løkken kører færdig.

Værdien af I vil derfor nu *enten* pege på en timer som ligger senere end den nye timer, *eller* på den første plads efter listen af timere. Nu laves der yderligere en test, for at undersøge om der allerede er indstillet en timer til det pågældende tidspunkt. Hvis det er tilfældet laves der en fejlmelding, og der vendes tilbage til menuen. Hvis tidspunktet ikke er 'optaget' sættes den nye timers plads TNUM lig I.

Linie 5130-5170: Her bedes om den besked, som skal knyttes til timeren.

Linie 5190-5290: Nu ved vi vor den nye timer skal stå i listen, og vi kan nu flytte timerne fra denne position og opefter et trin, så der bliver et 'hul'. Der er knyttet fire sæt data til hver timer og de ligger i disse fire arrays: TTIMER, TMINUT, TTID og BESKED$.

Linie 5250-5290: Nu bliver variablen TIMERS opskrevet, for at angive at der er kommet en timer mere med i listen, og de fire sæt data indsættes i de fire arrays på pladsen angivet af TNUM.

Test

Nu skal det være muligt at teste modulet ved at køre hele programmet. Når menuen kommer frem så vælg '1' og indtast

J, 0, 1 og TEST

som svar på de fire spørgsmål. Når menuen kommer frem igen så vælg '3' for at få timerne frem på skærmen. Nu vil du se at timeren har fået en tid der er mindre end ét minut fra hvad klokken er (afhængigt af hvor hurtig du har været), og den har fået 'TEST' knyttet til sig.

Modul 1.3.13: Sletning af skærmen

Selve ideen med dette program er at det skal kunne køre i baggrunden, mens man foretager sig andre ting, – din Amstrad kan på den måde f.eks. bruges som vækkeur. Maskinen tager ikke spor skade af at være tændt igennem længere tid, men monitoren har ikke godt af at stå med det samme skærmbillede i længere perioder. På længere sigt kan det betyde at skærmbelægningen bliver 'slidt' i skærmbilledets lyse områder. På den måde kan der komme nogle lyse 'skygger' på din monitor. (Men der er ikke grund til at være nervøs for din monitor, blot fordi den står tændt med det samme skærmbillede i nogle få timer – problemet kommer først, når det sker i længere perioder.)

For at slippe udenom dette problem, er der indbygget en beskyttelse af skærmen i programmet, således at skærmen står uden skærmbillede indtil en tilfældig tast trykkes ned. Timerne checkes og alarmen starter på normal måde, når en af timerne løber ud.

Modulet her giver alle INK-farver baggrundsfarve. Det næste modul kalder dette modul og sætter maskinen i venteposition, indtil skærmen skal bruges igen.

Modul 1.3.13: Linie 13000-13080

```
13ØØØ REM ****************************
13Ø1Ø REM Slet skærmen
13Ø2Ø REM ****************************
13Ø3Ø INK Ø,Ø
13Ø4Ø INK 1,Ø
13Ø5Ø INK 2,Ø
13Ø6Ø INK 3,Ø
13Ø7Ø BORDER Ø
13Ø8Ø RETURN
```

Modul 1.3.14: Venteposition med slettet skærm

Dette modul er blot en løkke, der venter på samme måde som løkken i

f.eks. menuen. Derudover kalder dette modul det foregående, så vi kan få en 'sort skærm'. Når en tilfældig tast trykkes ned får skærmen de rigtige farver igen.

Modul 1.3.14: Linie 8000-8130

```
8ØØØ REM ******************************
8Ø1Ø REM Slet skærmen og vent
8Ø2Ø REM ******************************
8Ø3Ø k$=""
8Ø4Ø WHILE k$=""
8Ø5Ø GOSUB 13ØØØ
8Ø6Ø alarm=Ø
8Ø7Ø WHILE k$="" AND alarm=Ø
8Ø8Ø k$=INKEY$
8Ø9Ø IF alarm=1 THEN k$=""
81ØØ WEND
811Ø WEND
812Ø GOSUB 12ØØØ
813Ø RETURN
```

Test

Indstil en eller to timere til at udløbe efter en kort periode. Vælg nu '4' på menuen og vent. Alarmen skal så komme frem på normal vis, når tiden er inde.

Modul 1.3.15: Slet en timer

Nogle gange bliver en timer indstillet forkert, eller måske bliver den overflødig af en eller anden grund. Dette modul giver så mulighed for at fjerne en sådan timer igen. Den grundliggende teknik her er meget anvendt – og den er, som du kan se, ret enkel.

Modul 1.3.15: Linie 6000-6140

```
6ØØØ REM ******************************
6Ø1Ø REM Slet timer
6Ø2Ø REM ******************************
6Ø3Ø IF timers=Ø THEN PRINT "  **** INGE
N TIMERE INDSTILLET ****":FOR i=1 TO 2ØØ
```

```
Ø:NEXT:RETURN
6Ø4Ø INPUT "Hvilken timer";tnum
6Ø5Ø IF tnum<1 OR tnum>timers THEN PRINT
 "  **** DEN TIMER FINDES IKKE ****":FOR
 i=1 TO 2ØØØ:NEXT:RETURN
6Ø6Ø FOR i=tnum TO timers-1
6Ø7Ø ttimer(i-1)=ttimer(i)
6Ø8Ø tminut(i-1)=tminut(i)
6Ø9Ø ttid(i-1)=ttid(i)
61ØØ besked$(i-1)=besked$(i)
611Ø NEXT i
612Ø timers=timers-1
613Ø IF timers=Ø THEN ttid(Ø)=Ø
614Ø RETURN
```

Kommentar

Linie 6030-6050: Her gives fejlmeldinger hvis der ikke er nogen timere at slette, eller hvis den angivne timer ikke findes.

Linie 6060-6110: Her slettes timeren ved at alle data der ligger ovenfor rykkes en plads frem.

Test

Nu skal det være muligt at vælge '2' og slette en timer før den løber ud. Hvis denne test er OK skulle programmet være helt klar til brug.

Program 1.4: Stopur

Det sidste program i kapitel 1, hvor vi arbejder med tid, er ret usædvanligt idet vi her laver Amstrad'en om til et stopur. Selvfølgelig er en lille computer ikke så præcis som specielle elektroniske stopure, men computeren har til gengæld den fordel at den kan huske mellemtider, lave beregninger med tiderne osv. Dette program er beregnet til at tage tid på en eller flere hændelser, udskrive tider og mellemtider – og eventuelt knytte en lille kommentar til tiderne. Derudover kan programmet lave en udskrift på en printer, så man kan få det hele på tryk.

```
10:46:01 (00:00:44)
10:46:15 (00:00:14)
10:46:22 (00:00:07) bus
10:46:27 (00:00:05)
10:46:36 (00:00:09)
10:46:48 (00:00:12) tankvogn
10:46:57 (00:00:08)
10:47:05 (00:00:08)
10:47:15 (00:00:10) ambulance
10:47:27 (00:00:12)
10:47:30 (00:00:03)
10:47:32 (00:00:02)
10:47:40 (00:00:08) blokvogn
10:47:45 (00:00:05)
10:47:51 (00:00:06)
10:48:01 (00:00:10) turistbus
10:48:07 (00:00:06)
10:48:12 (00:00:04)
10:48:19 (00:00:07)
```

Fig. 1.4: Printerudskrift fra Stopur.

I dette program præsenteres følgende:

1) Udregning af tider.
2) Udskrift på printer.

Modul 1.4.1: Initialisering

Et standard initialiseringsmodul hvor der bl.a. defineres tre vinduer: en linie øverst (#1), hovedskærmen (#2) og en linie nederst, hvor udskrifter uden datastrømnummer kommer frem.

Modul 1.4.1: Linie 2000-2160

```
2ØØØ REM ******************************
2Ø1Ø REM Initialisering
2Ø2Ø REM ******************************
2Ø3Ø MODE 1
2Ø4Ø INK Ø,1
2Ø5Ø INK 1,24
2Ø6Ø INK 2,26
```

```
2Ø7Ø INK 3,2
2Ø8Ø BORDER 1
2Ø9Ø WINDOW 1,4Ø,25,25
21ØØ PAPER Ø:PEN 1:CLS
211Ø WINDOW #1,4,36,1,1
212Ø PAPER #1,Ø:PEN #1,2:CLS #1
213Ø PRINT #1,"Klokken er:"
214Ø WINDOW #2,1,4Ø,3,23
215Ø PAPER #2,3:PEN #2,2:CLS #2:PRINT #2
216Ø RETURN
```

Modul 1.4.2: Indstilling af uret

Dette modul kendes fra de tidligere programmer.

Modul 1.4.2: Linie 3000-3090

```
3ØØØ REM ******************************
3Ø1Ø REM Indstilling af uret
3Ø2Ø REM ******************************
3Ø3Ø CLS:PRINT"Klokken er:":PRINT
3Ø4Ø INPUT "Timer (1-12):";timer
3Ø5Ø IF timer<1 OR timer>12 THEN PRINT "
***** Udenfor området - prøv igen *****"
:GOTO 3Ø4Ø
3Ø6Ø INPUT "Minutter (Ø-59):";minut
3Ø7Ø IF minut<Ø OR minut>59 THEN PRINT "
***** Udenfor området - prøv igen *****"
:GOTO 3Ø6Ø
3Ø8Ø sekund=Ø
3Ø9Ø RETURN
```

Modul 1.4.3: Vent...

INKEY$ bruges også her for at undgå problemerne med EVERY og INPUT. Udover at maskinen sættes i venteposition, sørger modulet også for at maskinen kan modtage besked om at give udskrift på en printer.

Modul 1.4.3: Linie 4000-4180

```
4000 REM ******************************
4010 REM Vent
4020 REM ******************************
4030 t$="":besked$=""
4040 WHILE t$=""
4050 WHILE t$=""
4060 t$=INKEY$
4070 WEND
4080 WHILE LOWER$(t$)="p"
4090 printer=1-printer
4100 t$=""
4110 WEND
4120 WHILE t$=CHR$(13)
4130 INPUT "Besked (max. 18 tegn):",besk
ed$
4140 IF LEN(besked$)<=18 THEN t$="*"
4150 WEND
4160 CLS
4170 WEND
4180 RETURN
```

Kommentar

Linie 4030: Strengen T$ bruges til at huske hvilken tast brugeren trykker ned.

Linie 4040-4170: Den store løkke her kører indtil T$ tildeles et indhold – enten af brugeren eller af programmet.

Linie 4050-4070: Dette er selve ventepositionen, som fortsætter indtil en tast trykkes ned.

Linie 4080-4110: Hvis der tastes 'P' skiftes værdien af variablen PRINTER mellem 0 og 1. Det er nok værd at se nærmere på denne enkle linie, for den viser en helt klassisk måde at skifte en variabel frem og tilbage mellem to værdier. Hvis du således har en variabel X som skal skifte mellem værdierne V1 og V2 (hvor V1 er den mindste), kan det gøres således:

X = V1 + V2 - X

men da den mindste værdi her er nul, kan det skæres ned til:

X = V2 - X

Linie 4120-4150: Hvis det er <ENTER>-tasten der trykkes ned, bliver brugeren bedt om at skrive en lille besked i forbindelse med en mellemtid. Det sker i linien nederst på skærmen, som slettes igen når beskeden er skrevet.

Test

Kør først initialiseringsmodulet, og skriv så:

GOTO 4000 <ENTER>

og maskinen skal blive stillet i venteposition. Hvis du taster 'P' sker der ikke noget på skærmen. Hvis du taster <ENTER> bliver du bedt om at afgive en lille besked nederst på skærmen. Trykker du på en af de andre taster stopper programkørslen.

Modul 1.4.4: Justering af tiden

Dette modul svarer til tidsjusteringsmodulerne i de to tidligere programmer, bortset fra at det her er to tider der bliver justeret: den almindelige tid og tiden siden en tast sidst blev trykket ned. De variabler, der indeholder mellemtiderne, starter med bogstavet P.

Modul 1.4.4: Linie 6000-6280

```
6ØØØ REM ******************************
6Ø1Ø REM Justering af tiden
6Ø2Ø REM ******************************
6Ø3Ø sekund=sekund+1
6Ø4Ø WHILE sekund>59
6Ø5Ø sekund=Ø:minut=minut+1
6Ø6Ø WHILE minut>59
6Ø7Ø minut=Ø:timer=timer+1
6Ø8Ø WHILE timer>12
6Ø9Ø timer=1
61ØØ WEND
611Ø WEND
612Ø WEND
```

```
613Ø psekund=psekund+1
614Ø WHILE psekund>59
615Ø psekund=Ø:pminut=pminut+1
616Ø WHILE pminut>59
617Ø pminut=Ø:ptimer=ptimer+1
618Ø WHILE ptimer>12
619Ø ptimer=1
62ØØ WEND
621Ø WEND
622Ø WEND
623Ø LOCATE#1,14,1
624Ø t=timer:m=minut:s=sekund:GOSUB 7ØØØ
625Ø PRINT#1,tid$;
626Ø t=ptimer:m=pminut:s=psekund:GOSUB 7
ØØØ
627Ø PRINT#1," (";tid$;")"
628Ø RETURN
```

Test

Dette modul kan ikke rigtig testes før det næste modul (hvor tiderne formatteres) er indlæst.

Modul 1.4.5: Formattering af tiden

Dette modul former, som i de to forrige programmer tiden om til en streng.

Modul 1.4.5: Linie 7000-7050

```
7ØØØ REM ******************************
7Ø1Ø REM Klokkestreng
7Ø2Ø REM ******************************
7Ø3Ø tid$=STR$(1ØØØØØØ+t*1ØØØØ+m*1ØØ+s)
7Ø4Ø tid$=MID$(tid$,3,2)+":"+MID$(tid$,5
,2)+":"+MID$(tid$,7,2)
7Ø5Ø RETURN
```

Test

Skriv:

CLEAR:CLS:GOTO 6000 <ENTER>

og følgende udskrift skal komme frem øverst på skærmen:

00:00:01 (00:00:01)

Hvis du indlæser GOTO 6000 flere gange bliver tiden opskrevet med et sekund hver gang.

Modul 1.4.6: Udskrift af resultaterne

Da vi nu har indlæst moduler, som stiller maskinen i venteposition og som kan justere tiden løbende, skal vi nu have et modul, som laver udskrift på skærm og printer, når en tast trykkes ned.

Modul 1.4.6: Linie 5000-5100

```
5ØØØ REM ******************************
5Ø1Ø REM Udskrifter
5Ø2Ø REM ******************************
5Ø3Ø t=timer:m=minut:s=sekund:GOSUB 7ØØØ
5Ø4Ø p$=" "+tid$
5Ø5Ø t=ptimer:m=pminut:s=psekund:GOSUB 7
ØØØ
5Ø6Ø p$=p$+" ("+tid$+") "+besked$
5Ø7Ø PRINT#2,p$
5Ø8Ø IF printer=1 THEN PRINT#8,p$
5Ø9Ø ptimer=Ø:pminut=Ø:psekund=Ø
51ØØ RETURN
```

Kommentar

Linie 5030-5070: Her laves en streng, bestående af den aktuelle tid (TIMER, MINUT, SEKUND), efterfulgt af mellemtiden (PTIMER,PMINUT,PSEKUND) og en eventuel besked, som brugeren har skrevet. Denne streng udskrives i vindue #2.

Linie 5080: Amstrad har i modsætning til de fleste andre en fast opkobling til en tilsluttet printer. På de fleste andre computere ville det være nødvendigt at 'åbne en fil til printeren', – noget der fylder adskillige linier i BASIC. På Amstrad'en er det nok blot at sende det, der skal skrives på printeren, via datastrøm 8.

Linie 5090: Mellemtiderne nulstilles, da de kun skal gælde tiden siden en tast sidst blev trykket ned.

Modul 1.4.7: Kontrolmodulet

Sidste modul er kontrolmodulet, der binder de enkelte moduler sammen.

Modul 1.4.7: Linie 1000-1110

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
1Ø3Ø ON BREAK GOSUB 111Ø
1Ø4Ø GOSUB 3ØØØ
1Ø5Ø EVERY 5Ø GOSUB 6ØØØ
1Ø6Ø GOSUB 2ØØØ
1Ø7Ø WHILE 1
1Ø8Ø GOSUB 4ØØØ
1Ø9Ø GOSUB 5ØØØ
11ØØ WEND
111Ø CLEAR:MODE 1:END
```

Kommentar

Linie 1070: En lille finesse som du måske kan bruge i andre sammenhænge – 1-tallet i forbindelse med WHILE får løkken til at fortsætte i det uendelige. Alle positive tal ville have samme funktion.

Afslutning

Der kan læres en hel del ting af programmerne i dette kapitel, – ikke mindst, at det kun er fantasien der sætter grænserne for, hvad din Amstrad kan bruges til. Vigtigst er nok, at du kan se hvor nemt det er at lave velfungerende programmer, når det hele opbygges i moduler, hvoraf mange kan bruges i forskellige programmer. Med Amstrads MERGE kommando er det så let, at maskinen næsten råber på at blive brugt på denne måde. Når du først har opbygget en større samling subrutiner, bliver meget af din programmering næsten som at lægge et puslespil. Men når programmet er blevet 'samlet' kan det som regel bruges til mere end et færdigt puslespil.

KAPITEL 2

Grafik

I dette kapitel går vi fra tidseksperimenterne til noget andet, som Amstrad også er meget god til: at fremstille tal og tabeller på en meget overskuelig og forståelig måde. Dette kapitel består af tre programmer, der laver forskellige grafer i både lav- og højopløsningsgrafik.

Hen igennem dette og det næste kapitel bliver forklaringerne efterhånden mere kortfattede, undtagen når det drejer sig om nye ideer og mere vanskelige moduler. Det sker som et forsøg på at finde en balance mellem forklaringerne og selve programmerne – som jo er kernen i bogen. *Hvis* du på et tidspunkt kører fast, vil det som regel være en god hjælp at se på de tidligere moduler i bogen, hvor tilsvarende emner blev behandlet, og læse kommentarerne der inden du går videre. Det er også årsagen til, at det fra start blev anbefalet at tage programmerne i rækkefølge, fremfor at springe frem til de mest indviklede programmer med det samme.

Programmerne i dette kapitel er:

GRAF: Hvor der laves en flot grafisk kurve i højopløsning.

CIRKELDIAGRAM: Hvor fordelingen af en begrænset mængde data illustreres som udsnit af en cirkel i flere farver.

3D GRAF: Hvor der laves et flot 'tre-dimensionalt' søjlediagram.

Program 2.1: Graf

Noget af det der tager mest tid og kræver mest hukommelsesplads er programmer, der bruger store mængder data. Det gør det ofte nødvendigt at have ret store programafsnit, som bruges til dataindlæsning og til ændringer i indlæste data.

I de første to programmer i dette kapitel prøver vi at komme rundt om disse begrænsninger ved at lave programmer, som er nemme at bruge, men som ikke bruger en masse hukommelsesplads på at bede brugeren om at indlæse data, på at lagre data i arrays eller lagre data på bånd. Interaktive programmer – dvs. programmer der, beder brugeren om indlæsninger, mens programmet kører – er meget rare at bruge, men de kan være lidt af en luksus, når der ikke er en kæmpe hukommelse til rådighed. I dette program, som tegner om kurver i højopløsning, bruger vi DATA-linier til at lave et program, som er enkelt at bruge, og som tilmed er enklere at skrive end et tilsvarende interaktivt program ville være.

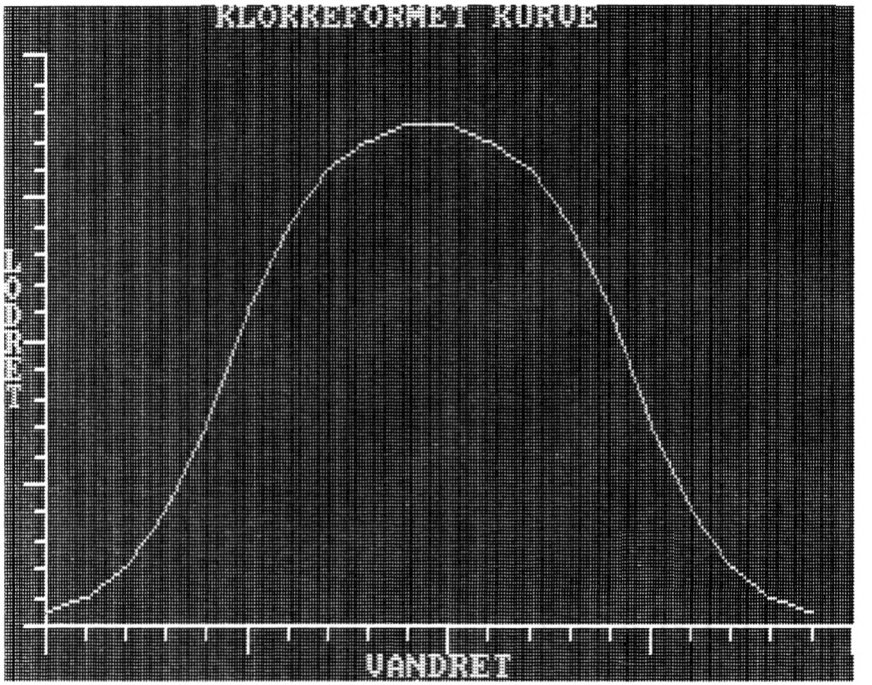


Fig. 2.1: Skærmdump fra programmet.

I dette program præsenteres:

1) Flexibel brug af DATA-linier.

Modul 2.1.1 og 2.1.2: Data til grafen

Disse to moduler er nøglen til programmets enkle opbygning. I disse få linier findes alle de data, som er nødvendige for at tegne en nydelig kurve, – med klare angivelser således at brugeren ikke får besvær med at få tegnet en graf. Tænk ikke på hvad indholdet i DATA-linierne skal bruges til – det bliver klart når programmet er indlæst og afprøvet et par gange med forskellige ændringer.

Modul 2.1.1 og 2.1.2: Linie 3000-4050

```
3ØØØ REM ******************************
3Ø1Ø REM Data 1
3Ø2Ø REM ******************************
3Ø3Ø DATA GRAFENS NAVN,KLOKKEFORMET KURV
E
3Ø4Ø DATA X-AKSENS NAVN,X AKSE
3Ø5Ø DATA Y-AKSENS NAVN,Y AKSE
3Ø6Ø DATA ANTAL ENHEDER/Y-AKSEN,2Ø
3Ø7Ø DATA ANTAL ENHEDER/X-AKSEN,2Ø
3Ø8Ø DATA MÆNGDE PR. Y-ENHED,1Ø
4ØØØ REM ******************************
4Ø1Ø REM Data 2
4Ø2Ø REM ******************************
4Ø3Ø DATA 5,1Ø,2Ø,4Ø,7Ø,11Ø,14Ø,16Ø,17Ø,
175
4Ø4Ø DATA 175,17Ø,16Ø,14Ø,11Ø,7Ø,4Ø,2Ø,1
Ø,5
4Ø5Ø DATA END
```

Kommentar

Linie 3030-3050: Her angives det navn som brugeren ønsker at give figuren, samt oplysninger om de to akser og enhederne. Bemærk at den del af linierne, der står foran kommaet *ikke* kommer med på skærmen – det er udelukkende en hjælp til brugeren. Men kommaet skal altså være der – ellers kommer hele datalinien med.

Linie 3060-3070: Grafens to akser bliver opdelt i enheder for at gøre en aflæsning nemmere. Akserne får altid samme længde, men brugeren kan bestemme hvor mange enheder hver af dem skal inddeles i.

Linie 3080: Hvis kurven – for at tage et eksempel – skal vise udviklingen i kornproduktionen i et land over en årrække, kan brugeren f.eks. bestemme at lade hver enhed på Y-aksen være 1000 tons. Fremfor at brugeren skal dividere alle tallene med 1000 før de indlæses, angives enhedsstørrelsen her, således at tallene kan bruges som de er.

Linie 4030-4040: Her angives de data der danner grundlaget for grafen. Tallene i dette program vil give en nydelig klokkeformet kurve. Bemærk at der er angivet ét tal for hver enhed på X-aksen – med start i aksernes skæringspunkt. Det er ikke nødvendigt at bruge hele X-aksen, og der kan bruges så mange data man vil i hver linie.

Linie 4050: Man kan bruge så mange data som maskinens hukommelse har plads til, blot skal rækken af data afsluttes med kommandoen END i en DATA-linie. Det fortæller programmet at der ikke skal bruges flere data til grafen – også selvom der eventuelt kommer flere data efter.

Modul 2.1.3: Tegning af X- og Y-akserne

I dette modul tegnes selve koordinatsystemet, – med enheder og navne på akserne.

Modul 2.1.3: Linie 1000-1190

```
1ØØØ REM ******************************
1Ø1Ø REM Tegn koordinatsystemet
1Ø2Ø REM ******************************
1Ø3Ø MODE 1
1Ø4Ø MOVE 32,368
1Ø5Ø DRAW 32,32,2
1Ø6Ø DRAW 639,32
1Ø7Ø RESTORE 3ØØØ
1Ø8Ø READ t$,t$:LOCATE 2Ø-LEN(t$)/2,1:PR
INT t$
1Ø9Ø READ t$,t$:LOCATE 21-LEN(t$)/2,25:P
RINT t$
11ØØ READ t$,t$:FOR i=1 TO LEN(t$):LOCAT
E 1,12-LEN(t$)/2+i:PRINT MID$(t$,i,1):NEXT i
111Ø READ t$,ay:py=338/ay
112Ø FOR i=Ø TO ay
113Ø MOVE 24+8*(i/5=INT(i/5)),32+i*py:DR
AW 32,32+i*py
```

```
114Ø NEXT i
115Ø READ t$,ax:px=6Ø6/ax
116Ø FOR i=Ø TO ax
117Ø MOVE 32+i*px,24+8*(i/5=INT(i/5)):DR
AW 32+i*px,32
118Ø NEXT i
119Ø READ t$,enhed
```

Kommentar

Linie 1040-1060: Her tegnes de to akser. Y-aksen går fra et punkt nær skærmens øverste venstre hjørne til et punkt nær skærmens nederste venstre. Herfra går Y-aksen vinkelret over til et punkt nær skærmens nederste højre hjørne.

Linie 1070: Maskinens *pointer* ('pegepind') sættes her til den første DATA-linie i det modul der starter med linie 3000. Brugen af RESTORE sikrer her at vi ikke får en 'DATA exhausted' melding hvis programmet startes med GOTO. Med RUN sættes *pointeren* i alle tilfælde ved den første DATA-linie i programmet.

Linie 1080-1100: Grafens navn og navnene på de to akser læses fra DATA-linierne og skrives ud på skærmen. Navnet på Y-aksen udskrives af en løkke – bogstav for bogstav – langs skærmens venstre kant. Læg mærke til at der i hvert tilfælde er to READ kommandoer. Den første læser teksten *foran* kommaet, og dette slettes straks ved at den anden læser et nyt indhold ind i den samme variabel, T$.

Linie 1110: Antallet af inddelinger på Y-aksen (IY) læses fra den næste DATA-linie. Længden af Y-aksen (338 pixels) divideres derefter med dette tal for at få længden af hver inddeling i pixels (PY).

Linie 1120-1140: Denne løkke tegner de små afmærkninger på Y-aksen efter brugerens angivelser. Læg specielt mærke til udtrykket 89(I/5= INT(I/5), som laver hver femte afmærkning længere end de øvrige. For at forstå udtrykket er det nødvendigt at vide noget om hvordan Amstrad behandler 'betingede sætninger'.
Når Amstrads BASIC-fortolker, det store kompakte maskinkodeprogram, møder en betingelse efter et IF som f.eks. 'A > B', 'A=B' eller 'A < B', så skal BASIC finde ud af om betingelsen er sand eller falsk, før den kan finde ud af, om den skal udføre den instruktion, som er knyttet til IF sætningen. Således vil sætningen

IF A > B THEN GOSUB 1000

føre til at der springes til subrutinen i linie 1000, hvis 'A > B' er *sand* (f.eks. A=10 og B=9), og programmet vil blot køre videre hvis 'A > B' er *falsk* (f.eks. A=9 og B=10). Når maskinen afgør om betingelsen er opfyldt for de aktuelle værdier af A og B (eller hvad variablerne nu måtte hedde), tildeler den betingelsen en værdi '-1' hvis den er sand og '0' hvis den er falsk. Det er *værdien* fremfor selve betingelsen, der er afgørende, – det kan du selv afprøve ved denne lille test. Indtast:

A=5 <ENTER>
IF A THEN PRINT »SAND« <ENTER>

Nu bliver 'SAND' udskrevet på skærmen, fordi værdien, der følger efter IF ikke er 0 (alle værdier, positive og negative, forskellige fra 0 vil give samme resultat). Prøv nu:

A=0 <ENTER>
IF A THEN PRINT »SAND« <ENTER>

Denne gang kommer der ingen udskrift på skærmen. Nu er det ikke så meget IF vi er interesseret i her, men i stedet måden betingelserne bedømmes på, – prøv derfor følgende:

A=1 <ENTER>
B=1 <ENTER>
PRINT A=B <ENTER>

Nu svarer maskinen '-1', som er værdien af en sand betingelse.
Prøv nu:

A=1 <ENTER>
B=2 <ENTER>
PRINT A=B <ENTER>

Denne gang er resultatet '0' idet betingelsen er falsk. Det ser måske ikke ud til at have betydning for os her, men det har det. – Denne evne til at tildele værdier betingelser er meget anvendelig i programmering – og linie 1130 er et eksempel på det.

Linie 1130 tegner nemlig de små afmærkninger på Y-aksen i henhold til brugerens oplysninger. Længden af disse små streger er normalt otte pixels, men når værdien af løkke variablen I er delelig med 5 (dvs. hver femte afmærkning), så er betingelsen (I/5=INT(I/5) sand og værdien bli-

ver -1 og ikke 0. Med andre ord så betyder brugen af udtrykket '8*(I/5=INT(I/5))', i linien som bestemmer afmærkningernes længde, at hver femte streg får den dobbelte længde, – uden at vi behøver at bruge et større konstruktion med IF...THEN...ELSE.

Læg mærke til at for at *lægge* otte *til* længden af stregen, skal vi *fratrække* otte gange værdien af betingelsen, idet værdien af et sandt udtryk er *minus* en – for at fratrække noget negativt er det samme som at lægge noget positivt til.

Linie 1150-1180: Her sker det samme med den anden akse.

Linie 1190: Mængden pr. afmærkning på Y-aksen indlæses fra datalinien.

Test

For at teste det der er indlæst indtil nu skal programmet blot køres. Koordinatsystemet skal komme frem på skærmen med 'KLOKKEFORMET KURVE' øverst, 'Y AKSE' ude langs venstre kant og 'X AKSE' nederst. De to akser skal være inddelt i hver 20 enheder.

Modul 2.1.4: Grafen tegnes

Når koordinatsystemet er tegnet mangler vi bare at tegne selve kurven på grundlag af tallene i Data 2-modulet.

Modul 2.1.4: Linie 2000-2160

```
2000 REM ******************************
2010 REM Tegn grafen
2020 REM ******************************
2030 ON ERROR GOTO 2140
2040 RESTORE 4000
2050 READ t
2060 PLOT 32,32+t/enhed*py,1
2070 kolonne=1
2080 READ t$
2090 IF t$="END" THEN GOTO 2130
2100 DRAW 32+px*kolonne,32+VAL(t$)/enhed
*py
2110 kolonne=kolonne+1
2120 GOTO 2080
```

```
213Ø IF INKEY$="" THEN GOTO 213Ø
214Ø CLS
215Ø LIST 3ØØØ-
216Ø END
```

Kommentar

Linie 2030 og 2130-2160: Hvis der kommer en fejl mens kurven tegnes, eller hvis en af tasterne trykkes ned når tegningen er færdig, vil skærmen blive slettet og datamodulerne bliver listet. Dette giver en stor lettelse i arbejdet med at ændre og rette i de data som bruges til grafen.

Linie 2050: Den første dataenhed bliver indlæst.

Linie 2060: Grafen starter på y-aksen på det punkt, der svarer til den første datapost – under hensyntagen til mængden pr. enhed, som defineres af brugeren.

Linie 2070: KOLONNE bruges til at tælle hvor mange data der er indlæst og – dermed – hvor langt henad X-aksen grafen er nået.

Linie 2080-2090: De efterfølgende data indlæses i strenge. Dette muliggør at det kan checkes om det ordet 'END' nåes. Sker det, slutter programmet.

Linie 2100-2120: Da den grafiske markør allerede befinder sig dér, hvor kurven er nået til, er alt hvad der er nødvendigt, for at tegne ('DRAW') det næste stykke af kurven at opskrive KOLONNE for hver gang der indlæses data. X-koordinaterne udregnes ved at gange KOLONNE med længden (i pixels) af enhederne på X-aksen (PX) – konstanten 32 er afstanden fra skærmens venstre side til koordinatsystemet. Y-koordinaterne bliver først divideret med ENHED.

Hvis en datapost havde været 1.000.000 og brugeren havde bestemt at Y-aksen skulle inddeles i enheder af 100.000 (ENHED=100.000), så ville resultatet blive 1.000.000/1000.000 eller 10 enheder. Når antallet af enheder er udregnet ganges dette antal med længden af en Y-enhed i pixels (PY).

Test

Kør nu det færdige program og se den nydelige klokkeformede kurve blive tegnet. Når kurven er færdig, så tryk på en tilfældig tast og datamodulerne skal blive listet, således at der kan rettes i dem. Hvis denne afprøvning er tilfredsstillende, er programmet klar til brug.

Program 2.2: Cirkeldiagram

Skal man anskueliggøre fordelingen af en bestemt mængdes fordeling på mindre grupper, er 'cirkeldiagrammet' en meget enkel måde at præsentere denne fordeling på. I det følgende program skal vi bygge videre på det, vi lærte i de forrige programmer om at tegne cirkler og om brugen af datalinier. Hvis du er usikker på disse to elementer, så læs afsnittene en gang til, – vi kommer ikke nærmere ind på dem her.

Der er ikke grund til at slette de linier, som indlæses når vi tester modulerne, for linierne kommer til at indgå i det sidste modul, kontrolmodulet.

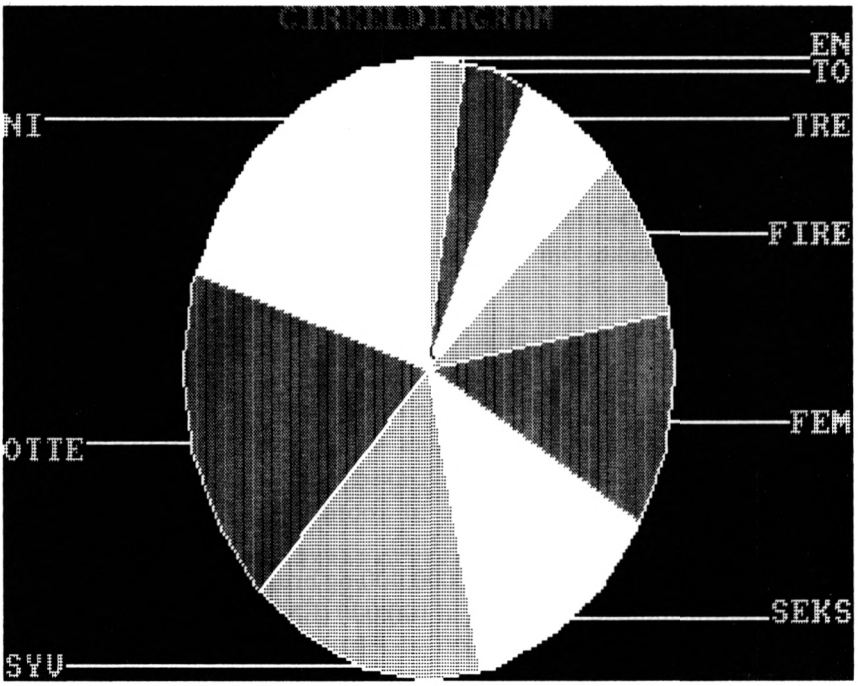


Fig. 2.2: Skærmdump fra programmet.

Modul 2.2.1: Data til diagrammet

Som i det forrige program forklarer data-linierne sig selv. Men læg mærke til at de to arrays i dette program, som indeholder navnene og størrelserne på de enkelte udsnit, ikke bliver dimensioneret, så derfor kan der

ikke bruges mere end 10 udsnit. Men det betyder ikke så meget i praksis, for et cirkeldiagram med flere udsnit er for uoverskueligt. Men hvis du alligevel gerne vil have flere udsnit, så skal du blot dimensionere de to arrays alt efter hvor mange udsnit du vil have.

Modul 2.2.1: Linie 4000-4080

```
4ØØØ REM ******************************
4Ø1Ø REM Data til diagrammet
4Ø2Ø REM ******************************
4Ø3Ø DATA NAVN,CIRKELDIAGRAM
4Ø4Ø DATA ANTAL UDSNIT,9
4Ø5Ø DATA NAVNE,EN,TO,TRE,FIRE,FEM,SEKS,
SYV,OTTE,NI,TI
4Ø6Ø DATA
4Ø7Ø DATA STØRRELSER,1,2,3,4,5,6,7,8,9,1
Ø
4Ø8Ø DATA
```

Modul 2.2.2: Databehandling til diagrammet

Informationerne i datamodulet læses ind i variablerne NAVN$ og UDSNIT og i de to arrays NAVNE$ og A.

Modul 2.2.2: Linie 5000-5110

```
5ØØØ REM ******************************
5Ø1Ø REM Databehandlingen
5Ø2Ø REM ******************************
5Ø3Ø RESTORE 4ØØØ
5Ø4Ø READ t$,navn$
5Ø5Ø READ t$,udsnit
5Ø6Ø READ t$:FOR i=Ø TO udsnit-1:READ na
vne$(i):NEXT i
5Ø7Ø RESTORE 4Ø7Ø
5Ø8Ø sum=Ø:READ t$:FOR i=Ø TO udsnit-1:R
EAD t:sum=sum+t:NEXT i
5Ø9Ø RESTORE 4Ø7Ø
51ØØ READ t$:FOR i=Ø TO udsnit-2:READ t:
a(i+1)=(t/sum)*36Ø+a(i):NEXT i
511Ø RETURN
```

Kommentar

Linie 5080-5100: Summen af de enkelte udsnit bliver først udregnet, for at finde hvor stor en mængde hele cirklen repræsenterer. DATA-pointeren bliver sat til linie 4000 med RESTORE, og hvert af tallene bliver 'oversat' til et andet tal, som – når det divideres op i 360 – giver samme resultat, som når det oprindelige tal divideres op i den samlede sum. Hvis summen f.eks. er 100 og størrelsen af et af udsnittene er 25, så ville dette blive 'oversat' til 90 – eller 25% af 360. Disse nye tal vil senere blive brugt til at bestemme hvor store dele af cirklen, hver af udsnittene skal fylde.

Test

Indlæs følgende linier, som senere vil indgå i kontrolmodulet, og kør programmet

```
1040 GOSUB 5000
1100 END
```

Hvis modulerne er rigtigt indlæst, skal der ikke ske noget på skærmen – kun hvis der er en fejl, skal der komme en udskrift. Hvis du vil, kan du få udskrevet indholdet af variabler og arrays, så du kan være helt sikker på at modulerne fungerer.

Modul 2.2.3: Skærmen

I dette modul sættes grafisk mode og farverne fastlægges. Der tegnes endvidere en cirkel på 80*80 midt på skærmen sammen med diagrammets navn.

Modul 2.2.3: Linie 2000-2090

```
2ØØØ REM ******************************
2Ø1Ø REM Skærmen
2Ø2Ø REM ******************************
2Ø3Ø MODE 1:ORIGIN 32Ø,184:BORDER Ø
2Ø4Ø INK Ø,Ø:INK 1,2:INK 2,6:INK 3,18
2Ø5Ø LOCATE 2Ø-LEN(navn$)/2,1:PEN 2:PRIN
T navn$
2Ø6Ø DEG
2Ø7Ø PLOT Ø,184,3
```

```
2Ø8Ø FOR a=Ø TO 36Ø STEP 15:DRAW 184*SIN
(a),184*COS(a):NEXT a
2Ø9Ø RETURN
```

Kommentar

Linie 2080: Hvis du har læst forklaringen på hvordan en cirkel tegnes i det første kapitel, kan du se at denne linie simpelthen tegner en cirkel. Fremfor at plotte punkterne et for et på cirkelperiferien plottes et punkt for hver 15 grader og der tegnes linier imellem dem.

Test

Indtast disse to linier og kør programmet.

```
1060 GOSUB 2000
1080 IF INKEY$="" THEN GOTO 1080
```

Nu skal cirkeldiagrammets navn og en gul cirkel komme frem på skærmen. Tryk nu på en tilfældig tast (ikke <ESC>), og skærmen skal blive 'normal' igen.

Modul 2.2.4: Detaljerne

Nu kommer vi til det modul, som opdeler cirklen i udsnit og sætter navne på, sådan som det blev defineret i data-modulet. For at forstå hvad der sker, må du tænke på hvordan en cirkel tegnes, sådan som det blev forklaret i ANALOG-UR programmet. Har du glemt det, så se programmet igennem igen og læs kommentaren.

Modul 2.2.4: Linie 3000-3170

```
3ØØØ REM ******************************
3Ø1Ø REM Detaljerne
3Ø2Ø REM ******************************
3Ø3Ø FOR i=Ø TO udsnit-1
3Ø4Ø MOVE Ø,Ø:DRAW 184*SIN(a(i)),184*COS
(a(i))
3Ø5Ø NEXT i
3Ø55 PEN 1
3Ø6Ø FOR i=Ø TO udsnit-1
```

```
3070 ta=((a(i)+a(i+1+udsnit*(i+1=udsnit)
)))/2
3080 IF a(i+1+udsnit*(i+1=udsnit))<a(i)
THEN ta=ta+180
3090 MOVE 180*SIN(ta),180*COS(ta):c=i MO
D 3+1:IF i=udsnit-1 AND c=1 THEN c=2
3100 GOSUB 6000
3110 tx=184*SIN(ta)
3120 ty=184*COS(ta)
3130 dx=320*SGN(tx)
3140 MOVE tx,ty:DRAW dx,ty,3
3150 LOCATE 1+(LEN(navne$(i))-40)*(dx=32
0),14-INT((ty+9)/16):PRINT navne$(i);
3160 NEXT i
3170 RETURN
```

Kommentar

Linie 3030-3050: Der tegnes en antal linier fra cirklens centrum og ud til cirkelperiferien, således at cirklen opdeles i de rigtige udsnit. De tal der anvendes blev udregnet i modul 2.2.2.

Linie 3070-3100: Her bliver der beregnet endnu flere vinkler, men hver af dem ligger midt i de udsnit der lige er tegnet. Det logiske udtryk i linie 3070 og 3080 'UDSNIT*(I+1=UDSNIT)', sikrer – når det sidste udsnit er nået – at begyndelsen af det første udsnit bruges som den anden vinkel. Den sidste beregning vil blive forkert: i stedet for at lægge en vinkel til en større vinkel og dividere summen med 2, bliver vinklen for det sidste udsnit lagt til 0 (starten på første udsnit), derfor får vi en vinkel der ligger mellem starten på det første og det sidste udsnit. Vi retter fejlen ved at lægge 180 grader til resultatet.

Linie 3090 beregner placeringen af et punkt tæt på periferien inde i de enkelte udsnit, på grundlag af de beregnede vinkler. Næste modul kaldes for at fylde farver i udsnittene (ligesom i ANALOG-UR). Linie 3090 sørger også for at der skiftes mellem tre farver når udsnittene farvelægges, dog kan sidste udsnit ikke få samme farve som det første. Grunden til dette er selvfølgelig at de to udsnit grænser op til hinanden, og diagrammet derfor ville blive vanskeligere at tyde.

Linie Linie 3110-3120: Disse to linier udregner et punkt på periferien i

en vinkel halvvejs mellem begyndelsen og slutningen på det aktuelle udsnit.

Linie 3130: Da vi gjorde skærmen klar til grafikken, flyttede vi startpunktet for den grafiske markør ind midt på skærmen med ORIGIN. Denne linie beregner en placering på skærmen 320 pixels fra centrum. Variablen TX indeholder allerede X-koordinaten for et punkt på cirkelperiferien, som enten er negativt (til venstre for centrum) eller positivt (til højre for centrum). Ved at gange 320 med SGN(TX) sikrer vi, at hvis udsnittes midtpunkt ligger til venstre på skærmen, så kommer DX også til venstre – og omvendt.

Linie 3140-3150: En linie tegnes fra periferien til skærmens kant enten til venstre eller højre, bestemt af DX. For enden af linien, eller rettere hen over den, skrives navnet på det udsnit som linien fører hen til. Positionen for tekstmarkøren i højre side flyttes til venstre, så der præcis er plads på skærmen. Her bruges der et logisk udtryk til at bestemme X-koordinaten. Linien ser mere kompliceret ud end den egentlig er, fordi vi er nødt til at omsætte positionen i pixels til en kolonne (karakter-position).

Test

Tilføj disse to programlinier og kør programmet:

```
1070 GOSUB 3000
6000 RETURN
```

Nu skulle der komme en figur, som svarer til fig. 2.2, de enkelte udsnit har ikke fået farver endnu.

Modul 2.2.5: Der fyldes ud med farver

Dette modul er næsten magen til modulet i ANALOG-UR. Den eneste forskel er at udfyldningen fortsætter indtil der stødes en hvilken som helst farve, som er forskellig fra baggrundsfarven.

Modul 2.2.5: Linie 6000-6300

```
6ØØØ REM ******************************
6Ø1Ø REM Fyldning med farver
6Ø2Ø REM ******************************
6Ø3Ø IF TESTR(Ø,Ø)<>Ø THEN RETURN
```

```
6Ø4Ø s=2
6Ø5Ø MOVE s*INT(XPOS/s),2*INT(YPOS/2)
6Ø6Ø :
6Ø7Ø :
6Ø8Ø REM check op til højre **********
6Ø9Ø IF TESTR(Ø,s)=Ø THEN GOTO 6Ø9Ø
61ØØ IF TESTR(s,-s)=Ø THEN GOTO 6Ø9Ø
611Ø :
612Ø :
613Ø REM check op til venstre *********
614Ø MOVER -s,Ø
615Ø IF TESTR(Ø,s)=Ø THEN GOTO 6Ø9Ø
616Ø IF TESTR(-s,-s)=Ø THEN GOTO 615Ø
617Ø :
618Ø :
619Ø REM farve i linier vandret *******
62ØØ x1=XPOS
621Ø MOVER s,Ø
622Ø PLOTR Ø,Ø,c
623Ø IF TESTR(s,Ø)=Ø THEN GOTO 622Ø
624Ø x2=XPOS-s
625Ø MOVE x1,YPOS-s
626Ø IF TESTR(s,Ø)=Ø THEN GOTO 629Ø
627Ø IF XPOS=x2 THEN RETURN
628Ø GOTO 626Ø
629Ø IF TESTR(-s,Ø)<>Ø THEN GOTO 62ØØ
63ØØ GOTO 629Ø
```

Test

Kør programmet igen – denne gang skal der være individuelle farver i udsnittene.

Modul 2.2.6: Kontrol-modulet

De fleste af linierne her er allerede indlæst, – husk at få resten med ellers kommer der til at mangle et par finesser.

Modul 2.2.6: Linie 1000-1120

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
1Ø3Ø ON ERROR GOTO 111Ø
1Ø4Ø GOSUB 5ØØØ
1Ø5Ø ON BREAK GOSUB 1Ø9Ø
1Ø6Ø GOSUB 2ØØØ
1Ø7Ø GOSUB 3ØØØ
1Ø8Ø IF INKEY$="" THEN GOTO 1Ø8Ø
1Ø9Ø CLEAR:CLS:LIST 4ØØØ-4999
11ØØ END
111Ø PRINT"*** SANDSYNLIGVIS FORKERT TYP
E DATA ***"
112Ø FOR i=1 TO 3ØØØ:NEXT i:GOTO 1Ø9Ø
```

Kommentar

Linie 1030 og 1110-1120: En lille fejlmelding angiver at der sandsynligvis er fejl i data-modulet. Dette kan dog ikke gøre det ud for en test, idet der kan være fejl som ON ERROR ikke registrerer.

Test

Prøv at ændre et af tallene under 'STØRRELSER' til et bogstav og kør programmet. Nu skal fejlmeldingen komme og data-modulet skal blive listet. Ret fejlen igen og kør programmet endnu en gang. Prøv så at taste <ESC> to gange (eller en hvilken som helst tast, når programmet er kørt) og datamodulet skal igen blive listet.

Program 2.3: 3D Graf

Vi har nu st på to måder at anvende højopløsningsgrafikken på, når forskellige data skal præsenteres. Men Amstrad har jo også en meget alsidig lavopløsningsgrafik. Der findes en lang række grafiske karakterer, som ikke er umiddelbart tilgængelige via tastaruret. I manualens Appendix 3 er der en oversigt over de karakterer, som kan bruges til grafik, der vil være næsten umulige at lave med højopløsningsgrafik.

I dette program skal vi lave et tredimensionalt søjlediagram, som meget

overbevisende demonstrerer hvor god Amstrads lavopløsningsgrafik er. Skærmbilledet er faktisk så flot at det kan overbevise venner og familie om, at du virkelig er skrap til at programmere.

I 3D-programmet præsenteres:

1) Brug af dataoptageren til at lagre data til programmet.
2) Brug af lavopløsningsgrafik-karakterer.
3) Interaktiv indlæsning af data.

Modul 2.3.1: Initialisering

Et ganske enkelt modul, som dimensionerer et lille array og spørger, om der skal indlæses data fra bånd – det bliver behandlet senere.

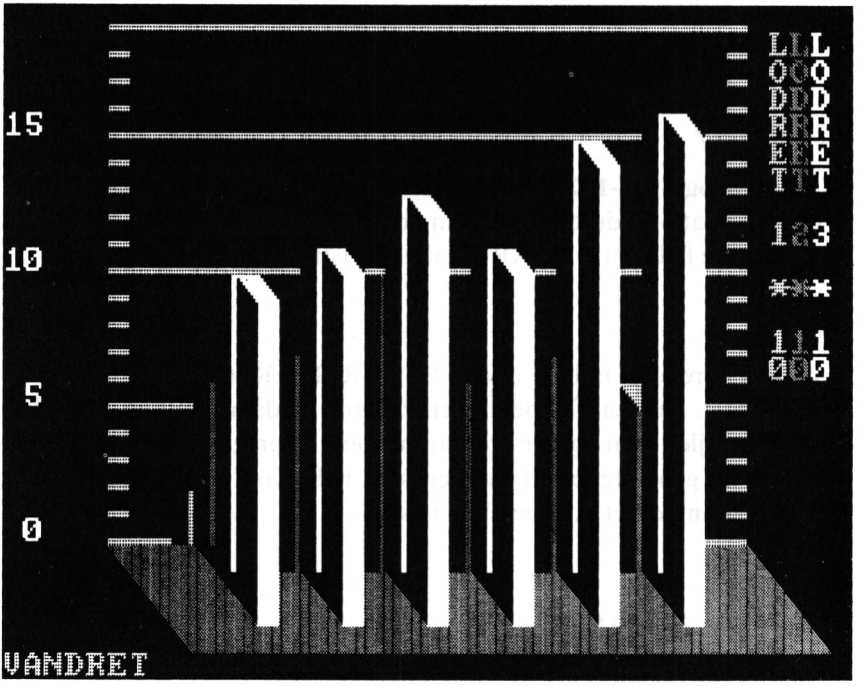


Fig. 2.3: Skærmdump fra 3D Graf.

Modul 2.3.1: Linie 2000-2080

```
2ØØØ REM ******************************
2Ø1Ø REM Initialisering
2Ø2Ø REM ******************************
```

```
2Ø3Ø CLS:MODE 1
2Ø4Ø PRINT TAB(17);"3D GRAF";TAB(17);"==
=====":PRINT
2Ø5Ø DIM hh(2,6)
2Ø6Ø INPUT "Hent data på bånd (j/n)";q$
2Ø7Ø r$=CHR$(13)
2Ø8Ø RETURN
```

Kommentar

Linie 2050: Arrayet HH skal bruges til at lagre data fra grafen. Da pladserne i DIM-linien starter med nul er der tre sæt med syv dataposter hver.

Linie 2060: Dette INPUT bruges bruges i forbindelse med et af de følgende moduler til at hente data til grafen fra dataoptageren.

Linie 2070: Strengen R$ skal bruges i data-lagrings modulet og vil blive forklaret der.

Modul 2.3.2: Dataindlæsning

I dette sidste program i kapitel 2, vil vi ikke bruge knebet med at lade data-linierne lagre vekslende data. De fleste programmer, der bearbejder data, giver brugeren mulighed for at indlæse data mens programmet kører – sådanne programmer kaldes 'interaktive'. I dette tilfælde indlæses alle data samlet, og i modulet her bedes der om indlæsninger, og på grundlag af de første indlæsninger bedes der om nye – og det checkes om de er 'gyldige'.

Modul 2.3.2: Linie 3000-3250

```
3ØØØ REM ******************************
3Ø1Ø REM Indlæsning af data
3Ø2Ø REM ******************************
3Ø3Ø CLS
3Ø4Ø PRINT TAB(17);"3D GRAF";TAB(17);"==
=====":PRINT
3Ø5Ø PRINT "Der er 19 enheder lodret.":P
RINT
3Ø6Ø INPUT "Størrelsen af hver Y-enhed:"
,ev:PRINT
```

```
3Ø7Ø INPUT "Søjler pr. række (1-6):",as:
PRINT
3Ø8Ø INPUT "Rækker (1-3):",ar:PRINT
3Ø9Ø PRINT "**************************
**********":PRINT
31ØØ INPUT "Navn for X-aksen:",nx$:PRINT
311Ø FOR i=Ø TO ar-1
312Ø PRINT "Navn for Y-aksen";i+1;:INPUT
 ny$(i):PRINT
313Ø NEXT i
314Ø CLS
315Ø FOR i=Ø TO ar-1
316Ø FOR j=1 TO as
317Ø t=2Ø*ev
318Ø WHILE t/ev>19
319Ø PRINT:PRINT "Indtast Række";i+1;"væ
rdi";j;":";
32ØØ INPUT t
321Ø IF INT(t/ev)>19 THEN PRINT:PRINT "*
***** Værdien er for høj ******"
322Ø WEND
323Ø hh(i,j)=t
324Ø NEXT j,i
325Ø RETURN
```

Kommentar

Linie 3050-3060: Hver enhed på Y-aksen repræsenterer en værdi, bestemt af brugeren, ligesom i det første graf-program. Læg mærke til, at vi ikke har de samme muligheder for at bestemme størrelsen af Y-enhederne, som i de tidligere programmer, – det skyldes at lavopløsningen er mindre flexibel. Den eneste praktiske størrelse for en enhed er én skærmlinie – og på denne graf er der plads til 19 enheder på Y-akse.

Linie 3070-3080: Som tidligere nævnt giver grafen plads til tre gange seks dataposter. De vil blive vist som tre *rækker* med op til seks *søjler* i hver.

Linie 3110-3130: Da der kan være op til tre rækker, skal der være plads til tre navne på rækkerne, f.eks. indkøbspris, salgspris og fortjeneste. Læg mærke til at navnene sættes ind i et array (NY$), som ikke behøver at blive dimensioneret, fordi det aldrig kommer til at indeholde mere end tre

elementer. Så snart et array nævnes bliver det straks dimensioneret med 10 elementer (0-9).

Linie 3150-3240: Brugeren bedes om at indlæse data til hver række søjler – alt efter hvor mange søjler brugeren ønsker at have. Indlæsningerne undersøges for at sikre at søjlerne holder sig under de 19 enheder, der er plads til – det sørger løkken i linie 3180-3220 for. Hvis et tal er for stort får brugeren det at vide og der bedes om en ny indlæsning.

Test

Indtast de følgende linier (som senere vil indgå i kontrolmodulet):

```
1030 GOSUB 2000
1040 GOSUB 3000
1110 CLS:END
```

Kør nu den del af programmet, der er indtastet indtil nu, og svar på de spørgsmål, der stilles, på denne måde:

```
Størrelsen af hver Y-enhed: 1
Søjler (1-6): 1
Rækker (1-3): 1
Navn for X-aksen: VANDRET
Navn for Y-aksen 1: LODRET 1
Række 1, værdi 1: 10
```

Efter den sidste indtastning bliver skærmen slettet, og programmet stopper med meldingen 'READY'. Indtast nu:

```
? EV,AS,AR,NX$, NY$(0), HH(0,1)
```

Og svaret skal blive:

```
1             1           1
VANDRET   LODRET 1    10
```

Hvis du vil, kan du prøve at køre programmet en gang til for at indtaste tal der er for store – dvs. større end 19 gange størrelsen af hver Y-enhed. Disse indtastninger skal blive forkastet af programmet.

Modul 2.3.3: Koordinatsystemet tegnes

Ligesom ved den første graf i dette kapitel skal vi bruge et koordinatsystem, således at selve søjlediagrammet kan forstås. Koordinatsystem la-

ves af dette modul, som anvender løkker, variabler og LOCATE kommandoen til at få placeret de grafiske karakterer rigtigt på skærmen.

Modul 2.3.3: Linie 4000-4260

```
4ØØØ REM ****************************
4Ø1Ø REM Koordinatsystemet tegnes
4Ø2Ø REM ****************************
4Ø3Ø CLS
4Ø4Ø FOR i=Ø TO 3
4Ø5Ø LOCATE 6+i,21+i
4Ø6Ø PEN 2:PRINT CHR$(213);STRING$(29,CH
R$(143));CHR$(215)
4Ø7Ø NEXT i:PEN 1
4Ø8Ø LOCATE 6,1:PRINT STRING$(3Ø,CHR$(21
Ø))
4Ø9Ø FOR i=2 TO 2Ø
41ØØ LOCATE 6,i:PRINT CHR$(21Ø)
411Ø LOCATE 35,i:PRINT CHR$(21Ø)
412Ø NEXT i
413Ø FOR i=5 TO 2Ø STEP 5
414Ø LOCATE 6,i:PRINT STRING$(3Ø,CHR$(21
Ø))
415Ø NFXT i
416Ø LOCATE 1,25:PRINT nx$
417Ø FOR h=Ø TO ar-1
418Ø PEN h+1
419Ø tt$=ny$(h)+" *"+STR$(ev)
42ØØ FOR i=1 TO LEN(tt$)
421Ø LOCATE 37+h,i+1:PRINT MID$(tt$,i,1)
422Ø NEXT i,h
423Ø FOR i=Ø TO 15 STEP 5
424Ø LOCATE 1,2Ø-i:PRINT USING "##";i
425Ø NEXT i
426Ø RETURN
```

Kommentar

Linie 4040-4070: Denne løkke laver 'grundfladen' for søjlediagrammet. De grafiske karakterer kan ses i manualens Appendix 3.

Linie 4080: En linie hen over skærmen, øverst oppe, laves ved hjælp af karakter nummer 210 (vandret streg), der gentages 30 gange.

Linie 4090-4120: Afmærkningen på Y-aksen med de 19 enheder.

Linie 4130-4150: Fire linier hen over skærmen for hver fem enheder på Y-aksen.

Linie 4160: Navnet på X-aksen.

Linie 4170-4220: Disse løkker udskriver navnene (eller navnet) for Y-aksen ude til højre, sammen med størrelsen af hver enhed. Løkke-variablen I bruges både til navnene og til at ændre farven for hvert navn. Farven for hvert navn kommer til at svare til farven på én af rækkerne.

Linie 4230-4250: Denne løkke sætter tallene 0, 5, 10 og 15 på Y-aksen.

Test

Indtast endnu en linie:

1050 GOSUB 4000

og kør programmet. Angiv enhedsstørrelsen, én søjle og tre rækker. Giv de tre rækker navne. Når du bliver bedt om at opgive søjleværdierne, skal du blot taste <ENTER>. Du skulle så se koordinatsystemet på skærmen, med navne på akserne som på dig. 2.3.

Modul 2.3.4: Diagrammet tegnes

Dette modul er faktisk lettere at forstå, når det først er indlæst, så du kan se hvordan det ser ud. Modulet er ikke baseret på generelle principper i særlig høj grad. I stedet har jeg prøvet mig frem indtil jeg fandt den rigtige kombination af grafiske karakterer.

Modul 2.3.4: Linie 5000-5220

```
5ØØØ REM ******************************
5Ø1Ø REM Diagrammet tegnes
5Ø2Ø REM ******************************
5Ø3Ø FOR h=Ø TO ar-1
5Ø4Ø PEN h+1
5Ø5Ø FOR i=as TO 1 STEP -1
```

```
5Ø6Ø betingelse=1:WHILE INT(hh(h,i)/ev)<
>Ø AND betingelse
5Ø7Ø FOR j=1 TO INT(hh(h,i)/ev)+1
5Ø8Ø LOCATE 9+4*(i-1)+h,22+h-j
5Ø9Ø IF j=1 THEN PEN 2:PAPER Ø:PRINT CHR
$(143);CHR$(215);:PEN h+1:PRINT CHR$(143
)
51ØØ IF j>1 AND j<INT(hh(h,i)/ev)+1 THEN
 PRINT CHR$(2Ø9);" ";CHR$(143)
511Ø IF j=INT(hh(h,i)/ev)+1 THEN PRINT C
HR$(2Ø9);CHR$(213);CHR$(215)
512Ø NEXT j
513Ø betingelse=Ø:WEND
514Ø NEXT i
515Ø NEXT h
516Ø FOR i=Ø TO as-1
517Ø FOR j=1 TO ar
518Ø LOCATE 9+4*i+j,2Ø+j
519Ø IF j>1 OR hh(2,i)=Ø OR (j=1 AND as=
Ø) THEN PEN 2:PRINT CHR$(215)
52ØØ NEXT j
521Ø NEXT i
522Ø RETURN
```

Kommentar

Linie 5030-5150: Den første løkke laver det ønskede antal rækker.

Linie 5050-5140: Den anden løkke laver det ønskede antal søjler – fra venstre mod højre.

Linie 5060-5130: Disse linier laver en *pseudo-* (eller *dummy-*) WHILE...WEND løkke, som laver en søjle hvis søjlens værdi er 0 – det er en pseudo-løkke, fordi den aldrig bliver kørt mere end én gang under nogen omstændigheder. Dette sikres ved at bestemme at værdien af variablen BETINGELSE er én af betingelserne for at løkken bliver gentaget. Da BETINGELSE sættes lig 0 før WEND nåes i linie 5130, kommer løkken aldrig til at køre mere end én gang. Det er en god teknik at huske på, for Amstrad mangler EXIT kommandoen, som ville bevirke at løkken automatisk blev stoppet.

Linie 5070-5120: Denne løkke laver en søjle. Linie 5080 angiver placeringen af hver karakter i søjlen. Startpositionen vil være søjlen yderst til højre (bestemt af I), så flyttes positionen til den enkelte søjle (bestemt af J), og derefter flyttes fire pladser til venstre for hver ny søjle. Når en række er færdig laves den næste række (bestemt af H) en plads ned til højre for den forrige.

Løkkevariablen J der som nævnt bevæger sig fra 1 til højden af søjlen. Første gang løkken køres igennem bliver søjlens grundflade lavet (linie 5090). Når løkken køres igennem de følgende gange bliver forskellige karakterer brugt til at lave henholdsvis siden og forsiden af søjlen, – LOCATE bruges til at placere karaktererne. Endelig sættes søjlens overside på (linie 5110).

Linie 5160-5220: Når søjlerne er tegnet, mangler der nogle små detaljer for neden, som klares af disse linier.

Test

Indtast en ny linie:

1060 GOSUB 5000

og kør programmet. Brug en enhedsstørrelse på 1, og brug tre søjler i hver række og tre rækker. Aksernes navne har ingen betydning, så dem kan du selv bestemme. Når tallene for søjlerne skal angives indtastes følgende tal:

6, 12, 18, 6, 12, 18, 6, 12, 18

Nu skulle tre rækker – med tre søjler i hver – komme frem på skærmen. Oversiden på de tre søjler skal se helt jævn ud. Læg mærke til at når værdierne for søjlerne skal aflæses, så må du lade som om at toppen af den forreste søjle fortsætter helt ind til den bageste søjle, således at søjlens højde aflæses ved søjlens bagside – som om den stod bagest. I dette tilfælde er søjlerne tre-og-tre lige høje – selvom de forreste faktisk er mindst. Det er nødvendigt for at give et tredimensionalt perspektiv.

Prøv programmet flere gange for at se hvordan søjlediagrammet kommer til at se ud med forskellige værdier. Du vil opdage at diagrammet kun rigtigt fungerer, når én række aldrig er højere end den der står bagved. Der er mange typer information, der kan indpasses i søjlediagrammet – f.eks. salgspris/købspris/overskud, som blev nævnt tidligere.

Modul 2.3.5 og 2.3.6: Lagring af data på bånd

Nu kommer vi for første gang til noget, som mange glemmer i deres egne programmer – gemning af data på bånd. Du har uden tvivl bemærket at hvis du har lavet et par fejl i indlæsningen af et program, så bliver det hurtigt ret trættende at skulle indlæse de samme data igen og igen. Dertil kommer, at der for mange programmers vedkommende ikke er meget ide i at indlæse data i computeren, hvis du alligevel skal huske dem, hver gang du slukker maskinen. – Måske er det ikke det store problem at skulle indtaste data flere gange i dette program, men hvad hvis det er et program med flere hundrede dataposter?

Alle disse problemer kan løses ved at bruge dataoptageren til at gemme data, som til en hver tid kan hentes tilbage i maskinen. Disse to moduler gemmer og henter data.

Modul 2.3.5 og 2.3.6: Linie 6000-7110

```
6ØØØ REM ******************************
6Ø1Ø REM Gem data på bånd
6Ø2Ø REM ******************************
6Ø3Ø OPENOUT "grafdata"
6Ø4Ø PRINT #9,ar;r$;as;r$;nx$;r$;ev
6Ø5Ø FOR i=Ø TO ar-1
6Ø6Ø PRINT#9,ny$(i)
6Ø7Ø FOR j=Ø TO as
6Ø8Ø PRINT#9,hh(i,j)
6Ø9Ø NEXT j,i
61ØØ PRINT#9:CLOSEOUT
611Ø RETURN
7ØØØ REM ******************************
7Ø1Ø REM Hent data fra bånd
7Ø2Ø REM ******************************
7Ø3Ø OPENIN "grafdata"
7Ø4Ø INPUT #9,ar,as,nx$,ev
7Ø5Ø FOR i=Ø TO ar-1
7Ø6Ø INPUT #9,ny$(i)
7Ø7Ø FOR j=Ø TO as
7Ø8Ø INPUT #9,hh(i,j)
7Ø9Ø NEXT j,i
71ØØ CLOSEIN
711Ø RETURN
```

Kommentar

Linie 6030: Før vi kan gemme data på bånd, skal vi først 'åbne en fil', og det sker med kommandoen:

OPENOUT "FILNAVN"

Linie 6040-6090: Når filen er åbnet skal vi blot indlæse data i den. Det sker med ordren PRINT #1. De poster der udlæses i filen består af vores hovedarray (HH$), og hovedvariablerne.

Linie 6040: Læg mærke til at der står flere R$'er i samme linie som PRINT #. Du husker måske at R$ blev sat lig med CHR$(13) i det første modul i programmet, det er karakterkoden for <ENTER>, og det viser at en post er slut og skal udlæses. Når flere poster bliver udlæst til en fil med kun én PRINT # kommando, så skal R$ bruges mellem de enkelte poster, ellers vil de blive udlæst sammen.

Linie 6060: I kommentaren til den forrige linie blev det hævdet at R$ (eller en anden variable som er tilskrevet CHR$(13)), skulle være med for at adskille de enkelte poster – så hvorfor sker det så ikke her i de to løkker, som udskriver indholdet af de to arrays til båndfilen? Svaret er, at når en PRINT eller PRINT # kommando ikke afsluttes med komma eller semikolon, så sætter Amstrad automatisk et <ENTER> efter den sidste post. – Det er derfor at posterne bliver udskrevet på hver sin linie på skærmen, når den foregående post ikke afsluttes med et komma eller et semikolon.

Linie 6100: Når en fil bliver udlæst er det en god ide at afslutte med PRINT # 9, som sørger for at eventuelt resterende poster i maskinens hukommelse, som venter på at blive udlæst, slettes. Endelig skal filen lukkes med CLOSEOUT. Hvis det ikke gøres kan datastrøm 9 ikke bruges, og det kan endda føre til at data på båndet går tabt, når programmet prøver at hente dem.

Linie 7000-7110: Disse linier udfører lige det modsatte af de forrige, idet de henter data, der er lagret på båndet.

Linie 7030: Filen skal igen åbnes. Den eneste forskel er at der denne gang bruges kommandoen OPENIN, som fortæller systemet at vi skal hente data *fra* båndet.

Linie 7040-7090: Det modsatte af PRINT # er INPUT #. Bemærk at vi

ikke skal bruge R$ som adskillelse, når vi *henter* data. De ligger i den måde INPUT og INPUT # er opbygget på, at de ikke registrerer at de har modtaget data før der tastes <ENTER> eller før de støder på CHR$(13) på båndet.

Linie 7100: Som i linie 6100 skal filen lukkes når indlæsningen er slut.

Test

Det er det nemmeste at vente med at teste disse moduler indtil de sidste linier i kontrolmodulet er indlæst.

Modul 2.3.7: Kontrolmodulet

Mange af linierne i dette modul er allerede indtastet i forbindelse med de tests vi har lavet. Husk nu at få de sidste med.

Modul 2.3.7: Linie 1000-1110

```
1ØØØ REM ******************************
1Ø1Ø REM Kontrol
1Ø2Ø REM ******************************
1Ø3Ø GOSUB 2ØØØ
1Ø4Ø IF LOWER$(q$)="y" THEN GOSUB 7ØØØ E
LSE GOSUB 3ØØØ
1Ø5Ø GOSUB 4ØØØ
1Ø6Ø GOSUB 5ØØØ
1Ø7Ø WHILE INKEY$=""
1Ø8Ø WEND
1Ø9Ø LOCATE 1,25:INPUT "Gem data på bånd
 (j/n)";q$
11ØØ IF LOWER$(q$)="j" THEN GOSUB 6ØØØ
111Ø CLS:END
```

Test

Kør programmet. Det skal nu være muligt at indlæse data til en graf. Når grafen kommer på skærmen, når en tilfældig tast trykkes ned, skal der

komme en linie, der spørger om de anvendte data skal gemmes. Før du svarer 'J', skal du sikre dig at der er et bånd i dataoptageren, som der må indspilles på (pas på ikke at få slettet et program). Når gemningen er slut kan du køre programmet igen og svare 'J', når der spørges om du ønsker at hente data fra et bånd. Når dine data igen er indlæst skal den samme graf komme på skærmen igen. Er testen OK, så er programmer klar til brug.

KAPITEL 3

Grafik og Lyd

I dette kapitel går vi mere i dybden med Amstrads grafik, og vi skal også arbejde med lyd. I de fleste tilfælde vil de programmer du selv laver have nogle små rutiner indbygget med både lyd og grafik, – det er ofte noget så enkelt som fremhævning af en overskrift eller en lille advarselslyd. Men i andre tilfælde er der brug for noget mere avanceret som f.eks. en fanfare til at live op i programmet. I sådanne tilfælde kan du skrive en rutine til at lave en bestemt grafisk effekt eller til at lave en lille melodi. Men det er noget mere praktisk at have nogle hjælpemidler ved hånden, som gør det nemt at lave f.eks. den grafik du skal bruge, og så hente rutinerne fra et bånd når du har brug for dem.

I dette kapitel finder du følgende tre programmer:

KARAKTERER: Som gør det muligt at omdefinere hele Amstrads karaktersæt.

DESIGNER: Et hjælpemiddel til at lave højopløsningsgrafik.

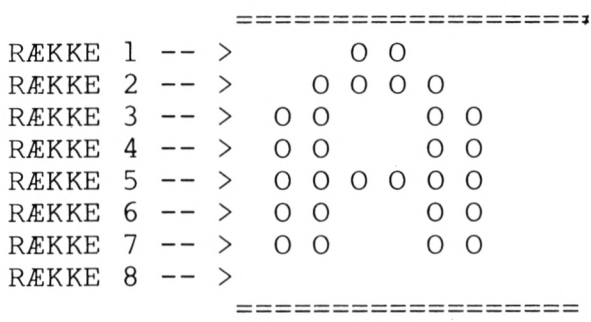
MUSIK: Som gør det muligt at lave og afspille to-stemmige melodier på en nem måde.

Program 3.1: Karakterer

I det følgende kapitel så vi hvordan vi kan bruge Amstrads grafiske karakterer i vores programmer. I dette program bliver vi lidt ved lavopløsningsgrafikken, idet vi her laver et *støtteprogram,* der gør det muligt at modificere eller helt ændre de karakterer Amstrad skriver ud på skærmen. Men før vi går igang med selve programmet, skal vi lige have lidt forklaring på, hvordan karaktererne laves og udskrives i lavopløsning.

Amstrads lavopløsningsskærm har plads til 25 linier med 40 karakterer (kolonner) i hver linie (i MODE 1) – i alt 1000 karakterpladser. Dvs. at der

kan skrives 1000 karakterer på skærmen, selvom nogle af dem måtte være ens, fordi maskinen ikke kan rumme 1000 *forskellige* tegn på samme tid. Men det er ikke slut her, for hvis du ser nærmere på karaktererne på skærmen, kan du se at de ikke består af sammenhængende linier, som de ord du læser nu, men af prikker. Faktisk så består hver eneste af de 1000 karakterer af 64 prikker, og kombinationer af disse 64 prikker, der giver hver eneste af de karakterer Amstrad kan lave. Bogstavet 'A' er f.eks. vist i fig. 3.1.



Figur 3.1: En forstørrelse af bogstavet 'A' som det ser ud på skærmen i MODE 2.

De prikker som karaktererne er opbygget af kaldes 'pixels', som er en forkortelse af 'picture elements' ('billed elementer') og det er de mindste elementer som Amstrads skærm har, uanset om det er høj- eller lavopløsning.

Sådan et indviklet mønster kommer jo ikke af sig selv, og der ligger selvfølgelig en kode et sted i maskinens hukommelse som bliver kaldt frem, når du trykker tast 'A'. Og sådan er det med alle de karakterer Amstrad råder over: de ligger i en bestemt del af hukommelsen (som kaldes 'karakterhukommelsen' eller 'karakter-ROM'), som tal-koder. Hver karakter fylder otte bytes i ROM, og hver af disse bytes bestemmer hvilke pixels, der skal bruges i hver række. Det sker ved at omsætte værdien af en byte til et 'billede' af en række. Det er det binære talsystem, der bruges her. I dette 'to-tals-system' udtrykkes tallene som potenser af 2, fremfor som ti-talspotenser, som i vores normale talsystem. Lad os tage et eksempel:

2013006

kan forstås således i vores normale talsystem:

(2*10^6)+(0*10^5)+(1*10^4)+(3*10^3)+(0*10^2)+(0*10^')+(10^0)

I det binære system bruges der kun to tal '1' og '0', og et tal som f.eks.

11001010

betyder

(1*2^7)+(1*2^6)+(0*2^5)+(0*2^4)+(1*2^3)+(0*2^2)+(1*2^1)+(0*2^0)

eller lidt kortere:

(2^7)+(2^6)+(2^3)+(2^1)

Regner du tallet ud vil du opdage at det svarer til tallet 202 i titalssystemet. Det er imidlertid ikke nødvendigt at forstå det binære talsystem fuldt ud. Husk blot på at en enkelt byte indeholder et ottecifret binært tal, og at alle disse nuller og ettaller er helt ideelle til registrering af, hvilke pixels der anvendes i en enkelt række. Bogstavet 'A' bliver f.eks. til følgende otte binære ottecifrede tal (bytes):

0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0

Hvis du ser godt efter kan du stadig se 'A'et, denne gang er det blot skrevet med '1' i stedet for med pixels. De enkelte rækker svarer i vores talsystem til:

24, 60, 102, 102, 126, 102, 102, 0

Når computeren skal skrive en karakter på skærmen, finder den altså karakteren i 'karakterhukommelsen' i ROM, og udskriver den.

Af dette følger, at hvis det var muligt at *ændre* indholdet i karakterhukommelsen, så vil karaktererne på skærmen også ændres. Så kunne vi lave vores egen typografi, nye grafiske karakterer osv. – alt hvad der kan laves af 8*8 pixels.

Problemet er imidlertid at karakterhukommelsen *ikke* kan ændres. Den ligger jo i ROM, som er en fast og uforanderlig del af maskinen. Men vi *kan* derimod godt lave en kopi af karakterhukommelsen et sted i RAM – maskinens arbejdshukommelse – og så bruge disse data i forbindelse med Am-

strads effektive SYMBOL kommando, og på den måde få lavet nye karakterer. Og det er netop det dette program kan hjælpe os med.

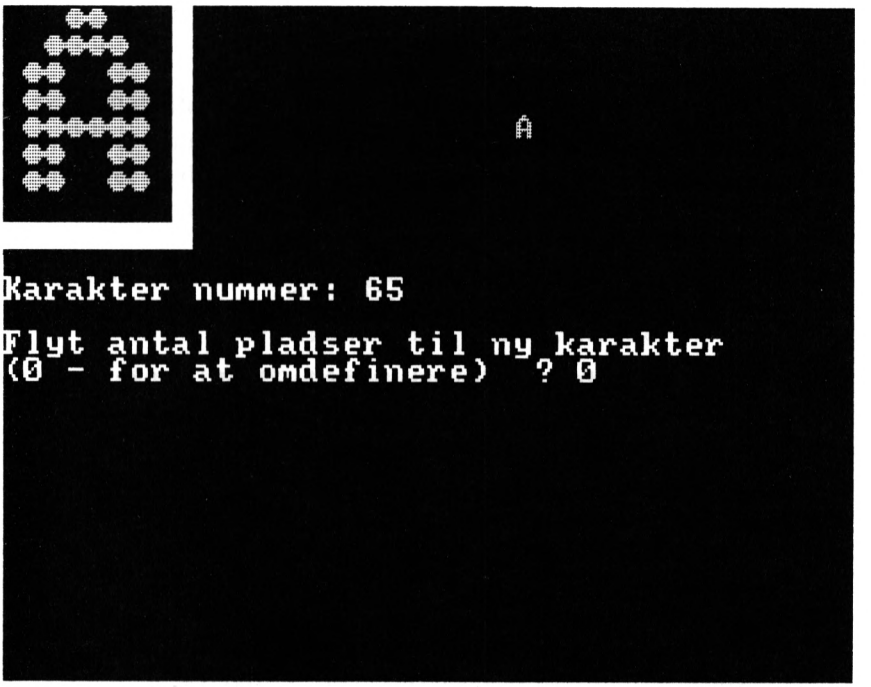


Fig. 3.2: Skærmdump fra KARAKTERER.

Modul 3.1.1: Kontrol

Normalt starter vi ikke med kontrolmodulet, men dette er meget kort, og der kræves i alle tilfælde en GOSUB kommando for at få startet programmet.

Modul 3.1.1: Linie 10000-10050

```
10000 REM ****************************
10010 REM Kontrol
10020 REM ****************************
10030 GOSUB 11000
10040 GOSUB 12000
10050 CLS:END
```

Modul 3.1.2: Skift til et nyt karaktersæt

Formålet med dette modul er at lave en kopi af karakterhukommelsen i ROM, som beskrevet i indledningen. Her bliver karakterkoderen indlæst i et array K%, som giver os de muligheder for at modificere og ændre, som vi har brug for.

Modul 3.1.2: Linie 11000-11210

```
11ØØØ REM ****************************
11Ø1Ø REM Initialisering
11Ø2Ø REM ****************************
11Ø3Ø CLS:PEN 1:PRINT "Første brugerdefi
nerede karakter":INPUT "(32-255)";fk
11Ø4Ø IF fk<32 OR fk>255 THEN PRINT "  *
*** UDENFOR OMRÅDE PRØV IGEN ****":GOTO
11Ø3Ø
11Ø5Ø SYMBOL AFTER fk
11Ø6Ø IF in=Ø THEN k=fk:DIM a%(7),t%(7),
m%(255,7):in=1
11Ø7Ø menu$="nsr1Ødighxe"+CHR$(24Ø)+CHR$
(241)+CHR$(242)+CHR$(243)
11Ø8Ø MODE 2
11Ø9Ø LOCATE 36,14:PRINT "VENT..."
111ØØ LOCATE 1,1
1111Ø FOR i=fk TO 255
1112Ø PRINT CHR$(i);
1113Ø NEXT i
1114Ø FOR i=Ø TO 255-fk
1115Ø FOR j=Ø TO 7
1116Ø m%(fk+i,j)=PEEK(48*1Ø24+j*2Ø48+i)
1117Ø NEXT j
1118Ø NEXT i
1119Ø MODE 1
112ØØ a$="r"
1121Ø RETURN
```

Kommentar

Linie 11030-11050: På Amstrad er der mulighed for at definere mellem 16 og 255 karakterer – men det er kun muligt at definere over 16 karakte-

rer hvis maskinen først får kommandoen SYMBOL AFTER. SYMBOL AFTER efterfulgt af et tal betyder at du vil kunne omdefinere alle karakterkoder fra dette tal og op til 255 – det vender vi tilbage til. Disse linier gør det muligt at omdefinere karakterer fra 32 og opefter. I normal brug kan de 16 karakterer fra 240 til 255 omdefineres.

Linie 11060: Bortset fra i ét bestemt tilfælde, så bruges denne linie til at dimensionere hovedarrayet K%, som indeholder data for karaktererne (256 linier med otte tal i hver linie), og to mindre arrays som bruges når der laves om på de enkelte karakterer.

Linie 11070: Denne linie er vigtig, men betydningen bliver først forklaret når vi kommer til det modul, der indeholder programmenuen.

Linie 11080-11190: Dette er ret snedigt: Fremfor at indlæse data fra hukommelsen, er det lettere at hente dem et mere indlysende sted – nemlig fra skærmen. Fordelen ved det er, at hvis du en anden gang vil hente data for udvalgte karakterer, så er det nok blot at udskrive dem på skærmen. For at kunne gøre det må vi først skrifte til MODE 2, tekstskærmen med den højeste opløsning. Grunden til det er at i MODE 2 står pixlerne nøjagtigt som det lige er beskrevet. I MODE 1 bliver de enkelte binære cifre ikke kopieret direkte over på skærmen, idet der bliver udskrevet *to* pixel vandret for hvert binært ciffer.

Når vi er i MODE 2, skal vi blot udskrive alle karaktererne fra FK (Første Karakter – defineret af brugeren) og op til 255 fra den første udskriftposition på skærmen. Det andet sæt løkker (linie 11140-11180) læser nu de bytes i hukommelsen, hvor de pågældende karakterer ligger, som giver udskriften på skærmen. Dette er ikke helt så enkelt, som det kunne være, og det skyldes at skærmhukommelsen er opbygget lidt specielt. Den del af hukommelsen, som indeholder det der står på skærmen, er opbygget sådan (vi går ikke ind på *hvorfor* det er sådan), at hvis du vil finde de otte bytes, der danner karakteren øverst til venstre (i MODE 2) så ligger karakterens øverste byte som byte 49152 i hukommelsen, den næste ligger som 51200 (49152 + 2048) den næste igen 2048 højere osv. Den næste karakter starter med byte 49153 og fortsætter ligeledes i trin på 2048 bytes. Når enden af den første linie karakterer er nået, vil vi komme til den første karakter i næste linie, hvis vi rykker yderligere én karakter frem. Hvis du stadig ikke er helt sikker på dette her, så prøv at indlæse dette lille program:

```
10 MODE 2
20 for I=49152 TO 49152+1024
30 POKE I,255
40 NEXT I
```

Du vil her se at selvom du POKEr skærmhukommelsen i trin af én byte, så bliver det kun den øverste række pixels i hver skærmlinie, der kommer til syne.

Nu skulle du kunne se hvad der sker i linie 11160, når der læses en karakter ad gangen fra skærmhukommelsen.

Linie 11200: Denne opstiller en variabel, som bruges til at lagre menuvalgene, således at programmet ikke afsluttes første gang det er kørt.

Modul 3.1.3: Udskrift af en forstørret karakter

Kernen i dette program er at gøre det lettere at omdefinere karakterer. En af måderne det sker på er ved at lave en forstørret udgave af en karakter, som markøren kan flyttes rundt på. Dette modul sørger for udskrift af karakteren i stort format.

Modul 3.1.3: Linie 12000-12220

```
12000 REM ****************************
12010 REM Lav symbolnet
12020 REM ****************************
12030 WHILE a$<>"e"
12040 CLS:PEN 1
12050 FOR i=0 TO 7
12060 IF ok=0 THEN a%(i)=m%(k,i)
12070 b$=RIGHT$(BIN$(256+a%(i)),8)
12080 FOR j=1 TO 8
12090 IF MID$(b$,j,1)="1" THEN PRINT CHR
$(231);ELSE PRINT " ";
12100 NEXT j
12110 PEN 3:PRINT CHR$(143):PEN 1
12120 NEXT i:ok=1
12130 PEN 3:PRINT STRING$(9,CHR$(143)):P
EN 1
12140 LOCATE 25,5:PRINT CHR$(k)
12150 PEN 3:LOCATE 1,11
```

```
12160 PRINT "Karakter nummer:";k:PRINT
12170 IF a$="r" THEN INPUT "Flyt antal p
ladser til ny karakter      (0 - for at
omdefinere)  ";mm:k=k+mm
12180 IF k<fk THEN k=fk
12190 IF k>255 THEN k=255
12200 IF mm=0 THEN GOSUB 13000 ELSE ok=0
12210 WEND
12220 RETURN
```

Kommentar

Linie 12040-12120: Data for den valgte karakter læses fra arrayet K% ind i A%, og udskriver derefter en forstørret udgave af karakteren på skærmen. Det gøres ved først at bruge BIN$ til at oversætte hver af de otte bytes til rækker af ettaller og nuller. Derefter udskrives en fyldt cirkel (CHR$(231)) på de steder der svarer til ettallerne i strengen. *Symbolnettet* på 8*8 indrammes af en række CHR$(143)-karakterer. Til slut sættes variablen OK lig 1, således at modulet, hvis det gentages uden at der er ændret på karakteren, ikke igen læser data ind i A%.

Linie 12140: En karakter i normal størrelse udskrives til højre for den forstørrede udgave.

Linie 12170-12190: Når programmet er kørt første gang vil den udskrevne karakter være den første i variablen FK – og derfor vil det være den første karakter der kan omdefineres. Linierne her gør det muligt at springe rundt mellem karaktererne fra FK til 255.

Test

Kør programmet og vælg karakter nummer 65 som den første karakter, der kan omdefineres. Efter linierne er udskrevet på skærmen – og en efterfølgende pause hvor dataoverførslen sker – skal der komme et stort 'A' tilsyne øverst til venstre. Det skal være muligt at få udskrevet alle karaktererne fra 'A' og frem, men det er endnu ikke mulige at ændre på dem. Senere vil det blive muligt ved at taste '0' og derved kalde et af de følgende moduler. Normalt kan programmet stoppes når der tastes '0', men nu kan det ske ved < ESC >.

Modul 3.1.4: En bruger-defineret markør

Dette ret enkle modul laver en speciel markør til symbolnettet.

Modul 3.1.4: Linie 14000-14120

```
14ØØØ REM ****************************
14Ø1Ø REM Markør
14Ø2Ø REM ****************************
14Ø3Ø b$=RIGHT$(BIN$(256+a%(y)),8)
14Ø4Ø LOCATE x+1,y+1:PAPER 2:PEN 1
14Ø5Ø IF MID$(b$,x+1,1)="1" THEN PRINT C
HR$(231);ELSE PRINT " ";
14Ø6Ø a$=""
14Ø7Ø WHILE a$=""
14Ø8Ø a$=LOWER$(INKEY$)
14Ø9Ø WEND
141ØØ LOCATE x+1,y+1:PAPER Ø
1411Ø IF MID$(b$,x+1,1)="1" THEN PRINT C
HR$(231);ELSE PRINT " ";
1412Ø RETURN
```

Kommentar

Linie 14040-14050: Markøren kaldes frem ved at ændre PAPER og INK farverne, for på den måde at få det til at ligne en 'rigtig' markør, og derefter udskrives enten et mellemrum eller den lille fyldte cirkel, som bruges til den forstørrede udgave af karakteren.

14060-14090: Her ventes indtil en tast trykkes ned.

Linie 14100-14110: Karakteren hvor markøren er udskrives igen normalt.

Test

Indlæs:

RUN 14000 <ENTER>

og der skal komme en markør frem øverst til venstre. Når en tilfældig tast trykkes ned stopper programmet med 'Unexpected RETURN'

Modul 3.1.5: Menu

Dette modul klarer flere opgaver: Det udskriver en menu, flytter den blinkende markør, laver eller sletter de forstørrede pixels og kalder de efterfølgende moduler, når der skal ændres ved den udskrevne karakter. Menuen indeholder brugervejledning. Det eneste nye her er faktisk måden de efterfølgende moduler kaldes på.

Modul 3.1.5: Linie 13000-13220

```
13ØØØ REM ******************************
13Ø1Ø REM Omdefinér karakter
13Ø2Ø REM ******************************
13Ø3Ø LOCATE 1,13:PRINT STRING$(8Ø," ")
13Ø4Ø LOCATE 1,13
13Ø5Ø PRINT "'n' - negativ"
13Ø6Ø PRINT "'s' - spejlvender"
13Ø7Ø PRINT "'r' - retur"
13Ø8Ø PRINT "'1' - farver pixel"
13Ø9Ø PRINT "'Ø' - sletter pixel"
131ØØ PRINT "'d' - drejer 9Ø grader"
1311Ø PRINT "'i' - indlæser i hukommelse
n"
1312Ø PRINT "'g' - gemmer på bånd"
1313Ø PRINT "'h' - henter på bånd"
1314Ø PRINT "'x' - giver standard karakt
ersæt"
1315Ø PRINT "'e' - END"
1316Ø PRINT "Markørpilene flytter markør
en"
1317Ø ret=Ø:WHILE ret=Ø
1318Ø GOSUB 14ØØØ
1319Ø z=INSTR(menu$,a$)
132ØØ ON z GOSUB 15ØØØ,16ØØØ,17ØØØ,18ØØØ
,19ØØØ,2ØØØØ,21ØØØ,22ØØØ,23ØØØ,24ØØØ,25Ø
ØØ,26ØØØ,27ØØØ,28ØØØ,29ØØØ
1321Ø WEND
1322Ø RETURN
```

Kommentar

Linie 13170-13210: Måske kan du huske at vi i initialiseringsmodulet lavede en ret speciel streng, som blev kaldt MENU$. I disse linier findes forklaringen. De gør det nemlig muligt at omsætte en indtastning af et bogstav til et tal ved hjælp af INSTR. Placeringen af en karakter i MENU$ er den værdi, der tildeles Z og som bruges til ON Z GOSUB. Variablen RET bruges til at fortælle modulet om den forstørrede karakter skal udskrives igen. Hvis RET er nul når der vendes tilbage efter GOSUB, så bliver den forstørrede karakter ikke udskrevet igen, og på den måde spares tid.

Test

Kør programmet. Når 'A' er blevet udskrevet, skal du kunne få menuen frem ved at taste '0'. Ingen af mulighederne kan endnu vælges – de giver blot en fejlmelding.

Modul 3.1.6: Retur til valg af karakter

Ved at vælge 'R' på menuen vendes tilbage til det foregående modul, hvor der kan vælges en ny karakter.

Modul 3.1.6: Linie 17000-17030

```
17ØØØ REM ******************************
17Ø1Ø REM Retur
17Ø2Ø REM ******************************
17Ø3Ø ret=1:RETURN
```

Test

Nu skal det være muligt at skifte frem og tilbage mellem menuen og modulet hvor der kan vælges karakterer.

Modul 3.1.7: Markøren flyttes

Den markør, der allerede er lavet, skal vi nu kunne flytte. Det sker simpelthen ved at ændre markørkoordinaterne – variablerne X og Y.

Modul 3.1.7: Linie 26000-29040

```
26000 REM ****************************
26010 REM Op
26020 REM ****************************
26030 IF y>0 THEN y=y-1
26040 RETURN
27000 REM ****************************
27010 REM Ned
27020 REM ****************************
27030 IF y<7 THEN y=y+1
27040 RETURN
28000 REM ****************************
28010 REM Venstre
28020 REM ****************************
28030 IF x>0 THEN x=x-1
28040 RETURN
29000 REM ****************************
29010 REM Højre
29020 REM ****************************
29030 IF x<7 THEN x=x+1
29040 RETURN
```

Test

Når menuen er på skærmen skal markøren ved den forstørrede karakter kunne flyttes. Det er endnu ikke muligt at ændre på karakteren.

Modul 3.1.8: Farvning og sletning af pixel

Ved at vælge '0' eller '1' på menuen kan en pixel enten slettes eller udskrives på den forstørrede karakter. Det klares i dette modul ved hjælp af de logiske operatorer AND og OR.

Modul 3.1.8: Linie 18000-19060

```
18000 REM ****************************
18010 REM Farv pixel
18020 REM ****************************
18030 a%(y)=a%(y) OR 2^(7-x)
18040 LOCATE x+1,y+1
18050 PRINT CHR$(231)
18060 RETURN
```

```
19000 REM ****************************
19010 REM Slet pixel
19020 REM ****************************
19030 a%(y)=a%(y) AND 255-2^(7-x)
19040 LOCATE x+1,y+1
19050 PRINT " "
19060 RETURN
```

Kommentar

Linie 18030: OR bruges ofte når et eller flere af de binære cifre i et tal skal skiftes til 1. OR sammenligner to binære tal og laver et tredie, hvor hvert binært ciffer er 1, hvis *et af de to* oprindelige cifre var 1. Et eksempel kan illustrere det, – hvis de to oprindelige cifre var:

124 = 01111100

og

3 = 00000011

bliver resultatet 127, eller 01111111, da hvert af de binære cifre var 1 i et af de to tal.

Hvis vi vil sikre at et bestemt binært ciffer – nummer N fra højre bliver 1 skal vi blot bruge OR 2^N.

I programmet er det at skifte et binært ciffer lig 1, det samme som at farve en pixel, idet de enkelte pixels tilstand bestemmes af de binære cifre i de otte tal som definerer en karakter.

Linie 19030: Det modsatte af OR, når det drejer sig om at skifte de enkelte binære cifre mellem 1 og 0, er AND. Når to cifre forbindes af AND, bliver det binære ciffer i det tal der kommer ud af det kun 1, hvis *begge* tal har 1 på denne position. Således bliver resultatet 0, hvis tallene 124 og 3 blev forbundet med AND. Hvis et tal mellem 0 og 255 forbindes med 255 (med AND) bliver resultatet det samme, da alle de binære cifre i 255 bliver skiftet til 0 ved at trække 2^N fra 255, hvor N er nummeret på det ciffer som skal skiftes til 0. Heraf følger at hvis et tal, X, forbindes med AND med 255-2^N, så bliver cifret N i tallet X skiftet til 0.

I programmet svarer det til at slette en pixel.

Test

Nu skulle det være muligt at slette pixels med '1' og '0' på menuen.

Modul 3.1.9: En invers (negativ) karakter laves

Nu kommer vi til en serie på tre moduler som gør det nemmere at ændre på en af karaktererne, idet vi her kan udføre operationer på hele karakteren. Dette modul laver en invers (eller negativ) udgave af karakteren, næste modul spejlvender og det tredie drejer karakteren 90 grader. Den negative karakter bliver lavet ved at de pixels der er farvet bliver slettet, og de pixels der er slettet bliver farvet. Det sker ved at de binære ettaller bliver til nuller og omvendt.

Modul 3.1.9: Linie 15000-15060

```
15000 REM ****************************
15010 REM Negativ (invers)
15020 REM ****************************
15030 FOR i=0 TO 7
15040 a%(i)=255-a%(i)
15050 NEXT i
15060 ret=1:RETURN
```

Test

Kør programmet og tast '0' så der kan redigeres. Vælg nu 'N' på menuen og den negative karakter skal komme frem. Når der tastes 'N' igen skal den oprindelige karakter komme frem. Læg mærke til at det er den forstørrede karakter der ændres – ikke den lille karakter til højre, som er i normal størrelse. Denne karakter ændres først, når du vil have den nye karakter ind i maskinens hukommelse, – og det kan ske med et senere modul.

Modul 3.1.10: Spejlvending

Dette modul vender karakteren om så den ses spejlvendt.

Modul 3.1.10: Linie 16000-16100

```
16000 REM ****************************
16010 REM Spejlvending
16020 REM ****************************
16030 FOR i=0 TO 7
16040 b$=RIGHT$(BIN$(256+a%(i)),8)
```

```
16Ø5Ø a%(i)=Ø
16Ø6Ø FOR j=Ø TO 7
16Ø7Ø a%(i)=a%(i)+2^j*VAL(MID$(b$,j+1,1)
)
16Ø8Ø NEXT j
16Ø9Ø NEXT i
161ØØ ret=1:RETURN
```

Kommentar

Linie 16060-16080: Da hver byte tages fra A% og omdannes til en binær streng, læses denne fra venstre mod højre som én streng. Det betyder at når løkkevariablen J er 0, så repræsenterer det binære ciffer, der læses, 128. Hvis det første ciffer i den binære streng er et 1-tal, så laves det om til 2^0 (eller 1 i almindelige decimaltal). Hvis det sidste ciffer i strengen er et 1-tal laves det om til 2^7 (eller 128 i decimaltal). På den måde bliver det binære tal 'vendt om'.

Test

Kør programmet og vælg 'S' på redigerings-menuen. Efter en lille pause hvor karakteren læses over i arrayet, skal karakteren komme frem i en spejlvendt udgave.

Modul 3.1.11: Drejning på 90 grader

Hvis du forestiller dig at den karakter, du er ved at ændre på, er tegnet over på et stykke gennemsigtigt plastik, så kan alt hvad du kan lave med dette stykke plastik opfattes som en kombination af en eventuel spejlvending og en eller flere drejninger på 90 grader. Dette modul drejer karakteren 90 grader mod uret.

Modul 3.1.11: Linie 20000-20130

```
2ØØØØ REM ****************************
2ØØ1Ø REM Drej 9Ø grader
2ØØ2Ø REM ****************************
2ØØ3Ø FOR i=Ø TO 7
2ØØ4Ø b$=RIGHT$(BIN$(256+a%(i)),8)
2ØØ5Ø FOR j=Ø TO 7
2ØØ6Ø IF i=Ø THEN t%(j)=Ø
```

```
20070 t%(j)=t%(j)+2^i*VAL(MID$(b$,j+1,1)
)
20080 NEXT j
20090 NEXT i
20100 FOR i=0 TO 7
20110 a%(i)=t%(i)
20120 NEXT i
20130 ret=1:RETURN
```

Kommentar

Linie 20050-20090: Denne løkke læser de binære cifre i et af de otte tal i A% ind i det midlertidige array T%. Læg mærke til at mens de binære cifre i hvert af tallene i A% læses fra venstre mod højre, så læses de ind i T% fra oven og nedefter – det første binære ciffer i A% læses ind i det første element i T%, det andet binære ciffer i A% læses ind i det andet element i T% osv. På den måde bliver de binære cifre i den yderste lodrette kolonne i originalen til den øverste vandrette række i T%, og på den måde drejes karakteren 90 grader.

Linie Linie 20100-20120: Den ændrede karakter læses fra T% tilbage i arbejdsarrayet A%.

Test

Kør programmet og kald 'D' på redigeringsmenuen. Efter en lille pause bliver karakteren drejet 90 grader. Hvis processen gentages tre gange til, skal vi have den oprindelige karakter igen. Prøv dig frem med kombinationer af spejlvendinger, drejninger og negativ, så du lærer funktionerne at kende.

Modul 3.1.12: Indlæsning af den ændrede karakter i hukommelsen

Indtil videre har det været muligt at ændre på den forstørrede karakter, så den måske har mistet en hver lighed med den oprindelige karakter. Men det har ikke ændret det fjerneste på den lille originale karakter (til højre). De ændringer, der er lavet, er ikke blevet indlæst i karakterhukommelsen, og der er ingen grund til at gøre det, før du er tilfreds med den nye karakter du har lavet. Men *når* du har fået lavet en ny karakter, som du er tilfreds med, så kan du indlæse den i det modificerede karaktersæt med dette modul.

Modul 3.1.12: Linie 21000-21070

```
21ØØØ REM ****************************
21Ø1Ø REM Indlæs i hukommelse
21Ø2Ø REM ****************************
21Ø3Ø FOR i=Ø TO 7
21Ø4Ø m%(k,i)=a%(i)
21Ø5Ø NEXT i
21Ø6Ø SYMBOL k,a%(Ø),a%(1),a%(2),a%(3),a
%(4),a%(5),a%(6),a%(7)
21Ø7Ø ret=1:RETURN
```

Kommentar

Linie 21040: Indholdet af A% (arbejdsarrayet, der indeholder data for den ændrede karakter, som den ser ud i forstørrelsen) bliver indlæst i M% på den plads, der hvor den originale karakter ligger.

Linie 21060: De data, der ligger i A%, bruges sammen med SYMBOL kommandoen til at tilskrive nye værdier til de otte bytes som bestemmer udseendet af den pågældende karakter.

Test

Kør programmet vælg karakter nummer 65 – 'A'. Vælg 'D' på menuen således at 'A"et bliver drejet 90 grader. Tast nu 'I' og se på skærmen. Det lille 'A' til højre bliver nu drejet, fordi maskinen laver karaktererne ud fra det modificerede karaktersæt. Det kan være en god idé kun at ændre på de grafiske karakterer, med mindre du vil lave et modificeret bogstavssæt. For hvis du laver helt om på for mange tal og bogstaver, kan det blive vanskeligt at forstå og betjene maskinen.

Modul 3.1.13: Gemning af karaktersættet

Når vi har lavet et modificeret karaktersæt, er det ret praktisk at kunne gennem det til senere brug, – ellers er det jo ret omsonst at bruge tid på at lave det. Dette modul gemmer det modificerede karaktersæt på bånd.

Modul 3.1.13: Linie 22000-22110

```
22ØØØ REM ****************************
22Ø1Ø REM Gem på bånd
22Ø2Ø REM ****************************
```

```
22Ø3Ø LOCATE 1,24:OPENOUT "kar data"
22Ø4Ø PRINT #9,fk
22Ø5Ø FOR i=fk TO 255
22Ø6Ø FOR j=Ø TO 7
22Ø7Ø PRINT #9,m%(i,j)
22Ø8Ø NEXT j
22Ø9Ø NEXT i
221ØØ CLOSEOUT
2211Ø ret=1:RETURN
```

Kommentar

Linie 22040-22090: For at spare tid både ved hentning og gemning, er det kun karaktererne fra FK og frem der gemmes. Der er ikke nogen idé i at vente på at alle 256 karakterer bliver gemt, hvis det kun er de sidste 20 der er ændret. Data for karaktersættet bliver gemt i form af tallene i M%.

Test

Efter du har modificeret nogle karakterer og indlæst dem i hukommelsen, gemmer du dem på bånd ('G' på menuen). Det eneste check, der kan laves nu er at se efter, at der ikke kommer nogen fejlmeldinger. Når det næste modul er indlæst, kan du hente karaktersættet og se, om det er blevet gemt, som det skulle.

Modul 3.1.14: Karaktersættet hentes

Når karaktersættet er gemt på bånd kan det blive hentet igen af dette modul. Det er vigtigt at du forstår dette modul, da det kan anvendes umiddelbart til at indlæse de modificerede karakterer til brug i andre programmer. Når dit modificerede karaktersæt er gemt på bånd, kan du blot lade dette modul indgå i et andet program (evt. omnummereret), hvor du gerne vil bruge karaktersættet. Kør modulet her og karaktersættet er klar til brug i det nye program!

Modul 3.1.14: Linie 23000-23170

```
23ØØØ REM ****************************
23Ø1Ø REM Hent på bånd
23Ø2Ø REM ****************************
23Ø3Ø LOCATE 1,24
```

```
23Ø4Ø OPENIN "kar data"
23Ø5Ø INPUT #9,fk
23Ø6Ø FOR i=fk TO 255
23Ø7Ø FOR j=Ø TO 7
23Ø8Ø INPUT #9,m%(i,j)
23Ø9Ø NEXT j
231ØØ NEXT i
2311Ø CLOSEIN
2312Ø SYMBOL AFTER fk
2313Ø FOR i=fk TO 255
2314Ø SYMBOL i,m%(i,Ø),k%(i,1),k%(i,2),k
%(i,3),k%(i,4),k%(i,5),k%(i,6),k%(i,7)
2315Ø NEXT i
2316Ø ok=Ø:ret=1
2317Ø RETURN
```

Kommentar

Linie 23040-23110: Den første karakter læses fra båndet, da den måske ikke svarer til den nuværende værdi, og de data der ligger på båndet læses ind i M% fra FK og frem.

Linie 23120-23150: Når data for karaktersættet ligger i M% bruges SYMBOL AFTER til at modificere det karaktersæt maskinen bruger. Normalt ville det være nemmest at gøre dette inde i den løkke, der indlæser data, således at karaktererne blev omdefineret en for en, hver gang otte bytes var indlæst. Men der er tilsyneladende en fejl i maskinens ROM, så SYMBOL AFTER virker ikke, når datastrøm 9 er åben. Det er nok en god ide at prøve, om din maskine har samme fejl.

Test

Nu skal det være muligt at hente de karaktersæt, som blev gemt på bånd i det forrige modul – vælg blot 'H' på menuen. Før du henter programmet må du sikre dig, at du har et 'normalt' karaktersæt i maskinen, – ellers er det jo umuligt at checke om modulet fungerer ordentligt.

Modul 3.1.15: Standard karaktersæt

Det er muligt at du på et tidspunkt finder ud af, at du har fået ændret for mange karakterer og at du derfor vil tilbage til standard karaktersættet.

Det kan du gøre ved at vælge 'X' på menuen. Programmet sendes tilbage til initialiseringsmodulet, så den første karakter kan blive defineret på ny. Hvis du vil have maskinen indstillet på 'afbudsværdien' (eller 'standardværdien') er den første karakter 240.

Modul 3.1.15: Linie 24000-24040

```
24ØØØ REM ******************************
24Ø1Ø REM Standard karaktersæt
24Ø2Ø REM ******************************
24Ø3Ø GOSUB 11ØØØ
24Ø4Ø ok=Ø:ret=1:RETURN
```

Test

Kør programmet og lav en ændring af en af karaktererne og indlæs denne ændrede karakter i hukommelsen. Vælg derefter 'X' på menuen og den oprindelige karakter kommer frem igen.

Modul 3.1.16: Afslutning af programmet

Ved at vælge 'E' på menuen afsluttes programmet uden at det oprindelige karaktersæt tages i brug igen.

Modul 3.1.16: Linie 25000-25030

```
25ØØØ REM ******************************
25Ø1Ø REM End
25Ø2Ø REM ******************************
25Ø3Ø ret=1:RETURN
```

Test

Modificér en af karaktererne, indlæs den i hukommelsen og afslut programmet med 'E'. Nu skal ændringen stadig ligge i maskinens karakterhukommelse.

Program 3.2: Designer

De fleste har uden tvivl set de flotte figurer, som kan laves med CAD-programmet (Computer Aided Design – 'computerstøttet grafik'). Her kan

brugeren nemt og hurtigt lave indviklede geometriske figurer, fylde ud med farver, forstørre og formindske, lave rettelser osv. Dette *designer-program* er – i noget mindre målestok – et CAD-program. Men selvom det ikke er så avanceret, så kan man alligevel bl.a. lave mønstre og figurer, som er meget større end selve skærmen, så man kan bruge skærmen som en slags 'flytbart vindue', således at detaljer af vores design kan undersøges, og omvendt kan hele designet formindskes, så det kan være på skærmen på én gang. Derudover kan linier, cirkler og polygoner tilføjes og slettes efter behag, uden at det påvirker resten af designet. Endelig kan det færdige design gemmes på bånd til senere brug.

I programmet introduceres følgende:

1) En brugerstyret blinkende markør i højopløsning.
2) Gemning af grafiske mønstre på bånd.
3) Arbejde med geometriske figurer.

Modul 3.2.1: Initialisering

Dette initialiseringsmodul bestemmer skærmfarverne, laver to vinduer og definerer en række variabler, som vi vender tilbage til i senere kommentarer.

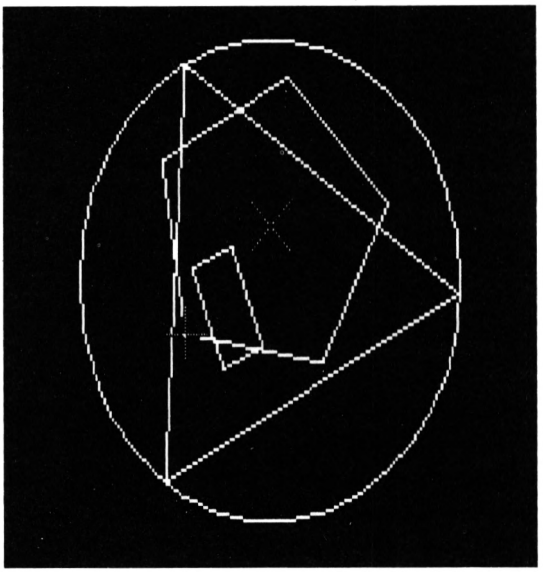


Fig. 3.3: Skærmdump fra designer.

Modul 3.2.1: Linie 11000-11250

```
11ØØØ REM ****************************
11Ø1Ø REM Initialisering
11Ø2Ø REM ****************************
11Ø3Ø MODE 1
11Ø4Ø BORDER 26
11Ø5Ø INK Ø,Ø
11Ø6Ø INK 1,26
11Ø7Ø INK 2,18
11Ø8Ø INK 3,18
11Ø9Ø PAPER 1:PEN Ø
111ØØ CLS
1111Ø WINDOW 27,4Ø,1,25
1112Ø ORIGIN 2ØØ,2ØØ,Ø,398,398,Ø
1113Ø CLG
1114Ø WINDOW #1,27,4Ø,25,25:PAPER #1,1:P
EN #1,Ø
1115Ø xormode$=CHR$(23)+CHR$(1)
1116Ø normal$=CHR$(23)+CHR$(Ø)
1117Ø menu$=""
1118Ø FOR i=24Ø TO 247:menu$=menu$+CHR$(
i):NEXT i
1119Ø menu$=menu$+"12pclf"+CHR$(13)+"svg
d|"
112ØØ enh=1
1121Ø pix=2
1122Ø venstre=-1ØØ:hojre=99
1123Ø top=99:bund=-1ØØ
1124Ø DIM a%(1ØØØ,4),x(1),y(1)
1125Ø RETURN
```

Modul 3.2.2: To grafiske markører

I ethvert design-program er det en nødvendighed at brugeren ved hvor 'tegnepositionen' på skærmen er. Det sker normalt ved, at der placeres en markør på det pågældende sted på skærmen. I dette program bruges der to markører til at definere de to ender af et liniestykke, de to modstående hjørner i en firkant osv. Begge markører kan flyttes rundt på hele skærmen med markørtasterne, uden at det påvirker det design, der allerede er på skærmen.

Modul 3.2.2: Linie 12000-13080

```
12000 REM ****************************
12010 REM Markør 1
12020 REM ****************************
12030 PRINT xormode$
12040 PLOT x(0)*pix-16,y(0)*pix,2
12050 DRAWR 32,0
12060 PLOT x(0)*pix,y(0)*pix-16
12070 DRAWR 0,32
12080 RETURN
13000 REM ****************************
13010 REM Markør 2
13020 REM ****************************
13030 PRINT xormode$
13040 PLOT x(1)*pix-16,y(1)*pix-16,2
13050 DRAWR 32,32
13060 PLOT x(1)*pix-16,y(1)*pix+16
13070 DRAWR 32,-32
13080 RETURN
```

Kommentar

Linie 12030 og 13030: XORMODE$ blev i initialiseringsmodulet defineret som CHR$(23) plus CHR$(1). CHR$(23) er en kontrolkarakter – en kontrolkarakter udskrives ikke på skærmen, men den påvirker den måde maskinen arbejder på.

Denne kontrolkarakter sætter maskinen i 'XOR-mode' (exclusive OR mode – se evt. manualen 4.18 og 9.1f). Det betyder, at det der tegnes på skærmen i XOR-mode, vil ændre de pixels, der tegnes med, til den modsatte status – hvis en pixel f.eks. har baggrundsfarve, vil den få forgrundsfarve – og omvendt. Fordelen ved det er, at hvis en figur tegnes *to* gange i XOR-mode, vil den simpelthen blive slettet igen. Med andre ord kan vi altså tegne en figur og slette den igen, uden at det påvirker de øvrige figurer på skærmen.

Linie 12040-12070 og 13040-13070: De to markører udskrives – den ene med et lodret/vandret kryds på og den anden med et diagonalt kryds på. Placeringen af krydset bestemmes af de to arrays X og Y (der hver indeholder to variable), sammen med værdien af variablen PIX. Grunden til at vi har brug for PIX er, at vi senere i programmet får mulighed for at se

vores design i forskellige målestokke, og her sørger PIX for at markøren bliver rigtigt placeret i forhold til den valgte målestok.

Modul 3.2.3: Valg af markør

Når vi har to markører, er det selvfølgelig nødvendigt, at vi kan bestemme, hvilken af dem vi vil flytte på med markørtasterne. Dette modul kaldes fra hovedmenuen, som vi kommer til om lidt, og der ændres simpelthen på værdien af variablen MN.

Modul 3.2.3: Linie 23000-24040

```
23000 REM ******************************
23010 REM Vælg markør 1
23020 REM ******************************
23030 mn=0
23040 RETURN
24000 REM ******************************
24010 REM Vælg markør 2
24020 REM ******************************
24030 mn=1
24040 RETURN
```

Modul 3.2.4: Markørerne flyttes

Endelig skal vi også kunne flytte markørerne. Det sker som nævnt med markørtasterne. Uden <SHIFT> flyttes de én skærmenhed ad gangen og med <SHIFT> flyttes de 16 enheder. Hver af subrutinerne kaldes direkte fra hovedmenuen. Hvis der kun bruges én subrutine med forskellige IF-sætninger til at styre markørerne, bliver markørbevægelserne ret langsomme. Denne løsning – med mange subrutiner fylder noget mere, men den er hurtigere.

Modul 3.2.4: Linie 15000-22040

```
15000 REM ******************************
15010 REM Markør 1 op
15020 REM ******************************
15030 y(mn)=y(mn)+enh
15040 RETURN
```

```
16000 REM ****************************
16010 REM Markør 1 ned
16020 REM ****************************
16030 y(mn)=y(mn)-enh
16040 RETURN
17000 REM ****************************
17010 REM Markør 1 til venstre
17020 REM ****************************
17030 x(mn)=x(mn)-enh
17040 RETURN
18000 REM ****************************
18010 REM Markør 1 til højre
18020 REM ****************************
18030 x(mn)=x(mn)+enh
18040 RETURN
19000 REM ****************************
19010 REM Markør 16 op
19020 REM ****************************
19030 y(mn)=y(mn)+enh*16
19040 RETURN
20000 REM ****************************
20010 REM Markør 16 ned
20020 REM ****************************
20030 y(mn)=y(mn)-enh*16
20040 RETURN
21000 REM ****************************
21010 REM Markør 16 til venstre
21020 REM ****************************
21030 x(mn)=x(mn)-enh*16
21040 RETURN
22000 REM ****************************
22010 REM Markør 16 til højre
22020 REM ****************************
22030 x(mn)=x(mn)+enh*16
22040 RETURN
```

Modul 3.2.5: Hovedmenuen

Nu har vi to markører som kan flyttes rundt på skærmen, og det næste vi skal lave er hovedmenuen, som kan kalde de forskellige subrutiner ved tryk på en tast. Modulet kan først testes når det efterfølgende kontrolmo-

dul er indlæst. De fleste af kommandoerne på menuen kan naturligvis først bruges senere, når de pågældende subrutiner er indlæst.

Modul 3.2.5: Linie 14000-14440

```
14ØØØ REM ****************************
14Ø1Ø REM Hovedmenu
14Ø2Ø REM ****************************
14Ø3Ø CLS
14Ø4Ø PRINT "   HOVEDMENU":PRINT
14Ø5Ø PRINT "X1:"
14Ø6Ø PRINT "Y1:":PRINT
14Ø7Ø PRINT "X2:"
14Ø8Ø PRINT "Y2:":PRINT
14Ø9Ø PRINT "Skala:";enh
141ØØ PRINT "Figurer:";fig:PRINT
1411Ø PRINT "'1' Markør 1"
1412Ø PRINT "'2' Markør 2"
1413Ø PRINT "'P' Polygon"
1414Ø PRINT "'C' Cirkel"
1415Ø PRINT "'L' Linie"
1416Ø PRINT "'F' Firkant"
1417Ø PRINT "'";CHR$(1);CHR$(13);"' OK"
1418Ø PRINT "'S' Slet"
1419Ø PRINT "'V' Vindue"
142ØØ PRINT "'G' Gem"
1421Ø PRINT "'D' Del-mode"
1422Ø PRINT "'|' END"
1423Ø t=Ø:WHILE t<19
1424Ø GOSUB 12ØØØ
1425Ø GOSUB 13ØØØ
1426Ø LOCATE 4,3:PRINT x(Ø);TAB(14)
1427Ø LOCATE 4,4:PRINT y(Ø);TAB(14)
1428Ø LOCATE 4,6:PRINT x(1);TAB(14)
1429Ø LOCATE 4,7:PRINT y(1);TAB(14)
143ØØ t=Ø:WHILE t=Ø
1431Ø t$="":WHILE t$=""
1432Ø t$=LOWER$(INKEY$)
1433Ø WEND
1434Ø t=INSTR(menu$,t$)
1435Ø WEND
1436Ø GOSUB 12ØØØ
```

```
14370 GOSUB 13000
14380 ON t GOSUB 15000,16000,17000,18000
,19000,20000,21000,22000,23000,24000,25
00,26000,27000,28000,29000,30000,31000,32000
14390 IF x(mn)>hojre THEN x(mn)=hojre
14400 IF x(mn)<venstre THEN x(mn)=venstr
e
14410 IF y(mn)>top THEN y(mn)=top
14420 IF y(mn)<bund THEN y(mn)=bund
14430 WEND
14440 RETURN
```

Kommentar

Linie 14030-14290: Programmenuen bliver udskrevet til højre på skærmen – det blev der sørget for i initialiseringsmodulet. Den største del af skærmen skal bruges til selve grafikken. Udover menuen bliver markørerne sat på plads og deres positioner udskrevet.

Linie 14300-14350: De to løkker afventer nedtrykning af en tast, og indtastningen bliver derefter sammenlignet med MENU$, som blev opstillet i initialiseringsmodulet. Metoden er den samme som blev brugt i det forrige program.

Linie 14390-14420: Hvis en af markørerne flyttes udenfor grafikvinduets rammer (som bestemmes af variablerne BUND, TOP, VENSTRE og HOJRE), så ændres markørkoordinaterne af disse linier.

Modul 3.2.6: Kontrolmodulet

Kontrolmodulet indlæses nu fordi vi dermed kan teste markørerne og de senere moduler, når de bliver indlæst. Bemærk at modulet kan kalde et senere modul, som kan hente data fra bånd – det vender vi tilbage til, når vi kommer til det pågældende modul.

Modul 3.2.6: Linie 10000-10100

```
10000 REM ****************************
10010 REM Kontrol
10020 REM ****************************
10030 CLS:INPUT "Hent fra bånd (j/n)";q$
```

```
1ØØ4Ø GOSUB 11ØØØ
1ØØ5Ø IF LOWER$(q$)="j" THEN GOSUB 33ØØØ
1ØØ6Ø WHILE 1
1ØØ7Ø GOSUB 14ØØØ
1ØØ8Ø IF t=2Ø THEN MODE 1:PEN 1:PAPER Ø:
BORDER 1:CLS:END
1ØØ9Ø GOSUB 4ØØØØ
1Ø1ØØ WEND
```

Test

Kør programmet. Det skal nu være muligt at flytte markørerne med piletasterne. De øvrige programfunktioner virker endnu ikke.

Modul 3.2.7: Tegning af en linie

Nu følger der tre moduler som klarer arbejdet med at tegne henholdsvis linier, firkanter og polygoner (incl. cirkler). De kan ikke beuges med det samme, idet de ikke kaldes direkte fra menuen men via andre moduler. Dette modul tegner simpelthen en linie fra den markør 2 til markør 1.

Modul 3.2.7: Linie 37000-37050

```
37ØØØ REM ****************************
37Ø1Ø REM Tegn linie
37Ø2Ø REM ****************************
37Ø3Ø PLOT x2*pix,y2*pix,c
37Ø4Ø DRAW x1*pix,y1*pix
37Ø5Ø RETURN
```

Test

Kør programmet, så vi får grafikskærmen frem, og stop med det med '†'. Skriv derefter:

C=2:X1=50:Y1=50:GOTO 370000 <ENTER>

nu skal der komme en linie fra midten af grafikskærmen og diagonalt op til højre.

Modul 3.2.8: Tegning af cirkel eller polygon

Dette er et alsidigt modul, som laver en polygon med afstanden mellem de to markører som radius. Afhængigt af antallet af sider tegnes en trekant, firkant, en 'rigtig' polygon eller en cirkel.

Modul 3.2.8: Linie 38000-38140

```
38000 REM ****************************
38010 REM Tegn Polygon
38020 REM ****************************
38030 r=SQR((x1-x2)*(x1-x2)+(y1-y2)*(y1-
y2))
38040 z2=CINT(8*LOG(r/enh))
38050 IF NOT(z1=2 OR z1>z2) THEN z2=z1
38060 IF x1=x2 THEN a=PI/2*SGN(y1-y2)
38070 IF x1<>x2 THEN a=ATN((y1-y2)/(x1-x
2))
38080 IF x1<x2 THEN a=a+PI
38090 PLOT (x2+r*COS(a))*pix,(y2+r*SIN(a
))*pix,c
38100 FOR s=1 TO z2
38110 a1=a+2*PI*s/z2
38120 DRAW (x2+r*COS(a1))*pix,(y2+r*SIN(
a1))*pix
38130 NEXT s
38140 RETURN
```

Kommentar

Linie 38030: Den diagonale afstand mellem markørerne.

Linie 38040-38050: Z2 er antallet sider i polygonen. Udtrykket i linie 38040 giver Z2 så høj en værdi at vi får en pæn cirkel – der er ingen grund til at have alt for mange sider – der bliver ikke nogen særlig forskel at se, og programmet kører langsommere. Hvis brugeren ikke vælger en cirkel (men en polygon) sætter linie 38050 det antal sider, som brugeren har valgt – hvis tallet ikke er større end det antal, der skal bruges til at tegne en cirkel.

Linie 38060-38080: Disse tre linier udregner markør 1's vinkel i forhold til markør 2.

Linie 38090: Første punkt på cirklen. Du kan måske huske udtrykket fra ANALOG-UR programmet i kapitel 1.

Linie 38100-38130: Denne løkke udregner Z2 positionerne rundt langs periferien, og tegner dem efterhånden som de udregnes. Disse linier svarer også til ANALOG-UR programmet.

Test

Der skal defineres så mange variable til en test af dette og det næste program, at det er bedre at vente indtil flere moduler er indlæst, således at modulerne kan testes som dele af selve programmet.

Modul 3.2.9: Tegning af et rektangel

Den sidste geometriske form som programmet kan tegne er en retvinklet firkant, med to modstående hjørner som punkt 1 og 2. Dette modul er lidt mere indviklet end de foregående. Der findes nemlig ikke nogen indbygget kommando vi kan bruge, og modulet giver samtidig mulighed for at figuren kan drejes.

Modul 3.2.9: Linie 36000-36210

```
36ØØØ REM ****************************
36Ø1Ø REM Tegn firkant
36Ø2Ø REM ****************************
36Ø3Ø a=-z1/1ØØØØ
36Ø4Ø cs=COS(a)
36Ø5Ø sn=SIN(a)
36Ø6Ø x=(x1+x2)/2
36Ø7Ø y=(y1+y2)/2
36Ø8Ø rx1=x+(x1-x)*cs+(y1-y)*sn
36Ø9Ø ry1=y+(y1-y)*cs-(x1-x)*sn
361ØØ rx2=x+(x2-x)*cs+(y1-y)*sn
3611Ø ry2=y+(y1-y)*cs-(x2-x)*sn
3612Ø rx3=x+(x2-x)*cs+(y2-y)*sn
3613Ø ry3=y+(y2-y)*cs-(x2-x)*sn
3614Ø rx4=x+(x1-x)*cs+(y2-y)*sn
3615Ø ry4=y+(y2-y)*cs-(x1-x)*sn
```

```
36160 PLOT rx1*pix,ry1*pix,c
36170 DRAW rx2*pix,ry2*pix
36180 DRAW rx3*pix,ry3*pix
36190 DRAW rx4*pix,ry4*pix
36200 DRAW rx1*pix,ry1*pix
36210 RETURN
```

Kommentar

Linie 36030-36050: Variablen A indeholder den vinkel, som rektanglet skal drejes – defineret af brugeren. CS og SN bruges til at gøre udstrykket, der bestemmer drejningen af figuren, kortere.

Linie 36080-36150: Formlen for et punkt der drejes rundt om et andet er

$$X2 = XC + XA \times COS(VINKEL) + YA \times SIN(VINKEL)$$
$$Y2 = YC + YA \times COS(VINKEL) - XA \times SIN(VINKEL)$$

X2 og Y2 er X- og Y-koordinaterne for det punkt som kommer ud af drejningen. XC og YC er koordinaterne for det punkt der drejes om. XA og YA er afstandene fra XC og YC til henholdsvis X og Y. VINKEL er den vinkel, som punktet defineret af X og Y, drejes.

Rektanglet har – når det ikke er drejet – hjørnerne X1/Y1, X2/Y1, X2/Y2 og X1/Y2. Hvis du ser på programlinierne og sammenligner dem med linierne ovenfor vil du kunne se at de giver koordinaterne for hjørnerne i det drejede rektangel.

Modul 3.2.10: Kontrolmodul for tegnemodulerne

Dette lille modul bruges til at sætte tegnemodulerne sammen med de efterfølgende moduler. Dets funktion bliver forklaret i næste modul.

Modul 3.2.10: Linie 35000-35060

```
35000 REM ****************************
35010 REM Tegnekontrol
35020 REM ****************************
35030 IF z1<1 THEN GOSUB 36000
35040 IF z1=1 THEN GOSUB 37000
35050 IF z1>1 THEN GOSUB 38000
35060 RETURN
```

Modul 3.2.11: Styring af tegnemodulerne

Nu kopmmer de tre subrutiner, som forbinder indtastninger til menuen med tegnemodulerne, ved at opstille de nødvendige parametre.

Hvert modul har følgende opgaver:

1) Om nødvendigt at bede om yderligere oplysninger som f.eks. 'antal sider' eller 'vinkel'.
2) At sætte maskinen i XOR-mode med PRINT XORMODE$.
3) At slette figurer, der ikke er godkendt af brugeren, angivet ved at variablen MIDL er lig 1.
4) At definere variablerne Z1, X1, Y1, X2 og Y2 – markørernes koordinater.
5) At kalde de respektive tegnemoduler.
6) At lade variablen MIDL indikere at den tegnede figur endnu ikke er godkendt af brugeren.

Modul 3.2.11: Linie 25000-28110

```
25000 REM ******************************
25010 REM Polygon
25020 REM ******************************
25030 z=0:e=0:WHILE z<3 OR z>32767
25040 IF e=1 THEN x$="*FORKERT OMR.*":GO
TO 34000
25050 INPUT #1,"Sider";z:PRINT #1
25060 e=1:WEND
25070 PRINT xormode$:c=2
25080 IF midl=1 THEN GOSUB 35000
25090 z1=z:x1=x(0):y1=y(0):x2=x(1):y2=y(
1)
25100 GOSUB 38000
25110 midl=1
25120 RETURN
26000 REM ******************************
26010 REM Cirkel
26020 REM ******************************
26030 PRINT xormode$:c=2
26040 IF midl=1 THEN GOSUB 35000
26050 z1=2:x1=x(0):y1=y(0):x2=x(1):y2=y(
1)
26060 GOSUB 38000
```

```
26Ø7Ø midl=1
26Ø8Ø RETURN
27ØØØ REM ****************************
27Ø1Ø REM Linie
27Ø2Ø REM ****************************
27Ø3Ø PRINT xormode$:c=2
27Ø4Ø IF midl=1 THEN GOSUB 35ØØØ
27Ø5Ø z1=1:x1=x(Ø):y1=y(Ø):x2=x(1):y2=y(
1)
27Ø6Ø GOSUB 37ØØØ
27Ø7Ø midl=1
27Ø8Ø RETURN
28ØØØ REM ****************************
28Ø1Ø REM Firkant
28Ø2Ø REM ****************************
28Ø3Ø INPUT #1,"Vinkel";a:PRINT #1
28Ø4Ø a=a-18Ø*INT(a/18Ø)
28Ø5Ø z=-CINT(a/18Ø*PI*1ØØØØ)
28Ø6Ø PRINT xormode$:c=2
28Ø7Ø IF midl=1 THEN GOSUB 35ØØØ
28Ø8Ø z1=z:x1=x(Ø):y1=y(Ø):x2=x(1):y2=y(
1)
28Ø9Ø GOSUB 36ØØØ
281ØØ midl=1
2811Ø RETURN
```

Kommentar

Frem for at kommentere hver subrutine for sig, bliver hovedtrækkene beskrevet her – og de gælder alle tre moduler.

Linie 25030-25060: Yderligere oplysninger for at figuren kan tegnes – dvs. antal sider.

Linie 25080: Hvis variablen MIDL er lig 1, er figuren blevet tegnet uden at være blevet godkendt af brugeren (vi kommer straks til godkendelsen). Før den aktuelle figur bliver tegnet, sørger kontrolmodulet for tegnemodulerne i linie 35000- for, at det forrige tegnemodul kaldes igen. Da maskinens er i XOR-mode bliver figuren tegnet igen og dermed slettet. Det betyder, at hvis brugeren er igang med at tegne et kompliceret design, så er det muligt at tegne en figur, som måske ikke helt passer, og derefter at

justere markørpositionerne med denne figur som støtte. Nu kan den nye figur blive tegnet, samtidig med at den gamle automatisk bliver slettet.

Linie 25090: De relevante variabler defineres.

Test

Nu skal det være muligt at køre programmet og tegne figurerne. Men endnu er det ikke muligt at have mere end en figur på skærmen – hver ny figur sletter den forrige.

Modul 3.2.12: Godkendelse af en figur

Et program som dette er ikke meget bevendt, hvis det ikke er muligt at gemme de designs man laver. Her bliver vores figurer indlæst i et array, som senere gør det muligt både at slette enkelte figurer og at gemme hele designet på bånd. Linier og figurer bliver lagret i maskinens hukommelse når dette modul aktiveres. Modulet sørger endvidere for at figurerne bliver på skærmen, selvom der tegnes nye figurer. Modulet kaldes ved at taste <ENTER> på menuen.

Modul 3.2.12: Linie 29000-29150

```
29000 REM ******************************
29010 REM Bevar figur (OK)
29020 REM ******************************
29030 IF midl=0 THEN RETURN
29040 IF fig=1001 THEN x$="IKKE MERE PLA
DS":GOSUB 34000:RETURN
29050 PRINT normal$:c=1
29060 GOSUB 35000
29070 a%(fig,0)=z1
29080 a%(fig,1)=x1
29090 a%(fig,2)=y1
29100 a%(fig,3)=x2
29110 a%(fig,4)=y2
29120 midl=0
29130 fig=fig+1
29140 LOCATE 9,10:PRINT fig;TAB(14)
29150 RETURN
```

Kommentar

Linie 29030: Hvis MIDL ikke er 1 er der ingen figur at godkende.

Linie Linie 29050-29060: NORMAL$ som blev defineret i initialiseringsmodulet sætter XOR-mode ud af funktion, således at det der tegnes tegnes på normal vis. Modulet i linie 35000 kaldes derefter for at kalde de forskellige tegnemoduler.

Linie 29070-29130: De forskellige variabler, der skal bruges til at tegne figuren lagres til arrayet A%. MIDL sættes lig 0, for at indikere at der ikke er nogen midlertidig (ikke godkendt) figur på skærmen. Variablen FIG opskrives for at vise at der nu indgår én figur mere i vores design.

Modul 3.2.13: Sletning af en figur

Dette lille modul sørger for at slette en midlertidig figur. Det sker på samme måde, som da den blev tegnet.

Modul 3.2.13: Linie 30000-30070

```
30000 REM ******************************
30010 REM Slet
30020 REM ******************************
30030 IF midl=0 THEN RETURN
30040 PRINT xormode$:c=2
30050 GOSUB 35000
30060 midl=0
30070 RETURN
```

Test

Nu skal du kunne slette en midlertidig figur ved at taste 'S' på menuen.

Modul 3.2.14: Genoptegning af hele designet

Da vi nu kan lagre et helt design i et array, så har vi brug for dette modul, som kan genoptegne hele designet på grundlag af de data der ligger i arrayet.

Modul 3.2.14: Linie 39000-39130

```
39ØØØ REM ****************************
39Ø1Ø REM Genoptegning
39Ø2Ø REM ****************************
39Ø3Ø IF fig=Ø THEN RETURN
39Ø4Ø PRINT normal$:c=1
39Ø5Ø FOR i=Ø TO fig-1
39Ø6Ø z1=a%(i,Ø)
39Ø7Ø x1=a%(i,1)
39Ø8Ø y1=a%(i,2)
39Ø9Ø x2=a%(i,3)
391ØØ y2=a%(i,4)
3911Ø GOSUB 35ØØØ
3912Ø NEXT i
3913Ø RETURN
```

Test

Kør programmet og tegn et par figurer, og stop derefter programmet med <ESC> og skriv:

CLS#2 <ENTER>
GOTO 39000 <ENTER>

og figurerne skal komme frem på skærmen igen.

Modul 3.2.15: Sletning af figurer

Da designet ikke er lagret som én stor figur, men som flere cirkler, linier osv., er det ret enkelt at give brugeren mulighed for at slette enkelte af figurerne. Dette modul tegner figurerne op en for en, med mulighed for at slette de figurer, der ikke ønskes bevaret.

Modul 3.2.15: Linie 40000-40290

```
4ØØØØ REM ****************************
4ØØ1Ø REM Del-mode
4ØØ2Ø REM ****************************
4ØØ3Ø CLS
4ØØ4Ø CLG
4ØØ5Ø PRINT "   DEL-MODE  ":PRINT
```

```
4ØØ6Ø PRINT "Figur # 1":PRINT
4ØØ7Ø PRINT "Skala:";enh
4ØØ8Ø PRINT "Figurer:";fig:PRINT
4ØØ9Ø PRINT "'";CHR$(1);CHR$(13);"' Næst
e fig"
4Ø1ØØ PRINT "'S' Slet"
4Ø11Ø PRINT "'|' Retur"
4Ø12Ø t$="":i=Ø:WHILE i<fig
4Ø13Ø z1=a%(i,Ø)
4Ø14Ø x1=a%(i,1)
4Ø15Ø y1=a%(i,2)
4Ø16Ø x2=a%(i,3)
4Ø17Ø y2=a%(i,4)
4Ø18Ø d=Ø:WHILE t$<>"|" AND d=Ø
4Ø19Ø PRINT xormode$:c=2
4Ø2ØØ GOSUB 35ØØØ
4Ø21Ø t$=""
4Ø22Ø d=1:WEND
4Ø23Ø WHILE t$=""
4Ø24Ø t$=UPPER$(INKEY$)
4Ø25Ø WEND
4Ø26Ø IF t$="S" THEN GOSUB 41ØØØ ELSE GO
SUB 42ØØØ
4Ø27Ø WEND
4Ø28Ø temp=Ø
4Ø29Ø RETURN
```

Kommentar

Linie 40120-40220: Data for de enkelte figurer findes frem fra arrayet A% og tegnes i XOR-mode.

Linie 40230-40280: Programmet afventer at brugeren enten taster 'S' for slette figuren eller trykker på <ENTER> (eller en anden tast) for at godkende figuren. Denne del af modulet fungerer først når de næste to moduler er indlæst.

3.2.16: En figur slettes fra arrayet

Når en figur først er tegnet, kan den fjernes fra arrayet ved at den tegnes endnu en gang i XOR-mode.

Modul 3.2.16: Linie 41000-41120

```
41ØØØ REM ****************************
41Ø1Ø REM Slet
41Ø2Ø REM ****************************
41Ø3Ø PRINT xormode$:c=2
41Ø4Ø GOSUB 35ØØØ
41Ø5Ø fig=fig-1
41Ø6Ø FOR j=i TO fig-1
41Ø7Ø FOR k=Ø TO 4
41Ø8Ø a%(j,k)=a%(j+1,k)
41Ø9Ø NEXT k
411ØØ NEXT j
4111Ø LOCATE 7,6:PRINT fig;TAB(14)
4112Ø RETURN
```

Modul 3.2.17: Godkendelse af en figur i DEL-mode

Hvis en figur skal godkendes skal den blot tegnes igen uden XOR-mode.

Modul 3.2.17: Linie 42000-42070

```
42ØØØ REM ****************************
42Ø1Ø REM Næste figur
42Ø2Ø REM ****************************
42Ø3Ø PRINT normal$:c=1
42Ø4Ø GOSUB 35ØØØ
42Ø5Ø i=i+1
42Ø6Ø LOCATE 8,3:PRINT i+1
42Ø7Ø RETURN
```

Test

Når du har tegnet et design bestående af nogle få figurer kan du prøve at taste 'D' og derved kalde de sidste tre moduler. Nu skal det være muligt at kalde figurerne frem en for en, og slette de figurer som ikke skal indgå i designet. Hvis du taster '|' med det samme, skal hele designet blive tegnet op uden ændringer.

Modul 3.2.18: Fejlmeldinger

Dette lille modul udskriver de fejlmeldinger, som findes rundt om i programmet.

Modul 3.2.18: Linie 34000-34070

```
34ØØØ REM ****************************
34Ø1Ø REM Fejl
34Ø2Ø REM ****************************
34Ø3Ø PRINT #1,x$;
34Ø4Ø SOUND 1,1ØØØ,1ØØ
34Ø5Ø FOR i=1 TO 15ØØ:NEXT i
34Ø6Ø PRINT #1
34Ø7Ø RETURN
```

Modul 3.2.19: Vinduer og skalaer

Indtil nu har vi slet ikke været inde på en af de mest anvendelige sider af Designer-programmet. Som nævnt i indledningen giver dette program mulighed for at lave designs, der er meget større end skærmen. Dette modul gør det muligt at bruge skærmen som et flytbart vindue, således at vi ikke er begrænset til at arbejde på det lille område skærmen dækker. Og omvendt kan et stort design formindskes, således at det kan være på skærmen på én gang. Det er et ret indviklet modul, men det ville være endnu mere kompliceret, hvis ikke Amstrad havde så god en grafik. Specielt er det her en fordel, at ORIGIN nemt kan flyttes og at der kan tegnes uden for selve skærmen, uden at det giver fejlmeldinger.

Modul 3.2.19: Linie 31000-31280

```
31ØØØ REM ****************************
31Ø1Ø REM Vindue
31Ø2Ø REM ****************************
31Ø3Ø x$="*FORKERT OMR.*"
31Ø4Ø e=Ø:WHILE e=Ø OR x<-16384 OR x>163
83
31Ø5Ø IF e=1 THEN GOSUB 34ØØØ
31Ø6Ø INPUT #1,"X";x
31Ø7Ø e=1:WEND
31Ø8Ø e=Ø:WHILE e=Ø OR y<-16384 OR y>163
83
```

```
31Ø9Ø IF e=1 THEN GOSUB 34ØØØ
311ØØ INPUT #1,"Y";y
3111Ø e=1:WEND
3112Ø e=Ø:WHILE e=Ø OR enh<1
3113Ø IF e=1 THEN GOSUB 34ØØØ
3114Ø INPUT #1,"Skala";enh:PRINT #1
3115Ø e=1:WEND
3116Ø pix=2/enh
3117Ø LOCATE 7,9:PRINT enh;TAB(14)
3118Ø venstre=x-1ØØ*enh:IF venstre<-1638
4 THEN venstre=-16384
3119Ø hojre=x+99*enh:IF hojre>16383 THEN
 hojre=16383
312ØØ top=y+99*enh:IF top>16383 THEN top
=16383
3121Ø bund=y-1ØØ*enh:IF bund<-16384 THEN
 bund=-16384
3122Ø x(Ø)=x:y(Ø)=y
3123Ø x(1)=x:y(1)=y
3124Ø midl=Ø
3125Ø ORIGIN 2ØØ-x*pix,2ØØ-y*pix,Ø,398,3
98,Ø
3126Ø CLG
3127Ø GOSUB 39ØØØ
3128Ø RETURN
```

Kommentar

Linie 31030 og 31050: Denne melding udskrives af fejlsubrutinen, som lige er blevet indlæst. Denne subrutine kan kaldes mange steder i dette modul, hvis der sker fejlindtastninger.

Linie 31060-31110: Brugeren skal her indtaste de X- og Y-koordinater, som bliver centrum i det nye vindue. Størrelsen af hele designet er 32768 x 32768 enheder, og vinduet kan flyttes rundt indenfor dette område.

Linie 31120-31150: Her skal 'formindskelses-graden' – *skalaen* – indtastes af brugeren. Jo højere tal der bruges jo større område dækker skærmbilledet – og jo mindre bliver designet på skærmen. På den måde bliver

det muligt at lave et design, der er meget større end skærmen, og derefter at se hele designet i mindre målestok.

Linie 31180-31210: De variable der bestemmer skærmens grænser defineres. Hvis en af grænserne går udover den maximale størrelse, flyttes vinduet så meget, at det ligger indenfor.

Linie 31220-31270: Begge markører flyttes til centrum af det nye vindue, og den grafiske ORIGIN defineres. Derefter slettes den grafiske skærm og genoptegningsmodulet kaldes og designet tegnes på ny. Det er muligt at hele designet – eller dele af det – falder udenfor vinduet, så det ikke bliver synligt.

Test

Når der ligger et design i maskinen, skal det være muligt at flytte skærmvinduet rundt på det. Endvidere skal det være muligt at formindske (og derefter forstørre) designet.

Modul 3.2.20: Gemning og hentning på bånd

Dette er et standardprogram til lagring af data. Forklaringer kan findes i 3D GRAF programmet i forrige kapitel.

Modul 3.2.20: Linie 32000-33130

```
32ØØØ REM ******************************
32Ø1Ø REM Gem på bånd
32Ø2Ø REM ******************************
32Ø3Ø INPUT #1,"Navn:",n$:PRINT #1
32Ø4Ø OPENOUT "!"+n$
32Ø5Ø PRINT #9,fig
32Ø6Ø FOR i=Ø TO fig-1
32Ø7Ø FOR j=Ø TO 4
32Ø8Ø PRINT #9,a%(i,j)
32Ø9Ø NEXT j
321ØØ NEXT i
3211Ø CLOSEOUT
3212Ø RETURN
```

```
33ØØØ REM ****************************
33Ø1Ø REM Hent fra bånd
33Ø2Ø REM ****************************
33Ø3Ø INPUT #1,"Navn:",n$:PRINT #1
33Ø4Ø OPENIN "!"+n$
33Ø5Ø INPUT #9,fig
33Ø6Ø FOR i=Ø TO fig-1
33Ø7Ø FOR j=Ø TO 4
33Ø8Ø INPUT #9,a%(i,j)
33Ø9Ø NEXT j
331ØØ NEXT i
3311Ø CLOSEIN
3312Ø GOSUB 39ØØØ
3313Ø RETURN
```

Test

Lav et design og tast 'G' for at gemme det på bånd. Stop nu programmet og kør det igen, således at alle variabler slettes. Svar 'J' til at hente data fra båndet. Brug det navn som programmet blev gemt under, og designet skal komme frem på skærmen igen.

Er testen OK er programmet klar til brug.

Program 3.3: Musik

Nu kommer vi til et af de områder hvor Amstrad klarer sig allerbedst i forhold til konkurrenterne – musikken. Amstrad har et ret avanceret lydsystem, som giver nogle muligheder, som mange andre hjemmecomputere mangler. Og de maskiner, der har et ligeså omfattende lydsystem, er oftest sværere at programmere – på Amstrad kan de fleste uden det store besvær få fornøjelse af den hardware, der ligger i maskinen.

Programmet her er et godt eksempel på hvad der kan laves på dette område. Ved hjælp af musik-programmet kan du lave flerstemmige melodier, lave harmonier, ændre toneart og tempo mv. Hvad enten du er vant til at spille eller du er nybegynder, så håber vi at dette program vil være en fornøjelse for dig at bruge.

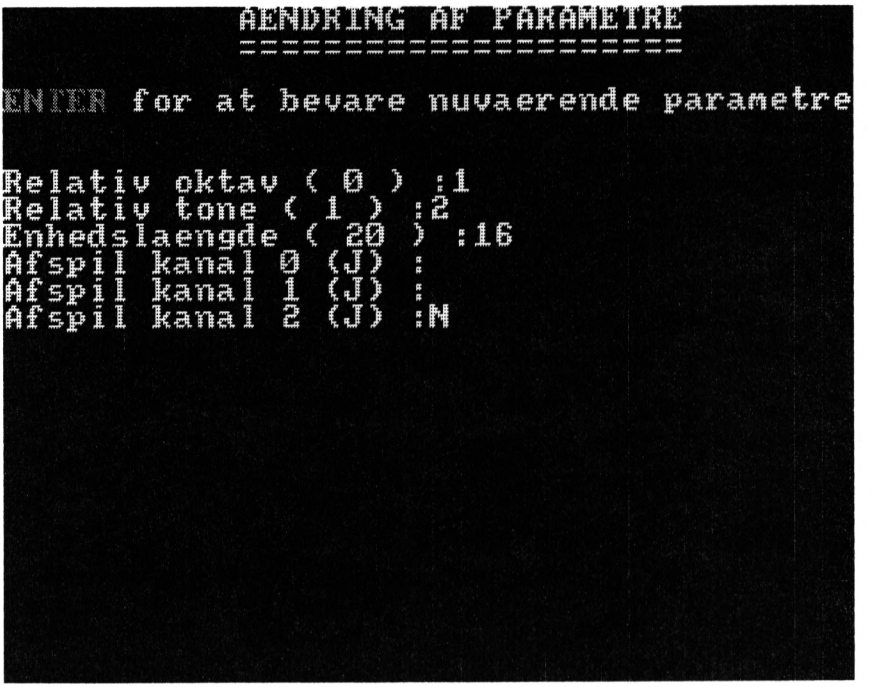


Fig. 3.4: Menu fra Musik.

Modul 3.3.1: Melodiens data

Ligesom i grafprogrammerne tidligere bruger vi her datalinier til programmets data. Melodiens data bliver på den måde nemmere at korrigere og ændre. Det betyder ikke så meget, om du forstår disse data nu – de bliver forklaret i forbindelse med de moduler, hvor de anvendes. De data du ser her, bruges til at give en nydelig udførelse af Beethovens 'Für Elise'.

Modul 3.3.1: Linie 26000-28390

```
26ØØØ REM ****************************
26Ø1Ø REM Kanal Ø Data
26Ø2Ø REM ****************************
26Ø3Ø DATA ENV1
26Ø4Ø DATA *
26Ø5Ø DATA L1,16,15
26Ø6Ø DATA 16,15,16,11,14,12
```

```
26Ø7Ø DATA L2,9,L1,P,Ø,4,9
26Ø8Ø DATA L2,11,L1,P,4,8,11
26Ø9Ø DATA L2,12,L1,P,4,16,15
261ØØ DATA 16,15,16,11,14,12
2611Ø DATA L2,9,L1,P,Ø,4,9
2612Ø DATA L2,11,L1,P,4,8,11
2613Ø DATA <1
2614Ø DATA L4,9
2615Ø DATA >
2616Ø DATA *2
2617Ø DATA <2
2618Ø DATA L2,9,L1,P,11,12,14
2619Ø DATA *
262ØØ DATA L3,16,L1,7,17,16
2621Ø DATA L3,14,L1,5,16,14
2622Ø DATA L3,12,L1,4,14,12
2623Ø DATA L2,11,L1,P,4,16,P
2624Ø DATA P,16,28,P2,15
2625Ø DATA 16,P2,15,16,15
2626Ø DATA 16,15,16,11,14,12
2627Ø DATA L2,9,L1,P,Ø,4,9
2628Ø DATA L2,11,L1,P,4,8,11
2629Ø DATA L2,12,L1,P,4,16,15
263ØØ DATA 16,15,16,11,14,12
2631Ø DATA L2,9,L1,P,Ø,4,9
2632Ø DATA L2,11,L1,P,4,12,11
2633Ø DATA <1
2634Ø DATA L2,9,L1,P,11,12,14
2635Ø DATA >
2636Ø DATA *2
2637Ø DATA L2,9,L1,P,Ø,4,9
2638Ø DATA 12,16,L4,21
2639Ø DATA end
27ØØØ REM ******************************
27Ø1Ø REM Kanal 1 Data
27Ø2Ø REM ******************************
27Ø3Ø DATA end
28ØØØ REM ******************************
28Ø1Ø REM Kanal 2 Data
28Ø2Ø REM ******************************
28Ø3Ø DATA ENV1,O-2
```

```
28Ø4Ø DATA *
28Ø5Ø DATA P2
28Ø6Ø DATA P6
28Ø7Ø DATA 9,16,21,P3
28Ø8Ø DATA 4,16,2Ø,P3
28Ø9Ø DATA 9,16,21,P3
281ØØ DATA P6
2811Ø DATA 9,16,21,P3
2812Ø DATA 4,16,2Ø,P3
2813Ø DATA <1
2814Ø DATA 9,16,21,P1
2815Ø DATA >
2816Ø DATA *2
2817Ø DATA <2
2818Ø DATA 9,16,21,P3
2819Ø DATA *
282ØØ DATA 12,19,24,P3
2821Ø DATA 7,19,23,P3
2822Ø DATA 9,16,21,P3
2823Ø DATA 4,16,28,P2,OØ,4
2824Ø DATA 16,P,P,15,16,P
2825Ø DATA P,15,16,P3
2826Ø DATA P6,O-2
2827Ø DATA 9,16,21,P3
2828Ø DATA 4,16,2Ø,P3
2829Ø DATA 9,16,21,P3
283ØØ DATA P6
2831Ø DATA 9,16,21,P3
2832Ø DATA 4,16,2Ø,P3
2833Ø DATA <1
2834Ø DATA 9,16,21,P3
2835Ø DATA >
2836Ø DATA *2
2837Ø DATA <2
2838Ø DATA 9,16,21,P3
2839Ø DATA end
```

Modul 3.3.2: Initialisering

Dette modul er lidt mere omfattende end sædvanligt, fordi melodiens data skal sættes på en form, som maskinen kan arbejde med, før programmet selv kan afspille melodien.

Modul 3.3.2: Linie 10000-10160

```
10000 REM ****************************
10010 REM Initialisering
10020 REM ****************************
10030 CLS:LOCATE 17,13:PRINT "Vent..."
10040 DIM a$(200,2),p$(2),p(2),d(2),s%(2
,500,4)
10050 FOR c=0 TO 2
10060 p=0
10070 READ a$(p,c)
10080 a$(p,c)=LOWER$(a$(p,c))
10090 IF a$(p,c)<>"end" THEN p=p+1:GOTO
10070
10100 p$(c)="J"
10110 NEXT c
10120 rok=0
10130 rto=0
10140 el=20
10150 GOSUB 25000
10160 GOSUB 15000
```

Kommentar

Linie 10040: At det er en større opgave at oversætte vores data, så maskinen kan forstå dem fremgår af de mange arrays, som bruges i dette program. De vil blive forklaret senere i programmet.

Linie 10050-10110: Dataposterne indlæses i de tre kolonner i arrayet A$. Det array, vi bruger her kan indeholde tre sæt data á 200 dataposter, svarende til Amstrads tre stemmer (tonegeneratorer). Først indlæses i data i den første kolonne. Når ordet 'end' nåes, fortsætter løkken med at læse (READ) data, men nu placeres de i arrayets anden kolonne. Du kan derfor se, at det er vigtigt, at afslutte dataposterne til hver stemme med ordet 'end', som det er gjort her. Til sidst sættes de tre elementer i arrayet P$ til 'J' for at angive at den pågældende stemme – i hvert fald til en start – skal afspilles.

Linie 10120: Variablen ROK (relativ oktav) bestemmer hvilken oktav melodiens skal spilles udfra – dataposterne bliver sat i forhold til denne oktav.

Linie 10130: På samme måde er RTO (relativ tone), den tone melodien starter med. Hvis den ændres, vil tonearten dermed også blive en anden.

Linie 10140: Variablen EL (enhedslængde) bruges til at lagre tonernes standardlængde. Hver tone i melodien vil få en længde som ganges med EL.

Linie 10150-10160: Her kaldes to subrutiner, som omsætter melodiens data til en form maskinen kan arbejde med.

Test

Indlæs disse midlertidige linier:

15000 RETURN
25000 RETURN

Kør nu programmet. Nu skal det vare et pænt stykke tid før markøren kommer frem igen. Endnu kan du ikke gøre noget med de data der er blevet behandlet, – men du kan checke, at der ikke er nogen syntax fejl i det, du har indtastet.

Modul 3.3.3: Lydenvelopes

En tones *volume-envelope* (ENV) kan forstås som den kurve en tones lydstyrke følger mens den lyder: fra den stiger op, fortsætter og klinger af igen. Ved *tone-envelopen* (ENT) forstås som de ændringer en lyds frekvens kan have, – i musikken er det oftest vibrato, der styres af ENT. Envelope-kommandoerne står lige før datalinierne, for at minde om at ENV og ENT i høj grad er med til at bestemme hvordan melodien lyder.

Vi går ikke i dybden med kommandoerne her. En ordentlig gennemgang ville fylde mange sider, og du lærer sikkert også mere af at eksperimentere med de to kommandoer, når programmet er færdigt. Som du kan se bruges ENT ikke i vores udgave af 'Für Elise'. Se i øvrigt gennemgangen af lydsystemet i manualen.

Modul 3.3.3: Linie 25000-25040

```
25ØØØ REM ****************************
25Ø1Ø REM ENV og ENT kommandoer her
25Ø2Ø REM ****************************
25Ø3Ø ENV 1,7,-1,1Ø,8,-1,4Ø
25Ø4Ø RETURN
```

Modul 3.3.4: Behandling af melodiens data.

På dette punkt er der nok nogen, der undrer sig over at vi ikke kan afspille endnu. Men det varer faktisk et stykke tid endnu inden vi kommer så langt, – først skal vi have 'oversat' melodiens data.

Grunden til at en oversættelse er nødvendig er, at vi har prøvet at gøre indlæsning og ændring af data så enkel som mulig, uden at skele til hvilken formattering af melodiens lydsystemets kommandoer kræver.

Selvom Amstrads BASIC er hurtig, kan den dog ikke klare både at formattere dataposterne for de tre stemmer og afspille dem samtidig. Hvis vi alligevel valgte at gøre det hele samtidig, ville melodien blive spillet for langsomt.

Vi gør derfor det, at vi først laver oversættelsen ved at lagre parametrene for hver tone i arrayet S% og derefter afspiller melodien på grundlag af S%.

Modul 3.3.4: Linie 15000-15140

```
15ØØØ REM ****************************
15Ø1Ø REM Databehandling
15Ø2Ø REM ****************************
15Ø3Ø FOR c=Ø TO 2
15Ø4Ø l=1:v=12:o=Ø:ev=Ø:et=Ø:p=Ø:g=1
15Ø5Ø n=Ø
15Ø6Ø WHILE a$(p,c)<>"end"
15Ø7Ø FOR i=1 TO 8
15Ø8Ø IF LEFT$(a$(p,c),1)<>MID$("ovl*<>e
p",i,1) THEN NEXT i
15Ø9Ø ON i GOSUB 16ØØØ,17ØØØ,18ØØØ,19ØØØ
,2ØØØØ,21ØØØ,22ØØØ,23ØØØ,24ØØØ
151ØØ p=p+1
1511Ø WEND
1512Ø d(c)=n
1513Ø NEXT c
1514Ø RETURN
```

Kommentar

Linie 15030-15130: Oversættelsen sker for alle tre stemmer (eller kanaler).

Linie 15040: Variablerne i denne linie bruges til følgende formål:

L – tonelængde
V – volume
O – oktav
EV – volume-envelope
ET – tone-envelope
P – placering i rækken af kommandoer for en kanal
G – gentagelse
N – antallet af spillede toner

Linie 15070-15090: Teknikken er den samme som blev brugt i forbindelse med MENU$ i karakterer. Her sammenligner løkken kommandoen i A% med det første bogstav i de mulige kommandoer i strengen i linie 15080. Hvis kommandoens første bogstav passer med en af bogstaverne i strengen, registrerer løkkevariablen I placeringen af kommandoen, hvorefter denne værdi bruges i ON...GOSUB. Virkningen af de enkelte kommandoer vil blive behandlet i forbindelse med de moduler, hvor de står. Hvis første bogstav i kommandoen *ikke* står i strengen, bliver kommandoen opfattet som en toneværdi.

Linie 15120: Antallet af toner i hver kanal lagres i arrayet D.

Test

Den eneste test, der kan laves nu er at skrive RETURN ved alle linienumrene i linie 15090. Programmet kan så køres og du kan kontrollere at der ikke er nogen syntax fejl. Disse RETURN linier vil gøre de muligt at køre programmet hver gang du har indlæst et af de efterfølgende moduler, som bearbejder melodiens data.

Modul 3.3.5: Tonerne oversættes

Hovedparten af melodien udgøres naturligvis af toner, og dette modul oversætter tonernes data i datalinierne og gør dem forståelige for programmet.

Tonerne i programmet indlæses på grundlag af 12-tonesystemet. En oktav udgøres i den vestlige musik af otte heltoner og fire halvtoner – 12 toner i alt, og dvs. at enhver tone i en oktav kan repræsenteres af et tal mellem 0 og 11. Da forholdet mellem tonerne i en oktav – og mellem tonerne i forskellige oktaver – kan udtrykkes rent matematisk, kan vi godt

bruge tal, der er større end 11. 12 bliver således første tone i den næste oktav, 24 bliver den første tone i den næste oktav igen osv. Oversættelsen af disse tal til tal som maskinen forstår, klares i dette modul.

Modul 3.3.5: Linie 24000-24070

```
24ØØØ REM ****************************
24Ø1Ø REM Tone
24Ø2Ø REM ****************************
24Ø3Ø fr=44Ø*(2^(o+(VAL(a$(p,c))-9)/12))
24Ø4Ø tp=ROUND(125ØØØ/fr)
24Ø5Ø s%(c,n,Ø)=tp:s%(c,n,1)=l:s%(c,n,2)
=v:s%(c,n,3)=ev:s%(c,n,4)=et
24Ø6Ø n=n+1
24Ø7Ø RETURN
```

Kommentar

Linie 24030: Denne formel oversætter de tal der repræsenterer de forskellige toner (under hensyn til tallet der bestemmer oktaven O, som vi vender tilbage til) til en frekvens.

Linie 24040: Når vi har fået et bestemt frekvens oversætter denne linie tallet videre til et tal, der kan indgå i SOUND kommandoen. Det er nemlig sådan, at Amstrad skal have omsat frekvenser efter formlen 125000/FREKVENS, for at kunne behandle dem.

Linie 24050: Når der står en talværdi i melodiens datalinier bliver det opfattet som en kommando om at afspille en tone. Og således bliver de forskellige poster, der skal bruges i SOUND kommandoen, flyttet fra de variabler, hvor de er lagret, til en linie i arrayet S%, som melodien til slut spilles på grundlag af.

Modul 3.3.6: Ændring af oktaven

Som du lige har set så indgår oktaven (O) i formlen, der omsætter tonernes tal-værdier til frekvenser. Det giver os flere muligheder, når vi indlæser melodiens data. Den første tone i den anden oktav kan således både indlæses som '12' eller som '0' hvor "O" får værdien '1'. Oktaven ændres ved at skrive et O efterfulgt af et tal.

Modul 3.3.6: Linie 16000-16040

```
16000 REM ****************************
16010 REM Oktav
16020 REM ****************************
16030 o=VAL(MID$(a$(p,c),2))
16040 RETURN
```

Modul 3.3.7: Volume

Volumen angives ved at skrive et V efterfulgt af et tal i melodiens data. Tallet skal være gyldigt som volume-angivelse (se evt. i manualen).

Modul 3.3.7: Linie 17000-17040

```
17000 REM ****************************
17010 REM Volume
17020 REM ****************************
17030 v=VAL(MID$(a$(p,c),2))
17040 RETURN
```

Modul 3.3.8: Tonelængden

Tonelængden angives ved at skrive L efterfulgt af et tal i melodiens data. Tonelængden fås ved at gange tallet med EL – enhedslængden.

Modul 3.3.8: Linie 18000-18040

```
18000 REM ****************************
18010 REM Længde
18020 REM ****************************
18030 l=VAL(MID$(a$(p,c),2))
18040 RETURN
```

Modul 3.3.9: Gentagelse af et melodiafsnit

De fleste melodier indeholder gentagelser, og en melodi uden gentagelser kan være ret anstrengende at lytte til. Nogle gange gælder gentagelsen større afsnit af melodien, andre gange er det kun småbidder der gentages. For at spare plads og indtastningsarbejde, har vi derfor indbygget en

funktion i programmet, der kan gentage et melodiafsnit, som skal afspilles flere gange i træk. Det angives med en stjerne, hvor gentagelsen skal starte og en stjerne hvor gentagelsen slutter. Efter den afsluttende stjerne sættes et tal, der angiver hvor mange gange afsnittet skal afspilles.

Modul 3.3.9: Linie 19000-19090

```
19ØØØ REM ****************************
19Ø1Ø REM Gentag
19Ø2Ø REM ****************************
19Ø3Ø IF a$(p,c)="*" THEN g=1:RETURN
19Ø4Ø IF VAL(MID$(a$(p,c),2))=g THEN RET
URN
19Ø5Ø g=g+1
19Ø6Ø WHILE a$(p,c)<>"*"
19Ø7Ø p=p-1:IF p=-1 THEN RETURN
19Ø8Ø WEND
19Ø9Ø RETURN
```

Kommentar

Linie 19030: For at køre dette modul skal datalinien *starte* med en stjerne. Hvis stjernerne står alene, bliver variablen G, der viser hvor mange gange afsnittet er blevet gentaget, sat til 1. Derefter vendes tilbage til programmet, således at oversættelsen af melodien kan fortsætte.

Linie 19040: Hvis en stjerne efterfølges af et tal, betyder det afslutningen på et afsnit. Størrelsen af variablen G, der viser hvor mange gange afsnittet er gentaget, bliver sammenlignet med tallet efter stjernen, som viser hvor mange gange afsnittet *skal* gentages. Hvis tallene er lige store, kører programmet videre uden yderligere gentagelser.

Linie 19050-19080: Antallet af gentagelser opskrives og programmet springer tilbage til starten af det afsnit, der skal gentages, før der vendes tilbage til hovedprogrammet.

Modul 3.3.10: Ændringer i gentagelserne

Ofte laves der små ændringer i de afsnit der gentages – den anden gentagelse kan f.eks. afsluttes med en række høje toner, mens den første gentagelse blev afsluttet med nogle lavere toner. Dette modul giver brugeren

mulighed for at tage højde for dette, ved at et delafsnit kun spilles ved én bestemt gentagelse. F.eks. kan et afsnit, der gentages, have en slutning ved første gentagelse, mens den anden gentagelse afsluttes med en anden slutning. I datalinierne mærkes sådanne delafsnit med et < efterfulgt af et tal, der angiver, ved hvilken gentagelse delafsnittet skal afspilles. Afslutningen på et delafsnit afmærkes med et enkelt >.

Modul 3.3.10: Linie 20000-21030

```
2ØØØØ REM ****************************
2ØØ1Ø REM Ved n'te gentagelse..
2ØØ2Ø REM ****************************
2ØØ3Ø IF VAL(MID$(a$(p,c),2))=g THEN RET
URN
2ØØ4Ø WHILE a$(p,c)<>">" AND a$(p+1,c)<>
"end"
2ØØ5Ø p=p+1
2ØØ6Ø WEND
2ØØ7Ø RETURN
21ØØØ REM ****************************
21Ø1Ø REM Afslut..
21Ø2Ø REM ****************************
21Ø3Ø RETURN
```

Kommentar

Linie 20030: For at dette modul skal køres skal programmet have mødt et < i datalinierne. Denne linie undersøger det tal, der står efter <. Hvis tallet er det samme som størrelsen af variablen G, som registrerer hvor mange gange afsnittet er blevet gentaget, fortsætter programmet med de toner, der kommer efter >.

Linie 20040-20060: Denne løkke bruges hvis delafsnittet ikke afspilles ved denne gentagelse, og den finder simpelthen det næste >.

Linie 21030: Når programmet møder et >, sørger denne linie ganske enkelt for, at kørslen fortsætter.

Modul 3.3.11: Ændringer i envelopes

Vi har allerede lavet en envelope, men Amstrad'en kan klare op til 16 envelopes for volume og tone. Hvis ENV eller ENT efterfølges af et gyl-

digt tal (uden mellemrum) for en envelope, som du har defineret i 25000, ændres den envelope, som SOUND kommandoen anvender.

Modul 3.3.11: Linie 22000-22050

```
22ØØØ REM ****************************
22Ø1Ø REM Envelopes
22Ø2Ø REM ****************************
22Ø3Ø IF LEFT$(a$(p,c),3)="env" THEN ev=
VAL(MID$(a$(p,c),4))
22Ø4Ø IF LEFT$(a$(p,c),3)="ent" THEN et=
VAL(MID$(a$(p,c),4))
22Ø5Ø RETURN
```

Modul 3.3.12: Pauser

Det må ikke glemmes at pauserne er en væsentlig del af en melodi, så derfor må vi kunne stoppe lyden, når det ønskes. Det sker ved at indsætte et P i melodiens data. Et enligt P laver en pause på én enhed, mens P efterfulgt af et tal laver en pause svarende til tallet.

Modul 3.3.12: Linie 23000-23060

```
23ØØØ REM ****************************
23Ø1Ø REM Pause
23Ø2Ø REM ****************************
23Ø3Ø IF a$(p,c)="p" THEN du=1 ELSE du=V
AL(MID$(a$(p,c),2))*1
23Ø4Ø s%(c,n,Ø)=Ø:s%(c,n,1)=du:s%(c,n,2)
=Ø:s%(c,n,3)=Ø:s%(c,n,4)=Ø
23Ø5Ø n=n+1
23Ø6Ø RETURN
```

Test

Hvis du ikke har testet de enkelte moduler, så kør programmet nu. Modulerne vil blive kaldt, når melodiens data analyseres.

Modul 3.3.13: Programmenuen

Denne lille menu kan kaldes de resterende programdele.

Modul 3.3.13: Linie 11000-11140

```
11ØØØ REM ****************************
11Ø1Ø REM Kontrol
11Ø2Ø REM ****************************
11Ø3Ø WHILE o<>3
11Ø4Ø CLS:PRINT TAB(18);"MUSIK";TAB(18);
"====="
11Ø5Ø PRINT:PRINT"MENU:":PRINT
11Ø6Ø PRINT "1) Ændring af parametre"
11Ø7Ø PRINT "2) Spil melodien"
11Ø8Ø PRINT "3) END"
11Ø9Ø PRINT:INPUT "Indtast dit valg: ",o
111ØØ ON o GOSUB 12ØØØ,13ØØØ
1111Ø WEND
1112Ø CLS
1113Ø LIST 25ØØØ-
1114Ø END
```

Modul 3.3.14: Ændring af melodiens parametre

Når en melodi skal afspilles kan forskellige styrende parametre ændres, – det sker med dette modul.

Modul 3.3.14: Linie 12000-12110

```
12ØØØ REM ****************************
12Ø1Ø REM Ændring af parametre
12Ø2Ø REM ****************************
12Ø3Ø CLS:PRINT TAB(12);"ÆNDRING AF PARA
METRE";TAB(12);"====================":PR
INT
12Ø4Ø PEN 2:PRINT "ENTER";:PEN 1:PRINT "
 for at bevare nuværende parametre":PRIN
T
12Ø5Ø PRINT "Relativ oktav (";rok;:INPUT
 ") :",x$:IF x$<>"" THEN rok=VAL(x$)
12Ø6Ø PRINT "Relativ tone (";rto;:INPUT
") :",x$:IF x$<>"" THEN rto=VAL(x$)
12Ø7Ø PRINT "Enhedslængde (";el;:INPUT "
) :",x$:IF x$<>"" THEN el=VAL(x$)
```

```
12Ø8Ø FOR c=Ø TO 2
12Ø9Ø PRINT "Afspil kanal";c;"(";p$(c);:
INPUT ") :",x$:IF x$<>"" THEN p$(c)=UPPE
R$(x$)
121ØØ NEXT c
1211Ø RETURN
```

Kommentar

Linie 12050: Den relative oktav kan ændres, – derved kan melodien sættes en eller flere oktaver op eller ned.

Linie 12060: Den relative tone kan også ændres, – på den måde kan tonearten ændres.

Linie 12070: Enhedslængden kan ændres her. Højere tal sætter tempoet ned, og lavere tal sætter tempoet op.

Linie 12080-12100: Det er ikke nødvendigt at afspille alle tre kanaler (stemmer), med denne løkke kan du bestemme hvilke der skal afspilles.

Test

Kør programmet. Efter en pause på ca. et minut skal menuen komme frem. Vælger du '1' kan du ændre parameterværdierne.

Modul 3.3.15: Kontrol med afspilningen

Nu kommer vi til de to sidste moduler, der gør det muligt at afspille melodien. Første modul er et kontrolmodul, som styrer afspilningen af melodien og sørger for afslutningen, og det sidste modul klarer afspilningen af tonerne i den rigtige rækkefølge.

Modul 3.3.15: Linie 13000-13160

```
13ØØØ REM ****************************
13Ø1Ø REM Afspil melodi
13Ø2Ø REM ****************************
13Ø3Ø rtk=2^-(rok+rto/12)
13Ø4Ø FOR c=Ø TO 2
13Ø5Ø p(c)=Ø
13Ø6Ø NEXT c
```

```
13Ø7Ø SOUND 199,Ø,Ø
13Ø8Ø GOSUB 14ØØØ
13Ø9Ø GOSUB 14ØØØ
131ØØ GOSUB 14ØØØ
1311Ø RELEASE 7
1312Ø ok=Ø
1313Ø WHILE NOT ok
1314Ø GOSUB 14ØØØ
1315Ø WEND
1316Ø RETURN
```

Kommentar

Linie 13030: Tonen som melodien spilles ud fra.

Linie 13040-13060: Tællerarrayet P, for de tre kanaler sættes til 0.

Linie 13070: Denne kommando laver ikke en tone, men sørger for følgende:

1) Sletter evt. toner som kunne stå i kø fra en tidligere afspilning.
2) Beordrer at kommandoer skal sendes til alle tre kanaler.
3) Sikrer at ingen toner afspilles.

Tallet 199 betyder i sig selv ikke noget: det er udelukkende en måde at sætte bestemte binære cifre (eller bits) til 1. Med 199 sættes 0, 1, 2, 6 og 7. Se i øvrigt i manualens kapitel 6, hvor du kan se kvilken effekt disse bits har.

Linie 13080-13110: Disse tre linier kalder næste modul, således at der laves lydkøer – rækker af ventende toner. Det betyder at små forsinkelser, som kan opstå når løkken afspiller melodien, ikke får nogen betydning for hvordan melodien lyder. RELEASE kommandoen i linie 13110 sætter alle tre kanaler i gang, således at den første tone i hver kø lyder samtidigt. Denne kø-funktion er noget af det, der gør Amstrad så enestående god til musik. På næsten alle andre hjemmecomputere, er det meget, meget vanskeligt at sikre den rigtige *timing,* fordi der ikke er nogen stødpude, der kan 'fange' eventuelle forsinkelser. På Amstrad er problemet løst, og programmeringen er nem.

Linie Linie 13120-13140: Resten af afspilningen klares simpelthen ved til stadighed at kalde det næste modul, indtil variablen OK, som sættes af næste modul, fortæller at melodien er slut.

Modul 3.3.16: Linie 14000-14120

Det sidste modul afspiller tonerne i de tre kanaler.

Modul 3.3.16: Linie 14000-14120

```
14ØØØ REM ****************************
14Ø1Ø REM Check kanalerne
14Ø2Ø REM ****************************
14Ø3Ø ok=-1
14Ø4Ø FOR c=Ø TO 2
14Ø5Ø IF p(c)=d(c) OR p$(c)="N" THEN GOT
O 1411Ø
14Ø6Ø ok=Ø
14Ø7Ø IF (SQ(2^c) AND 7)=Ø THEN GOTO 141
1Ø
14Ø8Ø n=p(c)
14Ø9Ø SOUND 2^c,s%(c,n,Ø)*rtk,s%(c,n,1)*
el,s%(c,n,2),s%(c,n,3),s%(c,n,4)
141ØØ p(c)=p(c)+1
1411Ø NEXT c
1412Ø RETURN
```

Kommentar

Linie 14040-14050 og 14110: Uanset om alle kanalerne skal afspilles, så afsøger denne løkke dem alle tre. Linie 14050 checker så, om der stadig er data tilbage i de forskellige kanaler, eller om brugeren eventuelt har ændret på parametrene, sådan at en eller flere kanaler ikke skal afspilles. hvis der er data tilbage i blot én af kanalerne sættes OK til 0.

Linie 14070: SQ funktionen undersøger om der er plads til nye toner i lydkøerne.

Linie 14080-14100: Hvis der *er* plads til en ny tone i køen bliver det næste sæt dataposter fra S% brugt i SOUND kommandoen – bemærk at tonen ikke lyder med det samme, den bliver blot sat ind i køen af ventende toner. Endelig bliver tælleren for den pågældende kanal opskrevet.

Test

Nu kan du endelig lave en rigtig test af programmet. Kør programmet og vælg '2' på menuen – og du vil finde ud af, om det virker, som det skal.

KAPITEL 4

Mere seriøse programmer

Nu er vi kommet dertil, hvor du efterhånden har lært din Amstrad helt godt at kende. Det er derfor på tide at gå i gang med nogle programmer, der sætter maskinen i stand til at arbejde med det, som microcomputerne er bedst til – ordne, behandle og genfinde informationer for brugeren.

I dette kapitel finder du programmerne:

DATAFIL: Et meget effektivt databaseprogram der kan tilpasses næsten alle anvendelser indenfor dette område.

NTAL: Et program, der kan bruges til næsten alt, hvad der har med tal-opstillinger at gøre. Du kan opstille regninger, lave en oversigt over aktiekurser eller holde styr på antallet af kalorier i kostplanen.

TEKST: Et lille tekstbehandlingsprogram, der kører i BASIC.

QUIZ: Et program, der kan lave såkaldte *multiple-choise* opgaver.

Program 4.1: Datafil

DATAFIL er en perle af et program, som er blevet udviklet gennem de seneste år i de forskellige udgaver af denne serie bøger. Folk, der har læst de tidligere bøger fortæller, at de bruger programmet i deres forretning; til at lære børn noget om hvordan datamater arbejder med information; til at hjælpe med arbejdet i mindre organisationer (sportsklubber mv.) eller til at holde styr på den hjemlige bog-, plade-, bånd- eller lysbilledsamling.

I dette program præsenteres:

1) Binær søgning.
2) Samling af data i fortløbende strenge.
3) Opstart af et program uden at programmets arrays slettes.

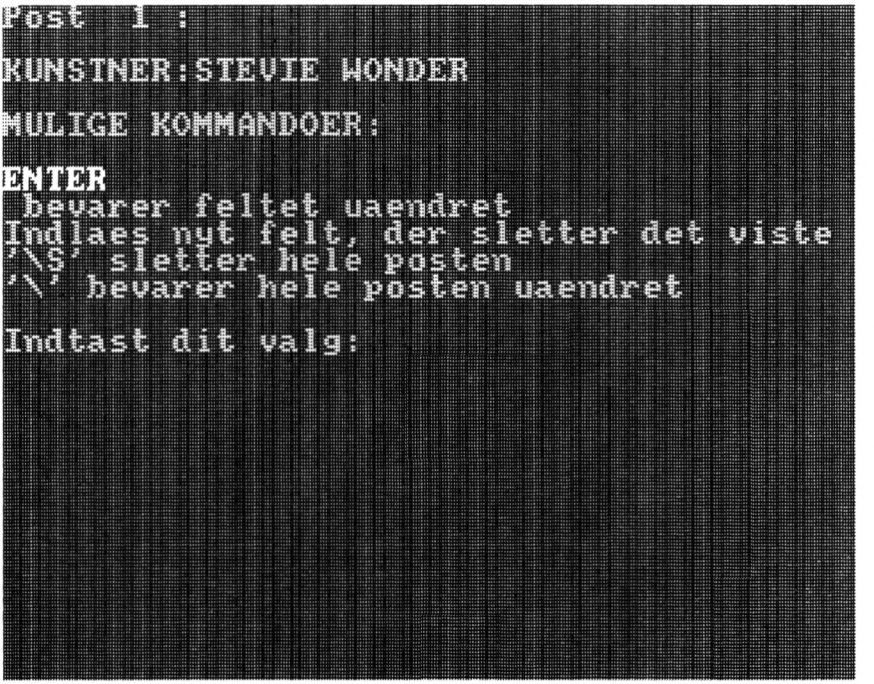


Fig. 4.1: Skærmdump fra DATAFIL.

Modul 4.1.1: Datafils struktur

Mange af de bøger, som henvender sig til ejere af hjemmecomputere, indeholder nogle ret dårlige databaseprogrammer, der ikke giver den flexibilitet, der er nødvendig, hvis man virkelig skal have glæde af programmet. F.eks. kan der være en fast struktur hvor rubrikkerne er Navn, Adresse, Telefonnummer el. lign. Det der gør DATAFIL så godt er, at vi frit kan vælge en struktur, uden at ændre i programmet. Vi kan have en eller flere rubrikker – præcis som vi ønsker. Jeg plejer at kalde DATAFIL for et 'kamæleon-program': et program der kan se ud på mange forskellige måder, alt efter hvad det skal anvendes til.

Inden vi går videre, må vi hellere lige få forklaret de begreber, der anvendes her. Datafilen kan sammenlignes med et kartotek, kartoteket indeholder en række kartotekskort, og i DATAFIL kaldes disse kort for *poster*. Hvert kort indeholder et vist antal rubrikker, stikord, punkter el. lign. – her kaldet *felter*.

Formålet med det første modul er dels at initialisere nogle variable, og dels at give brugeren mulighed for at bestemme hvilken struktur datafilen skal have.

Modul 4.1.1: Linie 20000-20110

```
20000 REM ****************************
20010 REM Opret datafil
20020 REM ****************************
20030 CLS:PRINT TAB(16);"NY DATAFIL":PRI
NT TAB(16);"==========="
20040 PRINT:INPUT"Er du sikker (j/n)";r$
:IF LEFT$(UPPER$(r$),1)="N" THEN RETURN
20050 CLEAR:DIM array$(1000)
20060 PRINT:INPUT"Hvor mange felter i hv
er post";x
20070 DIM felt$(x-1), ptr(x-1)
20080 PRINT:FOR i=0 TO x-1
20090 PRINT"Navn på felt #";i+1;:INPUT":
",felt$(i)
20095 felt$(i)=UPPER$(felt$(i))
20100 NEXT i
20110 in=-1:GOTO 11000
```

Kommentar:

Linie 20030-20040: Når en ny fil oprettes, slettes de data som filen måtte indeholde, derfor får brugeren mulighed for at 'fortryde' her.

Linie 20050: Den information, der skal ligge i DATAFIL bliver lagret i arrayet ARRAY$. I denne linie bliver arrayet dimensioneret til at indeholde 1001 poster. Du kan godt øge antallet til 2000, hvis du vil. Du skal blot være opmærksom på at hvert element i arrayet fylder tre bytes i hukommelsen *før* de anvendes. Hvis du dimensionerer arrayet til 5000 har du altså brugt 15000 bytes allerede før du begynder at indlæse data. Det er simpelthen et spørgsmål om at skønne sig frem – hvis du alligevel ikke får brug for mere end 1000 poster, kan du ligeså godt lade DIM sætningen være og dermed spare lidt plads i hukommelsen. Læg også mærke til at vi bruger et element i arrayet til at rumme hver post, så der er højest plads til 255 karakterer i hver post, skilletegnene medregnet, – svarende til den længste streng maskinen kan håndtere.

Linie 20060-20100: En del af hemmeligheden bag DATAFILENS flexibiblitet ligger her. For hver post kan du definere, hvor mange felter der skal være, og hvad disse felter skal hedde. Hvis det er pladesamling, du vil registrere, kan felterne f.eks. hedde: MELODI, ALBUM, KOMPONIST, KUNSTNER, LÆNGDE og ÅRSTAL. I det tilfælde skal du have 6 felter, som derefter kan tildeles de nævnte navne. Når du senere hen vil indlæse data i filen, kommer det til at ske under disse navne. De variable, der defineres her, bliver forklaret i de efterfølgende kommentarer i programmet.

Modul 4.1.2: Programmet startes

Det er ikke hver gang vi kører programmet, at vi ønsker at oprette en ny fil. Oftest vil vi i stedet hente data fra bånd. I dette startmodul kan brugeren vælge en af mulighederne.

Modul 4.1.2: Linie 10000-10150

```
10000 REM ****************************
10010 REM Start
10020 REM ****************************
10030 WHILE NOT in
10040 MODE 1
10050 PRINT TAB(16);"DATAFIL"
10060 PRINT TAB(16);"========"
10070 PRINT
10080 PRINT"MULIGE KOMMANDOER:"
10090 PRINT
10100 PRINT"1) Opret ny datafil"
10110 PRINT"2) Hent datafil fra bånd"
10120 PRINT
10130 INPUT"Indtast valg: ",z
10140 ON z GOSUB 20000,21000
10150 WEND
```

Kommentar

Linie 10030 og 10150: Disse to linier indeholder en teknik, som du uden tvivl vil få stor glæde af i dine egne programmer. Når programmet initialiseres af det modul, vi lige har indlæst, blev variablen IN (som står for INitialiseret) sat lig -1. Grunden til det kan ses ud af denne løkke. Hvis

værdien af IN er -1, når programmet når til linie 10030, bliver løkken ikke kørt. Den logiske operator NOT, som står i forbindelse med WHILE i linie 11030, sikrer at løkken kun køres, hvis IN er 0. Det betyder at du kan stoppe programmet og starte det igen med GOTO 10000 (eller GOTO 1 hvis du bruger det lille SAVE-modul, som jeg anbefalede i det allerførste program i bogen), uden at de data, der ligger i hukommelsen, går tabt.

Test

Indtast en midlertidig linie:

11000 STOP

Kør nu programmet og vælg at oprette en ny fil. Lav en fire-fem felter og giv dem nogle navne. Derefter skal programmet stoppe. Skriv nu:

GOTO 10000

og programmet skal stoppe med det samme – det opdager nemlig at der *er* oprettet en fil, selvom den er tom, og det springer derfor startmodulet over.

Modul 4.1.3: Menuen

Dette er en standard menu, men læg mærke til at der ikke er givet mulighed for at afslutte programmet med ON ERROR og <ESC>. DATAFIL er et komplekst program, og hvis det afbrydes under kørslen, kan dets informationer blive forvansket, fordi en vigtig operation ikke var afsluttet. Når du skal afslutte programmet skal du derfor *altid* bruge punkt '6' på denne menu.

Modul 4.1.3: Linie 11000-11210

```
11ØØØ REM ****************************
11Ø1Ø REM Hovedmenu
11Ø2Ø REM ****************************
11Ø3Ø WHILE NOT ok
11Ø4Ø CLS
11Ø5Ø PRINT TAB(16);"DATAFIL"
11Ø6Ø PRINT TAB(16);"======="
11Ø7Ø PRINT
```

```
11Ø8Ø PRINT"MULIGE KOMMANDOER:"
11Ø9Ø PRINT
111ØØ PRINT"1) Opret ny datafil"
1111Ø PRINT"2) Hent datafil fra bånd"
1112Ø PRINT"3) Indlæs information"
1113Ø PRINT"4) Søg/Vis/Ret"
1114Ø PRINT"5) Gem datafil på bånd"
1115Ø PRINT"6) Stop"
1116Ø PRINT
1117Ø INPUT"Indtast dit valg: ",z
1118Ø ON z GOSUB 2ØØØØ,21ØØØ,12ØØØ,17ØØØ
,18ØØØ,22ØØØ
1119Ø WEND
112ØØ ok=Ø
1121Ø END
```

Modul 4.1.4: Programmet stoppes

Dette lille modul laver en udskrift, der fortæller at programmet er stoppet, og variablen OK sættes til -1, for at fortælle menuen at programmet skal stoppe.

Modul 4.1.4: Linie 22000-22060

```
22ØØØ REM ******************************
22Ø1Ø REM Stop
22Ø2Ø REM ******************************
22Ø3Ø ok=-1
22Ø4Ø CLS:LOCATE 11,13
22Ø5Ø PRINT"DATAFILEN ER LUKKET":PRINT:P
RINT
22Ø6Ø RETURN
```

Test

Nu skal du kunne skrive

GOTO 11000 < ENTER >

og se menuen på skærmen. Vælg nu '6' og programmet skal stoppe.

Modul 4.1.5: Gemning af data på bånd

Når du begynder at arbejde med mere komplekse programmer, enten fra denne bog eller på egen hånd, vil du opdage at det er en stor fordel at indlæse et modul, som dette, hurtigst muligt. Grunden til det er, at det kun er muligt at lave nogle ordentlige tests af et program som DATAFIL, når der er indlæst en vis portion data. Da du altid laver nogle fejl, kan disse data blive forvansket, og du kan komme til at indlæse de samme data igen og igen. Løsningen på dette problem ligger naturligvis i at gemme disse data på bånd så tidligt som muligt, således at du kan hente dine data på båndet, når du har brug for dem.

Dette modul giver mulighed for at gemme data, og det giver også mulighed for noget vi kun har nævnt indtil nu, men ikke brugt, – at give programmet mulighed for at opsamle eller lagre filer med forskellige navne.

Modul 4.1.5: Linie 18000-18080

```
18ØØØ REM ****************************
18Ø1Ø REM Gem datafilen
18Ø2Ø REM ****************************
18Ø3Ø CLS:PRINT TAB(16);"GEM DATAFILEN":
PRINT TAB(16);"=============="
18Ø4Ø PRINT:INPUT"Gem datafilen under na
vnet: ",fi$
18Ø5Ø PRINT:OPENOUT fi$:PRINT#9,ft:PRINT
#9,x
18Ø6Ø FOR i=Ø TO ft-1:PRINT#9,array$(i):
NEXT i
18Ø7Ø FOR i=Ø TO x-1:PRINT#9,felt$(i):NE
XT i
18Ø8Ø CLOSEOUT:RETURN
```

Kommentar

Linie 18030-18040: Her kan filens navn indlæses.

Linie 18050-18070: Antallet af poster i DATAFIL huskes løbende af variablen FT, mens X husker antallet af felter i hver post. ARRAY$ rummer alle data, mens FELT$ rummer navnene på felterne.

Modul 4.1.6: Data hentes fra bånd

Når vores data ligger på båndet, skal vi jo også kunne hente dem igen. Og det er ikke så enkelt, for mens alle arrays allerede var dimensioneret i forhold til de data, der lå i hukommelsen, da filen blev gemt, så skal alle arrays dimensioneres forfra, når filen hentes fra bånd. Det er grunden til at variablerne FT og X blev placeret først i filen, så de kan blive indlæst i hukommelsen først, og dermed kan bruges til at dimensionere de forskellige arrays, ligesom i første modul, da de blev indlæst af brugeren og ikke af dataoptageren.

Modul 4.1.6: Linie 21000-21140

```
21ØØØ REM ****************************
21Ø1Ø REM Hent datafil
21Ø2Ø REM ****************************
21Ø3Ø CLS:PRINT TAB(16);"HENT DATAFIL":P
RINT TAB(16);"============"
21Ø4Ø PRINT:INPUT"Er du sikker (j/n)";r$
:IF LEFT$(UPPER$(r$),1)="N" THEN RETURN
21Ø5Ø CLEAR:DIM array$(1ØØØ)
21Ø6Ø PRINT:INPUT"Datafilen hedder: ",fi
$
21Ø7Ø PRINT:OPENIN fi$:INPUT #9,ft,x:DIM
 felt$(x-1),ptr(x-1)
21Ø8Ø FOR i=Ø TO ft-1
21Ø9Ø INPUT #9,array$(i)
211ØØ NEXT i
2111Ø FOR i=Ø TO x-1
2112Ø INPUT #9,felt$(i)
2113Ø NEXT i
2114Ø in=-1:CLOSEIN:GOTO 11ØØØ
```

Kommentar

Linie 21140: Der er nok nogen, der undrer sig over at finde et GOTO her i slutningen af en subrutine. Men GOTO kan ikke erstattes af et RETURN, idet hukommelsen blev slettet i dette modul, for at vi kunne dimensionere arrays igen. Når hukommelsen slettes, bliver oplysningerne om den oprindelige GOSUB kommando også slettet, og RETURN ville derfor give en fejlmelding.

Test

Kør programmet og vælg at oprette en ny fil. Lav fire felter med navnene EN, TO, TRE og FIRE. Når menuen kommer frem vælger du '5' og et filnavn som f.eks. FILDATA. Husk at have det rigtige bånd i maskinen før du starter optagelsen. Når menuen kommer frem igen stopper du programmet med '6' og kører programmet en gang til. Men denne gang vælger du naturligvis at hente filen fra båndet. Brug det samme navn igen, spol båndet tilbage og hent filen ind i maskinen igen. Når menuen kommer frem stopper du igen programmet. Skriv nu:

FOR I=0 TO X-1:PRINT FELT$(I):NEXT <ENTER>

og du skal få navnene på de fire felter udskrevet.

Modul 4.1.7: En hurtigere måde at søge på

I dette og de to følgende moduler, skal vi se på hvordan en ny post bliver føjet til hovedfilen, som ligger i ARRAY$. Selve kernen i metoden ligger imidlertid her, for det er dette modul, der gør det muligt for DATAFIL at søge hurtigt gennem en lang række af poster, for at finde det rigtige sted at indsætte den nye post, eller hurtigt at finde et søgeord i filen.

Denne metode kaldes *binær søgning,* og den kan nedsætte søge-tiden ret betydeligt, når der søges i en lang række ordnede data. Prøv at se på dette eksempel.

Vi har en fil med 2000 alfabetisk ordnede navne. Vores opgave er nu at indsætte et nyt navn i filen på det rigtige sted. Hvis vi snyder lidt og ser i navnelisten, kan vi se at det nye navn 'POULSEN' skal ind i filen som nummer 1731. Men det kan computeren naturligvis ikke vide på forhånd.

Nu kunne vi sætte maskinen i gang med at gennemgå filen fra en ende af. Den starter med 'AGERSKOV', registrerer at 'POULSEN' skal komme efter det, så kommer 'ALBERTSEN' osv. Når den så kommer til 'POVLSEN', som skal komme *efter* 'POULSEN', er den rigtige alfabetiske placering fundet.

Dette er en pålidelig metode, – men det ville jo være godt, hvis vi kunne begrænse antallet af sammenligninger. Og det kan vi: Hvis vores fil indeholder 2000 navne kan vi faktisk nøjes med kun *10 sammenligninger!* Her kommer forklaringen.

Computeren begynder med at undersøge placering 1024, fordi 1024 er

den største to-tals potent (2^10), der ikke overstiger det totale antal poster (2000). Det navn der findes på placering 1024 er alfabetisk mindre end 'POULSEN', og computeren lægger derfor 1024/2 – eller 512 – til den oprindelige søgeplacering, og det giver 1536. Også denne gang er navnet alfabetisk mindre end 'POULSEN', så denne gang lægges 256 – eller 2^8 – til 1536, hvilket giver 1792. Nu sker der noget andet, for navnet på denne placering ligger *efter* 'POULSEN' i den alfabetiske rækkefølge. Derfor bliver 128 – eller 2^7 – *trukket fra* den seneste søgeplacering, og vi får nu 1664.

Sådan fortsætter søgningen med faldende potenser af to, og vi får et mønster, der ser således ud:

SAMMENLIGNING NR.	PLACERING	+/-
1	1024	+512
2	1536	+256
3	1792	-128
4	1664	+64
5	1728	+32
6	1760	-16
7	1744	-8
8	1736	-4
9	1732	-2
10	1730	+1

Prøv selv med forskellige antal poster og forskellige tal – du vil opdage at systemet altid virker.

Modul 4.1.7: Linie 13000-13110

```
13ØØØ REM ****************************
13Ø1Ø REM Binær søgning
13Ø2Ø REM ****************************
13Ø3Ø IF ft=Ø THEN ss=Ø:RETURN
13Ø4Ø po=INT(LOG(ft)/LOG(2)):ss=2^po-1
13Ø5Ø FOR i=po-1 TO Ø STEP -1
13Ø6Ø ss=ss+2^i*((array$(ss)>te$)-(array
$(ss)<te$))
13Ø7Ø IF ss<Ø THEN ss=Ø
13Ø8Ø IF ss>ft-1 THEN ss=ft-1
```

```
13Ø9Ø NEXT i
131ØØ IF array$(ss)<te$ THEN ss=ss+1
1311Ø RETURN
```

Kommentar

Linie 13030: Hvis der ikke er nogen poster i filen, vil udregningen give en fejlmelding. Denne linie forhindrer dette.

Linie 13040: Disse to udtryk finder frem til den højeste totals potens, der ligger indenfor antallet af poster i filen, og derefter sættes søge-pegepinden (SS) lig med det tal. '-1' i det andet udtryk tager højde for at arrayet starter med 0, ikke 1.

Linie 13050-13090: Denne løkke laver søgningen ved hjælp af faldende totals potenser. Hovedfilen ligger i ARRAY$ og den nye post ligger i TE$. Pegepindens størrelse bestemmes af de to logiske betingelser, som angiver om TE$ er større eller mindre end posten på ARRAY$(SS).

Linie 13100: I nogle tilfælde landes der på en placering, der er 1 lavere end den rigtige placering – i disse tilfælde lægges 1 til SS.

Modul 4.1.8: Indsæt en post

Dette modul indsætter nye poster på den placering, som variablen SS angiver. Det gøres simpelthen ved at flytte alt efter SS en plads længere op.

Modul 4.1.8: Linie 14000-14070

```
14ØØØ REM ******************************
14Ø1Ø REM Indsæt post
14Ø2Ø REM ******************************
14Ø3Ø IF ft=Ø THEN GOTO 14Ø7Ø
14Ø4Ø FOR i=ft TO ss+1 STEP -1
14Ø5Ø array$(i)=array$(i-1)
14Ø6Ø NEXT i
14Ø7Ø array$(ss)=te$:ft=ft+1:RETURN
```

Modul 4.1.9: Nye poster til filen

Efter vi har indlæst de moduler der klarer det 'hårde' arbejde, skal vi nu bruge et modul, som kan lade brugeren indlæse nye poster i filen. Modu-

let fungerer på den måde at brugeren bedes at indlæse de respektive felter i den rigtige rækkefølge. Felterne bindes sammen til en streng, som kan indsættes i en linie af ARRAY$. Derefter kaldes de to tidligere moduler, så posten kan blive indsat i hovedfilen.

Modul 4.1.9: Linie 12000-12320

```
12ØØØ REM ****************************
12Ø1Ø REM Nye poster
12Ø2Ø REM ****************************
12Ø3Ø WHILE ft<1ØØØ
12Ø4Ø te$=""
12Ø5Ø CLS
12Ø6Ø PRINT TAB(15);"NYE POSTER"
12Ø7Ø PRINT TAB(15);"==========="
12Ø8Ø PRINT
12Ø9Ø PRINT"Post nummer";ft+1
121ØØ PRINT
1211Ø PRINT"MULIGE KOMMANDOER:"
1212Ø PRINT
1213Ø PRINT"Indlæs data til feltet"
1214Ø PRINT"'|' for at vende tilbage til
 hovedmenuen"
1215Ø PRINT
1222Ø FOR i=Ø TO x-1
1223Ø PRINT felt$(i);:INPUT":",q$
1224Ø IF q$="|" THEN RETURN
1225Ø te$=te$+UPPER$(q$)+"|"
1226Ø NEXT i
1227Ø PRINT:PRINT"vent et øjeblik ..."
1228Ø GOSUB 13ØØØ:GOSUB 14ØØØ
1229Ø WEND
123ØØ CLS
1231Ø PRINT"*** IKKE PLADS TIL FLERE POS
TER ***"
1232Ø FOR i=1 TO 2ØØØ: NEXT i:RETURN
```

Kommentar

Linie 12030 og 12290-12320: Nye poster accepteres kun hvis der er dimensioneret plads til dem, ellers får brugeren besked om at der ikke er

plads. Bemærk at hvis du vil ændre dimensioneringen, skal du huske at ændre tallet i linie 12030 også. Du kunne indsætte en variabel her, hvis du ellers husker at definere denne variabel hver gang programmet initialiseres, og gemme den når filen gemmes.

Linie 12220-12260: Det er disse linier der beder om indlæsning af de enkelte felter i en ny post. Navnet for hvert felt tages fra FELT$, som blev oprettet i initialiseringsmodulet. Hver felt indlæses under navnet Q$, og indlæsningen af nye felter stoppes hvis Q$ er '|'. Hvis indlæsningen ikke er '|' bliver feltet tilføjet til TE$, som til slut vil indeholde hele posten, og endelig tilføjes '|' efter hvert felt. Formålet med at sætte '|' efter hvert felt har intet at gøre med anvendelsen af samme tegn til at afslutte indlæsningen, det er der udelukkende for at fortælle senere dele af programmet, hvor det ene felt ender og det næste begynder. En post som f.eks.

HANSEN
KARL
EDBVEJ 10
DATAKØBING

vil blive lagret i ARRAY$ i denne form:

HANSEN|KARL|EDBVEJ 10|DATAKØBING|

dette kaldes en 'pakket streng'. Grunden til at '|' er valgt er, at det er rimeligt let at undgå at bruge dette tegn i filens poster. Hvis du af en eller anden grund gerne vil bruge det kan du blot vælge et andet skilletegn. Bemærk at alle indlæste felter laves om til store bogstaver. Du vil senere se at det samme sker når du bruger søgeord i filen. Det forhindrer at der opstår problemer med at finde bestemte ord i filen, hvis brugen af store og små bogstaver ikke er konsekvent. Hvis du er nødt til at bruge små og store bogstaver sammen, kan du fjerne disse linier i programmet uden at det forstyrrer programmets funktion. Men vær opmærksom på at maskinen regner med at små bogstaver kommer senere i alfabetet end store bogstaver – og det kan få filen til at se ret besynderlig ud.

Test

Nu kan du teste de tre moduler, du lige har indlæst. Kør programmet og opret en fil med to felter – 'ET' og 'TO' – i hver post. Når det er gjort vælger du '3' på menuen. Nu bliver du bedt om at indlæse data til felt 'ET'. Indlæs 'AA1'. Tilsvarende indlæser du 'AA2' til felt 'TO'. Nu bliver du

bedt om at indlæse data til felt 'ET' i post 1 – gentag processen med indlæsningerne 'DD1', 'DD2' 'CC1', 'CC2', 'BB1', 'BB2'. Indtast til slut '|' og du vender tilbage til hovedmenuen. Stop nu programmet og skriv:

FOR I=0 TO 3:? ARRAY$(I):NEXT <ENTER>

og du skal få følgende udskrift på skærmen:

AA1|AA2|
BB1|BB2|
CC1|CC2|
DD1|DD2|

Hvis det hele fungerer, som det skal, kan du starte programmet igen med GOTO 10000, vælge '3' på menuen og gemme de indlæste data på bånd. Det vil gøre det mindre ensformigt at teste de følgende moduler.

Modul 4.1.10: Postens felter findes

Før vi kan gå videre til næste moduler, som gør det muligt at genfinde data i filen og bearbejde dem, skal vi have indlæst dette modul, hvis funktion det er at gennemsøge en post og registrere hvor '|' står, – eller med andre ord at finde ud af hvor de enkelte felter slutter. Placeringen bliver lagt ind i arrayet PTR, som kun rummer oplysninger om den post, der undersøges. Når en ny post skal undersøges, kaldes dette modul igen.

Modul 4.1.10: Linie 19000-19080

```
19000 REM ******************************
19010 REM Analyser posten
19020 REM ******************************
19030 pp=0
19040 FOR i=0 TO x-1
19050 ptr(i)=INSTR(pp+1,array$(s1),"|")
19060 pp=ptr(i)
19070 NEXT i
19080 RETURN
```

Kommentar

Linie 19050: I modul 4.1.7 (binær søgning) blev variablen SS brugt til at angive hvilken post, der blev undersøgt, så bruger det øvrige program va-

riablen S1 til at registrere, hvor den aktuelle post er placeret i ARRAY$. Denne linie har den funktion at der søges efter '|', - der startes én karakter efter det sted hvor det forrige skilletegn blev fundet.

Modul 4.1.11: Søgning i filen

Vi kommer nu til det modul, der gør at programmet overhovedet kan bruges – det er nemlig her data kan findes frem igen. Modulet giver fire muligheder for at genfinde en post:

1) Ved at 'bladre' frem post for post fra den aktuelle placering.
2) Ved at springe et angivet antal poster frem eller tilbage.
3) Ved at indlæse et søgeord – det første felt i en post – for at lave en lynhurtig søgning.
4) Ved at søge efter enhver sammenhængende sekvens af karakterer i en post.

Modul 4.1.11: Linie 17000-17430

```
17ØØØ REM ****************************
17Ø1Ø REM Søgning
17Ø2Ø REM ****************************
17Ø3Ø s1=Ø:CLS:PRINT TAB(17);"SØGNING":P
RINT TAB(17);"=======":PRINT
17Ø4Ø IF ft=Ø THEN PRINT"*** ENDNU INGEN
 DATA INDLÆST ***" ELSE GOTO 17Ø7Ø
17Ø5Ø FOR i=1 TO 2ØØØ: NEXT i
17Ø6Ø RETURN
17Ø7Ø PRINT"MULIGE KOMMANDOER:"
17Ø8Ø PRINT:PRINT"Indlæs ord/tal til nor
mal søgning"
17Ø9Ø PRINT"Sæt '*' foran for initial-sø
gning"
171ØØ PEN 2:PRINT"ENTER";
1711Ø PEN 1:PRINT" for første post i dat
afilen"
1712Ø te$="":PRINT:INPUT"Indtast søgekom
mando: ",te$:te$=UPPER$(te$)
1713Ø IF LEFT$(te$,1)="*"THEN te$=MID$(t
e$,2):GOSUB 13ØØØ:te$="":s1=ss
1714Ø p$="F":WHILE p$="F"
```

```
17150 IF te$="" THEN GOTO 17210
17160 ff=0:FOR i=s1 TO ft-1
17170 pp=INSTR(array$(i),te$)
17180 IF pp<>0 THEN ff=1:s1=i:i=ft-1
17190 NEXT i
17200 IF ff=0 THEN RETURN
17210 p$=""
17220 WHILE p$="" AND ft>0
17230 IF s1>ft-1 THEN s1=ft-1
17240 IF s1<0 THEN s1=0
17250 GOSUB 19000
17260 CLS:PRINT"POST ";s1+1;":":PRINT:pp
=0
17270 FOR i=0 TO x-1
17280 PRINT felt$(i);":";MID$(array$(s1)
,pp+1,ptr(i)-pp-1)
17290 pp=ptr(i):NEXT i
17300 PRINT:PRINT"MULIGE KOMMANDOER:"
17310 PRINT:PEN 2:PRINT"ENTER";:PEN 1:PR
INT" for næste post"
17320 PRINT"'R' for at rette"
17330 PRINT"'F' for at fortsætte søgning
"
17340 PRINT"'#' og et tal for at springe
 i filen"
17350 PRINT"'|' for at vende tilbage til
 hovedmenuen"
17360 PRINT:INPUT"Indtast dit valg: ",p$
17370 p$=UPPER$(p$):IF p$="" OR p$="F" T
HEN s1=s1+1
17380 IF p$="F" THEN GOTO 17420
17390 IF p$="R" THEN GOSUB 15000:p$=""
17400 IF LEFT$(p$,1)="#" THEN s1=s1+VAL(
MID$(p$,2)):p$=""
17410 WEND
17420 WEND
17430 RETURN
```

Kommentar

Linie 17030: S1 peger, som nævnt i kommentaren til det forrige modul, på den aktuelle post, og den starter på 0 hver gang søgningen starter.

Linie 17040-17060: En fejlmelding udskrives, hvis der endnu ikke er nogen data i filen.

Linie 17070-17120: Dette er den første menu, du vil se, når du vælger '3' på hovedmenuen. Denne menu kommer kun frem når søgningen starter. Med denne menu kan du vælge hvilken type søgning, du ønsker at foretage – hvis du vil lave en anden type søgning, skal du afbryde søgningen og gå via hovedmenuen og denne menu igen. 'NORMAL SØGNING' vil sige søgning efter en bestemt sekvens af karakterer, ligegyldigt hvor de står i posten – filen gennemsøges fra begyndelsen, og den første post, der kommer frem, er den første post, hvor den ønskede karakter-sekvens findes. 'INITIAL SØGNING' betyder, at der søges en post, der har de karakterer brugeren angiver, som *de første* karakterer i posten. Dvs. at indlæsningen '*SMI' vil finde en post, der begynder med bogstaverne SMI, men ikke nødvendigvis den første post med disse begyndelsesbogstaver. Hvis der søges en streng, som ikke står først i nogen af posterne, så vil du i stedet få den post frem, som står dér, hvor den pågældende streng *ville have stået.*

Du kan søge på mere end et felt i både normal og initial søgning, ved at indsætte et '|' i den streng du søger på. Bruger du denne teknik i en fil, hvor første felt er et efternavn og andet felt er et fornavn, ville en initial søgning som f.eks. '*SMITH|A' give en SMITH med 'a' som *initial* (første bogstav) i fornavnet, men igen, ikke nødvendigvis den første.

Hvis du, i normal søgning, har angivet en streng, som slet ikke findes i filens poster, vender programmet tilbage til hovedmenuen.

Linie 17130: Denne linie gør alt arbejdet med 'initial søgningen', ved at fjerne den indledende '*' og derefter kalde modulet med binær søgning, for at finde den rigtige placering.

Linie 17140-17420: Hovedløkken gentager normal søgning, hvis brugeren ønsker det, i den af de efterfølgende menuer. Bemærk at initialsøgnings-rutinen ikke ligger indenfor denne løkke. Grunden er naturligvis, at der ikke er nogen pointe i at gentage denne – resultatet vil altid blive den samme post.

Linie 17150-17210: Når vi er nået hertil i kørslen af dette modul, skal den streng, vi leder efter, findes ved normal søgning. Det sker simpelthen ved at afsøge hver post med INSTR. Hvis strengen findes, bliver variablen FF sat til 1, for at angive dette. Placeringen registreres af S1, løkkevariablen I sættes til den højeste værdi, således at løkken slutter.

Linie 17220-17410: Denne løkke vil køre så længe som P$, indlæsningen til den næste søgemenu, er en tom streng. Grunden til at værdien af FT skal med i betingelsen for denne løkke er, at der inde i løkken gives mulighed for at slette felter. Hvis alle felterne slettes skal vi kunne komme ud af løkken, ellers vil der komme en fejlmelding.

Linie 17230-17240: Inde i løkken har brugeren mulighed for at springe rundt i filen, disse linier checker ved de efterfølgende gennemløb af løkken, at pegepinden S1 ikke kommer udenfor det gyldige område – det aktuelle antal poster.

Linie 17250-17290: Disse linier kalder det forrige modul og bruger oplysningerne om placeringen af skilletegnene til at udskrive de felter, der hører til den aktuelle post, sammen med felternes navne. For hvert gennemløb af FOR-løkken bliver de karakterer, der er placeret mellem værdien af variablen PP og den relevante værdi i arrayet PTR, udskrevet. Når et sæt karakterer er udskrevet, bliver PP sat til den aktuelle værdi i arrayet PTR og løkkevariablen, I, henter den næste værdi i PTR.

Linie 17300-17360: Denne søgemenu kommer frem når posten er blevet udskrevet. Brugeren har så mulighed for at gå videre til den næste post, og kalde 'rette'-funktionen (endnu ikke indlæst), at fortsætte søgningen efter den oprindelige streng, at springe et antal poster frem eller tilbage i filen, eller at vende tilbage til hovedmenuen.

Linie 17370-17380: Disse to linier hænger sammen med de to løkker, der startede i linie 17140 og 17220. Hvis der indlæses en tom streng (dvs. <ENTER>), så gentages løkken i linie 17220, og den næste post, angivet af S1, udskrives. Hvis der tastes 'F', fortsætter løkken i 17140 søgningen fra den næste post.

Linie 17390: Rette-funktionen, der endnu ikke er indlæst.

Linie 17400: Hvis der indtastes et tal efter et '#' tegn, kan brugeren springe frem eller tilbage i filen. Ved at sætte P$ lig en tom streng, sættes løkken i linie 17220 til at udskrive den post, som S1 peger på.

Test

Hvis du har gemt de fire poster, der blev lavet i de tidligere tests, så kør programmet og hent posterne fra båndet. Vælg '4' på hovedmenuen og

prøv at 'bladre' igennem med <ENTER>. Hver post skal blive udskrevet i to linier med de rigtige navne, f.eks.

ET: AA1
TO: AA2

Når du kommer til den fjerde post vil du opdage at du ikke kan komme videre med < ENTER >. Indtast nu #—1 og du skal komme tilbage til post nr. 3. Fortsæt med at springe baglæns i filen – og du vil opdage, at du heller ikke kan komme længere tilbage end post nr. 1.

Tast '↑' så du vender tilbage til hovedmenuen og vælg '4'. Denne gang skal du skrive 'CC' som søgeord. Nu kommer post 3 frem, som er den eneste, der inderholder 'CC'. Nu har du samtidig den anden søgemenu på skærmen, fortsæt søgningen med 'F'. Nu skal programmet vende tilbage til hovedmenuen.

Vælg igen '4' og søg på '2'. Nu kommer post 1 frem fordi det er den første post, der indeholder karakteren '2'. Tast 'F' og post 2 kommer frem, da den også indeholder '2'. Fortsæt med at taste 'F' indtil alle fire poster er vist, derefter giver forsat søgning negativt resultat, og hovedmenuen kommer frem igen.

Vælg nu til slut igen '4' på hovedmenuen og indtast '*B' som søgeord. Nu skal post 2 komme frem – den eneste post, der begynder med 'B'. Tast '↑' så du vender tilbage til hovedmenuen og afslut derefter programmet.

Nu er alle søgefunktionerne testet.

Modul 4.1.12: Sletning af poster

De sidste detaljer i programmet kommer med de sidste to moduler, som gør det muligt at slette eller rette posterne. Vi indlæser først slettemodulet, idet det også anvendes af rettemodulet.

Modul 4.1.12: Linie 16000-16040

```
16ØØØ REM ******************************
16Ø1Ø REM Slet post
16Ø2Ø REM ******************************
16Ø3Ø FOR j=s1 TO ft-1:array$(j)=array$(
j+1):NEXT j
16Ø4Ø ft=ft-1:RETURN
```

Kommentar

Linie 16030-16040: Sletningen sker simpelthen ved at begynde med posten, der ligger lige ovenfor den, der skal slettes og flytte den et trin ned. Derefter nedskrives tælleren FT.

Test

Kør programmet og hent de fire poster fra båndet. Vælg '6' og stop programmet. Skriv nu:

S1= 0 < ENTER >
GOTO 16000 <ENTER>

Nu skal programmet stoppe med 'Unexpected RETURN'. Skriv:

GOTO 11000 <ENTER>

og vælg søgning på hovedmenuen. Du vil nu se at posten, der indeholdt AA1|AA2 er forsvundet fra filen.

Modul 4.1.13: Rettelse af poster

Programmet ville kun have begrænset anvendelse, hvis vi ikke kunne rette og ændre de bestående data. Det klares med dette sidste modul.

Modul 4.1.13: Linie 15000-15190

```
15ØØØ REM ******************************
15Ø1Ø REM Rettelse af post
15Ø2Ø REM ******************************
15Ø3Ø te$="":pp=Ø
15Ø4Ø FOR i=Ø TO x-1
15Ø5Ø CLS:PRINT"Post ";s1+1;":"
15Ø6Ø PRINT:PRINT felt$(i);":"MID$(array
$(s1),pp+1,ptr(i)-pp-1)
15Ø7Ø PRINT:PRINT"MULIGE KOMMANDOER:"
15Ø8Ø PRINT:PEN 2:PRINT"ENTER"
15Ø9Ø PEN 1:PRINT" bevarer feltet uændre
t"
151ØØ PRINT"Indlæs nyt felt, der sletter
 det viste"
1511Ø PRINT"'|S' sletter hele posten"
```

```
15120 PRINT"'|' bevarer hele posten uænd
ret"
15130 PRINT:INPUT"Indtast dit valg: ",q$
15140 q$=UPPER$(q$):IF q$="|S" THEN GOSU
B 16000:RETURN
15150 IF q$="|" THEN RETURN
15160 IF q$<>"" THEN q$=q$+"|"
15170 IF q$="" THEN q$=MID$(array$(s1),p
p+1,ptr(i)-pp)
15180 pp=ptr(i):te$=te$+Q$:NEXT i:GOSUB
16000:GOSUB 13000
15190 s1=ss:GOSUB 14000:RETURN
```

Kommentar

Linie 15040-15180: Denne løkke ligner løkken i modul 4.1.8, som udskriver posterne, men denne udskriver kun et felt ad gangen.

Linie 15140: En indtastning af '|S' sletter *hele* posten, selvom det kun er et af postens felter, der vises. Bemærk at de enkelte felter ikke kan slettes, idet antallet af felter pr. post er fast.

Linie 15150: En indtastning af '|' som svar på en hvilken som helst post, returnerer udførelsen til søgemodulet. Alle ændringer, der er foretaget i tidligere felter i posten, vil blive ignoreret og posten vil forblive uændret.

Linie 15160: Hvis der indtastes andet end '|S' eller '|', så tolkes det som et nyt felt, der skal erstatte det viste. Skilletegnet '|' tilføjes til slut til erstatningsfeltet – ligesom når felterne indlæses første gang.

Linie 15170: Hvis der blot tastes <ENTER> bliver det viste felt blot kopieret uden ændringer. Hvis der kun skal ske ændringer med et af felterne, skal du blot taste <ENTER> ved de øvrige. Læg mærke til forskellen mellem strengudtrykket her og udtrykket, som bruges til at udskrive feltet i linie 15060. '-1' i slutningen er fjernet, således at skilletegnet '|' ikke udelades.

Linie 15180-15190: Den ændrede post opbygges i TE$. Når posten er færdig, er den oprindelige post slettet fra filen. Grunden til dette er at rettelserne kan betyde, at den ændrede post skal stå et andet sted i den alfabetiske fil. Nu bliver posten sendt til binær søgning-modulet, og indsat i filen på ny. Placeringen i filen bliver lagt ind i S1, således at søgemodulet kan vise den rigtige post, hvis placeringen er blevet ændret.

Test

Kør programmet og hent de fire poster fra båndet endnu en gang. Vælg søgningen på hovedmenuen og tast <ENTER>, så vi får post 1 frem. Vælg nu 'R' på den anden søgemenu. Nu kommer det første felt, 'AA1', i denne post frem på skærmen, sammen med rettelsesmenuen. Indtast 'AAA1', og tast <ENTER> når det andet felt vises. Nu skal programmet vende tilbage til den anden søgemenu igen og posten skal nu se således ud:

ET: AAA1
TO: AA2

Prøv at slette hele poster og lav andre rettelser. Fungerer det, skulle programmet være klar til brug.

Program 4.2: NTAL

Arkivering omfatter ikke kun ord. En af de ting mikrocomputere er bedst til at lagre og behandle er tal. I dette program 'NTAL' (sammentrækning af 'navne og tal') kan du lagre navnene på forskellige ting, varer eller lignende (her kaldet 'poster'), sammen med de enheder de normalt måles i og et tal. F.eks. kan programmet bruges til at holde styr på dagens salg i en lille butik.

En butik har en række varer (posterne), som tilsammen udgør det der kaldes varelageret. Alle posterne har naturligvis navne og de sælges i forskellige enheder (f.eks. pose, dåse, kasse, flaske..), og til dem alle er der knyttet et ret vigtigt tal – nemlig prisen. Hvis vi derfor vil have computeren til at hjælpe os med at lave fakturaer, må vi have den til at huske disse data for os. Et andet eksempel er de fødevarer vi bruger i husholdningen. Disse fødevarer har forskellige navne og de beregnes i forskellige enheder (liter, kop, knivspids, kilo). Hvis vi vil holde øje med, hvor meget vi spiser, det det ikke prisen, men antallet af kalorier per enhed, vi lagrer.

Det er bare et par eksempler, du kan let finde på mange flere, hvor det er nyttigt at lagre navne, enheder og tilknyttede tal sammen for en lang række poster.

Ideen med programmet er at du laver et 'katalog' over nogle poster (op til 200), de enheder de måles i og et tal som f.eks. prisen. På grundlag af kataloget kan du derefter lave en liste over de poster og de mængder, du ønsker og programmet vil lægge de sammen for dig. Med NTAL kan du opregne dagens kalorieindtag, eller dagens salg eller noget helt tredie.

Modul 4.2.1: Initialisering

Et standardmodul, der giver brugeren mulighed for at angive hvilken type poster programmet skal arbejde med. Det kan f.eks. være fødevarer eller lagervarer. Det navn, du bruger her, vil gå igen i hele programmet.

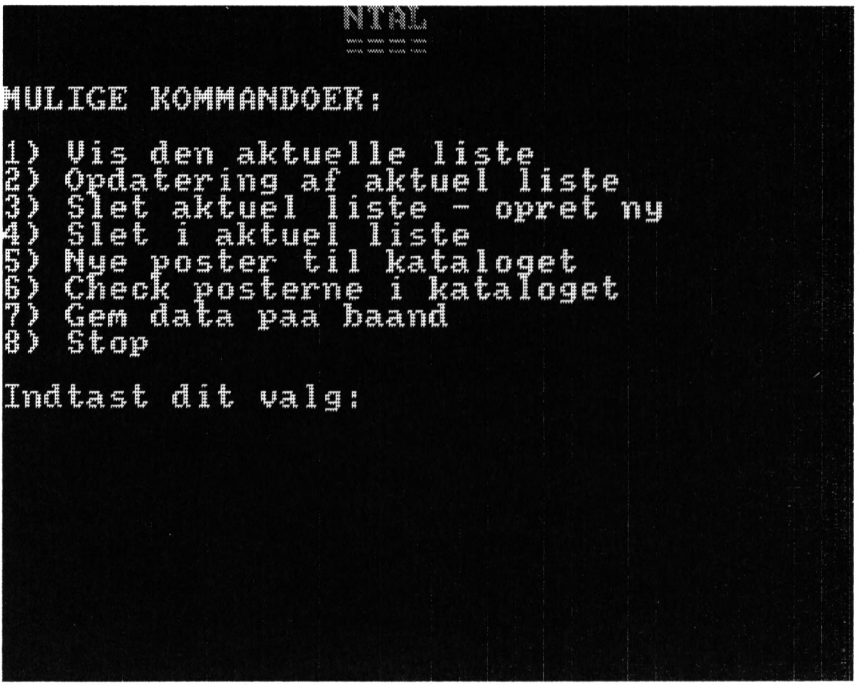


Fig. 4.2: Skærmdump fra NTAL

Modul 4.2.1: Linie 10000-10090

```
10000 REM ****************************
10010 REM Initialisering
10020 REM ****************************
10030 MODE 1
10040 DIM array$(1000,1),array(1000),te$
(100,1),te(100):ak=0:pt=0
10050 PEN 2:PRINT TAB(17);"NTAL":PRINT T
AB(17);"===="
10060 WINDOW 1,40,4,25
10070 PEN 1
```

```
1ØØ8Ø INPUT "Hent fra bånd (j/n)";q$
1ØØ9Ø IF LOWER$(q$)="j" THEN GOSUB 22ØØØ
 ELSE PRINT:INPUT "Navnet for denne fil:
 ",nf$:nf$=UPPER$(nf$)
```

Kommentar

Linie 10040: Arrayet ARRAY$ bruges til at lagre postens navn og enhedens navn i kataloget, – det tilknyttede tal lagres som det tilsvarende element i arrayet ARRAY.

TE$ og TE bruges til samme opgaver som ARRAY$ og ARRAY, blot er det for den 'aktuelle liste', som laves ud fra kataloget. Variablen AK indeholder antallet af poster i den aktuelle liste – som stadigvæk laves på grund af kataloget. PT er antallet af poster i kataloget.

Modul 4.2.2: Menuen

Igen et standard menumodul, – det eneste nye er den måde, som adgangen til bestemte programfunktioner hindres på, inden der er indlæst data i programmet. Denne funktion kan dog ikke testes før næste modul er indlæst.

Modul 4.2.2: Linie 11000-11210

```
11ØØØ REM ****************************
11Ø1Ø REM Hoved menu
11Ø2Ø REM ****************************
11Ø3Ø WHILE z<>8
11Ø4Ø CLS
11Ø5Ø PRINT"MULIGE KOMMANDOER:"
11Ø6Ø PRINT
11Ø7Ø PRINT "1) Vis den aktuelle liste"
11Ø8Ø PRINT "2) Opdatering af aktuel lis
te"
11Ø9Ø PRINT "3) Slet aktuel liste - opre
t ny"
111ØØ PRINT "4) Slet i aktuel liste"
1111Ø PRINT "5) Nye poster til kataloget
"
1112Ø PRINT "6) Check posterne i katalog
et"
```

```
11130 PRINT "7) Gem data på bånd"
11140 PRINT "8) Stop"
11150 PRINT:INPUT"Indtast dit valg: ",z
11160 IF (z=1 OR z=2 OR z=4 OR z=6 OR z=
7) AND pt=0 THEN x$="    ****** ENDNU IN
GEN DATA ******":GOSUB 23000:z=0
11170 ON z GOSUB 12000,13000,14000,20000
,15000,18000,21000
11180 WEND
11190 CLS
11200 LOCATE 11,11:PRINT "PROGRAM AFSLUT
TET"
11210 END
```

Modul 4.2.3: Fejlmeldinger

Dette lille modul sparer effektivt hukommelsesplads, ved at kunne lave forskellige fejlmeldinger for brugeren. Modulet udskriver blot strengen X$, udsender en tone og vender tilbage til programmet igen. Hvis du kigger tilbage i linie 11160 kan du se et eksempel på anvendelsen – X$ defineres og fejlmodulet kaldes med GOSUB.

Modul 4.2.3: Linie 23000-23070

```
23000 REM ****************************
23010 REM Fejl
23020 REM ****************************
23030 PRINT:PRINT x$
23040 SOUND 1,1000,100
23050 FOR i=1 TO 1500
23060 NEXT i
23070 RETURN
```

Test

Kør programmet. Når du kommer frem til menuen vælger du '1', og du skal få en fejlmelding om, at der endnu ikke er indlæst data.

Modul 4.2.4 og 4.2.5: Gemning og hentning på bånd

To standardmoduler.

```
21ØØØ REM ******************************
21Ø1Ø REM Gem på bånd
21Ø2Ø REM ******************************
21Ø3Ø CLS
21Ø4Ø PRINT "Gem data:":PRINT
21Ø5Ø INPUT "Filens navn:",fil$
21Ø6Ø OPENOUT fil$
21Ø7Ø PRINT #9,nf$
21Ø8Ø PRINT #9,ak
21Ø9Ø PRINT #9,pt
211ØØ FOR i=Ø TO ak-1
2111Ø PRINT #9,te$(i,Ø)
2112Ø PRINT #9,te$(i,1)
2113Ø PRINT #9,te(i)
2114Ø NEXT i
2115Ø FOR i=Ø TO pt-1
2116Ø PRINT #9,array$(i,Ø)
2117Ø PRINT #9,array$(i,1)
2118Ø PRINT #9,array(i)
2119Ø NEXT i
212ØØ CLOSEOUT
2121Ø RETURN
22ØØØ REM ******************************
22Ø1Ø REM Hent fra bånd
22Ø2Ø REM ******************************
22Ø3Ø CLS
22Ø4Ø PRINT "Hent data:":PRINT
22Ø5Ø INPUT "Filens navn:",fil$
22Ø6Ø OPENIN fil$
22Ø7Ø INPUT #9,nf$
22Ø8Ø INPUT #9,ak
22Ø9Ø INPUT #9,pt
221ØØ FOR i=Ø TO ak-1
2211Ø INPUT #9,te$(i,Ø)
2212Ø INPUT #9,te$(i,1)
2213Ø INPUT #9,te(i)
2214Ø NEXT i
2215Ø FOR i=Ø TO pt-1
2216Ø INPUT #9,array$(i,Ø)
2217Ø INPUT #9,array$(i,1)
```

```
2218Ø INPUT #9,array(i)
2219Ø NEXT i
2220Ø CLOSEIN
2221Ø RETURN
```

Modul 4.2.6: Binær søgning

Se kommentaren til DATAFIL. Det eneste du skal lægge mærke til her er, at sorteringen sker ud fra post-navnet i kolonne nul i ARRAY$.

Modul 4.2.6: Linie 16000-16110

```
16ØØØ REM ****************************
16Ø1Ø REM Binær søgning
16Ø2Ø REM ****************************
16Ø3Ø IF pt=Ø THEN ss=Ø:RETURN
16Ø4Ø po=INT(LOG(pt)/LOG(2)):ss=2^po-1
16Ø5Ø FOR i=po TO Ø STEP -1
16Ø6Ø ss=ss+2^i*((array$(ss,Ø)>t1$)-(arr
ay$(ss,Ø)<t1$))
16Ø7Ø IF ss<Ø THEN ss=Ø
16Ø8Ø IF ss>pt-1 THEN ss=pt-1
16Ø9Ø NEXT i
161ØØ IF array$(ss,Ø)<t1$ THEN ss=ss+1
1611Ø RETURN
```

Modul 4.2.7: Indsættelse af poster i kataloget

Dette modul svarer ret nøje til det tilsvarende modul i DATATIL. Grunden til at det er en smule længere er, at der skal indsættes to strenge og et tal i to arrays fremfor en i kun én streng.

Modul 4.2.7: Linie 17000-17110

```
17ØØØ REM ****************************
17Ø1Ø REM Indsæt post
17Ø2Ø REM ****************************
17Ø3Ø FOR i=pt TO ss+1 STEP -1
17Ø4Ø array$(i,Ø)=array$(i-1,Ø)
17Ø5Ø array$(i,1)=array$(i-1,1)
17Ø6Ø array(i)=array(i-1)
```

```
17Ø7Ø NEXT i
17Ø8Ø array$(ss,Ø)=t1$
17Ø9Ø array$(ss,1)=t2$
171ØØ array(ss)=nf
1711Ø RETURN
```

Modul 4.2.8: Indlæsning af poster i kataloget

Dette modul er en del enklere end modulet i DATATIL. Her bedes der om tre indlæsninger: (a) postens navn, (b) navnet på den enhed der bruges og (c) tallet. Bemærk at 'tallet' her i programmet er 'pris'. Det gør det lidt nemmere at forstå – når det altså er priser, det handler om. Hvis du vil, kan du rette det til 'tal', 'kvantum', 'mængde' el. lign.

Modul 4.2.8: Linie 15000-15170

```
15ØØØ REM ****************************
15Ø1Ø REM Nye poster til kataloget
15Ø2Ø REM ****************************
15Ø3Ø WHILE 1
15Ø4Ø CLS
15Ø5Ø PRINT "Nye poster til kataloget: "
:PRINT
15Ø6Ø IF pt>1ØØØ THEN x$="    ******* IK
KE MERE PLADS *******":GOSUB 23ØØØ:RETUR
N
15Ø7Ø q$="":WHILE LOWER$(q$)<>"j"
15Ø8Ø PRINT nf$:INPUT" (navn eller '|' f
or retur): ",t1$
15Ø9Ø IF t1$="|" THEN RETURN
151ØØ PRINT "Enhed";:INPUT ": ",t2$
1511Ø PRINT "Pris per ";LOWER$(t2$);:INP
UT ": ",nf
1512Ø PRINT:INPUT "Er det OK (j/n)";q$:P
RINT
1513Ø WEND
1514Ø GOSUB 16ØØØ
1515Ø GOSUB 17ØØØ
1516Ø pt=pt+1
1517Ø WEND
```

Test

Nu kan vi endelig lave en rigtig test, af det der indtil videre er indlæst.

Kør programmet. Du skal ikke hente fra båndet nu – brug navnet post som navn. På menuen vælger du '5' 'Nye poster til kataloget'. Derefter kan du lave disse tre indlæsninger:

```
TING1|KASSE|10
TING2|FLASKE|20
TING3|POSE|40
```

Betegnelsen giver ingen særlig mening, det bruges kun til vores test.

Når indlæsningen af dette er slut taster du '|'. Når menuen kommer frem igen vælger du '7' og gemmer disse informationer på bånd. Stop programmet med '8' og skriv:

```
FOR I=0 to 2:?ARRAY$(I,0),ARRAY$(I,1),ARRAY(I):NEXT
<ENTER>
```

Nu skal fu få dette frem på skærmen:

```
TING1    KASSE     10
TING2    FLASKE    20
TING3    POSE      40
```

Kør nu programmet og vælg at hente data fra bånd. Angiv det navn, som blev brugt da informationerne blev gemt. Når båndet stopper igen skal du kunne lave testen ovenfor en gang til.

Modul 4.2.9: Søgningsrutinen

Dette modul giver, ligesom i sidste program, mulighed for at 'bladre' eller springe i filen (kataloget), at søge på posternes navne og at slette poster fra filen. Modulet er noget enklere end DATAFIL-modulet, fordi der kun kan søges på hele poster, fremfor på tilfældige sekvenser af karakterer i en post. Dertil kommer, at de enkelte poster har en mere enkel struktur, end posterne i DATAFIL.

Modul 4.2.9: Linie 18000-18250

```
18ØØØ REM ****************************
18Ø1Ø REM Bruger søgning
18Ø2Ø REM ****************************
18Ø3Ø ss=Ø
18Ø4Ø t1$="":WHILE pt>Ø
18Ø5Ø CLS:PRINT "Søgning:":PRINT
18Ø6Ø PRINT "Post nummer:";ss+1
18Ø7Ø PRINT nf$;": ";array$(ss,Ø)
18Ø8Ø PRINT "Enhed: ";array$(ss,1)
18Ø9Ø PRINT "Pris per ";LOWER$(array$(ss
,1));":";array(ss)
181ØØ PRINT:PRINT "Mulige kommandoer:":P
RINT
1811Ø PRINT "Indtast den post du søger"
1812Ø PEN 2:PRINT "ENTER";:PEN 1:PRINT "
 for næste post"
1813Ø PRINT "'#' og et tal for at spring
e i filen"
1814Ø PRINT "'s' for slette en post"
1815Ø PRINT "'|' for at returnere til me
nuen"
1816Ø PRINT:INPUT "Hvad vælger du: ";t1$
1817Ø IF t1$="" THEN t1$="#1"
1818Ø IF t1$="|" THEN RETURN
1819Ø IF LOWER$(t1$)="s" THEN GOSUB 19ØØ
Ø:t1$=""
182ØØ IF LEFT$(t1$,1)="#" THEN ss=ss+VAL
(MID$(t1$,2)):t1$=""
1821Ø IF t1$<>"" THEN GOSUB 16ØØØ
1822Ø IF ss>pt-1 THEN ss=pt-1
1823Ø IF ss<Ø THEN ss=Ø
1824Ø WEND
1825Ø RETURN
```

Kommentar

Linie 18170: Fremfor specielt at tage højde for at der tastes <ENTER> dvs. en tom streng – bliver variablen T1$ fra start sat til 1. Hvis brugeren taster <ENTER> ændres T1$ ikke, og den næste post vil blive vist.

Test

Kør programmet, hent de tre poster fra båndet og vælg '6' på menuen. Nu skal du kunne 'bladre' frem og tilbage med '#' efterfulgt af et tal – ligesom i det forrige program. Det skal også være muligt at finde en post ved at søge på dets navn, – men ikke ved at søge på enhedens navn.

Modul 4.2.10: Sletning af en post

Svarer helt til modulet i sidste program.

Modul 4.2.10: Linie 19000-19090

```
19ØØØ REM ****************************
19Ø1Ø REM Slet
19Ø2Ø REM ****************************
19Ø3Ø FOR i=ss TO pt-2
19Ø4Ø array$(i,Ø)=array$(i+1,Ø)
19Ø5Ø array$(i,1)=array$(i+1,1)
19Ø6Ø array(i)=array(i+1)
19Ø7Ø NEXT i
19Ø8Ø pt=pt-1
19Ø9Ø RETURN
```

Test

Kør programmet, hent igen data fra båndet, vælg igen '6' på hovedmenuen og tast 'S' ved en af posterne. Denne post skal nu være fjernet fra kataloget.

Modul 4.2.11: Opdatering af de aktuelle lister

Formålet med NTAL er ikke kun at lave et katalog over en række poster med enheder og (her) priser, kataloget skal også kunne bruges som udgangspunkt for oprettelse af midlertidige (eller 'aktuelle') lister. Med modulet kan posterne i kataloget kopieres over i den aktuelle liste. Modulet giver samtidig brugeren mulighed for at indføje poster i den aktuelle liste, at udskrive denne liste, at slette enkelte poster eller at slette hele listen på en gang.

Modul 4.2.11: Linie 13000-13210

```
13000 REM ****************************
13010 REM Opdatering af aktuel liste
13020 REM ****************************
13030 WHILE 1
13040 CLS
13050 PRINT "Tilføjelse til den aktuelle
 liste:":PRINT
13060 IF ak>100 THEN x$="  **  DEN AKTUE
LLE LISTE ER FYLDT **":GOSUB 22000:RETUR
N
13070 PRINT nf$;:INPUT " ('|' for retur)
:",t1$
13080 IF t1$="|" THEN RETURN
13090 kendt=0:GOSUB 16000:IF array$(ss,0
)=t1$ THEN kendt=1
13100 IF kendt=0 THEN x$="*** "+UPPER$(n
f$)+" UKENDT, CHECK VENLIGST ***":GOSUB
23000:RETURN
13110 PRINT:PRINT "Enheder: ";array$(ss,
1)
13120 INPUT "Antal: ",q
13130 PRINT: INPUT "Er det OK (j/n)";q$
13140 WHILE LOWER$(q$)="j"
13150 te$(ak,0)=array$(ss,0)
13160 te$(ak,1)=MID$(STR$(q),2)+" "+arra
y$(ss,1)
13170 te(ak)=q*array(ss)
13180 ak=ak+1
13190 q$=""
13200 WEND
13210 WEND
```

Kommentar

Linie 13090-13100: Disse linier checker om den post, som brugeren indlæser, for at placere den i den aktuelle liste, findes i kataloget. Det sker ved at kalde binærsøgning modulet, som finder den placering hvor posten skal stå i kataloget. Posten i kataloget bliver derefter sammenlignet med brugerens indlæsning. Hvis den post brugeren har indlæst findes i

kataloget er de to poster ens, er de forskellige kommer der en fejludskrift, og der vendes tilbage til hovedmenuen.

Linie 13110: Når posten er fundet i kataloget, udskriver modulet de enheder posten måles i og spørger om hvor mange af disse enheder, der skal medregnes.

Linie 13120-13170: Her indlæses det faktiske antal enheder. Brugeren bedes om at bekræfte indlæsningen før den indgår i den aktuelle liste, som ligger i de to arrays TE$ og TE. Bemærk at den størrelse der lægges i TE ikke er størrelsen (prisen) i kataloget, men den *samlede* pris for det antal enheder, som brugeren har angivet.

Test

Kør programmet og hent som sædvanligt data fra båndet. Vælg '2' på menuen og indlæs de følgende poster og antal enheder:

TING1,1
TING2,2
TING3,3

Prøv nu at skrive 'TING4', som jo ikke findes i kataloget. Nu skal du få en fejlmelding og en opfordring til at checke postens navn. Før du gør mere skal du først gemme dine data på bånd. Stop endelig programmet med '8'.

Skriv nu følgende

```
FOR I=0 TO 2:?TE$(I,0),TE$(I,1),TE(I):NEXT <ENTER>
```

Nu skal du se denne udskrift:

```
TING1     1 KASSE      10
TING2     2 FLASKE     40
TING3     3 POSE      120
```

Modul 4.2.12: Vis den aktuelle liste

Formålet med dette modul er at udskrive posterne i den aktuelle liste en for en på skærmen. Efter hver post, skal brugeren trykke på en tast, for at få den næste post udskrevet. Grunden til dette er at listen oftest ikke vil kunne være på skærmen på én gang. Når listen ikke er længere, udskrives den samlede total for alle posterne.

Modul 4.2.12: Linie 12000-12160

```
12000 REM ****************************
12010 REM Vis den aktuelle liste
12020 REM ****************************
12030 IF ak=0 THEN RETURN
12040 CLS:at=0
12050 FOR i=0 TO ak-1
12060 PRINT nf$;": ";te$(i,0)
12070 PRINT "Enheder: ";te$(i,1)
12080 PRINT "Pris:";te(i)
12090 PRINT "-------------------------
------------"
12100 WHILE INKEY$="":WEND
12110 at=at+te(i)
12120 NEXT i
12130 PRINT:PRINT "Total:";at
12140 PRINT:PRINT "Retur til menuen: try
k på en af tasterne"
12150 WHILE INKEY$="":WEND
12160 RETURN
```

Modul 4.2.13: Sletning i den aktuelle liste

Dette er en forenklet udgave af søgemodulet til kataloget. Det tillader brugeren at 'bladre' gennem den aktuelle liste post for post, og når en post ønskes slettet tastes 'S'. Funktionen kan afbrydes med '|'.

Modul 4.2.13: Linie 20000-20250

```
20000 REM ****************************
20010 REM Sletning i aktuel liste
20020 REM ****************************
20030 i=0
20040 WHILE i<ak
20050 CLS
20060 PRINT "Sletning i aktuel liste: ":
PRINT
20070 PRINT te$(i,0)
20080 PRINT te$(i,1)
20090 PRINT:PRINT "Mulige kommandoer:":P
RINT
```

```
20100 PEN 2:PRINT "ENTER";:PEN 1:PRINT "
 for næste post"
20110 PRINT "'s' for at slette"
20120 PRINT "'|' for at returnere til me
nuen"
20130 PRINT:INPUT "Hvad vælger du: ";q$
20140 IF q$="|" THEN RETURN
20150 IF LOWER$(q$)<>"s" THEN i=i+1
20160 WHILE LOWER$(q$)="s"
20170 FOR j=i TO ak-1
20180 te$(j,0)=te$(j+1,0)
20190 te$(j,1)=te$(j+1,1)
20200 te(j)=te(j+1)
20210 NEXT j
20220 ak=ak-1
20230 q$="":WEND
20240 WEND
20250 RETURN
```

Kommentar

Linie 20150-20230: Den variabel, der bestemmer hvilken post der vises er 'I'. Bemærk at den *ikke* opskrives hvis der tastes 'S'. 'I' vil i dette tilfælde pege på den post der skal slettes, og det sker simpelthen ved at den næste post i rækken rykkes frem til denne plads.

Test

Kør programmet og hent den fil der blev gemt efter den sidste test. Vælg nu '4' 'Slet i aktuel liste' på hovedmenuen. Nu skal du kunne 'bladre' gennem de tre poster i den aktuelle liste og slette en af dem. Check ved at vælge 'I'.

Modul 4.2.14: Initialisering af aktuel liste

I mange tilfælde vil der være behov for at oprette en aktuel liste, få udregnet en total og derefter at få lavet en ny aktuel liste. Fremfor at slette den 'gamle' liste post for post slettes hele listen på én gang med dette modul.

Modul 4.2.14: Linie 14000-14090

```
14ØØØ REM ****************************
14Ø1Ø REM Opret ny aktuel liste
14Ø2Ø REM ****************************
14Ø3Ø FOR i=Ø TO ak-1
14Ø4Ø te$(i,Ø)=""
14Ø5Ø te$(i,1)=""
14Ø6Ø te(i)=Ø
14Ø7Ø NEXT i
14Ø8Ø ak=Ø
14Ø9Ø RETURN
```

Test

Kør programmet og hent data på båndet som i sidste test. Vælg nu '3' på menuen. Nu vender menuen straks tilbage. Hvis du nu vælger '1' vender menuen igen tilbage med det samme, fordi der nu ikke er nogen aktuel liste at vise.

Hvis denne test er OK, er programmet klar til brug.

Program 4.3: Tekst

Noget af det, computeren anvendes mest til, er tekstbehandling. En tekstbehandler kan klare meget af det besværlige indskrivningsarbejde, og selv opstilling af dokumenter er en ret enkel opgave. Imidlertid er det desværre sådan, at et rigtigt tekstbehandlingsprogram fylder en del mere end en hjemmecomputer som f.eks. Amstrad kan rumme. Endvidere er BASIC normalt for langsomt til tekstbehandling, så sådanne programmer kører normalt i maskinkode – det sprog som kan forstås direkte af Amstrads Z80 processor.

Programmet her kan ikke rigtigt betegnes som et tekstbehandlingsprogram, men det kan trods alt få din Amstrad til at blive en rimelig avanceret skrivemaskine. Du kan således flytte rundt på tekstblokke, slette linier og ord mv., og få lavet udskrifter på en printer. Hvis du skal bruge noget der kan anvendes på et kontor, så er dette program ikke sagen, men hvis du skal skrive et brev af og til, så klarer 'TEKST' – opgaven fint.

```
Kære læser

Dette tekstprogram har ikke så mange faciliteter, så det
kan ikke konkurrere med store tekstbehandlingsprogrammer
som fx 'Wordstar'.

Og dog, - det fylder langt mindre, så det kan bruges på
hjemmecomputere. Og så er det også en anelse billigere.

Med venlig hilsen ...
```

Fig. 4.3: Et lille brev skrevet med 'TEKST'.

I programmet præsenteres følgende:

1) Program-styrede markører og indlæsning af tekst.
2) Bearbejdelse af materiale, der ligger i store strengarrays.
3) Udskrift på printer af tekst.

Modul 4.3.1: Initialisering

Modul 4.3.1: Linie 11000-11100

```
11ØØØ REM ******************************
11Ø1Ø REM Initialisering
11Ø2Ø REM ******************************
11Ø3Ø in=1
11Ø4Ø DIM text$(1ØØ):si=1:pc=1
11Ø5Ø text$(Ø)=STRING$(32,CHR$(245))
11Ø6Ø text$(1)=STRING$(32,CHR$(244))
11Ø7Ø a$=" "
11Ø8Ø INPUT "Vil du hente tekst fra bånd
 (j/n)";q$
11Ø9Ø IF LOWER$(q$)="j" THEN GOSUB 19ØØØ
111ØØ RETURN
```

Kommentar

Linie 11040: Arrayet TEXT$ skal bruges til at indeholde teksten.

Linie 11050-11060: Disse to linier laver grafiske symboler, som afmærker henholdsvis top og bund af teksten.

Modul 4.3.2: Indlæsning af karakterer

Det første programmet skal kunne er at modtage en indlæsning via tastaturet og *gøre* noget med den, og det klares at dette modul. De fleste linier i programmet beskæftiger sig med indlæsning af 'kommando-karakterer', som giver programmet besked om at udføre forskellige funktioner som f.eks. at slette en karakter. Nogle af linierne styrer markøren, og disse linier minder om dem vi kender fra grafikkapitlet.

Modulet søger for at vi kan skrive tekst på linie 23 og 24 på skærmen, og at der kan rettes i denne linie. Senere moduler vil så kunne flytte disse linier ind i hovedteksten.

Læg i øvrigt mærke til at der her er brugt tomme programlinier, for at gøre det nemmere at overskue programmets opbygning. Da 'TEKST' indeholder en masse små løkker og en masse variabler, vil det være meget svært at finde rundt i, hvis ikke opbygningen blev anskueliggjort på denne måde. Som i det tidligere program kan du springe de tomme programlinier (:) over hvis du vil.

Modul 4.3.2: Linie 12000-12430

```
12ØØØ REM ****************************
12Ø1Ø REM Skrivning af teksten
12Ø2Ø REM ****************************
12Ø3Ø :
12Ø4Ø :
12Ø5Ø WHILE 1
12Ø6Ø :
12Ø7Ø :
12Ø8Ø t$="":WHILE t$=""
12Ø9Ø LOCATE p-4Ø*INT(p/4Ø)+1,23+INT(p/4
Ø):PEN 1:PRINT CHR$(143)
121ØØ FOR tt=1 TO 2Ø:NEXT tt
1211Ø LOCATE p-4Ø*INT(p/4Ø)+1,23+INT(p/4
Ø):PEN 2:PRINT MID$(a$,p+1,1)
1212Ø t$=INKEY$
1213Ø WEND
1214Ø :
```

```
12150 :
12160 IF t$=CHR$(13) OR (LEN(a$)>40 AND
t$=" ") THEN t$="":GOSUB 13000
12170 IF t$="^" THEN GOSUB 15000
12180 :
12190 :
12200 WHILE t$="|"
12210 IF p<>0 THEN t=0 ELSE t=LEN(a$)-1
12220 p=t
12230 t$="":WEND
12240 :
12250 :
12260 WHILE p>0 AND t$=CHR$(127)
12270 a$=LEFT$(a$,p-1)+MID$(a$,p+1)
12280 p=p-1
12290 t$="":WEND
12300 :
12310 :
12320 WHILE t$>=CHR$(32) AND t$<=CHR$(16
3) AND t$<>CHR$(127)
12330 a$=LEFT$(a$,p)+t$+MID$(a$,p+1)
12340 p=p+1
12350 t$="":WEND
12360 :
12370 :
12380 IF t$=CHR$(242) AND p>0 THEN p=p-1
12390 IF t$=CHR$(243) AND p<LEN(a$)-1 TH
EN p=p+1
12400 :
12410 :
12420 LOCATE 1,23:PRINT a$
12430 WEND
```

Kommentar

Linie 12080-12130: Her laves en blinkende (eller flimrende) markør, som skifter mellem 'inverst' mellemrum og en karakter i strengen A$, hvori teksten først indlæses. Markørens placering bestemmes af variablen P

Linie 12160: Denne linie kalder en senere del af programmet, som indfø-

jer den skrevne linie i hovedteksten. Det sker når der tastes <ENTER>, eller det sker automatisk, når der kommer et mellemrum og liniens længde er mere end 40 karakterer.

Linie 12170: Ved at taste '↑' (potenstegnet) – på samme tast som '£') kaldes et senere modul, som giver adgang til at udføre en række redigeringsfunktioner på hovedteksten.

Linie 12200-12230: Ved at taste '|' flyttes markøren til starten af linien, eller – hvis den allerede befinder sig der – til slutningen af linien.

Linie 12260-12290: Forudsat at markøren ikke befinder sig på første karakter i linien, vil et tryk på <DEL> slette karakteren til venstre for markøren. Det gøres meget enkelt ved at lave strengen A$ uden karakteren på pladsen P.

Linie 12320-12350: Disse linier godkender alle karakterer med en kode mellem 32 og 163 (se listen i manualens appendix III), *bortset fra* <DEL> som er behandlet ovenfor. Den nye karakter indsættes derefter i tekstlinien, dér hvor markøren står.

Linie 12380-12390: Markøren flyttes til højre eller venstre ved at ændre værdien af P, alt efter hvilken markørpil der trykkes ned. Når vi definerer vores egen markør skal vi, som i grafikkapitlet, lægge mærke til at vi selv kan bestemme om kontroltasterne, som f.eks. markørpilene, <CLR> og <DEL>, skal have nogen funktion, idet de ligger udenfor det område der godkendes af løkken ovenfor. Vi kan imidlertid godt bruge dem som funktionstaster, hvor vi selv bestemmer hvilken funktion, de skal have – men <ESC> stopper dog altid programmet.

Test

Som i tidligere programmer skal du allerede nu indlæse nogle af linierne i kontrolløkken, for at vi kan lave en test. Skriv:

```
10040 IF IN=0 THEN GOSUB 11000
10060 GOSUB 12000
```

og kør så programmet. Svar 'n' til det første spørgsmål, og du skal derefter se en flimrende markør nederst på skærmen. Nu kan du begynde at skrive på denne linie, og du skal kunne slette med <DEL>. Endvidere skal du kunne flytte markøren trin for trin med markørpilene, og fra den ene

ende af linien til den anden med '|'. Hvis du taster <ENTER> eller hvis du skriver mere end én linie efterfulgt af et mellemrum, stopper programmet med 'Undefined line'.

Modul 4.3.3: Overflytning af tekst til hovedteksten

De næste to moduler sørger sammen for, at den tekst, der skrives nederst på skærmen, bliver behandlet og indsat i hovedteksten, som ligger i TEXT$, og som kan rumme 100 linier med 32 karakterer i hver. Selve arbejdet gøres af dette modul, men du vil først kunne se resultatet, når næste modul, som udskriver hovedteksten, er indlæst.

Modul 4.3.3: Linie 13000-13310

```
13000 REM ****************************
13010 REM Indsæt linie
13020 REM ****************************
13030 IF a$="* " THEN a$=SPACE$(33)
13040 x=0
13050 WHILE a$<>""
13060 :
13070 :
13080 d=0:WHILE LEN(a$)<34 AND d=0
13090 mt$(x)=LEFT$(a$,LEN(a$)-1)
13100 a$=""
13110 d=1:WEND
13120 :
13130 :
13140 WHILE LEN(a$)>33
13150 FOR i=33 TO 1 STEP -1
13160 IF MID$(a$,i,1)<>" " THEN NEXT i:i
=33
13170 mt$(x)=LEFT$(a$,i-1)
13180 a$=MID$(a$,i+1)
13190 x=x+1
13200 WEND
13210 WEND
13220 x=x+1
13230 :
13240 :
13250 FOR i=si+x TO pc+x STEP -1
```

```
13260 text$(i)=text$(i-x)
13270 NEXT i
13280 FOR i=0 TO x-1
13290 text$(pc+i)=mt$(i)
13300 NEXT i
13310 a$=" ":p=0:si=si+x:pc=pc+x
```

Kommentar

Linie 13030: Dette er et lille arrangement, som bruges, når der skal laves udskrift på en printer. Der kommer nemlig et lille problem når vi vil lave udskrifter til højre på en side. Teksten på skærmen fylder nemlig 32 karakterer i bredden, mens en printer normalt skriver med 80 karakterer per linie.

Når der laves udskrift samler TEKST to skærmlinier til en linie i den endelige udskrift. Når der skrives tekst i skærmens højre side, vil det derfor betyde, at det enten udskrives midt på siden, eller at det bliver koblet på en foregående skærmlinie. En lidt besværlig måde at klare det problem på er, at indlæse en tom linie ved at taste <ENTER> ved starten af en ny linie. Det betyder at vi får en tom linie på papiret og at den næste linie starter helt til venstre, – derefter skal vi lave en hel linie med mellemrum efterfulgt af endnu en linie med mellemrum, frem til det eller de ord vi vil have udskrevet til højre. Resultatet bliver så at printeren vil udskrive alle mellemrummene, og at det (de) sidste ord vil blive udskrevet til højre på papiret.

Men i programmet her kan vi i stedet lave en linie med mellemrum ved at indlæse en linie med en enkel '*'. Modulet her oversætter automatisk '*' til en linie med 33 mellemrum.

Linie 13040: X er en variabel, der bruges til at vise hvor mange tekstlinier, der skal bruges til den nye tekst.

Linie 13050-13210: Hver gang en linie karakterer indføjes i hovedteksten fjernes de samtidig fra A$ – den nye tekst, som indlæses ved hjælp af det forrige modul. Dette fortsætter indtil der ikke er noget tilbage af A$.

Linie 13080-13110: Først bliver den nye tekst placeret i et midlertidigt array MT$. Hvis der er mindre end 34 karakterer (mellemrummet for enden medregnet), passer den nye tekst ind i én linie med 32 karakterer. Denne skal blot overflyttes og A$ slettes.

Linie 13140-13200: Hvis der er mere end 33 karakterer, så den nye tekst ikke kan være i en enkelt linie, søges der tilbage fra karakter 33 efter det første mellemrum. Den del af A$ der står før dette mellemrum overflyttes nu – og på den måde sikres det at ingen ord bliver delt.

Denne del af A$ slettes nu, og hovedløkken kører nu en gang til på samme måde. Den eneste forskel er at teksten indsættes i en anden linie i MT$.

Linie 13250-13300: Variablen PC registrerer den placering i hovedteksten, hvor den nye tekst skal indsættes – den starter på 0 når der ikke ingen tekst er, og står normalt nederst i hovedteksten. Ved brug af senere moduler bliver det muligt for brugeren at flytte stedet, hvor ny tekst skal placeres, op og ned i hovedteksten. Den første af de to løkker i denne del af teksten starter med at gøre plads, der hvor den nye tekst skal indsættes, til det antal linier, der angives af X. Det sker ved at flytte alt det, der står efter PC og frem til sidste linie (SI), det antal linier op, som angives af X. Den anden løkke kopierer indholdet af MT$ ind på den ledige plads.

Test

Indlæs en ekstra linie:

14130 RETURN

og kør programmet. Når den flimrende markør viser sig, skriver du:

AAA <ENTER>
BBB <ENTER>
CCC <ENTER>
DDD <ENTER>

Derefter trykker du på <ESC>, så kørslen stoppes. Nu skriver du:

FOR I=0 TO 5:PRINT TEXT$(I):NEXT <ENTER>

Nu skal du se en række savtakker, efterfulgt af de fire linier tekst du har skrevet og endnu en række savtakker. Savtakkerne markerer begyndelsen og slutning på hovedteksten.

Modul 4.3.4: Vis hovedteksten

Dette modul udskriver 15 linier af hovedteksten på skærmen.

4.3.4: Linie 14000-14130

```
14ØØØ REM ****************************
14Ø1Ø REM Vis teksten
14Ø2Ø REM ****************************
14Ø3Ø ss=pc-7
14Ø4Ø IF si-pc<8 THEN ss=si-15
14Ø5Ø IF ss<Ø THEN ss=Ø
14Ø6Ø CLS
14Ø7Ø FOR i=ss TO ss+15
14Ø8Ø PEN 2
14Ø9Ø PRINT TEXT$(i)
141ØØ IF i=pc-1 THEN PEN 1:PRINT ">"
1411Ø NEXT i
1412Ø LOCATE 1,23:PEN 2:PRINT a$:PEN 1
1413Ø RETURN
```

Kommentar

Linie 14030-14050: Variablen SS står for starten af det afsnit af teksten, der skal vises på skærmen. Det udregnes således at det normalt er otte linier før det sted hvor ny tekst indsættes. Hvis dette sted imidlertid er tæt på tekstens afslutning – f.eks. helt forneden, så vil det betyde at der kun kommer otte linier på skærmen. I dette tilfælde træder linie 14040 i funktion og flytter SS længere tilbage. Og hvis der endelig er færre end 15 liniers tekst, eller hvis ny tekst skal indsættes i hovedtekstens start, så vil startpunktet falde før tekstens start, og i det tilfælde sættes det til 0.

Linie 14070-14110: Nu udskrives de 15 linier. Alle linierne slutter med et semikolon for at hindre at udskriftpositionen rykker en linie ned. Det skyldes at en fyldt linie med 40 karakterer vil sende markøren ned til den næste linie, og derved give en tom linie. For linier med mindre end 40 karakterer bliver semikolonets virkning ophævet af en 'tom' PRINT kommando. Den eneste finesse er ellers at stedet hvor den nye tekst indsættes afmærkes med '>' – alle redigeringsfunktioner, som f.eks. indsættelse af en linie, kopiering eller sletning vil gælde linien lige under denne markering.

Linie 14120: Og endelig udskrives A$ nederst på skærm, efter den bliver slettet.

Test

Lav samme test som ved sidste modul, med den ene forskel at den nye tekst nu skal blive flyttet op i hovedteksten, når der tastes <ENTER>.

Modul 4.3.5: Redigering af hovedteksten

Nu kan vi redigere i den nye tekst, mens den skrives, og næste trin er derefter at vi skal kunne redigere i hovedteksten også. Dette modul giver brugeren mulighed for at flytte indsættelsesstedet i hovedteksten – at kopiere en linie fra hovedteksten tilbage nederst på skærmen, så der kan rettes i den, samt at slette linier i hovedteksten.

Modul 4.3.5: Linie 15000-15330

```
15000 REM ******************************
15010 REM Redigerings mode
15020 REM ******************************
15030 WHILE 1
15040 p2=pc-ss
15050 :
15060 :
15070 t1$="":WHILE t1$=""
15080 LOCATE 1,p2+1:PRINT " ":FOR i=1 TO
 5:NEXT i
15090 LOCATE 1,p2+1:PRINT ">":FOR i=1 TO
 20:NEXT i
15100 t1$=LOWER$(INKEY$)
15110 WEND
15120 :
15130 :
15140 pc=pc+(t1$=CHR$(240))+10*(t1$="o")
:IF pc<1 THEN pc=1
15150 pc=pc-(t1$=CHR$(241))-10*(t1$="n")
:IF pc>si THEN pc=si
15160 IF t1$=CHR$(13) THEN t$="":RETURN
15170 :
15180 :
15190 WHILE pc<si AND t1$=CHR$(127)
15200 FOR i=pc TO si:text$(i)=text$(i+1)
:NEXT i
15210 si=si-1
15220 t1$="":WEND
```

```
15230 :
15240 :
15250 IF pc<si AND t1$="k" THEN a$=text$
(pc)+" "
15260 IF t1$="p" OR t1$="v" THEN GOSUB 1
7000
15270 IF t1$="g" THEN GOSUB 18000
15280 IF t1$="f" THEN GOSUB 16000
15290 IF t1$="h" THEN GOSUB 20000
15300 :
15310 :
15320 GOSUB 14000
15330 WEND
```

Kommentar

Linie 15040: I dette modul bliver linienummeret for '>', som viser hvor den nye tekst indsættes, angivet af variablen P2.

Linie 15070-15110: Redigeringsmarkøren ('>') sættes til at blinke for at angive at programmet er i redigerings *mode.*

Linie 15140-15150: Redigeringsmarkøren kan flyttes op og ned i teksten, når programmet er i redigeringsmode. Med de to markørpile flyttes markøren en linie op eller ned. Hvis der tastes 'O' eller 'N' flyttes markøren 10 linier op eller ned ad gangen. Bemærk at i praksis er det ikke markøren, der flyttes, men teksten, – med mindre markøren står i begyndelsen eller slutningen af teksten.

Linie 15160: Et tryk på <ENTER> afslutter redigeringsmode.

Linie 15190-15220: Trykkes der på <DEL> i redigeringsmode slettes linien under markøren.

Linie 15250: Trykkes der på 'K' i redigeringsmode, bliver linien under markøren kopieret nederst på skærmen, således at den kan ændres på forskellig vis.

Linie 15260: Trykkes der på 'P' eller 'V' i redigeringsmode kaldes et efterfølgende modul, som udskriver teksten på printer, eller viser i den på mode 2 skærmen.

Linie 15270: Trykkes der på 'G' i redigeringsmode, kaldes det modul, som gemmer teksten på bånd.

Linie 15280: Trykkes der på 'F' i redigeringsmode kaldes et efterfølgende modul, som formattterer teksten så pænt som muligt.

Linie 15290: Trykkes der på 'H' i redigeringsmode kaldes et efterfølgende 'hjælpe'-modul, der giver en vejledning i brugen af TEKST.

Test

Lav igen samme test med fire grupper bogstaver.

1) Tryk på '↑' og redigeringsmarkøren skal begynde af blinke.
2) Prøv at flytte redigeringsmarkøren op og ned i teksten med markørpilene og med 'O' og 'N'. De to sidste funktioner flytter kun markøren til hhv. top og bund af teksten, fordi der kun er fire linier.
3) Flyt redigeringsmarkøren hen over den sidste linie, tryk på 'K' og 'DDD' skal blive kopieret nederst på skærmen.
   4) Tryk på <DEL> og 'DDD' skal forsvinde fra hovedteksten.
5) Flyt redigeringsmarkøren op over første linie i teksten.
6) Tryk på <ENTER> så programmet kommer ud af redigeringsmode.
7) Når den blinkende markør kommer frem igen nederst på skærmen trykker du på <ENTER>, og 'DDD' skal komme frem øverst i hovedteksten.

Modul 4.3.6: Formattering af teksten

Vi mangler endnu en meget nyttig funktion i vores program: at formattere hovedteksten. Det er nødvendigt at kunne, fordi en af de største fordele ved tekstbehandling (og altså også ved dette lille program) er, at vi kan lave om på den bestående tekst. Når vi gør det, bliver teksten meget ofte meget ujævn at se på, fordi de fleste tekstbehandlingsprogrammer ikke automatisk om-formatterer teksten, således at hullerne bliver fyldt ud osv. Derfor kan brugeren kalde en formatteringsfunktion. I den enkleste udgave sker formatteringen ved, at det undersøges om der er plads til første ord i den anden linie i linie et, er det tilfældet flyttes ordet op. Resultatet bliver alt i alt en pænere tekst – også selvom der ikke indsættes mellemrum i linierne, således at der bliver 'lige højrekant'. Dette modul udfører formatteringen af den tekst der ligger i TEXT$, ved hjælp af den afskæring af strenge, som vi så på allerede i kapitel 1.

Modul 4.3.6: Linie 16000-16310

```
16ØØØ REM ****************************
16Ø1Ø REM Formattering af linie
16Ø2Ø REM ****************************
16Ø3Ø FOR i=1 TO si-2
16Ø4Ø d=Ø:WHILE text$(i)<>"" AND text$(i
+1)<>"" AND text$(i)<>SPACE$(32) AND tex
t$(i+1)<>SPACE$(32) AND d=Ø
16Ø5Ø :
16Ø6Ø :
16Ø7Ø sp=32-LEN(text$(i))
16Ø8Ø ord=INSTR(text$(i+1)," ")
16Ø9Ø IF ord=Ø THEN ord=LEN(text$(i+1))+
1
161ØØ IF ord>sp THEN d=1
1611Ø :
1612Ø :
1613Ø d2=Ø:WHILE sp>=ord AND sp<=LEN(tex
t$(i+1)) AND d2=Ø
1614Ø text$(i)=text$(i)+" "+LEFT$(text$(
i+1),ord-1)
1615Ø text$(i+1)=MID$(text$(i+1),ord+1)
1616Ø d2=1:WEND
1617Ø :
1618Ø :
1619Ø sp=32-LEN(text$(i))
162ØØ d2=Ø:WHILE LEN(text$(i+1))<sp AND
d=Ø AND d2=Ø
1621Ø text$(i)=text$(i)+" "+text$(i+1)
1622Ø FOR j=i+1 TO si
1623Ø text$(j)=text$(j+1)
1624Ø NEXT j
1625Ø si=si-1:pc=pc-1
1626Ø d2=1:WEND
1627Ø :
1628Ø :
1629Ø WEND
163ØØ NEXT i
1631Ø RETURN
```

Kommentar

Linie 16030-16300: Formatteringen starter med den første linie og kører hele teksten igennem.

Linie 16040-16290: Operationen udføres kun for linier der indeholder noget. F.eks. kan det tænkes, at der ved slutningen af et afsnit findes en linie, hvori der er mindre end 32 karakterer - vi ønsker ikke at begyndelsen på det næste afsnit hæfter sig på lige efter denne linie. Dette undgås ved at brugeren sætter en tom linie ind efter slutningen på afsnittet. Formateringsmodulet vil hverken tilføje denne tomme linie til den tidligere linie eller kopiere nogle af de ord, der er i næste linie, over i den tomme linie.

Linie 16070: Variablen SP angiver hvor meget plads, der er for enden af den linie, der undersøges.

Linie 16080: ORD angiver længden af det første ord i næste linie – dvs. antallet af karakterer frem til det første mellemrum.

Linie 16090: Hvis der slet ingen mellemrum er i den følgende linie, sættes ORD til længden af hele linien.

Linie 16100: Hvis længden af første ord i næste linie er større end den overskydende plads i linien, er der ingen grund til at fortsætte undersøgelsen. WHILE løkken afsluttes derfor og FOR løkken går videre med den næste linie i teksten.

Linie 16130-16160: Disse linier aktiveres hvis der *er* plads til det første ord i næste linie, men ikke til hele linien. Ordet bliver så kopieret op i den første linie og 'skåret fra' i den anden linie.

Linie 16190-16260: Det er muligt, at der i linien er plads til hele den næste linie. Hvis det er tilfældet, er det ikke nok blot at kopiere den anden linie op i den første – hele arrayet skal flyttes en linie op, således at den tomme plads udfyldes, ellers vil der komme en uoverensstemmelse med instruktionen ovenfor om, at tomme linier skal blive stående.

Test

Lav den samme test igen med fire grupper bogstaver. Når de alle står oppe i hovedteksten, trykker du først på '↑' og derefter på 'F'. Efter en pause (som kan være ret lang, hvis der er meget tekst), kommer hovedteksten frem igen, og denne gang skal de fire grupper bogstaver være samlet på en linie.

Modul 4.3.7: Gemning og hentning af tekst

Her bruges to standardmoduler. Bemærk at der bruges LINE INPUT i stedet for INPUT, det sikrer at vi kan bruge kommaer mv., som INPUT ikke kan håndtere.

Derudover har dette modul vist mig at der må være en eller anden skjult fejl ved Amstrad'en – eller i hvert fald ved nogle eksemplarer. Fejlen kan ikke ses direkte af brugeren, men den viser sig ved at strengen FI$ på en eller anden måde ændres, således at maskinen ikke kan genkende det filnavn, der blev brugt ved gemningen, selvom de to er ens når de udskrives. Hvis du indlæser linien:

19075 FI$=FI$

er problemet løst, – i hvert fald i dette program. Hvis programmet fungerer fint uden denne tilføjelse, så skal du blot se bort fra denne lille kommentar.

Modul 4.3.7: Linie 18000-19150

```
18ØØØ REM ******************************
18Ø1Ø REM Gem på bånd
18Ø2Ø REM ******************************
18Ø3Ø q$="":WHILE LOWER$(q$)<>"j"
18Ø4Ø CLS:INPUT "Fil navn: ",fi$:fi$=UPP
ER$(fi$)
18Ø5Ø PRINT "Filen der skal gemmes er: "
;fi$
18Ø6Ø INPUT "Er det rigtigt (j/n)";q$
18Ø7Ø WEND
18Ø8Ø OPENOUT fi$
18Ø9Ø PRINT #9,pc
181ØØ PRINT #9,si
1811Ø FOR i=Ø TO si
1813Ø PRINT #9,text$(i)
1814Ø NEXT i
1815Ø CLOSEOUT
1816Ø RETURN
19ØØØ REM ******************************
19Ø1Ø REM Hent fra bånd
19Ø2Ø REM ******************************
19Ø3Ø q$="":WHILE LOWER$(q$)<>"j"
```

```
19Ø4Ø PRINT:INPUT "Fil navn: ",fi$:fi$=U
PPER$(fi$)
19Ø5Ø PRINT "Filen der skal hentes er: "
;fi$
19Ø6Ø INPUT "Er det rigtigt (j/n)";q$
19Ø7Ø WEND
19Ø8Ø OPENIN fi$
19Ø9Ø INPUT #9,pc
191ØØ INPUT #9,si
1911Ø FOR i=Ø TO si
1912Ø LINE INPUT #9,text$(i)
1913Ø NEXT i
1914Ø CLOSEIN
1915Ø RETURN
```

Modul 4.3.8: Kontrolmodulet

Et standardmodul, hvor linie 10030 kalder et modul, der oplyser om de forskellige funktioner.

Modul 4.3.8: Linie 10000-10060

```
1ØØØØ REM ****************************
1ØØ1Ø REM Kontrolløkke
1ØØ2Ø REM ****************************
1ØØ3Ø GOSUB 2ØØØØ
1ØØ4Ø IF in=Ø THEN GOSUB 11ØØØ
1ØØ5Ø GOSUB 14ØØØ
1ØØ6Ø GOTO 12ØØØ
```

Modul 4.3.9: Udskrift på printer

Hvis du har en printer – og programmet her er jo ikke meget værd uden – så sørger dette modul for, at den tekst du har skrevet, kan komme ud på papir. Derudover giver modulet mulighed for at se teksten på skærmen, som den vil blive udskrevet, i mode 2 med plads til 80 tegn per linie. På den måde kan du arbejde med teksten i mode 1, som er rarest at bruge (i hvert fald på en farveskærm), og alligevel se det endelige resultat *inden* det kommer på papiret.

Modul 4.3.9: Linie 17000-17260

```
17000 REM ****************************
17010 REM Udskrift på printer
17020 REM ****************************
17030 channel=8
17040 IF t1$="v" THEN channel=0:PEN 1:MO
DE 2
17050 x=1
17060 WHILE x<si
17070 d=0
17080 :
17090 :
17100 IF text$(x)="" THEN PRINT #channel
: d=1
17110 IF d=0 THEN PRINT #channel,SPACE$(
8);text$(x);" ";
17120 x=x+1
17130 :
17140 :
17150 WHILE x<si AND d=0
17160 PRINT #channel,text$(x)
17170 IF text$(x)="" THEN PRINT #channel
17180 x=x+1
17190 IF channel=0 THEN IF INKEY$="" THE
N GOTO 17190
17200 d=1:WEND
17210 :
17220 :
17230 WEND
17240 PRINT #channel
17250 IF channel=0 THEN IF INKEY$="" THE
N GOTO 17250
17260 MODE 1 : RETURN
```

Kommentar

Linie 17030-17040: Dette modul kaldes hvad enten brugeren vil se teksten på papir eller på skærmen i mode 2. Forskellen er at udskriften på skærmen kommer via datastrøm 0, mens printerudskriften kommer via datastrøm 8.

Linie 17050: Variablen X registrerer hvor langt udskriften er nået i linierne i TEXT$.

Linie 17070: Variablen D bruges til at springe rundt mellem modulets forskellige funktioner, når behandlingen af en tekstlinie er slut.

Linie 17100: Ligesom ved formatteringen skal de tomme linier respekteres – når programmet støder på en, bruges en tom PRINT kommando, således at udskriften flyttes en linie ned. Variablen D sættes til 1 for at markere at en tekstlinie er færdig.

Linie 17110: Hvis linien ikke er tom bliver den udskrevet som første halvdel af en linie med 64 karakterer, efterfulgt af et mellemrum. Bemærk semikolonnet, som sikrer at de næste karakterer bliver udskrevet i samme linie.

Linie 17150-17200: Den anden halvdel af linien udskrives. Hvis det er en tom linie, sørger et ekstra PRINT for at flytte udskriften en linie ned. Hvis udskriften sendes til skærmen, skal brugeren trykke på en tast for hver ny linie. Det sikrer at brugeren kan nå at se teksten, selvom den skulle fylde mere end der kan være på skærmen på én gang.

Linie 17250: Hvis udskriften kommer på skærmen venter denne linie på endnu et tryk på en tast, når teksten er færdig, før mode 2 forlades. Det sikrer at brugeren ikke kommer til at forlade mode 2 udskriften, selvom der ikke er flere linier at udskrive.

Modul 4.3.10: Instruktion/hjælp

Dette modul kaldes først, når programmet køres, og giver en udskrift på skærmen, som fortæller om de forskellige funktioner. Ved et tryk på en tast kører programmet videre. Hvis du ikke kan huske funktionerne, kan du senere i programmet kalde dette modul ved at taste 'H' (Hjælp) i redigeringsmode, og ved et tryk på en tast vende tilbage til hvor du var.

Modul 4.3.10: Linie 20000-20220

```
20000 REM ****************************
20010 REM Instruktion
20020 REM ****************************
20030 MODE 2:PEN 1:PRINT "H J Æ L P : FU
NKTIONSTASTER - TEKSTBEHANDLER":PRINT
20040 PRINT "Ved tekstskrivning:":PRINT
```

```
2ØØ5Ø PRINT "ENTER          Indlæser ny te
kst i hovedteksten"
2ØØ6Ø PRINT "|              Flytter markør
 til begyndelse eller slutning af linie"
2ØØ7Ø PRINT "DEL            Sletter tegn t
il venstre for markøren"
2ØØ8Ø PRINT "MARKØR PILE    Flytter markør
 til højre eller venstre"
2ØØ9Ø PRINT "^              Skift til redi
gerings-mode":PRINT
2Ø1ØØ PRINT "Ved redigering:":PRINT
2Ø11Ø PRINT "ENTER          Retur til teks
tskrivning"
2Ø12Ø PRINT "MARKØR PILE    Flytter markør
 en linie op eller ned"
2Ø13Ø PRINT "O el. N        Flytter markør
 ti linier op eller ned"
2Ø14Ø PRINT "DEL            Sletter linien
 lige under redigeringsmarkøren"
2Ø15Ø PRINT "K              Kopierer linie
n lige under redigeringsmarkøren"
2Ø16Ø PRINT "P              Udskriver hove
dteksten på printer"
2Ø17Ø PRINT "G              Gemmer hovedte
ksten på bånd"
2Ø18Ø PRINT "F              Formatterer ho
vedteksten"
2Ø19Ø PRINT "V              Viser teksten
i mode 2 (8Ø tegn per linie)"
2Ø2ØØ PRINT "H              F O R   H J Æ
L P":PRINT:PRINT
2Ø21Ø PRINT "- tryk på en tast for at fo
rtsætte"
2Ø22Ø WHILE INKEY$="":WEND:MODE 1:RETURN
```

Test

Hvis du har en printer, skal du sørge for at den er koblet til og tændt (husk altid, at både printer og computer skal være slukket, når du kobler dem sammen). Skriv nu nogle tekstlinier og tryk først '^' og derefter 'P'. Nu skal udskriften komme på printeren. Hvis testen er OK er programmet klar til brug.

```
H J Æ L P : FUNKTIONSTASTER - TEKSTBEHANDLER

Ved tekstskrivning:

ENTER         Indlæser ny tekst i hovedteksten
\             Flytter markør til begyndelse eller slutnin
              af linie
DEL           Sletter tegn til venstre for markøren
MARKØR PILE   Flytter markør til højre eller venstre
^             Skift til redigerings-mode

Ved redigering:

ENTER         Retur til tekstskrivning
MARKØR PILE   Flytter markør en linie op eller ned
O el. N       Flytter markør ti linier op eller ned
DEL           Sletter linien lige under redigeringsmarkør
K             Kopierer linien lige under redigeringsmarkø
P             Udskriver hovedteksten på printer
G             Gemmer hovedteksten pÅ bånd
F             Formatterer hovedteksten
V             Viser teksten i mode 2 (80 tegn per linie)
H             F O R   H J Æ L P

- tryk på en tast for at fortsætte
```

Fig. 4.4: 'Hjælp'. I TEKST udskrives hjælpen når programmet starter, og hjælpen kan også kaldes når programmet kører.

Program 4.4: Quiz

Det sidste program i dette kapitel er et spørgeprogram, der både kan bruges til undervisningsformål eller til almindelig underholdning – 'QUIZ'. Jeg ved ikke, hvor meget man faktisk kan lære af et program som dette, men det er skægt at bruge, så det kan være svært at stoppe igen. En tidligere udgave af dette program har forøvrigt været anvendt af en engelsk radiostation i en lytterquiz.

QUIZ er meget bredt anvendeligt, det ene øjeblik kan det bruges til indlæring af fremmede gloser, og fem minutter senere kan det bruges til at spørge om historiske årstal. Det sker ved at programmet tilfældigt finder et spørgs-

mål, og giver dig fem svarmuligheder, hvoraf kun én er rigtig (en såkaldt *multiple choise* opgave). Programmet fører pointregnskab, så du kan se, hvor skrap du er til at svare på det valgte emne. Inden programmet kan bruges skal brugeren indlæse spørgsmål og svar – det kan programmet ikke selv klare. Programmet fungerer bedst, hvis der indlæses mange spørgsmål, så de samme ikke dukker op hele tiden.

Modul 4.4.1: Initialisering

Modul 4.4.1: Linie 10000-10100

```
10000 REM ****************************
10010 REM Initialisering
10020 REM ****************************
10030 DIM navn$(1),tt(4),array$(499,1),s
type$(9),atype(1,9)
10040 rigtigt=0:total=0:pt=0
10050 MODE 1
10060 PEN 2:PRINT TAB(18);"QUIZ":PRINT T
AB(18);"===="
10070 WINDOW 1,40,4,25
10080 PEN 1
10090 INPUT "Hent fra bånd (j/n)";q$
10100 IF LOWER$(q$)="j" THEN GOSUB 23000
 ELSE GOSUB 24000
```

Kommentar

Linie 10030: Arrayet NAVN$ indeholder de navne brugeren giver hhv. spørgsmål og svar, TT bruges når der laves tilfældige spørgsmål.

ARRAY$ indeholder alle svarene og spørgsmålene, STYPE$ indeholder navnene på de forskellige spørgsmålstyper, som kan anvendes, og endelig indeholder ATYPE antallet af hver type i ARRAY$.

Modul 4.4.2: Quiz'ens opbygning

QUIZ stiller, som allerede nævnt, et spørgsmål og angiver fem mulige svar. Disse linier giver brugeren mulighed for at give spørgsmål og svar nogle overordnede navne. Hvis formålet med programmet f.eks. er at lære engelske gloser kan spørgsmålene hedde 'DANSK ORD' og svarene 'ENGELSK OVERSÆTTELSE'.

Fig. 4.5: Skærmdump fra QUIZ

Modul 4.4.2: Linie 24000-24130

```
24000 REM ****************************
24010 REM Quiz'ens opbygning
24020 REM ****************************
24030 q$="":WHILE LOWER$(q$)<>"j"
24040 CLS
24050 PRINT "Quiz'ens opbygning:":PRINT
24060 INPUT "Svarets navn: ",navn$(0)
24070 INPUT "Spørgsmålet navn: ",navn$(1
)
24080 PRINT:INPUT "Er de rigtige (j/n) "
;q$
24090 WEND
24100 stype$(0)="Ingen type"
24110 atyper=1
24120 GOSUB 12000
24130 RETURN
```

Modul 4.4.3: Fejlmeldinger

Et standardmodul til udskrivning af fejlmeldinger med advarselstone.

Modul 4.4.3: Linie 25000-25070

```
25ØØØ REM ****************************
25Ø1Ø REM Fejl
25Ø2Ø REM ****************************
25Ø3Ø PRINT:PRINT x$
25Ø4Ø SOUND 1,1ØØØ,1ØØ
25Ø5Ø FOR i=1 TO 15ØØ
25Ø6Ø NEXT i
25Ø7Ø RETURN
```

Modul 4.4.4: Menuen

Et standard menumodul.

Modul 4.4.4: Linie 11000-11210

```
11ØØØ REM ****************************
11Ø1Ø REM Hovedmenu
11Ø2Ø REM ****************************
11Ø3Ø WHILE z<>7
11Ø4Ø CLS
11Ø5Ø PRINT"MULIGE KOMMANDOER:"
11Ø6Ø PRINT
11Ø7Ø PRINT "1) Indlæs nye poster"
11Ø8Ø PRINT "2) Indlæs nye typer"
11Ø9Ø PRINT "3) Søg/Slet"
111ØØ PRINT "4) Træk spørgsmål"
1111Ø PRINT "5) Vis eller nulstil pointr
egnskab"
1112Ø PRINT "6) Gem data på bånd"
1113Ø PRINT "7) Stop"
1114Ø PRINT
1115Ø INPUT"Indtast dit valg: ",z
1116Ø IF z>2 AND z<7 AND pt=Ø THEN x$="
   ****** ENDNU INGEN DATA ******":GOSUB
 25ØØØ:z=Ø
```

```
11170 ON z GOSUB 13000,12000,20000,17000
,19000,22000
11180 WEND
11190 CLS
11200 LOCATE 13,11:PRINT "QUIZ'EN ER SLU
T"
11210 END
```

Modul 4.4.5: Spørgsmålstyper

Når du indlæser de følgende moduler, vil du opdage at QUIZ kan bruges med to forskellige sværhedsgrader. Det sker på basis af spørgsmålstyperne. Hvis vi vender tilbage til eksemplet med de engelske gloser, er det klart at disse kan inddeles i en række undergrupper som f.eks. navneord, udsagnsord og biord. Hvis det engelske ord der spørges om er et udsagnsord, og kun én af fem svarmuligheder er et udsagnsord, så er opgaven jo noget lettere at løse end hvis alle fem svarmuligheder er udsagnsord.

Med dette modul kan brugeren angive op til 10 forskellige spørgsmålstyper, og derefter ved indlæsning af spørgsmål og svar at oplyse hvilken type indlæsningen tilhører. I et senere modul gives der mulighed for at vælge om svarene skal tages fra alle typer, eller én type.

Modul 4.4.5: Linie 12000-12150

```
12000 REM ******************************
12010 REM Nye typer
12020 REM ******************************
12030 q$="":WHILE 1
12040 CLS
12050 PRINT "Typer:":PRINT
12060 PRINT "Typer indtil videre: ":PRIN
T
12070 FOR i=0 TO atyper-1
12080 PRINT i+1;") ";stype$(i)
12090 NEXT i
12100 IF atyper=10 THEN x$="    ** IKKE
PLADS TIL FLERE TYPER **":GOSUB 25000:RE
TURN
12110 PRINT:INPUT "Indlæs ny type ('|' f
or retur): ";q$
12120 IF q$="|" THEN RETURN
```

```
12130 stype$(atyper)=q$
12140 atyper=atyper+1
12150 WEND
```

Test

Du skal nu kunne køre programmet og indlæse indtil 10 spørgsmålstyper. Du kan checke om typerne er blevet indlæst ved at stoppe programmet og skrive:

FOR I= 0 TO 9:PRINT STYPE$(I):NEXT < ENTER >

Modul 4.4.6: Binær søgning

Et standardmodul, der arbejder alfabetisk på grundlag af svarene på spørgsmålene. foran hvert svar står en enkelt karakter (0-9), som indikerer hvilken type det er. Filen bliver først sorteret efter type og derefter – indenfor hver type – alfabetisk.

Modul 4.4.6: Linie 14000-14110

```
14000 REM ****************************
14010 REM Binær søgning
14020 REM ****************************
14030 IF pt=0 THEN ss=0:RETURN
14040 po=INT(LOG(pt)/LOG(2)):ss=2^po-1
14050 FOR i=po TO 0 STEP -1
14060 ss=ss+2^i*((array$(ss,0)>t1$)-(arr
ay$(ss,0)<t1$))
14070 IF ss<0 THEN ss=0
14080 IF ss>pt-1 THEN ss=pt-1
14090 NEXT i
14100 IF array$(ss,0)<t1$ THEN ss=ss+1
14110 RETURN
```

Modul 4.4.7: Indsættelse af en post

Et standard indsættelsesmodul.

Modul 4.4.7: Linie 15000-15110

```
15ØØØ REM ****************************
15Ø1Ø REM Indsæt post
15Ø2Ø REM ****************************
15Ø3Ø FOR i=pt TO ss+1 STEP -1
15Ø4Ø array$(i,Ø)=array$(i-1,Ø)
15Ø5Ø array$(i,1)=array$(i-1,1)
15Ø6Ø NEXT i
15Ø7Ø array$(ss,Ø)=t1$
15Ø8Ø array$(ss,1)=t2$
15Ø9Ø pt=pt+1
151ØØ GOSUB 16ØØØ
1511Ø RETURN
```

Modul 4.4.8: Opdatering af typerne

Vi har allerede indlæst et modul, som gør det muligt at lave nye typer, men vi skal også sørge for at programmet kan registrere hvor mange af hver type, der er i filen, og hvor hver type starter.

Registreringen af hvor mange af hver type der er, sker simpelthen ved at lægge 1 til det pågældende element i arrayet ATYPE. Dette modul kaldes hver gang, der oprettes eller slettes en post. Formålet med det er, at lade den anden side af ATYPE registrere det samlede antal af hver af typerne fra 0 til 9. Svarene vil så blive sorteret i efter type i ARRAY$, og når vi kender det samlede antal poster, som falder i type 0, 1 og 2, kan vi derud fra vide hvor posterne i type 3 starter.

Modul 4.4.8: Linie 16000-16080

```
16ØØØ REM ****************************
16Ø1Ø REM Opdatering
16Ø2Ø REM ****************************
16Ø3Ø su=Ø
16Ø4Ø FOR i=Ø TO 9
16Ø5Ø atype(1,i)=su
16Ø6Ø su=su+atype(Ø,i)
16Ø7Ø NEXT i
16Ø8Ø RETURN
```

Modul 4.4.9: Indlæsning af en ny post

Et ret enkelt modul, der lader brugeren indlæse svar, spørgsmål og typenummer.

Modul 4.4.9: Linie 13000-13300

```
13ØØØ REM ****************************
13Ø1Ø REM Nye poster
13Ø2Ø REM ****************************
13Ø3Ø WHILE 1
13Ø4Ø CLS
13Ø5Ø PRINT "Nye poster:":PRINT
13Ø6Ø IF pt=5ØØ THEN LET x$="    *******
 IKKE MERE PLADS *******":GOSUB 25ØØØ:RE
TURN
13Ø7Ø PRINT"Indlæs posten"
13Ø8Ø PRINT"'|' for retur til hovedmenue
n"
13Ø9Ø PRINT
131ØØ PRINT navn$(Ø);
1311Ø INPUT ":";t1$
1312Ø IF t1$="|" THEN RETURN
1313Ø PRINT navn$(1);
1314Ø INPUT ":";t2$
1315Ø IF t2$="|" THEN RETURN
1316Ø PRINT:PRINT "Typer:":PRINT
1317Ø FOR i=Ø TO atyper-1
1318Ø PRINT i+1;") ";stype$(i)
1319Ø NEXT i
132ØØ PRINT:INPUT "Indtast typens nummer
: ",t3
1321Ø t3=t3-1
1322Ø IF t3<Ø OR t3>atyper THEN t3=Ø
1323Ø PRINT:INPUT "Er det rigtigt (j/n)
";q$
1324Ø WHILE LOWER$(q$)="j"
1325Ø atype(Ø,t3)=atype(Ø,t3)+1
1326Ø t1$=MID$(STR$(t3)+t1$,2)
1327Ø GOSUB 14ØØØ
1328Ø GOSUB 15ØØØ
1329Ø q$="":WEND
133ØØ WEND
```

Kommentar

Linie 13250: Det element i ATYPE, som repræsenterer det aktuelle spørgsmåls (og svars) type opskrives med 1.

Test

Kør programmet, angiv 3 typer og vælg derefter '1' på menuen, så du kan indlæse spørgsmål og svar. For at checke at posterne bliver modtaget rigtigt, indlæser du følgende data:

SVAR	SPØRGSMÅL	TYPE
A111	Q111	3
A222	Q222	2
A333	Q333	1

og stop programmet. Skriv nu:

FOR I= 0 TO 2:FOR J= 0 TO 1:PRINT ARRAY$(I,J):NEXT J:NEXT I
< ENTER >

Nu skal du se følgende:

0A333
Q333
1A222
Q222
2A111
Q111

Modul 4.4.10: Lagring af data

Da vi nu kan indlæse data, er det på tide også at indlæse de to standardmoduler, som kan gemme og hente data.

Modul 4.4.10: Linie 22000-23220

```
22ØØØ REM ****************************
22Ø1Ø REM Gem på bånd
22Ø2Ø REM ****************************
22Ø3Ø CLS
22Ø4Ø PRINT "Gem data:":PRINT
22Ø5Ø INPUT "Navn på filen: ",fil$
```

```
22Ø6Ø OPENOUT fil$
22Ø7Ø PRINT #9,pt
22Ø8Ø PRINT #9,atyper
22Ø9Ø FOR i=Ø TO 1
221ØØ PRINT #9,navn$(i)
2211Ø FOR j=Ø TO atyper-1
2212Ø PRINT #9,atype(i,j)
2213Ø NEXT j
2214Ø FOR j=Ø TO pt-1
2215Ø PRINT #9,array$(j,i)
2216Ø NEXT j
2217Ø NEXT i
2218Ø FOR i=Ø TO atyper-1
2219Ø PRINT #9,stype$(i)
222ØØ NEXT i
2221Ø CLOSEOUT
2222Ø RETURN
23ØØØ REM ****************************
23Ø1Ø REM Hent fra data bånd
23Ø2Ø REM ****************************
23Ø3Ø CLS
23Ø4Ø PRINT "Hent data:":PRINT
23Ø5Ø INPUT "Navn på filen: ",fil$
23Ø6Ø OPENIN fil$
23Ø7Ø INPUT #9,pt
23Ø8Ø INPUT #9,atyper
23Ø9Ø FOR i=Ø TO 1
231ØØ INPUT #9,navn$(i)
2311Ø FOR j=Ø TO atyper-1
2312Ø INPUT #9,atype(i,j)
2313Ø NEXT j
2314Ø FOR j=Ø TO pt-1
2315Ø INPUT #9,array$(j,i)
2316Ø NEXT j
2317Ø NEXT i
2318Ø FOR i=Ø TO atyper-1
2319Ø INPUT #9,stype$(i)
232ØØ NEXT i
2321Ø CLOSEIN
2322Ø RETURN
```

Modul 4.4.11: Brugersøgning

Dette enkle søgemodul, gør det muligt at 'bladre' frem og tilbage i posterne i hovedfilen, og – når næste modul er indlæst – at slette poster.

Modul 4.4.11: Linie 20000-20220

```
20000 REM ****************************
20010 REM Søgning
20020 REM ****************************
20030 ss=0
20040 t1$="":WHILE t1$<>"|" AND pt>0
20050 CLS:PRINT "Søgning:":PRINT
20060 PRINT "Post nummer:";ss+1
20070 PRINT navn$(0);": ";MID$(array$(ss
,0),2)
20080 PRINT navn$(1);": ";array$(ss,1)
20090 PRINT "Type: ";stype$(VAL(LEFT$(ar
ray$(ss,0),1)))
20100 PRINT:PRINT "Mulige kommandoer:":P
RINT
20110 PEN 2:PRINT "ENTER";:PEN 1:PRINT "
 for næste post"
20120 PRINT "'#' og et tal for at spring
e i filen"
20130 PRINT "'s' for at slette en post"
20140 PRINT "'|' for retur"
20150 PRINT:INPUT "Hvilken vælger du: ";
t1$
20160 IF t1$="" THEN t1$="#1"
20170 IF LOWER$(t1$)="s" THEN GOSUB 2100
0
20180 IF LEFT$(t1$,1)="#" THEN ss=ss+VAL
(MID$(t1$,2))
20190 IF ss>pt-1 THEN ss=pt-1
20200 IF ss<0 THEN ss=0
20210 WEND
20220 RETURN
```

Test

Kør programmet gem data på bånd og hent dem igen. Prøv at 'bladre' gennem filen ('3' på menuen).

Modul 4.4.12: Slet en post

Et standard slettemodul.

Modul 4.4.12: Linie 21000-21100

```
21ØØØ REM ****************************
21Ø1Ø REM Slet
21Ø2Ø REM ****************************
21Ø3Ø atype(Ø,VAL(LEFT$(array$(ss,Ø),1))
)=atype(Ø,VAL(LEFT$(array$(ss,Ø),1)))-1
21Ø4Ø FOR i=ss TO pt-2
21Ø5Ø array$(i,Ø)=array$(i+1,Ø)
21Ø6Ø array$(i,1)=array$(i+1,1)
21Ø7Ø NEXT i
21Ø8Ø pt=pt-1
21Ø9Ø GOSUB 16ØØØ
211ØØ RETURN
```

Test

Hent data fra båndet, vælg '3' på menuen og prøv at slette nogle poster.

Modul 4.4.13: Spørgsmålene stilles

Vi kommer nu til det, der er det nye i QUIZ – spørgsmålene stilles og svarmulighederne oplyses. Dette modul tager sig af det der vises på skærmen, mens det næste 'trækker' de tilfældige spørgsmål.

Modul 4.4.13: Linie 17000-17430

```
17ØØØ REM ****************************
17Ø1Ø REM Spørgsmål
17Ø2Ø REM ****************************
17Ø3Ø CLS
17Ø4Ø PRINT "SPØRGSMÅL:":PRINT
17Ø5Ø PRINT "Vil du have de mulige svar
trukket fra"
17Ø6Ø PRINT "kun een type (svært), eller
 fra alle"
17Ø7Ø PRINT "typerne (lettere)?"
17Ø8Ø PRINT:PRINT "1) Kun een type"
```

```
17Ø9Ø PRINT "2) Alle typer"
171ØØ PRINT:INPUT "Hvilken: ",spt
1711Ø IF spt<1 OR spt>2 THEN spt=2
1712Ø q$="":WHILE LOWER$(q$)<>"n"
1713Ø GOSUB 18ØØØ
1714Ø CLS
1715Ø PRINT navn$(1);":";array$(tt(spmu)
,1)
1716Ø PRINT:PRINT navn$(Ø);":"
1717Ø FOR i=Ø TO 4
1718Ø PRINT i+1;") ";MID$(array$(tt(i),Ø
),2)
1719Ø NEXT i
172ØØ bs=Ø:WHILE bs<1 OR bs>5
1721Ø PRINT:INPUT "Hvilket svar er det r
igtige (1-5)";bs
1722Ø WEND
1723Ø WHILE bs-1=spmu
1724Ø PEN 3:PRINT
1725Ø PRINT "**********************"
1726Ø PRINT "*                     *"
1727Ø PRINT "* R I G T I G T !   *"
1728Ø PRINT "*                     *"
1729Ø PRINT "**********************"
173ØØ FOR i=8ØØ TO 1ØØ STEP -1ØØ
1731Ø SOUND 1,i,1Ø
1732Ø NEXT i
1733Ø PEN 1
1734Ø rigtigt=rigtigt+1
1735Ø bs=Ø:WEND
1736Ø WHILE bs-1<>spmu AND bs>Ø
1737Ø PRINT:PRINT "Det er desværre forke
rt. Det rigtige"
1738Ø PRINT "svar er ";MID$(array$(hosp,
Ø),2);"."
1739Ø bs=Ø:WEND
174ØØ total=total+1
1741Ø PRINT:INPUT "Flere spørgsmål (j/n)
 ";q$
1742Ø WEND
1743Ø RETURN
```

Kommentar

Linie 17040-17100: Vi har allerede nævnt at QUIZ kan have to sværhedsgrader. Disse linier giver brugeren mulighed for at bestemme, om de mulige svar skal tages blandt alle typer eller fra samme type som spørgsmålet.

Linie 17150-17190: Spørgsmålet og de fem svarmuligheder, som findes af det næste modul, udskrives. Placeringerne af de fem svarmuligheder indeholdes i TT, mens det rigtige svar indenfor TT ligger i variablen SPMU.

Linie 17230-17350: Hvis brugerens svar, BS, svarer til placeringen af det rigtige svar (SPMU), så udskrives ordet 'RIGTIGT !' samtidig med at en lille fanfare lyder. Variablen RIGTIGT, som registrerer antallet af rigtige svar, opskrives med 1. Hvis svaret er forkert, får brugeren det at vide, og det oplyses hvad det rigtige svar er.

Linie 17400: TOTAL, variablen, der registrerer det samlede antal spørgsmål, opskrives med 1.

Modul 4.4.14: Spørgsmålene 'trækkes'

Nu kommer vi så til det vanskeligste – den tilfældige udvælgelse af spørgsmål og svar. Før vi kommenterer modulet i detaljer, skal vi se lidt på den metode, der bruges.

Vi skal trække et tilfældigt spørgsmål med tilhørende svar og derefter indlæse fem tal i arrayet TT, som står for placeringerne af svarmulighederne i hovedfilen – og en af dem skal være det rigtige svar. Hovedspørgsmålet med tilhørende svar vælges tilfældigt blandt samtlige poster i filen, og deres placering i hovedfilen indsættes et tilfældigt sted i arrayet TT.

Når hovedspørgsmålet er placeret i TT, skal der findes fire andre svarmuligheder. Afhængigt af, om brugeren vil have en nem eller en svær opgave, trækkes svaret enten fra hele filen eller indenfor den type, som spørgsmålet tilhører. De fire svar trækkes derefter, og det checkes at samme svar ikke optræder flere gange, og at ingen svar er magen til det rigtige svar.

Modul 4.4.14: Linie 18000-18210

```
18ØØØ REM ****************************
18Ø1Ø REM Tilfældig trækning
18Ø2Ø REM ****************************
18Ø3Ø hosp=INT(RND*(pt))
18Ø4Ø ctype=VAL(LEFT$(array$(hosp,Ø),1))
18Ø5Ø r1=atype(1,ctype)
18Ø6Ø r2=atype(Ø,ctype)
18Ø7Ø IF r2<5 OR spt=2 THEN r1=Ø:r2=pt
18Ø8Ø spmu=INT(RND*5)
18Ø9Ø FOR i=Ø TO 4
181ØØ duff=hosp
1811Ø dup=1:WHILE dup=1 AND i<>spmu
1812Ø dup=Ø
1813Ø duff=r1+INT(RND*r2)
1814Ø IF MID$(array$(duff,Ø),2)=MID$(arr
ay$(hosp,Ø),2) THEN dup=1
1815Ø FOR j=Ø TO i-1
1816Ø IF MID$(array$(duff,Ø),2)=MID$(arr
ay$(tt(j),Ø),2) THEN dup=1
1817Ø NEXT j
1818Ø WEND
1819Ø tt(i)=duff
182ØØ NEXT i
1821Ø RETURN
```

Kommentar

Linie 18030-18040: Hovedspørgsmålet og svaret trækkes og lægges i variablen HOSP. Svarets type registreres af variablen CTYPE.

Linie 18050-18070: De to variabler R1 og R2 registrerer begyndelsen og slutningen af den del af filen, som spørgsmålene skal trækkes fra. Hvis brugeren har valgt 'den svære', så bliver R1 og R2 sat til at pege på begyndelse og slutning af den type spørgsmål, som hovedspørgsmålet tilhører. Hvis det viser sig at der er mindre end fem forskellige svar, – eller hvis brugeren vælger 'den lette', så sættes R1 og R2 til at pege på begyndelse og slutning af hovedfilen. Hvis du vælger 'den svære' og programmet alligevel giver dig 'den lette', så er forklaringen sandsynligvis, at der ikke er nok svar af samme type som hovedspørgsmålet.

Linie 18080: SPMU er det korrekte svars placering blandt de fem svarmuligheder.

Linie 18090-18200: Denne løkke vælger fire andre svarmuligheder fra den del af filen, der er afmærket af R1 og R2, og hver af dem lagres midlertidigt i DUFF. Inde i løkken sammenlignes det nye svar med det rigtige svar, som kan ligge hvor som helst i TT, og med de svar, der tidligere er placeret i TT. Bemærk at det ikke er nok blot at checke om det samme svar i filen bruges to gange, for to forskellige spørgsmål kan godt have samme svar. Når løkken afsluttes indeholder TT placeringerne på fem forskellige svar i hovedfilen.

Test

Den eneste ordentlige måde at teste disse moduler på, er at indlæse en rigelig mængde data, så programmet kan afprøves i praksis. En god test kunne være at angive seks typer spørgsmål kaldet TYPE 1, TYPE 2 ... TYPE 6. Indlæse derefter en række svar A1, A2 ... A10 med tilhørende spørgsmål Q1, Q2 ... Q10. De første fem spørgsmål skal være TYPE 1 og de sidste TYPE 2 ... TYPE 6. Det giver mulighed for ét sæt spørgsmål af den svære slags, og fem andre med kun ét spørgsmål i hver.

Vælg nu '4' på menuen og vælg 'den svære'. Nu skal du kunne besvare spørgsmålene rigtigt og forkert, og derved checke at programmet reagerer, som det skal. Når spørgsmålet trækkes blandt de fem første, skal svarene ligge mellem 1 og 5. Vælger du 'den lette' skal svarene ligge mellem 1 og 10.

4.4.15: Pointregnskab

Endelig skal vi kunne lade programmet føre regnskab med pointgivningen. Det er ikke helt så let som man skulle tro, for det gælder ikke blot om at udregne den rigtige svarprocent (antal rigtige svar i procent af antallet af spørgsmål). For hvis brugeren hver gang f.eks. vælger det første svar, så vil der jo være et rigtigt svar i ca. 20% af tilfældene. Dvs. at en svarprocent på 20 godt kan betyde, at brugeren ikke har begrebet noget som helst. Vi har derfor valgt at trække en femtedel af antallet af spørgsmål (den del, der kunne være rigtig ved et rent tilfælde) fra antallet af rigtige svar, og derefter udtrykke dette tal i procent af fire femtedele af antallet af svar.

Modul 4.4.15: Linie 19000-19110

```
19ØØØ REM ****************************
19Ø1Ø REM Pointregnskab
19Ø2Ø REM ****************************
19Ø3Ø CLS
19Ø4Ø PRINT "POINT:":PRINT
19Ø5Ø IF total=Ø THEN x$="    ****** END
NU INGEN POINT ******":GOSUB 25ØØØ:RETUR
N
19Ø6Ø PRINT "Samlet antal svar:";total
19Ø7Ø PRINT "Samlet antal rigtige svar:"
;rigtigt
19Ø8Ø PRINT:PRINT "Points:";INT((rigtigt
-total/5)/(total*Ø.8)*1ØØ+Ø.5);"%"
19Ø9Ø PRINT:INPUT "Vil du nulstille poin
tregnskabet (j/n) ";q$
191ØØ IF LOWER$(q$)="j" THEN total=Ø:rig
tigt=Ø
1911Ø RETURN
```

Test

Lav testen for foregående modul igen. Når du har svaret på nogle få spørgsmål, så kald pointregnskabet via menuen. Nu skal din svarprocent se nogenlunde rimelig ud (det er svært helt at kontrollere dette). Derudover skal du kunne nulstille regnskabet.

Er begge dele i orden skulle programmet være klar til brug.

KAPITEL 5

Pengesager

I det sidste kapitel skal vi se på noget, som EDB bruges meget til, men som vi endnu ikke har beskæftiget os med – penge. Det er vigtigt at få det med, for mikrocomputere er virkelig fremragende til at håndtere den slags. Det er sjældent de helt store beløb, der indgår, og udregningerne er enkle – som regel skal der jo lægges til og trækkes fra.

Men fordelen ved den lille computer er ikke at den skal lægge til og trække fra, for det kan den menneskelige hjerne jo også. Nej, det som computeren er så fantastisk til er, at lagre oplysninger og hurtigt finde dem frem igen og vise det hele på en måde, der er til at forstå. De to programmer i dette kapitel giver eksempler på dette, det ene giver brugeren mulighed for at holde styr på bevægelserne på en bankkonto eller lignende, og det andet kan bruges til at føre et regnskab.

Det drejer sig om disse to programmer:

KONTO: Bruges til at holde styr på betalingerne ud og ind på en bankkonto, måned for måned.

BOGHOLDER: Bruges til at føre et regnskab på normal vis.

Program 5.1: Konto

Målet med dette program er, at give brugeren mulighed for at få et godt overblik over bevægelserne på en bankkonto, at specificere indtægter og udgifter og opdatere kontoen løbende. Der er mulighed for både at operere med faste udgifter/indtægter og med engangsbeløb. Programmet dækker alle årets 12 måneder.

```
                OKTOBER
-------------------------------------
DAG & OPLYSNINGER     BELØB      SALDO
-------------------------------------
   Transport                    8556.35
 1 afdrag/bil        1345.00-   7211.35
 5 husleje           3406.00-   3805.35
 5 A-kasse            563.00-   3242.35
 5 restskat          2055.00-   1187.35
12 telefon            512.20-    675.15
15 forsikring         767.00-     91.85-
25 løn               8604.00    8512.15
```

Fig. 5.1: Eksempel på printerudskrift fra programmet.

Modul 5.1.1: Initialisering

Et standard initialiseringsmodul.

Modul 5.1.1: Linie 10000-10160

```
1Ø13Ø DATA Juli,August,September,Oktober
,November,December
1Ø14Ø WEND
1Ø15Ø INK Ø,24:INK 1,3:INK 2,12
1Ø16Ø WINDOW #1,1,4Ø,16,25
1ØØØØ REM ****************************
1ØØ1Ø REM Initialisering
1ØØ2Ø REM ****************************
1ØØ3Ø MODE 1
1ØØ4Ø WHILE in=Ø
1ØØ5Ø in=1:cr$=CHR$(13):
1ØØ6Ø DIM a$(99,1),a(99,1):a(Ø,1)=999
1ØØ7Ø RESTORE
1ØØ8Ø DIM md$(11)
1ØØ9Ø FOR i=Ø TO 11
1Ø1ØØ READ md$(i)
1Ø11Ø NEXT i
1Ø12Ø DATA Januar,Februar,Marts,April,Ma
j,Juni
```

Kommentar

Linie 10060: Arrayet A$ bruges til at rumme navnene på betalingerne og en speciel streng, der vil blive forklaret senere, som angiver hvilken måned betalingen gælder. Talarrayet A rummer tilsvarende beløb og dato for hver betaling. A(0,1) sættes til 999 (en 'umulig' dato), så den senere kan bruges til at signalere filens slutning.

Linie 10080-10130: Denne løkke indlæser månedernes navne ind i arrayet MD$.

Modul 5.1.2: Programmenuen

Et standard menumodul. Hvis brugeren beder om udskrift, uden der er indlæst data, giver modulet endvidere besked om dette.

Modul 5.1.2: Linie 11000-11210

```
11ØØØ REM ****************************
11Ø1Ø REM Menu
11Ø2Ø REM ****************************
11Ø3Ø CLS:CLS #1:PEN 2:PRINT TAB(18);"KO
NTO";TAB(18);"=====":PEN 1:WINDOW 1,4Ø,4
,25
11Ø4Ø z=Ø:WHILE z<>6:CLS
11Ø5Ø PRINT "Mulige kommandoer:":PRINT
11Ø6Ø PRINT "1) Nye betalinger"
11Ø7Ø PRINT "2) Check/Slet betalinger"
11Ø8Ø PRINT "3) Udskriv månedsoversigt"
11Ø9Ø PRINT "4) Gem fil"
111ØØ PRINT "5) Hent fil"
1111Ø PRINT "6) Stop"
1112Ø PRINT:INPUT "Hvad vælger du: ",z
1113Ø WHILE be=Ø AND (z=2 OR z=3 OR z=4)
1114Ø PRINT:PRINT:PRINT"    ****** ENDNU
 INGEN DATA ******":SOUND 1,1ØØØ,1ØØ:FOR
 i=1 TO 15ØØ:NEXT i
1115Ø z=Ø
1116Ø WEND
1117Ø ON z GOSUB 12ØØØ,13ØØØ,14ØØØ,15ØØØ
,16ØØØ
```

```
11180 WEND
11190 MODE 1:INK 0,0:INK 1,24
11200 LOCATE 11,11:PRINT "KONTOEN ER LUK
KET"
11210 END
```

Modul 5.1.3: Indlæsning af nye betalinger

Dette indlæsningsmodul er mere omfattende end dem vi tidligere har brugt, og det skyldes simpelthen at selve indlæsningerne er mere omfattende. For hver indlæst post, skal der bruges fem oplysninger: om betalingen er kredit eller debet (indtægt eller udgift), betalingens navn, beløbet samt måned og dato for betalingen.

Modul 5.1.3: Linie 12000-12400

```
12000 REM ****************************
12010 REM Indlæs nye betalinger
12020 REM ****************************
12030 CLS
12040 PRINT "Nye poster:":PRINT
12050 PRINT "1) Kredit":PRINT "2) Debet"
12060 PRINT:INPUT "Hvad vælger du: ",kd:
kd=kd-1
12070 q$="":WHILE q$="" OR LEN(q$)>14:PR
INT:PRINT "Betalingens navn (max. 14 bog
staver)":INPUT ":",q$:WEND
12080 PRINT:INPUT "Beløb:",q
12090 en=1
12100 WHILE en
12110 PRINT:INPUT "Måned (fx 01040710):"
,r$:PRINT
12120 en=0:FOR i=1 TO LEN(r$) STEP 2
12130 m=VAL(MID$(r$,i,2))-1
12140 IF m<0 OR m>11 THEN PRINT:PRINT "
 *** UGYLDIG MÅNED INDLÆST ***":SOUND 1,
1000,100:en=1:i=LEN(r$):FOR j=1 TO 1500:NEXT
12150 IF en=0 THEN PRINT md$(m);"/";
12160 NEXT i:PRINT:PRINT
12170 WEND
```

```
12180 INPUT "Betalingsdato:",s
12190 PRINT:INPUT "Er det OK (j/n)";t$
12200 IF LOWER$(t$)<>"j" THEN RETURN
12210 be=be+1
12220 j=be-1
12230 WHILE s<a(ABS(j),1) AND j>=0
12240 FOR k=0 TO 1
12250 a$(j+1,k)=a$(j,k)
12260 a(j+1,k)=a(j,k)
12270 NEXT k
12280 j=j-1
12290 WEND
12300 j=j+1
12310 a$(j,1)="000000000000"
12320 FOR i=1 TO LEN(r$) STEP 2
12330 m=VAL(MID$(r$,i,2))
12340 a$(j,1)=LEFT$(a$(j,1),m-1)+"1"+RIG
HT$(a$(j,1),12-m)
12350 NEXT i
12360 a$(j,0)=q$
12370 a(j,0)=q
12380 a(j,1)=s
12390 IF kd=1 THEN a(j,0)=-a(j,0)
12400 RETURN
```

Kommentar

Linie 12040-12060: Programmet skal naturligvis vide om det er en ind- eller udbetaling, kredit eller debet. Det registreres af variablen KD (Kredit/Debet).

Linie 12090-12170: Betalingsmånederne bliver indlæst som en streng med to-cifrede tal, uden adskillelse imellem. Derfor skal der, hvis det f.eks. er en kvartalsvis betaling i februar, maj, august og november indlæses '02050811', som betyder månederne 2, 5, 8 og 11. FOR løkken, som starter i linie 12120, checker om strengen indeholder nogen 'ugyldige' måneder, og giver brugeren besked, hvis det er tilfældet. Som en yderligere sikring udskriver løkken månedernes navne (i MD$), så brugeren sikrer sig månederne ikke blot er 'gyldige', men at det også er de rigtige. Rutinen i linie 12090 gentages, hvis en ugyldig måned bliver indlæst, mens

WHILE-løkken afhænger af værdien af EN (Error Nummer), som sættes lig 1, hvis der laves en ugyldig indlæsning.

Linie 12210: På dette punkt er den indlæste information godkendt af brugeren, og variablen, der registrerer antallet af betalinger, BE, opskrives med 1.

Linie 12230-12300: Formålet med denne løkke er at indplacere den nye betaling efter *dato* i filen. Frem for at lave kopier af de betalinger, som gælder flere måneder, er systemet opbygget på den måde, at alle betalingerne samlet opstilles i dato-rækkefølge. Når en betalingsoversigt for en bestemt måned skal udskrives, vil en senere del af programmet checke alle betalingerne, for at undersøge om de falder i den pågældende måned eller før, – hvis de falder før, skal tallene bruges til at udregne saldo for den pågældende måned.

Når en ny betaling indlæses, starter løkken fra linie 12230 til 12290 med den højeste dato (som er 999, som blev indsat i initialiseringsmodulet), og den arbejder sig derefter nedad til den første betaling, hvis dato er *mindre* end værdien S, – den dag der netop er indlæst af brugeren. Hvis en sådan dato ikke findes, flyttes den sidst undersøgte betaling en plads op. Med andre ord så flyttes en tom linie med når filen undersøges, og når den rigtige placering findes, er den tomme linie allerede placeret. Til sidst bliver J, som registrerer placeringen af den første betalingsdato mindre end S, opskrevet med 1, så den peger på den tomme linie.

Linie 12310-12350: Næste opgave bliver at formattere den indlæste måned, så den kan bruges i senere dele af programmet. Den enkle fremgangsmåde er at bruge en streng med 12 nuller, som registrerer de måneder hvor der *ikke* skal ske betalinger, og derefter bruge en løkke til at ændre nuller til ettaller i de måneder hvor der *skal* ske betalinger. Således kommer strengen for de kvartalsvise betalinger ovenfor til at se sådan ud: '010010010010'.

Linie 12360-12390: De nye oplysninger placeres i de to hovedarrays. Hvis variablen KD viser at betalingen er debet, bliver beløbet ganget med minus 1, så det bliver negativt.

Test

Kør programmet og vælg '1' på menuen. Indlæse nu denne betaling:

Debet ('2' ved valget mellem kredit og debet)
Navn: TEST
Beløb: 100
Måned: 02050811
Betalingsdato: 15

Efter en pause kommer hovedmenuen frem. Stop programmet med '6' på menuen. Skriv nu:

PRINT A$(0,0),A$(0,1),A(0,0),A(0,1)

og resultatet skal være:

TEST 010010010010 -100 15

Modul 5.1.5: Udskrift på månedsoversigt

Selvom der kommer flere moduler, kommer vi allerede nu til det, der er programmets formål, nemlig at samle betalingerne, så der kan laves udskrift af saldo og betalinger måned for måned. Modulet samler betalingerne for den ønskede måned og udskriver dem, og udregner saldoen for månederne forud og derefter for hver betaling. Udskriften kan både fås på skærmen og på papir via en printer.

Modul 5.1.4: Linie 14000-14290

```
14ØØØ REM ****************************
14Ø1Ø REM Samling af månedsoversigt
14Ø2Ø REM ****************************
14Ø3Ø sum=Ø
14Ø4Ø CLS:PRINT "OVERSIGT:":PRINT
14Ø5Ø q=Ø:WHILE q<1 OR q>12
14Ø6Ø PRINT:INPUT "Oversigt for måned nu
mmer (1-12):",q
14Ø7Ø WEND
14Ø8Ø FOR j=1 TO q-1
14Ø9Ø FOR i=Ø TO be-1
141ØØ IF MID$(a$(i,1),j,1)="1" THEN sum=
sum+a(i,Ø)
```

```
14110 NEXT i
14120 NEXT j
14130 PRINT:INPUT "Udskrift på printer (
j/n)";p$:ch=0:IF LOWER$(p$)="j" THEN ch=
8
14140 CLS:PRINT #ch,TAB (20-LEN(md$(q-1)
)/2);UPPER$(md$(q-1)):PRINT #ch,STRING$(
39,"-")
14150 PRINT #ch,"DAG & OPLYSNINGER";TAB(
23);"BELØB      SALDO"
14160 PRINT #ch,STRING$(39,"-")
14170 PRINT #ch,"   Transport";:PEN 2-SG
N(sum+0.001):PRINT #ch,TAB (31);USING "#
####.##-";sum
14180 FOR i=0 TO be-1
14190 d=0:WHILE MID$(a$(i,1),q,1)="1" AN
D d=0:d=1
14200 PEN 1:PRINT #ch,USING "## &";a(i,1
);a$(i,0);
14210 PEN 2-SGN(a(i,0+0.001)):PRINT #ch,
TAB (20);USING "#####.##-";a(i,0);
14220 sum=sum+a(i,0)
14230 PEN 2-SGN(sum+0.001):PRINT #ch,TAB
 (31);USING "#####.##-";sum
14240 WEND
14250 NEXT i:PEN 1
14260 IF ch=8 THEN RETURN
14270 PRINT:PRINT "Tryk på en tast for a
t fortsætte: "
14280 WHILE INKEY$="":WEND
14290 RETURN
```

Kommentar

Linie 14030: Variablen SUM bruges til at registrere kontoens saldo – både frem til den pågældende måned (transport) og for hver betaling.

Linie 14080-14120: Forudsat at oversigten ikke gælder den første måned, hvor der ikke skal udregnes nogen transport-saldo, gennemgår disse to løkker betalingslisten en gang for hver måned, der kommer før oversigtsmåneden. På den måde bliver hver betaling undersøgt, for at finde ud af,

om der er lavet betalinger i en af de foregående måneder, – er det tilfældet bliver beløbet lagt til SUM. Når de to løkker er afsluttet indeholder SUM saldoen for året frem til den pågældende måned. (Hvis der er en saldo per 1. januar kan du blot anføre saldoen som kredit eller debet den 1. januar).

Linie 14170: Læg mærke til at hvis SUM er negativ ændres skriftfarven til rød. Den samme teknik er anvendt i linierne nedenfor. Bemærk også brugen af PRINT USING kommandoen, som sikrer at vi får en pæn udskrift med samme decimalpunktummer under hinanden.

Linie 14180-14250: Denne løkke gennemgår hele betalingslisten, mens løkken i linie 14190-14240 kun udvælger de betalinger, der har et '1' på den plads i strengen, som signalerer den pågældende måned. Når en betaling skal laves i den pågældende måned udskriver løkken datoen, A(I,1), navnet A$(I,0), beløbet A(I,0) og den nye saldo, som findes ved at lægge beløbet til SUM. Bortset fra datoen udskrives bliver debet-betalinger med rødt. Udskriften bliver nydelig selvom både tal og ord har forskellige længder, på grund af brugen af TAB og PRINT USING.

Test

Indlæs de data du brugte i sidste test. Vælg derefter '3' på hovedmenuen.

Begynd med måned 1 og check oversigterne måned for måned. Kun februar, maj, august og november skal vise oplysning om betalinger.

Modul 5.1.5: Gem/hent data

Et standard modul.

Modul 5.1.5: Linie 15000-16140

```
15ØØØ REM ****************************
15Ø1Ø REM Gem på bånd
15Ø2Ø REM ****************************
15Ø3Ø CLS:q$="":WHILE LOWER$(q$)<>"j"
15Ø4Ø INPUT "Filens navn: ",fi$:fi$=UPPE
R$(fi$)
15Ø5Ø PRINT:PRINT "Filen gemmes under na
vnet ";fi$
15Ø6Ø PRINT:INPUT "Er det rigtigt (j/n)"
;q$:PRINT
```

```
15Ø7Ø WEND
15Ø8Ø OPENOUT fi$
15Ø9Ø PRINT #9,be
151ØØ FOR i=Ø TO be-1
1511Ø PRINT #9,a$(i,Ø);cr$;a$(i,1);cr$;a
(i,Ø);cr$;a(i,1)
1512Ø NEXT i
1513Ø CLOSEOUT
1514Ø RETURN
16ØØØ REM ****************************
16Ø1Ø REM Hent fra bånd
16Ø2Ø REM ****************************
16Ø3Ø CLS:q$="":WHILE LOWER$(q$)<>"j"
16Ø4Ø INPUT "Filens navn: ";fi$:fi$=UPPE
R$(fi$)
16Ø5Ø PRINT:PRINT "Filen der hentes hedd
er ";fi$
16Ø6Ø PRINT:INPUT "Er det rigtigt (j/n)"
;q$:PRINT
16Ø7Ø WEND
16Ø8Ø OPENIN fi$
16Ø9Ø INPUT #9,be
161ØØ FOR i=Ø TO be-1
1611Ø INPUT #9,a$(i,Ø),a$(i,1),a(i,Ø),a(
i,1)
1612Ø NEXT i
1613Ø CLOSEIN
1614Ø RETURN
```

Modul 5.1.6: Ændring og sletning af betalinger

Ligesom i NTAL, er dette et ret enkelt ændringsmodul, som arbejder på grundlag af en FOR-løkke, som gennemgår betalingerne en for en.

Modul 5.1.6: Linie 13000-13250

```
13ØØØ REM ****************************
13Ø1Ø REM Check/Slet betalinger
13Ø2Ø REM ****************************
13Ø3Ø FOR i=Ø TO be-1
13Ø4Ø CLS
```

```
13Ø5Ø PRINT "Betaling: ";a$(i,Ø)
13Ø6Ø PRINT "Beløb: ";a(i,Ø)
13Ø7Ø PRINT "Måned(er): ";
13Ø8Ø FOR j=1 TO 12
13Ø9Ø IF MID$(a$(i,1),j,1)="1" THEN PRIN
T md$(j-1);"/";
131ØØ NEXT j:PRINT
1311Ø PRINT "Betalingsdato: ";a(i,1)
1312Ø PRINT:PRINT "Mulige kommandoer:":P
RINT
1313Ø PEN 2:PRINT "ENTER";:PEN 1:PRINT "
 næste betaling"
1314Ø PRINT "'|' retur"
1315Ø PRINT "'|S' slet betaling"
1316Ø q$="":PRINT:INPUT "Hvad vælger du:
 ",q$
1317Ø WHILE UPPER$(q$)="|S"
1318Ø FOR j=i TO be-1
1319Ø FOR k=Ø TO 1
132ØØ a$(j,k)=a$(j+1,k):a(j,k)=a(j+1,k)
1321Ø NEXT k,j
1322Ø be=be-1:q$="|"
1323Ø WEND
1324Ø IF q$="" THEN NEXT i
1325Ø RETURN
```

Test

Indlæs nogle data og gem dem på bånd, hent dem derefter igen og vælg derefter '2' på menuen. Nu skal du kunne 'bladre' gennem betalingerne og slette dem du ønsker. Hvis denne funktion også er i orden skulle programmet være klar til brug.

Program 5.2: Bogholder

Det andet – og sidste – program i dette kapitel er lidt mere indviklet end KONTO. Dets funktion er – som navnet siger – at føre kredit- og debetsiden af et regnskab. Det opstilles som et normalt bogholderi, hvor nogle poster står alene (som f.eks. 'ferie' i fig. 5.2), mens andre er inddelt i forskellige hovedgrupper (som f.eks. 'husholdning'). Programmet er indrettet på at kunne håndtere beløb op til kr. 99999.99. Større beløb bliver beregnet præcist, men

selve opstillingen bliver ikke så pæn. Hvis du gerne vil operere med større beløb kan du blot ændre på PRINT USING kommandoerne, og evt. på TAB-indstillingerne.

Modul 5.2.1: Initialisering

Et standard initialiseringsmodul.

Modul 5.2.1: Linie 10000-10080

```
10000 REM ****************************
10010 REM Initialisering
10020 REM ****************************
10030 CLS
10040 WHILE in=0
10050 in=1:cr$=CHR$(13):
10060 DIM a$(1,99),a(1,99)
10070 WEND
10080 MODE 1
```

```
                  DEBET

HUSHOLDNING
  BAGER             87.10
  EL OG GAS        457.90
  KØBMAND         1285.25
                  --------
                                1830.25
FERIE                           4568.00

BILUDGIFTER
  VASK              40.00
  BENZIN           290.00
  DÆK             1256.40
                  --------
                                1586.40
TV-REPARATION                    654.05
                                --------
TOTAL:                          8638.70
```

Fig. 5.2: Printerudskrift fra BOGHOLDER.

Kommentar

Linie 10060: Bogholderiets to sider, kredit og debet, indeholdes sammen med posternes navne, i de to sider af de to arrays A og A$. Der er plads til 100 poster på hver side, men dimensioneringen kan gøres større, hvis det er nødvendigt.

Modul 5.2.2: Hovedmenuen

Et standardmodul, som spærrer for visse funktioner, indtil der er indlæst data.

Modul 5.2.2: Linie 11000-11220

```
11ØØØ REM ******************************
11Ø1Ø REM Menu
11Ø2Ø REM ******************************
11Ø3Ø CLS:PEN 2:PRINT TAB(17);"BOGHOLDER
";TAB(17);"BOGHOLDER":PEN 1:WINDOW 1,4Ø,
4,25
11Ø4Ø z=Ø:WHILE z<>6:CLS
11Ø5Ø PRINT "Mulige Kommandoer:":PRINT
11Ø6Ø PRINT "1) Nye poster"
11Ø7Ø PRINT "2) Ret/slet poster"
11Ø8Ø PRINT "3) Udskriv regnskabet"
11Ø9Ø PRINT "4) Gem fil"
111ØØ PRINT "5) Hent fil"
1111Ø PRINT "6) End"
1112Ø PRINT:INPUT "Hvad vælger du: ",z
1113Ø IF z<4 AND z>Ø THEN GOSUB 12ØØØ
1114Ø WHILE c(kd)=Ø AND (z=2 OR z=3 OR z
=4)
1115Ø PRINT:PRINT:PRINT"     ******* ENDN
U INGEN DATA ******":SOUND 1,1ØØØ,1ØØ:FO
R i=1 TO 15ØØ:NEXT i
1116Ø z=Ø
1117Ø WEND
1118Ø ON z GOSUB 13ØØØ,16ØØØ,18ØØØ,19ØØØ
,2ØØØØ
1119Ø WEND
112ØØ CLS
```

```
11210 LOCATE 11,11:PRINT "PROGRAMMET ER
SLUT"
11220 END
```

Modul 5.2.3: Kredit eller debet?

Til forskel fra KONTO, skal programmet her tit have besked om det er kredit- eller debetposter, der behandles, derfor er der lavet et specielt modul, til at klare dette.

Modul 5.2.3: Linie 12000-12110

```
12000 REM ****************************
12010 REM Kredit eller debet?
12020 REM ****************************
12030 kd=-1:WHILE kd<>0 AND kd<>1
12040 CLS
12050 PRINT "Kredit eller debet: ":PRINT
12060 PRINT "1) Kredit":PRINT "2) Debet"
12070 PRINT:INPUT "Hvad vælger du: ",kd:
kd=kd-1
12080 kd$="KREDIT"
12090 IF kd=1 THEN kd$="DEBET"
12100 WEND
12110 RETURN
```

Modul 5.2.4: Hvilken type post?

Dette modul er i sig selv nemt at forstå, men det giver en forklaring på, hvorfor et program som dette bliver længere end et program som f.eks. KONTO. Formålet med modulet er, at give brugeren mulighed for at vælge hvilken type post indlæsningen gælder – der er tre muligheder:

1) En enkelt post: Her skal der blot angives et navn på posten og et beløb. Når regnskabet udskrives får enkelt-posterne navnet udskrevet til venstre og beløbet i hovedkolonnen til højre. I fig. 5.2 er 'ferie' et eksempel på en enkelt-post.

2) En hovedrubrik: Man kan gruppere sammenhørende poster under en hovedrubrik. I fig. 5.2 er 'biludgifter' et eksempel på en hovedrubrik, som er en samlet betegnelse for den type udgifter, der har med bilen at

gøre. Som det ses af figuren udskrives hovedrubrikken til venstre, men der er ikke noget tilhørende beløb til højre.

3) Underrubrikker: I fig. 5.2 er 'vask', 'benzin' og 'dæk' naturligvis underrubrikker under hovedrubrikken 'biludgifter'. Som det ses er navnene trukket lidt ind fra venstre margin, og beløbet bliver udskrevet i en speciel kolonne til venstre for hovedkolonnen.

Modul 5.2.4: Linie 13000-13120

```
13ØØØ REM ****************************
13Ø1Ø REM Indlæs hovedrubrikker
13Ø2Ø REM ****************************
13Ø3Ø CLS
13Ø4Ø PRINT "Nye poster:":PRINT
13Ø5Ø PRINT kd$:PRINT
13Ø6Ø PRINT "Er posten:":PRINT
13Ø7Ø PRINT "1) En enkelt post;"
13Ø8Ø PRINT "2) En hovedrubrik eller"
13Ø9Ø PRINT "3) En underrubrik"
131ØØ PRINT:INPUT "(Ø for retur)";type
1311Ø ON type GOSUB 14ØØØ,14ØØØ,15ØØØ
1312Ø RETURN
```

Test

De dele af programmet der er indlæst indtil nu omfatter ikke nogen udregninger, og inden vi kommer til det kan vi teste om menuen fungerer. Vælg '1' og du skal få valget mellem kredit og debet, og derefter kan du vælge hvilken type post, der drejer sig om, – og videre kan du ikke komme nu. Derudover kan du stoppe programmet med '6'. Hvis du prøver '2' eller '3' skal du få en melding om, at der ingen data er.

Modul 5.2.5: Enkelt post eller hovedrubrik

Et modul tager sig af indlæsning af enkelt-poster og hovedrubrikker, mens et andet tager sig af underrubrikkerne. For at forstå senere dele af programmet, er det vigtigt at du lægger mærke til hvordan posterne lagres og hvilke indikator-karakterer, der anvendes til de forskellige typer.

Modul 5.2.5: Linie 14000-14120

```
14ØØØ REM ****************************
14Ø1Ø REM Enkelt post el. hovedrubrik
14Ø2Ø REM ****************************
14Ø3Ø q=Ø:q$="":r$=""
14Ø4Ø PRINT:INPUT "Navn: ",q$
14Ø5Ø IF type=1 THEN PRINT:INPUT "Posten
s beløb: ",q
14Ø6Ø PRINT:INPUT "Er det rigtigt (j/n)
";r$
14Ø7Ø IF LOWER$(r$)<>"j" THEN PRINT:PRIN
T "    ****** IKKE REGISTRERET ******":S
OUND 1,1ØØØ,1ØØ:FOR i=1 TO 15ØØ:NEXT:RET
URN
14Ø8Ø IF type=1 THEN q$="%"+q$ ELSE q$="
*"+q$
14Ø9Ø a$(kd,c(kd))=q$
141ØØ a(kd,c(kd))=q
1411Ø c(kd)=c(kd)+1
1412Ø RETURN
```

Kommentar

Linie 14050: Som nævnt i indledningen til det forrige modul, er der ikke knyttet noget beløb til hovedrubrikkerne så denne linie tager kun imod beløb knyttet til enkelt-poster.

Linie 14080: De forskellige typer lagres i de samme arrays og skilles kun i kredit og debet. Senere dele af programmet vil bestemme typen udfra den indikator-karakter, som sættes foran postens navn. '%' signalerer en enkelt-post og '*' signalerer en hovedrubrik.

Linie 14090-14120: Du er allerede stødt på variablen KD som registrerer om posten er kredit eller debet. Her bruges KD til at bestemme hvilken side af de to arrays A og A$, den nye post skal placeres i. Derudover skal vi holde tal på hvor mange poster der er på hhv. kredit- og debetsiden, da de normalt vil være forskellige. Det sker med arrayet C. Arrayet blev ikke dimensioneret i begyndelsen, fordi det kun indeholder to elementer, C(0) og C(1), svarende hhv. til kredit- og debetsiden i de to hovedarrays. KD

bruges også her til at afgøre hvilken af de to elementer i C, der skal anvendes. F.eks. skal 'A(KD,C(KD))' forstås således:

```
A     ( KD,     C(KD))
1       2         3
```

1) Et element i talarrayet A.
2) Den side der indikeres af KD, dvs. kredit eller debet.
3) Det første tomme element på denne side, bestemt af hvad der allerede er lagret.

Test

Kør programmet og vælg '1'. Vælg derefter at indlæse en hovedrubrik på kreditsiden med TEST HOVED som navn – der er ikke brug for noget beløb. Vælg igen '1' og indlæs denne gang en enkelt-post på kreditsiden med TEST som navn og beløbet 100. Udfør derefter nøjagtigt den samme procedure på debetsiden.

Stop programmet med '6' og skriv:

PRINT A(0,0),A(1,0),A$(0,0),A$(1,0) <ENTER>

og du skal se følgende:

```
0     0     *TEST HOVED      *TEST HOVED
```

Gentag proceduren for linie 1 i de to arrays, f.eks. A(0,1) osv. Nu skal du se dette:

```
100     100     %TEST      %TEST
```

Udskriv til sidst værdien af C(0) og C(1) – begge skal være lig 2.

Modul 5.2.6: Indlæsning af underrubrik

Det er ikke helt så let at indlæse en underrubrik, som det er at indlæse en enkelt-post. For hver ny underrubrik, der indlæses skal det nemlig checkes om der findes en tilsvarende hovedrubrik, og underrubrikken skal placeres ved siden af denne, fremfor blot at blive sat efter de øvrige poster.

```
15ØØØ REM ****************************
15Ø1Ø REM Underrubrikker
15Ø2Ø REM ****************************
15Ø3Ø PRINT:INPUT "Hovedrubrik: ",q$
15Ø4Ø q$="*"+q$
15Ø5Ø p=-1:FOR i=Ø TO c(kd)-1
15Ø6Ø IF a$(kd,i)=q$ THEN p=i+1
15Ø7Ø NEXT i
15Ø8Ø IF p=-1 THEN PRINT:PRINT "   ** IN
GEN RUBRIKKER MED DET NAVN**":SOUND 1,1Ø
ØØ,1ØØ:FOR i=1 TO 15ØØ:NEXT:RETURN
15Ø9Ø PRINT:INPUT "Underrubrikkens navn:
 ",q$
151ØØ PRINT:INPUT "Beløb: ",q
1511Ø PRINT:INPUT "Er det rigtigt (j/n)
";r$
1512Ø IF LOWER$(r$)<>"j" THEN PRINT:PRIN
T "     ****** IKKE REGISTRERET ******":
SOUND 1ØØØ,1ØØ:FOR i=1 TO 15ØØ:NEXT:RETU
RN
1513Ø q$="$"+q$
1514Ø FOR i=c(kd)+1 TO p+1 STEP -1
1515Ø a$(kd,i)=a$(kd,i-1)
1516Ø a(kd,i)=a(kd,i-1)
1517Ø NEXT i
1518Ø a$(kd,p)=q$
1519Ø a(kd,p)=q
152ØØ c(kd)=c(kd)+1
1521Ø RETURN
```

Kommentar

Linie 15030-15080: Først beder programmet om navnet på den hovedrubrik, som underrubrikken skal høre til, og derefter checkes det om hovedrubrikken findes – gør den ikke det udskrives der en melding om det, og der vendes tilbage til menuen.

Linie 15140-15200: Som nævnt tidligere, er hele ideen med at bruge underrubrikker, at disse kan samles i grupper under bestemte hovedrubrikker. Derfor lagres de ved siden af deres hovedrubrik i filen. Placeringen af

første post efter hovedrubrikken er allerede fundet af FOR-løkken i linie 15050, så tilbage er blot at flytte alle posterne fra det sted et trin op, og indsætte den nye post der.

Test

Indlæs posterne fra testen i modul 2.5.2 og vælg derefter igen '1' på menuen. Indlæses nu en ny underrubrik på kreditsiden med navnet TEST UNDER og beløbet 200. Gør det samme på debetsiden. SKRIV NU:

FOR I=0 TO 2:PRINT A(0,I),A(1,I),A$(0,I),A$(1,I):NEXT <ENTER>

og du skal se følgende:

0	0	*TEST HOVED	*TEST HOVED
200	200	$TEST UNDER	$TEST UNDER
100	100	%TEST	%TEST

Check derefter værdierne af C(0) og C(1) – begge skal være lig 3.

Modul 5.2.7: Gem og hent data

Da datastrukturen for BOGHOLDER er ret omfattende, er det en god idé og tage gemme- og hentemodulet nu, så vi kan undgå at indlæse det samme igen og igen. Når modulet er indlæst indlæser du data fra den sidste test og gemmer dem på bånd.

Modul 5.2.7: Linie 19000-20160

```
19000 REM ******************************
19010 REM Gem på bånd
19020 REM ******************************
19030 CLS:q$="":WHILE LOWER$(q$)<>"j"
19040 INPUT "Navnet på filen: ";fi$
19050 PRINT:PRINT "Filen der gemmes hedd
er ";fi$
19060 PRINT:INPUT "Er det rigtigt (j/n)"
;q$:PRINT
19070 WEND
19080 OPENOUT fi$
19090 FOR i=0 TO 1
19100 PRINT #9,c(i)
```

```
19110 FOR j=0 TO c(i)-1
19120 PRINT #9,a$(i,j);cr$;a(i,j)
19130 NEXT j
19140 NEXT i
19150 CLOSEOUT
19160 RETURN
20000 REM ****************************
20010 REM Hent fra bånd
20020 REM ****************************
20030 CLS:q$="":WHILE LOWER$(q$)<>"j"
20040 INPUT "Navnet på filen: ";fi$
20050 PRINT:PRINT "Filen der hentes hedd
er ";fi$
20060 PRINT:INPUT "Er det rigtigt (j/n)"
;q$:PRINT
20070 WEND
20080 OPENIN fi$
20090 FOR i=0 TO 1
20100 INPUT #9,c(i)
20110 FOR j=0 TO c(i)-1
20120 INPUT #9,a$(i,j),a(i,j)
20130 NEXT j
20140 NEXT i
20150 CLOSEIN
20160 RETURN
```

Modul 5.2.8: Ændringer

Et ret enkelt modul, hvor der er taget hensyn til at nogen poster ikke står alene, men hører til hele grupper af poster under en fælles hovedrubrik.

Modul 5.2.8: Linie 16000-16290

```
16000 REM ****************************
16010 REM Ret eller slet
16020 REM ****************************
16030 FOR i=0 TO c(kd)-1
16040 d=0:WHILE d=0:d=1
16050 CLS
16060 PRINT "Ret eller slet: ":PRINT
16070 IF LEFT$(a$(kd,i),1)<>"$" THEN PRI
NT MID$(a$(kd,i),2);
```

```
16Ø8Ø IF LEFT$(a$(kd,i),1)="*" THEN hh$=
MID$(a$(kd,i),2):PRINT
16Ø9Ø IF LEFT$(a$(kd,i),1)="$" THEN PRIN
T hh$:PRINT "  ";MID$(a$(kd,i),2);
161ØØ IF a(kd,i)<>Ø THEN PRINT TAB(31);U
SING"#####.##";a(kd,i)
1611Ø PRINT:PRINT "Mulige kommandoer:":P
RINT
1612Ø PRINT "1) Næste post"
1613Ø PRINT "2) Ændring af beløb"
1614Ø PRINT "3) Retur til menu"
1615Ø PRINT "4) Slet post"
1616Ø PRINT:PRINT "Hvad vælger du?"
1617Ø q$="":WHILE q$=""
1618Ø q$=INKEY$
1619Ø WEND
162ØØ IF q$="4" THEN GOSUB 17ØØØ:RETURN
1621Ø IF q$="3" THEN RETURN
1622Ø WHILE q$="2" AND LEFT$(a$(kd,i),1)
<>"*"
1623Ø PRINT:INPUT "Beløb som skal lægges
 til: ",q
1624Ø PRINT:INPUT "Er det rigtigt (j/n)"
;r$
1625Ø IF LOWER$(r$)="j" THEN a(kd,i)=a(k
d,i)+q:q$=""
1626Ø WEND
1627Ø WEND
1628Ø NEXT i
1629Ø RETURN
```

Kommentar

Linie 16070-16090: Hvis posten i filen er en enkelt-post udskrives den blot – dog uden indikator-karakteren som er sat foran navnet. Hvis posten er en hovedrubrik, bliver den ikke blot skrevet ud – dens navn lagres også i HH$, så det kan blive udskrevet over eventuelt efterfølgende underrubrikker.

Linie 16220-16260: Dette modul kan ikke blot slette poster, det kan også ændre beløbene ved posterne. Det sker ved at lægge et tal – positivt eller

negativt – til beløbet, så beløbet på den måde gøres større eller mindre. Når vi ikke blot indlæser et helt nyt beløb i stedet for, skyldes det at de fleste ændringer alligevel sker i forhold til det aktuelle beløb. Hvis vi derfor bruger benzin for 200 kr. mere, kan vi blot finde 'benzin' i filen og lægge 200 til det beløb, der står der i forvejen.

Test

Kør programmet og hent de gamle data fra båndet. Vælg nu '2' på menuen og prøv at 'bladre' gennem de tre poster. Vælg igen '2' og prøv derefter at trække et beløb fra de to beløb der findes i forvejen. Check at beløbene bliver ændret som de skal. Lav testen for både kredit og debet.

Modul 5.2.9: Sletning af poster

Nu mangler vi at kunne slette poster. I BOGHOLDER er slettemodulet noget mere indviklet end i tidligere tilsvarende moduler. Grunden til det er at posterne er grupperet under hovedrubrikker. Der er ingen problemer med at slette enkelt-poster eller underrubrikker, – men hvad hvis vi sletter en hovedrubrik? Svaret er naturligvis at vi ikke blot skal have slettet hovedrubrikken, men også de tilhørende underrubrikker – ellers vil der blive rod i bogholderiet.

Modul 5.2.9: Linie 17000-17140

```
17ØØØ REM ******************************
17Ø1Ø REM Sletning
17Ø2Ø REM ******************************
17Ø3Ø p=i:gr=1
17Ø4Ø d=Ø:WHILE LEFT$(a$(kd,p),1)="*" AN
D d=Ø:d=1
17Ø5Ø WHILE LEFT$(a$(kd,p+gr),1)="$"
17Ø6Ø gr=gr+1
17Ø7Ø WEND
17Ø8Ø WEND
17Ø9Ø FOR k=p TO c(kd)-gr-1
171ØØ a(kd,k)=a(kd,k+gr)
1711Ø a$(kd,k)=a$(kd,k+gr)
1712Ø NEXT k
1713Ø c(kd)=c(kd)-gr
1714Ø RETURN
```

Kommentar

Linie 17030: Stedet hvor der skal slettes sendes fra det forrige modul i form af variablen I. Til brug i dette modul laves den om til P. Variablen GR (GRuppe) registrerer hvor mange poster der skal slettes. Den sættes til start lig 1 og ændres kun, hvis det er en hovedrubrik med tilhørende underrubrikker, der skal slettes.

Linie 17040-17080: Disse to løkker aktiveres kun hvis det er en hovedrubrik, der skal slettes. Den indre løkke checker de efterfølgende poster for at undersøge om de er mærket med '$' – en indikering af at det er underrubrikker, antallet registreres af GR.

Linie 17090-17120: Dette er en løkke, som trækker arrayet sammen for at slette en post. Forskellen fra tidligere eksempler er blot, at i stedet for at kopiere post X ind på plads X1, og derved kopiere elementerne en plads ned hver, så flyttes posterne GR pladser, og derved slettes GR poster – eller det antal poster, som var knyttet til en hovedrubrik.

Test

Lav samme test som ved det forrige modul, – nu skal du ikke blot kunne 'bladre' posterne igennem og ændre dem, du skal også kunne slette dem. Hvis du sletter en hovedrubrik (TEST HOVED) skal tilknyttede underrubrikker (TEST UNDER) også forsvinde.

Modul 5.2.10: Udskrift af regnskabet

Efter alle disse forberedelser kommer vi nu til det modul, som er begrundelsen for at lave alt det andet – nemlig udskrift af hhv. kredit- og debetoversigter. Ligesom ved det tilsvarende modul ved KONTO, ser det indviklet ud, men når du først har set udskriften (på skærmen eller på papir), er det ret nemt at forstå hvad det er der sker.

Modul 5.2.10: Linie 18000-18310

```
18ØØØ REM ****************************
18Ø1Ø REM Udskriv regnskabet
18Ø2Ø REM ****************************
18Ø3Ø tt=Ø:ss=Ø
```

```
18Ø4Ø CLS:PRINT "Udskriv regnskabet:":PR
INT
18Ø5Ø PRINT:INPUT "Udskriv regnskabet på
 printer (j/n) ";p$:ch=Ø:IF LOWER$(p$)="
j" THEN ch=8
18Ø6Ø CLS
18Ø7Ø PRINT #ch,TAB(2Ø-LEN(kd$)/2);kd$:P
RINT
18Ø8Ø FOR i=Ø TO c(kd)-1
18Ø9Ø tt=tt+a(kd,i)
181ØØ IF LEFT$(a$(kd,i),1)="*" THEN PRIN
T #ch
1811Ø IF LEFT$(a$(kd,i),1)="$" THEN PRIN
T #ch,"  ";
1812Ø PRINT #ch,MID$(a$(kd,i),2);
1813Ø d=Ø:WHILE LEFT$(a$(kd,i),1)<>"*" A
ND d=Ø:d=1
1814Ø PRINT #ch,TAB(18);
1815Ø IF LEFT$(a$(kd,i),1)="%" THEN PRIN
T #ch,TAB(29);
1816Ø PRINT #ch,USING "#####.##";a(kd,i)
;
1817Ø IF LEFT$(a$(kd,i),1)="$" THEN ss=s
s+a(kd,i)
1818Ø WEND
1819Ø PRINT #ch
182ØØ WHILE ss<>Ø AND LEFT$(a$(kd,i+1),1
)<>"$"
1821Ø PRINT #ch,TAB(18);"--------"
1822Ø PRINT #ch,TAB(29);USING "#####.##"
;ss
1823Ø ss=Ø
1824Ø WEND
1825Ø NEXT i
1826Ø PRINT #ch,TAB(29);"--------"
1827Ø PRINT #CH,"TOTAL:";TAB(29);USING "
#####.##";tt
1828Ø IF ch=8 THEN RETURN
1829Ø PRINT:PRINT "Tryk på en tast for a
t fortsætte"
183ØØ WHILE INKEY$="":WEND
1831Ø RETURN
```

Kommentar

Linie 18090: Variablen TT bruges til at lagre den løbende total for regnskabet, mens det udskrives.

Linie 18100-18120: Disse linier udskriver posternes navne. Drejer det sig om en hovedrubrik, laves der først en tom linie, for at adskille det fra den ovenstående post. Underrubrikker rykkes to pladser ind.

Linie 18130-18180: Disse linier udskriver beløbene for enkeltposter og underrubrikker. Underrubrikker udskrives fra plads 18, mens enkeltposter udskrives fra plads 29. Hvis posten er en underrubrik, bliver under-totalen for posterne i denne gruppe midlertidigt lagret i SS.

Linie 18190-18230: Denne løkke er kun i funktion, når en gruppe underrubrikker behandles, – indikeret ved at SS ikke er lig 0, og når den næste post ikke er en del af gruppen – dvs. at der ikke er flere underrubrikker i denne gruppe. Løkken udskriver så totalen for gruppen i hovedkolonnen fra plads 29.

Test

Hent for sidste gang de gemte data fra båndet og vælg '3' på menuen. Får du en rigtig udskrift (se evt. fig. 5.2), så er programmet klar til brug.

BILAG A:

Danske tegn, funktionstaster og cross reference

Program A.1 i dette bilag er beregnet til at hente ind i maskinen hver gang du tænder den. Det giver maskinen danske tegn, som får karakterkoderne 91, 92, 93, 123, 124, 125 svarende til Æ, Ø, Å, æ, ø, å.

Derudover programmeres en række funktionstaster, – de fremgår af programmet. Bemærk at programmet sletter med NEW – dvs. at selve programmet slettes når det er kørt, – men det påvirker ikke de programmerede funktioner. Lav om på programmet, så det bliver som du vil have det.

PROGRAM A.1: DANSKE TEGN OG FUNKTIONSTASTER.

```
4 REM ****************
5 REM * STARTPROGRAM *
6 REM ****************
1Ø SYMBOL AFTER 91
2Ø SYMBOL 91,62,122,216,22Ø,248,218,222
3Ø SYMBOL 92,124,2Ø6,222,254,246,23Ø,124
4Ø SYMBOL 93,56,Ø,124,198,254,198,198
5Ø SYMBOL 123,Ø,Ø,118,26,126,216,11Ø
6Ø SYMBOL 124,Ø,Ø,118,2Ø4,214,1Ø2,22Ø
7Ø SYMBOL 125,48,Ø,12Ø,12,124,2Ø4,118
8Ø KEY DEF 17,1,123,91
9Ø KEY DEF 26,1,124,92
1ØØ KEY DEF 19,1,125,93
11Ø KEY DEF 22,1,64,96,Ø
12Ø KEY DEF 7,1,46,44
13Ø KEY DEF 69,1,97,65,141 : KEY 141,"AU
TO "
14Ø KEY DEF 62,1,99,67,142 : KEY 142,"CL
S"
```

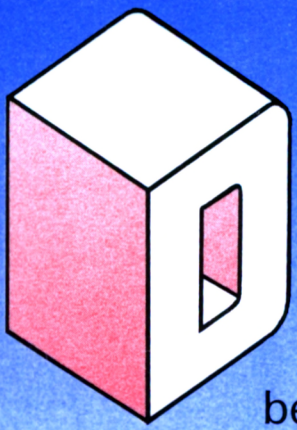
```
15Ø KEY DEF 58,1,1Ø1,69,143 : KEY 143,"E
DIT "
16Ø KEY DEF 36,1,1Ø8,76,144 : KEY 144,"L
IST "
17Ø KEY DEF 15,1,128,128,145 : KEY 145,"
MODE Ø"
18Ø KEY DEF 13,1,129,129,146 : KEY 146,"
MODE 1"
19Ø KEY DEF 14,1,13Ø,13Ø,147 : KEY 147,"
MODE 2"
2ØØ KEY DEF 27,1,112,8Ø,148 : KEY 148,"W
HILE 1:LINE INPUT A$:?#8,A$:WEND"
21Ø KEY DEF 6Ø,1,115,83,149 : KEY 149,"S
PEED WRITE Ø"
22Ø KEY DEF 59,1,119,87,15Ø : KEY 15Ø,"W
IDTH 4Ø"
23Ø KEY DEF 51,1,116,84,151 : KEY 151,"?
TAB("
1ØØØ MODE 1:PRINT "   ******************
****************";"   * F U N K T I O N S
 T A S T E R *"
1Ø1Ø PRINT "   ************************
*********"
1Ø2Ø PRINT:PRINT:PRINT"  * MASKINEN ER P
ROGRAMMERET SÅLEDES *"
1Ø3Ø PRINT:PRINT:PRINT" Med <CTRL> fås f
ølgende funktioner:":PRINT
1Ø4Ø PRINT" A giver AUTO"
1Ø5Ø PRINT" C giver CLS"
1Ø6Ø PRINT" E giver EDIT"
1Ø7Ø PRINT" L giver LIST"
1Ø8Ø PRINT" S giver SPEED WRITE Ø"
1Ø9Ø PRINT" T giver PRINT TAB("
11ØØ PRINT" W giver WIDTH 4Ø"
111Ø PRINT" Ø,1 og 2 giver de tre MODEs"
112Ø PRINT:PRINT" P kobler printeren til
"
113Ø PRINT TAB(8)"- afbrydes med <ESC>"
114Ø PRINT:PRINT:PRINT"  *  SPEED WRITE
1   *   MODE 2   *"
2ØØØ BORDER 3:WIDTH 8Ø:SPEED WRITE 1
2Ø1Ø WHILE INKEY$="":WEND
2Ø2Ø MODE 2:NEW
```

Program A.2 er lånt fra 'Alt om data' (2/85). Det er et såkaldt 'Cross reference' program. Når du i forvejen har et program i maskinen kan dette program give dig en oversigt over hvilke variabler der anvendes, og i hvilke linier de står. Programmet hentes fra båndet med MERGE og startes med RUN 65000. Det kan bl.a. være et stor hjælp, hvis du har problemer med at få et program til at køre. Prøv!

Har du en printer kan du få udskriften på papir ved at ændre 'PRINT' til 'PRINT#8,' i linie 65120 og 65130.

PROGRAM A.2: CROSS REFERENCE.

```
65ØØØ DEF FNv=ASC(MID$(l$(x),y,1))+256*A
SC(MID$(l$(x),y+1,1))
65Ø1Ø b$="ABCDEFGHIJKLMNOPQRSTUVWXYZabcd
efghijklmnopqrstuvwxyzØ123456789":t$(2)=
"%":t$(3)="$":t$(4)="!":DIM v$(1ØØ),l$(1
ØØ):a=-1:c=&17Ø
65Ø2Ø p=c:c=p+PEEK(p)+256*PEEK(p+1):l=PE
EK(p+2)+256*PEEK(p+3):IF l=65ØØØ THEN 65
12Ø ELSE PRINT l;CHR$(13);:p=p+4:b=PEEK(
p)
65Ø3Ø IF b=2 OR b=3 OR b=4 OR b=13 THEN
n=3:n$="":GOTO 65Ø5Ø
65Ø4Ø p=p+1:IF p<c THEN b=PEEK(p):GOTO 6
5Ø3Ø ELSE 65Ø2Ø
65Ø5Ø y$=CHR$(PEEK(p+n)):IF INSTR(b$,y$)
 THEN n$=n$+y$:n=n+1:GOTO 65Ø5Ø
65Ø6Ø IF ASC(y$)<128 THEN 65Ø4Ø
65Ø7Ø y$=CHR$(ASC(y$)-128):IF INSTR(b$,y
$)=Ø THEN 65Ø4Ø ELSE n$=n$+y$
65Ø8Ø IF b<5 THEN n$=n$+t$(b)
65Ø9Ø n$=LOWER$(n$):FOR x=Ø TO a:IF n$<>
v$(x) THEN NEXT:v$(x)=n$:a=x:GOTO 6511Ø
651ØØ y=LEN(l$(x))-1:IF l=FNv THEN 65Ø4Ø
6511Ø l$(x)=l$(x)+CHR$(l-256*INT(l/256))
+CHR$(INT(l/256)):p=p+n-1:GOTO 65Ø4Ø
6512Ø PRINT"the program uses"a+1"variabl
es.":PRINT"Press any key to list.":FOR x
=Ø TO a:CALL &BB18:PRINT v$(x);LEN(l$(x)
)/2;"times : "TAB(2Ø);:FOR y=1 TO LEN(l$
(x)) STEP 2:PRINT USING" #####";FNv;:IF
y MOD 2Ø=19 THEN PRINT:PRINT TAB(2Ø);
6513Ø NEXT:PRINT:NEXT
```



Denne bog henvender sig til AMSTRAD-brugere der er nået ud over begynderstadiet og som ønsker at bruge maskinen seriøst. Adskillige meget nyttige programmer præsenteres - trin for trin forklares opbygningen, således at læseren oplæres i mere avanceret programmeringsteknik.

ISBN 87-571-0913-3

D. Lawrence S. Lane **Programudvikling på AMSTRAD**