ABC
om programmering och dokumentation

Jan Lundgren
Bengt Lundin
Sören Thornell

ABC80

Åkerö

ABC
om programmering och dokumentation

Jan Lundgren
Bengt Lundin
Sören Thornell

EMMDATA Box 70, 901 03 UMEÅ
☎ 090 - 13 65 50

Fotografier PALLAS, WT-KONSULT, EMMDATA
Figurer Marianne Lundin, Ulf Nilsson

Första upplagan, första tryckningen
ISBN 91-86064-00-2

Tryckeriet, Linköping 1980

FÖRORD

Den nuvarande utvecklingen med små prisbilliga datorsystem innebär att fler och fler lär sig grunderna i programmeringsspråket BASIC. På kort tid lär man sig att skriva enkla BASIC-program. Snart inser man dock att enbart grundläggande kunskaper om de olika instruktionerna i BASIC inte räcker till för att skriva väl fungerande datorprogram.

Denna bok ger dig den ytterligare kunskap om programmering och dokumentation som behövs för att skriva överskådliga och användarvänliga program i BASIC.

I bokens första fem kapitel ingår bl a avsnitt om programuppbyggnad, speciella programmeringsrutiner och menyteknik. Därefter beskrivs utförligt olika aspekter på filhantering i kap 6 t o m 11. Avslutningsvis ges i kap 12 anvisningar om hur program kan dokumenteras.

Boken innehåller lösningar till många praktiska problem som möter BASIC-programmeraren. Förslag och anvisningar varvas med fullt körbara programillustrationer. Dessa är avsedda för den svenska datorn ABC 80 och finns tillgängliga på flexskivor. Körning av programexemplen i kap 6, 8, 9 och 10 kräver tillgång till flexskivenhet.

Boken kan med fördel användas i gymnasieskolans olika linjer t ex i ämnet datalära, samt i kurser anordnade av studieförbund och utbildningsföretag. De utförligt redovisade programexemplen underlättar också bokens användning vid självstudier.

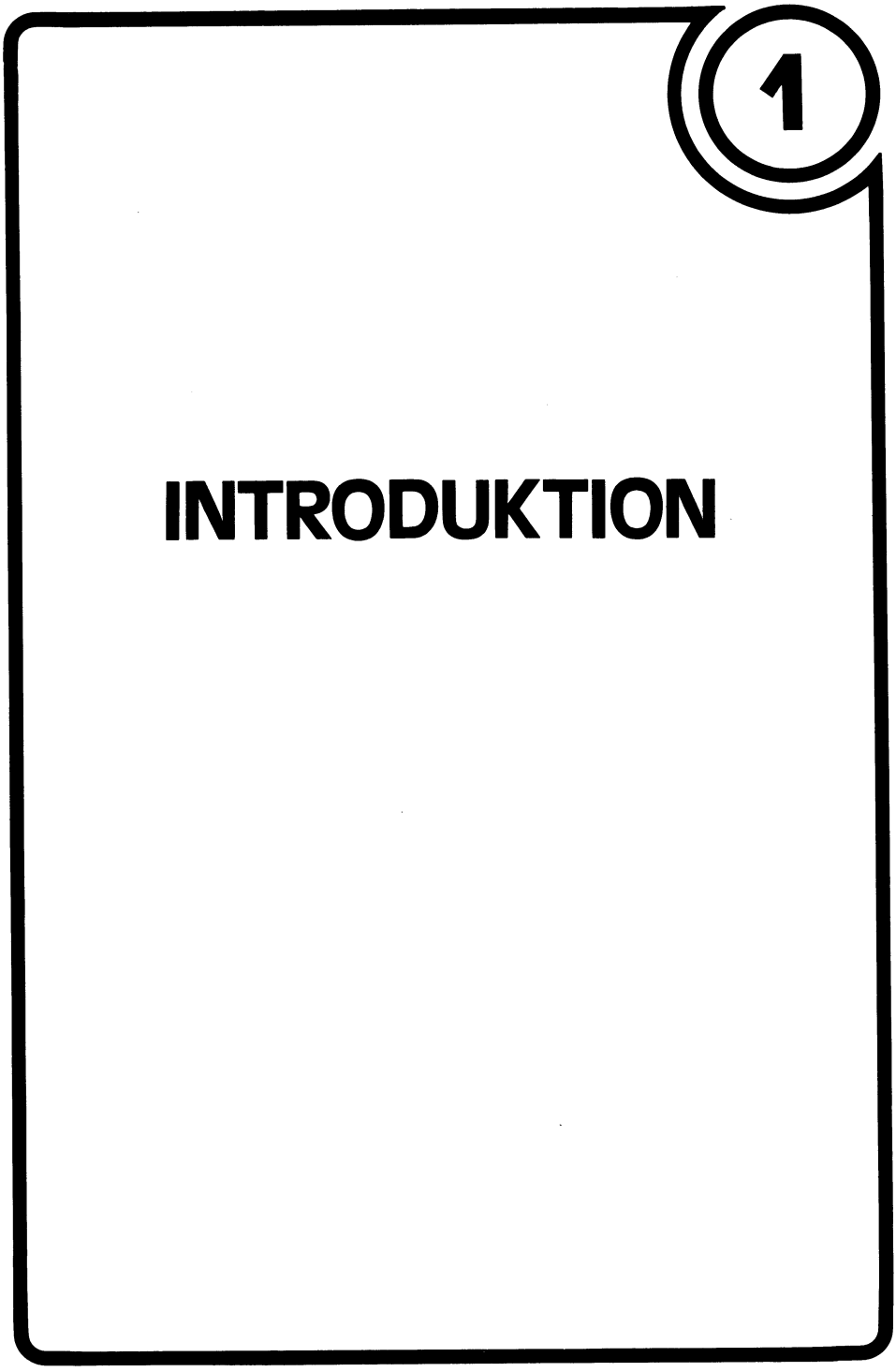
Förslag till egna övningar och ytterligare lösta exempel återfinns i övningsboken ABC-UPPGIFTER I PROGRAMMERING.

Vi tackar alla dem som på olika sätt medverkat vid tillkomsten av denna bok, speciellt medarbetarna på LUXOR och SCANDIA METRIC. De synpunkter på innehållet som Margit Lundgren bidragit med under arbetets gång har varit mycket värdefulla.

Linköping i september 1980

Författarna

INNEHÅLL



1

INTRODUKTION

INTRODUKTION

Du som har tidigare erfarenhet av programmeringsspråket BASIC vet att man mycket snabbt löser enkla problem med hjälp av en dator och ett program i BASIC. Förutsättningen är normalt att du själv samtidigt fungerar som problemställare, programmerare och användare. Den teknik som då används vid programmeringen består ofta i att du efterhand översätter tankar till instruktioner i en löpande följd. Som du kanske själv redan upptäckt fungerar denna "metod" sämre när programmets omfattning eller komplexitet ökar. Detta beror huvudsakligen på att programtexten tillkommer och kompletteras samtidigt som problemet formuleras och löses.

Låt oss illustrera detta med ett programexempel som tillkommit på detta sätt. Mata in programmet i din ABC 80 och kör sedan programmet. Försök samtidigt observera vilka frågor och reflektioner som dyker upp inom dig under programexekveringen!

```
100 PRINT CHR¤(12)
110 PRINT : PRINT
120 INPUT A : INPUT G
130 R=(A+(G*G))/(2*G) : ; R
140 R3=R2 : R2=R1 : R1=R
150 IF R=R3 GOTO 110
160 G=R : GOTO 130
```

Säkert hade du stora svårigheter att köra programmet och tolka resultatet, även om du kan förstå den använda algoritmen.

Kör nu istället den följande förbättrade versionen av samma problemlösning. Det är här inte programmets tekniska utformning utan dess funktion som är av intresse.

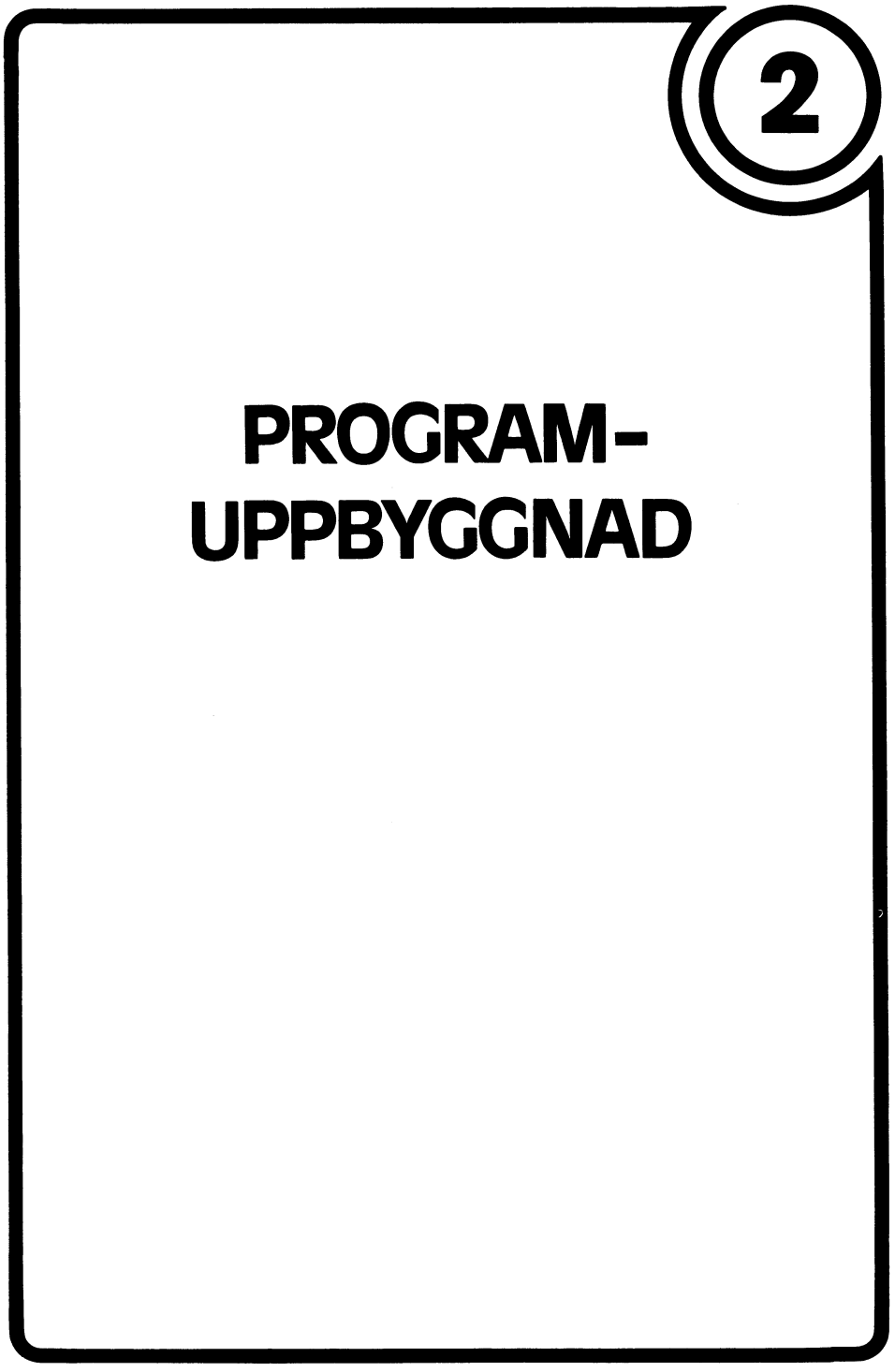
```
100 PRINT CHR¤(12)
110 PRINT "************************"
120 PRINT "*   PROGRAM-EXEMPEL 2.  *"
130 PRINT "************************"
140 PRINT
150 PRINT "PROGRAMMET BERÄKNAR  KVADRAT-"
160 PRINT "ROTEN UR ETT TAL MED HJÄLP AV"
170 PRINT "SUCCESSIVA   APPROXIMATIONER."
180 PRINT : PRINT
190 PRINT "---------------------------"
200 ONERRORGOTO 470
210 PRINT
220 PRINT "UR VILKET TAL (>1) SKALL"
230 PRINT "KVADRATROTEN BERÄKNAS.....";
240 INPUT A : PRINT
250 IF A<1 GOTO 470
260 PRINT "GÖR EN GROV GISSNING"
270 PRINT "AV ROTENS VÄRDE.....";
280 INPUT G : PRINT
290 IF G=0 THEN G=1
300 IF G<0 THEN G=G*(-1)
310 PRINT : PRINT "BERÄKNINGSRESULTAT:"
320 PRINT
330 REM ..........................
340 REM .  BERÄKNINGSALGORITM      .
350 REM
360 R=(A+(G*G))/(2*G) : ; R
370 R3=R2 : R2=R1 : R1=R
380 IF R=R3 THEN PRINT : PRINT : GOTO 410
390 G=R : GOTO 360
400 REM ..........................
410 PRINT "---------------------------"
420 PRINT : PRINT "NY BERÄKNING? (JA) ";
430 GET S¤
440 IF S¤="J" OR S¤="j" THEN 470
450 IF S¤=CHR¤(13) THEN 470
460 PRINT "NEJ!!" : GOTO 480
470 PRINT CHR¤(12) : GOTO 190
480 PRINT : PRINT "Körningen klar!"
490 PRINT "     STOP!" : PRINT
500 END
```

Den senare programversionen exemplifierar en bättre kommunikation mellan datorn och användaren. Din uppgift som programmerare är att möjliggöra en så fullständig dubbelriktad kommunikation som möjligt mellan datorn och användaren.

Grundtanken med denna bok är att underlätta dina ansträngningar att göra väl fungerande BASIC-program. Vi avser att göra detta genom att

- anvisa en struktur och en stomme för programtextens uppbyggnad
- peka på några ofta förekommande punkter i programmets instruktionsdel och ge förslag till lösningar
- uppmärksamma den stora betydelsen av riktig felhantering
- förklara hur olika typer av filer används
- presentera några tips kring filhantering på ABC 80
- föreslå en mall för dokumentation av program

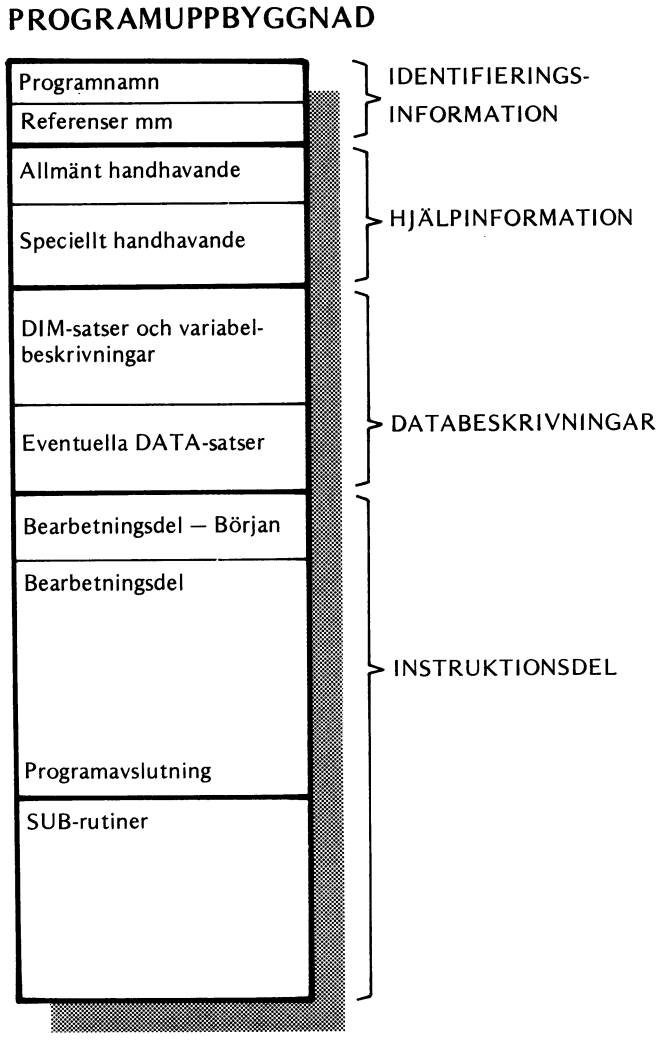
Det är också vår mening att du har mycket att vinna i tid och överskådlighet om uppbyggnad och dokumentation av dina program får en enhetlig utformning.



PROGRAM-UPPBYGGNAD

RIKTLINJER FÖR PROGRAMUPPBYGGNAD

Programspråket BASIC ger stor frihet vid planeringen av t ex DIM- och DATA-satser. Subrutiner och REM-satser kan i princip också sprängas in här och var i programtexten. Sättet att utnyttja BASIC-språkets möjligheter för att skapa en överskådlig programtext varierar med programmerarens erfarenhet och känsla för behovet av överskådlighet i samband med ändringar och dokumentation. Nedanstående figur är ett förslag till en lämplig mall för uppbyggnad av BASIC-program.

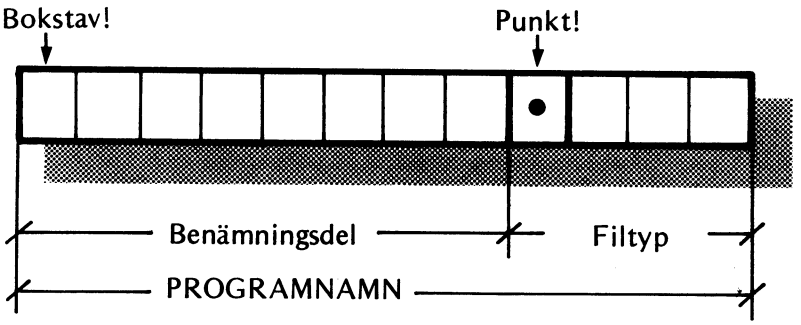


Om det finns starka motiv för detta kan givetvis avsteg från mallen göras. Observera att kravet på den separata dokumentationen utöver programtexten därvid kan komma att öka.

NAMNGIVNINGSREGLER

För att underlätta hantering, dokumentation, ändringar mm är det nödvändigt med enhetliga regler för namngivning av program.

Till förfogande för programnamn finns 12 teckenpositioner, varav den första måste vara en bokstav och den nionde måste vara en punkt. Om enbart benämningsdelen nyttjas för namngivningen fyller datorn själv i de tre sista positionerna med BAC eller BAS (programmet lagrat i okompilerad form till skillnad från BAC-angivelsen). Väljer du att själv använda tilläggstecken kan dessa lämpligen användas för att markera ny utgåva eller ändrad version. Siffror eller bokstäver kan nyttjas efter behag.



Programnamn bör väljas så att namnet ger information om programmets funktion.

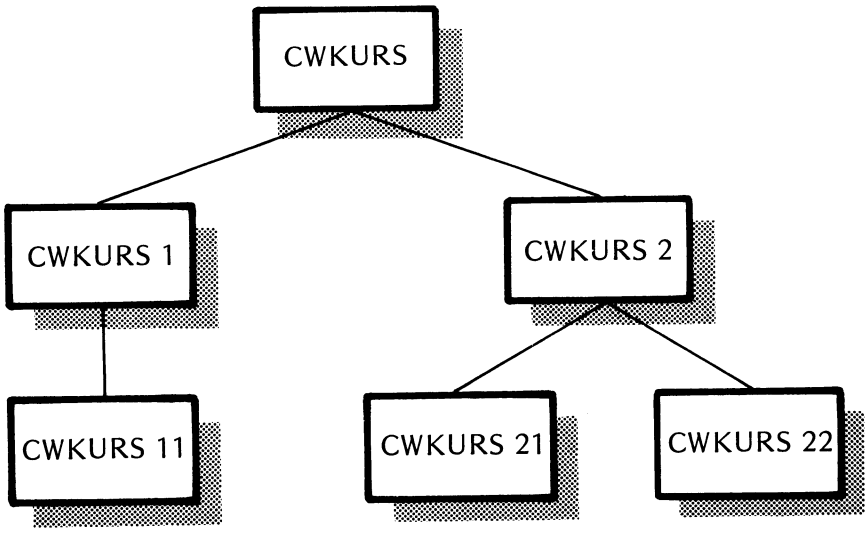
Om programmet består av ett huvudprogram och ett eller flera underprogram utformas huvudprogrammets benämningsdelar utan siffror.

Ex: CWKURS

I underprogrammen som anropas av huvudprogrammet med instruktionen CHAIN förses benämningsdelen med en siffra.

Ex: CWKURS1
CWKURS2

Figuren visar hur underprogram på olika nivåer namnges. Linjerna i figuren representerar möjliga programanrop.



De tre sista tecknen efter punkten i programnamnet kan vara utgåvebeteckning. Det första tecknet, som är en siffra, är versionsbeteckning och de två sista tecknen, som är bokstäver, är ändringsbeteckning.

Ex: CWKURS.1AB
CWKURS.1AC
CWKURS.3AB

Man bör alltid koppla programtexten till programmets namn genom att börja programmet med en PRINT-sats som innehåller programnamnet.

Ex: 10 PRINT "*** CWKURS.1AB ***"

REFERENSER

Omedelbart efter PRINT-satsen med programnamnet bör en REM-sats följa som innehåller uppgifter om namn och adress på programkonstruktören, samt datum för programversionens giltighetsstart.

Ex:
```
100 PRINT CHR¤(12)
110 REM ********************************
120 REM
130 PRINT "*** TEXTEX.1 ***"
140 REM
150 REM ********************************
160 REM * PROGRAMMET AVSETT FÖR ABC-80 *
170 REM * GÄLLER F O M  80-09-01       *
180 REM * KONSTRUKTÖR   A.BECOTTI      *
190 REM * ADRESS        DATAGATAN 80   *
200 REM *               STOCKALA       *
210 REM ********************************
220 REM
230 REM
```

DIM- OCH REM-SATSER

Alla DIM-satser samlas i början av programmet. I anslutning till varje DIM-sats anges med en REM-sats vad den aktuella variabeln ska användas till.

Ex:
```
120 REM ***************************************
130 REM .   DIM-satser och VARIABEL-
140 REM .        beskrivningar
150 REM .....................................
160 REM
170 REM
```
forts

```
180 DIM S(50) : REM .   INDATAFÄLT
190 DIM T(50) : REM .   FÖRSTA SORTERINGSAREA
200 DIM U(50) : REM .   ANDRA  SORTERINGSAREAN
210 DIM A¤(50)=12 : REM BENÄMNINGMATRIS
220 REM
230 REM S¤              ALLMÄN SVARSSTRÄNG
240 REM I% J% K%        LOOPVARIABLER
250 REM
260 REM
270 REM ....................................
```

VAL AV VARIABELBETECKNINGAR

Det är inte nödvändigt att reservera beteckningar för vissa data men det underlättar både programkonstruktionen och tolkningen av programmet om användningen av variabelbeteckningarna systematiseras.

Tabellen nedan är en rekommendation:

BETECKNING	DATASLAG	VARIABELTYP
A - H	konstanter/indata	flyttal
A% - H%	konstanter/indata	heltal
A¤ - H¤	konstanter/indata	strängvariabel
I - N	index/loopvariabler	flyttal
I% - N%	index/loopvariabler	heltal
O, O%, O¤	undvikes på grund av likheten med siffran 0	flyttal, heltal, strängvariabel
P - U	variabler för lagring av mellanresultat, s k slaskvariabler	flyttal
P% - U%		heltal
P¤ - U¤		strängvariabel
V - Z	resultatvariabel	flyttal
V% - Z%	resultatvariabel	heltal
V¤ - Z¤	resultatvariabel	strängvariabel

I variabelbeteckningarna ovan kan alla bokstäver ersättas med en bokstav kombinerad med en siffra 0 - 9.

I vissa printertyper motsvaras ABC-80:s valutatecken (¤) av dollartecknet ($). Båda har i BASIC-sammanhang samma betydelse och betecknar STRÄNGVARIABEL.

Bakgrunden till den här skillnaden är att den kod/teckenuppsättning som används på sådana här utrustningar kan variera från land till land på några punkter. Den teckenuppsättning (grafisk mod ej inräknad) som ABC-80 använder ansluter sig till version 2 av den svenska varianten av den internationellt standardiserade teckenuppsättningen ISO 646. Denna grundar sig på en något äldre internationellt rekommenderad industristandard CCITT Internationellt alfabet nr 5.

ANVÄNDNING OCH PLACERING AV SUBRUTINER

Sist i programmet samlas alla subrutiner. Inled varje subrutin med en REM-sats som beskriver subrutinens funktion.

Ex:
```
100 REM
110 REM
120 REM .......
130 REM
140 REM ********************************
150 REM :          SUBRUTINER
160 REM ................................
170 REM
180 REM
190 REM ======== UPP-PACKNING =========
200 REM .......
210 REM .......
220 RETURN
230 REM
240 REM ========= VARVRÄKNING =========
250 REM .......
260 REM .......
270 RETURN
280 REM
290 REM ========= DELSUMMA 2  =========
300 REM .......
310 REM .......
320 RETURN
330 REM
340 REM ********************************
```

ÅTGÄRDER VID "MINNET FULLT"

Ibland händer det att det tilltänkta programmet inte ryms i datorns minne. Du bör då systematiskt gå igenom programmet för att reducera dess storlek. De viktigaste åtgärderna är

- inför DIM-satser så att indexerade variabler inte tar mer plats än vad som är nödvändigt
- använd heltalsvariabler där så är möjligt. En heltalsvariabel tar mindre än hälften så stor plats i minnet som en flyttalsvariabel
- undersök om något programavsnitt förekommer flera gånger i programmet. Skriv i så fall avsnitten som en subrutin
- använd datafiler i stället för datasatser i programmet
- använd satsen ON ... GOSUB ... , ... vid flerval
- utnyttja möjligheten att skriva flera satser på samma programrad
- reducera antalet REM-satser. OBS! Detta medför att den separata programbeskrivningen måste bli utförligare

Även om ett program blir så omfattande att programtexten inte får plats i ABC-80 :s minne kan programmet ändå köras. Dela in programmet i huvudprogram och underprogram (jfr Namngivningsregler). Varje underprogram ska vara ett avgränsat programavsnitt som fungerar oberoende av huvudprogram och övriga underprogram.

När huvudprogrammet överförts till minnet och programkörningen startats anropar datorn underprogrammet med CHAIN-satser. Tekniken innebär i de flesta fall att datorn måste kunna anropa underprogram i godtycklig ordning. Detta medför normalt att underprogrammen måste vara lagrade på flexskiva. Endast i undantagsfall kan underprogrammen anropas i en sekvens och följaktligen finnas lagrade på kassettband.

STOMME TILL ETT PROGRAM

Vi ska nu belysa vad som sagts i detta kapitel om programuppbyggnad genom att redovisa en stomme till ett program. Programmet är fullt körbart och illustreras med kommentarer och bilder i anslutning till de olika avsnitten i programmet.

```
100 PRINT CHR¤(12) : REM RADERA SKÄRMEN
110 REM
120 REM ********************************
130 REM
140 PRINT "*** STOMME ***"
150 REM
160 REM ********************************
```

En programkörning börjar lämpligen med att radera skärmen och presentera programmets namn i bildskärmens första rad.

```
160 REM ********************************
170 REM * PROGRAMMET AVSETT FÖR ABC-80*
180 REM * GÄLLER  F O M  80-09-15     *
190 REM * KONSTRUKTÖR    A.BECEOTTI   *
200 REM * ADRESS         DATAGATAN 80 *
210 REM *                 STOCKALA     *
220 REM ********************************
```

Detta avsnitt innehåller de nödvändiga referenserna.

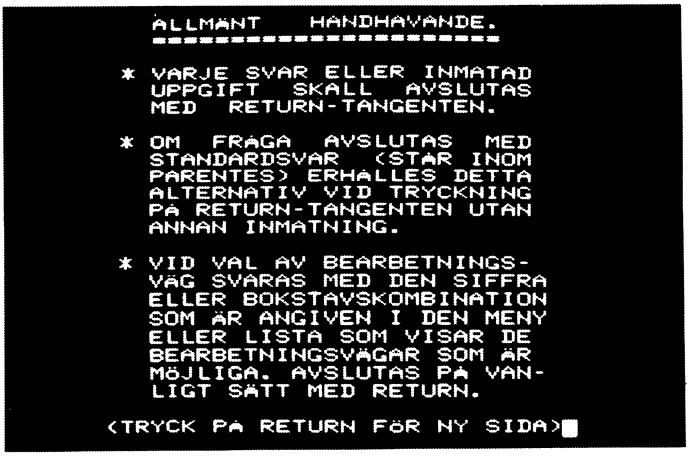
```
230 REM
240 PRINT
250 PRINT "BEHÖVER DU HJÄLP (NEJ)? ";
260 INPUTLINE S¤
270 S¤=LEFT¤(S¤,1)
280 IF S¤<>"J" AND S¤<>"j" THEN 1090
290 REM
300 REM ********************************
310 REM
320 REM
```

Här väljer man omfattningen av den hjälpinformation som ska presenteras

på bildskärmen. Enligt avsnittet VAL AV VARIABELBETECKNINGAR används S ¤ som en allmän svarsvariabel. Olika inmatningsmetoder beskrivs närmare i kapitlet SPECIELLA PROGRAMMERINGSRUTINER.

```
330 PRINT CHR¤(12)
340 PRINT "   ALLMÄNT   HANDHAVANDE."
350 PRINT "   ======================="
360 PRINT
370 PRINT " * VARJE SVAR ELLER INMATAD"
380 PRINT "   UPPGIFT  SKALL  AVSLUTAS"
390 PRINT "   MED  RETURN-TANGENTEN."
400 PRINT
410 PRINT " * OM  FRÅGA  AVSLUTAS  MED"
420 PRINT "   STANDARDSVAR  (STÅR INOM"
430 PRINT "   PARENTES) ERHÅLLES DETTA"
440 PRINT "   ALTERNATIV VID TRYCKNING"
450 PRINT "   PÅ RETURN-TANGENTEN UTAN"
460 PRINT "   ANNAN INMATNING."
470 PRINT
480 PRINT " * VID VAL AV BEARBETNINGS-"
490 PRINT "   VÄG SVARAS MED DEN SIFFRA"
500 PRINT "   ELLER BOKSTAVSKOMBINATION"
510 PRINT "   SOM ÄR ANGIVEN I DEN MENY"
520 PRINT "   ELLER LISTA SOM VISAR DE"
530 PRINT "   BEARBETNINGSVÄGAR SOM ÄR"
540 PRINT "   MÖJLIGA. AVSLUTAS PÅ VAN-"
550 PRINT "   LIGT SÄTT MED RETURN."
560 PRINT
570 PRINT "(TRYCK PÅ RETURN FÖR NY SIDA)";
580 GET S¤
```

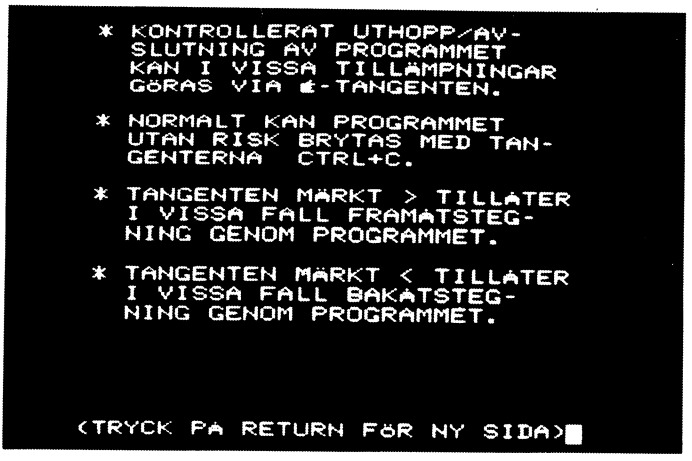
Här presenteras den första sidan med allmän hjälpinformation.



Bildskärmen har nu detta utseende.

```
590 PRINT CHR¤(12)
600 PRINT
610 PRINT " * KONTROLLERAT UTHOPP/AV-"
620 PRINT "   SLUTNING AV PROGRAMMET  "
630 PRINT "   KAN I VISSA TILLÄMPNINGAR"
640 PRINT "   GÖRAS VIA É-TANGENTEN."
650 PRINT
660 PRINT " * NORMALT KAN PROGRAMMET "
670 PRINT "   UTAN RISK BRYTAS MED TAN-"
680 PRINT "   GENTERNA  CTRL+C."
690 PRINT
700 PRINT " * TANGENTEN MÄRKT > TILLÅTER"
710 PRINT "   I VISSA FALL FRAMÅTSTEG-"
720 PRINT "   NING GENOM PROGRAMMET."
730 PRINT
740 PRINT " * TANGENTEN MÄRKT < TILLÅTER"
750 PRINT "   I VISSA FALL BAKÅTSTEG-"
760 PRINT "   NING GENOM PROGRAMMET."
770 PRINT : PRINT : PRINT
780 PRINT : PRINT
790 PRINT "(TRYCK PÅ RETURN FÖR NY SIDA)";
800 REM
810 GET S¤
820 REM ..............................
830 REM
```

Eventuellt behövs flera sidor hjälpinformation.



Detta visar då bildskärmen.

```
840 PRINT CHR¤(12)
850 PRINT "     SPECIELLT     HANDHAVANDE"
860 PRINT "     ========================="
870 PRINT
880 PRINT " *  HÄR FINNS PLATS FÖR KOMMEN-"
890 PRINT "    TARER BETRÄFFANDE DET HÄR"
900 PRINT "    PROGRAMMET T.EX."
910 PRINT
920 PRINT "    *  HANDHAVANDE"
930 PRINT
940 PRINT "    *  ÖVRIGA PROGRAM I SYSTEMET"
950 PRINT
960 PRINT "    *  AKTUELLA FILER"
970 PRINT
980 PRINT "    *  ......"
990 PRINT
1000 PRINT "     *  ......"
1010 PRINT
1020 PRINT "     *  ......"
1030 PRINT
1040 PRINT : PRINT : PRINT
1050 PRINT "(TRYCK PÅ RETURN FÖR START)";
1060 GET S¤
1070 REM
```

Som komplement till beskrivningen om allmänt handhavande behöver man i vissa tillämpningar presentera mera speciell hjälpinformation.

Bildskärmens utseende bestäms av tillämpningen.

```
1080 REM
1090 REM
1100 REM ******************************
1110 REM .     DIM-SATSER OCH VARIABEL-
1120 REM .          BESKRIVNINGAR.
1130 REM ..............................
1140 REM
1150 REM
1160 REM
1170 REM
1180 REM
1190 REM
1200 REM ..............................
1210 REM .          DATA-SATSER
1220 REM ..............................
1230 REM
1240 REM
1250 REM
1260 REM
```

Efter hjälpinformationen finns databeskrivningar.

```
1270 REM
1280 REM
1290 REM
1300 REM
1310 REM ******************************
1320 REM .      BEARBETNINGSDEL BÖRJAN
1330 REM ..............................
1340 REM
1350 REM .        BEARBETNINGSDEL
1360 REM
1370 REM
1380 REM
1390 REM
1400 REM
```

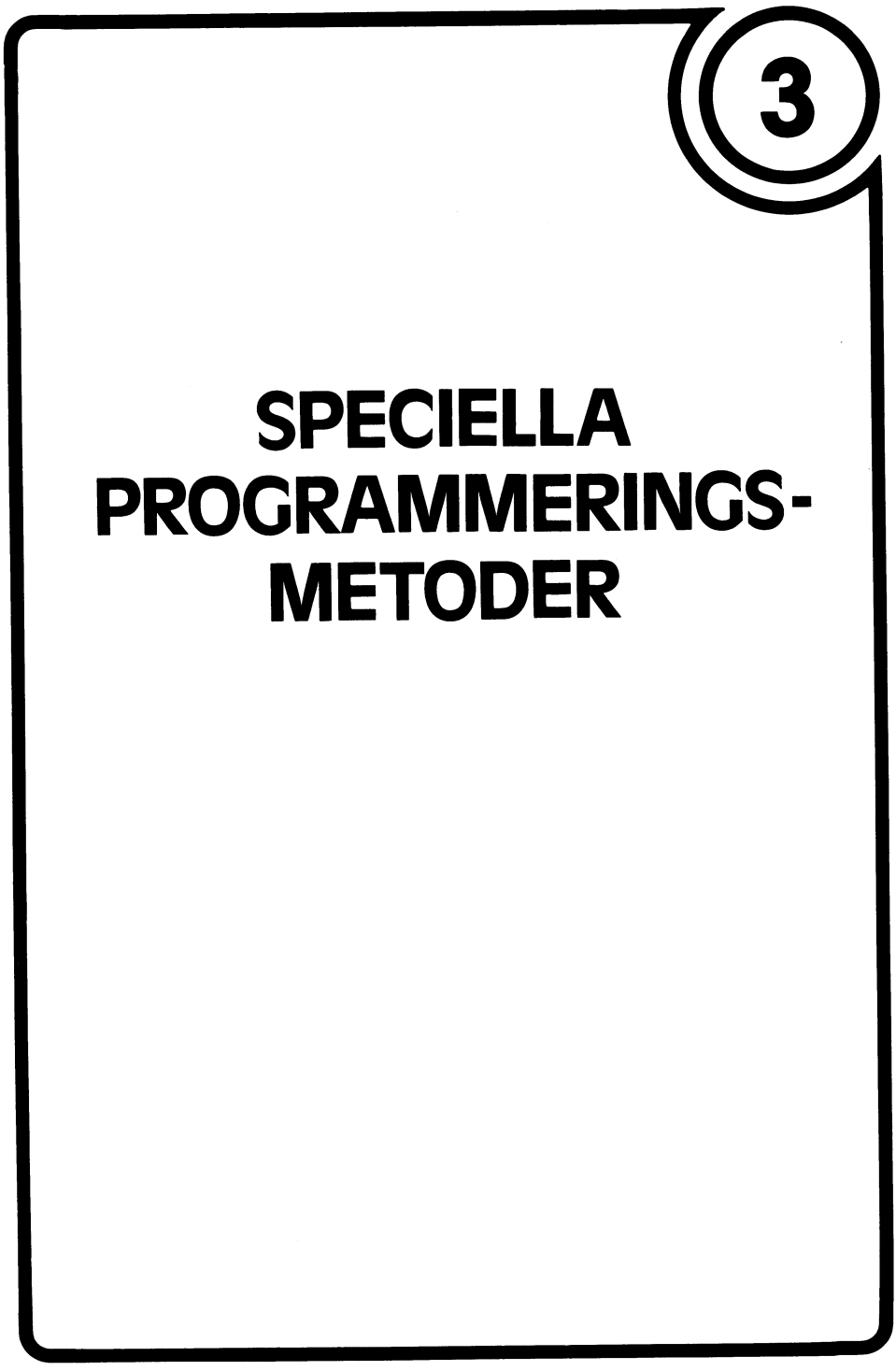
Här ingår den bearbetningsdel som karakteriserar det aktuella problemet.

```
1410 REM ..............................
1420 REM .          PROGRAMAVSLUTNING
1430 PRINT CHR¤(12)
1440 PRINT "    Omstart (JA)? ";
1450 INPUTLINE S¤
1460 S¤=LEFT¤(S¤,1)
1470 IF S¤<>"N" AND S¤<>"n" THEN 100
1480 REM
1490 PRINT CHR¤(12) : PRINT : PRINT
1500 PRINT "Körningen klar!"
1510 END
```

Programavslutningen innehåller en eller flera kontrollfrågor. Detta beskrivs närmare i kapitlet SPECIELLA PROGRAMMERINGSRUTINER. Programraden 1510 END utgör här programmets logiska avslutning.

```
1520 REM ******************************
1530 REM .         SUBRUTINER
1540 REM ..............................
1550 REM
1560 REM ======== SUBRUTIN-1 ==========
1570 REM
1580 REM
1590 RETURN
1600 REM ======== SUBRUTIN-2 ==========
1610 REM
1620 REM
1630 RETURN
1640 REM ======== SUBRUTIN-3 ==========
1650 REM
1660 REM
1670 REM
1680 RETURN
1690 REM ******************************
```

Efter programmets logiska slut och sist i programmets instruktionsdel ingår samtliga subrutiner. I ett väl strukturerat program utgör subrutinerna huvuddelen av instruktionsdelen.

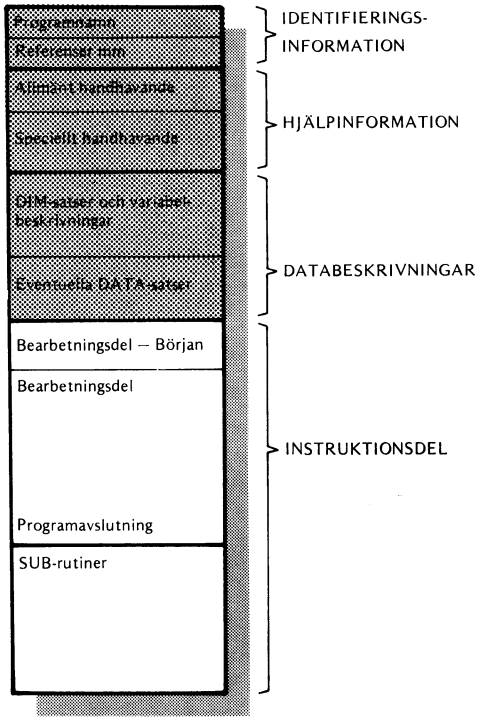


3

SPECIELLA PROGRAMMERINGS-METODER

STRUKTURERAD PROGRAMTEXT

I kapitel 2 har vi med ett exempel antytt hur programtext kan struktureras så att den blir överskådlig och därigenom lättare att förstå. Den visade programstommen kan ses som en samling fack där bestämd information ska placeras.



Även det fack som i stommen kallas för "instruktionsdelen" är möjligt att bygga – strukturera – på ett sätt som gör också den delen lätt hanterbar. Det är inte möjligt att här i detalj presentera hur "programbitarna" ska placeras i "instruktionsdelen", eftersom programbitarnas utseende helt beror på den aktuella tillämpningen. Exempel på hur instruktionsdelen kan struktureras kommer att redovisas i de kapitel som beskriver filhantering.

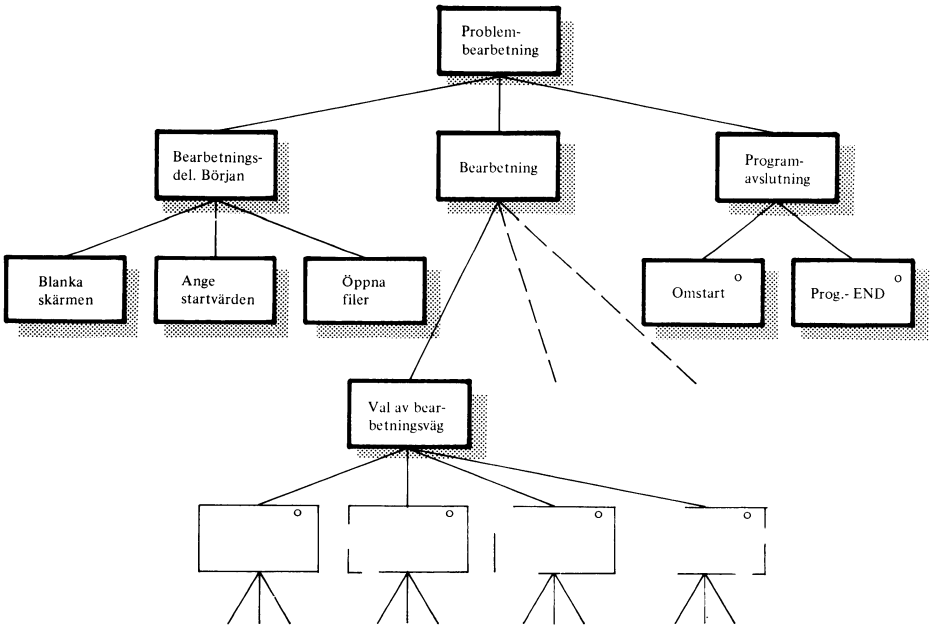
Vikten av en väl genomtänkt programstruktur kan inte nog understrykas. Så kallad "smartprogrammering", d v s att använda programtekniskt eleganta lösningar och överdrivet sofistikerade algoritmer medför fler olägenheter än fördelar. Sådana program tenderar ibland efter ett antal ändringar och tillägg att få karaktären av "spagettiprogram", d v s en samling instruktioner, som med hopp fram och tillbaka ger ett program som fullständigt saknar överskådlighet.

Utan att närmare gå in på vad som avses med begreppet strukturerad programmering antyder vi här principerna. Börja med att bryta ner programmet i olika bitar, inledningsvis i stora block för överskådlighetens skull. Dessa stora block spjälkas sedan ned i allt mindre enheter, vilka dock är logiskt och funktionsmässigt hopknutna. När nedbrytningen har nått sin lägsta nivå, kanske redan tidigare, börjar instruktionerna i den blivande programtexten att kunna skönjas.

Den enda grovindelning av ett program vi kan göra här utan kännedom om det aktuella problemet blir då följande – se fig

- Bearbetningsdel. Början
- Bearbetning
- Programavslutning

Speciella programmeringsrutiner avsedda att användas i dessa programblock redovisas senare i detta kapitel.



BEARBETNINGSDELENS BÖRJAN

Hur bör bearbetningsdelens början se ut? Ur användarens synpunkt är det ju trevligt att få starta arbetet vid datorn med ett oskrivet blad. Du bör alltså börja bearbetningsdelen med en sats som raderar skärmen av typen

Ex: `1350 PRINT CHR¤(12)`

Användaren bör sedan få en bekräftelse på att rätt program körs genom att programnamnet presenteras på skärmen. Sedan blir det lagom att låta programmet ge användaren frågan om han behöver ytterligare kör-information utöver den hjälpinformation som tidigare presenterats. Därmed torde de viktigaste uppstartningsaktiviteterna vara genomförda.

Oftast återstår dock ytterligare litet arbete "bakom kulisserna". Dit hör att ge vissa variabler lämpliga startvärden. Här bör speciellt observeras det fallet att programmet senare kan komma att köras från början igen utan att någonsin ha passerat en END- eller STOP-instruktion. Om en variabel vid programstarten skall innehålla värdet 0 är det då lätt att glömma en nollställningssats i början av programmet, eftersom ABC-80 vid uppstartandet av ett program ju automatiskt nollställer alla variabler.

Slutligen skall filer öppnas, om sådana ingår, och eventuella tabeller eller andra initieringsvärden läsas in.

Sammanfattningsvis omfattar bearbetningsdelens början följande punkter

- Radera skärmen
- Lägg ut programidentitet på skärmen
- Lägg ut eventuell körinformation
- Ge erforderliga startvärden. Glöm inte bort noll-värden där så behövs
- Öppna filer

SVARSTEKNIKER

En vanlig teknik för att styra programflödet är att ställa flervalsfrågor till användaren i form av menyer. Menyteknik beskrivs utförligt i ett särskilt kapitel.

Ofta kan emellertid programflödet styras av frågor med få alternativ. Det förhållandet att alternativen är få och svaren enkla bör inte förleda programmeraren att lämna ofullständiga uppgifter till användaren. Följande tips för frågeutformning sammanfattar de viktigaste synpunkterna.

- Varje uppgift som skall lämnas av användaren skall föregås av lämplig "ledtext" eller fråga
- Gör alla ledtexter entydiga och lättbegripliga
- Varje fråga bör bara resultera i ett svar
- Om flera alternativa svar är möjliga bör samtliga indikeras eller visas t ex inom parentes efter frågan
- Om frågan är av typen JA/NEJ kan det oftast förekommande svaret, standardsvaret, anges inom parentes efter frågan. För att erhålla just det svaret ska det vara tillräckligt med enbart nedtryckning av returtangenten. Tekniken är givetvis användbar även vid angivande av data om t ex vissa värden är normalt förekommande
- Felaktigt svar bör resultera i någon form av larm/felutskrift

Om programmet kräver stor aktivitet av användaren t ex i form av inmatning av många data, eller på grund av programmets omfattning, bör även följande saker beaktas för att underlätta för användaren.

- Lägg inte ut mer information på skärmen än vad den aktuella situationen kräver
- Kräv inte mer än en sak i taget av användaren
- Radera skärmen vid behov
- Se till att svaren från användaren kan göras korta
- Använd om möjligt standardsvar
- Varje svar eller uppgift som inmatats bör resultera i någon form av reaktion på skärmen som svar till användaren

OLIKA INMATNINGSMETODER

Inmatning av data via tangentbordet kan ske med instruktionerna GET, INPUT, INPUTLINE och INP(56). Fördelar och nackdelar med var och en av dessa ska belysas här. Nedan följer några förslag till minnesregler som sedan exemplifieras på de följande sidorna.

1a Mata helst in data till en strängvariabel när det inte är självklart att svaret ska utgöras av siffror

1b Omvandla inmatad strängvariabel till önskad variabeltyp

2 Gör inmatningen oberoende av mellanrum, space, när tal matas in.

3 Se till att oväntat stopp i programkörningen inte kan uppstå på grund av felaktig inmatning (se även FELHANTERING)

4 Standardisera inmatningen så att operatören bara behöver lära sig en inmatningsprocedur

Punkt 1a motiveras av att inmatning till en strängvariabel underlättar felhanteringen. Programavsnittet nedan ger avbrott i programkörningen om operatören av misstag trycker på returtangenten eller om vederbörande mot förmodan svarar TVÅ eller dyligt. (Datorn ger då felmeddelandet Err 12, d v s FELAKTIGT TAL).

```
100 PRINT "VILKET ANTAL ÖNSKAS";
110 INPUT S
120 GOTO 100
```

Om flyttalsvariabeln S byts mot strängvariabeln S¤ inträffar inte något avbrott i detta programavsnitt om antalet inmatade tecken understiger 80. Ändra alltså till:

```
100 PRINT "VILKET ANTAL ÖNSKAS";
110 INPUT S¤
120 GOTO 100
```

I "minnesreglerna" ovan är punkt 1b en följd av 1a. Eftersom inmatning ofta sker till en strängvariabel, måste då det följande programavsnittet innehålla en omvandling till önskad variabeltyp.

I den följande rutinen sker omvandlingen från strängvariabel till talvariabel med hjälp av strängfunktionen VAL (...) Om den inmatade strängen inte kan tolkas som ett tal tar instruktionen ONERRORGOTO ... hand om detta fel. Programexekveringen börjar då om från rad 100. (Se även kapitlet FELHANTERING)

```
100 PRINT CHR¤(12) : REM RADERA SKÄRMEN
110 PRINT "VILKET ANTAL ÖNSKAS";
120 ONERRORGOTO 110
130 INPUT S¤
140 S=VAL(S¤)
150 PRINT S
160 GOTO 110
```

Om inmatning av denna typ förekommer flera gånger i ett program bör inmatningskontrollen utföras som en subrutin. Rutinen är här också kompletterad med ett meddelande för att uppmärksamma användaren på att ett fel har begåtts. Observera också satsen ONERRORGOTO 0 som har en speciell innebörd. Detta utreds närmare i kapitlet FELHANTERING.

```
100 PRINT CHR¤(12) : REM RADERA SKÄRMEN
110 PRINT "VILKET ANTAL ";
120 INPUT S¤
130 GOSUB 210
140 GOTO 110
150 ;
160 ;
170 REM ============================
180 REM .         SUBRUTIN
190 REM ============================
200 REM
210 REM --- INMATNINGSKONTROLL ----
220 REM
230 ONERRORGOTO 250
240 S=VAL(S¤) : GOTO 280
250 PRINT
260 PRINT "SVARA MED ETT TAL!"
270 PRINT : PRINT
280 ONERRORGOTO 0
290 RETURN
```

Ovanstående enkla programexempel fungerar eftersom själva inmatningen sker i en oändlig loop.

I ett normalt program måste huvudprogram och subrutin samarbeta så att om ett fel upptäcks i subrutinen ska huvudprogrammet kräva en förnyad inmatning.

Principen kan utformas på många sätt. Se exempel på nästa sida.

Ett exempel på hur denna princip kan utformas ges i följande program

```
100 PRINT CHR¤(12) : REM - radera skärmen
110 REM ........................
120 REM
130 PRINT "VILKET ANTAL  (st) ";
140 INPUT S¤
150 GOSUB 330
160 IF R=1 THEN 130 : REM - fel inmatning
170 S1=S
180 REM ........................
190 REM
200 PRINT "VILKEN VIKT/ST (g) ";
210 INPUT S¤
220 GOSUB 330
230 IF R=1 THEN 200 : REM - fel inmatning
240 S2=S
250 PRINT
260 GOTO 130
270 ;
280 ;
290 REM ========================
300 REM .        SUBRUTIN
310 REM ========================
320 REM
330 REM --- INMATNINGSKONTROLL ---
340 REM
350 R=0 : REM nollställ felvariabeln
360 REM
370 ONERRORGOTO 390
380 S=VAL(S¤) : GOTO 430
390 PRINT "SVARA MED ETT TAL!"
400 R=1 : REM . R=1  indikerar här att
410 REM .       fel har uppstått.
420 PRINT : PRINT
430 ONERRORGOTO 0
440 RETURN
```

Här lagras informationen om eventuellt fel i inmatningen i variabeln R. I utgångsläget sätts alltid R lika med noll för att indikera att inget fel hittills har upptäckts. Med satsen ONERRORGOTO sker ett hopp om fel upptäcks och variabeln R sätts då till ett.

I programmets bearbetningsdel på raderna 100--280 kontrolleras värdet av variabeln R efter varje ny inmatning. Om R är lika med ett kräver programmet en förnyad inmatning efter det att felmeddelande skrivits ut enligt subrutinen.

I exemplet ovan är förutsättningen för subrutinen att inmatningen alltid sker till strängvariabeln S¤ . (INPUT-satsen kan emellertid också med fördel placeras i subrutinen). I huvudprogrammet överförs sedan värdet av variablen S till variablerna S1, S2, o s v.

Inledningsvis gavs förslag till minnesregler och enligt punkt 2 bör inmatningen göras oberoende av mellanrum, space, när tal matas in. Om operatören oavsiktligt trycker på mellanslagstangenten innan de rätta siffrorna matas in, eller efter det att siffrorna matats in, kan detta inte lätt observeras med hjälp av utskriften på bildskärmen. Det är därför ett rimligt krav att svaret ger en korrekt inmatning oberoende av antalet mellanrum.

Om instruktionen INPUT används behöver inga extra åtgärder vidtas för detta. Alla mellanslag negligeras när en variabel tilldelas ett värde via en INPUT-sats, såvida inte inmatade data omges med citationstecken.

I punkt 3 nämns fortsättningsvis att avbrott i programkörningen, d v s oväntade stopp, måste undvikas. Detta finns närmare beskrivet i kapitlet FELHANTERING. Här ska bara nämnas att om inmatning sker till en strängvariabel med INPUT eller INPUTLINE kan avbrott i programkörningen inte ske om strängvariabeln är korrekt dimensionerad.

Punkt 4 i minneslistan avser standardiseringen av inmatningsrutinerna. Det är en fördel för operatören om de flesta frågorna kan besvaras enligt samma procedur. Exempelvis avslutas en inmatning med RETURN om INPUT- eller INPUTLINE- sats används. Däremot är detta inte fallet om en GET-sats används. Eftersom GET-satsen är speciell därvidlag är det mindre lämpligt att använda denna vid inmatning utom i speciella situationer. Inmatning av ett enda tecken sker enklast via en GET-sats. Exempel på användning kan vara programstyrd frammatning av text på bildskärmen. När man vill tilldela vissa tangenter speciella funktioner är också användning av GET-sats det naturligaste.

Hittills har fördelarna med att använda INPUT-sats poängterats. I vissa fall är mellanrum i en inmatad strängvariabel väsentlig och då bör INPUT-satsen inte användas, eftersom inmatad sträng då måste omges av citationstecken. Inmatning av löpande text sker med fördel till en strängvariabel via en INPUTLINE-sats. Observera dock att strängvariabelns två sista element alltid blir vagnretur och radframmatning.

En möjlighet att hämta data från tangentbordet utan att programkörningen avbryts erbjuder funktionen INP(56). Denna funktion ger den senast nedtryckta tangentens ASCII-värde som resultat. Under den tid som tangenten hålls nedtryckt är funktionsvärdet lika med summan av ASCII-värdet och talet 128. Detta gör det möjligt att programvarumässigt avgöra om tangenten fortfarande hålls nedtryckt. Prova nedanstående programexempel och tryck ned olika tangenter under programkörningen.

```
100 ; CHR¤(12) : REM radera skärmen
110 A=INP(56) : REM avläs tangentbordet
120 ; A,CHR¤(A)
130 GOTO 110
```

NÅGRA INMATNINGSEXEMPEL

Avslutningsvis följer här som exempel tre olika utföranden på inmatningsrutin, där operatören ska ge svaret JA eller NEJ på en presenterad fråga. Som svar NEJ uppfattas här alla uttryck, bestående av ett eller flera tecken som börjar med n eller N. Alla andra uttryck accepteras som standardsvaret JA.

1. Rutinen utförd med hjälp av GET-sats:

```
100 REM .......
110 ;
120 PRINT "ÖNSKAS FLERA ALTERNATIV ?(JA)"
130 ;
140 ;
150 GET S¤
160 IF S¤<>"N" AND S¤<>"n" THEN 200
170 ;
180 ;
190 PRINT "SVAR  N E J  HAR GIVITS" : END
200 PRINT "SVAR  J A  HAR GIVITS" : END
210 REM .......
```

2. Rutinen utförd med INPUT-sats:

```
100 REM .......
110 ;
120 PRINT "ÖNSKAS FLERA ALTERNATIV (JA);"
130 ;
140 ;
150 INPUT S¤
160 IF LEN(S¤)=0 THEN 210
170 IF LEFT¤(S¤,1)="N" THEN 220
180 IF LEFT¤(S¤,1)="n" THEN 220
190 ;
200 ;
210 PRINT "SVAR  J A  HAR GIVITS" : GOTO 110
220 PRINT "SVAR N E J HAR GIVITS" : GOTO 110
230 REM .......
```

3. Rutinen utförd som subrutin med INPUT-sats:

```
100 REM .......
110 ;
120 PRINT "ÖNSKAS FLERA ALTERNATIV (JA)";
130 ;
140 GOSUB 300
150 ;
160 ON R GOSUB 240,270
170 ;
180 ;
190 GOTO 120
200 REM ================================
210 REM .        SUBRUTINER
220 REM ================================
230 REM
240 REM ------------ JA ----------------
250 PRINT "SVAR J A HAR GIVITS" : RETURN
260 REM
270 REM ------------ NEJ ---------------
280 PRINT "SVAR N E J HAR GIVITS" : RETURN
290 REM
300 REM ------- SVARSKONTROLL ----------
310 REM små bokstäver blir STORA
320 ;
330 ;
340 S1¤="" : REM startvärde = tom sträng
350 INPUT S¤
360 FOR N=1 TO LEN(S¤)
370 S=ASC(MID¤(S¤,N,1))
380 IF S>95 AND S<127 THEN S=S-32
390 S1¤=S1¤+CHR¤(S)
400 NEXT N
410 R=1 : REM svarsalternativ ja
420 S¤=S1¤
430 IF LEN(S¤)=0 THEN 450
440 IF LEFT¤(S¤,1)="N" THEN R=2 : REM nej
450 RETURN
460 REM ------------------------------
```

Den första rutinen är onekligen kortast och tar också minst minnes-

utrymme i anspråk. Nackdelen med rutinen är att programkörningen fortsätter direkt efter det att någon tangent tryckts ned. Möjligheten att korrigera eventuella misstag finns alltså inte.

Vid inmatning med INPUT som i rutin 2 och 3 är denna nackdel eliminerad. Detta sker dock till priset av att kontrollen av den inmatade strängen blir mer omfattande. Fallet att strängens längd är noll måste beaktas särskilt eftersom funktionen LEFT¤ inte kan användas på en tom sträng. Enbart en tryckning på returtangenten ger t ex som resultat en tom sträng.

I rutin 3 är inmatningen oberoende av om den inmatade strängen innehåller en blandning av små och stora bokstäver. En generell rutin för teckenomvandling ingår, så att små bokstäver i inmatad sträng ändras till stora bokstäver (se vidare i nästa avsnitt).

Den subrutin som används i alternativ 3 kan också lätt utökas så att ytterligare svarsalternativ kan utvärderas om så behövs. Denna rutin är också konstruerad så att strukturerad programmering underlättas. Svarsalternativen är här omvandlade till numeriska värden som kan användas för subrutinanrop med satsen ON ... GOSUB ... , ...

Man skulle kunna tro att inmatningsrutiner utförda enligt alternativ 3 tar stort minnesutrymme i anspråk. Praktiska prov visar att om antalet inmatningar överstiger 10 är rutin 3 den som kräver det minsta minnesutrymmet av alla de tre redovisade exemplen.

SUBRUTIN FÖR SMÅ BOKSTÄVER TILL STORA

För att underlätta arbetet för operatören är det lämpligt att göra inmatning av data oberoende av om det är små eller stora bokstäver som matas in (jämför t ex BASIC-tolken som i detta avseende är oberoende av vad som matas in).

En subrutin som sköter denna omvandling följer här:

```
100 REM .......
110 REM
120 PRINT "MATA IN EN STRÄNG"
130 INPUT S¤ : GOSUB 220
140 ;
150 PRINT S¤
160 GOTO 120
170 ;
180 REM ================================
190 REM .        SUBRUTIN
200 REM ================================
210 REM
220 REM --- BOKSTAVSKONVERTERING ---
230 REM
240 S1¤="" : REM startvärde = tom sträng
250 FOR N=1 TO LEN(S¤)
260 S=ASC(MID¤(S¤,N,1))
270 IF S>95 AND S<127 THEN S=S-32
280 S1¤=S1¤+CHR¤(S)
290 NEXT N
300 S¤=S1¤
310 RETURN
320 REM --------------------------------
```

Omvandlingen tillgår så att alla element i inmatad sträng undersöks i tur och ordning. Efter kontroll av att ASCII-värdet ligger mellan 95 och 127, d v s motsvarande små bokstäver, sker omvandling till motsvarande stora bokstävers ASCII-värden genom subtraktion av talet 32 (jämför med ASCII-tabellen i programmeringskortet). I strängen S1¤ lagras efter hand alla omvandlade tecken.

VÄNTETIDER

Ett program är normalt så konstruerat att resultatet av datorns arbete presenteras utan uppehåll på bildskärm eller skrivare.

I ett program med lång exekveringstid uppstår ibland situationen att datorn under flera sekunder bearbetar data utan att något meningsfullt resultat kan presenteras.

Man bör då inte gå in för att presentera meningslösa mellanresultat eftersom dessa bara skapar förvirring.

Eftersom programmeraren ändå vill låta operatören leva i trygg förvissning om att datorn fortfarande arbetar normalt, måste något meddelande ges på bildskärmen, om väntetiden är längre än någon sekund.

Lämpligt är då att konstruera en subrutin, som ger ett pausmeddelande. I följande exempel är bearbetningsdelen simulerad med en fördröjningsloop som tar en sekund och upprepas 12 gånger. Återstående väntetid visas kontinuerligt på bildskärmen. Efter en lång exekveringstid kan det ibland vara praktiskt att datorn ger en ljudsignal (PRINT CHR¤ (7)) för att ge besked om att körningen är klar.

```
100 REM .......
110 REM
120 PRINT CHR¤(12)
130 PRINT "DATORN ARBETAR OFÖRTRUTET !"
140 PRINT "AV VÄNTETIDEN ÅTERSTÅR";"      SEKUNDER"
150 REM
160 REM LÅNG EXEKVERINGSTID
170 U=12
180 GOSUB 280
190 IF U=0 THEN 230
200 FOR K=1 TO 1000 : NEXT K
210 U=U-1
220 GOTO 180
230 PRINT CHR¤(12)
240 PRINT "KÖRNINGEN KLAR"
250 PRINT CHR¤(7) : END
260 REM
270 REM ================================
280 REM .    SUBRUTIN PAUSFÅGEL
290 REM ================================
300 PRINT CUR(2,23);U;" "
310 RETURN
320 REM --------------------------------
```

PROGRAMAVSLUTNING

Program, som är av engångstyp, d v s som bara finns anledning att genomlöpa en enstaka gång vid en bearbetning, kan avslutas med direkt utgång genom någon av instruktionerna END eller STOP.

Lämpligt är dock att alldeles före dessa instruktioner lägga ut en PRINT-sats med t ex följande text: KÖRNINGEN KLAR, eventuellt föregånget av en radering av skärmen under förutsättning att denna inte innehåller resultat av intresse.

Om körningen utförs med hjälp av flera programavsnitt, som anropas med CHAIN-instruktionen är det ofta viktigt att veta i vilket skede bearbetningsprocessen befinner sig, varför avslutningsfasen ovan bör kompletteras med programnamnet t ex KASSABOK – INMATNING, KÖRNINGEN KLAR.

Många gånger är dock programmet av sådan karaktär att en ny körning är aktuell omedelbart efter en fullföljd bearbetning. Givetvis är det inte så svårt att på nytt starta körning med RUN, men det finns tillfällen då man önskar ett ännu enklare återstartsförfarande. Så är t ex fallet när en uppstartning med RUN innebär att ett antal startparametrar måste matas in medan det i själva verket bara är en enstaka indataparameter som behöver ändras jämfört med föregående körning.

Många program kräver att körningen avslutas med att bearbetat material sparas på ett externminne. För sådana program är en styrd programavslutning nödvändig. Då kan uppbromsningen för den manuellt styrda programavslutningen tjäna som ett tvåhandsgrepp. Med lämpligt utformad text utlagd på skärmen ges användaren en påminnelse, så att han kan ta ställning till om bearbetningen verkligen skall avslutas.

Slutligen skall man inte underskatta den psykologiska betydelse det innebär att den styrda programavslutningen låter människan känna att det är hon som styr maskinen och inte tvärt om!

Till sist följer här ett par programexempel som illustration :

Ex 1:

```
100 PRINT CHR¤(12)
110 PRINT "****************************"
120 PRINT "*  PROGRAMAVSLUTNING EX.1  *"
130 PRINT "****************************"
140 REM
150 REM -------- PROGRAM-START --------
160 FOR K=1 TO 5
170 FOR L=1 TO 7
180 PRINT ".";
190 NEXT L
200 PRINT
210 NEXT K
220 REM
230 REM
240 REM ------ PROGRAMAVSLUTNING ------
250 REM
260 PRINT "ÅTERSTART AV PROGRAMMET? (JA)";
270 INPUT S¤ : PRINT
280 IF LEN(S¤)=0 THEN 150
290 IF LEFT¤(S¤,1)="N" THEN 320
300 IF LEFT¤(S¤,1)="n" THEN 320
310 GOTO 150
320 PRINT CHR¤(12)
330 PRINT "*  PROGRAMAVSLUTNING EX.1  *"
340 PRINT : PRINT : PRINT "KÖRNINGEN KLAR!"
350 END
```

Ex 2:

```
100 PRINT CHR¤(12)
110 PRINT "****************************"
120 PRINT "*  PROGRAMAVSLUTNING EX.2  *"
130 PRINT "****************************"
140 REM --- BEARBETNINGSDEL BÖRJAN ---
150 REM
160 REM ------ BEARBETNINGS-TYP ------
170 ;
180 PRINT "VÄLJ BEARBETNING (I/U/S): ";
```

(forts)

```
190 INPUT S¤
200 IF LEN(S¤)=0 THEN 250
210 S¤=LEFT¤(S¤,1)
220 IF S¤="I" OR S¤="i" THEN 270
230 IF S¤="U" OR S¤="u" THEN 340
240 IF S¤="S" OR S¤="s" THEN 410
250 PRINT "OTILLÅTEN BEARBETNINGSTYP!";
260 PRINT CHR¤(7) : GOTO 160
270 REM --------- INMATNING ----------
280 A¤="DIV. INMATNINGSRUTINER"
290 GOSUB 580
300 REM
310 REM
320 R=1 : REM alternativet INMATNING valt
330 GOTO 160
340 REM --------- UTMATNING ----------
350 A¤="UTMATNING!!"
360 GOSUB 580
370 REM
380 REM
390 R=2 : REM alternativet UTMATNING valt
400 GOTO 160
410 REM ------ PROGRAMAVSLUTNING ------
420 ;
430 PRINT "ÅTERSTART AV PROGRAMMET? (NEJ)";
440 INPUT S¤ : PRINT
450 IF LEN(S¤)=0 THEN 480
460 S¤=LEFT¤(S¤,1)
470 IF S¤="J" OR S¤="j" THEN 140
480 IF R<>1 GOTO 550
490 PRINT
500 PRINT "DEN SISTA INMATNINGEN ÄR"
510 PRINT "I N T E   U T M A T A D! ";CHR¤(7)
520 R=3 : REM stopp utan utmatning
530 PRINT
540 GOTO 410
550 PRINT CHR¤(12) : PRINT : PRINT N¤
560 PRINT "KÖRNINGEN KLAR!"
570 END
580 REM ====== SUBRUTIN UTSKRIFT ======
590 FOR K=1 TO 5
600 PRINT A¤
610 NEXT K
620 PRINT
630 RETURN
640 REM ------------------------------
```

Det första exemplet visar hur en enkel rutin för återstart kan se ut. Den andra rutinen avser att belysa att kontroller bör läggas in i programmet så att inte utmatningsdelen kan förbigås utan att användaren uppmärksammas på detta. Under inmatningsdelen (I) sätts variabeln R till ett och under utmatningsfasen (U) sätts R till två. Värdet av variabeln R blir sedan avgörande för vilken programavslutning (S) som presenteras för användaren. Önskar användaren avsluta körningen utan att utföra någon utmatning sätts variabeln R till tre. När avsnittet programavslutning på nytt genomlöps leder detta till att programkörningen avslutas.

MENYTEKNIK

VAD ÄR EN MENY?

Med ordet meny menar man i databehandlingssammanhang den "matsedel", som kan behöva presenteras för en användare som information till denne om vilka möjligheter som programmet har. I menybegreppet ligger även användarens möjlighet till att välja vad han önskar ur menyn. För datorer utrustade med bildskärm är den naturligaste menytekniken att presentera en lista med någon form av sökindex eller menypekare.

```
*** DATAHANTERING ***

    1  INMATNING
    2  UTSKRIFT
    3  UPPDATERING
    4  FELSTATISTIK
    5  SUMMERING
    6  SAMMANFATTNING
```

sökindex
menypekare

Siffrorna 1--6 i ovanstående menyexempel kallas här sökindex eller menypekare. Vid första påseendet verkar det vara en självklar metod att låta sökindex i menyn vara en eller flera siffror. Metoden blir inte fullt så självklar när vi tittar lite närmare på för- och nackdelar och jämför med alfabetiskt respektive alfanumeriskt sökindex.

Det finns ytterligare en menyteknik som kan vara intressant för vissa tillämpningar. Den innebär att markören används för att peka ut önskad aktivitet och även den varianten ska vi titta på.

Efter genomgång av de olika typerna av menypekare presenteras ett programexempel där alla varianterna förekommer. I anslutning till varje menytyp kommenteras motsvarande avsnitt i programexemplet.

NUMERISK MENYPEKARE

Den numeriskt styrda menyn anses ofta vara den bästa. Det är dock diskutabelt för vem den är bäst, för användaren eller för programmeraren.

Ur programmerarens synvinkel är det skenbart en bra teknik eftersom den tillåter användning av satsen ON ... GOTO ... , ... Men vad händer om användaren ger sig utanför de tillåtna numeriska menypekarna, eller kanske rentav anger en alfabetisk variabel?

För att klara detta måste du alltså innan instruktionen ON ... GOTO ... , ... används göra två kontroller

- Rimlighetskontrollera det inmatade värdet
- Konstatera att det inmatade värdet inte ligger utanför tillåtna gränser

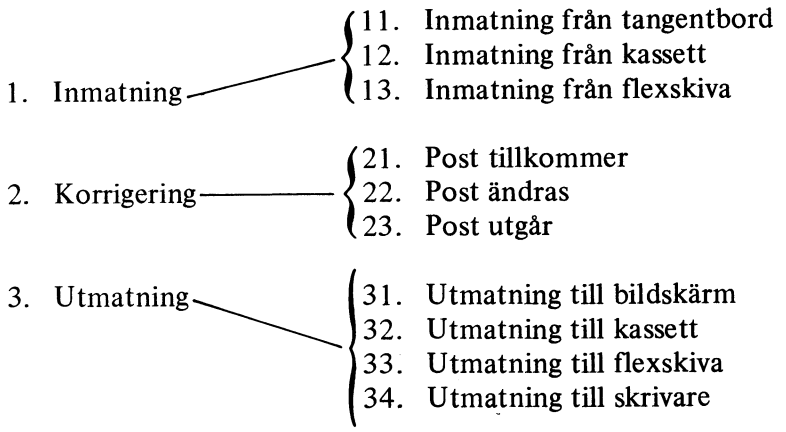
Rimlighetskontrollen innebär här att ickenumeriska värden fångas upp med instruktionen ONERRORGOTO ... för att undvika ej önskade programstopp. Användning av instruktionen ONERRORGOTO ... ingår i en teknik som brukar kallas felhantering. Denna teknik beskrivs utförligt i kapitlet FELHANTERING men används redan här i viss utsträckning. Gränskontrollen ska ta hand om felaktiga numeriska värden.

En nackdel med en numerisk menypekare är att den inte tillåter utnyttjande av ickenumeriska tecken såsom ← för bakåtstegning eller S för stopp/sluta o s v.

Ur användarens synpunkt kan det ibland också uppfattas som ett irritationsmoment att ofta tvingas läsa igenom långa menyer med sifferpekare för att finna rätt siffra. Sådana situationer kan uppkomma om menyer med sifferpekare förekommer ofta. Det kan då bli svårt att komma ihåg vad en viss siffra betyder i en given situation.

En numeriskt styrd meny bör endast komma ifråga vid små menyer och i program där endast en meny förekommer. Om du ändå anser att du måste ha flera sådana menyer i samma program, se till att du hjälper

användaren genom att använda hierarkiska sökindex. Ett exempel visas i figuren.



EXEMPEL PÅ NUMERISK MENYPEKARE

Nedanstående foto visar den numeriska menyns utseende på bildskärmen i vårt programexempel.

Motsvarande avsnitt i programtexten utgörs av nedanstående programrader:

```
380 PRINT CHR¤(12)
390 ONERRORGOTO 380
400 REM
410 PRINT "*** MENYEXEMPEL-1 ***"
420 PRINT
430 PRINT "**********************"
440 PRINT "* (NUMERISK PEKARE) *"
450 PRINT "**********************"
460 PRINT
470 PRINT " 1.   NUMERISK       MENY"
480 PRINT " 2.   ALFABETISK     MENY"
490 PRINT " 3.   ALFANUMERISK   MENY"
500 PRINT " 4.   MARKÖRSTÖDD    MENY"
510 PRINT " 5.   MARKÖRSTYRD    MENY"
520 PRINT " 6.   SLUT"
530 PRINT
540 GOSUB 2030
550 INPUT S
560 IF S<1 OR S>6 THEN GOSUB 1950 : GOTO 380
570 ON S GOTO 380,590,860,1080,1400,1780
```

PRINT-satserna på raderna 410 -- 530 presenterar den text som är speciell för den aktuella menyn.

Eftersom viss del av texten är gemensam för flera menyer har motsvarande PRINT-satser placerats i en subrutin kallad LEDTEXT som anropas på raden 540 GOSUB 2030

```
2030 REM --------- LEDTEXT ------------
2040 REM
2050 PRINT "SÖK ÖNSKAD MENY MED HJÄLP AV"
2060 PRINT "ANGIVNA MENYPEKARE OMEDEL-"
2070 PRINT "BART FÖLJT AV RETURN."
2080 PRINT : PRINT "VILKEN MENY VÄLJER DU ";
2090 RETURN
```

På rad 550 inmatas sökindex till variablen S. Rimlighetskontrollen av

sökindex sker på två sätt. Först kontrolleras att det inmatade talet är en siffra. Raden

```
390 ONERRORGOTO 380
```

medför att programexekveringen fortsätter på rad 380 om inmatat sökindex inte är numeriskt. Se vidare kapitlet FELHANTERING.

Därefter kontrolleras på rad 560 att inmatat sökindex ligger i rätt intervall. Om sökindex ligger utanför tillåtet intervall anropas en subrutin kallad FELTEXT.

```
1950 REM --------- FELTEXT ------------
1960 REM
1970 PRINT "FELAKTIGT VÄRDE!";CHR¤(7)
1980 FOR N=1 TO 1500 : NEXT N : REM .VÄNTA !
1990 PRINT CHR¤(12)
2000 RETURN
```

Datorn presenterar då en hjälptext och uppmärksammar användaren på detta med en ljudsignal. Därefter visas menyn återigen på bildskärmen.

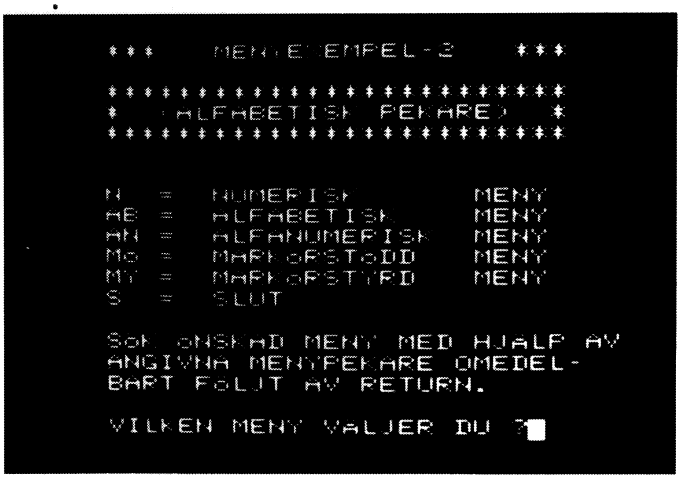
Det kontrollerade värdet på variabeln S, sökindex, kan nu utan risk för fel användas på rad 570, där S används för att utpeka det önskade programavsnittet.

ALFABETISK MENYPEKARE

Fördelen med att använda en alfabetisk pekare är att den kan väljas så att dess betydelse är lätt att komma ihåg. Antalet tecken kan varieras efter behov och hierarkiska pekare kan skapas även här.

Ur programkonstruktionssynpunkt blir antalet hoppsatser givetvis lika många som antalet pekarlägen. Vidare är det nödvändigt att kunna ändra eventuellt inmatade små bokstäver till stora, alternativt att göra avkänning av även små bokstäver i anslutning till IF-satserna. Rimlighetskontroll är aktuell även här för att avkänna otillåtna kombinationer.

Fotot visar ett exempel på en alfabetisk meny.



Motsvarande programavsnitt återfinner du i programexemplet på raderna 590--840.

Inmatningen av sökindex sker nu till en strängvariabel. Denna rutin är gemensam för flera menyer och har därför placerats i en subrutin kallad BOKSTAVSKONVERTERING, som anropas på rad 760 (se programexemplet). En subrutin för bokstavskonvertering har tidigare diskuterats.

```
760 GOSUB 2120
770 IF S¤="N" THEN GOTO 380
780 IF S¤="AB" THEN GOTO 590
790 IF S¤="AN" THEN GOTO 860
800 IF S¤="MÖ" THEN GOTO 1080
810 IF S¤="MY" THEN GOTO 1400
820 IF S¤="S" THEN GOTO 1780
830 GOSUB 1950 : GOTO 590
```

Uthopp till önskat programavsnitt styrs av de sex IF ... THEN ...-satserna. Rimlighetskontrollen tas här om hand på rad 830.

ALFANUMERISK MENYPEKARE

Här sker valet liksom i förra fallet genom inmatning av pekarvärdet till en sträng. Skillnaden är dock att det här är frågan om alfanumeriska tecken. Tack vare de möjligheter till stränghantering som finns i ABC-80:s BASIC kan vi återvinna önskad information ur inmatad sträng. Vi kan t ex med VAL(S¤) bestämma värdet av ett stränginmatat tal och den vägen skapa ett numeriskt värde. Detta hanteras sedan som i det numeriska fallet ovan. Ur användarens synpunkt är fallen likvärda. Ur programsynpunkt erhålls en viss förenkling. Gränskontroller måste dock givetvis göras även här.

Bilden visar ett exempel på en alfanumerisk meny.

Motsvarande programavsnitt återfinner du i programexemplet på raderna 860--1060. Jämför här beskrivningen i avsnittet Numerisk Menypekare.

MARKÖRSTYRD PEKARE

Ett för användaren annorlunda sätt att markera sitt önskemål är att använda markörförflyttning. Den bakomliggande programtekniken är något omständligare än i de tidigare relaterade fallen. Markörförflyttningen sker med hjälp av PRINT CUR(R, K) och styrs av två tangenter för radvis fram- respektive bakåtstegning. Vid stegningen sker en upp- eller nedräkning av en pekarvariabel, som från början tilldelats ett definierat startvärde beroende på menyns startposition på skärmen.

Pekarvariabeln används sedan dels för att positionera markören och dels som basvärde för den ON ... GOTO ... , ...-sats som styr det programhopp som skall göras. Även här är det nödvändigt med gränsvärdeskontroller.

Ett exempel på en markörstyrd pekare visar detta foto.

Studera motsvarande avsnitt i programexemplet på raderna 1400--1730.

MARKÖRSTÖDD PEKARE

Slutligen skall här nämnas möjligheten till kombination av tidigare presenterade tekniker.

Vid stora menyer blir ren markörstyrning av pekaren omständlig. Å andra sidan kan det vara lätt att göra ett läsfel, speciellt om pekaren är numerisk.

För användaren erbjuder då kombinationstekniken en viss kontroll av att rätt funktion är vald. Den tvåstegsmanöver som måste utföras ger ofta anledning till den eftertanke som kan vara nödvändig för att undvika misstag vid t ex filuppdatering.

Valet i menyn görs alltså först med en numerisk eller alfanumerisk menypekare. Den efterföljande nedtryckningen av returtangenten resulterar inte i något programhopp utan placerar bara markören (eller annat lämpligt tecken) på motsvarande plats i menyn. Är valet korrekt utlöser nästa nedtryckning av returtangenten programhoppet. Indikerar markörläget att valet är felaktigt anges bara nytt läge varefter ny markörflyttning sker.

Ett exempel på en markörstödd meny återfinner du på raderna 1080--1380.

PRAKTISKT MENYEXEMPEL

Nedan följer programtexten till exemplet som illustrerar samtliga ovan beskrivna menytekniker. Genom att köra programmet får du en bra demonstration av hur de olika menyteknikerna upplevs av användaren. Genom att studera programtexten får du tips om hur motsvarande programavsnitt kan utformas.

```
100 PRINT CHR¤(12)
110 REM
120 REM ******************************"
130 REM
140 PRINT "***  MENYEXEMPEL  ***"
150 PRINT
160 REM ******************************
170 REM *PROGRAMMET AVSETT FÖR ABC 80*
180 REM *GÄLLER F O M 80-09-01       *
190 REM *KONSTRUKTÖR  A.BECEOTTI     *
200 REM *             DATAGATAN 80   *
210 REM *             STOCKALA       *
220 REM ******************************
230 REM
240 PRINT
250 PRINT "SPECIELLT   HANDHAVANDE"
260 PRINT "======================="
270 PRINT
280 PRINT "PROGRAMMET  EXEMPLIFIERAR NÄGRA"
290 PRINT "OLIKA TYPER AV MENYPEKARE."
300 PRINT "EFTERSOM PROGRAMMET ÄR UPPBYGGT"
310 PRINT "AV 5 LIKA MENYER MELLAN VILKA HOPP"
320 PRINT "KAN GÖRAS GODTYCKLIGT, SKER HOPP-"
330 PRINT "VALEN MED HJÄLP AV OLIKA TYPER AV"
340 PRINT "PEKARE I DE OLIKA MENYERNA." : PRINT
350 PRINT "(TRYCK PÅ RETURN NÄR DU VILL STARTA!)"
360 PRINT : PRINT : GET S¤
370 REM ================================
380 PRINT CHR¤(12)
390 ONERRORGOTO 380
400 REM
410 PRINT "*** MENYEXEMPEL-1 ***"
420 PRINT
430 PRINT "**********************"
440 PRINT "* (NUMERISK PEKARE) *"
450 PRINT "**********************"
460 PRINT
470 PRINT " 1.   NUMERISK      MENY"
480 PRINT " 2.   ALFABETISK    MENY"
490 PRINT " 3.   ALFANUMERISK  MENY"
500 PRINT " 4.   MARKÖRSTÖDD   MENY"
510 PRINT " 5.   MARKÖRSTYRD   MENY"
520 PRINT " 6.   SLUT"
530 PRINT
540 GOSUB 2030
550 INPUT S
560 IF S<1 OR S>6 THEN GOSUB 1950 : GOTO 380
570 ON S GOTO 380,590,860,1080,1400,1780
```

```
580 REM ================================
590 PRINT CHR¤(12)
600 ONERRORGOTO 590
610 REM
620 PRINT "***   MENYEXEMPEL-2   ***"
630 PRINT
640 PRINT "************************"
650 PRINT "*  (ALFABETISK PEKARE)  *"
660 PRINT "************************"
670 PRINT : PRINT
680 PRINT "N  =  NUMERISK      MENY"
690 PRINT "AB =  ALFABETISK    MENY"
700 PRINT "AN =  ALFANUMERISK  MENY"
710 PRINT "MÖ =  MARKÖRSTÖDD   MENY"
720 PRINT "MY =  MARKÖRSTYRD   MENY"
730 PRINT "S  =  SLUT"
740 PRINT
750 GOSUB 2030
760 GOSUB 2120
770 IF S¤="N" THEN GOTO 380
780 IF S¤="AB" THEN GOTO 590
790 IF S¤="AN" THEN GOTO 860
800 IF S¤="MÖ" THEN GOTO 1080
810 IF S¤="MY" THEN GOTO 1400
820 IF S¤="S" THEN GOTO 1780
830 GOSUB 1950 : GOTO 590
840 REM
850 REM ================================
860 PRINT CHR¤(12)
870 ONERRORGOTO 860
880 REM
890 PRINT "***   MENYEXEMPEL-3   ***"
900 PRINT
910 PRINT "************************"
920 PRINT "* (ALFANUMERISK PEKARE) *"
930 PRINT "************************"
940 PRINT : PRINT
950 PRINT "(10)   NUMERISK      MENY"
960 PRINT "(11)   ALFABETISK    MENY"
970 PRINT "(12)   ALFANUMERISK  MENY"
980 PRINT "(13)   MARKÖRSTÖDD   MENY"
990 PRINT "(14)   MARKÖRSTYRD   MENY"
1000 PRINT "(15)   SLUT"
1010 PRINT
1020 GOSUB 2030
1030 INPUT S¤
1040 S=VAL(S¤)
1050 IF S<10 OR S>15 THEN GOSUB 1950 : GOTO 860
1060 ON (S-9) GOTO 380,590,860,1080,1400,1780
```

```
1070 REM ================================
1080 PRINT CHR¤(12)
1090 ONERRORGOTO 1080
1100 REM
1110 PRINT "*** MENYEXEMPEL-4  ***"
1120 PRINT
1130 PRINT "**********************"
1140 PRINT "*   (MARKÖRSTÖDD)   *"
1150 PRINT "**********************"
1160 PRINT : PRINT
1170 PRINT "  A=NUMERISK       MENY"
1180 PRINT "  B=ALFABETISK     MENY"
1190 PRINT "  C=ALFANUMERISK   MENY"
1200 PRINT "  D=MARKÖRSTÖDD    MENY"
1210 PRINT "  E=MARKÖRSTYRD    MENY"
1220 PRINT "  F=SLUT"
1230 PRINT
1240 GOSUB 2030
1250 GOSUB 2120
1260 P=ASC(S¤)-64
1270 IF P<1 OR P>6 THEN GOSUB 1950 : GOTO 1080
1280 PRINT CUR(15,0);
1290 PRINT STRING¤(120,32)
1300 PRINT "KONTROLLERA ATT KVITTENSEN SOM"
1310 PRINT "BESTÅR AV MARKÖRENS PLACERING I"
1320 PRINT "MENYN, MOTSVARAR ÖNSKAT VAL!"
1330 PRINT "OM OK GE RETURN, ANNARS NYTT VÄRDE!"
1340 PRINT CUR((P+7),1);
1350 GET S¤
1360 IF S¤=CHR¤(13) THEN GOTO 1380
1370 GOSUB 2150 : GOTO 1260
1380 ON P GOTO 380,590,860,1080,1400,1780
```

```
1390 REM ================================
1400 PRINT CHR¤(12)
1410 ONERRORGOTO 1400
1420 REM
1430 PRINT "*** MENYEXEMPEL-5 ***"
1440 PRINT
1450 PRINT "**********************"
1460 PRINT "*   (MARKÖRSTYRD)    *"
1470 PRINT "**********************"
1480 PRINT
1490 PRINT
1500 PRINT "    NUMERISK      MENY"
1510 PRINT "    ALFABETISK    MENY"
1520 PRINT "    ALFANUMERISK  MENY"
1530 PRINT "    MARKÖRSTÖDD   MENY"
1540 PRINT "    MARKÖRSTYRD   MENY"
1550 PRINT "    SLUT"
1560 PRINT
1570 PRINT "SÖK ÖNSKAD MENY MED NÅGON AV"
1580 PRINT "TANGENTERNA FÖR FÖLJANDE TECKEN:"
1590 PRINT "  >  (FRAMÅTSTEGNING)"
1600 PRINT "  <  (BAKÅTSTEGNING)"
1610 PRINT "SAMT GE RETURN NÄR DU ÄR NÖJD!"
1620 REM
1630 P=8
1640 PRINT CUR(P,2);
1650 GET S¤
1660 IF S¤=">" THEN P=P+1
1670 IF P>13 THEN P=8
1680 IF S¤="<" THEN P=P-1
1690 IF P<8 THEN P=13
1700 IF S¤=CHR¤(13) THEN GOTO 1720
1710 GOTO 1640
1720 PRINT CHR¤(12)
1730 ON P-7 GOTO 380,590,860,1080,1400,1780
1740 REM
1750 REM ================================
1760 REM
1770 REM
```

```
1780 PRINT CHR¤(12)
1790 PRINT "*** PROGRAMAVSLUTNING ***"
1800 REM
1810 PRINT : PRINT
1820 PRINT "KÖRNINGEN KLAR!"
1830 PRINT "OMSTART (NEJ)";
1840 INPUTLINE S¤
1850 S¤=LEFT¤(S¤,1)
1860 IF S¤="J" OR S¤="j" THEN GOTO 100
1870 PRINT CHR¤(12)
1880 PRINT "KÖRNINGEN KLAR!"
1890 END
1900 REM ==============================
1910 REM .         SUBRUTINER
1920 REM ==============================
1930 REM
1940 REM
1950 REM --------- FELTEXT ------------
1960 REM
1970 PRINT "FELAKTIGT VÄRDE!";CHR¤(7)
1980 FOR N=1 TO 1500 : NEXT N : REM .VÄNTA !
1990 PRINT CHR¤(12)
2000 RETURN
2010 REM
2020 REM
2030 REM --------- LEDTEXT ------------
2040 REM
2050 PRINT "SÖK ÖNSKAD MENY MED HJÄLP AV"
2060 PRINT "ANGIVNA MENYPEKARE OMEDEL-"
2070 PRINT "BART FÖLJT AV RETURN."
2080 PRINT : PRINT "VILKEN MENY VÄLJER DU ";
2090 RETURN
2100 REM
2110 REM
2120 REM ---- BOKSTAVSKONVERTERING ----
2130 REM
2140 INPUT S¤
2150 S1¤=""
2160 FOR N=1 TO LEN(S¤)
2170 S=ASC(MID¤(S¤,N,1))
2180 IF S>95 AND S<127 THEN S=S-32
2190 S1¤=S1¤+CHR¤(S)
2200 NEXT N
2210 S¤=S1¤
2220 RETURN
2230 REM
2240 REM
2250 REM ------------------------------
```

FELHANTERING

OLIKA TYPER AV FEL

Som ett utmärkt hjälpmedel för programmeraren finns ett stort antal kontroller i BASIC-interpretatorn. Felaktiga instruktioner eller kommandon upptäcks redan när programmet matas in i datorn. Datorn accepterar då inte den inmatade programraden eller det givna kommandot utan skriver ut ett felmeddelande.

Andra fel uppstår först under programkörning exempelvis beroende på att utförda operationer ger ett resultat som datorn inte kan hantera. Dessa fel kan resultera i att programkörningen avbryts och en felkod presenteras på skärmen.

De feltyper som kan uppstå under programkörning i ABC 80 är huvudsakligen av två slag:

- Fel som i programmet kan hanteras med instruktionen ONERRORGOTO ...
- Fel som alltid leder till avbrott i programkörningen

När det gäller fel av den första typen kan oönskade avbrott i programkörningen i de flesta fall undvikas. I stället kan då en informativ hjälptext presenteras som leder användaren rätt utan att programkörningen avbryts.

Fel av den andra typen undviks av den förutseende programmeraren redan vid programmets konstruktion.

INTRESSANTA FELKODER

De fel som kan uppstå under programkörning och som i programmet kan hanteras med instruktionen ONERRORGOTO ... är 16 stycken.

* 4 FÖR STORT FLYTTAL.

* 7 FÖR STORT HELTAL. Försök att använda ett tal som ligger utanför datorns kapacitetsområde för heltal.

* 12 FELAKTIGT TAL. Talet innehåller tecken som inte är siffror.

* 19 KAN EJ ÖPPNA FLER FILER. För många filer redan öppnade.

* 21 HITTAR EJ FILEN. Datorn kan inte hitta filen. Filen finns inte tillgänglig eller har sökts under fel namn.

* 30 DATA SLUT. Datalistan har blivit tömd och en READ-sats efterfrågade fler data.

* 34 SLUT PÅ FILEN. Försök att läsa efter filens slut.

* 35 CHECKSUMMAFEL VID LÄSNING. Flexskivan eller kassetten skadad.

* 36 CHECKSUMMAFEL VID SKRIVNING. Flexskivan skadad.

* 37 FELAKTIGT RECORDFORMAT. Icke kompatibelt inspelningsformat. Kan också betyda fel på flexskiva eller kassett.

* 38 RECORDNUMMER UTANFÖR FILEN. Försök att läsa längre än filen medger.

* 39 FILEN SKRIVSKYDDAD. Filen är skyddad för utskrift.

* 40 FILEN RADERINGSSKYDDAD. Filen är skyddad mot borttagning.

* 41 FLEXSKIVAN FULL. Filen får ej plats på flexskivan.

* 42 FLEXSKIVAN EJ KLAR. Ingen flexskiva inmatad i flexskivminnet eller spaltluckan inte stängd.

* 43 FLEXSKIVAN SKRIVSKYDDAD.

FELAKTIG INMATNING

Vid inmatning till en flyttalsvariabel uppstår ett fel om inmatat värde inte är rent numeriskt. Detta fel uppstår om du vid körningen av exemplet nedan svarar t ex TVÅ.

```
150 PRINT "MATA IN ETT TAL";
160 INPUT T
170 GOTO 150
```

En felaktig inmatning resulterar som du ser både i en utskrift av aktuell felkod (12) och i ett avbrott i programkörningen.

Vid körningen av nästa program ignorerar datorn en felaktig inmatning. En ledtext hjälper användaren att göra inmatningen på rätt sätt och därigenom undviks avbrott i programkörningen.

```
140 ONERRORGOTO 180
150 PRINT "MATA IN ETT TAL";
160 INPUT T
170 GOTO 150
180 PRINT "MATA IN TALET MED SIFFROR!"
190 PRINT
200 GOTO 150
```

HUR FUNGERAR ONERRORGOTO ...?

I programmeringsspråket BASIC utförs normalt instruktionerna i den ordning som radnumren anger. Det förhållandet gäller inte instruktionen ONERRORGOTO ... Denna instruktion förbereder endast datorn för att senare eventuellt utföra ett hopp i programmet till angiven programrad. Det första fel som sedan uppträder under programkörningen utlöser det förberedda hoppet. Om mer än ett fel ska tas om hand av datorn måste instruktionen ONERRORGOTO ... utföras på nytt före varje fel.

Hur bör därför föregående programexempel ändras, för att upprepade fel vid inmatningen ska omhändertas av programmet? Lämpligen ändras den sista programraden till

```
200 GOTO 140
```

Hur går man tillväga om flera inmatningar förekommer i ett program? Studera följande programexempel.

```
100 ONERRORGOTO 100
101 PRINT
110 PRINT "MATA IN FÖRSTA TALET";
120 INPUT T1
130 ONERRORGOTO 130
131 PRINT
140 PRINT "MATA IN ANDRA TALET";
150 INPUT T2
160 GOTO 100
```

Den senare ONERRORGOTO ...-satsen förbereder datorn för ett nytt hopp till en annan programrad. Det här gäller även om den första ONERRORGOTO ...-satsen inte resulterade i något hopp på grund av felaktig inmatning. Det är alltid den senast utförda ONERRORGOTO ...-instruktionen som gäller när ett fel inträffar.

I dessa programexempel har enbart felet 12, FELAKTIGT TAL, varit aktuellt. Givetvis kan alla de 16 tidigare redovisade feltyperna hanteras på motsvarande sätt.

ONERRORGOTO ... I SUBRUTINER

När subrutiner används måste återhoppet till huvudprogrammet ske med instruktionen RETURN. Det är inte tillåtet att använda en GOTO-sats för återhoppet. Ett program med detta utseende är därför givetvis felaktigt.

```
100 REM .......
110 GOSUB 150
120 REM
130 GOTO 100
140 END
150 REM ******** SUBRUTIN ********
160 N=N+1 : PRINT N
170 GOTO 120
180 RETURN
190 REM **************************
```

När subrutinanropet på rad 110 utförs sparas i datorns minne information om att programkörningen ska fortsätta på rad 120 när subrutinen utförts. Vid återhopp med instruktionen RETURN "förbrukas" den lagrade informationen om radnumret 120. När återhopp från subrutinen sker felaktigt med en GOTO-sats "förbrukas" inte informationen om radnumret 120 utan minnet fylls slutligen helt med denna information.

Av samma anledning kan fel inte hanteras med instruktionen ONERRORGOTO ... på detta sätt.

```
100 ONERRORGOTO 100
110 GOSUB 150
120 REM
130 GOTO 100
140 END
150 REM ******** SUBRUTIN ********
160 N=N+1 : ; N
170 S%=1.23456E+6 : REM FEL 7 för stort heltal
180 RETURN
190 REM **************************
```

Det fel som uppstår i körningen – fel 7, FÖR STORT HELTAL – uppträder i subrutinen på rad 170. Satsen ONERRORGOTO 100 har förberett datorn för ett hopp till rad 100. Återhoppet från subrutinen sker alltså inte med instruktionen RETURN och därmed kan vi konstatera att programmet är felkonstruerat. Vid körning fylls minnet och programkörningen avbryts. En allvarligare konsekvens av att subrutiner lämnas på ett felaktigt sätt är att oordning skapas i återhoppen från olika subrutiner.

Ett sätt att här åstadkomma en korrekt felhantering är att placera instruktionen ONERRORGOTO ... i subrutinen. Dessutom måste man se till att återhoppet från subrutinen sker med instruktionen RETURN. Detta exempel visar ett program med fungerande felhantering.

```
100 GOSUB 140
110 REM
120 GOTO 100
130 END
140 REM ******** SUBRUTIN ********
150 ONERRORGOTO 190
160 N=N+1 : ; N; : REM varvräknare presenteras
170 S%=1.23456E+6 : REM FEL 7 för stort heltal
180 GOTO 200
190 PRINT " FÖR STORT HELTAL"
200 RETURN
210 REM ***************************
```

ONERRORGOTO 0

Vi har i föregående avsnitt sett ett exempel på oförsiktigt användande av instruktionen ONERRORGOTO ... En generell metod att undvika problem med felaktiga hopp från subrutiner är att använda satsen ONERRORGOTO 0 före varje subrutinanrop samt före varje RETURN i subrutiner. ONERRORGOTO 0 innebär att inget hopp i programmet

utförs om ett fel uppstår, d v s verkan av en tidigare ONERRORGOTO ...-sats upphävs. Nedanstående programexempel visar principen.

```
100 ONERRORGOTO 100 : REM felhantering TILL
110 PRINT "MATA IN ETT TAL"
120 INPUT S% : REM ev fel 12 felaktigt tal
130 ONERRORGOTO 0 : REM . felhantering FRÅN
140 GOSUB 160
150 GOTO 100
160 REM ********** SUBRUTIN **********
170 ONERRORGOTO 200 : REM felhantering TILL
180 S%=S : REM ev fel 7 för stort heltal
190 GOTO 210
200 PRINT "FÖR STORT TAL!"
210 ONERRORGOTO 0 : REM . felhantering FRÅN
220 RETURN
230 REM ******************************
```

Undantag från den angivna generella regeln kan göras om anropad subrutin är sådan att inget av de tidigare nämnda felen kan inträffa. Satsen ONERRORGOTO 0 kan då utelämnas.

ERRCODE – DATORNS INTERNA FELKODER

En möjlighet att ta reda på vilken feltyp som har utlöst ett hopp med instruktionen ONERRORGOTO ... är att använda funktionen ERRCODE. Värdet av funktionen ERRCODE är den numeriska felkoden för det senast inträffade felet (jämför programmeringskortet).

Om flera olika feltyper kan tänkas uppstå i ett programavsnitt under körning kan värdet av funktionen ERRCODE användas för att välja ut en ledtext som är lämplig för den aktuella feltypen. Följande förenklade programexempel illustrerar principen.

```
100 GOTO 130
110 IF ERRCODE=7 THEN ; : PRINT "för stort tal"
120 IF ERRCODE=12 THEN ; : PRINT "obs siffror"
130 ONERRORGOTO 110
140 PRINT "MATA IN ETT TAL"
150 INPUT S : REM ev fel 12 felaktigt tal
160 S%=S : REM ev fel 7 för stort heltal
170 GOTO 130
```

FEL UTANFÖR PROGRAMMETS KONTROLL

Som nämndes inledningsvis finns i ABC 80 två huvudtyper av fel som kan uppstå under programkörning. Dels de som kan hanteras med instruktionen ONERRORGOTO ... och dels de som alltid förorsakar programavbrott. Den första typen har redan studerats i flera exempel och kommer att ingå som en naturlig del i de kommande kapitlens programexempel. Den feltyp som inte kan hanteras med instruktionen ONERRORGOTO ... bör inte förekomma i väl genomtänkta program. Låt oss illustrera hur man i några fall kan förekomma uppkomsten av detta fel.

Vårt första exempel ska visa hur man undviker felet 10, TEXTEN FÅR EJ PLATS I STRÄNGEN. De data som matas in ska överföras till ett strängfält X¤ (1) t o m X¤ (4). För att spara utrymme i datorns minne dimensioneras oftast indexerade strängvariabler så, att utrymmet för varje sträng begränsas till ett visst antal tecken, här 20 stycken.

Svarssträngen S¤ bör dimensioneras så att användaren felaktigt kan mata in en sträng med för många tecken, utan att avbrott i programkörningen sker. Felet 10 undviks i exemplet genom att en rimlighetskontroll genomförs. Inmatade strängars längd kontrolleras och de strängar som är för långa kortas av.

```
100 DIM X¤(4)=20
110 DIM S¤=130
120 REM inmatning av strängfältet
130 FOR N=1 TO 4
140 PRINT "MATA IN STRÄNG NUMMER ";N
150 INPUT S¤
160 IF LEN(S¤)>20 THEN S¤=LEFT¤(S¤,20)
170 X¤(N)=S¤
180 NEXT N
190 REM utskrift av strängfältet
200 PRINT
210 FOR N=1 TO 4
220 PRINT "STRÄNG NUMMER ";N;" ";X¤(N)
230 NEXT N
```

Om ett program vid körning förutsätter att systemet innehåller en printer, kan felet 8, FINNS EJ I DETTA SYSTEM, uppträda. Det kan då bero på att printern inte anslutits eller att printerns nätströmbrytare inte slagits till.

Programmet bör då inledas med en fråga till användaren om huruvida printern är ansluten eller ej. Blir då användarens svar JA fortsätter körningen med att öppna PR: som en fil. Denna procedur utgör endast en påminnelse till användaren och är naturligtvis inget absolut skydd mot avbrott i programkörningen. Det är dock en fördel att ett eventuellt avbrott inträffar i början av programkörningen.

```
100 PRINT CHR¤(12)
110 PRINT
120 PRINT "DETTA PROGRAM FÖRUTSÄTTER PRINTER !"
130 PRINT
140 PRINT "ÄR PRINTERN ANSLUTEN/PÅSLAGEN (JA)";
150 INPUT S¤
160 IF LEN(S¤)=0 THEN 190
170 S¤=LEFT¤(S¤,1)
180 IF S¤="N" OR S¤="n" THEN 110
190 OPEN "PR:" ASFILE 1
200 REM .......
210 REM .......
220 REM .......
230 END
```

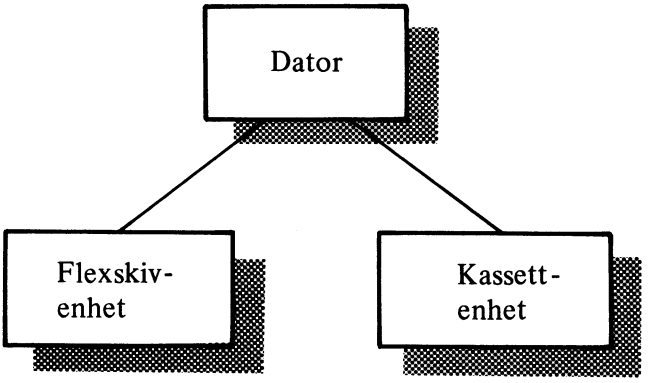
De flesta orsaker till avbrott i programkörningen kan visserligen undanröjas med en väl genomförd felhantering, men programmeringsarbetet kan ändå inte anses slutfört förrän programmet testats noggrant. Använd då din fantasi för att försöka sätta dig in i användarens situation.

6

PROGRAMLAGRING PÅ EXTERNMINNE

FLERA MASSMINNEN I DATORSYSTEMET

Även till små datorsystem finns nu flera externa minnesenheter att tillgå med möjlighet att lagra både program och data. Vanligast i små system är kassettbandspelare och flexskivenhet.



Genom att ansluta ett extra massminne av flexskivtyp till ett datorsystem som tidigare enbart haft kassettminne vinner man följande fördelar:

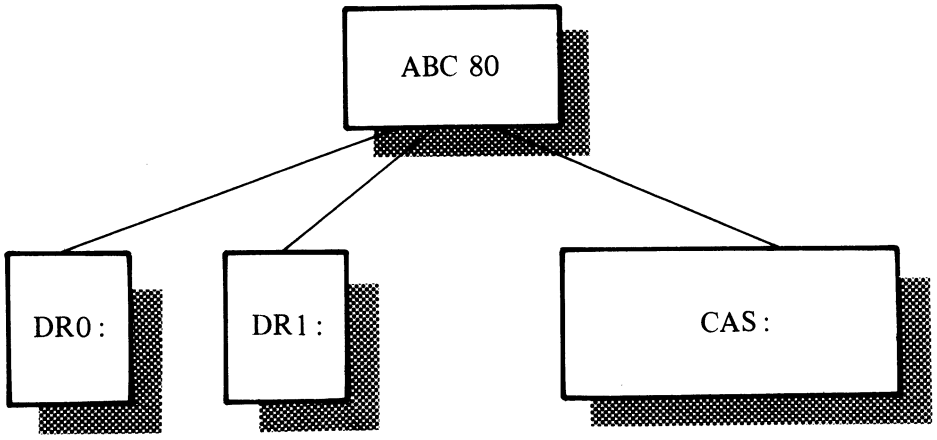
- enklare användning
- ökad snabbhet
- ökad tillförlitlighet

I detta avsnitt beskrivs hur datorprogram kan överföras mellan ABC 80 och de båda anslutna externminnena. När flera minnesenheter är anslutna måste givetvis instruktioner och kommandon utformas så att datorn kan avgöra vilken enhet som instruktionen eller kommandot avser, d v s vilken enhet som "adresseras".

Flexskivenheten innehåller normalt två separata minnesenheter med plats för var sin flexskiva. Flexskivminnets enheter betecknas i bruksanvisningen med DR0: (eng. drive 0) respektive DR1: (eng. drive 1). Datorn kan således adressera tre olika massminnen, flexskivans bägge drivenheter samt kassettbandspelaren.

För att kunna skilja på de olika minnesenheterna inför man i instruktioner och kommandon beteckningarna DR0:, DR1: respektive CAS: Om beteckningen utelämnas adresseras automatiskt flexskivminnets DR0:

ABC 80 prioriterar alltså enheterna så som figuren visar.



Prioritet 1

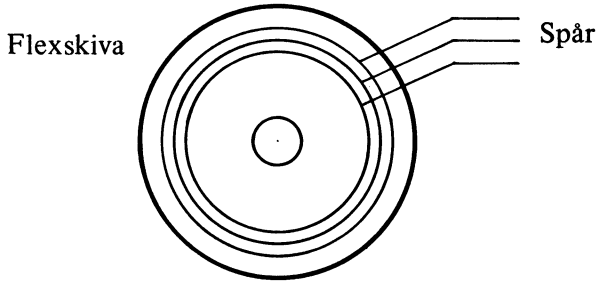
Prioritet 2

Enhetstyp (DR1:) behöver ibland anges i instruktioner och kommandon

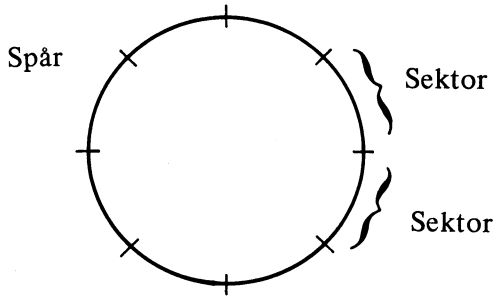
Enhetstyp (CAS:) måste alltid anges

FLEXSKIVANS INDELNING

När ett program ska överföras från datorns interna minne till flexskivan sker lagringen av informationen på ett eller flera spår på flexskivan med en teknik som liknar inspelning på band. Varje flexskiva är normalt indelad i 40 spår.



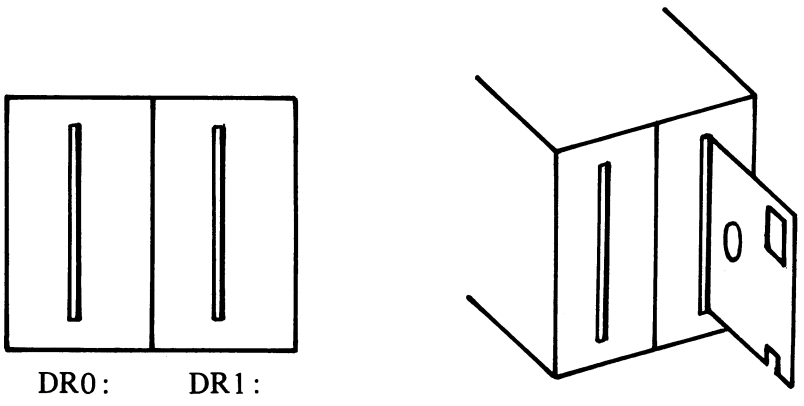
Om programmet inte ryms på ett spår, tar datorn flera spår i anspråk till dess hela programmet är lagrat. Vid in- och avspelning roterar flexskivan. Med hjälp av sinnrik elektronik och precisionsmekanik placeras läs/skriv-huvudet på rätt spår. Varje spår är dessutom indelat i ett antal sektorer, normalt 8 stycken, som bilden visar.



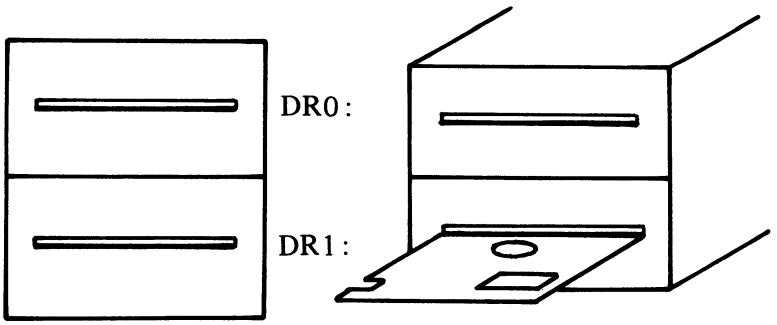
Datorn måste förutom rätt spår även hitta den rätta sektorn på varje spår. För att detta ska vara möjligt måste viss information finnas inspelad på flexskivan innan man kan börja lagra program. Denna preparering av en oanvänd flexskiva kan användaren själv utföra med hjälp av speciell programvara som medföljer flexskivenheten.

FLEXSKIVAN FÖRBEREDS FÖR ANVÄNDNING

Vi förutsätter att datorsystemet är uppkopplat enligt bruksanvisningen. Till flexskivenheten finns en systemflexskiva som innehåller den nödvändiga programvaran. Den systemprogramvara som behövs redovisas i programmeringskortet. Placera systemflexskivan i drivenhet 0 (DR0:) och en oanvänd flexskiva i drivenhet 1 (DR1:) Observera att flexskivorna måste vändas rätt. I figuren ges exempel på vad som avses med DR0: respektive DR1: samt på hur flexskivorna placeras.



Stående flexskivor



Liggande flexskivor

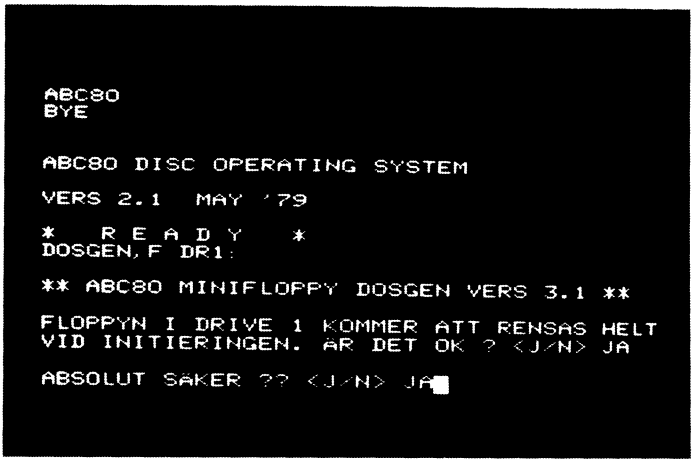
Förutsättningen för att systemprogramvaran ska kunna användas är att du ger kommandot

```
BYE
```

Därefter kör du det speciella programmet som preparerar skivan i drivenhet 1 med kommandot

```
DOSGEN,F DR1:
```

Observera att mellanslaget mellan F och DR1 : måste finnas med. Efter det att du svarat på några kontrollfrågor startar preparering och testning av flexskivan.



Efter någon minut ger datorn besked om att flexskivan är klar för användning.

FLEXSKIVANS PROGRAMBIBLIOTEK

Flexskivan är nu klar att använda för programlagring. En flexskiva rymmer normalt flera program. För att kunna hålla reda på vilka program som finns och var de är lagrade, upprättar datorn ett programbibliotek på några av flexskivans yttersta spår.

Med ett speciellt BASIC-program som vanligen kallas LIB (eng. library, d v s bibliotek) kan datorn läsa hela biblioteket och presentera detta på bildskärm eller skrivare.

Det är därför lämpligt att överföra ett sådant LIB-program från systemflexskivan i DR0: till den oanvända flexskivan i DR1: Detta kan göras på olika sätt, t e x med kommandot

```
COPY DR0:LIB.BAC,DR1:
```

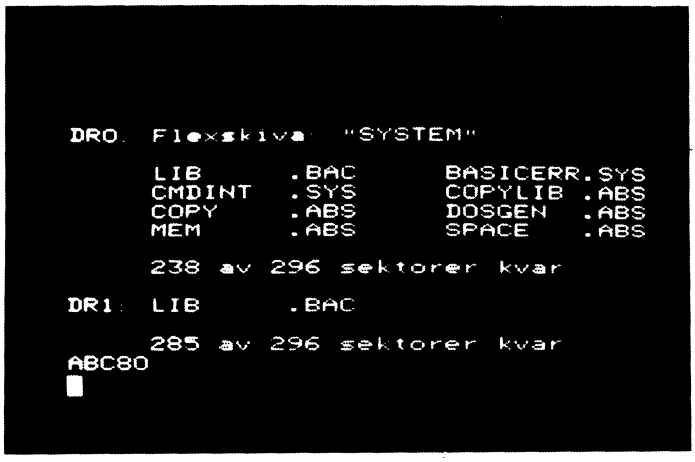
Efter några sekunder är proceduren avslutad. Nu återstår bara att avsluta användningen av systemprogramvaran med kommandot

```
¤BAS
```

Använd LIB-programmet för att kontrollera innehållet på de båda flexskivorna i DR0: och DR1: Eftersom LIB-programmet är ett vanligt BASIC-program kan du på vanligt sätt köra programmet med kommandot

```
RUN LIB
```

Resultatet av en sådan körning visar detta foto.



För att underlätta ditt fortsatta arbete är det lämpligt att du framställer ytterligare en flexskiva, preparerad och försedd med ett LIB-program.

LAGRA PROGRAM PÅ FLEXSKIVA

Nu är det dags att lagra egna program på de två iordningställda flexskivorna. Skriv in ett program i datorn, t ex

```
100 PRINT CHR¤(12)
110 REM ********************************
120 REM
130 PRINT "*** PROGEX1 ***"
140 REM
150 REM ********************************
160 REM * PROGRAMMET AVSETT FÖR ABC-80 *
170 REM * GÄLLER F O M  80-09-01       *
180 REM * KONSTRUKTÖR   A.BECOTTI      *
190 REM * ADRESS        DATAGATAN 80   *
200 REM *               STOCKALA       *
210 REM ********************************
```

Lagra programmet på ena flexskivan med kommandot

```
SAVE DR0:PROGEX1
```

Med en indikator på flexskivenhetens framsida indikeras vilken minnesenhet som för tillfället körs. Observera att indikatorerna visar att programmet lagras på flexskivan i DR0:

Samma funktion erhålls med kommandot

```
SAVE PROGEX1
```

Detta beror på att datorn i första hand adresserar flexskivenhetens DR0: om kommandot saknar tillägg för enhet.

Ändra nu programmet genom att byta ut en rad i föregående program.

```
130 PRINT "*** PROGEX2 ***"
```

Detta program kan nu lagras på flexskivan i DR1: med kommandot

```
SAVE DR1:PROGEX2
```

Givetvis kan programmet också lagras på kassettband genom att man ger kommandot

```
SAVE CAS:PROGEX3
```

De program som hittills lagrats på minnesenheterna är alltså:

DR0: PROGEX1
DR1: PROGEX2
CAS: PROGEX3

Som du vet kan flexskivornas innehåll presenteras på bildskärm eller printer genom att man kör programmet LIB.

Ändra nu återigen i programmet på rad 130 och lagra det nya programmet med namnet PROGEX4 på flexskivan i DR0: Kommandot blir t ex

```
SAVE PROGEX4  (eller  SAVE DR0:PROGEX4)
```

Datorsystemet söker nu automatiskt upp ledig plats på flexskivan. Det tidigare lagrade programmet PROGEX1 på samma skiva påverkas givetvis inte av detta. Skulle däremot ett nytt program lagras med samma namn som ett tidigare lagrat program så kommer det gamla programmet att raderas och ersättas av det nya programmet.

Kommandot LIST ... kan liksom kommandot SAVE ... användas för att lagra program på flexskivor och kassetter. När kommandot LIST ... används lagras programtexten tecken för tecken (ASCII-form). Jämför presentationen på bildskärmen när kommandot LIST används. Med kommandot SAVE ... lagras däremot programmet i delvis kompilerad form.

HÄMTA PROGRAM FRÅN FLEXSKIVA

Töm datorns interna minne med kommmandot

```
NEW
```

Hämta programmet PROGEX1 med kommandot

```
LOAD DR0:PROGEX1
```

Enhetstypen (DR0:) kan även här utelämnas. Att programmet verkligen överförts till datorn kan kontrolleras med kommandot LIST.

Vad händer om man ger kommandot

```
LOAD PROGEX2
```

Programmet PROGEX2 är lagrat på DR1: Vid provkörning visar flexskivenhetens indikatorer att datorn först adresserar DR0: och därefter DR1: När program hämtas från flexskiva behöver man således inte ange om programmet finns på flexskivan i DR0: eller DR1: Datorn söker om det behövs på bägge flexskivorna.

Ska program däremot hämtas från kassettminnet måste detta alltid anges med tillägget CAS:

```
LOAD CAS:PROGEX3
```

Detta kommando medför att datorn söker på kassettbandet till dess programmet PROGEX3 påträffas. Utelämnas programnamn efter kommandot LOAD CAS: hämtar datorn det program som först påträffas på bandet. ABC 80 skriver i detta fall det påträffade programmets namn på bildskärmen.

RADERA PROGRAM PÅ FLEXSKIVA

När program ska avlägsnas från en flexskiva ger man ett kommando till datorn i vilket man anger det program som ska raderas. Att avlägsna programmet PROGEX2 kan utföras med något av dessa alternativ:

```
UNSAVE DR1:PROGEX2          (eller UNSAVE PROGEX2)

KILL "DR1:PROGEX2.BAC"      (eller KILL "PROGEX2.BAC")
```

För att kontrollera att programmet inte längre finns med i flexskivans programbibliotek kan nu programmet LIB köras på nytt.

DOKUMENTERA FLEXSKIVANS INNEHÅLL

En utskrift av programbiblioteket är en utmärkt innehållsförteckning över en flexskiva. En sådan utskrift åstadkommes lätt med programmet LIB och en printer. Om det första program som lagras på flexskivan förses med filtypen LBL (eng. label) kommer benämningsdelen för detta programnamn att skrivas som rubrik i förteckningen.

För att erhålla utskriften som figuren visar måste den första lagringen på flexskivan ske med programnamnet SYSTEM.LBL.

UTÖKA DATORNS KAPACITET

Storleken av det program som ABC 80 kan bearbeta begränsas inte enbart av datorns internminneskapacitet. Det anslutna externminnets lagringsförmåga och snabbhet blir avgörande för hur stora program som kan köras. Kravet på externminnet är att programavsnitten ska kunna hämtas i godtycklig ordning och överföras på kort tid. Dessa förutsättningar uppfylls av flexskivenheten men inte av kassettenheten.

Den metodik som man använder när omfattande program konstrueras ska här kort beskrivas.

Dela in programmet i huvudprogram och underprogram. Observera att alla olika delprogram måste fungera oberoende av varandra. Varje delprogram avslutas med instruktionen CHAIN "...", som medför att efterföljande programavsnitt överförs från externminnet till ABC 80:s internminne.

Eftersom alla variabler nollställs när instruktionen CHAIN "..." utförs, måste alla variabelvärden som är gemensamma för skilda delprogram normalt lagras på externminnet. Dessa data lagras i datafiler och hur detta går till beskrivs utförligt i följande kapitel.

KOPIERING AV FLEXSKIVOR

Den information som finns lagrad på flexskivor i form av program och/eller data representerar ofta ett stort värde för användaren. Tyvärr är inga lagringsmedia absolut tillförlitliga. Vid läsning och skrivning utsätts flexskivan för mekaniskt slitage, vilket resulterar i begränsad livslängd för flexskivan. Även ett oväntat strömavbrott som inträffar under körning kan förorsaka att den information som finns lagrad på flexskivan förstörs. Dessutom kan en flexskiva lätt skadas genom ovarsam hantering. Det är således mycket väsentligt att för säkerhets skull skaffa sig kopior av flexskivornas innehåll.

Med hjälp av speciell programvara kan flexskivor lätt kopieras. Det finns olika utföranden på programvarorna, men bland den systemprogramvara som levereras med flexskivenheten finns alltid ett program som kan användas för kopiering. Vi ger här ett exempel på hur sådan programvara används.

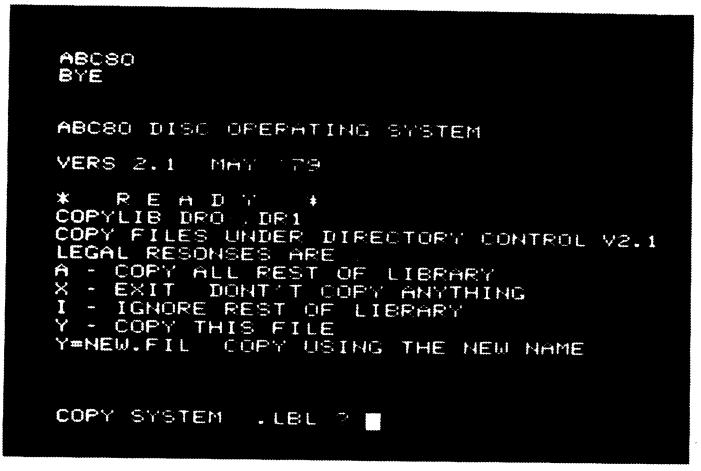
En systemflexskiva som innehåller CMDINT.SYS och COPYLIB. ABS placeras i DR0: och en preparerad flexskiva i DR1: Du kan nu kopiera valda delar av systemflexskivans innehåll till flexskivan i DR1:

De program som används vid kopieringen är inte vanliga BASIC-program. För att kunna få tillgång till den speciella programvara som behövs för kopieringen ger du därför kommandot

```
BYE
```

När texten * READY * presenteras på bildskärmen kan kopieringen påbörjas med kommandot

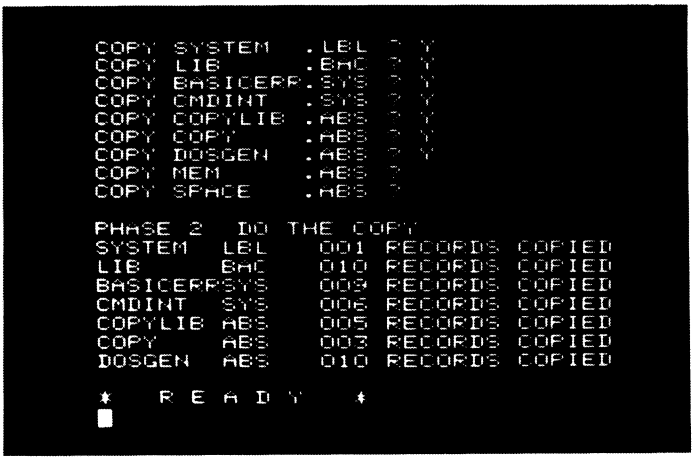
```
COPYLIB DR0:,DR1:
```



Det är nu möjligt att kopiera önskade filer från flexskivan i DR0: till flexskivan i DR1: På bildskärmen presenteras efterhand alla filer som finns på flexskivan i DR0: Genom att svara enligt anvisningarna på bildskärmen väljer du vilka filer som ska kopieras.

Exempelvis betyder svaret Y (YES) följt av en tryckning på returtangenten att presenterad fil ska kopieras. Enbart en tryckning på returtangenten innebär att aktuell fil inte kopieras. När alla filer presenterats startar kopieringen av önskade filer.

Efter slutförd kopiering kan bildskärmen se ut på följande sätt



För att avsluta användningen av systemprogramvaran och återgå till BASIC-interpretatorn ges kommandot

```
¤BAS
```

Den flexskiva som finns i DR1 : innehåller nu de mest användbara systemprogrammen. Det är praktiskt att redan vid prepareringen förse alla flexskivor med denna systemprogramvara. Alla flexskivor kan då användas för preparering och kopiering av flexskivor.

7

INTRODUKTION TILL FILHANTERING

VAD ÄR EN FIL?

Vårt svenska ord FIL kommer från det franska "file". Ordet "file" har i franska språket två huvudbetydelser. Dels betyder det snöre, sträng, snodd och dels en räcka eller följd av saker eller föremål med likartade egenskaper. I svenska språket kan man ibland träffa på uttrycken "rummen låg i fil" eller "en fil av rum".

Vanligare idag är kanske det trafiktekniska ordet fil, som betyder platsen eller körfältet där en rad fordon åker. Ursprungsbetydelsen gäller som synes även nu, d v s "EN FÖLJD AV LIKARTADE TING"

På kontor kan man ibland få höra ett mycket slarvigt bruk av ordet fil, t ex "jag ska ta och fajla in det här dokumentet" Med detta uttryck avser vederbörande att placera en handling bland en mängd andra i någon form av dokumentarkiv eller dokumentregister, oftast efter bestämda insorteringsregler. Den här betydelsen kommer från den engelskspråkiga varianten av ordet, nämligen "file".

Fil kan således också betyda en samling dokument, eller en följd av dokument, av likartad karaktär. I överförd bemärkelse betyder fil även platsen, d v s skåpet, mappen, kortlådan e t c, där dessa likartade dokument förvaras.

ETT KORTREGISTER ÄR EN FIL

Ett kortregister är ett utmärkt och klassiskt exempel på en fil. Registret kan utgöra en förteckning över alla dina grammofonskivor och deras innehåll, eller vara en lista över vilka bilar som en bilfirma har i sitt lager. Det kan vara en förteckning hos en firma över kundfakturor som förfallit till betalning. Det kan också vara en medlemsförteckning, en receptsamling eller en enkel förteckning över telefonnummer till vänner och bekanta.

Låt oss nu titta litet mer i detalj på vad ett kortregister egentligen är och hur man arbetar med det. Som exempel väljer vi ett litet register med telefonnummer.

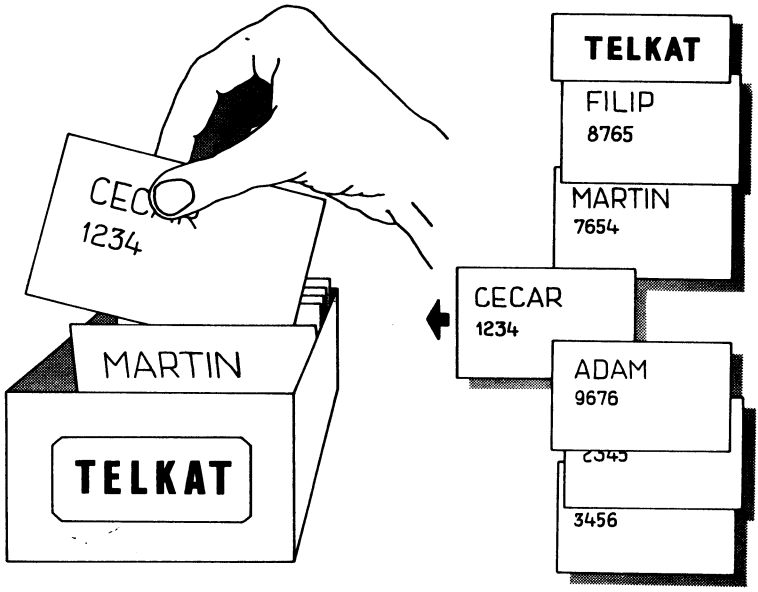
I de flesta fall används den första termen som sorteringsbegrepp (sökterm) för att posten ska vara lätt att hitta i filen/registret.

Allt eftersom du skriver korten färdiga, stoppar du in dem på rätt plats i registret. Om du är litet förutseende när korten skrivs skapar du dem redan från början i en sådan följd att du inte behöver söka efter kortens rätta platser i registret. Du behöver då bara lägga in de färdiga korten efter varandra i tur och ordning.

När du är färdig med nyuppläggningen är det bara ett STÄNGA (eng. close) registerlådan och ställa undan den.

ATT SÖKA I EN FIL

Den viktigaste användningen av din registerlåda är här informationssökning. För att kunna läsa informationen måste du först ÖPPNA (eng. open) kortlådan.



Du vet vems telefonnummer du söker och håller det namnet i minnet. Jämför sedan detta namn med söktermen/namnet som står överst på registerkorten. Du läser på så sätt registerkorten, ett i taget, i den följd de ligger i lådan till dess att du antingen träffar på ett kort med samma namn som du har i minnet, eller träffar på ett kort med "större namn" i namntermen.

I det första fallet har du fått "träff" och kan läsa vidare på det kortet för att få tag på det sökta telefonnumret. I det andra fallet saknas post/kort med det sökta namnet. Det sista gäller under förutsättning att korten ligger sorterade efter namntermernas "storleksordning".

När du är färdig med letandet STÄNGER (eng. close) du som vanligt kortlådan!

UPPDATERA EN FIL

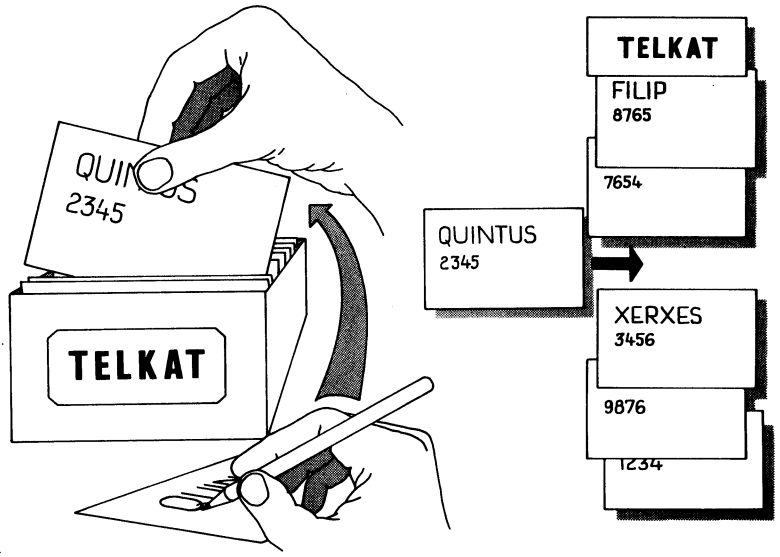
Efter en tid upptäcker du att innehållet i ditt register måste förändras. Du har fått nya bekanta som skall in i registret. Andra har flyttat och bytt adress och kanske även telefonnummer. Eventuellt har några personer försvunnit ur din bekantskapskrets. Du behöver UPPDATERA ditt register.

Vid uppdatering av en fil (register) är således tre olika aktiviter aktuella, nämligen att

- poster tillkommer
- poster ändras
- poster utgår

POSTER TILLKOMMER

När en post ska tillkomma innebär detta följande moment. Först öppnar du registret så att det står färdigt att ta emot de tillkommande posterna (registerkorten).



Om inte korten är sorterade behöver du bara lägga in de nya korten längst fram i registret och sedan stänga lådan. Dock innebär detta två väsentliga nackdelar för dig.

- Du måste i sämsta fall söka igenom hela registret för att vara säker på att finna det önskade telefonnumret.
- När nya poster tillkommer måste du gå igenom hela registret för att förvissa dig om att det inte finns något gammalt kort med den tillkommande personen.

Allt talar således för att kortregistret skall vara sorterat, i detta fallet i namnordning.

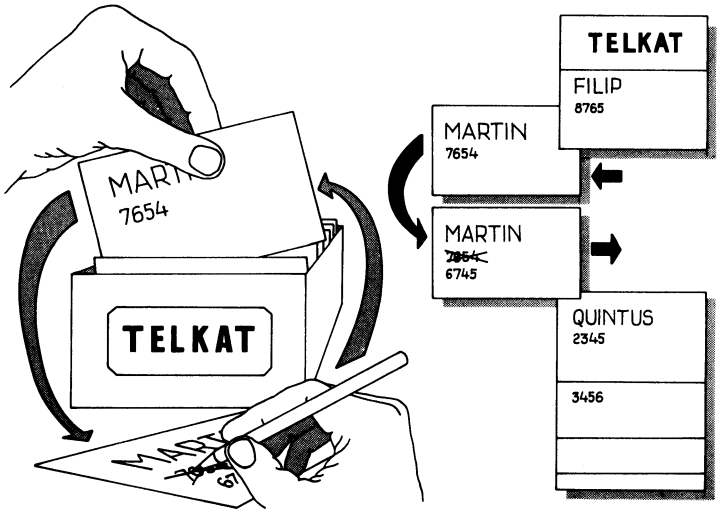
Genom att läsa varje kort i registret i tur och ordning kan du hitta den rätta platsen för den tillkommande posten. Om där redan finns ett kort för den aktuella personen d v s "POSTEN FINNS REDAN, KAN EJ TILLKOMMA" måste du avgöra

- om registerkortets innehåll skall vara oförändrat
- om det gamla registerkortet skall bytas ut mot det nya
- om det gamla registerkortet skall finnas kvar, men med förändrade uppgifter

Om du nu har flera tillkommande registerkort så är det lämpligt att även dessa är sorterade i samma ordningsföljd som själva kortregistret. Vid uppdateringen sparar du då tid eftersom du endast behöver gå igenom kortregistret en enda gång.

POSTER ÄNDRAS

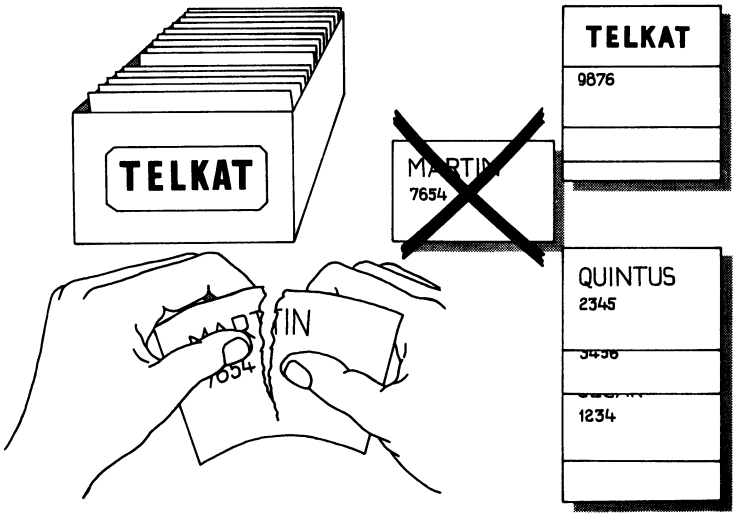
När en post ändras söker du efter ett kort som gäller den person vars namn du för ögonblicket håller i minnet. Kort för kort jämför du namnet du läser med det namn (söknyckel) som du håller i minnet. Om du får träff, d v s kortet finns, behöver du bara plocka fram det och göra erforderliga ändringar.



Därefter skall kortet återföras till sin rätta plats i lådan. Finns däremot ej det sökta kortet i lådan kan vi bara konstatera att "POSTEN SAKNAS, KAN EJ ÄNDRAS". Gå därefter vidare och sök efter nästa post som skall ändras till dess att alla ändringar är utförda och registret kan stängas.

POSTER UTGÅR

När poster ska utgå och poster ska ändras går sökningen till på samma sätt.



När du hittat rätt post (kort) i filen lyfter du bort kortet och fortsätter uppdateringen. Får du ingen träff blir den naturliga kommentaren "POSTEN SAKNAS, KAN EJ UTGÅ!

Den här beskrivna manuella metoden att hantera ett kortregister är analog med datorns sätt att arbeta med en fil. I nästa kapitel ska vi gå närmare in på filhantering i en dator.

8

SEKVENTIELLA FILER PÅ FLEXSKIVA

DATAFILER OCH PROGRAMFILER

Vi har tidigare sett hur man kan skriva och läsa program på flexskiva med speciella kommandon, t ex SAVE ... , LOAD ... och CHAIN ... Program lagrade på flexskiva kallas ibland programfiler, eftersom ett program utgörs av en följd av programrader.

Av samma anledning som man vill lagra programfiler permanent vill man ofta lagra data från en programkörning till nästa körning. Sådana data lagrade på flexskiva eller kassett kallas datafiler. Med filer avser vi fortsättningsvis sådana datafiler.

Skrivning eller läsning av filer sker med speciella instruktioner i programtexten och de följande avsnitten beskriver uppbyggnaden av sådana program.

SEKVENTIELL FILHANTERING

En sekventiell fil kan definieras som en logisk följd av poster eller termer, som kan hämtas från ett yttre minnesmedium en efter en, i tur och ordning, till dess att filen är slut.

Den logiska följden kan jämföras med en fysisk följd och begreppet "sekventiell fil" förstår man bäst om man tänker på en datafil lagrad på kassettband. På bandet ligger alla data, logiskt och fysiskt placerade, i tur och ordning. Man kan inte få fram ett data mitt i filen utan att först läsa alla värden dessförinnan.

På en flexskiva kan den sekventiella filen ligga fysiskt uppdelad på ett flertal spår. Ur logisk synpunkt betraktas även denna skivlagrade fil som en kontinuerlig följd av termer. Den sökta termen kan läsas först efter det att man läst igenom alla framförvarande.

Vi ska i det här kapitlet titta på hur man arbetar med sekventiella filer, vilka "fällor" som kan finnas och hur man kan undvika dem. Då och då ska vi göra jämförelser med kortregistret i föregående kapitel.

Som genomgående praktiskt exempel lägger vi upp en personlig telefonkatalog med hjälp av ABC 80. Vi ska här enbart ägna oss åt sekventiella filer lagrade på flexskiva.

ATT SKAPA EN FIL

Låt oss börja med att på några olika sätt skapa en fil som påminner om det ursprungliga kortregistret med korten inlagda efterhand som de skrivs.

En tom flexskiva som förberetts för användning placeras i valfri enhet. Mata in nedanstående program i ABC 80 och lagra programmet på flexskivan.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = SKRIV8A"
120 ; "************************"
130 ; "  SKRIVNING  AV DATA PÅ"
140 ; "  SEKVENTIELL FIL, EX.8A"
150 ; "************************"
160 ;
170 REM
180 REM .  SKRIVNING AV ETT ANTAL
190 REM .  FASTA TERMER PÅ FILEN
200 REM .  "TELKAT1.DAT"
210 REM .
220 REM .  INGEN SORTERING UTFÖRS
230 REM
240 REM ----------------------------
250 REM
260 PREPARE "TELKAT1.DAT" ASFILE 1
270 REM
280 PRINT #1,"JONAS"
290 PRINT #1,"33451"
300 PRINT #1,"ANDERS"
310 PRINT #1,"12345"
320 PRINT #1,"CECILIA"
330 PRINT #1,"23456"
340 CLOSE 1
350 PRINT : PRINT
360 PRINT "SKRIVNINGEN AVSLUTAD!"
370 END
```

Vid körningen av programmet skapas (PREPARERAS) en sekventiell fil när instruktionerna på raderna 260--340 utförs. Vi börjar genomgången med raden

```
260 PREPARE "TELKAT1.DAT" ASFILE 1
```

De moment som ingår när datorn preparerar filen "TELKAT1.DAT" är dessa:

- Datorn undersöker om filen redan finns. Finns redan en fil med samma namn, "TELKAT1.DAT", kommer denna fil att skrivas över och försvinna.

- Finns ingen fil med samma namn söker datorn upp ett ledigt utrymme. Detta utrymme tilldelas filnamnet "TELKAT1.DAT", som skrivs på flexskivan.

- Filen tilldelas ett logiskt nummer, i detta fall 1. När filen är skapad ska detta nummer användas som beteckning på filen.

När man väljer namn till datafiler gäller samma regler som för val av namn till programfiler (se programmeringskortet). För att markera att denna fil är en datafil har filtypen här valts till .DAT.

För att skriva på filen används instruktionen PRINT med tillägg av nummertecknet # , filens nummer samt ett kommatecken. Den första skrivningen på denna fil sker på rad 280.

```
280 PRINT #1,"JONAS"
```

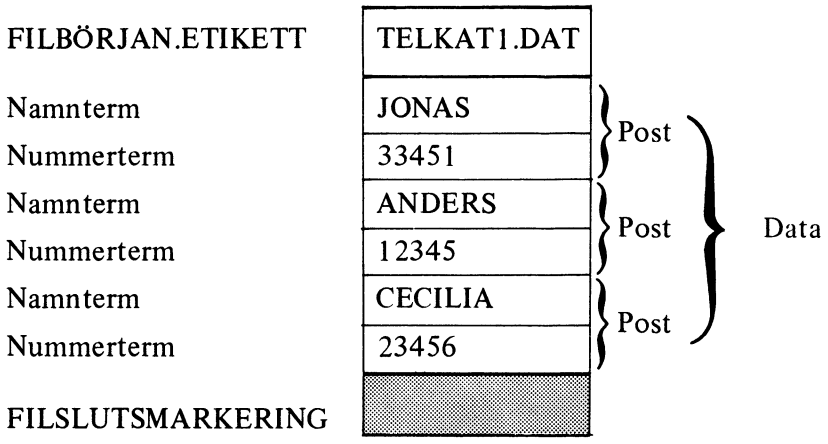
Denna programrad innebär att strängen "JONAS" skrivs på fil nummer 1. Enligt PREPARE-satsen har fil nummer 1 namnet "TELKAT1.DAT". På samma sätt lagras sedan strängarna "33451", "ANDERS" o s v i tur och ordning på fil nummer 1 enligt programraderna 290--330.

För att avsluta skrivning på en fil används instruktionen CLOSE med tillägg av filnummer. Detta sker i vårt programexempel på rad 340.

```
340 CLOSE 1
```

När denna programrad utförs skriver datorn en s k filslutsmarkering på flexskivan för att markera slutet på filen "TELKAT 1.DAT".

Filen "TELKAT 1.DAT", som skapats kan åskådliggöras som i figuren nedan.



Observera att på filen ligger data i följd (sekvens) samt så placerade att samhörande namn och telefonnummer hänger ihop.

ATT LÄSA EN FIL

Programmet "LÄS8A" nedan läser den skapade filen "TELKAT1.DAT" och presenterar filens innehåll på bildskärmen.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LÄS8A"
120 ; "************************"
130 ; "  LÄSNING  AV DATA FRÅN"
140 ; "  SEKVENTIELL FIL, EX.8A"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  UTLÄSNING GÖRS AV TERMERNA
200 REM .  NAMN     (A¤),(C¤),(E¤)
210 REM .  NUMMER   (B¤),(D¤),(F¤)
220 REM .  FRÅN FILEN "TELKAT1.DAT"
230 REM .
240 REM .  RESULTATET PRESENTERAS PÅ
250 REM .  BILDSKÄRMEN.
260 REM
```

(forts)

```
270 REM ------------------------------
280 REM
290 OPEN "TELKAT1.DAT" ASFILE 1
300 REM .       START  LÄSNING AV FILEN
310 INPUT #1,A¤
320 INPUT #1,B¤
330 INPUT #1,C¤
340 INPUT #1,D¤
350 INPUT #1,E¤
360 INPUT #1,F¤
370 CLOSE 1
380 REM .       SLUT   LÄSNING AV FILEN
390 REM
400 REM .       START UTSKRIFT AV FILEN
410 ; A¤
420 ; B¤
430 ; C¤
440 ; D¤
450 ; E¤
460 ; F¤
470 REM .       SLUT  UTSKRIFT AV FILEN
480 PRINT : PRINT
490 PRINT "BEARBETNINGEN  KLAR!"
500 END
```

De i programmet delvis nya och för filhanteringen viktiga raderna är 290, 310--360 samt 370.

```
290 OPEN "TELKAT1.DAT" ASFILE 1
```

När programrad 290 utförs söker datorn reda på filen "TELKAT1.DAT" och tilldelar denna fil nummer 1. Man brukar kalla detta att ÖPPNA filen.

Läsning från filen sker sedan när programraderna 310--360 utförs.

```
310 INPUT #1,A¤
```

Med programraden ovan tilldelas variabeln A¤ första data, "JONAS", på fil nummer 1.

På samma sätt läses sedan "33451", "ANDERS" o s v i tur och ordning från fil nummer 1 och överförs till strängvariablerna B¤, C¤ o s v. Dessa variabler måste givetvis vara av samma typ som de data, som hämtas från filen. Vid läsningen används i vårt exempel för enkelhetens skull genomgående strängvariabler.

Enligt vad vi vet om filen sedan tidigare innehåller den 6 data. När raden 360 är utförd är således alla data på filen lästa. Läsningen avslutas med att filen stängs och det sker när programraden 370 utförs.

```
370 CLOSE 1
```

Återstoden av programexemplet innebär utskrift på bildskärmen.

Med de första två exemplen har vi velat visa följande:

- Data lagras på den sekventiella filen i precis den följd de skrivs på filen.
- Data måste läsas från filen i tur och ordning och från filens början.
- Data flyttas normalt från variabler/fält i internminnet till aktuell plats på filen.
- Data läses/kopieras från filen på flexskivan till angivna variabler, "fack", i internminnet.

VARIERA FILENS LÄNGD

Det är sällan som man så exakt som i de föregående exemplen vet hur många poster eller termer en fil kommer att rymma. Normalfallet ser något annorlunda ut. Uppläggningen av termer på filen måste kunna göras ett godtyckligt antal gånger – i varje fall fram till dess att den använda flexskivan är full. Likaså måste man kunna läsa data från filen ända till dess den tar slut och utan att man från början behöver veta hur lång den är.

Det antal termer som ska ingå i filen bestäms under programkörningens gång i detta exempel:

```
100 PRINT CHR¤(12)
110 ; "Programnamn = SKRIV8C"
120 ; "************************"
130 ; "  SKRIVNING  AV DATA PÅ"
140 ; "  SEKVENTIELL FIL, EX.8C"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  SKRIVNING AV TERMERNA
200 REM .  NAMN    (A¤)
210 REM .  NUMMER  (B¤)
220 REM .  PÅ FILEN "TELKAT1.DAT"
230 REM .
240 REM .  INGEN SORTERING GÖRS.
250 REM .  INMATNINGEN AVBRYTS NÄR
260 REM .  ORDET "SLUT" GES ISTÄLLET
270 REM .  FÖR NAMN.
280 REM
290 REM ----------------------------
300 REM
310 PREPARE "TELKAT1.DAT" ASFILE 1
320 PRINT
330 PRINT "SKRIV:"
340 PRINT "NAMN/(SLUT).... ";
350 INPUT A¤
360 IF A¤="SLUT" THEN 420
370 PRINT "TELEFONNUMMER.. ";
380 INPUT B¤
390 PRINT #1,A¤
400 PRINT #1,B¤
410 GOTO 320
420 CLOSE 1
430 PRINT : PRINT
440 PRINT "INMATNINGEN AVSLUTAD!"
450 END
```

Varje gång du kör programexemplet skapas en ny version av filen "TELKAT1.DAT" eftersom prepareringen av filen på raden 310 sker med PREPARE "... ..." ASFILE ...

Inmatningen av de data som ska skrivas på filen sker från tangentbordet till strängvariablerna A¤ och B¤ . Dessa variabelvärden skrivs sedan på filen när programraderna 390 respektive 400 utförs. Både inmatningen och skrivningen på filen sker i en loop. När önskat antal poster överförts till filen avslutas skrivningen på filen då namntermen, A¤ , tilldelas värdet "SLUT",

När raden 420 utförs, stängs filen på vanligt sätt med satsen CLOSE 1. Det är oerhört viktigt vid arbete med filer att den sista skrivningen på filen följs av att en CLOSE-sats utförs, vilket stänger den aktuella filen. Detta är en förutsättning för att datorns operativsystem ska kunna hantera filer på ett tillfredsställande sätt.

LÄS EN FIL MED OKÄND LÄNGD

I de båda första exemplen, "SKRIV8A" och "LÄS8A", skrevs och lästes filer med ett antal termer som bestämts på förhand. Vid körning av programmet "SKRIV8C" skapas däremot en fil vars längd bestäms av användaren. Vi behöver därför ett program som kan läsa en fil med okänd längd. Försök med nedanstående program!

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LÄS8B"
120 ; "************************"
130 ; "  LÄSNING  AV DATA FRÅN"
140 ; "  SEKVENTIELL FIL, EX.8B"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  LÄSNING GÖRS AV TERMERNA
200 REM .  NAMN    (A¤)
210 REM .  NUMMER  (B¤)
220 REM .  FRÅN FILEN "TELKAT1.DAT"
230 REM .
240 REM .  RESULTATET PRESENTERAS PÅ
250 REM .  BILDSKÄRMEN.
260 REM
270 REM ----------------------------
```

(forts)

```
280 REM
290 OPEN "TELKAT1.DAT" ASFILE 1
300 INPUT #1,A¤
310 INPUT #1,B¤
320 PRINT A¤,,B¤
330 GOTO 300
340 CLOSE 1
350 PRINT : PRINT
360 PRINT "LÄSNINGEN AV FILEN KLAR!"
370 END
```

Enligt programtexten sker läsningen från filen "TELKAT1.DAT" till variablerna A¤ respektive B¤ i en oändlig loop på raderna 300 -- 330.

Som framgår av felutskriften på bildskärmen uppstår ett avbrott i programkörningen på rad 300 på grund av felet 34, SLUT PÅ FILEN. Avbrottet i körningen beror uppenbarligen på att datorn vid läsningen nått filslutsmarkeringen. Enligt våra regler i kapitlet FELHANTERING bör inte denna situation medföra avbrott i programkörningen eftersom felet 34 kan hanteras med instruktionen ONERRORGOTO ... Det föregående programexemplet bör därför kompletteras med instruktionen ONERRORGOTO ... på lämplig plats i programmet.

ANVÄND FILSLUTSMARKERINGEN

Det följande programexemplet är försett med den nödvändiga kompletteringen, en ONERRORGOTO ...-sats. När datorn vid läsning på filen når filslutsmärket, utlöses det hopp som förbereds när ONERRORGOTO ...-satsen utförs. I exemplet sker hoppet till den programrad som medför stängning av filen. Läsningen av filen avslutas därför på ett tillfredsställande sätt och programkörningen fortsätter utan avbrott.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LÄS8C"
120 ; "************************"
130 ; "  LÄSNING  AV DATA FRÅN"
140 ; "  SEKVENTIELL FIL, EX.8C"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  LÄSNING GÖRS AV TERMERNA
200 REM .  NAMN    (A¤)
210 REM .  NUMMER  (B¤)
220 REM .  FRÅN FILEN "TELKAT1.DAT"
230 REM .
240 REM .  RESULTATET PRESENTERAS PÅ
250 REM .  BILDSKÄRMEN.
260 REM
270 REM ------------------------------
280 REM
290 OPEN "TELKAT1.DAT" ASFILE 1
300 ONERRORGOTO 380
310 INPUT #1,A¤
320 INPUT #1,B¤
330 PRINT A¤;TAB(28);B¤ : REM utskrift
340 R=R+1 : REM .              radräknare
350 IF R<17 THEN 310
360 PRINT "NY SIDA (RETURN)"; : GET S¤
370 PRINT CHR¤(12) : R=0 : GOTO 310
380 CLOSE 1
390 PRINT : PRINT
400 PRINT "LÄSNINGEN AV FILEN KLAR!"
410 END
```

På raderna 340--370 har programmet utökats för att användaren ska kunna begära ny sida i de fall telefonlistan upptar fler än 16 poster.

Att använda fellarm för att kunna läsa en fil av godtycklig längd är en normal teknik vid filhantering. Detta gäller inte enbart i BASIC utan även i andra högnivåspråk.

I detta exempel medför alla fellarm, d v s inte enbart filslut, samma hopp i programmet. Genom att testa ERRCODE kan givetvis feltypen "filslut" väljas ut. Detta är här inte nödvändigt (jfr avsnittet KOMPLETTERA TELEFONLISTAN).

FLERA PROGRAM KAN ANVÄNDA SAMMA FIL

Vi har nu visat hur man kan skapa en sekventiell fil av godtycklig längd och även hur man med hjälp av ett enkelt program ska kunna läsa en sådan fil. Vi ska fortsättningsvis titta på några andra möjligheter att arbeta med en redan skapad datafil.

I vårt exempel med telefonkatalogen skulle bland andra följande funktioner vara användbara

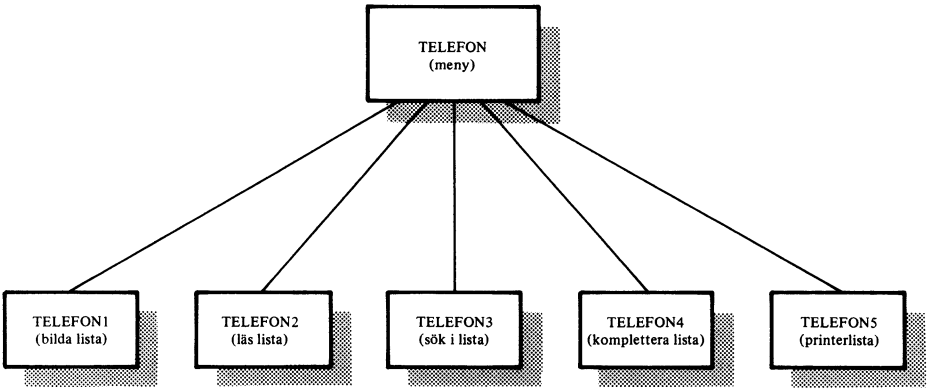
- söka telefonnummer för viss person
- söka person med visst telefonnummer
- utöka telefonkatalogen
- skriva ut katalogen på printer

De program som följer kommer att visa principer för hur dessa funktioner kan realiseras. För enkelhetens skull utgörs en post även i fortsättningen endast av en namnterm och en nummerterm. En utökning av antalet termer i varje post komplicerar inte programmen men är ej nödvändig för att illustrera principerna. Ytterligare förslag på hur felhantering används kommer att illustreras i de olika programmen.

ANVÄND MENYTEKNIK

Vi ska nu tillämpa den menyteknik som beskrivs i kap 4 för att underlätta körningen av de olika programexemplen. Dessa programexempel kan oberoende av varandra bearbeta filen "TELKAT1.DAT". Detta gör det möjligt att anropa de olika programmen med instruktionen CHAIN "...". Tekniken används normalt också när storleken hos ett program överstiger datorns internminneskapacitet (se avsnittet UTÖKA DATORNS KAPACITET, sid 85).

De program som ska ingå i systemet framgår av figuren på nästa sida.



Programmet "TELEFON" nedan presenterar en meny för val av önskat program. Mata in menyprogrammet i datorn och lagra sedan programmet på flexskiva med namnet "TELEFON" före programkörning. Eftersom du inte lagrat något av underprogrammen, "TELEFON1"--"TELEFON5" på flexskivan, fungerar inledningsvis bara menyns alternativ 6, AVSLUTA BEARBETNINGEN. Efterhand som du lagrar underprogram på flexskivan blir allt fler av menyns alternativ tillgängliga.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON"
120 ;
130 ; "**************************"
140 ; "        Meny för"
150 ; " H a n t e r i n g    a v"
160 ; "     SEKVENTIELL FIL."
170 ; "**************************"
180 ;
190 ONERRORGOTO 100
200 PRINT
210 PRINT
220 PRINT "1 = BILDA  NY TELEFONLISTA"
230 PRINT "2 = SKRIV  UT TELEFONLISTA"
240 PRINT "3 = SÖKNING I TELEFONLISTA"
250 PRINT "4 = KOMPLETTERING AV LISTA"
260 PRINT "5 = PRINTERUTSKRIVEN LISTA"
270 PRINT "6 = AVSLUTA  BEARBETNINGEN"
280 PRINT : PRINT
```

(forts)

```
290 PRINT "Välj alternativ:";
300 INPUT S
310 IF S<1 OR S>6 THEN 100
320 ON S GOTO 330,340,350,360,370,380
330 CHAIN "TELEFON1"
340 CHAIN "TELEFON2"
350 CHAIN "TELEFON3"
360 CHAIN "TELEFON4"
370 CHAIN "TELEFON5"
380 PRINT "KÖRNINGEN AVSLUTAD !"
390 END
```

SKAPA TELEFONLISTAN

Vi använder det tidigare presenterade programmet "SKRIV8C" kompletterat med raderna 300 samt 450--550. Mata in detta program och lagra det med namnet "TELEFON1".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON1" : PRINT
120 ; "************************"
130 ; "  SKRIVNING  AV DATA PÅ"
140 ; "  SEKVENTIELL FIL"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  SKRIVNING AV TERMERNA
200 REM .  NAMN    (A¤)
210 REM .  NUMMER  (B¤)
220 REM .  PÅ FILEN "TELKAT1.DAT"
230 REM .
240 REM .  INGEN SORTERING GÖRS.
250 REM .  INMATNINGEN AVBRYTS NÄR
260 REM .  ORDET "SLUT" GES ISTÄLLET
270 REM .  FÖR NAMN.
280 REM
```

```
290 REM ------------------------------
300 ONERRORGOTO 500 : REM test filfel
310 PREPARE "TELKAT1.DAT" ASFILE 1
320 PRINT : PRINT
330 PRINT "SKRIV:(MAX 12 TECKEN)" : PRINT
340 PRINT "NAMN/(SLUT).... ";
350 INPUT A¤
360 IF A¤="SLUT" THEN 420
370 PRINT "TELEFONNUMMER.. ";
380 INPUT B¤
390 PRINT #1,A¤
400 PRINT #1,B¤
410 GOTO 320
420 CLOSE 1
430 PRINT : PRINT
440 PRINT "INMATNINGEN AVSLUTAD!"
450 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
460 CHAIN "TELEFON"
470 REM
480 REM ------------------------------
490 REM
500 REM ******** FELMEDDELANDE ********
510 REM
520 ; "ETT FEL MED FELKOD ";ERRCODE;" HAR UPPSTÅTT"
530 ; "KONTROLLERA I PROGRAMMERINGSKORTET!"
540 GOTO 420
550 REM ------------------------------
```

Om något fel inträffar vid bearbetningen av filen "TELKAT1.DAT" skrivs ett felmeddelande på skärmen. Observera att instruktionen CLOSE 1 alltid utförs även om ett fel uppstått under körningen. Det är viktigt att öppnad fil alltid stängs innan programkörningen avslutas.

När den önskade filen har skapats sker återgång till huvudprogrammet "TELEFON" när denna programrad utförs:

```
460 CHAIN "TELEFON"
```

Alla underprogram som ingår i systemet avslutas på motsvarande sätt.

LÄS TELEFONLISTAN

Nedanstående program, som du lagrar med namnet "TELEFON2", känner du igen som programmet "LÄS8C" kompletterat på motsvarande sätt som programmet i föregående avsnitt.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON2"
120 ; "************************"
130 ; "  LÄSNING  AV DATA FRÅN"
140 ; "  SEKVENTIELL FIL"
150 ; "************************"
160 ;
170 REM .  KOMMENTAR :
180 REM .  =========
190 REM .  LÄSNING GÖRS AV TERMERNA
200 REM .  NAMN    (A¤)
210 REM .  NUMMER  (B¤)
220 REM .  FRÅN FILEN "TELKAT1.DAT"
230 REM .
240 REM .  RESULTATET PRESENTERAS PÅ
250 REM .  BILDSKÄRMEN.
260 REM
270 REM ----------------------------
280 ONERRORGOTO 460 : REM test filfel
290 OPEN "TELKAT1.DAT" ASFILE 1
300 ONERRORGOTO 380 : REM test filslut
310 INPUT #1,A¤
320 INPUT #1,B¤
330 PRINT A¤;TAB(28);B¤ : REM utskrift
340 R=R+1 : REM .              radräknare
350 IF R<17 THEN 310
360 PRINT "NY SIDA (RETURN)"; : GET S¤
370 PRINT CHR¤(12) : R=0 : GOTO 310
380 CLOSE 1
390 PRINT : PRINT
400 PRINT "LÄSNINGEN AV FILEN KLAR!"
410 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
420 CHAIN "TELEFON"
430 REM
440 REM ----------------------------
450 REM
460 REM ****** FELMEDDELANDE ******
470 REM
480 ; "ETT FEL MED FELKOD ";ERRCODE;" HAR UPPSTÅTT"
490 ; "KONTROLLERA I PROGRAMMERINGSKORTET!"
500 GOTO 380
510 REM ----------------------------
```

SÖK I TELEFONLISTAN

För att söka på filen "TELKAT1.DAT" använder du detta program. Sökbegrepp kan vara antingen namn eller telefonnummer. Lagra programmet med namnet "TELEFON3".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON3" : PRINT
120 ; "************************"
130 ; "  SÖKNING AV DATA PÅ"
140 ; "  EN SEKVENTIELL FIL."
150 ; "************************"
160 ;
170 ; " 1 = NAMN  -SÖKNING"
180 ; " 2 = NUMMER-SÖKNING"
190 ;
200 ; "VILKET ALTERNATIV (1/2).. "; : INPUT T¤
210 IF T¤="1" THEN ; "VILKET NAMN SÖKS..... ";
220 IF T¤="2" THEN ; "VILKET NUMMER ÖNSKAS..... ";
230 IF T¤="1" OR T¤="2" THEN 250
240 GOTO 200
250 INPUT S¤
260 PRINT
270 REM ------------------------------
280 ONERRORGOTO 500 : REM test filfel
290 OPEN "TELKAT1.DAT" ASFILE 1
300 ONERRORGOTO 390 : REM test filslut
310 INPUT #1,A¤,B¤
320 IF LEN(A¤)<LEN(S¤) THEN 340
330 IF T¤="1" AND LEFT¤(A¤,LEN(S¤))=S¤ THEN 370
340 IF LEN(B¤)<LEN(S¤) THEN 310
350 IF T¤="2" AND LEFT¤(B¤,LEN(S¤))=S¤ THEN 370
360 GOTO 310
370 ; (LEFT¤((A¤+" ................."),17))+" "+B¤
380 GOTO 310
390 CLOSE 1
400 PRINT : PRINT
410 PRINT "NY SÖKNING (J)"; : INPUT S¤
420 IF LEN(S¤)=0 THEN 100
430 S¤=LEFT¤(S¤,1)
440 IF S¤<>"N" AND S¤<>"n" THEN 100
450 PRINT "SÖKNINGEN I  FILEN KLAR!"
460 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
470 CHAIN "TELEFON"
```

(forts)

```
480 REM ------------------------------
490 REM
500 REM ******** FELMEDDELANDE ********
510 REM
520 ; "ETT FEL MED FELKOD ";ERRCODE;" HAR UPPSTÅTT"
530 ; "KONTROLLERA I PROGRAMMERINGSKORTET!"
540 GOTO 390
550 REM ------------------------------
```

Den sökning som ska utföras bestäms när programavsnittet på raderna 170--200 utförs. Om T¤ tilldelas värdet 1 avses namnsökning, medan nummersökning utförs om T¤ erhåller värdet 2.

När raderna 210--270 utförts har variabeln S¤ det värde som sökningen avser. På raden 310 läses posterna i tur och ordning från filen. Jämförelserna görs sedan på raderna 320--350. Satserna där är så konstruerade att jämförelserna bara avser så stor del av A¤ respektive B¤ som motsvarar storleken (=antalet tecken) hos S¤ .

Om det inmatade värdet, S¤ , överensstämmer med början eller hela värdet av A¤ respektive B¤ skrivs hela posten ut, d v s namn och telefonnummer presenteras på skärmen. Jämförelserna fortsätter på detta sätt till dess att hela filen är genomsökt.

KOMPLETTERA TELEFONLISTAN

Detta program ger möjlighet att utöka innehållet i en befintlig sekventiell fil. I datorn ABC 80 kan skrivning inte göras på en stängd sekventiell fil. Den möjlighet som står till buds är att skriva på en fil som är preparerad med instruktionen PREPARE "... ... ..." ASFILE ... (jfr sid 103).

När man preparerar en ny fil med samma namn som en befintlig fil kommer den gamla filen att skrivas över och försvinna. För att klara kompletteringen måste man därför skaffa sig en tillfällig hjälpfil (slaskfil) med ett annat namn än originalfilen.

För att ett system bestående av flera program ska kunna fungera, måste en fil ha samma namn före och efter en komplettering. Av den anledningen innehåller detta programexempel två kopieringar av filer. Lagra detta program med namnet "TELEFON4".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON4" : PRINT
120 ; "************************"
130 ; "  KOMPLETTERING   AV"
140 ; "  EN SEKVENTIELL FIL."
150 ; "************************"
160 ;
170 ;
180 ; "KOPIEFIL SKAPAS." : PRINT "VÄNTA!"
190 ONERRORGOTO 690 : REM test filfel
200 OPEN "TELKAT1.DAT" ASFILE 1
210 PREPARE "KOPIA.DAT" ASFILE 2
220 REM .......................
230 ; "ORIGINALFIL (1) TILL KOPIEFIL (2)"
240 PRINT "VÄNTA !"
250 ONERRORGOTO 300 : REM test filslut
260 INPUT #1,A¤,B¤
270 PRINT #2,A¤
280 PRINT #2,B¤
290 GOTO 260
300 IF ERRCODE<>34 THEN 690
310 CLOSE 1
320 CLOSE 2
330 REM .......................
340 ; "KOPIEFIL (2) TILL NY ORIGINALFIL (1)"
350 PRINT "VÄNTA !"
360 ONERRORGOTO 690 : REM test filfel
370 PREPARE "TELKAT1.DAT" ASFILE 1
380 OPEN "KOPIA.DAT" ASFILE 2
390 ONERRORGOTO 440 : REM test filslut
400 INPUT #2,A¤,B¤
410 PRINT #1,A¤
420 PRINT #1,B¤
430 GOTO 400
440 CLOSE 2
450 IF ERRCODE=34 THEN KILL "KOPIA.DAT" ELSE 690
460 PRINT
470 ; "KLART FÖR KOMPLETTERING!"
480 PRINT
490 REM
```

(forts)

```
500 REM ----------------------------
510 PRINT
520 PRINT "SKRIV:(MAX 12 TECKEN)" : PRINT
530 PRINT "NAMN/(SLUT).... ";
540 INPUT A¤
550 IF A¤="SLUT" THEN 620
560 PRINT "TELEFONNUMMER.. ";
570 INPUT B¤
580 ONERRORGOTO 690 : REM test filfel
590 PRINT #1,A¤
600 PRINT #1,B¤
610 GOTO 510
620 CLOSE 1
630 PRINT : PRINT
640 PRINT "KOMPLETTERINGEN KLAR!"
650 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
660 CHAIN "TELEFON"
670 REM ------------------------------
680 REM
690 REM ******** FELMEDDELANDE ********
700 REM
710 ; "ETT FEL MED FELKOD ";ERRCODE;" HAR UPPSTÅTT"
720 ; "KONTROLLERA I PROGRAMMERINGSKORTET!"
730 CLOSE 1
740 CLOSE 2
750 GOTO 630
760 REM ------------------------------
```

Programmets första avsnitt, raderna 180--320, innebär att innehållet i originalfilen "TELKAT1.DAT" kopieras över post för post till en hjälpfil med namnet "KOPIA.DAT". Eftersom den slutliga kompletterade filen bör ha samma namn som ursprungsfilen, d v s "TELKAT1.DAT", prepareras på rad 370 en ny "originalfil" med samma namn, nämligen "TELKAT1.DAT", varvid den gamla originalfilen försvinner.

I programavsnittet på raderna 340--440 kopieras innehållet i filen "KOPIA.DAT" över till den nya originalfilen. Den nya filen är nu öppen för fortsatt skrivning och komplettering av telefonkatalogen sker på raderna 460--760. Jämför programmet "TELEFON1".

Endast om något fel inträffat under kopieringen av data från tillfälliga filen till ny originalfil måste den tillfälliga filen sparas. I annat fall raderas filen enligt raden

```
450 IF ERRCODE=34 THEN KILL "KOPIA.DAT" ELSE 690
```

Detta är ett exempel på en enkel komplettering av en fil genom tillägg av nya poster omedelbart efter befintliga poster. Ett exempel som visar hur poster tillkommer på rätt plats i en ordnad sekventiell fil visas i nästa kapitel.

SKRIV UT TELEFONKATALOGEN MED PRINTER

Det sista av dessa programexempel används till att dokumentera filens innehåll. Sett ur datorns synpunkt är printern att betrakta som en sekventiell fil med det permanenta namnet "PR:". Lagra detta program med namnet "TELEFON5".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = TELEFON5" : PRINT
120 ; "**************************"
130 ; "  UTSKRIFT PÅ PRINTER"
140 ; "  AV SEKVENTIELL FIL."
150 ; "**************************"
160 ;
170 ;
180 PRINT "DETTA PROGRAM FÖRUTSÄTTER PRINTER !"
190 PRINT "ÄR PRINTERN ANSLUTEN/PÅSLAGEN (JA)";
200 INPUT S¤ : IF LEN(S¤)=0 THEN 230
210 S¤=LEFT¤(S¤,1)
220 IF S¤="N" OR S¤="n" THEN 430
230 PREPARE "PR:" ASFILE 2
```

(forts)

```
240 REM ----------------------------
250 ONERRORGOTO 450 : REM test filfel
260 OPEN "TELKAT1.DAT" ASFILE 1
270 ; "UTSKRIFT PÅBÖRJAS"
280 PRINT #2,"MIN EGEN PRIVATA TELEFONKATALOG"
290 PRINT #2,"=============================="
300 PRINT #2
310 ONERRORGOTO 380 : REM test filslut
320 INPUT #1,A¤,B¤
330 C¤="....................."
340 C¤=A¤+C¤
350 C¤=LEFT¤(C¤,20)
360 PRINT #2,C¤;B¤
370 GOTO 310
380 PRINT #2
390 PRINT #2,TAB(12);"===<*>==="
400 ; "UTSKRIFT AVSLUTAD"
410 CLOSE 1
420 CLOSE 2
430 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
440 CHAIN "TELEFON"
450 REM ----------------------------
460 REM
470 REM ****** FELMEDDELANDE ******
480 REM
490 ; "ETT FEL MED FELKOD ";ERRCODE;" HAR UPPSTÅTT"
500 ; "KONTROLLERA I PROGRAMMERINGSKORTET!"
510 GOTO 410
520 REM ----------------------------
```

ANPASSA PROGRAMMET FÖR ANDRA UPPGIFTER

Efter det att vi nu gått igenom några principer för hur en sekventiell fil används är det lämpligt att du gör några egna experiment för att bli mera förtrogen med filhantering.

De programexempel som har visats här kan modifieras för att klara andra filer än telefonkatalogen. Genom att byta den ledtext som presenteras på bildkärmen kan termerna namn – telefonnummer bytas mot t ex namn – adress eller artikelnamn – artikelnummer.

Det kan också vara en lämplig övning att modifiera programmen så att posterna utökas med ytterligare termer, t ex kan posterna i telefonkatalogen utökas med en adressterm.

En datafil som skapats med programmet "TELEFON2" kan användas även av det program som presenteras i nästa kapitel. En förutsättning för detta är dock att filen är sorterad så att namntermerna ligger i bokstavsordning. Detta kan åstadkommas antingen när filen skapas eller i efterhand med ett speciellt sorteringsprogram.

Ett sådant sorteringsprogram lämpligt att tillfoga i detta programsystem återfinns i övningsboken ABC-UPPGIFTER I PROGRAMMERING.

9

UPPDATERING AV ORDNAD SEKVENTIELL FIL

KRAV STÄLLS PÅ FILEN

Våra första experiment med en sekventiell fil har inneburit att vi har

- nyskapat en fil
- sökt i filen
- utökat filen med nya poster
- listat filen på skärm eller printer

Filen har skapats utan krav på att posterna ska ligga sorterade i någon bestämd ordning. Dubblettuppgifter har kunnat smyga sig in och vid sökning har datorn tvingats läsa igenom hela filen för att vara säker på att inte missa den sökta posten.

Utöver möjligheten att göra ett tillägg i slutet av filen har ingen egentlig uppdatering utförts. Felaktiga uppgifter såväl som föråldrade data har följaktligen inte kunnat ändras.

För att kunna utföra uppdateringen på ett smidigt sätt och även kunna hantera den sekventiella filen på ett rationellt vis inför vi några nya regler och begrepp.

- En av postens termer får bli filens nyckelterm och därmed också sökterm i anslutning till uppdateringen.

- Nyckeltermernas innehåll skall vara unikt i filen och får alltså bara förekomma en enda gång i filen.

- Filen skall alltid vara sorterad enligt innehållet i nyckeltermerna.

I vårt exempel, telefonkatalogen, består varje post enbart av två termer, d v s namn och telefonnummer. I den färdiga listan sker sökning efter namn och därför låter vi namntermen bli vår nyckelterm.

Detta innebär att ett givet namn bara får förekomma en enda gång per fil, men samma telefonnummer får återkomma för skilda namn. Vidare skall posterna alltid ligga sorterade i alfabetisk ordning utgående från namnen. Vi behöver inte använda något separat sorteringsprogram, utan ser helt enkelt till att posterna redan från början kommer i rätt följd.

Skulle du ha två bekanta med samma namn som ska vara med i listan löser du det genom att utöka söktermen med någon för varje person unik information, t ex adress.

OMFATTANDE SÖKNYCKLAR

Många tillämpningar med filer kräver omfattande nycklar bestående av antingen flera termer eller flera olika begrepp samlade i samma term. Ibland kan en söknyckel vara så omfattande att den kräver för stort utrymme i posten. Söknyckeln är kanske inte heller lämpad för sökning eller lagring av data i aktuell filtyp – det finns ju fler sätt att organisera data i filform än i sekventiella filer, även om den formen både är lätt att förstå och samtidigt vanlig. Det händer därför ibland att man ersätter det aktuella sökbegreppet med ett nytt. De ursprungliga nyckeluppgifterna kan då tillsammans med den nya nyckeln bilda poster på en egen fil, och sedan används bara det nya sökbegreppet i övriga filer.

Behövs sedan vid något tillfälle de ursprungliga uppgifterna läser man de båda filerna samtidigt – parallellt – och väljer ut de ursprungliga uppgifterna från den ena filen, nyckelfilen, och parar ihop dem med motsvarande post i den andra filen. Eftersom nyckelfilen kan innehålla många fler poster än vad som är aktuellt i den fil som bearbetas är det nödvändigt att posterna ligger sorterade på samma sätt, t ex i stigande följd. Självklart är nu att filen under arbete inte får innehålla sökbegrepp som saknar motsvarighet i nyckelfilen.

Exempel på en ersättningsnyckel är vårt personnummer, som ju ersätter just namn, adress m fl uppgifter, som behövs för att exakt kunna identifiera en person och därmed kunna skilja vederbörande från en annan person med nästan samma uppgifter. Ett annat exempel är ett s k artikelnummer, som används för att definiera ett speciellt föremål som kan finnas i ett förråd, eller i en konstruktion. Så skulle t ex 693954-2 kunna betyda "Skruv, M3-gänga, längd 15 mm, försänkt skalle, material stål".

Försök inte hitta någon överensstämmelse mellan numret och sakuppgifterna för det finns ingen! Ett varningens ord kan här vara på sin plats. Nästan undantagslöst spricker förr eller senare de system av nummer där man försöker lägga någon form av begrepp i delar av eller i hela numret. Har numren hunnit in i ett antal datafiler kan det kosta stora pengar att göra erforderliga byten. Så har du någon gång behov av identifieringsdata kom ihåg att ett vanligt hederligt löpnummer alltid går att utöka.

UPPDATERING AV TELEFONKATALOGEN

Låt oss efter den här kortfattade beskrivningen över söknycklar i samband med sekventiella filer återvända till telefonkatalogen. Vi får för exemplets skull finna oss i söknyckelförenklingen, som bara tillåter oss att ha personer med olika namn i vår telefonlista. Vi vill nu kunna uppdatera filen. Detta innebär att

- placera nya poster på rätt plats på den ordnade filen
- ändra befintliga poster på filen
- avlägsna icke önskade poster från filen

Du känner igen dessa moment från arbetet med kortlådan i kap 7. Med en datafil utförs dessa moment på motsvarande sätt. Datorn letar i filen till dess sökt post påträffas, och gör sedan avsedd uppdatering i filen. Förfarandet blir något olika beroende på om uppdateringen gäller

- post tillkommer
- post ändras
- post utgår

Låt oss illustrera tillvägagångssättet med ett programexempel som gäller alternativet "post tillkommer".

MANUELL UPPDATERING VISAR PRINCIPEN

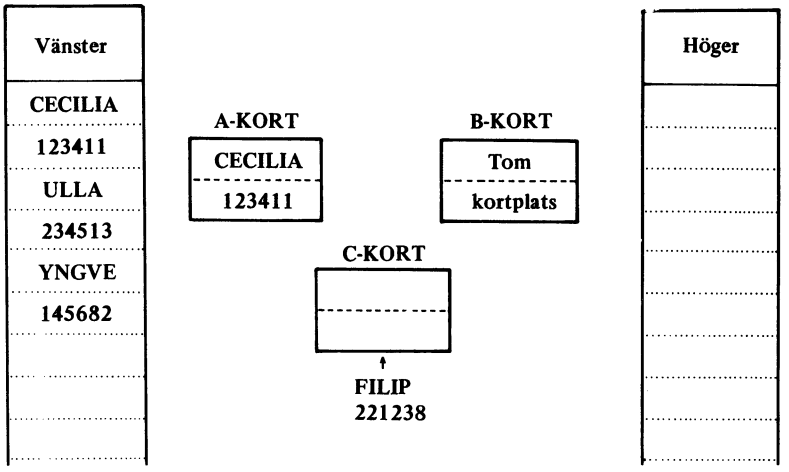
För att få en praktisk bakgrund till kommande beskrivning utgår vi från följande sammanställning och försöker analysera i detalj hur vi egentligen skulle resonera vid manuell utökning av telefonkatalogen. Det är nämligen på samma sätt vi måste få maskinen att göra analysen via programmet. När man bläddrar i en kortbunt har man möjlighet att bläddra åt båda hållen. När datorn läser en datafil kan däremot filen enbart läsas post för post från filens början. Vi vill därför undvika att datorn tvingas läsa hela filen för varje ny tillkommande post. För att göra exemplet med manuell bearbetning så maskintroget som möjligt utgår vi från följande förutsättningar:

- Telefonlistan föreligger i form av ett kortregister
- Hela posten (namn och telefonnummer) finns på varje korts framsida
- På baksidan av varje kort finns dessutom sökordet – namnet
- Du plockar ut kortbunten ur registerlådan och lägger bunten till vänster (se fig)
- Du flyttar vid behov korten från den vänsta högen till en ny hög till höger, med baksidan upp så att du ser söktermen (se fig)
- Korten – posterna – kan bara flyttas styckvis från vänster till höger
- Förflyttning av hel/full hög kan bara ske från höger till vänster

När du nu startar, har du ett utgångsläge enligt figuren. Kortbunten ligger till vänster med första posten synlig. Den högra kortplatsen är tom. På

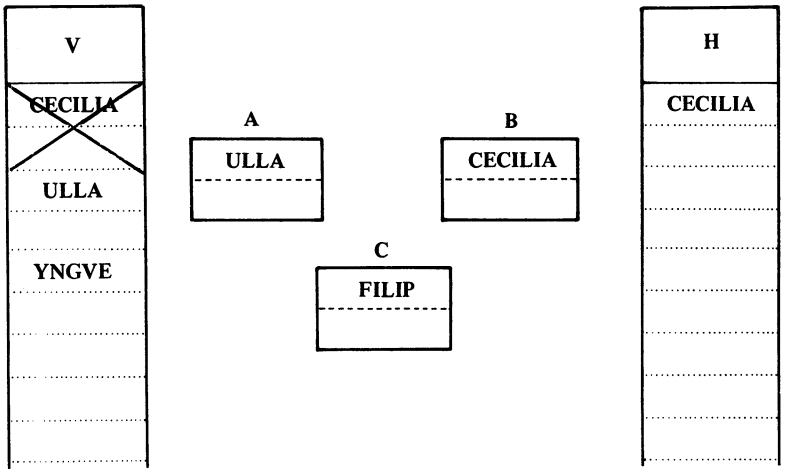
de följande bilderna visar vi bara söktermen och underförstår att tillhörande telefonnummerterm följer med vid förflyttningarna. Tre kort kommer att vara synliga under uppdateringen.

- A-kortet i figurerna är den synliga posten i vänster kortbunt
- B-kortet visar sökordet för den senast tillförda posten i den högra kortbunten
- C-kortet är den post som skall tillföras kortregistret



Anta nu att du vill uppdatera katalogen med posten "FILIP", "221238". Skriv ett sådant kort och placera det på C-kortets plats.

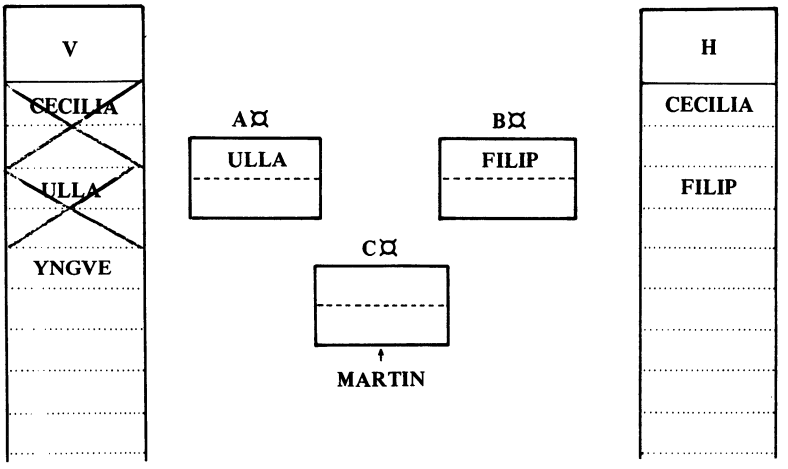
Eftersom CECILIA < FILIP och båda är större än den tomma kortplatsen, väljer du att flytta kortet med minsta värdet av de två, nämligen "CECILIA", till den tomma högen. Varje förflyttning av ett kort från vänster hög innebär att nästa kort blir synligt, nämligen "ULLA".



Kortplatserna har efter förflyttningen följande innehåll:

A	C	B
ULLA	FILIP	CECILIA

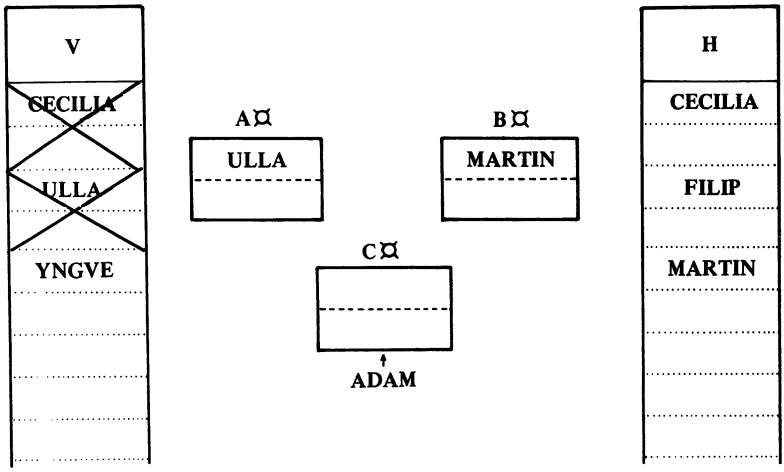
En jämförelse ger att C < A och C > B, alltså flyttar vi enligt reglerna FILIP till den högra kortplatsen.



Eftersom C-platsen nu blivit tom kan uppdateringen fortsätta med ett nytt registerkort, t ex med söktermen "MARTIN".

A¤	C¤	B¤
ULLA	MARTIN	FILIP

Eftersom jämförelsen egentligen avser strängar inför vi strängvariabler som beteckningar för kortplatserna. Fortfarande gäller C¤ < A¤ och C¤ > B¤ alltså MARTIN ut till höger!



Dags igen för ett nytt kort: ADAM

A¤	C¤	B¤
ULLA	ADAM	MARTIN

Tydligen gäller C¤ < A¤ och C¤ < B¤. Här är det egentligen ointressant att det nya kortets värde är mindre än det som är aktuellt i ursprungshögen – ULLA – eftersom ju även nytillkommande registerkort enligt reglerna bara kan flyttas till höger.

För det är förargligt nog att C¤ < B¤! Detta innebär att det nya kortet ska ligga någonstans nere i den påbörjade högra högen. Detta hade vi kunnat undvika om de tillkommande posterna varit sorterade i förväg.

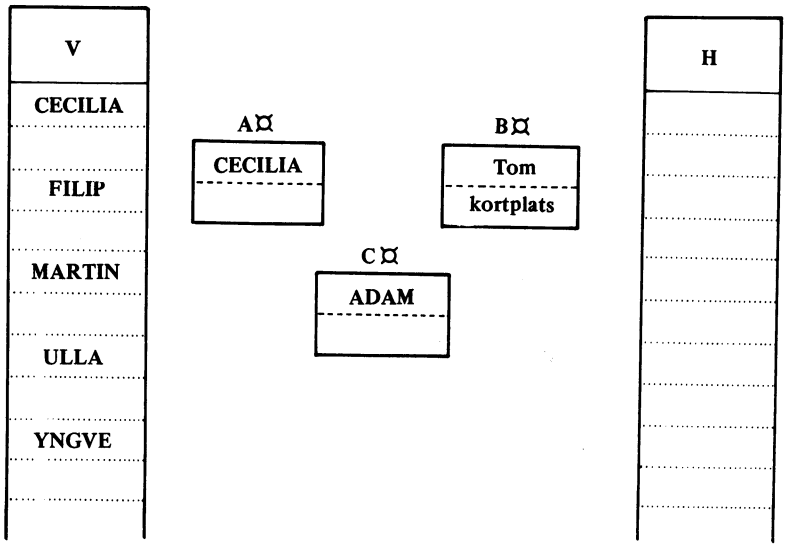
Nåväl, skadan är inte värre än att den går att reparera, men det blir på bekostnad av tid och arbete. Och även om vi hade haft en dator till vår hjälp hade det kostat tid eftersom inte ens den kan göra allt arbete på "nolltid" trots att det ibland verkar så.

I princip måste vi tyvärr börja om från början, men samtidigt vill vi ju inte förstöra det som redan är gjort. Eftersom vi vet att alla posterna i vänstra högen ligger sorterade i stigande ordning och att samtliga är större än det

översta i den högra, tar vi korten ett i taget och flyttar från vänster till höger. När den vänstra högen är tom så vänder vi på den högra och flyttar den till den vänstra högens plats (kortet längst ned i den "gamla" högerhögen ligger alltså nu överst till vänster. Nu gäller

A¤	C¤	B¤
CECILIA	ADAM	"tom"

Därmed fungerar våra regler igen!

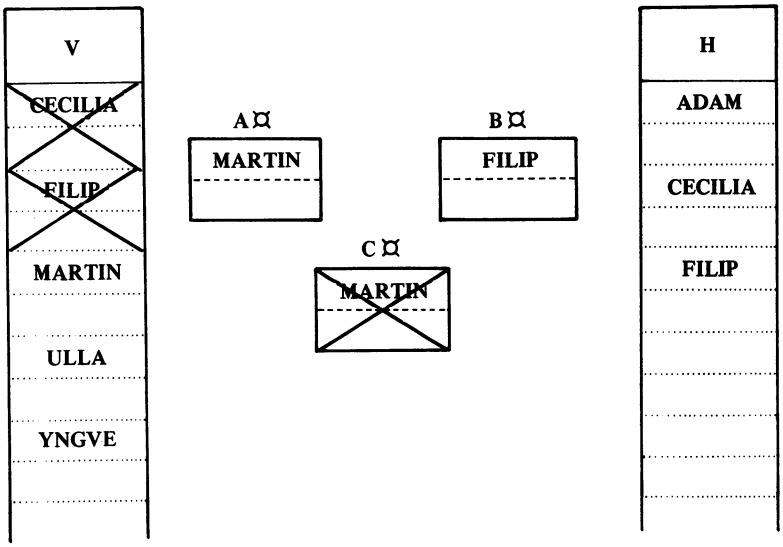


Här är C¤ < A¤ och C ¤ > B¤ och eftersom vi alltid flyttar kortet med det minsta värdet av A¤ och C¤ till höger – under förutsättning att det samtidigt är större än B¤ – hamnar här ADAM som första kort i den högra högen.

Dags att skriva ett nytt kort eftersom C¤ nu är tom. Vi ska avslutningsvis studera vad som händer när vi försöker uppdatera registret med en post vars söknyckel redan finns på ett kort i registret. Skriv ett kort med namnet MARTIN igen. Just nu kan vi inte se att det redan finns en post med denna söknyckel. Korten vi ser är nämligen

A¤	C¤	B¤
CECILIA	MARTIN	ADAM

Därför sätter du igång och flyttar korten enligt våra regler.



När A¤ blir lika med MARTIN drar vi oss plötsligt till minnes en av våra regler, som säger att det bara får finnas en enda post för varje sökbegrepp. Alltså är det bara att kasta det sista, nyskrivna registerkortet. Om så önskas kan uppdateringen fortsätta med ett annat namn men vi väljer att sluta uppdateringen här.

Eftersom den vänstra högen inte är tom, måste vi göra som då vi tidigare behövde vända registret, nämligen flytta korten, ett och ett, från vänster till höger. När den vänstra högen är tom vänder vi den högra bunten och placerar den i originalfodralet. Därmed är vår uppdatering av tillkommande poster klar. Du behöver aldrig bläddra igenom alla korten när du letar efter numret till en viss person. Skulle du t ex inte finna NILS efter MARTIN i registret behöver du inte leta länge eftersom registret i så fall inte innehåller någon sådan uppgift.

Nu kan det vara dags att studera hur vi skall göra denna behandling av våra data med hjälp av ett program och ABC 80.

NYCKELTERMER AV INTRESSE

Det första vi ska undersöka är vilka relationer som egentligen kan råda mellan de data vi måste ha i internminnet. Det är resultatet av den utvärderingen, som i varje läge bestämmer programmets fortsatta behandling av aktuella data. Liksom i föregående exempel är det tre datatermer/nyckeltermer som samtidigt är av intresse nämligen

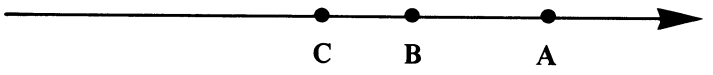
A¤ = senast lästa post från utgångsfilen

B¤ = senast skrivna post på resultatfilen

C¤ = aktuell post att tillföra resultatfilen

Det finns flera olika sätt att komma fram till vilka teoretiska relationer – storleksförhållanden – som kan existera mellan dessa dataelement. Det är lätt att förbise vissa samband, som inte är självklara. Den följande uppställningen torde dock täcka in samtliga fall som är av intresse. För att få tabellerna mer renodlade och lättlästa är ¤-tecknet borttaget.

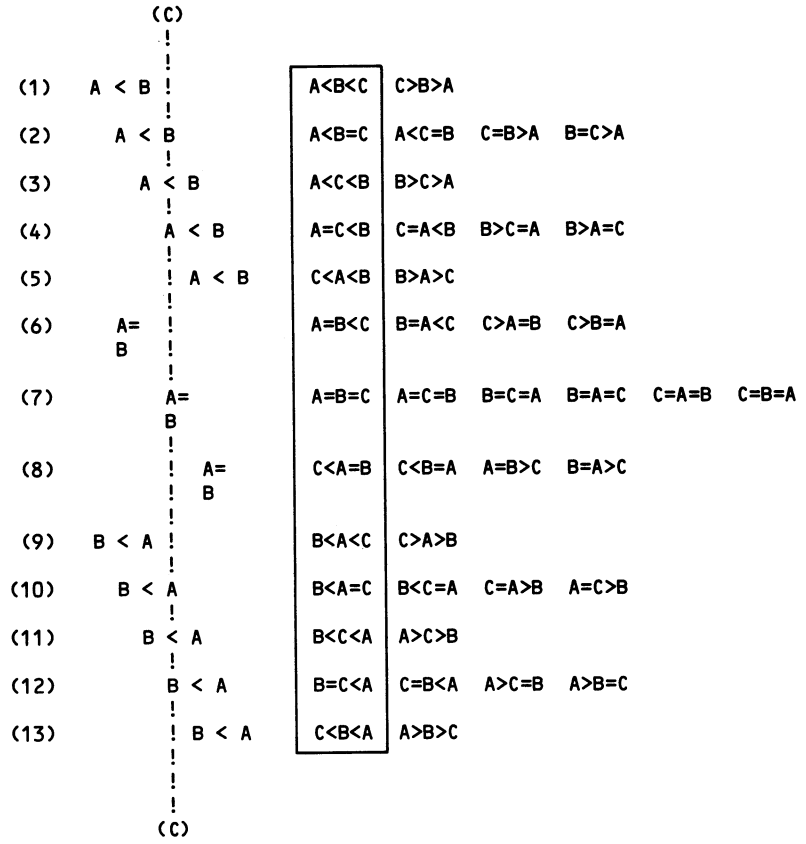
Du tolkar tabellen på följande sätt. Tänk dig en slags tallinje, eller kanske bättre en värdelinje där vi låter variabeln C (den tillkommande posten) utgöra en referenspunkt. Allt till vänster om denna punkt är då mindre än (<) C och allt till höger större än (>) C.



Denna relation uttrycker vi som C < B < A.

TABELL ÖVER MÖJLIGA RELATIONER

Sammanställningen nedan visar vilka relationer som kan förekomma.



"En sådan rysansvärd mängd fall" tänker du kanske en smula uppgivet! Men var lugn, det hela är mycket enklare än vad du tror.

Det finns 13 olika fall i uppställningen numrerade 1-13. I den "grafiska" framställningen omedelbart till höger om numreringen ser du dessa fall bildmässigt inplacerade på så sätt att två av våra storheter (A och B) är relaterade till den tredje (C).

Sedan följer till höger sex spalter med varierande antal förhållanden angivna. De relationer som står på en och samma rad, uttrycker samma sak men med de förändringar som uppstår om vi ändrar operandernas (A – B – C) inbördes placeringar. De för vår fortsatta framställning grundläggande 13 relationerna finns alltså i den inramade spalten.

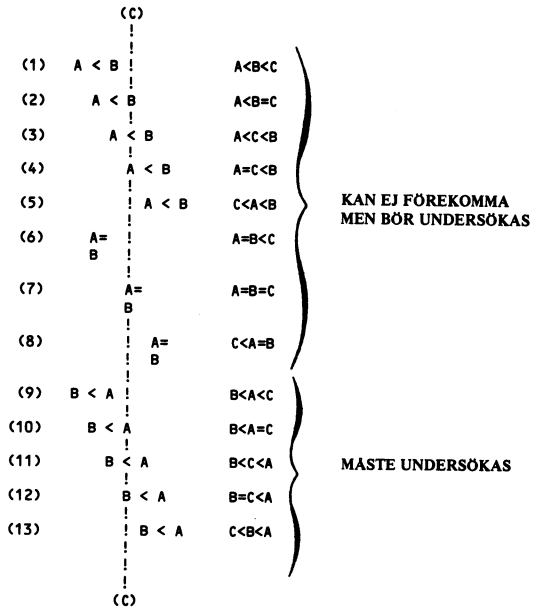
De till synes onödiga dubbletterna har tagits med därför att du sannolikt kommer att behöva fundera över sådana här relationer i samband med egna tillämpningar. Då underlättar sammanställningen den tankemöda som ofta krävs för att genomskåda att en aktuell relation endast är en av de välbekanta 13.

FILENS BEGRÄNSNINGAR UNDERLÄTTAR JÄMFÖRELSERNA

Vi har tidigare ställt krav på den fil som ska uppdateras, bl a att

- filens poster är lagrade i stigande följd (ASCII-ordning, jfr programmeringskortet)
- samma sökterm kan bara förekomma en gång

Detta betyder i vårt fall att relationerna 1-8 ej behöver undersökas, eftersom A¤ < B¤ eller A¤ = B¤ inte får förekomma. Skulle de ändå finnas betyder det att du har ett datafel i filen. Det kan uppstå när du försöker uppdatera en fil som inte uppfyller dessa krav.



Återstår gör nu bara de 5 relationerna 9-13. Därmed går vi över till vårt uppdateringsprogram "TELEFON6" där vi nu skall se hur vi tar hand om de här relationerna.

PROGRAMMETS FÖRUTSÄTTNINGAR

Programmet förutsätter i den form som ska presenteras, att uppdateringen sker på en fil som redan existerar och som måste vara upplagd med posterna i rätt följd enligt sorteringsnyckeln. Filen måste alltså ha minst 1 post. För att inte skymma de väsentliga principerna har t ex felhanteringen inskränkts till minsta möjliga d v s i stort sett endast till kontroll av filslut. Därför bör du vara medveten om att fel kan inträffa som avbryter körningen för dig, men du kan utan vidare starta om igen. Du börjar då vid samma punkt som där du startade den avbrutna körningen.

Det kompletta exemplet innehåller också ett antal rader vars enda uppgift är att visa dig i vilket skede programgenomloppet befinner sig. Före varje utvärdering av relationerna visas på bildskärmen de värden som variablerna A¤ , C¤ och B¤ för tillfället har. När du trycker på returtangenten utförs den bearbetning som innebär att posterna flyttas enligt reglerna.

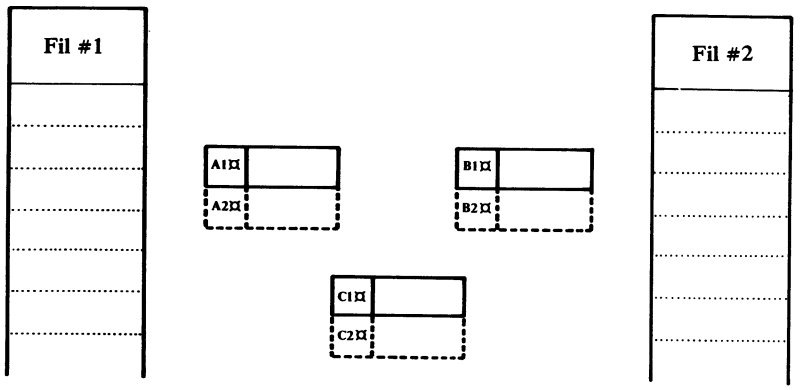
PROGRAMMETS FILER

Det är önskvärt att programmet skall kunna ta hand om flera varianter av vår telefonkatalog. Därför är det inte knutet till "TELKAT1.DAT" enbart utan filens namn kan anges av användaren under körning.

Eftersom allt vårt uppdateringsarbete innebär att läsa och skriva fram och tillbaka mellan filer, kopieras därför vår originalfil till en tillfällig arbetsfil, en "slaskfil" (jfr beskrivningen i kap 8).

Denna fil får tjäna som vårt nya arbetsoriginal. Vi skulle mycket väl kunna arbeta med vår ursprungliga originalfil, men för att inte riskera att förstöra den gör vi en kopia. Det värsta som då kan hända är att den uppdatering vi avser att göra går förlorad (vi förutsätter att du har en

säkerhetskopia av din originalfil på en helt annan skiva!). Vad som nu skall göras illustrerar vi på följande vis:



Som du ser använder vi oss av samma figur som användes i exemplet med manuell uppdatering. De båda högarna har bara bytts mot filerna #1 och #2.

Vårt nya arbetsoriginal, "SLASKA", kallar vi fil #1. Från den filen läser vi posten, namn och telefonnummer, till variablerna A1¤ (namn) och A2¤ (telefonnummer). I variablerna C1¤ och C2¤ lägger vi det nya namn med tillhörande telefonnummer som vi ska placera på rätt plats i filen. I B1¤ och B2¤ har vi också namn och telefonnummer, men dessa motsvarar innehållet i den senaste posten vi skrev på fil #2. Fil #2 är en annan arbetsfil, som vi skapat på flexskivan under namnet "SLASKB". Innehållet i B1¤ och B2¤ kan således komma antingen från senast överförda post från fil #1 (A1¤ , A2¤) eller från vår senast tillkommande post (C1¤ , C2¤).

I programavsnittet på följande sida utförs de beskrivna uppstartningsåtgärderna. Den första posten hämtas fram från fil #1 samt initieras den första inmatningen av en tillkommande post till C1¤ , C2¤ .

```
100 ; CHR¤(12)
110 ; "Programnamn = TELEFON6"
120 ;
130 ; "************************"
140 ; "  UPPDATERING  AV DATA"
150 ; "  PÅ  SEKVENTIELL  FIL"
160 ; "------------------------"
170 ; "  POSTER TILLKOMMER!"
180 ; "************************"
190 ;
200 ;
210 REM . _____________________ <FILINITIERING>
220 ; "GE NAMNET PÅ  FILEN"
230 ; "SOM SKALL BEARBETAS: "
240 ; "(TELKAT1.DAT)"; : INPUT F¤ : REM stand.svar
250 IF LEN(F¤)=0 THEN F¤="TELKAT1.DAT"
260 ;
270 ; "KOPIERING FRÅN ORIGINALFIL"
280 ; "========================" : ;
290 ; "   Förberedelser.... START"
300 ONERRORGOTO 310
310 S=ERRCODE
320 IF S=21 THEN ; : ; : ; "   FILEN FINNS EJ!"
330 IF S=21 THEN FOR I=1 TO 5000 : NEXT I : GOTO 1660
340 OPEN F¤ ASFILE 3
350 PREPARE "SLASKA.DAT" ASFILE 1
360 ; "                    KLART" : ;
370 ; "   Kopiering..... START"
380 ONERRORGOTO 420
390 INPUT #3,A¤
400 PRINT #1,A¤
410 GOTO 390
420 CLOSE 1
430 CLOSE 3
440 ; "                    KLART" : ;
450 ; "   Filöppningar..... START"
460 F1¤="SLASKA.DAT"
470 F2¤="SLASKB.DAT"
480 REM . _____________ <INMATNINGSINITIERING>
490 OPEN F1¤ ASFILE 1
500 PREPARE F2¤ ASFILE 2
510 INPUT #1,A1¤,A2¤
520 B1¤="" : REM tom sträng
530 B2¤="" : REM tom sträng
540 ; "                    KLART" : ;
550 GOSUB 1730 : REM =(NY POST IN)=====
```

Inmatningen av tillkommande post utförs med hjälp av följande subrutin (raderna 1730--1860)

```
1680 REM
1690 REM
1700 REM ******************************
1710 REM .     S U B R U T I N E R
1720 REM ..............................
1730 REM . ____________________ <INMATN. AV NY POST>
1740 ; CHR¤(12)
1750 GOSUB 1870 : REM =(TESTHJÄLP)=====
1760 ; CHR¤(12)
1770 ;
1780 PRINT "TILLKOMMANDE POST"
1790 ; "(MAX 12 TECKEN/TERM)" : PRINT
1800 ; "NAMN/(SLUT).... ";
1810 INPUT C1¤
1820 IF C1¤="SLUT" THEN 1860
1830 ; "TELEFONNUMMER.. ";
1840 INPUT C2¤
1850 ;
1860 RETURN
```

På raden 1750 anropas nedanstående subrutin, kallad TESTHJÄLP. Denna rutin har som enda uppgift att illustrera hur posterna förflyttas mellan variabler och filer under körningens gång. Den här subrutinen samt motsvarande GOSUB-satser kan givetvis avlägsnas från programmet när du väl har förstått hur uppdateringen går till.

```
1870 REM . ____________________________ <TESTHJÄLP>
1880 ;
1890 ; "======================================="
1900 REM ;
1910 ; "A1¤/A2¤","C1¤/C2¤","B1¤/B2¤"
1920 ; "_____________________________________" : ;
1930 ; A1¤,C1¤,B1¤
1940 ; A2¤,C2¤,B2¤
1950 REM ;
1960 ; "======================================="
1970 ;
1980 ; "(TRYCK PÅ RETURN!)" : GET S¤
1990 RETURN
```

PROGRAMMETS TESTER

Nu är det dags för programmet att ta hand om våra tidigare nämnda relationer för utvärdering.

Detta sker i 5 på varandra föjande avsnitt, som vardera börjar med en IF-sats där utvärderingen sker. Om villkoret inom parentesen inte är uppfyllt sker hopp till nästa utvärderingsläge.

TEST 1

```
600 IF NOT (C1¤=A1¤ OR C1¤=B1¤) THEN 660
```

TEST 2

```
680 IF NOT (A1¤=B1¤ OR A1¤<B1¤) THEN 760
```

TEST 3

```
780 IF NOT (B1¤<A1¤ AND A1¤<C1¤) THEN 1000
```

TEST 4

```
1020 IF NOT (B1¤<C1¤ AND C1¤<A1¤) THEN 1130
```

TEST 5

```
1150 IF NOT (C1¤<B1¤) THEN 1280
```

Här följer nu motsvarande programavsnitt i tur och ordning.

TILLKOMMANDE POST FINNS REDAN
TEST 1 (C = A ELLER C = B)

Första relationen som testas motsvarar alternativen 10 och 12 i vår tidigare sammanställning. Här är det dock ointressant med relationerna mellan A¤ och B¤ . Det räcker att konstatera att C¤ är lika med

endera A¤ eller B¤ . Det innebär ju att posten redan finns i filen och då får det alltså inte tillkomma någon ny med samma sökbegrepp.

```
560 REM .                    <UNDERSÖKNING AV POST>
570 IF C1¤="SLUT" THEN 1340
580 REM .  ..................  <Posten finns redan>
590 ; "TEST 1   (C1¤=A1¤ OR  C1¤=B1¤)"
600 IF NOT (C1¤=A1¤ OR C1¤=B1¤) THEN 660
610 GOSUB 1870 : REM =(TESTHJÄLP)======
620 S¤="POSTEN FINNS REDAN! KAN EJ TILLKOMMA!"
630 GOSUB 2000 : REM =(LARMTEXT UT)====
640 GOSUB 1730 : REM =(NY POST IN)=====
650 GOTO 560
```

För att lägga ut en larmtext om att POSTEN FINNS REDAN, KAN EJ TILLKOMMA används subrutinen FEL/LARM-UTSKRIFT.

```
2000 REM .                   <FEL/LARM-UTSKRIFT.>
2010 REM
2020 ; CUR(24,1);"                                "
2030 ; CUR(24,1);S¤
2040 OUT 6,7
2050 FOR I=1 TO 1250 : NEXT I
2060 OUT 6,0
2070 ; "(TRYCK PÅ RETURN!)" : GET S¤
2080 RETURN
2090 REM
2100 REM
```

Efter det att en ny post matats in startar en ny jämförelse på programrad 560.

KONTROLLERA URSPRUNGSFILEN
TEST 2 (A = B ELLER A < B)

Den andra test som utförs motsvarar de 8 första alternativen i vår relationstabell. Dessa alternativ kan inte inträffa om filen uppfyller de givna förutsättningarna.

```
660 REM .  ..................... <Datafel i filen>
670 ; "TEST 2   (A1¤=B1¤ OR  A1¤<B1¤)"
680 IF NOT (A1¤=B1¤ OR A1¤<B1¤) THEN 760
690 GOSUB 1870 : REM =(TESTHJÄLP)======
700 ;
710 ; "A1¤= ";A1¤
720 ; "B1¤= ";B1¤
730 S¤="DATAFEL FUNNET. KONTROLLERA DINA FILER!"
740 GOSUB 2000 : REM =(LARMTEXT UT)====
750 ; : CLOSE 1 : CLOSE 2 : END
```

Denna variant är dock medtagen för alla eventualiteters skull, för om den inträffar så har du fått ett så allvarligt fel i filen att det är nödvändigt att avbryta bearbetningen, dock först sedan vi gett information om detta på skärmen.

BLÄDDRA I FILEN
TEST 3 (B < A OCH A < C)

Detta är ett av de mest använda avsnitten i programmet. Som du väl redan upptäckt, motsvarar villkoret bläddringsfunktionen i kortlådan.

```
760 REM .  ............................ < B<A<C >
770 ; "TEST 3   (B1¤<A1¤ AND A1¤<C1¤)"
780 IF NOT (B1¤<A1¤ AND A1¤<C1¤) THEN 1000
790 GOSUB 1870 : REM =(TESTHJÄLP)======
800 ONERRORGOTO 920
810 PRINT #2,A1¤
820 PRINT #2,A2¤
830 B1¤=A1¤
840 B2¤=A2¤
850 ONERRORGOTO 900 : REM test filslut #1
860 INPUT #1,A1¤
870 INPUT #1,A2¤
880 GOTO 560 : REM =(POSTUNDERSÖKNING)=
```

(forts)

```
890 REM ----------------------------
900 PRINT #2,C1¤
910 PRINT #2,C2¤
920 ONERRORGOTO 960 : REM skivan full?
930 CLOSE 1
940 CLOSE 2
950 GOTO 970
960 ; "FILFEL! (ERR=";ERRCODE;")" : END
970 GOSUB 2110 : REM =(FILVÄNDNING)===
980 GOSUB 1730 : REM =(NY POST IN)====
990 GOTO 560 : REM =(POSTUNDERSÖKNING)
```

I detta fall är söktermens värde hos den post vi skall tillföra (C1¤ , C2¤) större än båda A1¤ och B1¤ . Vi flyttar alltså den senast lästa posten från fil #1 till fil #2 vilket sker i raderna 810 och 820.

Nu måste vi i B1¤ - och B2¤ -termerna lagra de värden vi senast skrivit på fil #2. Detta hittar du i raderna 830 och 840. Slutligen återstår det allra viktigaste, nämligen att läsa nya värden från fil #1 till A1¤ - och A2¤ -termerna och det görs i raderna 860 och 870. Därefter upprepas relationsundersökningen igen.

Här är på sin plats att ge en praktisk regel som undanröjer många fällor och bekymmer när du kontruerar program med sekventiella filer.

VARHELST I PROGRAMMET EN POST FRÅN EN VISS FIL ÄR FÖRBRUKAD, SKALL OMEDELBART NÄSTA POST HÄMTAS FRAM

I det här programavsnittet finns en sådan fälla som vi nu klarar av tack vare en omedelbar läsning på fil #1. Problemet uppstår när det inte finns något mer att hämta ur fil #1 d v s vi har hittat filslut. Filslutet medför här två saker:

- Den tillkommande posten C1¤ , C2¤ har funnit sin rätta plats, d v s den ska ligga sist i fil #2.
- För att kunna fortsätta uppdateringen av filen måste vi kunna bläddra i den och det kan vi bara göra från dess början. Vi måste alltså vända på den, d v s vi låter fil #1 och fil #2 byta plats.

Allt detta sker i raderna 900--990. Filbytet utförs även i andra avsnitt av programmet och är därför utfört som en subrutin, FILVÄND. STARTVÄRDE.

```
2110 REM .  _____________________  <FILVÄND.STARTVÄRDE.>
2120 S¤=F1¤
2130 F1¤=F2¤ : F2¤=S¤
2140 ; "VÄNTA!"
2150 ONERRORGOTO 2210
2160 OPEN F1¤ ASFILE 1
2170 PREPARE F2¤ ASFILE 2
2180 INPUT #1,A1¤,A2¤
2190 B1¤="" : B2¤="" : REM tom sträng
2200 ONERRORGOTO 0 : GOTO 2240
2210 ; "FEL VID FILVÄNDNING= ";ERRCODE
2220 CLOSE 1 : CLOSE 2 : CLOSE 3
2230 ; "VI AVBRYTER!" : STOP
2240 RETURN
2250 REM ******************************
```

Som du ser sker bytet helt enkelt genom att filnamnen byter plats i variablerna F1¤ och F2¤ . Och eftersom det är variablerna och inte filnamnen som är kopplade till filnumren i de efterföljande filöppningssatserna behöver vi egentligen aldrig själva hålla reda på vilken av våra slaskfiler som är ut-fil (#1) respektive in-fil (#2).

Den uppmärksamme läsaren har redan insett att vi egentligen inte skulle ha gjort den här filvändningen ännu, eftersom nästa post kanske också är större än den sist inlästa och alltså också ska ligga sist i fil #2. En sådan utökning av programmet är givetvis möjlig och skulle i vissa fall spara körningstid. I vårt program jämförs dock nästa tillkommande post med första posten i vår fil #1.

TILLKOMMANDE POST SKRIVS PÅ FILEN
TEST 4 (B < C OCH C < A)

När detta vilkor är uppfyllt är det dags att skriva den tillkommande posten på filen.

```
1000 REM .  ..............................  < B<C<A >
1010 ; "TEST 4   (B1¤<C1¤ AND C1¤<A1¤)"
1020 IF NOT (B1¤<C1¤ AND C1¤<A1¤) THEN 1130
1030 GOSUB 1870 : REM =(TESTHJÄLP)=====
1040 ONERRORGOTO 1100 : REM skivan full?
1050 PRINT #2,C1¤
1060 PRINT #2,C2¤
1070 B1¤=C1¤ : B2¤=C2¤ : ONERRORGOTO 0
1080 GOSUB 1730 : REM =(NY POST IN)====
1090 GOTO 560
1100 ; "FILFEL /SKIVAN FULL?/"
1110 ; "FELKOD = ";ERRCODE
1120 CLOSE 1 : CLOSE 2 : END
```

Vi skriver alltså C1¤ och C2¤ på fil #2. Efter kopiering till B1¤ - och B2¤ -termerna sker inmatning av nästa nya post, som ska tillföras filen.

LÄS TILL FILEN SLUT
TEST 5 (C < B)

Här är det uppenbart att den tillkommande posten ska ligga någonstans bland de poster som redan finns inlagda i fil #2. Vi måste då kunna läsa den filen från början. Innan vi kan göra det måste resten av fil #1 föras över till fil #2 (raderna 1180–1220) för att slutet av fil #1 inte ska gå förlorad. Därefter byter filerna namn och det görs som förut i subrutinen FILVÄND STARTVÄRDE.

```
1130 REM .  ..............................  <  C<B  >
1140 ; "TEST 5   (C1¤<B1¤)"
1150 IF NOT (C1¤<B1¤) THEN 1280
1160 GOSUB 1870 : REM =(TESTHJÄLP)=====
1170 PRINT "VÄNTA!"
1180 ONERRORGOTO 1240 : REM test filslut #1
1190 PRINT #2,A1¤
1200 PRINT #2,A2¤
1210 INPUT #1,A1¤,A2¤
1220 GOTO 1180
1230 ; "FEL NR ";ERRCODE;"VID FILSTÄNGNING"
1240 ONERRORGOTO 1230
1250 CLOSE 1 : CLOSE 2
1260 GOSUB 2110 : REM =(FILVÄNDNING)===
1270 GOTO 560 : REM =(POSTUNDERSÖKNING)
```

Alla tänkbara relationer skall därmed ha fått varsitt avsnitt i programmet, och återgång görs alltid till rad 560 som utgör startpunkten för relationsundersökningarna.

Om programmet av någon anledning spårar ur eller om det trots allt skulle dyka upp en relation som vi inte tagit hänsyn till går programgenomloppet till avslutningssekvensen.

```
1280 REM .  ........................  <Otroligt fel!>
1290 S¤="ICKE FÖRUTSETT FEL HAR INTRÄFFAT!"
1300 GOSUB 2000 : REM =(LARMTEXT UT)===
1310 REM
1320 S¤="VI BRYTER OCH AVSLUTAR BEARBETNINGEN."
1330 GOSUB 2000 : REM =(LARMTEXT UT)===
```

PROGRAMMETS AVSLUTNING

Programkörningen avslutas normalt när strängen "SLUT" inmatas som namnterm. Test av namntermen sker på rad 570. I avslutningssekvensen skapas bl a en ny uppdaterad originalfil med samma namn som utgångsfilen.

```
1340 REM .  _____ <AVSLUTNING>
1350 ;
1360 ; "SLUTKOPERING TILL ORIGINALFIL"
1370 ; "============================"
1380 ;
1390 ; "   Förberedelser.... START"
1400 ONERRORGOTO 1450 : REM test filslut #1
1410 PRINT #2,A1¤
1420 PRINT #2,A2¤
1430 INPUT #1,A1¤,A2¤
1440 GOTO 1400
1450 CLOSE 1
1460 CLOSE 2
1470 ; "                        KLART" : ;
1480 ; "   Filöppningar..... START"
1490 OPEN F2¤ ASFILE 2
1500 PREPARE F¤ ASFILE 3
1510 ; "                        KLART" : ;
1520 ; "   Slutkopiering.... START"
1530 ONERRORGOTO 1570
1540 INPUT #2,A¤
1550 PRINT #3,A¤
1560 GOTO 1530
1570 ; "                        KLART" : ;
1580 ; "   Filstängning..... START"
1590 CLOSE 2
1600 CLOSE 3
1610 ; "                        KLART" : ;
1620 ; "   Slaskfilborttagn. START"
1630 KILL F1¤
1640 KILL F2¤
1650 ; "                        KLART" : ;
1660 REM
1670 END
```

Slutsekvensen startar med att överföra resten av fil #1 till fil #2. Sedan vi stängt filerna öppnar vi fil #2 för läsning samt förbereder fil #3, som är vår gamla originalfil, för skrivning. Därefter flyttas innehållet från fil #2 till fil #3 och det kallar vi för slutkopiering. Som du ser av raderna 1530--1560 läser och skriver vi inte postvis utan termvis, vilket är möjligt att göra här eftersom både namn och telefonnummer ligger lagrade i strängform. Det allra sista, som sedan händer i programmet, är att båda slaskfilerna raderas. Eftersom de nyskapas varje gång vi behöver dem är det onödigt att de dessemellan ska ligga kvar och ta upp skivutrymme.

De redovisade programraderna 100--2240 utgör ett komplett körbart program som du nu bör ha lagrat på flexskivan under namnet "TELEFON6" Provkör programmet och begrunda!

MENYN KAN ANVÄNDAS

Programexemplet "TELEFON6" är inte ett fullständigt program för uppdatering. Fortfarande saknas funktionerna POST ÄNDRAS och POST UTGÅR. Jämfört med programmet "TELEFON4", som vi också använde för att komplettera en befintlig fil, har programmet "TELEFON6" dock flera fördelar. En sådan fördel är att en från början ordnad fil även efter uppdateringen är ordnad.

Genom att tillfoga nedanstående två programrader i avslutningen av "TELEFON6" kan du via menyprogrammet "TELEFON" anropa programmet "TELEFON6".

```
1660 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
1670 CHAIN "TELEFON"
```

Det kan då också vara lämpligt att avlägsna de utskriftsrutiner som är inlagda i "TELEFON6" enbart för att underlätta förståelsen av uppdateringsproceduren. Radera bl a följande rader: 590, 610, 670, 690, 770, 790, 1010, 1030, 1140, 1160, 1740, 1750.

Den enda programrad som behöver ändras i menyprogrammet "TELEFON" är denna

```
360 CHAIN "TELEFON6"
```

När du lagrat de förändrade programmen på flexskiva kan programpaketet i sin helhet köras. Observera dock att filerna måste vara ordnade i bokstavsordning (ASCII-ordning) efter namntermerna.

Programexempel som visar varianterna POST ÄNDRAS och POST UTGÅR återfinner du i övningsboken ABC-UPPGIFTER I PROGRAMMERING.

10

DIREKTFILER PÅ FLEXSKIVA

VAD ÄR EN DIREKTFIL?

Som vi tidigare visat i kapitel 8 har sekventiella filer vissa begränsningar. En viss post i en sekventiell fil kan inte ändras eller avlägsnas utan att hela filen måste kopieras minst en gång. Detta beror ju på att poster i en sekventiell fil alltid måste läsas/skrivas från filens början.

I en direktfil däremot behöver inte posterna läsas/skrivas i någon viss ordning utan önskad post kan hanteras oberoende av dess plats i filen. Det är dessutom möjligt att på en öppnad direktfil både skriva och läsa poster omväxlande.

En direktfil definierar vi alltså som en logisk följd av poster på ett yttre minnesmedium och där posterna kan läsas/skrivas i godtycklig ordning. Direktfiler kallas ibland filer med "random access".

EN FIL ÄR INDELAD I BLOCK

En direktfil består i ABC 80 av ett antal block, vardera omfattande 253 tecken. I varje fil numreras blocken löpande med början från block nr 1 (block nr 0 används ej i vår framställning). Läsning och skrivning sker alltid blockvis och vid varje läsning/ skrivning anges blocknumret.

För att underlätta förståelsen för hur direktfiler fungerar låter vi i detta kapitel varje block innehålla endast en post. För att utnyttja utrymmet på flexskivan bättre kan i vissa fall flera poster lagras i varje block.

SKRIV BLOCK I EN DIREKTFIL

Som kommer att framgå av de följande programexemplen, kan direktfiler öppnas respektive stängas med samma instruktioner som tidigare användes för hantering av sekventiella filer. Vid läsning och skrivning av block i direktfiler är tekniken däremot helt annorlunda.

Vid skrivning av ett bestämt block måste följande moment utföras.

- Ange filens nummer
- Tilldela systemvariabeln Q0¤ värdet av det block som ska skrivas. Q0¤ består av maximalt 253 tecken
- Ange numret på det block där skrivning ska ske

Dessa moment motsvaras i ett program av följande rader

```
10 Z=CALL(28666,N) : REM Fil nr N utpekas
20 Q0¤="Nya innehållet i blocket"
30 Z=CALL(28670,B) : REM Block nr B utpekas
```

De följande programraderna innebär exempelvis att filen "DIREKT.DAT" öppnas som fil #1, posten "DATUM" skrivs i block nr 1, varefter filen stängs. Programraden 410 motsvarar nu de ovanstående programraderna 10 -- 30.

```
370 DIM Q0¤=253
390 PREPARE "DIREKT.DAT" ASFILE 1
410 Z=CALL(28666,1) : Q0¤="DATUM" : Z=CALL(28670,1)
430 CLOSE 1
```

Som du vet skulle motsvarande programavsnitt som hanterar en sekventiell fil #2 se ut på följande sätt:

```
390 PREPARE "SEKVENS.DAT" ASFILE 2
410 PRINT #2,"DATUM"
430 CLOSE 2
```

ANROPA SUBRUTINER

Att skrivning på en sekventiell fil ser enklare ut i programmet beror på att man i detta fall kan använda en enda instruktion, en variant av instruktionen PRINT. I fallet direktfil saknas motsvarande instruktion i BASIC, och ett programavsnitt bestående av tre instruktioner måste utföras vid varje skrivning av ett block på filen.

I programavsnittet ingår instruktionen CALL (... , ...) som innebär att subrutiner i maskinspråk med angivna startadresser anropas. För att kunna använda direktfiler behöver du inte känna till innebörden av instruktionen CALL (... , ...). Det är fullt tillräckligt att du lär dig att vid skrivning använda följande satser

Sats	Förklaring
Z = CALL (28666, N)	Denna sats används för att välja önskad fil, fil nummer N.
Q0¤ = C¤	De tecken som vid skrivning överförs till flexskiva är alltid värdet av Q0¤ . Ingen annan variabel än Q0¤ kan användas för detta ändamål. Här tilldelas Q0¤ önskat värde, d v s värdet av variabeln C¤.
Z = CALL (28670, B)	Denna sats används för att peka ut önskat block i filen, nämligen block nummer B, samt för att utföra skrivningen.

En komplett programrad som innehåller alla satserna och utför skrivningen ser således ut på detta sätt

```
100 Z=CALL(28666,N) : Q0¤=C¤ : Z=CALL(28670,B)
```

Värdet av variabeln Z används inte, men här måste en variabel ingå i de satser där funktionen CALL (... , ...) används.

RESERVERA UTRYMME PÅ FLEXSKIVAN

För att kunna läsa och skriva poster i godtyckliga block på en direktfil måste filen först skapas. Detta innebär bl a att önskat utrymme för filen reserveras på flexskivan.

Plats reserveras för filen genom att alla block i filen skrivs i tur och ordning. Den information som man skriver på filen är i detta fall ointressant. Vi väljer därför att för enkelhetens skull skriva 253 st mellanrum (space) i varje block. I följande programexempel tilldelas filen "TELKAT2.DAT" blocken nr 0--10 (som tidigare nämnts används fortsättningsvis inte block nr 0). Lagra detta program med namnet PREP10 på flexskivan.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = PREP10 "
120 ; "**************************"
130 ; "PREPARERING AV DIREKTFIL"
140 ; "**************************"
150 ;
160 REM . PREPARERING AV FILEN
170 REM . "TELKAT2.DAT" MED
180 REM . 10 BLOCK OM VARDERA
190 REM . 253 TECKEN (SPACE)
200 REM
210 REM
220 REM STARTADRESSER I MINNET FÖR FUNKTIONERNA
230 REM . Val av önskad fil ( V )
240 REM . Skrivning på fil  ( S )
250 REM
260 REM
270 REM STRÄNG SOM ÖVERFÖRS TILL FIL
280 REM . Q0¤ - Sträng med 253 tecken
290 REM
300 REM VARIABEL FÖR SUBRUTINANROP (CALL)
310 REM . Z - Värdet av Z används ej!
320 REM
330 REM ------------------------------
340 REM
350 DIM Q0¤=253 : REM Blocklängd
360 V=28666 : S=28670
370 PREPARE "TELKAT2.DAT" ASFILE 1
380 FOR B=0 TO 10 : REM . Block 0 används ej i forts.
390 Z=CALL(V,1) : Q0¤=SPACE¤(253) : Z=CALL(S,B)
400 NEXT B
410 CLOSE 1
420 ; "PREPARERINGEN AV FILEN ÄR KLAR !"
430 ; "FILEN INNEHÅLLER 10 BLOCK"
440 END
```

För att programmen ska bli lättlästa har CALL-funktionernas startadresser 28666 resp 28670 angivits med konstanterna V resp S.

Startadresser	Satser	Kommentarer
V = 28666	Z = CALL (V, 1)	Val av fil # 1
S = 28670	Z = CALL (S, 1)	Skrivning av block nr 1

ATT SKRIVA PÅ EN DIREKTFIL

När du kört programmet "PREP10" finns filen "TELKAT2.DAT", som omfattar 10 block, lagrad på flexskivan. Med nedanstående program, "SKRIV10", kan du nu skriva en post i varje block i godtycklig ordning.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = SKRIV10 "
120 ; "************************"
130 ; "SKRIVNING AV DATA PÅ"
140 ; "DIREKTFIL."
150 ; "************************"
160 ;
170 REM
180 REM . SKRIVNING AV TERMERNA
190 REM . NAMN   (A¤)
200 REM . NUMMER  (B¤)
210 REM . PÅ FILEN "TELKAT2.DAT"
220 REM
230 REM
240 REM STARTADRESSER I MINNET FÖR FUNKTIONERNA
250 REM . Val av önskad fil ( V )
260 REM . Skrivning på fil  ( S )
270 REM
280 REM
290 REM STRÄNG SOM ÖVERFÖRS TILL FIL
300 REM . Q0¤ - Sträng om 253 tecken
310 REM
320 REM VARIABEL FÖR SUBRUTINANROP (CALL)
330 REM . Z - Värdet av Z används ej!
340 REM
```

```
350 REM ------------------------------
360 REM
370 DIM Q0¤=253 : REM Blocklängd
380 V=28666 : S=28670
390 OPEN "TELKAT2.DAT" ASFILE 1
400 PRINT
410 PRINT "SKRIV:"
420 PRINT "BLOCKNUMMER (1-10)";
430 ONERRORGOTO 420
440 INPUT B
450 IF B<1 OR B>10 THEN GOTO 420
460 PRINT "NAMN/(SLUT)....";
470 INPUTLINE S¤
480 ;
490 S¤=LEFT¤(S¤,LEN(S¤)-2)
500 IF S¤="SLUT" THEN 630
510 GOSUB 710 : REM =(STRÄNGHANTERING)==
520 A¤=S¤ : REM Namn
530 PRINT "TELEFONNUMMER..";
540 INPUTLINE S¤
550 ;
560 S¤=LEFT¤(S¤,LEN(S¤)-2)
570 GOSUB 710 : REM =(STRÄNGHANTERING)==
580 B¤=S¤ : REM Nummer
590 C¤=A¤+B¤ : REM Hel post, 2*16 tecken
600 REM Skriv  block nr B på direktfilen #1
610 Z=CALL(V,1) : Q0¤=C¤ : Z=CALL(S,B)
620 GOTO 400
630 CLOSE 1
640 ;
650 ; "SKRIVNINGEN PÅ DIREKTFIL ÄR KLAR"
660 END
670 REM ********************************
680 REM .           SUBRUTIN
690 REM ********************************
700 REM
710 REM ======= STRÄNGHANTERING =======
720 REM
730 REM . Strängen utökas till 16 tecken
740 IF LEN(S¤)<16 THEN S¤=S¤+SPACE¤(16-LEN(S¤))
750 REM . För lång sträng kortas av
760 S¤=LEFT¤(S¤,16)
770 RETURN
780 REM ********************************
```

Filen "TELKAT2.DAT" öppnas när nedanstående programrad utförs.

```
390 OPEN "TELKAT2.DAT" ASFILE 1
```

Observera att direktfilen nu är öppen för både läsning och skrivning av block! Programmet illustrerar möjligheten att skriva block i godtycklig ordning. Du får själv välja ut det block där inmatad post ska skrivas enligt raderna 400--450.

Inmatningen av termerna sker med instruktionen INPUTLINE ... (jfr kap 3, sid 34). De två termerna A¤ och B¤ sammanfogas till en post, C¤ , på raden 590. Därefter skrivs posten på angivet block med programraden

```
610 Z=CALL(V,1) : Q0¤=C¤ : Z=CALL(S,B)
```

ALLA POSTER HAR SAMMA LÄNGD

I föregående programexempel får alla termer längden 16 tecken och följaktligen alla poster längden 32 tecken. Detta ombesörjs av subrutinen "stränghantering". Eftersom en post, 32 tecken, endast utgör en del av ett block, 253 tecken, underlättas en kommande läsning av att alla poster har samma längd. Det är inget krav att poster på en direktfil ska ha samma längd, men det förenklar stränghanteringen, "upp-packningen" av inlästa block.

LÄS BLOCK I EN DIREKTFIL

Liksom vid skrivning av ett block på en direktfil sker läsning av ett block med hjälp av instruktionen CALL (... , ...). Sammanställningen på nästa sida gäller läsning.

Sats	Förklaring
Z = CALL (28666, N)	Denna sats används för att välja önskad fil, fil nummer N.
Z = CALL (28668, B)	Denna sats används för att peka ut önskat block i filen, d v s block nr B, samt för att utföra läsning. Q0¤ tilldelas värdet av det lästa blocket.
C¤ = Q0¤	Önskad variabel t ex C¤ tilldelas värdet av systemvariabeln Q0¤.

En fullständig programrad för läsning kan således utformas på detta sätt.

```
100 Z=CALL(28666,N) : Z=CALL(28668,B) : C¤=Q0¤
```

LÄS FILENS POSTER

Med hjälp av programmet "LÄS10" kan datorn läsa alla block på filen "TELKAT2,DAT" och presentera posterna på bildskärmen.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LÄS10 "
120 ; "**************************"
130 ; "LÄSNING AV DATA FRÅN"
140 ; "DIREKTFIL."
150 ; "**************************"
160 ;
170 REM . LÄSNING GÖRS AV POSTER TILL C¤
180 REM . FRÅN FILEN "TELKAT2.DAT"
190 REM . POST = NAMN + NUMMER (32 TECKEN)
200 REM
210 REM STARTADRESSER I MINNET FÖR FUNKTIONERNA
220 REM . Val av önskad fil ( V )
230 REM . Läsning från fil  ( L )
240 REM
250 REM STRÄNG SOM ÖVERFÖRS FRÅN FIL
260 REM . Q0¤ - Sträng om 253 tecken
270 REM
280 REM VARIABEL FÖR SUBRUTINANROP (CALL)
290 REM . Z - Värdet av Z används ej!
```

(forts)

```
300 REM --------------------------------
310 REM
320 DIM Q0¤=253 : REM Blocklängd
330 DIM C¤=253 : REM Blocklängd
340 V=28666 : L=28668
350 ; "POST";TAB(7)"NAMN";TAB(23)"TELEFON"
360 ;
370 OPEN "TELKAT2.DAT" ASFILE 1
380 FOR B=1 TO 10
390 REM Läs block nummer B på direktfil
400 Z=CALL(V,1) : Z=CALL(L,B) : C¤=Q0¤
410 C¤=LEFT¤(C¤,32)
420 PRINT B;TAB(7)C¤
430 NEXT B
440 CLOSE 1
450 PRINT : PRINT
460 ; "LÄSNINGEN AV DIREKTFILEN ÄR KLAR"
470 END
```

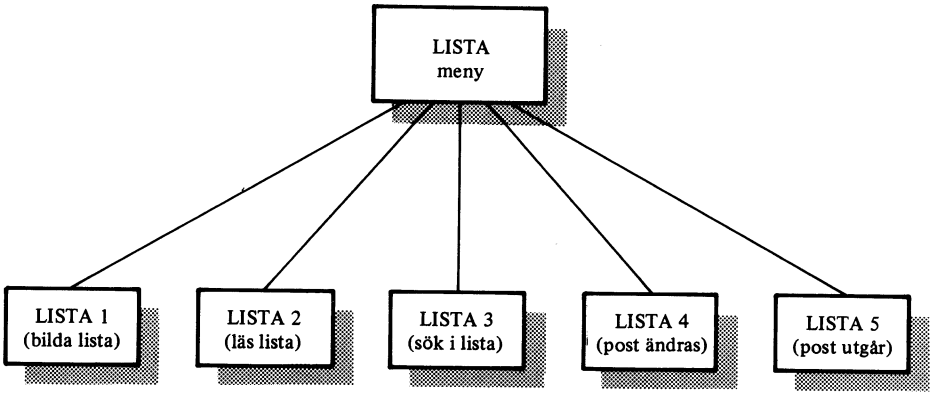
Alla de tio blocken i filen läses successivt i en loop på raderna 380--440. Läsningen av ett helt block sker när denna programrad utförs.

```
400 Z=CALL(V,1) : Z=CALL(L,B) : C¤=Q0¤
```

Efter läsning återfinns blockets hela innehåll i variabeln C¤. Den aktuella posten utgörs av blockets 32 första tecken som blir det nya innehållet i variabeln C¤ enligt raden 410. Posten presenteras sedan på bildskärmen när rad 420 utförs.

MENYN SOM PROGRAMMERINGSHJÄLPMEDEL

Vi ska fortsättningsvis presentera några programexempel som oberoende av varandra kan bearbeta direktfilen "TELKAT2.DAT". Det vore fullt möjligt att sammanfoga alla dessa exempel till ett enda stort program. Ett sådant skulle dock bli svårhanterligt och oöverskådligt. Vi väljer därför att utföra de olika funktionerna med hjälp av fristående program och programvalet sker från menyprogrammet "LISTA" enligt figuren presenterad på nästa sida.



Lagra detta program på flexskiva med namnet "LISTA".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LISTA"
120 ;
130 ; "**************************"
140 ; "        Meny för"
150 ; " H a n t e r i n g    a v"
160 ; "        DIREKTFIL"
170 ; "**************************"
180 ;
190 ONERRORGOTO 100
200 PRINT
210 PRINT
220 PRINT "1 = BILDA   NY  TELEFONLISTA"
230 PRINT "2 = SKRIV   UT  TELEFONLISTA"
240 PRINT "3 = SÖKNING  I  TELEFONLISTA"
250 PRINT "4 = UPPDATERING. POST ÄNDRAS"
260 PRINT "5 = UPPDATERING.  POST UTGÅR"
270 PRINT "6 = AVSLUTA    BEARBETNINGEN"
280 PRINT : PRINT
290 PRINT "Välj alternativ:";
300 INPUT S
310 IF S<1 OR S>6 THEN 100
320 ON S GOTO 330,340,350,360,370,380
330 CHAIN "LISTA1"
340 CHAIN "LISTA2"
350 CHAIN "LISTA3"
360 CHAIN "LISTA4"
370 CHAIN "LISTA5"
380 PRINT "KÖRNINGEN AVSLUTAD !"
390 END
```

MODIFIERA SKRIVPROGRAMMET

Vi förutsätter fortsättningsvis att filen "TELKAT2.DAT" finns tillgänglig på flexskiva skapad med programmet "PREP10". För att lägga upp en ny telefonlista kan givetvis programmet "SKRIV10" användas. Inmatningen underlättas dock om blocknumren uppräknas automatiskt för varje ny post. Vi ska också införa en kontroll som innebär att blocknumret inte överstiger 10. De programrader som behöver tillkomma i exemplet "SKRIV10" för att åstadkomma detta är:

```
110 ; "Programnamn = LISTA1"
372 B=1 : REM .                 Första blocknummer
612 B=B+1 : REM .               Blocknr inkrementeras
614 IF B>10 THEN 622 : REM . För stort blocknummer
622 PRINT : PRINT "ALLA POSTER PÅ FILEN ÄR SKRIVNA"
652 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
660 CHAIN "LISTA"
```

(Raderna 410--450 raderas)

Här ingår också de rader som behövs för att programmet ska bli ett underprogram till menyprogrammet "LISTA". Lagra det modifierade programmet under namnet "LISTA1".

ANPASSA LÄSPROGRAMMET

För att anpassa vårt tidigare program "LÄS10" som läser direktfilen "TELKAT2.DAT" behöver du bara tillfoga nedanstående programrader.

```
110 ; "Programnamn = LISTA2"
462 ; "ÅTER TILL MENYN (RETURN)"; : GET S¤
470 CHAIN "LISTA"
```

Lagra det kompletterade programmet under namnet "LISTA2".

ATT SÖKA PÅ DIREKTFIL

Sökbegreppet som används i detta avsnitt är personnamn. Det är tillräckligt att ange några av namnets första bokstäver. Datorn söker fram alla poster i registret vars namn inleds med de angivna bokstäverna. Eftersom man vill ha resultatet av sökningen i form av namn och telefonnummer räcker det att presentera hela den sökta posten på bildskärmen (jfr avsnittet SÖK I TELEFONLISTAN, kap 8). Lagra programmet under namnet "LISTA3".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LISTA3"
120 ; "************************"
130 ; "  SÖKNING AV DATA PÅ"
140 ; "  EN DIREKTFIL."
150 ; "************************"
160 ;
170 ; " ENDAST NAMN-SÖKNING"
180 ;
190 REM ------------------------------
200 DIM Q0¤=253 : REM Sträng med 1 block
210 DIM C¤=253 : REM .Sträng med 1 post
220 V=28666 : L=28668 : S=28670
230 ONERRORGOTO 400 : REM test filfel
240 OPEN "TELKAT2.DAT" ASFILE 1
250 PRINT "VILKET NAMN SÖKS ......";
260 INPUTLINE S¤
270 ;
280 S¤=LEFT¤(S¤,LEN(S¤)-2)
290 FOR B=1 TO 10 : REM 10 Block
300 Z=CALL(V,1) : Z=CALL(L,B) : C¤=Q0¤
310 C¤=LEFT¤(C¤,32)
320 IF LEFT¤(C¤,LEN(S¤))=S¤ THEN PRINT : PRINT C¤
330 NEXT B
340 CLOSE 1
350 PRINT : PRINT
360 PRINT "NY SÖKNING (J)"; : INPUT S¤
370 IF LEN(S¤)=0 THEN 100
380 S¤=LEFT¤(S¤,1)
390 IF S¤<>"N" AND S¤<>"n" THEN 100
400 IF ERRCODE<>-1 THEN PRINT "FILFEL!" : PRINT
410 PRINT "SÖKNINGEN I  FILEN KLAR!"
420 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
430 CHAIN "LISTA"
```

Läsningen av filens poster sker i en loop på raderna 290--330. Utskrift av sökt post sker med programraden 320 om angivet sökvillkor är uppfyllt.

UPPDATERING – POST ÄNDRAS

De moment som ingår i uppdatering av en fil är som vi tidigare nämnt följande

- placera nya poster på rätt plats i filen (post tillkommer)
- ändra befintliga poster på filen (post ändras)
- avlägsna icke önskade poster från filen (post utgår)

Alternativet "post tillkommer" finns redovisat i kap 9 för en ordnad sekventiell fil. Programmet för denna funktion kan utformas på motsvarande sätt för en direktfil.

Vi ska här endast redovisa två programexemel som realiserar funktionerna "post ändras" respektive "post utgår" för en direktfil. I dessa programexempel behöver den fil som bearbetas inte vara ordnad. De enda krav som ställs på filen är de som gällt tidigare för filens organisation d v s att varje block innehåller 1 post och varje post utgörs av namn och telefonnummer. Postens totala längd ska vara 32 tecken.

Studera programexemplet och observera fördelen med att kunna läsa och skriva block på filen i godtycklig ordning.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LISTA4"
120 ; "**************************"
130 ; "  UPPDATERING AV EN"
140 ; "     DIREKTFIL."
150 ; "**************************"
160 ;
170 ; "INNEHÅLLER ENDAST "
180 ; "   POST ÄNDRAS   "
190 ;
200 REM
210 REM VARIABEL FÖR FALLET POST SAKNAS
220 REM . P - Värdet 0 om POST SAKNAS
230 REM . P - Värdet 1 om POST FINNS
240 REM
250 REM ------------------------------
```

```
260 DIM Q0¤=253 : REM Sträng med 1 block
270 DIM C¤=253 : REM .Sträng med 1 post
280 P=0 : REM Nollställn. av variabel för POST SAKNAS
290 V=28666 : L=28668 : S=28670
300 ONERRORGOTO 490 : REM test filfel
310 OPEN "TELKAT2.DAT" ASFILE 1
320 PRINT "VILKET NAMN "
330 PRINT "SKA ÄNDRAS (SLUT)";
340 INPUTLINE S¤
350 PRINT : PRINT : PRINT
360 S1¤=LEFT¤(S¤,LEN(S¤)-2)
370 IF S1¤="SLUT" OR S1¤="" THEN 470
380 FOR B=1 TO 10 : REM 10 Block
390 REM Läs ett block från filen
400 Z=CALL(V,1) : Z=CALL(L,B) : C¤=Q0¤
410 C¤=LEFT¤(C¤,32)
420 IF LEFT¤(C¤,LEN(S1¤))=S1¤ THEN GOSUB 570
430 NEXT B
440 IF P=0 THEN PRINT "POST SAKNAS! KAN EJ ÄNDRAS"
450 IF P=0 THEN PRINT : GOTO 320
460 CLOSE 1 : GOTO 100
470 CLOSE 1
480 PRINT "UPPDATERINGEN AV FILEN KLAR!"
490 IF ERRCODE<>-1 THEN PRINT "FILFEL!" : CLOSE 1
500 PRINT "ÅTER TILL MENYN (RETURN)"; : GET S¤
510 CHAIN "LISTA"
520 REM
530 REM ********************************
540 REM .          SUBRUTINER
550 REM ********************************
560 REM
570 REM ========= POST ÄNDRAS =========
580 REM
590 PRINT : PRINT
600 P=1 : REM . Sökt post finns i filen
610 ; "I REGISTRET FINNS DENNA POST SOM"
620 ; "ÖVERENSSTÄMMER MED SÖKORDET"
630 ;
640 PRINT C¤
650 ;
660 PRINT "SKA DENNA POST ÄNDRAS (J)"; : INPUT S¤
670 IF LEN(S¤)=0 THEN 710
680 S¤=LEFT¤(S¤,1)
690 IF S¤<>"N" AND S¤<>"n" THEN 710 ELSE 720
700 GOTO 720
710 GOSUB 740 : REM =(POST INMATAS)=======
720 RETURN
730 REM
```

(forts)

```
740 REM ========= POST INMATAS ========
750 REM
760 REM
770 OPEN "TELKAT2.DAT" ASFILE 1
780 PRINT
790 PRINT "SKRIV:"
800 PRINT "NAMN..........";
810 INPUTLINE S¤
820 ;
830 S¤=LEFT¤(S¤,LEN(S¤)-2)
840 GOSUB 970 : REM =(STRÄNGHANTERING)==
850 A¤=S¤
860 PRINT "TELEFONNUMMER..";
870 INPUTLINE S¤
880 ;
890 S¤=LEFT¤(S¤,LEN(S¤)-2)
900 GOSUB 970 : REM =(STRÄNGHANTERING)==
910 B¤=S¤
920 C¤=A¤+B¤ : REM Hel post, 2*16 tecken
930 REM Skriv på direktfil
940 Z=CALL(V,1) : QO¤=C¤ : Z=CALL(S,B)
950 RETURN
960 REM
970 REM ======= STRÄNGHANTERING =======
980 REM
990 REM . Strängen utökas till 16 tecken
1000 IF LEN(S¤)<16 THEN S¤=S¤+SPACE¤(16-LEN(S¤))
1010 REM . För lång sträng kortas av
1020 S¤=LEFT¤(S¤,16)
1030 RETURN
1040 REM ********************************
```

Programmets första del, raderna 260--460, innehåller en rutin för att ta emot inmatad namnterm samt söka efter motsvarande post på filen (jfr programmet LISTA3). Om den sökta namntermen återfinns i filen anropas subrutinen "POST ÄNDRAS", där hela den lästa posten presenteras. Användaren får här möjlighet att kontrollera om den lästa posten är den rätta. Om den lästa posten ska ändras anropas subrutinen "POST INMATAS" där ny post inmatas och skrivs på filen (jfr programmet SKRIV10).

Det fallet kan ju också inträffa att den sökta posten inte finns i filen. I programmet används variabeln P för att markera detta fall. Inför varje sökning tilldelas variabeln P värdet 0. Endast om sökt namnterm återfinns kan detta värde ändras och det sker på raden 600, där variabeln P ges värdet 1. Om variabeln P efter slutförd läsning av filen fortfarande har värdet 0 erhålls utskriften "POST SAKNAS! KAN EJ ÄNDRAS" enligt raden 440.

UPPDATERING – POST UTGÅR

Endast några få förändringar behöver utföras i programexemplet för "POST ÄNDRAS" om i stället funktionen "POST UTGÅR" ska åstadkommas. I stället för att skriva en ny post på filen skrivs nu bara ett tomt block. Det sker när den nedanstående programraden 710 utförs. Övriga programrader utgör endast förändringar av ledtexter och REM-satser.

Utför nedanstående ändringar i programmet "LISTA4" och lagra det nya programmet under namnet "LISTA5".

```
110 ; "Programnamn = LISTA5"
180 ; "  POST UTGÅR"
330 PRINT "SKA UTGÅ (SLUT)";
440 IF P=0 THEN PRINT "POST SAKNAS! KAN EJ UTGÅ"
570 REM ========= POST UTGÅR ==========
660 PRINT "SKA DENNA POST UTGÅ (J)"; : INPUT S¤
702 REM Skriv ett tomt block
710 Z=CALL(V,1) : Q0¤=SPACE¤(253) : Z=CALL(S,B)
```

(raderna 730--1040 kan raderas).

INFORMATION OM FILENS LÄNGD

I våra enkla programexempel har vi genomgående använt en enda fil omfattande 10 block. Vanligtvis innehåller ett programsystem däremot ett flertal filer med olika längder. Information om längden hos en direktfil är därför nödvändig för att kunna undvika läsning/skrivning utanför filen. Denna information behöver inte finnas i programmet utan lagras med fördel på flexskivan, t ex i filens block nr 0.

DATAFILER PÅ SEPARAT FLEXSKIVA

Vi har hittills lagrat programfiler och datafiler på samma flexskiva. Ett bättre utnyttjande av datorsystemet är att använda olika drivenheter för lagring av program- respektive datafiler.

Anledningen är att stora datamängder kan hanteras av ett enda program, om möjlighet finns att successivt byta de flexskivor där datafilerna lagras. Alla datafiler kan placeras på den ena flexskivan genom att enhetstyp anges i programmet tillsammans med filnamnet, t ex

```
370 PREPARE "DR1:TELKAT2.DAT" ASFILE 1
```

FILER PÅ KASSETTBAND

SEKVENTIELLA FILER PÅ BAND

Vi har tidigare definierat en sekventiell fil som en logisk följd av poster lagrade på ett yttre minnesmedium (kap 8). På ett kassettband ligger alltid posterna lagrade i tur och ordning. För att kunna läsa en post mitt i filen måste därför först alla framförliggande poster läsas. På motsvarande sätt måste alla poster skrivas på filen i tur och ordning. På kassettband kan följaktligen enbart sekventiella filer förekomma.

LAGRA KONSTANTER PÅ KASSETTFIL

Hantering av sekventiella filer på kassettband skiljer sig i enklaste fallet inte nämnvärt ifrån motsvarande filhantering på flexskiva. Programexemplen "SKRIV8A" och "LÄS8A" kan därför användas även med kassettenheten som yttre minne, förutsatt att flexskivenhet inte är inkopplad. Genom att filnamnen utökas med enhetstyp kommer rätt enhet alltid att adresseras vid programkörningen, även om både flexskivenhet och kassettenhet är samtidigt inkopplade.

För att underlätta handhavandet av kassettenheten förser vi programmet med PRINT-satser som innehåller lämpliga ledtexter. I de följande två programexemplen har dessa förändringar införts.

Lagra nedanstående program på kassett under namnet "SKRIV11A" och placera en ny kassett i bandspelaren före programkörning.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = SKRIV11A"
120 ; "**************************"
130 ; "  SKRIVNING  AV DATA PÅ"
140 ; "  SEKVENTIELL (CAS:) FIL"
150 ; "**************************"
160 ;
170 REM
180 REM .  SKRIVNING AV ETT ANTAL
190 REM .  FASTA TERMER PÅ FILEN
200 REM .  " CAS : TELKAT3.DAT "
210 REM .
220 REM ----------------------------
```

(forts)

```
230 REM .  starta bandsp. för inspelning
240 REM
250 ; "SPOLA TILLBAKA BANDET!" : PRINT
260 ; "NOLLSTÄLL RÄKNEVERKET " : PRINT
270 ; "TRYCK NED TANGENTERNA 'PLAY' OCH 'REC'" : PRINT
280 ; "NÄR DU ÄR FÄRDIG TRYCK PÅ 'RETURN'"; : GET S¤
290 ;
300 ;
310 ; "VÄNTA!"
320 PREPARE "CAS:TELKAT3.DAT" ASFILE 1
330 REM
340 PRINT #1,"JONAS"
350 PRINT #1,"33451"
360 PRINT #1,"ANDERS"
370 PRINT #1,"12345"
380 PRINT #1,"CECILIA"
390 PRINT #1,"23456"
400 CLOSE 1
410 PRINT : PRINT
420 PRINT "SKRIVNINGEN AVSLUTAD!"
430 END
```

Filen skapas när raderna 320--400 utföres. Under förutsättning att tangenterna "PLAY" och "REC" är nedtryckta utförs två moment när nedanstående programrad genomlöps.

```
320 PREPARE "CAS:TELKAT3.DAT" ASFILE 1
```

De aktuella momenten är:

- kassettenhetens motor startas
- programnamnet (TELKAT3.DAT) skrivs på bandet

Skrivningen av de 6 konstanterna sker med hjälp av raderna

```
340 PRINT #1,"JONAS"
350 PRINT #1,"33451"
360 PRINT #1,"ANDERS"
370 PRINT #1,"12345"
380 PRINT #1,"CECILIA"
390 PRINT #1,"23456"
```

En instruktion för att skriva på en sekventiell fil innehåller på vanligt sätt PRINT, ett nummertecken (#), filnummer och kommatecken. Observera att PRINT-satsen får innehålla endast en variabel eller en konstant term efter kommatecknet.

Raden 340 innebär således att strängen "JONAS" skrivs på den angivna filen #1 som i vårt fall har namnet "TELKAT3.DAT". Filen #1 stängs avslutningsvis med hjälp av raden.

```
400 CLOSE 1
```

som bland annat innebär att filslutsmärke skrivs på filen och bandspelarens motor stoppas.

Som du ser är programavsnittet för att skapa en sekventiell fil på kassettband identiskt med motsvarande program för att skapa filen på flexskiva.

ATT LÄSA EN SEKVENTIELL FIL

Att läsa en sekventiell fil på kassettband tillgår på samma sätt som att läsa en motsvarande fil på flexskiva. Vi kan därför använda programmet "LÄS8A" kompletterat med lämpliga ledtexter och tillägg av enhetstyp i filnamnet.

Lagra programmet med namnet "LÄS11A" på programkassetten och byt sedan till kassetten som innehåller datafilen "TELKAT3.DAT".

```
100 PRINT CHR¤(12)
110 ; "Programnamn = LÄS11A"
120 ; "************************"
130 ; "  LÄSNING  AV DATA FRÅN"
140 ; "  SEKVENTIELL (CAS:) FIL"
150 ; "************************"
160 ;
170 REM
180 REM . LÄSNING GÖRS AV TERMERNA
190 REM . NAMN    (A¤),(C¤),(E¤)
200 REM . NUMMER  (B¤),(D¤),(F¤)
210 REM . FRÅN FILEN "CAS : TELKAT3.DAT"
220 REM .
230 REM . RESULTATET PRESENTERAS PÅ
240 REM . BILDSKÄRMEN.
250 REM
260 REM ------------------------------
270 REM . starta bandsp. för avspelning
280 REM
290 ; "SPOLA TILLBAKA BANDET!" : PRINT
300 ; "NOLLSTÄLL RÄKNEVERKET " : PRINT
310 ; "TRYCK NED TANGENTEN 'PLAY'" : PRINT
320 ; "NÄR DU ÄR FÄRDIG TRYCK PÅ 'RETURN'"; : GET S¤
330 ;
340 ;
350 ; "VÄNTA!"
360 REM ------------------------------
370 REM
380 OPEN "CAS:TELKAT3.DAT" ASFILE 1
390 INPUT #1,A¤
400 INPUT #1,B¤
410 INPUT #1,C¤
420 INPUT #1,D¤
430 INPUT #1,E¤
440 INPUT #1,F¤
450 CLOSE 1
460 PRINT CHR¤(12)
470 PRINT "UTSKRIFT AV FILEN TELKAT3.DAT"
480 PRINT "=============================="
490 PRINT
500 ; A¤,B¤
510 ; C¤,D¤
520 ; E¤,F¤
530 PRINT : PRINT
540 PRINT "BEARBETNINGEN  KLAR!"
550 END
```

Om bandspelaren är inställd för avspelning när raden

```
380 OPEN "CAS:TELKAT3.DAT" ASFILE 1
```

utförs, söker datorn på bandet efter en fil med namnet "TELKAT3.DAT". Om filen återfinns öppnas den för läsning och tilldelas filnumret 1.

Innehållet i filen överförs vid läsningen till variablerna A¤ , B¤ , ... F¤ när dessa programrader utförs:

```
390 INPUT #1,A¤
400 INPUT #1,B¤
410 INPUT #1,C¤
420 INPUT #1,D¤
430 INPUT #1,E¤
440 INPUT #1,F¤
```

En sats som innebär läsning av en term innehåller alltid INPUT eller INPUTLINE följt av ett nummertecken (#), filnummer, kommatecken och variabelnamn.

Läsningen avslutas med att filen stängs på vanligt sätt enligt programraden

```
450 CLOSE 1
```

Enligt raderna 500--520 presenteras därefter filens innehåll på bildskärmen.

KASSETTENHETEN – FLEXSKIVENHETEN

Skrivning och läsning på ett kassettband kan inte ske samtidigt. Enbart en fil kan läsas/skrivas åt gången. Dessa egenskaper hos kassettbandfiler medför att filhantering på det sätt som vi beskrivit i kap 8 och 9 inte kan förekomma, när kassettenheten är datorsystemets enda yttre minne.

Sådana datafiler som kräver uppdatering bör lagras på band enbart om filens hela innehåll kan rymmas i datorns internminne vid uppdateringen. De program som behövs för de olika uppdateringsfunktionerna blir i dessa fall enkla, men filens omfattning begränsas naturligtvis av internminnets storlek.

Filer som enbart innehåller konstanter som inte behöver ändras, t ex en serie mätvärden, kan hanteras av datorn, även om inte alla konstanterna vid körningen samtidigt ryms i internminnet. Statistisk bearbetning av en serie mätvärden är en sådan tillämpning.

I de följande två programexemplen behandlas bearbetningen av en fil, vars hela innehåll ryms i internminnet.

SKAPA ETT ADRESSREGISTER

Vi har tidigare konstaterat att en fil lagrad på kassettband som man önskar uppdatera måste rymmas i datorns internminne. Det är lämpligt att filens termer då lagras i minnet som indexerade variabler och det är naturligast att index får utgöras av postnummer. När vi skapar en fil som ska användas på detta sätt inmatas termerna därför till ett antal fält i minnet. Dessa fält överförs därefter till kassettband när inmatningen är avslutad.

I tillämpningar av detta slag är det viktigt att varje variabel dimensioneras så att minnet utnyttjas på bästa sätt. Det utrymme som automatiskt reserveras för strängvariabler, nämligen plats för 80 tecken, är i de flesta fall onödigt stort.

Det programexempel som följer skapar 3 fält i minnet. Varje post innehåller termerna A¤ (I), B¤ (I) och C¤ (I) där I är postens löpnummer. Efter avslutad inmatning till de tre fälten överförs termerna till filen "VYKORT.DAT" på kassettband.

Lagra följande program under namnet "SKRIV11B" på en programkassett och placera därefter en datakassett i bandspelaren.

```
100 PRINT CHR¤(12)
110 ; "Programnamn = SKRIV11B"
120 ; "************************"
130 ; "  SKRIVNING  AV DATA PÅ"
140 ; "  SEKVENTIELL (CAS:) FIL"
150 ; "************************"
160 ;
170 REM .  INMATNING AV MAX 80
180 REM .  POSTER MED TERMERNA
190 REM .  NAMN           A¤(I)
200 REM .  ADRESS         B¤(I)
210 REM .  POSTADRESS     C¤(I)
220 REM .  TILL FILEN " CAS : VYKORT.DAT"
230 REM .
240 REM .  INMATNINGEN AVBRYTS NÄR
250 REM .  NAMNTERMEN GES VÄRDET "SLUT"
260 REM
270 REM ----------------------------
280 DIM A¤(81)=30,B¤(81)=30,C¤(81)=30
290 I=1 : REM . INDEX  STARTVÄRDE = 1
300 REM
310 REM
320 REM
330 ; "SKRIV POST Nr. ";I
340 ;
350 ; "NAMN (SLUT)       : ";
360 GOSUB 700 : A¤(I)=S¤
370 IF A¤(I)="SLUT" THEN 470
380 REM
390 ; "ADRESS            : ";
400 GOSUB 700 : B¤(I)=S¤
410 REM
420 ; "POSTADRESS        : ";
430 GOSUB 700 : C¤(I)=S¤
440 PRINT : PRINT
450 I=I+1
460 IF I<81 THEN 330
470 ; CHR¤(12)
480 IF A¤(I)="SLUT" THEN 500
490 ; "INGA YTTERLIGARE POSTER KAN MATAS IN" : PRINT
500 ; "SPOLA TILLBAKA BANDET!" : PRINT
510 ; "NOLLSTÄLL RÄKNEVERKET" : PRINT
520 ; "TRYCK NED TANGENTERNA 'PLAY' OCH 'REC'" : PRINT
530 ; "NÄR DU ÄR FÄRDIG TRYCK PÅ 'RETURN'" : GET S¤
540 ;
550 ;
```

forts.

```
560 ; "SKRIVNING PÅ FILEN ..... START"
570 REM
580 PREPARE "CAS:VYKORT.DAT" ASFILE 1
590 FOR K=1 TO I-1
600 PRINT #1,A¤(K)
610 PRINT #1,B¤(K)
620 PRINT #1,C¤(K)
630 NEXT K
640 CLOSE 1
650 REM
660 ; "                              KLART"
670 ; "Körningen klar !"
680 END
690 REM ******************************
700 REM .===== SUBRUTIN INMATNING =====
710 REM
720 INPUTLINE S¤
730 S¤=LEFT¤(S¤,LEN(S¤)-2)
740 IF LEN(S¤)>30 THEN S¤=LEFT¤(S¤,30)
750 PRINT
760 RETURN
770 REM ******************************
```

Dimensioneringen utförs enligt raden

```
280 DIM A¤(81)=30,B¤(81)=30,C¤(81)=30
```

Plats reserveras således för 81 poster som vardera innehåller termerna A¤ (I), B¤ (I) och C¤ (I). Varje term är dimensionerad till maximalt 31 tecken (jfr programmeringskortet).

Inmatningen av data till de tre fälten sker enligt raderna 330--460. På raden 460 kontrolleras att antalet inmatade poster inte överstiger 80. Programkörningen kan därför inte avbrytas på grund av att minnet blir fullt.

Det programavsnitt där skrivning utförs inleds med ledtexter för bandspelarens inställningar på raderna 500--530. En fil öppnas med namnet "CAS:VYKORT.DAT" varefter skrivningen sker term för term i en loop på raderna 590--630, varefter filen stängs på rad 640.

I subrutinen "inmatning" avlägsnas de två sista tecknen, d.v.s tecknen för vagnretur och radframmatning, i varje sträng. Därefter kortas varje sträng som är längre än 30 tecken så att maxlängden hos varje term blir 30 tecken.

UTÖKA ADRESSREGISTRET

En uppdatering av en fil på kassettband kan endast utföras om filens poster först överförs till variabler i datorns minne. När posterna finns som variabelvärden i minnet kan utökningen av registret lätt utföras utan kommunikation med filer på yttre minnen. Sedan de önskade förändringarna utförts skrivs den uppdaterade filen på kassettband.

Vi ska illustrera detta med ett programexempel som visar en enkel uppdatering i form av tillägg av poster i registret. Utöka programmet "SKRIV11B" med följande programrader och lagra det resulterande programmet på programkassetten under namnet "UPPDAT11". Placera därefter datakassetten innehållande filen "VYKORT.DAT" i bandspelaren och kör programmet.

```
110 ; "Programnamn = UPPDAT11"
130 ; "  KOMPLETTERING AV        "
170 REM .  LÄSNING AV POSTER MED
180 REM .  TERMERNA :
220 REM .  FRÅN FILEN " CAS : VYKORT.DAT"
240 REM .  KOMPLETTERINGEN AVBRYTS NÄR

300 ; "SPOLA TILLBAKA BANDET!" : PRINT
301 ; "NOLLSTÄLL RÄKNEVERKET" : PRINT
302 ; "TRYCK NED TANGENTEN 'PLAY'" : PRINT
303 ; "NÄR DU ÄR FÄRDIG TRYCK PÅ 'RETURN'" : GET S¤
304 ;
305 ; "LÄSNING FRÅN FILEN ....... START"
306 OPEN "CAS:VYKORT.DAT" ASFILE 1
307 ONERRORGOTO 312
308 GOSUB 780 : A¤(I)=S¤
309 GOSUB 780 : B¤(I)=S¤
310 GOSUB 780 : C¤(I)=S¤
311 I=I+1 : GOTO 308
312 CLOSE 1
313 ; CHR¤(12)
314 ; "UTSKRIFT AV FILENS INNEHÅLL"
315 ; "=========================="
316 FOR K=1 TO I-1
317 ;
318 PRINT A¤(K) : PRINT B¤(K) : PRINT C¤(K)
319 ;
320 ; "TRYCK PÅ 'RETURN' FÖR NÄSTA POST" : GET S¤
321 NEXT K
322 ;
323 ;
324 ; "INMATNING AV NYA POSTER"
325 ;
```

```
770 REM
780 REM .==== SUBRUTIN LÄS FIL ========
790 REM
800 INPUTLINE #1,S¤
810 S¤=LEFT¤(S¤,LEN(S¤)-2)
820 RETURN
830 REM ******************************
```

Det kompletta programmet består av läsning av filen "VYKORT.DAT", inmatning av tillkommande poster samt skrivning av den kompletterade filen på kassettband.

Läsning från filen "VYKORT.DAT" sker på raderna 306--312. Läsning utförs term för term med hjälp av subrutinen enligt raderna 780--820 till dess att filslutsmärke påträffas (jfr kap 8, sid 109). Därefter presenteras lästa poster på bildskärmen.

Inmatningen av nya poster kan sedan utföras med programavsnittet 330 – 460. Efter avslutad komplettering skrivs den nya listan på kassettband enligt raderna 500 -- 640.

ATT LÄSA EN FIL I ETAPPER

Den metod som vi hittills använt vid läsning/skrivning av kassettfiler har förutsatt att filens hela innehåll ryms i datorns internminne. Vi har dessutom förutsatt att datorn förbrukar/skapar data i en snabbare takt än överföring av data sker från/till kassettbandet.

Om dessa förutsättningar inte är uppfyllda krävs andra metoder för hantering av filer på kassettband. Det är t ex möjligt att under läsning/skrivning av en fil stoppa och starta bandspelarmotorn. Överföring av data kan dock bara ske när kassettbandet rör sig med konstant hastighet. För att kunna stoppa/starta motorn i kassettbandspelaren under läsning/skrivning måste kassettbandet innehålla avsnitt där inga data finns lagrade.

Uppenbarligen måste kassettbandet förses med pauser på lämpliga ställen redan när filen skapas. Ett praktiskt sätt att utföra detta är att placera en paus efter varje block. I ABC 80 sker läsning och skrivning av filer alltid i block om 253 tecken.

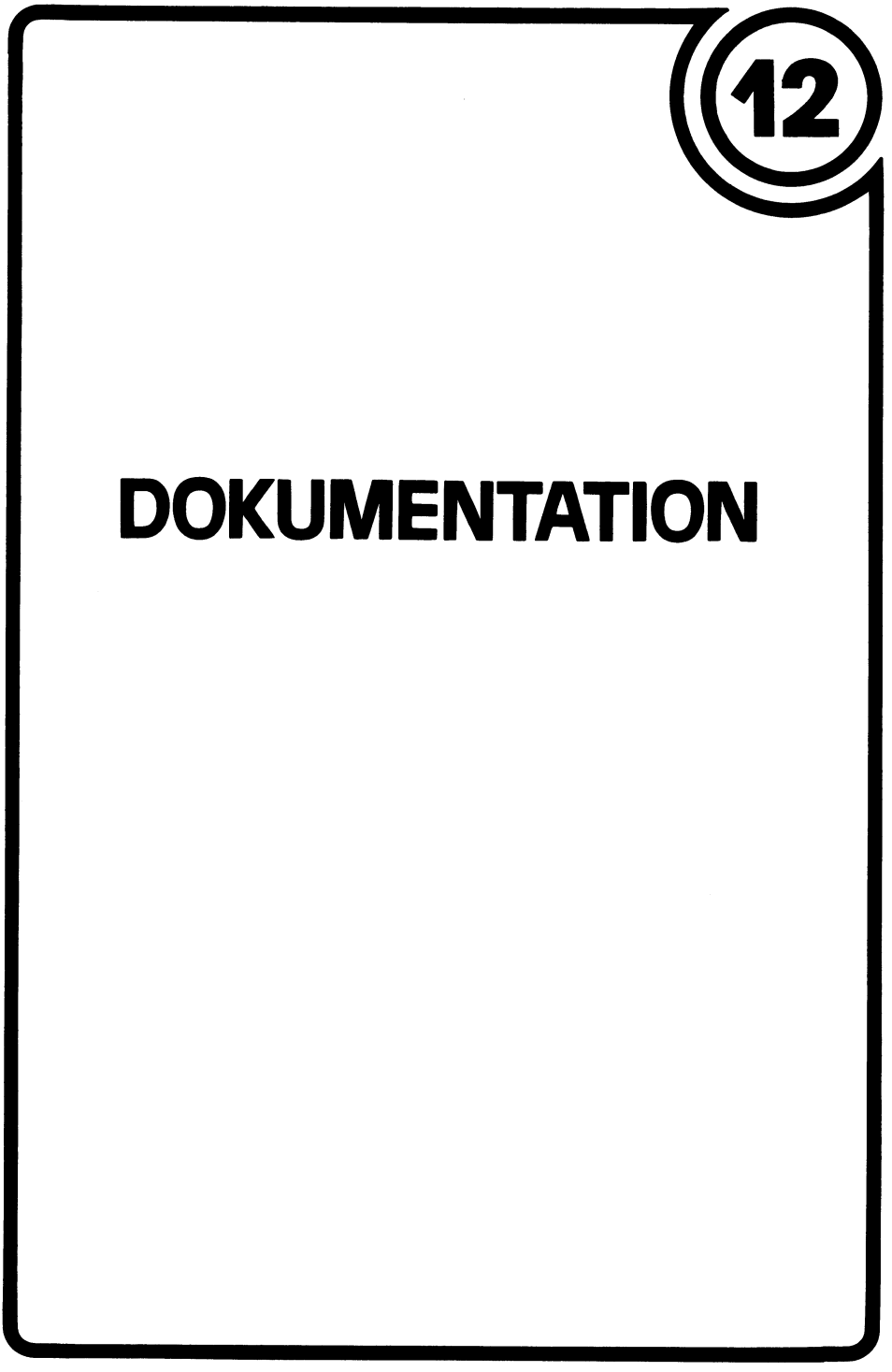
OMFATTANDE FILER PÅ KASSETTBAND

Tekniken att starta och stoppa bandspelarmotorn gör det möjligt att hantera mycket stora filer som endast begränsas av kassettbandets kapacitet. En sådan fil innehåller en större mängd data än internminnet kan rymma på en gång. Läsning och bearbetning av data måste alltså kunna ske i etapper.

Genom att bl a övervaka systemvariabeln på adressen 65021 kan en fil skapas som innehåller en paus mellan varje block. Värdet av denna variabel motsvarar numret på det block som senast skrevs på bandet. När systemvariabelns värde ändras efter skrivning görs ett uppehåll på ca 2 sek före nästa skrivning (se vidare referens 2).

De tillfällen när start/stopp av bandspelarmotorn kan tillgripas är:

- vid bearbetning av datafiler vars innehåll ej ryms i internminnet
- när bearbetning av data utförs långsammare än läsning sker från kassettband
- när data som ska skrivas på filen skapas långsamt, t ex vid inmatning från tangentbordet



DOKUMENTATION

PROGRAMMERINGENS SLUTFAS

Konstruktionen av ett program kan inte anses slutförd förrän dokumentationen av programmet utformats. En väl strukturerad programtext som är välförsedd med kommentarer kan för den vane programmeraren i enstaka fall vara tillräcklig dokumentation. De hjälptexter och svarsanvisningar som presenteras på bildskärmen under programkörningens gång utgör också en väsentlig del i dokumentationen.

I de allra flesta fall behövs dock någon form av tilläggsinformation som kompletterande dokumentation till programmet. Detta gäller speciellt i de fall då användaren inte själv konstruerat programmet. När kommentarer i programtexten måste begränsas på grund av tillgängligt minnesutrymme, är separat dokumentation av programmet givetvis oumbärlig. Hur skall nu sådan dokumentation vara utförd? Innan vi försöker besvara den frågan, låt oss titta på olika målgrupper och deras behov!

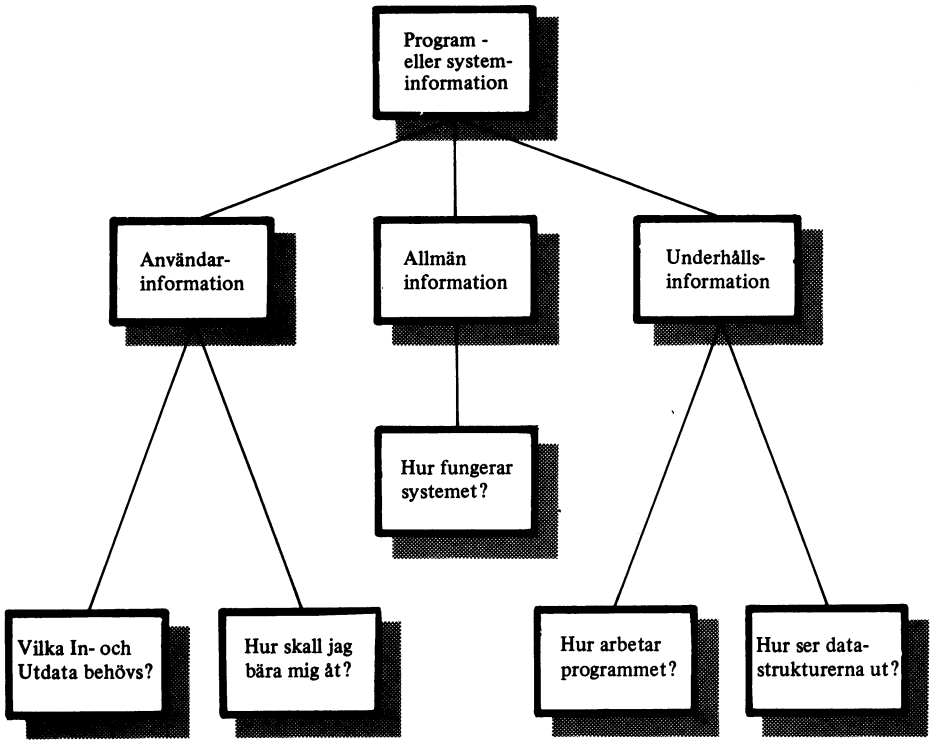
Vi har redan konstaterat att det i huvudsak är två målgrupper vi vänder oss till nämligen:

- användare
- underhållare

Underhållaren – d v s programändraren – kan mycket väl vara samma person som programkonstruktören, men oftast är så inte fallet.

Både användare och underhållare har i vissa fall gemensamt intresse i samma dokumentation. Oftast behöver underhållaren känna till allt vad användaren vet medan användaren inte har samma behov av den program-/systemtekniska dokumentationen.

Om vi försöker strukturera informationen med hänsyn till dessa båda huvudgruppers behov kan resultatet illustreras i bilden på nästa sida. De informationsgrupper som växer fram ger underlag för det fortsatta resonemanget om en mer konkret uppdelning i separata beskrivningar.



PROBLEMÖVERSIKT

För att göra läsaren införstådd med din lösning av problemet bör du börja dokumentationen med att definiera problemet. Beroende på dess omfattning och komplexitet kan det vara nödvändigt att också titta på vilka teorier, beräkningsmetoder, rutiner o s v som vanligen används för att lösa det aktuella problemet utan hjälp av databehandling.

Ibland är den här informationen möjlig att ge i några få meningar, som då med fördel kan bilda inledningen till systembeskrivningen.

SYSTEMBESKRIVNING

En systembeskrivning kan i det här sammanhanget utgöras av en översiktlig beskrivning av det aktuella systemet och dess olika delar, d v s av de olika programmen och de ingående filerna samt deras inbördes relationer.

Beskrivningen bör göras så att den kan förstås av alla typer av läsare. Den bör därför vara hållen i en lättillgänglig form, inga svåra eller ovanliga ord, gärna utförliga beskrivningar av besvärliga avsnitt o s v.

Omfattningen av systembeskrivningen beror givetvis av antalet ingående program och filer.

HANDHAVANDEBESKRIVNING

Handhavandebeskrivningen måste ses som ett av de allra viktigaste avsnitten i dokumentationen. Användaren har inte tillgång till den som konstruerat programmet när han kör det. Om han "kör" fast och programmets ledtexter inte är tillräckliga går han till sin handhavandebeskrivning och där måste han kunna hitta svar på sina frågor.

Hur handhavandebeskrivningen ska se ut beror i hög grad på vad programmet gör, men allmänt sett ska flera saker inrymmas. Erforderlig utrustning måste vara definierad liksom det minnesutrymme som behövs. Likaså ska beskrivas vilka förberedelser som måste utföras, i vilken form indata skall finnas tillgängligt, som underlag för manuell inmatning, eller som fil på skiva eller band.

Därefter bör hela körningsförloppet beskrivas med speciell betoning på de avsnitt som är användarstyrda. Alla frågor som systemet ställer till användaren samt tillhörande svarsalternativ ska anges. Om programmet innehåller feltexter som läggs ut på skärmen i händelse av fel ska texterna samt tillhörande åtgärder redovisas.

Utseende och format på både in- och utdata ska finnas beskrivet liksom hur dessa data lämnas respektive levereras.

I handhavandedelen bör även eventuella åtgärder som ska utföras efter körningen anges, t ex säkerhetskopiering av skivor och band.

PROGRAMBESKRIVNING

Beskrivningen krävs för att underlätta förståelsen av funktion och struktur hos de ingående programmen, så att ändringar i form av rättelser och kompletteringar kan göras på ett säkert och smidigt sätt.

Här är det också viktigt att beskriva vad de olika variablerna används till, eftersom variabelbeteckningarna i BASIC inte ger någon information om detta.

Samma variabelbeteckning, som används med olika betydelse och på skilda ställen i programmet, kan under olyckliga omständigheter ge upphov till felsituationer som ställer till onödigt arbete och förtret.

Försök beskriva programmets stora, självständiga funktionsblock. Bryt ner dessa i mindre enheter och beskriv dem. Ange hur data tas in, förändras och påverkas i programmet.

Programsatser vars funktion i sammanhanget inte är helt uppenbar kan behöva en extra förklaring.

Eftersom de flesta människor snabbare förstår ett bildmässigt visat förlopp än en verbal beskrivning, bör programbeskrivningen åtföljas av en flödesplan eller strukturplan beroende på vilket grafiskt uttryckssätt du föredrar.

DATASTRUKTURER

Alla indata och utdata ska definieras till typ och storlek. Data för lagring/hämtning på filer ska givetvis också beskrivas. Bygger du poster genom att packa ihop strängar måste du klart ange vilka variabler en sådan post består av samt de enskilda termernas storlek.

SAMMANFATTNING

Problemöversikt:

- Vilket problem gäller det?
- Vilka teorier är aktuella?

Systembeskrivning:

- Vilka olika program ingår i systemet?
- Vad gör de olika programmen?
- Vilka filer utnyttjas?
- Vilka kopplingar finns mellan programmen?
- Vilka kopplingar finns mellan program och filer?

Handhavandebeskrivning:

- Vilken ABC 80-konfiguration krävs?
- Hur stort minnesutrymme behövs?
- Vilka yttre enheter (bandspelare, flexskivenhet, skrivare) ska vara anslutna?
- Vilket band ska sitta i bandspelaren?
- Vilka skivor ska användas?
- Behöver byte av skivor eller band ske under körningen?
- Hur ser indata ut?
- Hur lämnas indata?
- Hur ser utdata ut?
- Hur presenteras utdata av systemet?

(forts)

Handhavandebeskrivning – forts

- Hur startas programmet?
- Vilka frågor ställs och i vilken ordning kommer frågorna?
- Vilka svarsalternativ finns?
- Vilka feltexter kan förekomma?
- Hur ska feltexterna tolkas?
- Vilka åtgärder vidtas när fel inträffar?
- Hur avslutas programmet?
- Vad ska göras efter avslutad körning?
- Vad ska säkerhetskopieras?

Programbeskrivning:

- Vilka funktionsmässiga delar kan urskiljas?
- Hur kan dessa brytas ner ytterligare?
- Vad gör varje del?
- Vilka data tar varje del emot?
- Vad händer med de data som varje del tar emot?
- Vad lämnar varje del ifrån sig?
- Vilka "flaggor" eller andra markeringar finns?
- Vad betyder de flaggor som finns?
- Hur och när förändras flaggorna?
- Vad innebär egentligen villkorssatsernas villkor?
- Vilka variabler har du använt och vad använder du dem till?
- Var i programmet förekommer de variabler som du använt?

(forts)

Programbeskrivning – forts

- Vilka strängvariabler har du nyttjat och hur kombinerar du dem?
- När läser du poster till och från filer?
- Hur lagras data på filer?
- Hur sker eventuell övergång till andra program?
- Flödes- eller strukturplan?

Datastrukturer:

- Vilka indatatermer finns?

  - Hur stora tal/hur många tecken ryms?
  - Vilka gränsvärden gäller?
  - Finns förbjudna värden?

- Vilka utdatatermer finns?

  - Hur stora tal/hur många tecken?
  - Vilka gränsvärden gäller?
  - Finns förbjudna värden?

- Vilka speciella datagrupperingar (t ex filposter) finns?

  - Hur är dessa byggda?
  - Speciella avgränsare mellan termerna?
  - Vad betyder respektive term?

- Har du beskrivit och definierat använda mellanlagringsvariabler, loopvariabler, svarsvariabler?

SAKREGISTER OCH REFERENS-LITTERATUR

SAKREGISTER

REFERENSLITTERATUR

1 **ABC om BASIC**
Anders Andersson, Arne Kullbjer, Jan Lundgren, Sören Thornell

2 **Avancerad programmering på ABC 80**
Anders Isaksson, Örjan Kärrsgård

3 **Filsystemer och databaser**
Kjell Bratbergsengen, Knut Hofstad, Kjell Wibe

4 **Mikrodatorns ABC**
Gunnar Markesjö

5 **ABC om mätdatorsystem**
Per Eriksson, Håkan Magnusson, Mats Rasmusson

6 **Styr och mät för ABC 80**
Åke West

7 **Bruksanvisning ABC 80**
LUXOR AB, SCANDIA METRIC AB

8 **Bruksanvisning för FD2 och FD2U**
LUXOR AB, SCANDIA METRIC AB

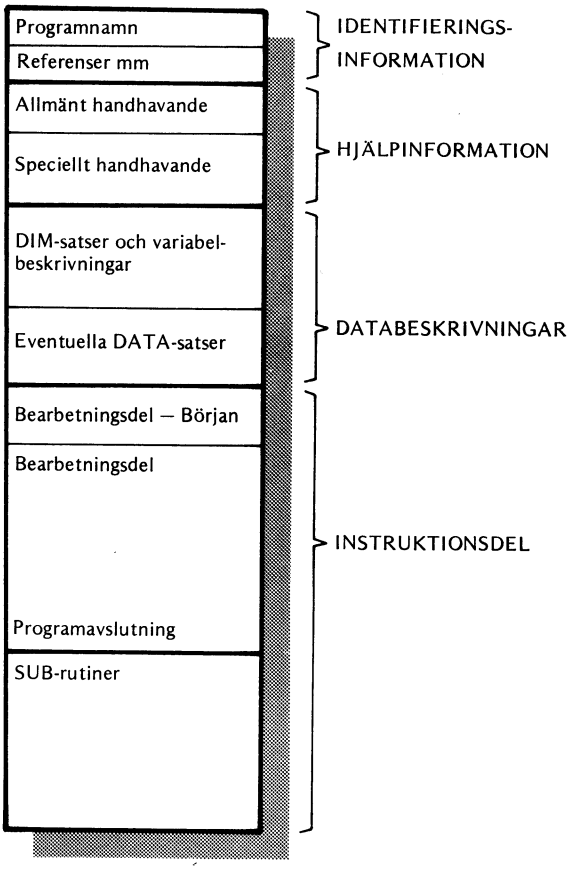
9 **ABC om användardokumentation**
LUXOR AB

PROGRAMMERINGS-KORT

ABC80

PROGRAMMERINGSKORT

PROGRAMUPPBYGGNAD



Bilaga till
ABC om Programmering och Dokumentation

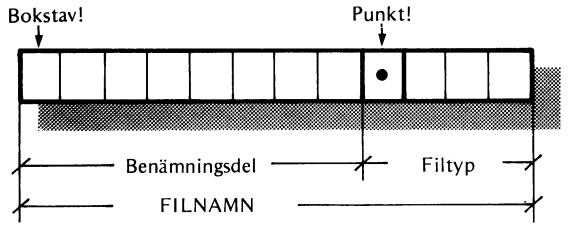
VARIABELBETECKNINGAR

BETECKNING	DATASLAG	VARIABELTYP
A - H	konstanter/indata	flyttal
A% - H%	konstanter/indata	heltal
A¤ - H¤	konstanter/indata	strängvariabel
I - N	index/loopvariabler	flyttal
I% - N%	index/loopvariabler	heltal
O, O%, O¤	undvikes på grund av likheten med siffran 0	flyttal, heltal, strängvariabel
P - U	variabler för lagring	flyttal
P% - U%	av mellanresultat,	heltal
P¤ - U¤	s k slaskvariabler	strängvariabel
V - Z	resultatvariabel	flyttal
V% - Z%	resultatvariabel	heltal
V¤ - Z¤	resultatvariabel	strängvariabel

I variabelbeteckningarna ovan kan alla bokstäver ersättas med en bokstav kombinerad med en siffra 0 - 9.

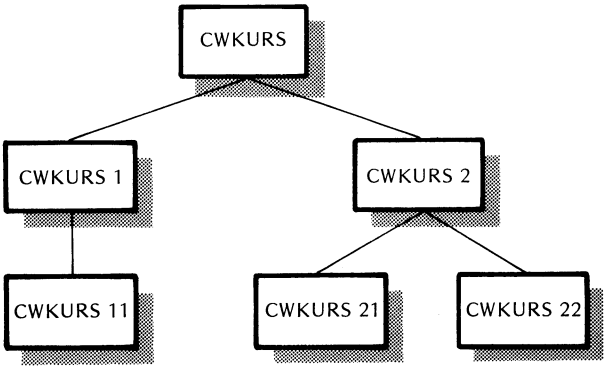
Ex: A8, B3%, C5¤

NAMNGIVNINGSREGLER



Tillsammans med filnamn anges ibland aktuell enhet för filen, t ex DR0:, DR1:, CAS:

Huvudprogram — Underprogram



INSTRUKTIONER

CHAIN "... ... ..."

Överför angivet program från kassett eller flexskiva till ABC 80 och startar programkörning. Har samma effekt som RUN ... ...

```
20 CHAIN "CAS: KATALOG.007"
60 CHAIN "PROGRAM2"
90 CHAIN A¤
```

CLRDOT R,K

Släcker en grafisk punkt på bildskärmen. Jfr SETDOT R,K

```
30 CLRDOT 40, 60
```

CLOSE ...

Stänger angiven fil, d v s avslutar läsning eller skrivning på filen. Om filen öppnats för skrivning skrivs ett filslutsmärke. Är angiven fil lagrad på kassett stoppas bandspelarmotorn.

```
50 CLOSE 4
```

DATA ...

Lagrar variabelvärden som ska läsas av en READ-sats.

```
40 DATA 13, 2.8, EVA
```

Om strängar innehåller mellanslag eller kommatecken måste strängarna omges av citationstecken.

```
50 DATA "BJÖRN STARK"
```

DEF FN ... (...) = ...

Definierar en egen funktion

```
70 DEF FNY (X) = PI * R * R
90 DEF FNZ (B, H) = (B * H)/2
```

DIM ...

Reserverar utrymme för fältvariabler

```
10 DIM P(20)
```

(Reserverar utrymme för 21 flyttalsvariabler P(0) t o m P(20))

```
20 DIM P¤ = 1000
```

(Reserverar utrymme för en sträng med max 1000 tecken)

```
30 DIM P¤ (20) = 10
```

(Reserverar utrymme för 21 strängvariabler om vardera max 10 tecken)

```
40 DIM P % (10, 20)
```

(Reserverar utrymme för en matris med heltalsvariabler innehållande 11 rader och 21 kolumner)

```
50 DIM P¤ (10, 20) = 40
```

(Reserverar utrymme för en strängmatris med 11 rader och 21 kolumner och där varje element omfattar max 40 tecken)

END

Avslutar programkörning, stänger alla filer och nollställer alla variabler.

INSTRUKTIONER (forts)

FOR ... TO ... STEP ...

NEXT ...

Utför ett programavsnitt det antal gånger som bestäms av loopvariabelns start- och slutvärde samt stegvärdet. Utelämnat stegvärde sätts till 1.

```
20 FOR K = 1 TO 20 STEP 2
40 NEXT K
60 FOR L % = A % * 3 % TO B %
90 NEXT L %
```

GET ...

Väntar på tangentnedtryckning. Lagrar mottaget tecken i angiven strängvariabel utan att skriva inmatat tecken på bildskärmen. Alla tecken accepteras.

```
20 GET S7¤
```

GOSUB ...

Medför hopp till subrutin på angiven rad. Jfr RETURN.

```
80 GOSUB 2000
```

GOTO ...

Medför hopp till angiven rad.

```
60 GOTO 10
```

IF ... THEN ... ELSE ...

Om uttrycket efter IF är sant (≠ 0) så utförs det som står mellan THEN och ELSE. I annat fall utförs det som står efter ELSE. (ELSE ... kan utelämnas).

```
40 IF A = 0 THEN GOTO 60 ELSE 10
   (GOTO kan utelämnas efter THEN och ELSE)
```

INPUT ...

Skriver ett frågetecken på bildskärmen och väntar därefter på inmatning av data via tangentbordet. Data som innehåller mellanslag eller kommatecken måste omges av citationstecken.

```
30 INPUT B¤
40 INPUT A, B, C
```

INPUT # ... , ...

Överför data från angiven fil till angiven variabel. Data som innehåller mellanslag packas, d v s tecken för mellanslag överförs inte.

```
50 INPUT # 2, R
60 INPUT # F, R2
```

INPUTLINE ...

Överför en inmatad rad från tangentbordet till angiven strängvariabel. Alla tecken accepteras. Strängvariabelns två sista tecken blir alltid CR (vagnretur) och LF (radframmatning). Inmatade tecken förutom CR och LF skrivs på bildskärmen.

```
90 INPUTLINE S¤
```

INSTRUKTIONER (forts)

INPUTLINE # ... , ...

Överför data från angiven fil till angiven strängvariabel. Samtliga tecken inklusive mellanslag, kommatecken och citationstecken överförs. Strängvariabelns två sista tecken blir CR (vagnretur) och LF (radframmatning)

```
60 INPUTLINE # 2, S¤
90 INPUTLINE # F, S¤
```

KILL "... ... ..."

Raderar angiven fil från flexskiva.

```
30 KILL  "DR 1: TEST.BAC"
90 KILL   A¤+ "TEST.TXT"
```

LET ...

Tilldelar en variabel ett värde.

```
10 LET X = 5 * S
20 T¤ = "LENA"  (LET kan utelämnas)
```

NAME "... ... ..." AS "... ..."

Byter namn på en fil. Filtyp måste anges.

```
90  NAME "DR1: GAMMAL.BAS" AS "NY.BAS"
```

NEXT ...

Avgränsar en slinga (loop) Jfr FOR ... TO ... STEP ...

NOTRACE

Upphäver verkan av TRACE. Jfr TRACE.

ONERRORGOTO ...

Medför hopp till angiven programrad om ett fel, markerat med * i felkodlistan, uppstår under körning.

```
10 ONERRORGOTO 10
```

ON ... GOSUB ... , ...

Medför hopp till olika subrutiner.

```
60 ON P GOSUB 100, 200, 300
```

(P = 1 medför subrutinhopp till rad 100, P = 2 medför subrutinhopp till rad 200 o s v)

ON ... GOTO ... , ...

Medför hopp till olika programrader.

```
70  ON R GOTO 100, 200, 300
```

(R = 1 medför hopp till rad 100, R = 2 medför hopp till rad 200 o s v)

ON ... RESTORE ... , ...

Medför att olika RESTORE ... utförs beroende på värdet av en variabel. Jfr RESTORE.

```
90 ON S RESTORE 100, 200, 300
```

(S = 1 medför RESTORE 100, S = 2 medför RESTORE 200 o s v)

OPEN "... ... ..." ASFILE ...

Öppnar angiven fil för läsning och tilldelar filen ett filnummer (1 - 255).

```
40 OPEN "DR1: ADRESS.TXT" ASFILE 3
60 OPEN "ADRESS.TXT" ASFILE 4
80 OPEN A¤ +".TXT" ASFILE F
```

INSTRUKTIONER (forts)

OUT 6, ...

Aktiverar den inbyggda ljudgeneratorn:

```
20 OUT 6, N      (1≤N≤255, udda tal)
```

Stänger ljudgeneratorn:

```
30 OUT 6, N      (0≤N≤255, jämnt tal)
```

PREPARE "... ... ..." ASFILE ...

Skapar och öppnar en angiven fil för skrivning och tilldelar filen ett filnummer (1 - 255).

```
30 PREPARE "DR1:ADRESS.TXT" ASFILE 3
50 PREPARE A¤ + ".TXT" ASFILE F
```

PRINT ... (; ...)

Skriver på bildskärmen. Kan följas av beräkningsuttryck samt specialfunktionerna TAB(K) och CUR(R,K).

```
20 PRINT "MARJA"
30 PRINT A, B, C  (Utskrift i kolumner)
40 ; "AREA = " ; Y (Utskrift utan mellanrum)
```

PRINT # ... , ... (; # ... , ...)

Skriver på angiven fil.

```
30 PRINT # 2, "SLUMPTAL"
```

RANDOMIZE

Ger RND-funktionen ett slumpmässigt startvärde.

READ ...

Tilldelar variabler värden från DATA-satser.

```
10 READ B, C¤
```

REM ...

Används för kommentarer i program.

```
90 REM *** SUBRUTIN ***
```

RESTORE ...

Medför att nästa READ-sats läser data från och med en viss DATA-sats.

```
50 RESTORE
```

(Läser data från och med den första DATA-satsen)

```
70 RESTORE 100
```

(Läser data från DATA-satser från och med rad 100)

RETURN

Medför återhopp från en subrutin till satsen efter motsvarande GOSUB-sats.

SETDOT R,K

Tänder en grafisk punkt på bildskärmen. Radpositionen måste vara satt i grafisk mod. Jfr CLRDOT R,K
$0 \leq R \leq 71$ och $2 \leq K \leq 79$.

```
70 SETDOT 40, 60
```

STOP

Avslutar programkörning utan att nollställa variabler.

TRACE

Radnummer på utförda programrader skrivs på bildskärmen under körning.

STRÄNGFUNKTIONER

ASC (A¤)

Ger ASCII-värdet för första tecknet i angiven sträng.

```
20 LET P = ASC (B¤ )
```

CHR¤ (...)

Ger ett till fyra tecken motsvarande angivna ASCII-värden.

```
10 PRINT CHR¤ (12)
20 PRINT CHR¤ (78, 73, 76, 83)
```

INSTR (S, A¤ , B¤)

Söker efter strängen B¤ i strängen A¤ med början i position S. Funktionen ger läget för första förekomsten av B¤ i A¤ . Om B¤ ej finns i A¤ erhålls värdet 0.

```
30 R = INSTR (S, A¤ , B¤ )
```

LEFT¤ (A¤ , S)

Ger de S första tecknen i angiven sträng.

```
30 D¤ = LEFT¤ (C¤ , S)
```

LEN (A¤)

Ger aktuella längden av angiven sträng.

```
20 X = LEN (A¤ )
```

MID¤ (A¤ , S, T)

Ger T tecken från och med teckenposition S i angiven sträng.

```
30 B¤ = MID¤ (A¤ , S, T)
```

NUM¤ (S)

Ger en sträng med samma tecken som erhålls när angiven talvariabel skrivs med en PRINT-sats.

```
10 B¤ = NUM¤ (S)
```

RIGHT¤ (A¤ , S)

Ger alla tecken i angiven sträng från och med teckenposition S.

```
40 B¤ = RIGHT¤ (A¤ , S)
```

SPACE¤ (S)

Ger en sträng med angivet antal mellanslag (space).

```
20 B¤ = SPACE¤ (253)
```

STRING¤ (S, T)

Ger en sträng av S stycken tecken som har ASCII-värdet T.

```
30 C¤ = STRING¤ (20, 45)
```

VAL (A¤)

Överför angiven sträng till en talvariabel.

```
10 S = VAL (A¤ )
20 S = VAL (".000000003")
```

+

Sammanfogar strängar till stränguttryck.

```
30 S¤ = A¤ + B¤ + ".BAS"
```

ASCII-FUNKTIONER

ADD¤ (A¤ , B¤ , T)

Adderar de angivna numeriska strängarna. Resultatet avrundas till angivet antal (T) korrekta decimaler. Numeriska strängar får innehålla siffror (max 29 st), tecknen + och – samt decimalpunkt.

```
20 C¤ = ADD¤ (A¤ , B¤ , T)
```

DIV¤ (A¤ , B¤ , T)

Dividerar en numerisk sträng (A¤) med en annan (B¤). Jämför ADD¤ (A¤ , B¤ , T).

```
30 C¤ = DIV¤ (A¤ , B¤ , 10)
```

MUL ¤ (A¤ , B¤ , T)

Multiplicerar en numerisk sträng (A¤) med en annan (B ¤). Jämför ADD¤ (A¤ , B¤ , T).

```
40 D¤ = MUL¤ (A¤ , B¤ , T)
```

SUB¤ (A¤ , B¤ , T)

Subtraherar en numerisk sträng (B¤) från en annan (A¤). Jämför ADD¤ (A¤ , B¤ , T).

```
50 C¤ = SUB¤ (A¤ , B¤ , 2)
```

COMP % (A¤ , B¤)

Jämför två numeriska strängar. Ger följande tal som resultat.

- 1 om A¤ < B¤
- 0 om A¤ = B¤
- + 1 om A¤ > B¤

```
10 P = COMP % (A¤ , B¤ )
```

SPECIALFUNKTIONER

CUR (R,K)

Flyttar markören (cursor) till rad R och kolumn K på bildskärmen. Förekommer endast i PRINT-satser.

$0 \leqslant R \leqslant 23$ (rader)
$0 \leqslant K \leqslant 39$ (kolumner)

```
20 PRINT CUR (23, 0); "NEDTILL-VÄNSTER"
```

DOT (R,K)

Ger värdet SANT (– 1) om den grafiska punkten R,K på bildskärmen är tänd och annars värdet FALSKT (0).

```
10 IF DOT (R,K) THEN CLRDOT R,K ELSE
SETDOT R,K
```

ERRCODE

Ger som resultat senaste felkod.

```
30 E9 = ERRCODE
```

TAB (K)

Flyttar markören till position K på raden. Förekommer endast i PRINT-satser.

$0 \leqslant K \leqslant 255$ (teckenpositioner)

```
40 PRINT TAB (13); "** RUBRIK **"
```

UTTRYCK OCH OPERATORER

Ett uttryck består av tal och funktioner åtskilda av operatorer. Följande operatorer finns:

PRIORITET	OPERATOR	BETYDELSE	EXEMPEL
1	Ü (alt * *)	exponentiering	A Ü B
2	*	multiplikation	A * B
2	/	division	A / B
3	+	addition	A + B
3	-	subtraktion	A - B
4	>	större än	A > B
4	<	mindre än	A < B
4	=	lika med	A = B
4	>=	större än el lika med	A >= B
4	<=	mindre än el lika med	A <= B
4	<>	ej lika med	A <> B

Relationsoperatorerna med prioritet 4 ger värdet SANT (- 1) om uttrycket är sant, annars värdet FALSKT (0). Det finns också logiska operatorer. Dessa har samtliga prioritet större än fyra. Se LOGISKA OPERATORER.

LOGISKA OPERATORER

NOT

ICKE. Är sant om operanden är falsk.

```
10 IF NOT A < B THEN GOTO 20
```

AND

OCH. Är sant om båda operanderna är sanna.

```
20 IF A > B AND C = D THEN 30
```

OR

ELLER. Är sant om minst en av operanderna är sann.

```
30 IF A < B OR C = 10 THEN 40
```

XOR

EXKLUSIVT ELLER. Är sant om endera av operanderna är sann men inte båda.

```
40 IF A = B XOR C = D THEN 50
```

IMP

IMPLICERAR. A IMP B är falskt endast om A är sann och B är falsk.

```
50 IF A IMP B THEN 60
```

EQV

EKVIVALENS. Är sant om båda operanderna är sanna eller om båda är falska.

```
60 IF A = B EQV C = D THEN 70
```

De logiska operatorerna kan också användas på godtyckliga heltal. Operatorerna verkar då bit för bit på motsvarande binära tal.

```
70 A % = B % AND 15 %
```

FELMEDDELANDEN

0 Ej tillåtet öka "DIM"
1 Fel antal index
2 Otillåtet som kommando
3 Minnet fullt
★ 4 För stort flyttal
5 För stort index
6 Hittar ej detta radnummer
★ 7 För stort heltal
8 Finns ej i detta system
9 Index utanför strängen
10 Texten får ej plats i strängen
11 Förstår ej
★ 12 Felaktigt tal
13 Fel antal eller typ av argument
14 Otillåtet tecken efter satsen
15 "=" saknas eller på fel plats
16 Radnummer saknas
17 Otill. blandn. av tal och strängar
18 ")" saknas eller på fel plats
★ 19 Kan ej öppna fler filer
20 För lång rad (> 120 tkn)
★ 21 Hittar ej filen
22 Otillåten sats
23 "TO" saknas
24 "NEXT" saknas
25 Felaktig sats efter "ON"
26 Fel i "ON"-uttryck
27 "NEXT" utan "FOR"
28 Fel variabel efter "NEXT"
29 "RETURN" utan "GOSUB"
★ 30 Data slut
31 Fel data till kommando
32 Filen ej öppnad
33 "AS FILE" saknas
★ 34 Slut på filen
★ 35 Checksummafel vid läsning
★ 36 Checksummafel vid skrivning
★ 37 Felaktigt recordformat
★ 38 Recordnummer utanför filen
★ 39 Filen skrivskyddad
★ 40 Filen raderingsskyddad
★ 41 Skivan full
★ 42 Skivan ej klar
★ 43 Skivan skrivskyddad
44 Logisk fil ej öppen
45 Fel logiskt filnummer
46 Fel enhetsnummer
47 Fel trapnummer
48 Fel i biblioteket
49 Felaktigt fysiskt filnummer
50 Kvadratrot ur negativt tal
51 Enheten upptagen
52 Ej till denna enhet
53 Felaktig rad
54 IEC både sändare och mott.
55 IEC mottagare ej aktiv
56 IEC sändare ej aktiv
57 Funktionen ej definierad
58 Ogiltigt tecken inläst
59 Fel programformat
60 Bit adress > 16 bitar
61 Komma saknas
62 "DOT"-adress utanför skärmen
63 "AS" saknas
64 Felaktig "RENAME"
65 Spill i ASCII-aritmetik
66 Sträng ej numerisk

★ Hanteras med ON ERROR GOTO

TANGENTBORDET

CTRL C Avbryter pågående programkörning eller listning
CTRL X Backstegar och raderar hel rad vid inmatning
CTRL H Backstegar och raderar ett tecken vid inmatning
CTRL ≧ Ger tecknet ■ (fylld ruta)

EMMDATA Box 70
901 03 UMEÅ

MINNESÅTKOMST OCH IN/UT-PORTAR

CALL (A)

Anropar subrutin i maskinspråk på angiven adress (A). Efter utförd subrutin är CALL (A) lika med talet i HL-registret i mikroprocessorn Z80.

```
10 A = CALL (65408)
```

CALL (A, U)

Anropar subrutin i maskinspråk på angiven adress (A). Före subrutinanropet laddas mikroprocessorns DE-register med angivet uttryck (U). Efter utförd subrutin är CALL (A, U) lika med talet i HL-registret.

```
20 B % = CALL (65408 %, U %)
```

INP (P)

Hämtar en byte (oktett) från angiven port (P).

```
30 C % = INP (58)
```

OUT P1, D1, P2, D2, ...

Överför data D1 till port P1, osv.

```
40 OUT 58, 32
```

PEEK (A)

Hämtar en byte (oktett) från angiven minnesadress (A).

```
50 PRINT PEEK (65008)
```

POKE A, D1, D2, ...

Överför data D1, D2, ... till angivna minnesceller från och med angiven adress (A). Data och adresser anges decimalt.

```
60 POKE 65008, 10, 5, 2
```

SWAP % (D)

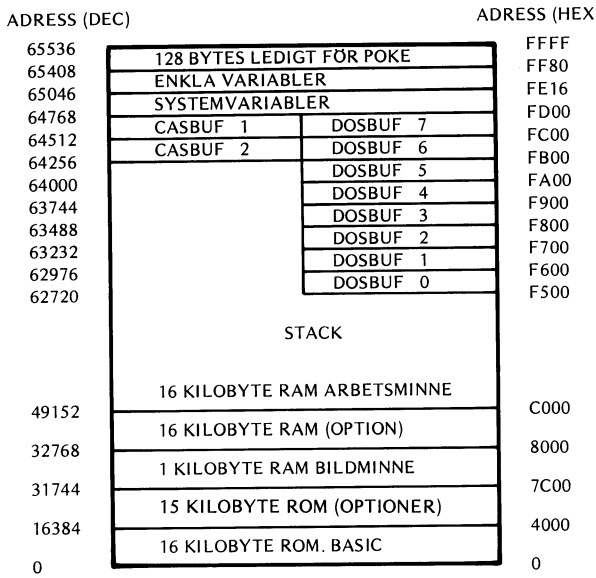
Ger värdet av D tolkat som heltal (2 bytes), men med första och andra byten omkastade.

```
70 B % = SWAP % (D %)
```

SPECIALTECKEN

CHR¤ (7)	Ger pip i den inbyggda högtalaren	(BELL)
CHR¤ (10)	Radframmatning	(LF)
CHR¤ (12)	Tömmer bildskärmen och flyttar markören till övre vänstra hörnet	(FF)
CHR¤ (13)	Motsvarar tangentbordets returtangent	(CR)
CHR¤ (135)	"SLUT GRAFIK" återställer resten av raden till teckenmod	
CHR¤ (151)	"START GRAFIK" försätter återstoden av raden i grafisk mod	

MINNESKARTA



65063	Stackpekarens startadress
65054	"Slut på program"-pekaren
65052	"Början på program"-pekaren

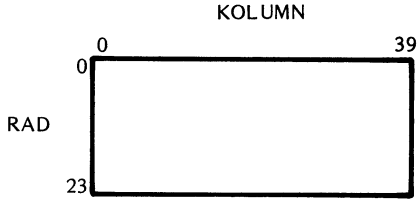
MATEMATISKA FUNKTIONER

ABS (X)	absolutbeloppet av x, $\|x\|$
ATN (X)	arctan x
COS (X)	cos x, x i radianer
EXP (X)	e^x
FIX (X)	heltalsdelen av x, [x]
INT (X)	största heltalet mindre än eller lika med x
LOG (X)	e-logaritmen av x
LOG 10 (X)	10-logaritmen av x
SGN (X)	–1 om $x < 0$. 0 om $x = 0$. 1 om $x > 0$
SIN (X)	sin x, x i radianer
SQR (X)	kvadratroten ur x, $\sqrt{x}$
TAN (X)	tan x, x i radianer
PI	π, talet 3.14159
RND	representerar ett slumptal, 0 - 0.999999

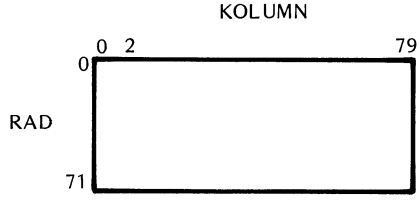
ASCII-TECKEN I TECKENMOD (T) OCH GRAFMOD (G)

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

BILDSKÄRMSKARTA – TECKENMOD



BILDSKÄRMSKARTA – GRAFMOD



CHR¤(151) START GRAFIK
CHR¤(135) SLUT GRAFIK

KOMMANDON
(Diskoperativsystemet)

Förutsatt programvara på flexskiva:

CMDINT.SYS	COPY.ABS	COPYLIB.ABS
DOSGEN.ABS	MEM.ABS	SPACE.ABS

BYE (kommando i OS)

Medför övergång till diskoperativsystemet (DOS). Startar körning av huvudprogrammet CMDINT.SYS.

COPY ... ... , ...

Angiven fil på flexskiva kopieras. Kopians filtyp kan utelämnas.

COPY DR0: SORTERA. BAC, DR1: SORT. BAC
COPY DR1: LISTA. BAS, DR0: LISTA

COPYLIB . , ...

Kopierar flera filer på flexskiva från en enhet till en annan enhet. Olika alternativ väljs enligt programmets anvisningar.

COPYLIB DR1:, DR0:

DOSGEN ...

Flexskiva i angiven enhet testas för filhantering. Existerande filer på flexskivan raderas.

DOSGEN DR1:

DOSGEN, F ...

Flexskiva i angiven enhet prepareras (formatteras) och testas för filhantering. Om filer finns på flexskivan raderas dessa.

DOSGEN, F DR1:

MEM

Tillgängligt minnesutrymme presenteras på bildskärmen.

SPACE

Presenterar återstående tillgängligt utrymme på flexskivorna, samt redovisar det antal nya filnamn (directory entries) som kan skapas.

¤BAS

Medför återgång till BASIC. Motsvarar användning av RESET-knappen.

En flexskiva med systemprogramvara bör innehålla:

SKIVNAMN	.LBL	Flexskivans namn
LIB	.BAC	Biblioteklistningsprogram
BASICERR	.SYS	Felkommandon i klartext
CMDINT	.SYS	Kommandointerpretator för .ABS program
COPYLIB	.ABS	Kopiering av många filer
COPY	.ABS	Kopiering av en fil
DOSGEN	.ABS	Dosgenerering av flexskiva
MEM	.ABS	Minneskarta över ABC 80
SPACE	.ABS	Resterande utrymme på flexskivorna

KOMMANDON
(Operativsystemet)

CLEAR

Nollställer alla variabler och stänger alla filer.

CTRL C

Avbryter programkörning.

ED ...

Ger möjlighet till ändring i en programrad utan att den behöver skrivas om

```
ED 70
```

KILL "... ... ..."

Raderar angiven fil från flexskiva. Filtyp måste anges.

```
KILL "KATALOG.BAC"
KILL "DR1:KATALOG.BAC"
```

LIST ...

Ger utskrift på bildskärmen av det program som finns i ABC 80. Följande varianter finns:

LIST	(samtliga rader)
LIST 30	(endast en rad)
LIST 30-60	(allt mellan två radnummer)
LIST -30	(allt t o m viss rad)
LIST 60-	(allt fr o m viss rad)

LIST ... ...

Överför program som finns i ABC 80 till en fil på kassett eller flexskiva i okompilerad form (ASCII-form). Utelämnad filtyp sätts till .BAS

```
LIST SORTERA
LIST DR1:SORTERA
LIST CAS:TEST.001
```

LIST PR:

Ger utskrift på printer av det program som finns i ABC 80. PR: motsvaras i viss programvara av V24:

LOAD ... ...

Överför program med angivet namn från kassett eller flexskiva till ABC 80. Raderar befintligt program i ABC 80.

```
LOAD BUDGET
LOAD CAS:BUDGET
LOAD DR1:KALKYL.001
```

LOAD CAS: (Överför det första program som påträffas på kassetten till ABC 80)

MERGE ... ...

Överför angivet program från kassett eller flexskiva till ABC 80 utan att radera redan befintligt program i ABC 80. (Radering i det befintliga programmet sker endast om samma radnummer används i det överförda programmet)

```
MERGE PROGRAM2
MERGE DR1:PROGRAM2
MERGE CAS:PROGRAM.003
```

MERGE CAS: (Överför det första program som påträffas på kassetten till ABC 80)

KOMMANDON
(Operativsystemet)

NAME "... ... ..." AS "... ..."

Byter namn på en fil. Filtyp måste anges.

```
NAME "DR1: GAMMAL.BAS" AS "NY.BAS"
```

NEW (alternativt SCR)

Raderar programmet i ABC 80 och stänger alla filer.

NOTRACE

Upphäver verkan av TRACE. Jämför TRACE.

POKE ... , ... ,

Överför data i decimal form, byte för byte, till angiven minnesadress och därefter följande adresser:

```
POKE 65053, 200
POKE 65410, 61, 97, 124, 27, 83
```

PRINT ...
; ...

Skriver på bildskärmen. Jämför instruktionen PRINT.

```
PRINT (PI * 10.5)/7
PRINT X, Y, Z      (jämför instruktionen STOP)
```

REN ...

Numrerar om alla rader i programmet. Följande varianter finns.

```
REN          (första radnummer: 10. Intervall: 10)
REN 50       (första radnummer: 50. Intervall: 50)
REN 90, 20   (första radnummer: 90. Intervall: 20)
```

RUN

Kör det program som finns i ABC 80 efter att ha nollställt alla variabler.

RUN ... ...

Motsvarar kommandot LOAD ... ... följt av kommandot RUN. Jämför LOAD ... ...

```
RUN CAS: KATALOG.007
```

SAVE ... ...

Överför program som finns i ABC 80 till en fil på kassett eller flexskiva i kompilerad form. Utelämnad filtyp sätts till .BAC

```
SAVE SORTERA
SAVE DR1: SORTERA
SAVE CAS: TEST.001
```

TRACE

Radnummer på utförda programrader skrivs på bildskärmen under körning.

UNSAVE ... ...

Raderar angiven fil på flexskiva. Utelämnad filtyp ersätts med .BAC alternativt .BAS

```
UNSAVE SORTERA
UNSAVE DR1: KALKYL.001
```

ABC om programmering och dokumentation

Den nuvarande utvecklingen med små prisbilliga datorsystem innebär att fler och fler lär sig grunderna i programmeringsspråket BASIC. På kort tid lär man sig att skriva enkla BASIC-program. Snart inser man dock att enbart grundläggande kunskaper om de olika instruktionerna i BASIC inte räcker till för att skriva väl fungerande datorprogram.

Denna bok ger dig den ytterligare kunskap om programmering och dokumentation som behövs för att skriva överskådliga och användarvänliga program i BASIC.

Boken beskriver bl a programuppbyggnad, speciella programmeringsrutiner, menyteknik och filhantering.

Boken innehåller lösningar till många praktiska problem som möter BASIC-programmeraren. Förslag och anvisningar varvas med fullt körbara programillustrationer. Dessa är avsedda för den svenska datorn ABC 80 och finns tillgängliga på flexskivor.

Boken kan med fördel användas i gymnasieskolans olika linjer, t ex i ämnet datalära, samt i kurser anordnade av studieförbund och utbildningsföretag. De utförligt redovisade programexemplen underlättar bokens användning vid självstudier.

Förslag till egna övningar och ytterligare lösta exempel återfinns i övningsboken ABC-UPPGIFTER I PROGRAMMERING.

EMMDATA
WT KONSULT
ISBN 91 - 86064 - 00 - 2

LiberLäromedel
Artikelnummer 40–11286–1