SPECTRAVIDEO
SV318/SV328
HEMDATOR
HANDBOK BASIC

INNEHÅLLSFÖRTECKNING

	Sid
1. Grunder	1
1.1 Inmatning	1
1.2 Radformat	3
1.3 Konstanter	3
1.4 Beteckning av enkel och dubbel precision	5
1.5 Variabler	5
1.6 Konvertering av variabeltyper	7
1.7 Uttryck och operatorer	8
2. Kommandon och instruktioner i BASIC	14
2.1 Kommandon i BASIC	15
2.2 Instruktioner	21
2.3 Instruktioner för in- och utmatning	34
2.4 Instruktioner för grafik	60
2.5 Instruktioner för avbrottshantering	68
3. Funktioner	73
3.1 Funktioner med numeriskt resultat	73
3.2 Funktioner med strängresultat	79
3.3 In- och utfunktioner	82
3.4 Funktioner för grafik	87
4. Minnesåtkomst och IN/UT-portar	89
4.1 Instruktioner	89
4.2 Funktioner	91
BILAGOR	
A ASCII-koder	93
B Filer	94
C Reserverade ord	97
D Programexempel PUT SPRITE, ON KEY, STICK	98
E Escapesekvenser för skärmredigering	100
F Parameteröverföringar vid USR	101
G Videoramadresser	103
H Felmeddelanden och felkoder	105
I Matematiska funktioner	109
SAKREGISTER	110

HANDBOK SV BASIC

1. GRUNDER

1.0 Funktionssätt

När BASIC initieras (startas) visas följande text på skärmen:

```
SV extended BASIC version 1.0
Copyright 1983 (C) by Microsoft Corp
xxxxx Bytes free
Ok
```

xxxxx skrivs inte ut utan ersätts av datorn med ett tal som anger hur många bytes som är lediga för BASIC-programmet. Ett tecken motsvarar en byte. För att få antalet lediga kilobyte divideras detta tal med 1024.

"Ok" betyder att BASIC befinner sig på kommandonivå och att datorn är beredd att ta emot kommandon. BASIC-språket kan nu användas på ettdera av två sätt - direktkommandon eller programkommandon.

I direktkommandoläget föregås kommandon och instruktioner i BASIC inte av radnummer. När ett direktkommando matats in i datorn och tangenten <ENTER> tryckts ner utförs detta direkt. Resultat från aritmetiska eller logiska beräkningar visas omedelbart och lagras för framtida användning, men själva instruktionen raderas efter utförandet. Denna metod är värdefull att använda vid felsökning eller när man använder BASIC som "kalkylator" för snabba beräkningar som inte kräver hela program.

Programkommandoläget används för inmatning av program. Programrader föregås av radnummer och lagras i minnet när <ENTER> trycks ner. Programmet som är lagrat i minnet utförs när kommandot RUN matas in.

1.1 Inmatning

Man kan bara mata in en rad med instruktioner i taget och raden avslutas i allmänhet <ENTER>. Det är inte nödvändigt att markören (skrivmärket) står i slutet på raden när man trycker på <ENTER>. Med den i BASIC inbyggda bildskärmsredigeraren (editorn) är det möjligt att på nytt mata in en rad som visas på skärmen. Markören kan då flyttas med hjälp av markörkontrollplattan (skivan med de fyra pilarna) på SV-318 eller med de fyra piltangenterna på SV-328. Eventuella ändringar görs därefter och <ENTER> trycks ner. På detta sätt kan man t.ex ändra radnummer på en rad utan att behöva skriva om hela raden.

En inmatad rad kan sträcka sig över flera rader på bildskärmen. Om markören under inskrivningen av raden flyttar sig förbi bildskärmskanten och ner på nästa rad, så hör denna

rad till den inmatade raden, oavsett om det står något på den eller inte. I detta fall kan det vara lämpligt att trycka <CTRL-E> för att radera det gamla radinnehållet eftersom det annars kommer att matas in tillsammans med raden när <ENTER> trycks ner.

Skrivsättet <CTRL>-U betyder: tryck ner tangenten <CTRL> (på vänstra sidan av tangentbordet över SHIFT) och tryck därefter tangenten U medan <CTRL> fortfarande är nertryckt. Vid styrning av markören under inmatning, kan följande tangenter användas:

Tangenter:	Motsvarar:	Funktion:
<CTRL>-B	CHR$(2)	flytta markören till början av föregående ord.
<CTRL>-E	CHR$(5)	skär av rad (radera från markörposition till radslut).
<CTRL>-F	CHR$(6)	flytta markören till början av nästa ord.
<CTRL>-G	CHR$(7)	ger pip i högtalaren.
<CTRL>-H	CHR$(8)	bakåtstegning, radera tecken till vänster om markören och dra raden ett tecken åt vänster (även tangenten med dubbelpilen ovanför <ENTER>).
<CTRL>-I	CHR$(9)	tabulering. Ger åtta tomtecken (blanka eller space). (Samma funktion som dubbelpilen på vänstra sidan av tangentbordet).
<CTRL>-K	CHR$(11)	markör till hemaläge i övre vänstra hörnet av tangentbordet (sama funktion som <SHIFT>-<CLS/HM>).
<CTRL>-L	CHR$(12)	blanka (radera) skärmen, flyttar markören till hemmaläget (även <CLS/HM>)
<CTRL>-M	CHR$(13)	även <ENTER>, inmatning av aktuell rad till BASIC.
<CTRL>-N	CHR$(14)	flytta markör till sista position på raden (inte alltid position 40).
<CTRL>-R	CHR$(18)	omkoppling mellan infoga och ersätt (även tangenten <INS>). Infogaläget kan även upphävas genom att flytta markören med pilarna eller genom att radera ett tecken.
<CTRL>-U	CHR$(21)	radera logisk rad (kan vara mer än en bildskärmsrad), placerar markören vid början av den tomma raden.
<CTRL>-Ö	CHR$(28)	flytta markör ett tecken åt höger (samma funktion som pilen mot höger på markörkontrollplattan).
<CTRL>-Å	CHR$(29)	flytta markör ett tecken åt vänster (samma funktion som pilen mot vänster på markörkontrollplattan).
<CTRL>-Ü	CHR$(30)	flytta markör en rad uppåt (samma funktion som pilen uppåt på markörkontrollplattan).
<CTRL>-_	CHR$(31)	flytta markör en rad neråt (samma funktion som pil neråt på markörkontrollplattan).

<DEL>	CHR$(127)	raderar tecknet som markören står på och förskjuter raden ett tecken åt vänster.

Övriga specialtangenter är:

<STOP>	avbryt programkörning eller visning på bildskärmen tills nästa gång <STOP> trycks ner.
<CTRL>-<STOP>	avbryter programkörning eller utförande av ett kommando.

1.2 Radformat

Programrader i ett BASIC-program har följande format (hakparenteser anger att uppgiften är valfri):

```
nnnnn BASIC instruktion [:BASIC instruktion...] <ENTER>
```

En BASIC-instruktion består av ett BASIC-ord (kan vara ett kommando) och med efterföljande parametrar.

Flera BASIC-instruktioner kan skrivas på samma rad allt efter programmerarens önskemål men varje instruktion på en rad måste åtskiljas från den föregående med ett kolon. Minst en instruktion måste finnas på varje rad. En programrad börjar alltid med ett radnummer, avslutas med <ENTER> och får innehålla högst 255 tecken.

Radnumren anger i vilken ordning programraderna lagras i minnet och kan användas också som referenser vid förgreningar i programmet och vid redigering. Radnumren måste ligga mellan gränserna 0 och 65529. I SV BASIC kan en punkt (.) användas tillsammans med kommandona LIST, AUTO och DELETE för att ange senast bearbetade rad.

1.3 Konstanter

Konstanter är de värden som BASIC använder under programkörningen. Det finns två typer av konstanter: teckensträngar och numeriska konstanter.

En strängkonstant är en följd av upp till 255 alfanumeriska tecken som innesluts av citationstecken. Exempel på strängkonstanter:

"HEJ"	
"25,000 KR"	
"Antal anställda"	
""	betecknar en "tom" teckensträng eller ett tomtecken

Som tecken i teckensträngar är alla på bildskärmen skrivbara tecken (även grafiska tecken) tillåtna.

Numeriska konstanter är positiva eller negativa tal. Numeriska konstanter i BASIC får inte innehålla kommatecken. Det finns fem typer av numeriska konstanter:

1. Heltalskonstanter — Hela tal mellan -32768 och +32767. Heltalskonstanter innehåller inte decimalpunkter

2. Decimaltal — Positiva eller negativa reella tal dvs tal som innehåller decimalpunkt

3. Flyttalskonstanter — Positiva eller negativa tal representerade i exponentialform. En flyttalskonstant består av ett (eventuellt) teckensatt heltal eller decimaltal (mantissan) följt av bokstaven E och ett heltal (exponenten) inom intervallet -64 till +62.
Exempel:

```
    1E5 = 10000
 -1E-5 = -0.000001
31415926E-7
```

4. Hexadecimala konstanter — Tal med basen 16 och med prefixet &H. Eftersom siffrorna 0 till 9 inte räcker till för dessa tal används även bokstäverna A till F. Talen är heltal och anges med högst 4 siffror. Exempel:

Hexadecimalt		Decimalt
&HA	=	10
&H0f	=	15
&H1000	=	16384
&H9	=	9

5. Oktala konstanter — Tal med basen 8 och med prefixet &O.

Exempel:

Oktalt		Decimalt
&O77	=	63
&O100	=	64

6. Binära konstanter — Tal med basen 2 och med prefixet &B. Giltiga siffror är endast 0 och 1 och högsta antalet siffror 16. Exempel:

Binärt		Decimalt
&B10000	=	16

1.4 Beteckning av enkel och dubbel precision vid numeriska konstanter

I SV BASIC kan konstanter ha antingen enkel eller dubbel precision. Om de anges med dubbel precision lagras och skrivs talet ut med upp till 16 siffror.

En numerisk konstant med enkel precision är ett tal som har:

1. sex eller färre siffror, eller
2. ett efterföljande utropstecken (!)

En numerisk konstant med dubbel precision är ett tal som har:

1. mellan sju och 14 siffror, eller
2. ett efterföljande nummertecken (#)

Blandning av exponentiell skrivning och tecknen "!" resp "#" är inte tillåtet eftersom det ger falska värden.

Exempel:

Konstanter med enkel precision	Konstanter med dubbel precision
46.8	345692811
-10.9E-06	-1.09432E-06
3489.0	3489.0#
22.5!	7654321.1234
12345678 (observera: efter 6:an sker avrundning!)	

1.5 Variabler

Variabler är namn som används för att representera värden (numeriska eller strängar) som används i BASIC-program. Värdet på variabeln kan direkt anges av programmeraren eller tilldelas som resultat av beräkningar i programmet. Innan en variabel har tilldelats ett värde, är dess värde noll. Innan en strängvariabel tilldelats ett värde är det en tom sträng ("").

1.5.1 Variabelnamn

I SV BASIC får variabelnamn vara hur långa som helst men bara 2 tecken är signifikanta (används för att skilja variablerna åt). Någon skillnad mellan stora och små bokstäver görs inte. Tillåtna tecken i variabelnamn är bokstäver (A-Z) och siffror. Första tecknet måste dock vara en bokstav. Ett variabelnamn får inte vara ett reserverat ord, t.ex PRINT, LET eller ON. Reserverade ord får heller inte

förekomma inuti variabelnamn. Reserverade ord är samtliga BASIC-kommandon, -instruktioner, funktionsnamn och namn på operatorer. Om ett variabelnamn börjar med FN antar BASIC att det är ett anrop till en användardefinierad funktion.

Variabler kan representera antingen ett numeriskt värde eller en teckensträng. Strängvariabler skrivs med ett dollartecken ($) som sista tecken. Exempel: A$ = "ÅRSRAPPORT". Dollartecknet är ett deklarationstecken, dvs det "deklarerar" att variabeln representerar en teckensträng.

I SV BASIC kan variabelnamn för numeriska variabler deklareras som heltal, för enkel precision eller för dubbel precision. Deklarationstecknen för dessa variabler är följande:

%	heltalsvariabel
!	variabel med enkel precision
#	variabel med dubbel precision

Standardvärdet för numerisk variabel är dubbel precision. En ytterligare möjlighet att deklarera variabeltyper är med hjälp av instruktionerna DEFINT, DEFSNG, DEFDBL och DEFSTR (se dessa).

1.5.2 Listvariabler

En lista är en grupp eller en tabell med värden av samma variabeltyp till vilka man refererar med samma variabelnamn. Varje element i listan är definierat av en listvariabel som i sin tur definieras med en DIM-sats som har formen DIM variabel (antal element). Högsta antalet dimensioner för en lista är 255. Maximala antalet element för varje dimension är 32767. Antal element i listan bestäms med instruktionen DIM. Exempel:

```
DEFINT A : DIM A(10)
```

10 är här listans index. Eftersom bara ett tal finns är listan endimensionell. Listan innehåller 11 heltalsvariabler, A(0)...A(10).

```
DEFDBL M : DIM M(9,9)
```

Denna sats definierar en lista med två dimensioner (index) och ger 100 variabler med dubbel precision, M(0,0)...M(0,9), M(1,0)...M(1,9), ..., M(9,0)...M(9,9)

Av ovanstående exempel framgår att elementen i en lista alltid numreras från noll. Om en lista har färre än 10 element behöver den inte dimensioneras. På samma sätt som vid numeriska variabler kan även strängvariabler definieras. Detta beteckningssätt för variabler är särskilt användbart vid vektor- och matrisräkning, dvs där flera variabler används för likartade beräkningar.

1.5.3 Utrymmesbehov i minnet

Numeriska variabler:	Antal bytes
Heltal	2
Enkel precision	4
Dubbel precision	8

Strängvariabler:
Strängar behöver för varje tecken som de innehåller, en byte plus tre extra bytes där informationen för BASIC lagras.

Listvariabler:
Vid listvariabler får minnesbehovet genom att multiplicera antalet element med behovet för varje tecken.

1.6 Konvertering av variabeltyper

SV BASIC kan vid behov konvertera numeriska konstanter från en typ till en annan. Följande regler och exempel gäller då:

1. Om en numerisk variabel med en viss typ sätts lika en variabel med en annan typ kommer talet att lagras med den typ som anges i variabelnamnet. Om en strängvariabel sätts lika en numerisk variabel eller vice versa kommer felmeddelandet "Type mismatch" att visas.

   Exempel:

   ```
   10 A% = 23.42        lagra värdet 23.42 i heltalsvariabeln A%
   20 PRINT A%          skriver ut värdet av A%
   RUN                  startar programkörning
    23                  programresultatet
   ```

   Försöker man att tilldela en variabel ett värde som ligger utanför gränserna för variabeln så förrblir det tidigare värdet hos variabeln oförändrat och felmeddelandet "Overflow" ("Spill") fås och programkörningen avbryts.

   Om ett tal med enkel eller dubbel precision omvandlas till ett heltal så avrundas talet.

   ```
   10 A!=55.88
   20 B%=A!
   30 PRINT A!,B%
   RUN
     55.88         56
   ```

Om ett värde med dubbel precision tilldelas ett värde med enkel precision kommer endast de avrundade sju första siffrorna i det konverterade talet att vara giltiga. Detta beror på att endast sju siffror angavs i värdet med enkel precision. Absolutvärdet av skillnaden mellan det utskrivna talet med dubbel precision och det ursprungliga talet med enkel precision kommer att vara mindre än 6.3E-8 gånger det ursprungliga talet med enkel precision.

```
10 A = 2.04
20 B# = A
30 PRINT A,B#
RUN
 2.04     2.039999961853027
```

2. Vid beräkning av uttryck konverteras alla operander i ett aritmetiskt eller relationsuttryck till samma precisionsnivå och då till nivån hos den operand som har den högsta precisionen. Resultatet av en aritmetisk beräkning får också denna precisionsnivå.

```
10 D# = 6#/7
20 PRINT D#
RUN
 .8571428571428571
```

Beräkningen utfördes med dubbel precision och resultatet lagrades i D# som ett värde med dubbel precision

```
10 D = 6#/7
20 PRINT D
RUN
 .857143
```

Beräkningen utfördes med dubbel precision och resultatet lagrades i D (variabel med enkel precision)

3. Logiska operatorer konverterar operanderna till heltal och ger heltalsresultat. Operanderna måste ligga inom intervallet -32768 till 32767 eftersom annars ett "Overflow"-fel uppstår.

1.7 Uttryck och operatorer

Ett uttryck kan i enklaste fallet vara en enstaka sträng eller numerisk konstant eller en variabel eller en kombination av konstanter och variabler som tillsammans med operatorer ger ett enstaka värde.

Operatorer genomför matematiska eller logiska beräkningar av värden. Operatorerna i BASIC kan finnas inom följande fyra kategorier:

1. aritmetiska operatorer — högsta prioritet
2. relationsoperatorer
3. logiska operatorer
4. funktionsoperatorer — lägsta prioritet

Man kan blanda de olika typerna av operatorer i ett uttryck. Därvid utförs först operatorer med den högre prioriteten, därefter dem med lägre prioritet. Ordningsföljden kan dock ändras med parenteser och verkställigheten följer då den som gäller för aritmetiska operatorer.

1.7.1 Aritmetiska operatorer

Om ett uttryck innehåller flera operatorer så utförs de i en bestämd ordningsföljd, i allmänhet från vänster till höger. Därvid jämförs varje operator med den som står till höger. Har den högra operatorn högre prioritet jämförs den återigen med den som står till höger tills dess att en operator med samma eller lägre prioritet hittas. Därefter utförs operationen med två likvärdiga operatorer eller operatorer med samma prioritet. Sedan görs en jämförelse åt vänster tills hela uttrycket är beräknat.

De aritmetiska operatorerna, i den ordning de utförs, är:

Operator	Operation	Exempel på uttryck
Ü	exponentiering	XÜY
-	negation	-X
*	multiplikation, division	X*Y, X/Y
+ , -	addition, subtraktion	X+Y, X-Y

För att ändra ordningsföljden som operationa utförs med, används parenteser. Operationer inom parenteser utförs först. Inom parenteserna gäller den vanliga ordningsföljden.

Nedan visas några exempel på algebraiska uttryck och motsvarigheterna i BASIC.

Algebraiskt uttryck	Uttryck i BASIC
X + 2Y	X+Y*2
X - Y/Z	X-Y/Z
XY/Z	X*Y/Z
(X + Y)/Z	(X+Y)/Z
(XÜ2)ÜY	(XÜ2)ÜY
XÜ(YÜZ)	XÜ(YÜZ)
X (-Y)	X*(-Y)

Två operatorer efter varandra måste åtskiljas av parenteser

Det finns två aritmetiska operatorer till:

1.	Ö	Delar heltalen med varandra
2.	MOD	Ger resten vid heltalsdivision

Exempel:

```
PRINT 10.4 MOD 4              (10/4 = 2 resten 2)
 2
Ok
```

```
PRINT 25.68 mod 6.99            (25/6 = 4 resten 1)
 1
Ok

PRINT 25.68Ö6.99                (25/6 = 4)
 4
Ok
```

1.7.1.1 spill och division med noll

Om det under beräkning av ett uttryck påträffas division med noll, eller om noll upphöjs till ett negativt tal visas på skärmen felmeddelandet "Division by zero" ("division med noll").

Om spill (tal utanför maskinens kapacitet) uppstår vid en beräkning skrivs felmeddelandet "Overflow" ("spill") ut på skärmen men programexekveringen fortsätter dock.

1.7.2 Relationsoperatorer

Relationsoperatorer används för att jämföra två tal med varandra. Resultatet av jämförelsen är antingen "sant" (motsvarar värdet -1) eller "falskt" (motsvarar värdet 0). Detta resultat kan användas för beslut angående flödet i programmet (se instruktionen IF).

Operator	Prövad relation	Uttryck
=	likhet	X = Y
<>	olikhet (skilt från)	X <> Y
<	mindre än	X < Y
>	större än	X > Y
<=	mindre än eller lika med	X <= Y
>=	större än eller lika med	X >= Y

(likhetstecken kan även användas för att tilldela ev variabel ett värde, se instruktionen LET)

När aritmetiska och relativa operatorer kombineras i ett uttryck utförs alltid de aritmetiska beräkningarna först. Så är t ex uttrycket

```
X + Y < (T - 1) / Z
```

sant om och endast om värdet av X plus Y är mindrw än värdet av T - 1 dividerat med Z. Fler exempel:

```
IF SIN(X) < 0 GOTO 1000
IF I MOD J <> 0 then K = K + 1
```

1.7.3 logiska operatorer

Logiska operatorer gör jämförelser av flerfaldiga förhållanden, styr bitvisa storheter och genomför booleanska operationer. Den logiska operatorn ger ett bitvis resultat som antingen är "sant" (inte noll) eller "falskt" (noll". I uttryck genomförs de logiska operationerna efter aritmetiska och relationsoperationer. Resultatet av logiska operationer visas i nedanstående tabell ("sanningstabell"). Operatorerna visas i den ordning de genomförs.

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	0

På samma sätt som de relationsoperationer kan användas för att "fatta beslut" om flödet i ett program så kan de logiska operatorerna knyta ihop två eller flera relationer och ge ett sant eller falskt värde som kan användas i ett "beslut" (se IF, avsnitt 2.22).

Logiska operatorer fungerar så att de konverterar respektive operand till sexton bitars, teckensatta, tvåkomplementära heltal inom talområdet -32768 till +32767 (om operanderna inte finns inom detta område fås ett felmeddelande). Om bägge operanderna är 0 eller -1 ges resultatet 0 och -1. Operationen (jämförelsen) utförs bitvis på detta sätt dvs genom att varje bit i resultatet bestäms av motsvarande bitar i de bägge operanderna.

På detta sätt är det möjligt att använda logiska operationer för att testa vissa bitar i en viss byte. Det är till exempel möjligt att "maskera" alla bitarna utom en hos den byte som anger statusen i datorns in-utport. Operatorn OR kan användas för att "slå ihop" två bytes för att få ett visst binärt tal. Följande exempel visar hur de logiska operatorerna fungerar.

63 AND 16 = 16	63 = binärt 111111 och 16 = binärt 10000, eftersom det vid AND bara blir 1 där ettan finns i bägge talen är 63 AND 16 = 16
15 AND 14 = 14	15 = binärt 1111 och 14 = binärt 1110, alltså är 15 AND 14 = 14 (binärt 1110)
4 OR 2 = 6	4 = binärt 100 och 2 = binärt 010, eftersom det vid OR bara blir 1 där minst en etta står är alltså 4 OR 2 = 6 (binärt 110)

1.7.4 Funktionsoperatorer

En funktion används i ett uttryck för att anropa en förutbestämd operation som skall utföras med en operand. BASIC har två typer av funktioner, dels "inbyggda" funktioner som finns i systemet som t ex SQR (kvadratroten) eller SIN (sinus) som beskrivs i avsnitt 3 och dels "användardefinierade" funktioner som skrivs av programmeraren. Se avsnitt om DEF FN och USR.

1.7.5 Strängoperationer

I likhet med vid numeriska operatorer finns det operatorer för strängar. Först och främst kan sammankedjning av strängar (konkatering) utföras. För detta används "+".

Exempel:

```
10 A$ = "FIL" : B$ = "NAMN"
20 PRINT A$ + b$
30 PRINT "NYTT " + A$ + B$
RUN
FILNAMN
NYTT FILNAMN
```

Förutom sammankedjning av strängar kan operationer med delsträngar utföras. Dessa beskrivs närmare i avsnitt 3.2 (funktionerna LEFT$, MID$ och RIGHT$).

Strängar kan jämföras med samma relationsoperatorer som används vid tal och siffror:

```
=    < >   <    >    <=    >=
```

Jämförelser av teckensträngar görs genom att tecknen tas ett i taget från varje sträng och därefter jämförs ASCII-värdena. Om dessa är lika fortsätter man med nästa tecken. Om alla ASCII-koder är likadana är strängarna likadana. Om koderna är olika sätts det lägre värdet före det större och jämförelsen slutar. Om, vid jämförelsen, slutet på en sträng nås, sägs den kortare strängen vara mindre. Blanka tecken i början och slutet är signifikanta (har betydelse).

Exempel:

"AB" är större än "AA". Först jämförs de första tecknen med varandra. Eftersom de är lika jämförs nästa tecken.
ASCCI-koden för "A" är 65
ASCII-koden för "B" är 66. Eftersom 66 är större än 65 så är "AB" större än "AA".

```
"FILNAMN" = "FILNAMN"
"X&" > "X L"
"CL" = "CL"
"kg" > "KG"
"SMUTT" > "SMUTTE"
B$ < "830716" där B$ ="830715"
```

Således kan jämförelser av strängar användas för att göra tester på värdet av teckensträngare eller för att sortera strängar i bokstavsordning. Alla strängkonstanter som används vid jämförelser måste omges av citationstecken (").

2. KOMMANDON OCH INSTRUKTIONER I BASIC

Alla beskrivningar av kommandon och instruktioner i detta avsnitt är uppställda på följande sätt:

Format	Visar det rätta skrivsättet för varje kommando. Se nedan för närmare beskrivning av hur format skrivs.
Version	Anger om kommandot finns i grundversionen av BASIC eller om det laddas in från flexskiva.
Funktion	Anger vad instruktionen används till.
Användning	Beskriver i detalj användningen.
Exempel	Ger exempel på program eller programavsnitt som visar användningen av instruktionen.

Formatangivelser:
När formatet för en instruktion eller ett kommando anges gäller följande regler:

1. Uttryck skrivna med stora bokstäver måste skrivas som de anges.

2. Uttryck skrivna med små bokstäver och inneslutna mellan vinkelparenteser (<>) anges av programmeraren.

3. Uttryck inom hakparenteser ([]) är valfria och behöver inte anges.

4. Alla interpunktionsmärken såsom kommatecken, parenteser, semikolon, bindestreck och likhetstecken (men inte vinkel-och hakparenteser) måste anges exakt som de är skrivna. Mellanslag är ibland inlagda för att öka tydligheten och kan men behöver inte anges.

5. Uttryck följda av en punktrad (...) får upprepas valfritt antal gånger (dock högst upp till radens maximala längd som är 255 tecken).

6. Uttryck som åtskiljs av snedstreck (/) är alternativa och endast ett av dem får väljas.

7. Av uttryck som skiljs åt av]/[måste ett väljas (obligatoriskt alternativ).

Regler för angivande av stränguttrycken <fil> och <filnamn>:

<fil> definieras som:	[<enhet>:]<filnamn>
<filnamn> som:	<namn>[[.]<filtyp>]

<enhet> är en sträng som anger från/till vilken enhet som filen skall läsas/skrivas. Följande strängar är tillåtna:

CAS:	anger kassettbandspelaren
CRT:	anger bildskärmen (endast utmatning)
KYBD:	anger tangentbordet (endast utmatning)
LPT:	anger skrivare (endast utmatning)
MDM :	anger modem
1:	anger flexskivenhet 1
2:	anger flexskivenhet 2

<namn> är en sträng med maximalt 6 tecken.
<filtyp> är en sträng med maximalt tre tecken som används för att ange typ av fil.
Om <enhet> inte anges, antas kassettbandspelaren.
Om använd enhet är kassettbandspelare (särskilt vid CLOAD och CSAVE) så består <filnamn> bara av <namn>.

Exempel:

```
LOAD "CAS:GRAFIK"

LOAD "1:ORMEN.BAS"
```

2.1 Kommandon i BASIC

Kommandon skiljer sig från instruktionen genom att när de utförts så återgår BASIC till kommandonivå. Samma sak gäller när kommandona avbryts genom <CTRL>-<STOP>. Om ett kommando utförs indirekt (i ett program) så är det aktuella kommandot den sista instruktion som utförs i programmet och programmet avbryts efter kommandot.

Exempel:

```
10 A=10
20 B=20
30 LIST
40 PRINT A,B
RUN                 Start av programmet
10 A=10             Denna utskrift är en följd
20 B=20             av LIST-kommandot på rad 30.
30 LIST             Värdena på A och B skrivs
40 PRINT A,B        inte ut eftersom rad 30 är
OK                  den sista rad som utförs.
```

2.1.1 AUTO

Format AUTO [<radnummer>[,<stegning>]]

Funktion Ger automatiskt ett nytt radnummer efter varje vagnretur (tryckning på ENTER-tangenten).

Användning AUTO startar radnumreringen med <radnummer> och ökar sedan detta till nästa rad med <stegning>. Standardvärdet för bägga värdena är 10. Om <radnummer> följs av ett kommatecken utan att <stegning> används, antas automatiskt att <stegning> som angavs i förra AUTO-kommandot gäller.

Om AUTO kommer till ett radnummer som redan är upptaget, skrivs en asterisk "*" ut efter siffran för att varna för att en ny inmatning raderar en befintlig rad. Om man då trycker <ENTER> direkt efter asterisken sparas befintlig rad och nästa radnummer skrivs ut.

AUTO avbryts med <CTRL>-<STOP> eller <CTRL>-C. Den rad i vilken AUTO avbryts, sparas inte. BASIC återgår till kommandonivån.

Exempel AUTO 100,50 ger radnumren 100, 150, 200,....., osv

AUTO ger radnumren 10, 20, 30, 40 osv

2.1.2 CONT

Format: CONT

Funktion: Fortsätter programkörning efter det att <CTRL>-<STOP> har tryckts ner eller ett STOP eller END har utförts i ett program.

Användning: Programkörningen startar på samma ställe som avbrottet skedde. Om avbrottet sker efter en INPUT-fråga, fortsätter programmet med att visa frågan (eller ?) igen.

CONT används vanligen tillsammans med STOP vid felsökning i program. När programmet avbryts kan tillfälliga värden studeras och ändras med direktkommandon. Körningen kan återupptas med CONT eller ett direkt GOTO som startar programmet på ett angivet radnummer.

CONT kan inte användas om programmet har förändrats under avbrottet (felmeddelande "Can't continue").

```
Exempel:     10  INPUT A,B,C
             20  K = AÜ2*5.3 : L = BÜ3/.26
             30  STOP
             40  M = C*K+100 : PRINT M
             RUN                          Efter inmatning av
                                          tre
             ? 1,2,3                      tal fortsätter
                                          programmet.
             BREAK IN 30                  Stopp på rad 30.
             Ok
             PRINT L                      Kontroll av värdet
                                          på L
               30.7692
             Ok
             CONT                         Fortsättning på
                                          rad 40.
               115.9
             Ok
```

2.1.3 DELETE

Format: DELETE [<radnummer>][-<radnummer>]

Funktion: Raderar programrader

Användning: Med detta kommando kan radering av en eller flera programrader göras. BASIC återvänder alltid till kommandonivån efter det att DELETE har utförts. Om <radnummer> inte finns fås felmeddelandet "Illegal function call". Ett av radnumren kan ersättas med en punkt och då placeras den sist bearbetade radens nummer in.

```
Exempel:     DELETE 40          Raderar rad 40

             DELETE 40-100      Raderar raderna från och med
                                40 till och med 100

             DELETE -40         Raderar raderna från och med
                                den första till och med 40
```

2.1.4 END

Format: END

Funktion: Avslutar programmet, stänger alla filer och återgår till kommandonivån.

Användning: Instruktionen END får placeras var som helst i programmet. Till skillnad från instruktionen STOP skrivs inte meddelandet "BREAK IN" ut. Det är inte nödvändigt att skriva END sist i ett program.

```
Exempel:     520  IF K > 1000 THEN END ELSE GOTO 20
```

2.1.5 LIST

Format: LIST [<radnummer>][- [<radnummer>]]

Funktion: Kommandot LIST visar programraderna i programmet som finns i arbetsminnet på skärmen.

Användning: <radnummer> är ett befintligt radnummer mellan 0 och 65529. Om <radnummer> utelämnas visas hela programmet. Om endast <radnummer> anges visas endast denna rad. Visningen kan, som alla andra utmatningar på bildskärmen, avbrytas med tangenten <STOP>. En tryckning till på tangenten fortsätter visningen. Visningen avbryts helt med <CTRL>-<STOP>.

Exempel:

LIST	Visar programmet i arbetsminnet
LIST 500	Visar rad 500
LIST 150-	Visar alla rader från 150 till slutet på programmet
LIST -1000	Visar alla rader från den första till rad 1000
LIST 150-1000	Visar raderna 150 till 1000 (även 150 och 1000)
LIST .	Visar senast behandlade rad (kan användas t.ex vid felsökning)

2.1.6 LLIST

Format: LLIST [<radnummer>][-[<radnummer>]]

Funktion: Skriver ut programraderna på skrivare.

Användning: LLIST förutsätter en skrivare med 132 teckens bredd.

BASIC återvänder alltid till kommandonivån efter det att LLIST har utförts. Valmöjligheterna för LLIST är motsvarande som för LIST.

2.1.7 MERGE

Format: MERGE <filnamn>

Funktion: Slår ihop angiven ASCII-fil på ett kassettband med programmet i arbetsminnet.

Användning: <filnamn> är det namn som användes när filen sparades. Filen måste ha sparats i ASCII-format (genom att sätta A efter <filnamn> ex SAVE "1:HEJ",A), annars fås felmeddelandet "Bad file mode".

Om rad(er) i filen har samma radnummer som rad(er) i programmet i minnet, kommer raden(raderna) från filen att ersätta motsvarande rader i minnet. (MERGE motsvarar "inskjutning" av programrader i programmet i minnet). BASIC återgår alltid till kommandonivå efter det att ett MERGE-kommando har utförts.

Exempel: MERGE "ANTAL" slår ihop programmet i minnet med programmet "ANTAL"

2.1.8 NEW

Format: NEW

Funktion: Raderar programmet i arbetsminnet och nollställer alla variabler.

Användning: NEW anges som direktkommando för att rensa minnet innan ett nytt program skrivs in. BASIC återgår till kommandonivå efter NEW. Alla öppna filer stängs.

2.1.9 RENUM

Format: RENUM [[<nytt nummer>][,[<gammalt nummer>] [,<steg>]]]

Funktion: Omnumrering av programrader.

Användning: <nytt nummer> är det första radnumret som användes i den nya numreringen. Standardvärdet är 10. <gammalt nummer> är den rad i programmet där omnumreringen skall starta. Standarvärdet är första radnumret i programmet. <steg> är ökningen i den nya radnumreringen. Standardvärdet är 10.

RENUM ändrar också alla hänvisningar till radnummer som följer efter GOTO, GOSUB, THEN, RESTORE, RETURN, ON...GOTO, ON...GOSUB och ERL så att de stämmer med nya radnummer. Om en rad som inte finns kommer efter någon av dessa instruktioner fås felmeddelandet "Undefined line xxx in yyy". Den felaktiga radhänvisningen (xxx) ändras inte av RENUM men radnumret yyy kan ändras.

Observera: Renum kan inte användas för att ändra ordningsföljden av programrader. Om man t.ex skriver RENUM 15,30 när programmet har raderna 10, 20 och 30 fås ett felmeddelande ("Illegal function call"). Samma sak inträffar om man begär radnummer större än 65529 (se även funktionen ERL).

Exempel:

RENUM	Numrerar om hela programmet. De nya radnumren blir 10, 20, 30,....
RENUM 300,,50	Numrerar om hela programmet. Första nya radnummer blir 300. Stegningen blir 50 (radnummmer 300, 350, 400, 450,....).
RENUM 1000,900,20	Numrerar om raderna från rad nummer 900 och uppåt så att de i stället börjar på 1000 och stegar 20 (radnummer 1000, 1020, 1040, ...).

2.1.10 RUN

Format: RUN <namn>[[.]<filtyp>][, <radnummer>]

Funktion: Utför det program som finns i datorns arbetsminne.

Användning: Om <radnummer> anges börjar programkörningen på denna rad. Om inte rad nummer anges börjar programmet på lägsta radnummer. När RUN är utfört dvs programmet är genomfört, återgår BASIC till kommandoläge.

2.1.11 STOP

Format: STOP

Funktion: Avbryter programkörningen

Användning: STOP-instruktionen kan användas på valfritt ställe i ett program för att avbryta exekveringen. När programmet påträffar ett STOP skrivs följande meddelande ut:

```
BREAK IN nnnn
```

där nnnn är radnumret vid avbrottet.

Till skillnad från instruktionen END, stängs inga filer och variablerna behåller sina värden. Programmet kan fortsättas med kommandot CONT om ingen ändring av programmet gjorts.

Exempel: Se CONT.

2.2 INSTRUKTIONER

2.2.1 CLEAR

Format: CLEAR [[<uttryck 1>][,<uttryck 2>]]

Funktion: Används för att reservera utrymme i minnet för textsträngar samt för att bestämma slutadressen i minnet.

Användning: <uttryck 1> anger totalt utrymmesbehov för samtliga i programmet förekommande teckensträngar och <uttryck 2> anger slutadressen för minnet (egentligen den sista fria minnescellen).

Exempel:
```
CLEAR
CLEAR 100
CLEAR 100, &HF000
CLEAR,&HF00
```

2.2.2 DATA

Format: DATA <konstant>[,<konstant>].....

Funktion: Lagrar numeriska och strängkonstanter som läses med READ-satser i programmet.

Användning: DATA-satser utförs inte och kan placeras varsomhelst i programmet. En DATA-sats får innehålla så många konstanter som får plats på en rad (skilda åt av kommatecken) och valfritt antal DATA-satser får användas i ett program. READ-satserna läser DATA-satserna i radnummerordning. DATA kan betraktas som en kontinuerlig lista med värden, oberoende av hur många värden det finns på varje rad eller var de är placerade i programmet.

<konstant> får innehålla konstanter med valfritt format, dvs decimaltal, flyttal eller heltal (numeriska uttryck är dock inte tillåtna i listan). Strängkonstanter i DATA-satser måste omges med citationstecken om de innehåller kommatecken, kolon eller mellanslag i början eller slutet. I andra fall är citationstecken inte nödvändiga.

Variabeltypen (numerisk eller sträng) som anges i READ-satsen måste stämma överens med motsvarande konstant i DATA-satsen. Se READ, RESTORE.

Exempel:
```
DATA 1,2,3
DATA SMITH, "555.1212", 11018
```

2.2.3 DEF FN

Format: DEF FN<namn>[(<parameterlista>)] = <funktions-definition>

Funktion: Definierar och namnger en funktion som skrivs av programmeraren.

Användning: <namn> måste vara ett tillåtet variabelnamn, Detta namn, föregånget av FN blir namnet på funktionen. <parameterlista> består av de variabelnamn som ersätts med variabelvärden när funktionen anropas. Variabelnamnen skiljs åt av kommatecken. <funktionsdefinition> är det uttryck som utför operationen i funktionen. Det är begränsat till en rad. Variabelnamnen som finns i uttrycket har endast uppgiften att definiera funktionen, de påverkar inte programvariabler som har samma namn. Ett variabelnamn som används i en funktionsdefinition kan eller kan inte förekomma i parameterlistan. Om det förekommer, tilldelas det ett värde när funktionen anropas. Annars används det aktuella värdet på variabeln.

Variablerna i parameterlistan representerar, i tur och ordning, de variabler eller variabelvärden som anges vid anropet av funktionen.

Den användardefinierade funktionen kan vara i numerisk form eller i strängform. Om en viss form anges i funktionsnamnet antas alltid denna form innan värdet returneras till den anropande instruktionen. Om specifikation och aktuell form inte stämmer fås felmeddelandet "Type mismatch".

En DEF FN-instruktion måste utföras innan den definierade funktionen anropas. Om en funktion anropas innan den har definierats fås felmeddelandet "Undefined user function". DEF FN kan inte användas som direktkommando.

Exempel:

```
 50  DEF FNPYT(A,B)=SQR(AÜ2+BÜ2)
 100 B=4.0
 110 C=FNPYT(3.0,B)
 120 PRINT C
 RUN
  5
 Ok

 10 DEFFNADR$(PNR%,O$)=LEFT$(STR$(PNR%),3)-
    +" "+RIGHT$(STR$(PNR%),2)+"  "+O$
 20  P%=10022
 30  L3$=FNADR$(P%,"STOCKHOLM")
 40  PRINT L3$
 RUN
  100 22  STOCKHOLM
 Ok
```

2.2.4 DEFINT, DEFSNG, DEFDBL, DEFSTR

Format: DEF <typ>, <teckenområde>
där <typ> är INT, SNG, DBL eller STR.

Funktion: Deklarerar variabler vars namn börjar på bokstäver som anges av
<teckenområde> som heltal, enkel precision, dubbel precision eller strängar.

Användning: Om en variabel deklareras i programmet och samma variabel finns deklarerad med instruktionen DEF-typ gäller alltid deklarationen vid DEFtyp.

Om ingen variabel deklarerats antar BASIC alltid att variabeln har typen enkel precision.

Exempel:

10 DEF DBL L-P	Alla variabler som börjar med bokstäverna L, M, N och P har dubbel precision
10 DEFSTR A	Alla variabler som börjar med A är strängvariabler
10 DEFINT I-N,W-Z	Alla variabler som börjar med bokstäverna I, J, K, L, M, N, W, X, Y och Z är heltalsvariabler.

2.2.5 DIM

Format: DIM <variabel> (<index>[,<index>]...)
[,<variabel>(<index>[,<index>]...)]....

Funktion: Anger högsta antalet värden för indexerade listvariabler och reserverar motsvarande minnesutrymme.

Användning: Om en listvariabel används utan att vara dimensionerad med DIM, antas högsta tillåtna värde på index vara 10. Om ett index som är större än det högsta angivna används fås felmeddelandet "Subscript out of range". Minimivärdet är alltid 0. Om inget DIM anges så är maximala antalet index 10. Instruktionen DIM nollställer alla variabelvärden i de angivna listorna (för strängar anges då tomma strängar). Om försök görs att flera gånger dimensionera en lista, utan att dessförinnan ERASE angivits, fås felmeddelandet "Redimensioned array". Index är alltid ett numeriskt uttryck (se avsnittet om listvariabler, 1.5.2).

Exempel:
```
10  DIM A(5)        dimensionerar listan A för
                    6 element med enkel preci-
                    sion
20  FOR I=0 TO 5    inläsning av elementen
30  READ A(I)
40  NEXT I
50  DATA 1,3,7,3,2,1

10  DIM A(30),S$(5),M#(8,8)
```

Ovan dimensioneras följande listor:
en lista A med 31 element med enkel precision, en lista S med 6 strängar och en matris M med dubbel precision och 9 gånger 9 element.

2.2.6 ERASE

Format: ERASE <variabelnamn>[,<variabelnamn>]...

Funktion: Raderar (eliminerar) listvariabler i ett program

Användning: Listor kan dimensioneras om efter det att de eliminerats med ERASE eller också kan det tidigare tilldelade minnesutrymmet användas för andra ändamål. <variabelnamn> måste vara namn på listvariabler. Om man försöker att dimensionera om en lista utan att ha eliminerat den, fås felmeddelandet "Redimensioned array".

Exempel:
```
10   DIM A(30),B(50)
        *
        *
450  ERASE A,B
460  DIM A(15),B(99)
        *
```

2.2.7 ERROR

Format: ERROR <heltalsuttryck>

Funktion: Simulerar förekomsten av ett fel

Användning: Värdet på <heltalsuttryck> måste vara större än eller lika med 0 men mindre än eller lika med 255. Om värdet är lika med en felkod som redan finns i BASIC kommer ERROR-instruktionen att ge samma resultat och motsvarande felmeddelande skrivs ut. Om en ERROR-instruktion anger en felkod som det inte finns något felmeddelande för, skrivs meddelandet "Unprintable error" ut. Utförande av en felrutin för vilken det inte finns någon felhanteringsrutin medför att felmeddelande skrivs ut och programmer stoppar. Instruktionen ON ERROR kan också användas för fel som angivits med ERROR x.

Exempel:
```
10 ERROR 15
RUN
String too long in line 10

ERROR 15
String too long

ERROR 255
UNPRINTABLE ERROR
```

2.2.8 FOR...NEXT

Format:
```
FOR <variabel> = <start> TO <slut> [STEP
<steg>]
        *
        *
NEXT [<variabel>][,<variabel>...]
```

Funktion: Ger programslingor (loopar) där satser genomförs ett antal gånger.

Användning: <variabel> används som räknare. Det första numeriska uttrycket <start> är startvärdet för räknaren. Det andra uttrycket <slut> är slutvärdet. Programraderna som kommer efter FOR-instruktionen utförs tills NEXT-instruktionen påträffas. Räknaren ökas sedan med värdet angivet av STEP. En kontroll utförs om värdet på räknaren är större än slutvärdet <slut>. Om det inte är större hoppar programmet tillbaka till satsen efter FOR-instruktionen och processen upprepas. Om värdet är större, fortsätter programmet med satsen efter NEXT. Detta är en FOR...NEXT-slinga. Om STEP inte anges, antas det vara 1. Om STEP är negativt blir slutvärdet på räknaren mindre än startvärdet. Räknaren räknas då ner varje gång slingan genomlöps och slingan utförs till dess att räknaren har ett värde som är lägre än slutvärdet.

Instruktionerna som står mellan FOR och NEXT utförs alltid en gång, även om slutvillkoret är uppfyllt redan vid instruktionen FOR.

Inkapslade slingor
FOR...NEXT-slingor kan vara inkapslade vilket innebär att en FOR...NEXT-slinga kan placeras inom en annan slinga. När slingor är kapslade måste räknarna ha unika variabelnamn i varje slinga. NEXT-instruktionen för en kapslad slinga måste komma före NEXT för den yttre slingan. Om kapslade slingor har samma slutpunkt kan ett enda NEXT användas för samtliga.

Exempel:
```
NEXT A,B,C
```

Variabeln i NEXT-instruktionen kan uteslutas och NEXT gäller då för den senast påträffade FOR-instruktionen. Det är dock god programmeringssed att alltid ange variabeln. Om ett NEXT påträffas utan att ett motsvarande FOR har passerats fås ett "NEXT without FOR"-felmeddelande och programkörningen avbryts.

Exempel 1:

```
10  K = 10
20  FOR I=1 TO K STEP 2
30 PRINT I;
40  K = K + 10
50  PRINT K
60  NEXT
RUN
 1     20
 3     30
 5     40
 7     50
 9     60
Ok
```

Exempel 2:

```
10  I = 5
20  FOR I = 1 TO I + 5
30  PRINT I;
40  NEXT
RUN
 1   2  3   4   5   6   7   8   9   10
Ok
```

I detta exempel genomlöps slingan tio gånger. Det slutliga värdet på slingvariablen (räknaren) bestäms varje gång innan startvärdet bestäms.

2.2.9 GOSUB...RETURN

Format:

```
GOSUB <radnummer>
        *
        *
        *
RETURN [<radnummer>]
```

Användning: Hoppar till och återvänder från en subrutin (underprogram).

Användning: <radnummer> är första raden i subrutinen.

En subrutin kan anropas flera gånger i ett program och en subrutin kan anropas inuti en annan subrutin. Sådan kapsling av subrutiner begränsas endast av tillgängligt minne.

Om <radnummer> utelämnas vid RETURN medför RETURN-instruktionen i subrutinen att programmet

återgår till satsen som följer efter den senast utförda GOSUB-instruktionen. En subrutin får innehålla mer än en RETURN-instruktion om logiken kräver återgång på olika ställen i subrutinen. Subrutiner får förekomma var som helst i programmet men det är lämpligt att subrutinerna lätt kan skiljas från från huvudprogrammet. För att förebygga oönskad ingång i en subrutin kan den föregås av STOP-, END- eller GOTO-instruktioner som styr programmet runt subrutinen.

Det är inte tillåtet att använda GOTO för hopp ut ur en subrutin. Detta beror på att GOSUB sparar återhoppsadressen i minnet. Detta läses sedan tillbaka av RETURN. Om GOTO används för uthopp kommer ett senare RETURN att medföra återgång till fel plats i programmet. Använd i stället RETURN nnnn.

Exempel:

```
10 GOSUB 100:                'Anrop av subrutin
20 PRINT "Tillbaka ";:       'på rad 100
30 PRINT "från subrutinen"
40 END
50     REM
60     REM
100 PRINT "Nu ";
110 PRINT "genomförs ";
120 PRINT "subrutinen."
130 RETURN:              'Återhopp till rad 20
RUN
Nu genomförs subrutinen
Tillbaka från subrutinen
Ok
```

Om rad 130 i exemplet ovan hade ersatts med
130 RETURN 30
så skulle programkörningen sett ut så här:

```
RUN
Nu genomförs subrutinen
från subrutinen
```

Denna utskrift kommer sig av att återhoppet inte går till rad 20 utan till rad 30 och rad 20 utföra inte.

2.2.10 GOTO

Format: GOTO <radnummer>

Funktion: Ovillkorligt hopp till annan programrad än den nästföljande.

Användning: Om <radnummer> är en utförbar sats utförs denna och påföljande satser. Om det inte är en utförbar sats (t.ex DATA) fortsätter programmet med första utförbara sats som påträffas efter <radnummer>.

Exempel:
```
10 PRINT "DENNA TEXT ";
20 GOTO 40
30 PRINT "SKULLE VARA LÅNG MEN"
40 PRINT "BLEV KORT MED GOTO"
RUN
DENNA TEXT BLEV KORT MED GOTO
```

2.2.11 IF

Format 1: IF <villkor> THEN <instruktion(er)> / <radnummer> [ELSE <instruktion(er)> / <radnummer>]

Format 2: IF <villkor> GOTO <radnummer> [ELSE <instruktioner(er)> / <radnummer>]

Funktion: Villkorlig styrning av programflödet (ordningsföljden mellan programraderna).

Användning: Först beräknas <villkor> (se även logiska jämförelseoperatorer). Om resultatet inte är noll så verkställs THEN eller GOTO-instruktionerna. THEN kan antingen följas av ett radnummer för hopp eller fler satser som skall utföras. GOTO följs alltid av ett radnummer. Om resultatet av <villkor> är noll bortses från THEN eller GOTO-instruktionerna och ELSE-instruktionen (om den är angiven) utförs. Programkörningen fortsätter med nästa utförbara sats.

Kapsling av IF-satser
IF...THEN...ELSE-satser kan kapslas. Kapsling begränsas endast av radlängden (som får vara maximalt 255 tecken). Till exempel:

```
IF X>Y THEN PRINT "STÖRRE ELSE IF Y>X THEN
PRINT "MINDRE ÄN" ELSE PRINT "LIKA MED"
```

som är en tillåten sats. Om satsen inte innehåller lika antal ELSE och THEN paras varje ELSE ihop med närmaste oparade THEN. Så till exempel:

```
IF A=B THEN IF B=C THEN PRINT "A=C" ELSE
PRINT "A<>C"
```

kommer inte att ge utskriften "A<>C" när A<>B.

Om en IF...THEN-sats följs av ett radnummer i ett direktkommando fås ett "Undefined line"-fel om inte en sats med angivet radnummer tidigare har skrivits in indirekt (som programrad).

Anmärkning: När IF används för att jämföra likhet med ett värde som är resultat av en flyttalsberäkning, kom då ihåg att datorns representation av värdet inte alltid är exakt. Därför måste jämförelsen göras med det intervall inom vilket värdet kan ligga. Om man till exempel vill jämföra ett beräknat värde A med 1.0 skrivs:

```
IF ABS(A-1.0) < 1.0E-6 THEN...
```

Denna jämförelse är sann om värdet på A är 1.0 med ett relativt fel som är mindre än 1.0E-6.

Exempel 1:

```
200 IF I THEN GET#1,I
```

Denna sats hämtar post nummer I om I inte är 0.

Exempel 2:

```
100 IF (I<20)*(I>10) THEN DB=1979-I : GOTO 300
110 PRINT "UTANFÖR OMRÅDET"
      *
      *
```

I detta exempel jämförs om I är större än 10 och mindre än 20. Om I är inom detta talområde beräknas DB och programmet går till rad 300. Om I är utanför området utförs rad 110.

Exempel 3:

```
210 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$
```

Denna sats får utskriften att gå antingen till skärmen eller till skrivaren beroende på värdet hos variabeln IOFLAG. Om denna är noll skrivs A$ ut på skrivaren, annars på skärmen.

2.2.12 LEFT$

Format: <variabel> = LEFTT$ (<variabel2>, <antal>)

Funktion: Tilldelar <variabel> de <antal> vänstra tecknen i <variabel 2>

2.2.13 LET

Format: [LET]<variabel> = <uttryck>

Funktion: Tilldelar <variabel> ett värde (<uttryck>).

Användning: Ordet LET kan utelämnas dvs likhetstecknet är tillräckligt för att tilldela ett värde till en variabel.

Exempel:

```
10 LET I%=100
20 K%=300/2
30 PRINT I%;K%
40 LET A$="abcdef"+"xyz"
50 PRINT A$;" ";A%
RUN
 100     150
abcdefxyz  0
Ok
```

2.2.14 MID$

Format: MID$ (<variabel>, <start>[,<antal>]) = <uttryck>

Funktion: Ersätter en del av en teckensträng med en annan teckensträng.

Användning: <variabel> och <uttryck> måste vara strängutryck, medan <antal> och <start> måste vara numeriska uttryck. Tecknen i <variabel>, med början i position <start> ersätts med tecknen i <uttryck>. Det valfria <antal> anger det antal tecken från <uttryck> som används vid utbytet. Om <antal> utelämnas används hela <uttryck>. Oberoende av om <antal> tas med eller inte ersätts aldrig fler tecken än vad som finns i <variabel>.

Exempel:

```
10 A$ = "KANSAS CITY,MO"
20 MID$(A$,13) = "KS"
30 PRINT A$
RUN
KANSAS CITY,KS
```

MID$ kan också används som en funktion som ger en delsträng av en given sträng.

```
A$ = MID$(Y$,6,3)
```

2.2.15 ON...GOSUB och ON...GOTO

Format: ON <uttryck> GOTO <lista med radnummer>
ON <uttryck> GOSUS <lista med radnummer>

Funktion: Medför hopp till ett av flera angivna radnummer beroende på vilket värde som fås när ett uttryck beräknas.

Användning: Värdet på <uttryck> anger till vilken rad i listan som hopp sker vid förgrening. Exempel: Om värdet är tre görs ett hopp till det tredje radnumret. Om värdet inte är ett helta (flyttal) avrundas talet. I en ON... GOSUB-sats måste varje radnummer referera till första raden i en subrutin. Om värdet av <uttryck> är noll eller större än antalet radnummer i listan (men mindre än eller lika med 255) fortsätter BASIC med nästa utförbara sats. Om värdet av <uttryck> är negativt eller större än 255 fås felmeddelandet "Illegal function call".

Exempel:

```
10 I%=2
20 ON I%+2 GOTO 50,60,70,80,90,100
30 PRINT "I>4 eller I=-2"
40 END
```

```
 50 PRINT "I=-1" : GOTO 110
 60 PRINT "I=0" : GOTO 110
 70 PRINT "I=1" : GOTO 110
 80 PRINT "I=2" : GOTO 110
 90 PRINT "I=3" : GOTO 110

100 PRINT "I=4 är det sista värdet som
    hopp görs till"
110 I=I+1
120 GOTO 20
RUN
I=2
I=3
I=4 är det sista värde som hopp görs till
I>4 eller I=-2
```

2.2.16 READ

Format: READ <variabel> [,<variabel>],...

Funktion: Läser värden från DATA-satser och tilldelar dem till variabler.

Användning: En READ-sats måste alltid användas tillsammans med en DATA-sats. READ tilldelar variablerna i listan i tur och ordning värden från DATA-satserna. Variablerna som läses av READ kan vara numeriska eller strängar men måste överensstämma med specificerade variabeltyper. Om de inte överensstämmer, fås felmeddelandet "Syntax error". Se vidare under DATA och RESTORE.

2.2.17 REM

Format: REM <anmärkning>

Funktion: Medger att förklarande text placeras in i programmet.

Användning: REM-satser utförs inte men skrivs ut exakt som de är vid LISTning av programmet. Hopp till REM-satser kan göras (med GOTO eller GOSUB). Programmet fortsätter därvid med den första utförbara raden efter REM-satsen.

Anmärkningar kan läggas till i slutet på en rad genom att skriva en apostrof (') i stället för REM. Detta tar dock mer minnesutrymme än en vanlig REM-sats.

OBS: Använd inte REM i en DATA-sats eftersom det då skulle kunna betraktas som giltiga data.

Exempel:
```
10 REM BERÄKNA GENOMSNITTSHASTIGHETEN
20 FOR I = 1 TO 20
30 SUM = SUM + V(I)     ' V(I) är hastigheten
        *
```

2.2.18 RESTORE

Format: RESTORE [<radnummer>]

Funktion: Medför att DATA-rader kan läsas om igen från angivet radnummer.

Användning: När RESTORE utförs, läser READ-satsen den första variabeln i den första DATA-raden i programmet. Om <radnummer> anges, läser nästa READ-sats den första variabeln i angiven DATA-rad. Om inte RESTORE används och alla DATA-satserna lästs en gång, kommer felmeddelandet "Out of DATA" att visas (data slut).

Exempel:
```
 10  READ A,B          Inläsning av data från rad
                       50
 20  RESTORE           Återställning av läsningen
                       till rad 50
 30  READ C,D,E        Förnyad inläsning från rad
                       50
 40  PRINT A;B;C;D;E
 50  DATA 57,68,79
 RUN
  57   68   57   68   79
 Ok

 10  RESTORE 70        Läsning från rad 70
 20  READ A$, B$       Inläsning av data
 30  RESTORE           Återställer läsningen till
                       rad 60 dvs första datara-
                       den i programmet
 40  READ C,D
 50  PRINT A$;B$;C;D
 60  DATA 57,68,79
 70  DATA "DATA 70",70
 RUN
 DATA 70 70  57  68
```

2.2.19 SWAP

Format: SWAP <variabel>,<variabel>

Funktion: Byter värdena på två variabler.

Användning: Alla typer av variabler kan bytas (heltal, enkel och dubbel precision, sträng) men de bägge variablerna måste ha samma typ eftersom annars ett "Type mismatch"-fel uppstår.

Exempel:
```
 10  A$ = "EN" : B$ = "ALLA" : C$ = "FÖR"
 20  PRINT A$,C$,B$
 30  SWAP A$,B$
 40  PRINT A$,C$,B$
 RUN
  EN FÖR ALLA
  ALLA FÖR EN
 Ok
```

2.2.20 RIGHT$

Format: <variabel> = RIGHT$ (<variabel2>, <antal>)

Funktion: Tilldelar <variabel> de <antal> högra tecknen i <variabel 2>

2.2.21 TRON/TROFF

Format:
```
TRON
TROFF
```

Funktion: Skriver ut radnumren på de programrader som genomlöps i programmet.

Användning: Instruktionen TRON, angiven som direktkommando eller i ett program, ger utskrift av de programrader som programmet genomlöper och används för felsökning. Radnumren skrivs ut inom fyrkantparenteser (Ä och Å om svenska tecken används). Funktionen TRON kopplas bort med TROFF (eller när kommandot NEW ges).

Exempel:
```
 10  K=10
 20  FOR J=1 TO 2
 30  L=K+10
 40  PRINT J;K;L
 50  K=K+10
 60  NEXT
 TRON
 Ok
 RUN
 [10] [20] [30] [40] 1 10 20
                 genomlöpta rader skrivs ut inom
                 []
 [50] [60] [30] [40] 2 20 30
                 därefter sker utskrift med
                 PRINT
 [50] [60] [70]
 Ok
 TROFF           bortkoppling av radutskrift
 Ok
```

2.3 INSTRUKTIONER FÖR IN- OCH UTMATNING

Lagring av program och data i BASIC på ett lagringsmedium (kassettbandspelare, flexskivenhet) görs med angivande av ett <filnamn>. Den lagrade informationen kan sedan hämtas tillbaka vid en senare tidpunkt, även om lagringsenheten varit urkopplad under tiden. Regler för filnamn finns i början av kapitel 2.

2.3.1 BEEP

Format: BEEP

Funktion: Ger en pipton i högtalaren.

Användning: En ton med c:a 1000 Hz frekvens och 1/4 sekunds varaktighet ges. Kommandona PRINT CHR$(7) och <CTRL>-G har samma funktion.

Exempel: -

2.3.2 BLOAD

Format: BLOAD <filnamn> [,R][,<adress>]

Funktion: Inladdning av en binär fil till arbetsminnet.

Användning: Filen måste vara lagrad med kommandot BSAVE. <adress> är den minnesadress där lagringen av filen i minnet påbörjas. Anges inte <adress> laddas filen in på den adress från vilken den lagrades med BSAVE. Man måste vid användning av BLOAD kontrollera att inga systemvariabler skrivs över, eftersom ingen automatisk kontroll görs. Värdet på <adress> får inte vara större än 65535. Med tillägget R (som betyder RUN) utförs programmet direkt efter inladdning (gäller maskinspråksprogram).

Exempel:		
	BLOAD "Bin01",R	Laddning och körning av assemblerprogrammet i filen "Bin01"
	BLOAD "Mem",,&HC000	Laddning av innehållet i filen "Mem" från och med adressen &HC000 i arbetsminnet.

2.3.3 BSAVE

Format: BSAVE <filnamn> , <start> , <slut> [,<början>]

Funktion: Lagring av innehållet i arbetsminnet på en fil

Användning: <start> är ett numeriskt uttryck mellan 0 och 65535 och som anger den adress i arbetsminnet från vilken innehållet skall lagras på en fil.
<slut> är ett uttryck som ovan som anger adressen till den sista minnesposition som skall skrivas på filen.
<början>, också ett numerisk uttryck, betecknar adressen på vilken exekveringen av maskinspråksprogrammet som är lagrat i filen skall starta. Standardvärdet för <början> är <start>.

Exempel: BSAVE "Mem",&HC000,&HC203 Lagring av minnesinnehållet från &HC000 till &HC203 i filen "Mem"

2.3.4 CLICK

Format: CLICK ON / OFF

Funktion: In- och urkoppling av tangentbordsklick.

Användning: -

Exempel:
```
CLICK ON
Ok
CLICK OFF
Ok
```

2.3.5 CLOAD

Format: CLOAD [<filnamn>]

Funktion: Laddar in en fil från kassettspelaren.

Användning: Om <filnamn> anges, söker CLOAD efter en fil med detta namn. Om <filnamn> inte anges, laddas nästa fil som finns på bandet in i arbetsminnet. Om det är en bild som skall visas på skärmen, ställs skärmen in i rätt läge. Om det är ett program i BASIC raderas befintligt program i minnet och det nya programmet laddas in. Skulle ett fel uppstå under inladdningen av ett program, fås felmeddelandet "Device I/O error". Samma felmeddelande fås om inmatningen avbryts med <CTRL>-<STOP>.

Exempel:		
	CLOAD	Laddar in nästa fil från bandet i arbetsminnet
	CLOAD "test"	Söker efter och laddar in filen "test" i arbetsminnet

2.3.6 CLOAD?

Format: CLOAD? [<filnamn>]

Funktion: Verifiering av program i arbetsminnet.

Användning: CLOAD? verifierar (kontrolläser) att programmet på kassettbandet är det samma som finns i minnet. Om så inte är fallet fås felmeddelandet "Verify error".

Exempel:		
	CLOAD?	Jämför nästa fil med programmet i minnet
	CLOAD?"hej"	Söker efter filen "hej" på bandet och jämför med minnesinnehållet

2.3.7 CLOSE

Format: CLOSE [[#]<filnummer>[,[#]<filnummer>]...]

Funktion: Avslutar in- eller utmatning till eller från en fil på skiva.

Användning: <filnummer> skall vara det nummer med vilket filen öppnades. CLOSE utan filnummer stänger alla öppna filer.

Knytningen mellan en viss fil och ett filnummer upphör när filen stängs med CLOSE. Filen kan nu öppnas på nytt med ett annat nummer. På samma sätt kan samma nummer användas för att öppna en annan fil. CLOSE vid sekventiell fil skriver ut sista bufferten. Instruktionen END och kommandot NEW stänger alla filer automatiskt (STOP stänger inte några filer).

Exempel: CLOSE

CLOSE #1,#5,#7

Utförligt exempel finns i bilagan.

2.3.8 CLS

Format: CLS

Funktion: Rensar bildskärmen

Användning: CLS rensar bildskärmen från text och bild och återför markören till "hemmaläget" (övre vänstra hörnet. Samma funktion som CLS har instruktionerna SCREEN 0 och PRINT CHR$(12) samt tangenterna <CTRL>-L och <CLS/HM>

Exempel: -

2.3.9 COPY

Format 1: COPY <enhetsnummer> FROM <enhetsnummer> / <filnamn>

Format 2: COPY <enhetsnummer> ; (<spår> , <sektor>) FROM <enhetsnummer>; (<frånspår>,<sektor>) - (<tillspår> , <sektor>)

Version: Endast vid flexskivor

Funktion: Kopiering av filer eller skivsektorer

Användning: Kopiering mellan spår 0 och andra spår är inte tillåten eftersom spår 0 är annorlunda formaterat.

Varning! Vid kopiering av sektorer kan filerna förstöras om fel kommandon ges.

Exempel:

COPY 2 FROM 1	kopierar hela skivan från skivenhet 1 till skivenhet 2
COPY 2 FROM "1:hjälp"	kopierar filen "hjälp" från skivenhet 1 till skivenhet 2
COPY 1;(10,10) FROM 1;(30,2)-(30,8)	kopierar sektorerna (30,2) till (30,8) från skivan i enhet 1 till sektorerna (10,10) på samma skiva

2.3.10 CSAVE

Format: CSAVE <"filnamn">[,S]

Funktion: Lagring av program eller skärmbilder på kassett

Användning: Det i arbetsminnet befintliga programmet sparas på kassetten. Man måste kontrollera att plats finns på kassetten eftersom eventuella program raderas där det nya spelas in. Om tillägget S anges sparas skärmbilden. Hela videominnets innehåll (den använda delen), totalt 16 K samt aktuell skärminställning sparas.

Exempel:

CSAVE "prog"	Lagrar innehållet i arbetsminnet under namnet "prog"
CSAVE "bild",S	Spara den aktuella (visade) skärmbilden under namnet "bild".

2.3.11 DSKO$

Format: DSKO$ <skivenhetsnummer>, <spår>, <sektor>

Version: Endast för flexskivor

Funktion: Skriver en sektor på skivan

Användning: De data som DSKO$ skall skriva på skivan finns i en FIELD-buffert. Innan instruktionen DSKO$ ges, måste en FIELD-instruktion med <filnummer> 0 (noll) utföras som framställer en buffert med 256 bytes (minst två strängvariabler) och data måste flyttas till denna buffert.
DSKO$ skriver sedan data på den angivna sektorn i <skivenhet>.
Användning av denna instruktion fordrar tillräckliga kunskaper om datastrukturerna på skivan eftersom filerna och/eller biblioteket på skivan annars kan förstöras.

Exempel: Detta exempel är bara till för att förklara användningen och skall absolut inte utföras eftersom då i vissa fall filer kan förstöras.

```
10  FIELD #0, 128 AS A$, 128 AS B$
20  A$ = "=1233444455555"
30  DSKO$ 2,20,3
```

2.3.12 FIELD

Format: FIELD [#] <filnummer>, <storlek> AS <strängvariabel> [,<storlek> AS <strängvariabel>]...

Version: Endast för flexskivor

Funktion: Tilldelar utrymme för variabler i bufferten för en direktfil (kallas även random accessfil).

Användning: Innan data läses från en direktfil med GET eller data skrivs in i filen med PUT, måste en FIELD-instruktion utföras. <filnummer> är det nummer som filen öppnades med (med OPEN). <storlek> är det antal (heltal) tecken som <strängvariabeln> tilldelas.

Totalt antal bytes som tilldelas i en FIELD-instruktion får inte överskrida den postlängd som bestämdes när filen öppnades. I så fall uppstår felet "Field overflow" (standardvärdet för postlängden är 128). Valfritt antal FIELD-instruktioner får utföras för samma fil och alla utförda FIELD-instruktioner gäller samtidigt. FIELD skriver INTE in några data i direktfilsbufferten (se LSET/RSET och GET).

Anmärkning: Använd inte variabelnamn som är definierade med FIELD i INPUT- eller LET-instruktioner. När en variabel är angiven med FIELD pekar den på rätt plats i direktfilsbufferten. Om ett påföljande INPUT eller LET-uttryck utförs flyttas variabelpekaren till fel plats.

Exempel:
```
...OPEN...#1
FIELD #1, 20 AS N$, 10 AS ID$, 40 AS ADD$
```

tilldelar de första 20 positionerna (bytes) i filbufferten till variabeln N$, nästa 10 positioner till ID$ och de följande 40 positionerna till ADD$. En post i denna fil blir således 70 bytes lång. Om man vill ha den längre måste man utföra en ny FIELD-instruktion.

2.3.13 FILES, LFILES

Format: [L]FILES [<skivenhet>]

Version: Endast för flexskivor

Funktion: Uppräkning av innehållsförteckningen (biblioteket) på en flexakiva.

Användning: FILES visar innehållsförteckningen på bildskärmen, LFILES skriver ut den på skrivare. Om <skivenhet> inte anges, visas biblioteket för skivenhet 1. Skiljetecknet mellan filnamn och filtyp har följande betydelse:

" "	fil eller program i ASCII-format
"."	programfil
"*"	fil med maskinspråksprogram
"#"	bildfil

Siffran efter filnamnet anger antal block som filen upptar.

Exempel: FILES 2 — Visar innehållsförteckningen för skivan i skivenhet 2 på skärmen.

2.3.14 GET

Format: GET [#] <filnummer>[,<postnummer>]

Version: Endast för flexskivor.

Funktion: Läser en post från en direktfil till en direktfilsbuffert.

Användning: <filnummer> är det nummer med vilket filen öppnades. <postnummer> är ett tal som bestäms med hänsyn till hur data matades in i filen, relativt filbörjan. Om <postnummer> utelämnas läses nästa post (efter det förra GET) in i bufferten. Största möjliga postnummer är 32767. Endast efter att en GET-instruktion har utförts kan INPUT# och LINE INPUT# användas för att läsa tecken från direktfilsbufferten.

2.3.15 INPUT

Format: INPUT [<sträng> ;] <variabel> [, <variabel>]...

Funktion: Hämtar in data från tangentbordet under programkörning.

Användning: När en INPUT-instruktion påträffas görs ett uppehåll i programkörningen och ett frågetecken skrivs ut för att visa att programmet väntar på data. Om <sträng> finns med skrivs denna ut före frågetecknet. Angivna data skrivs sedan in på tangentbordet. De data som skrivs in tilldelas variabeln(lerna) i <variabellistan>. Antalet måste vara likadant som i listan. Data skiljs åt av kommatecken.
Variablerna kan vara numeriska eller strängvariabler (även indexerade variabler). Typen måste överensstämma med den typ som anges av variabelnamnet (stränginmatning till en INPUT-sats behöver inte omges av citationstecken).
Om för många eller för få variabelvärden matas in som svar på en INPUT-sats eller om de är av fel typ (strängar i stället för numeriska värden osv) fås felmeddelandet "?Redo from start" och användaren får försöka igen ända tills ett giltigt värde anges. Ingen tilldelning till variabelvärden görs förrän godtagbara värden har matats in.

Exempel:
```
10 INPUT X
20 PRINT X "I KVADRAT ÄR"X↑2
30 END
RUN
? 5                          5 skrevs in av användaren som svar på frågetecknet)
  5 I KVADRAT ÄR 25
Ok

10 PI=3.14
20 INPUT "ANGE RADIEN";R
30 A=PI*R↑2
40 PRINT "CIRKELNS YTA ÄR";A
50 PRINT
60 GOTO 20
Ok
RUN
ANGE RADIEN? 7.4             användaren skriver 7.4
CIRKELNS YTA ÄR 171.946

ANGE RADIEN?
o.s.v.
```

2.3.16 INPUT#

Format: INPUT#<filnummer>,<variabellista>

Funktion: Läser in variabelvärden från en sekventiell fil och tilldelar dem till programvariabler.

Användning: <filnummer> är det nummer som angavs när filen öppnades med OPEN. <variabellista> innehåller variabelnamnen som tilldelas värden från filen (variabeltypen måste överensstämma med den typ som anges av variabelnamnen). Vid INPUT# skrivs inget frågetecken ut, som vid INPUT.
Datavärden i filen måste ligga som om de skrevs in som svar på en INPUT-sats. Vid numeriska värden ignoreras mellanslag före och efter, returer eller radmatningar. Första tecknet som påträffas och som inte är något av ovanstående antas vara första tecknet i talet. Talet avslutas med något av ovan angivna tecken eller kommatecken.

Motsvarande gäller om en teckensträng skall hämtas från filen. Om det första tecknet är ett citationstecken (") antas att teckensträngen består av alla tecken fram till nästa citationstecken. Detta innebär att citationstecken inte får finnas inne i strängen. Om filslutsmärke påträffas under läsning för INPUT avslutas läsningen av värdet och felmeddelandet "Input Past End" erhålls.

Exempel: Se bilaga.

2.3.17 IPL

Format: IPL <stränguttryck>

Version: Endast för flexskivor

Funktion: Inmatning av en autostartsträng

Användning: Den med IPL inmatade strängen utförs automatiskt vid start av systemet. Om <stränguttryck> är en tom sträng utförs ingen autostart.

Exempel: IPL "LOAD"+CHR$(34)+"1:demo"+CHR$(34)+",R"

Startar efter systemstart programmet som finns i filen "demo" i skivenhet 1.

2.3.18 KEY

Format: KEY <funktionstangentnummer> , <stränguttryck>

Funktion: Ändring av betydelsen hos funktionstangent

Användning: <funktionstangentnummer> är ett numeriskt uttryck mellan 1 och 10. <stränguttryck> är den text som skall visas på skärmen när motsvarande funktionstangent trycks ner. I detta <stränguttryck> kan också styrtecken som t.ex <ENTER>, <CTRL-L> eller <INS> läggas in men totala längden får inte överskrida 15 tecken. Skulle strängen vara längre används bara de första 15 tecknen.

Exempel: KEY 1, CHR$(12)+"LIST.-"+CHR$(13)

Efter denna omdefiniering visas ovanstående text på skärmen för funktionstangent 1 och när den trycks ner visas programmet från aktuell rad fram till programslut.
I exemplet ovan är CHR$(12) lika med <CTRL-L> och CHR$(13) lika med <ENTER>.

2.3.19 KEY LIST

Format: KEY LIST

Funktion: Visar de 10 funktionstangenternas definitioner på skärmen.

Anmärkning: För tecken som inte kan skrivas ut visas blanktecken.

2.3.20 KILL

Format: KILL <filnamn>

Version: Endast för flexskivor

Funktion: Raderar filer från flexskivor

Användning: Om försök göra att radera en öppnad fil, fås felmeddelandet "File already open" (filen öppnad).

Exempel: KILL "1:skrot" raderar filen "skrot" på skivenhet 1

2.3.21 LINE INPUT

Format: LINE INPUT [<sträng> ;] <strängvariabel>[;]

Funktion: Tar emot inmatning av en hel rad tecken (högst 254) till en strängvariabel.

Användning: <sträng> skrivs på skärmen innan inmatningen tas emot. Frågetecken skrivs inte ut om det inte ingår i <sträng>. Hela inmatningen inklusive <ENTER> lägga in i <strängvariabel>. Om radmatning/vagnretur tas emot (endast i denna ordning) visas bägge tecknen på skärmen men vagnreturen ignoreras, radmatningstecknet läggs in i <strängvariabeln> och inmatningen kan fortsätta.

En LINE INPUT kan förbigås genom att skriva <CTRL>-<STOP>. BASIC återgå då till kommandonivå och Ok skrivs ut. Med CONT kan programmet fortsättas vid LINE INPUT.

Exempel:
```
10 LINE INPUT "Inmatning av sträng:";A$
20 PRINT LEFT$(A$,5)
30 PRINT RIGHT$(A$,5)
RUN
Inmatning av sträng: 123,456,789,0ABCDEF
123,4
BCDEF
Ok
```

2.3.22 LINE INPUT#

Format: LINE INPUT# <filnummer> , <strängvariabel>

Funktion: Läser en hel rad tecken (högst 254) från en sekventiell fil till en strängvariabel.

Användning: <filnummer> är det nummer som filen öppnades med. <strängvariabel> är det variabelnamn som strängen får. LINE INPUT# läser alla tecken i filen fram till en vagnretur. Därefter hoppas vagnretur/radmatning över och nästa LINE INPUT# läser alla tecken fram till nästa vagnretur (om teckenföljden radmatning/vagnretur påträffas behålls den i strängen).

LINE INPUT# är särskilt användbar om varje rad i datafilen är uppdelad i fält eller om ett BASIC-program som sparats i ASCII-kod skall läsas av ett annat program.

Exempel: Se bilaga.

2.3.23 LOAD

Format: LOAD <filnamn> [,R]

Funktion: Laddar in en programfil från kassettband eller flexskivenhet i arbetsminnet.

Användning: <filnamn> är det namn som användes när filen sparades (med SAVE).Filen kan innehålla ett program i ASCII-format eller internkodsformat. LOAD stänger alla öppna filer och raderar alla variabler och programrader som finns i minnet innan den laddar angivet program.
Om tillägget R anges, startas programmet direkt efter inladdningen utan att RUN behöver anges. Används tillägget vid en bildfil, fås felmeddelandet "Illegal function call".

Exempel:

LOAD "prog"	laddar in innehållet i filen "prog" i arbetsminnet
LOAD "prog",R	startar programmet efter laddningen.

2.3.24 LPRINT och LPRINT USING

Format: LPRINT [<lista med uttryck>]
LPRINT USING <stränguttryck> ;[<lista med uttryck>]

Funktion: Ger utskrift på skrivare

Användning: Samma som PRINT och PRINT USING men utskriften styrs i stället till skrivare. Se dessa instruktioner.

LPRINT förutsätter en skrivare med teckenbredden 132.

2.3.25 LSET och RSET

Format: LSET <strängvariabel> = <stränguttryck>
RSET <strängvariabel> = <stränguttryck>

Version: Endast vid flexskivfiler.

Funktion: Flyttar data från arbetsminnet till direktfilsbuffert (förberedelse för en PUT-sats)

Användning: Om <stränguttryck> erfordrar färre bytes än som tilldelades <strängvariabel> med en FIELD-sats, vänsterjusterar LSET teckensträngen i fältet och RSET högerjuaterar (mellanslag läggs in för att fylla fältet). Om strängen är för lång skärs tecken av från höger vid LSET och från vänster vid RSET. Numeriska värden måste konverteras till strängar innan LSET och RSET används. Detta görs med funktionerna MKI$, MKS$, MKD$, se dessa.

Anmärkning: LSET och RSET kan också användas med en strängvariabel som inte tilldelats fältutrymme för att höger eller vänsterjustera en teckensträng i ett visat fält.

Exempel: Användning av RSET och LSET vid filer, se bilaga.

```
10 A$ = SPACE$(20)
20 RSET A$ = "TEST"
30 PRINT A$
RUN
                TEST
Ok
```

2.3.26 LOCATE

Format: LOCATE [<position>][,[<radnummer>][,[
<markör>]]]

Funktion: Placering av markören på bildskärmen

Användning: <radnummer> är ett numeriskt uttryck, vars värde skall ligga mellan 1 och 23 om funktionstangentraden är borta, annars mellan 1 och 22. 1 anger den översta skärmraden. (Detta är betydelsen vid SCREEN 0, gäller ej vid SCREEN 1 och 2.)

<position> är ett numeriskt uttryck som skall ligga mellan 1 (längst till vänster) och bildskärmens bredd (39,40 eller 80).
<markör> anger om markören skall visas (värdet 1) eller inte visas (värdet 0).

Exempel: LOCATE 2,25 placerar markören på rad 2 i position 25 och förändrar inte markörvisningen.

2.3.27 MAXFILES

Format: MAXFILES <numeriskt uttryck>

Funktion: Anger maximalt antal filer

Användning: <numeriskt uttryck> skall vara ett tal mellan 1 och 15. Detta värde ger högsta antalet filer som kan vara öppna samtidigt. Standardvärdet är 1. Fler OPEN-instruktioner med högre värde än <numeriskt uttryck> ger ett felmeddelande. Denna instruktion skall, när den används, stå först i programmet.

Exempel: 1 MAXFILES=4 efter denna sats kan högst 4 filer vara öppna samtidigt.

2.3.28 MOTOR ON / OFF

Format: MOTOR ON / OFF

Funktion: In- och urkoppling av kassettbandspelarmotorn

Avändning: Med denna instruktion kan avspelning av data eller ljud från kassett startas resp stoppas.

2.3.29 NAME

Format: NAME <gammalt filnamn> AS <nytt filnamn>

Version: Endast för flexskivor

Användning: <gammalt filnamn> måste finnas på skivan men <nytt filnamn> får inte finnas på skivan.

Exempel: NAME "2:LISTA" AS "LISTA.NY" omdöpning av "LISTA" till "LISTA.NY"

2.3.30 OPEN

Format: OPEN <filnamn> [FOR <form>] AS [#] <filnummer>

Funktion: Öppning av fil för bearbetning (läsning/skrivning).

Användning: Innan data kan läsas eller skrivas från/till en fil, måste den öppnas. Instruktionen tilldelar <filnam> ett <filnummer> och bestämmer formen av bearbetning. Som <form> kan följande beteckningar användas:

INPUT	för läsning från filen
OUTPUT	för skrivning på filen
APPEND	för komplettering (utökning) av filen (endast vid flexskiva)

Vid INPUT I denna form sätts datapekaren på det första tecknet i filen. Om filen inte existerar fås ett felmeddelande

Vid OUTPUT I denna form definieras filen om den inte finns. Om filen redan finns raderas den. Därefter sätts datapekaren till början av filen.

Vid APPEND Denna form kan bara användas vid direktfiler. Om filen inte finns fås ett felmeddelande. Datapekaren står vid början av sista satsen i filen. När sista satsen lästs (även om andra satser läses dessemellan) kopplas om till OUTPUT-form och data kan sedan bara skrivas på filen.

Om FOR <form> inte anges (endast vid direktfiler) så definieras filen om den inte finns. Datapekaren placeras före den första satsen och den första läsningen/skrivningen bestämmer bearbetningsformen. För ett och samma <filnummer> kan antingen bara läsas eller skrivas (med undantag för APPEND). Det är dock möjligt att tilldela filen flera filnummer och på så sätt både skriva och läsas på filen.

Om heltalsuttrycket <filnummer> är större än ett tidigare angivet MAXFILES fås felmeddelandet "Bad file number" (otillåtet filnummer).

Exempel: Se bilaga B sidan 94.

2.3.31 PLAY

Format: PLAY [<sträng>][,[<sträng2>][,
<sträng3>]]

Funktion: För framställning och spelning av melodier

Användning: Tre stränguttryck kan anges, ett för varje ljudkanal. Strängarna (1,2,3) sätts samman av följande delinstruktioner:

A..G	spelar tonen (B motsvarar tonen H).
# +	ett av dessa tecken efter en ton medför att tonen höjs en halvton.
-	efter en ton ger en sänkning med en halvton.
.	en punkt efter tonen medför att tonen spelas med 1.5 gånger perioden som angivits med L<n> och T.
L<n>	bestämmer längden på den följande tonen. L1 är en helton, L4 en fjärdedel osv. <n> måste ligga inom intervallet 1 till 64. Det går också att ange längden <n> direkt efter tonen när endast denna ton skall spelas med denna längd, t.ex C16.
M<n>	period hos enveloppkurvan (total amplitud hos tonen). <n> kan beräknas som 1789772.5 * frekvensen/1536 och skall ligga inom intervallet 1 till 65536.
N<n>	spelar tonen <n>. <n> kan vara mellan 0 och 84 (i 7 oktaver ligger 84 toner) där 0 betyder paus.
O<n>	bestämmer oktaven för efterföljande toner. Värdeområde för <n> är 0 till 6. I varje oktav finns tonerna: C D E F G A B
R<n>	ger en paus av längden <n> (motsvarar <n> i L).

S<n> anger tonformen. <n> kan anta värden mellan 0 och 15 som definieras:

0-3,9	
4-7,15	
8	
10	
11	
12	
13	
14	

T<n> anger antal fjärdedelstoner per sekund. Värdeområde 32 - 255. Standardvärde för T är 120.

V<n> anger ljudstyrkan (volymen). Värdeområde 0 - 15. Standardvärde är 8. Om V<n> anges i strängen kopplas enveloppgeneratorn ur. S<n> och V<n> får alltså inte användas samtidigt.

X<strängvariabel>;
användер en strängvariabel som definition för definiering av tonerna (se grafikinstruktionen DRAW).

Förutom konstanter kan för varje <n> ovan även variabler användas, som t.ex:
I stället för PLAY "O3C" kan man skiva PLAY "O=OK;C"
där variabeln OK innehåller värdet 3.
Observera att alla variabler som används måste föregås av ett likhetstecken och avslutas med ett semikolon.

Exempel:

```
PLAY "O4CEC"

A$="L1"
PLAY "O3CXA$;C"
```

2.3.32 PRINT

Format: PRINT [<lista med uttryck>]

Funktion: Utskrift av data på bildskärmen eller skrivare.

Användning: Om <lista med uttryck> utelämnas, skrivs en tom rad, annars beräknas och skrivs värde(na) på uttrycken ut. Uttrycken får vara numeriska och/eller stränguttryck (strängar måste inneslutas av citationstecken) och skall skiljas åt med kommatecken eller semikolon.

Utskriftspositioner:
Utskriftspositionen för varje uttryck som skrivs ut bestäms av de tecken som används för att skilja de olika uttrycken åt. BASIC delar in raden i utskriftszoner på vardera 14 positioner. I listan med uttryck medför ett kommatecken att nästa värde skrivs i första position i nästa utskriftszon. Ett semikolon medför att uttrycket skrivs omedelbart efter det föregående. Om ett eller flera mellanslag skrivs mellan stränguttrycken har det samma effekt som ett semikolon.

Om ett kommatecken eller ett semikolon avslutar listan med uttryck börjar nästa PRINT-sats utskrift på samma rad, med motsvarande mellanrum. Om listan med uttryck slutar utan kommatecken eller semikolon skrivs en vagnretur automatiskt vid radslut. Om den utskrivna raden har fler tecken än antalet positioner i raden, fortsätter utskriften på nästa rad.

Efter siffror skrivs alltid mellanslag ut. Positiva tal föregås av ett mellanslag. Negativa tal föregås av ett minustecken. Tal med enkel (dubbel) precision som kan skrivas ut med 6(14) eller färre siffror i decimalt format med samma noggranhet som i exponentiellt format, skrivs ut decimalt, annars i expontialformat.

Ett frågetecken kan användas i stället för ordet PRINT i en PRINT-sats.

Exempel:
```
? 1E-2;1E-16          det andra uttrycket behöver
  .01 1E-16           17 positioner

 10 X=5 : A$="####"
 20 PRINT X+5,X-5,X*(-5);XÜ5;A$+"$$"
 RUN
  10              0
 -25  3125   ####$$
 Ok
```

Värdena av de första två uttrycken skiva i olika utskriftszoner, övriga direkt efter varandra.

Exempel:
```
10 INPUT X
20 PRINT X "I KVADRAT ÄR" XÜ2 "OCH";
30 PRINT X "I KUBIK ÄR" XÜ3
40 PRINT
50 GOTO 10
Ok
RUN
? 9                  anges av användaren
  9 I KVADRAT ÄR 81 OCH 9 I KUBIK ÄR 729

? 4
  4 I KVADRAT ÄR 16 OCH 4 I KUBIK ÄR 64

?
```

I detta exempel medför semikolonet i slutet på rad 20 att bägge PRINT-satserna skrivs på samma rad och rad 40 medför att en blank rad skrivs före nästa frågetecken.

2.3.33 PRINT USING

Format: PRINT USING <stränguttr> ; <lista med uttryck>

Funktion: Utskrift av strängar eller tal med användning av speciellt format.

Användning <lista med uttryck> består av de stränguttryck eller numeriska uttryck som skall skrivas ut, åtskilda av semikolon. <stänguttr> är en sträng som består av särskilda formateringstecken. Dessa formateringstecken (se nedan) avgör hur utskriftsfälten och formatet hos de utskrivna strängarna eller talen ser ut. Om det finns fler tecken i listan än vad som anges i <stränguttr> påbörjas utskriften igen enligt formateringsangivelserna. Formateringstecknen för strängar får inte användas för numeriska uttryck eftersom då felet "Type mismatch" erhålls. Mellan de egentliga formateringsangivelserna kan valfria tecken stå som inte är formateringstecken. Dessa tecken skrivs ut exakt som de är angivna.

Formateringstecken för strängutskrifter:

! Anger att endast det första tecknet i angiven sträng skall skrivas ut.

Ö<k >Ö Antalet mellanslag + 2 anger hur många tecken från strängen som skrivs ut. Om strängen är längre än angivet fält kommer de extra tecknen att ignoreras. Om fältet är längre än strängen kommer strängen att vänsterställas i fältet och mellanslag fyller det kvarvarande utrymmet.

```
Exempel:
10  AS = "SE" : B$ = "UPP"
30  PRINT USING "!";A$;B$
40  PRINT USING "Ö   Ö";A$;B$
50  PRINT USING "Ö   Ö";A$;B$;"!!"
RUN
SU
SE   UPP
SE   UPP  !!
```

Formateringstecken för numeriska fält:

Ett nummertecken ("brädgård") används för att ange varje sifferposition. Positionerna fylls alltid ut. Om talet som skall skrivas ut har färre siffror än antalet angivna positioner kommer talet att högerställas (föregås av mellanslag) i fältet.

. En decimalpunkt kan placeras in på valfri position i fältet. Om formatsträngen anger att en siffra skall komma före decimalpunkten, skrivs siffran alltid ut (en nolla om så behövs). Tal avrundas vid behov.

+ Ett plustecken i början eller slutet av formatsträngen ger utskrift av talets tecken (plus eller minus).

- Minustecken i slutet av formatsträngen medför att negativa tal skrivs ut med efterföljande minustecken.

** Två asterisker i början av formatsträngen medför att tomma positioner framför talet fylls ut med asterisker. ** ger också utrymme för ytterligare två siffror.

$$ Två dollartecken i början av formatsträngen medför att ett dollartecken skrivs ut omedelbart till vänster om det utskrivna talet. $$ anger två extra positioner för tecken, varav det ena är ett dollartecken. Exponentialformat kan inte användas tillsammans

med $$. Negativa tal kan inte användas om inte minustecknet skrivs ut till höger.

$ Tecknen "$" i början av formatsträngen kombinerar funktionen av de två ovan angivna symbolerna. Mellanslag i början fylls med asterisker och ett dollartecken skrivs ut före talet.

, Om ett kommatecken placeras till vänster om decimalpunkten i en formatsträng så skrivs ett kommatecken ut till vänster om var tredje siffra som står till vänster om decimalpunkten. Ett kommatecken sist i strängen skrivs ut som en del i strängen. Kommatecknet anger även en extra sifferposition. Kommatecknet har ingen funktion vid exponentialformat (ÜÜÜÜ).

ÜÜÜÜ Fyra versala Ü kan placeras i slutet av formatsträngen för att ange exponentialformat. De fyra tecknen reserverar utrymme för E+xx. Valfri position för decimalpunkt kan anges. Exponenten anpassas till valt format. Om plats för + eller - inte anges, används en sifferposition till vänster om decimalpunkten. Här skrivs ett mellanslag eller ett minustecken.

- Ett understrykningstecken i formatsträngen medför att nästa tecken skrivs ut som det står i strängen.

Om talet, som skall skrivas, är större än det format som reserverats, skrivs ett procenttecken ut före talet. Procenttecken skrivs också, om en avrundning får talet att bli större än det reserverade fältet.

Exempel:
```
PRINT USING "##.## ";.789,55,-55,555,1.237,-5
 0.79 55.00 %-55.00 %555.00 1.24 -5.00
Ok
```

Genom mellanslaget efter nummertecknen skrivs alltid ett mellanslag ut mellan siffrorna.

```
PRINT USING "+#.##  #.##-  #.##+  ";
      -5,-5,-5,5,5,5
 -5.00  5.00-  5.00-  +5.00  5.00  5.00+
Ok
```

```
PRINT USING "**.# **##.# $$.# **$#.#",
             2.3, 2.3, 2.3, 2.3
  *2.3  ***2.3  $2.3  **$2.3
Ok

PRINT USING "#########,.####"; 123456789012
12,345,678.9012
Ok

PRINT USING "##.##ÜÜÜÜ  ######.###ÜÜÜÜ";
      2345.6, 123.456789
 2.35E+03  12345.679D-02
Ok
```

2.3.34 PRINT# och PRINT# USING

Format: PRINT# <filnummer>,[USING <stränguttr>;]<lista med uttryck>

Funktion: Skrivning av data i en sekventiell fil

Användning: <filnummer> är samma siffra som användes när filen öppnades (med OPEN). <stränguttr> består av formateringstecken som beskrevs i avsnittet om PRINT USING. Uttrycken i <lista med uttryck> är numeriska och/eller stränguttryck som skall skrivas in i filen. Uttrycken skivs in på filen på samma sätt som om de skrevs på skärmen med PRINT och PRINT USING. I listan med uttryck måste numeriska uttryck skiljas åt med semikolon. Samma skiljetecken bör användas som beskrivits vid instruktionen INPUT.

2.3.35 SAVE

Format: SAVE <filnamn>[,A / S]

Funktion: Lagrar ett program eller en skärmbild på kassett eller skiva.

Användning: <filnamn> är en sträng omgiven av citationstecken som är ett tillåtet filnamn.

Programmet eller bilden (för bild med tillägget ,S) sparas under angivet namn. Om tillägget ,A skrivs, lagras filen i ASCII-form (läsbar med INPUT#), annars lagras den i maskinkod. För att kunna använda kommandot "MERGE" krävs att filen är lagrad i ASCII-kod. Om filnamnet är kortare än ett tecken eller längre än 8 tecken fås felmeddelandet "Bad file Name" och lagringen avbryts.

Exempel:
```
SAVE "PROG"          lagrar i BASIC-kod
SAVE "PROG",A        lagrar i ASCII-kod
SAVE "BILD",S        lagrar bilden på bildskärmen
SAVE "1:PROG"        lagrar i BASIC-kod på skiva
```

2.3.36 SET

Format: SET <skivenhetsnummer> / #<filnummer> / <filnamn> , <attribut>

Version: Endast för flexskivor

Funktion: Ändring av attribut hos en flexskivenhet eller en fil

Användning: <attribut> är ett stränguttryck som kan ha följande värden:

R	kontrolläsning (verifiering)
P	skrivskyddad
E	konvertering till EBCDIC-kod

För en skivenhet så skrivs detta attribut endast i minnet på räknarenheten. När en fil framställs får filen samma attribut som skivenheten i övrigt. Anges <filnamn> ändras attributen på skivan endast för denna fil. Anges #<filnummer> ändras attributen endast i minnet och inte på skivan. Anges en tom sträng raderas alla attribut för fil eller skiva.

Exempel:

SET 1,"R"	efter varje skrivning på skivenhet 1 görs kontroll av data
SET #1,"P"	ger felmeddelandet "Filen skrivskyddad" om försök görs att före CLOSE#i skriva data på filen
SET 2,""	raderar alla attribut på skivenhet 2

2.3.37 SOUND

Format: SOUND <PSG-register> , <databyte>

Funktion: Skrivning av data direkt till tongeneratormodulen (ljudprocessorn)

Användning: <PSG-registeradress> skall ligga mellan 0 och 13 och anger vilket ljudregister som adresseras. <databyte> anger vilket värde som skall placeras i angivet register.
De 13 registrens funktioner är följande:

Register	Funktion	Tillåtna värden
0	Finjustering frekvens kanal A	0 - 255
1	Grovinställning frekvens kanal A	0 - 15
2	Finjustering frekvens kanal B	0 - 255
3	Grovinställning frekvens kanal B	0 - 15
4	Finjustering frekvens kanal C	0 - 255
5	Grovinställning frekvens kanal C	0 - 15
6	Brusfrekvens (vitt-"skärt" brus)	0 - 15
7	Mixer. Avgör om ljudkanalerna skall ge brus eller ton. Resultatet från mixern avgörs av hur bitarna placeras i registret. Ljudnivån påverkas ej. Bitvikter:	255 - 192

B7	B6	B5	B4	B3	B2	B1	B0
-Ingen--		----Brus----			-----Ton----		
funktion		C	B	A	C	B	A

Funktionerna är inverterade. 1 = från, 0 = till

8	Bit 0-3 : Volym kanal A Bit 4-7 : Enveloppstyrning kanal A	0 - 16
9	Bit 0-3 : Volym kanal B Bit 4-7 : Enveloppstyrning kanal B	0 - 16
10	Bit 0-3 : Volym kanal C Bit 4-7 : Enveloppstyrning kanal C	0 - 16
11	Fininställning enveloppfrekvens	0 - 255
12	Grovinställning enveloppfrekvens	0 - 255
13	Vågform envelopp: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	

Observera: För att enveloppgeneratorn skall vara inkopplad måste:
Register 7 innehålla ett tal mindre än 255
Register 8, 9 eller 10 innehålla 16
Register 13 innehålla ett tal som ger vald vågform.
Hastigheten (frekvensen) finjusteras därefter med register 11 och 12.

2.3.38 SOUND

Format: SOUND ON / OFF

Funktion: Omkoppling av anslutningen mellan kassettbandspelaren och förstärkaren i datorn

Användning: Denna instruktion används för att koppla in den andra kanalen på bandspelaren, på vilken inspelning med mikrofonen kan göras, till förstärkaren (ON) eller för att bryta inkopplingen (OFF). Se även instruktionen MOTOR.

2.3.39 SWITCH

Format: SWITCH [STOP]

Funktion: Inkoppling av extra minnesblock

Användning: Om STOP anges, utförs efter inkopplingen ett <CTRL-STOP>. SWITCH får bara användas när extra minneskort är anslutet.

2.3.40 WIDTH

Format: WIDTH <storlek>

Funktion: Med WIDTH anges antalet teckenpositioner på en bildskärmsrad.

Användning: <storlek> måste vara antingen 39 eller 40. Standardvärdet är 39. Om 80-teckenskort finns installerat kan även siffran 80 användas för att få 80 positioner på raden. Vid användning av WIDTH blankas skärmen.

2.3.41 WRITE#

Format: WRITE# <filnummer>, <uttr>[,<uttr>,...,<uttr>]

Version: Endast för flexskivor

Funktion: Skrivning av data på en fil på skiva.

Användning: <filnummer> är det nummer som angavs när filen öppnades med OPEN. Uttrycken kan vara stränguttryck eller numeriska uttryck och måste skiljas åt med kommatecken.

Skillnaden mellan WRITE# och PRINT# är att WRITE# skriver in kommatecken mellan uttrycken och skiljer strängar åt med citationstecken. Det är alltså då inte nödvändigt att lägga in skiljetecken i listan. Tecknen för vagnretur/radmatning skrivs automatiskt in efter det att det sista uttrycket skrivits in.

Om WRITE# används med mellanlagring på en direkt fil måste ett PUT följa efter varje WRITE# eftersom annars ett felmeddelande ("File overflow") fås efter nästa WRlTE#.

2.4 INSTRUKTIONER FÖR GRAFIK

Vid vissa av grafikinstruktionerna kan det inträffa att fel uppstår när de anropas med skärmen i textläge. Innan grafik framställs måste därför SEREEN med <läge> större än noll anges. Origo (x=0, y=0) ligger i övre vänstra hörnet av bildskärmen (hemmaläget). Den positiva X-axeln går mot höger, den positiva Y-axeln går neråt.

2.4.1 CIRCLE

Format: CIRCLE [STEP] (<X>,<Y>), <radie> [, [<färg>] / [, [<start>] /],] <slut>] / [, <förhållande>]]]]

Funktion: Ritar en cirkel eller en ellips

Användning: <X> och <Y> är koordinater för medelpunkten hos figuren. De kan anges antingen absolut eller relativt (STEP). <färg> bestämmer färgen för ramlinjen hos figuren. Standardvärdet för färgen är förgrundsfärgen (se COLOR). Med <start> och <slut> kan cirkelbågar ritas genom att start och slutvinkel anges i radianer. (-2 pi till +2 pi>. Om antingen <start> eller <slut> är negativa ersätts de med noll och en linje dras från slutpunkten till figurens medelpunkt.

<förhållande> bestämmer förhållandet mellan radien a och radien b i en ellips. Standardvärdet är 1. Därigenom skall teoretiskt en fullständig cirkel ritas. Monitorns (TV-apparatens) geometri påverkar dock detta varför en justering normalt behöver göras.

Exempel:
```
CIRCLE (128,96),90,,0,3.14*1.5

CIRCLE (128,96),100,,0,3.14,0.5
```

2.4.2 COLOR

Format COLOR [<förgrund>] / [, [<bakgrund>] / [, <ram>]]

Funktion: Förändring av färgerna på skärmen

Användning: <föryrund> anger färgen på figurer på skärmen (i textläge på tecknen). <bakgrund> anger färgen på bakgrunden och <ram> färgen på ramen. Om inga värden anges, antas föregående värden. Färgnumren anges i följande tabell. För inte angivna värden fås felmeddelandet "Illegal funktion call".

Värde	Färg	Värde	Färg
0	genomskinlig	8	mellanröd
1	svart	9	ljusröd
2	mellangrön	10	mörkgul
3	ljusgrön	11	ljusgul
4	mörkblå	12	mörkgrön
5	ljusblå	13	violett
6	mörkröd	14	grå
7	blå (cyan)	15	vit

Färgerna kan variera i styrka och färgton beroende på monitorns (TV-apparatens) egenskaper.

Exempel:

```
        .
100 COLOR 8,3
        .
        .
200 COLOR 9,,5
```

2.4.3 DRAW

Format: DRAW <kommandosträng>

Funktion: Ritar figurer på bildskärmen

Användning: <kommandosträng> består av kommandon ur det grafiska makrospråket GML (Grafic Macro Language). Makrospråk innebär att korta kommandon ger stora förändringar. Den form som definieras i strängen ritas med instruktionen DRAW. Följande GML-kommandon kan användas:

U<n>	flyttar markören uppåt
D<n>	flyttar markören neråt
L<n>	flyttar markören åt vänster
R<n>	flyttar markören åt höger
E<n>	flyttar markören diagonalt uppåt höger
F<n>	flyttar markören diagonalt neråt höger
G<n>	flyttar markören diagonalt neråt vänster
H<n>	flyttar markören diagonalt uppåt vänster

<n> anger i ovanstående kommandon förflyttningslängden i varje riktning med utgångspunkt från aktuell markörposition. Den verkliga förflyttningssträckan är <n> gånger skalfaktorn från GML-kommandot S<>.

M<X> , <Y>	Absolut eller relativ rörelse med angiven storlek (beror på skalfaktorn) som ritar en linje till ny ändpunkt. Om ett tecken (+ eller -) skrivs före <x> så uppfattas det som relativ angivelse, dvs de läggs till aktuell markörposition. Annars antas punkten <X>,<Y> som ny slutpunkt.

Avståndet mellan punkterna dit markören flyttas är beroende av sidoförhållandet hos bildskärmen (skärmbredd/skärmhöjd).

Följande två GML-kommandon kan placeras framför vart och ett av ovanstående förflyttningskommandon:

B	flyttar markören utan att någonting ritas
N	markören återförs efter förflyttningen tillbaka till utgångspunkten för samma rörelse

Övriga CML-kommandon

A<n> sätter vinkeln för förflyttningen beroende på <n>. Tillåtna värden på <n> är:
0 motsvarar 0 grader
1 " 90 "
2 " 180 "
3 " 270 "
Figurer som ritas i vinklarna 90 eller 270 grader ges en sådan skala att de vid ett standardbildskärmsförhållande på 4:3 får korrekta längder.

C<n> ger färgen <n> (färgnummer se COLOR)

S<n> sätter skalfaktorn <n>/4. <n> får ha värden mellan 1 och 255. Som skalfaktor tas 1/4 av det angivna värdet. Skalfaktorn gånger storleken <n> hos CML-kommandona U, D, L, R, E, F, G, H och M ger storleken på förflyttningen.

Man kan även låta DRAW styras med variabler i BASIC.

X<strängvariabelnamn>;
detta CML-kommando utför de i strängen <strängvariabelnamn> lagrade CML-kommandona. Observera semikolonet efter variabelnamnet.

<n> i ovan angivna kommandon kan <n> ersättas med =<variabelnamn>;. Detta måste då vara en numerisk variabel och ligga inom de tillåtna gränserna för varje <n>. Förutom vid den första parametern hos M-kommandot, måste ett semikolon skrivas efter varje <variabelnamn>

Exempel:

```
10 SCREEN 1
20 K$="L200"
30 C0=10
40 DRAW "BM220,192"
50 DRAW "U8C=C0;XK$;U100C15R50"
60 GOTO 60
```

Rad 10: skärmen i högupplösningsgrafikläge
Rad 20: 200 enheter till vänster
Rad 40: medelpunkten flyttas neråt
Rad 50: rita 8 enheter uppåt med senast angiven färg C0 dvs 10, rita strängen K$ dvs 200 enheter åt vänster, rita 100 enheter uppåt, byt till färg 15 och rita därefter 50 enheter åt höger.
Rad 60: Oändlig slinga, programmet avbryts med <CTRL>-<STOP>

2.4.4 GET

Format: GET (<x1>,<y1>) - (<x2>,<y2>) , <fältnamn>

Funktion: Läser bildpunkter (pixels) från bildskärmen

Användning: <x1>, <y2>, <x2>, <y2> är koordinaterna för två hörnpunkter i en rektangulär yta på bildskärmen. Färgerna på alla bildpunkter inom rektangeln läses in i fältet. <fältnamn> måste vara namnet på ett numeriskt fält.

Exempel:

```
.
.
100 GET (10,10)-(40,40),C
.
.
```

2.4.5 LINE

Format: LINE [STEP] [(<x1>,<y1>)] - [STEP]
(<x2>,<y2>) [, [<färg >] / [, B [F]]]

Funktion: Ritar linjer och fyrkanter på bildskärmen.

Användning: - (<x2>,<y2>) ritar en linje från aktuell markörposition till punkten <x2>,<y2>. Om STEP anges, antas <x2>,<y2> vara relativa koordinater (relativa i förhållande till aktuell position). Om <x1>,<y1> anges används denna punkt som utgångspunkt för linjen i stället för aktuell markörposition. Färgen på linjen kan anges med <färg>. Om tillägget B (box) används, ritas en fyrkant med ett hörn i utgångspunkten och ett diagonalt motsatt hörn i punkten [STEP] (<x2>,<y2>). Genom tillägget F fylls fyrkanten med angiven färg (antingen uttryckt i instruktionen eller i SCREEN).

Exempel:

```
.
.
100 LINE - STEP (100,80),,B
```

ritar en fyrkant med ett hörn i aktuell position och diagonalt motsatt hörn i (X+100, Y+80)

2.4.6 PAINT

Format: PAINT (<x> , <y>) [, <färg>]

Funktion: Fyller en figur på bildskärmen med färg

Användning: Koordinaterna <x> och <y> måste ange en punkt (valfri) inom figuren och <färg> måste överensstämma med färgen på figurens ramlinje.

Om figuren har en mycket oregelbunden ramlinje kan det inträffa att felmeddelandet "Out of memory" fås. I detta fall kan instruktionen CLEAR användas, se 2.2.1

Exempel:

```
          .
          .
 100 PAINT (128,3*32),12
```

färglägger en figur inom vilken punkten (128,96) finns med färgen 12 (mörkgrön).

2.4.7 PSET, PRESET

Format: PSET [STEP] (<x> , <y>) [, <färg>]
PRESET [STEP] (<x> , <y>) [, <färg>]

Funktion: Tänder eller släcker en punkt på bildskärmen

Användning: Den med <x>,<y> angivna bildpunkten tänds med angiven eller förinställd <färg>. Om STEP skrivs blir koordinaterna relativa. Med PSET är förinställd färg lika med förgrundens färg, med PRESET är den lika med bakgrundens färg. Om <färg> anges blir resultatet av instruktionerna likadant.

Exempel:

```
          .
          .
 50 PSET (128,90),12
 70 PSET STEP (10,10),6
          .
          .
```

2.4.8 PUT

Format: PUT (<x> , <y>) , <fältnamn> [, <operation>]

Funktion: Utskrift av en lagrad bild på bildskärmen

Användning: <x>,<y> är koordinaterna för det övre vänstra hörnet av rektangeln som ritas ut på bildskärmen från det numeriska fältet. Genom att lägga till <operation> kan innehållet i fältet kopplas samman med befintligt innehåll på bildskärmen. Tillåtna namn på <operation> är:

PSET	påverkar ritandet av den ursprungliga bilden (motsatsen till GET).
PRESET	reverserar innehållet i fältet och ritar ut den nya bilden på bildskärmen (fotografiskt negativ).
AND	den i fältet lagrade bilden visas på bildskärmen bara där ett tecken redan finns.
OR	bilden i fältet överlagras på befintlig bild.
XOR	den i fältet lagrade bilden visas på bildskärmen endast där inget annat tecken visas.

Exempel:

```
.
.
200 PUT (100,100),A,OR
.
.
```

2.4.9 SCREEN

Format: SCREEN <läge> [, <tillägg>]

Funktion: Omställning av bildskärmsvisningen

Användning: <läge> är ett heltal som kan anta följande värden:

0	endast visning av text
1	högupplösningsgrafik 256 x 192 punkter
2	lågupplösningagrafik 64 x 48 punkter

<tillägg> är ett heltal vars funktion beror på angivet <läge>:

<läge>	<tillägg>	
0	0	ej visning av funktionstangenternas definitioner
	1,2,3	visning av funktionstangenternas definitioner
1, 2	0	spritestorlek 8 x 8
	1	" 8 x 8 förstorade
	2	" 16 x 16
	3	" 16 x 16 förstorade

Förstoringsfaktorn vid <tillägg> 1 och 3 i grafikläge är 2. Om <tillägg> inte anges bibehålls den senast angivna.
När denna instruktion utförs raderas bildskärmen.
Överlagring av text och grafik är möjlig då <läge> är 1 eller 2. I det senare fallet blir texten 8 gånger förstorad.
Om programmet påträffar en STOP- eller END-instruktion eller om <CTRL-STOP> tryckts ner, utförs genom BASIC-tolken ett SCREEN 0.

Exempel:

```
SCREEN 1

SCREEN 1,3
```

2.4.10 PUT SPRITE

Format: PUT SPRITE <yta> [, [[STEP] (<x>,<y>)] / [[<färg>] / [, <spritenummer>]]]

Funktion: Ritar en sprite på bildskärmen

Användning: <yta> är ett heltalsuttryck från 0 till 31 som anger den yta i bildskärmen där spriten skall ritas. I varje <yta> kan endast en sprite finnas. Om två spritar ligger på samma ställe syns endast den sprite som ligger på <yta> med lägsta värde (närmast betraktaren).
<x>,<y> är koordinaterna för det ställe på bildskärmen där spriten ritas (med STEP ritas den relativt aktuell markörposition). Utgångsvärde för <x>,<y> är aktuell markörposition.
<färg> anger färgen på spriten (om <färg> inte anges gäller förgrundsfärgen).
<spritenummer> är ett heltalsuttryck som anger vilken sprite som skall ritas på skärmen. En sprite med detta ordningsnummer måste dessförinnan ha definierats med SPRITE$()=... Om <spritenummer> inte anges sätts värdet på <yta> in.
Värdet på <x> måste ligga från -32 till 255, värdet på <y> från -32 till 191.
Med värdena <y>=209 och <x>=208 kan alla spritar på skärmen raderas.

Exempel:

```
          .
100 PUT SPRITE 3
          .
400 PUT SPRITE 2,(X+8,Y-10),7,13
           ritar en sprite som definierats med SPRI-
           TE$(13) på yta 2 i angiven koordinatpunkt
```

2.4.11 SPRITE$

Format: SPRITE$(<spritenummer>) = <stränguttryck>

Funktion: Definiering av spritar

Användning: Vid SCREEN-<tillägg> 0 eller 1 måste <stränguttryck> innehålla 8 och vid <tillägg> 2 eller 3 32 tecken för att definiera hela spriten. Det bitvisa ritandet av det första tecknet sker vågrätt från vänster. Övriga tecken ritas därunder. Är spritestorleken 16x16 ritas det 17:e tecknet till höger om det första tecknet. En angiven bit i tecknet ritas vid PUT SPRITE med angiven (förgrunds)färg, en icke angiven bit med bakgrundsfärg.
Tillåtet värde på <spritenummer> är 0 till 31.

Exempel:

```
SPRITE$(K)="0876gdjk"
          .
 100 SPRITE$(I)=A$+CHR$(17)
```

Utförligare exempel finns i bilaga

2.5 INSTRUKTIONER FOR AVBROTTSHANTERING

Avbrott (Interrupts) är händelser som när de inträffar medför att en subrutin eller delprogram anropas. Avbrotten kan åstadkommas genom tryckning på en funktionstangent, tryckning på <CTRL-STOP>-tangenterna, uppträdandet av ett fel, passage av en bestämd tidsperiod, tryck på en styrspakstangent eller då två spritar på skärmen kolliderar med varandra. Avbrottshändelserna lagras tills dess att aktuell programrad genomlöpts. Den händelse som då har den högsta prioriteten (se nedan) verkställs då, dvs motsvarande subrutin utförs. Finns ytterligare avbrottshändelser lagrade, verkställs dessa efter RETURN. Nästa programrad, efter avbrottsrutinerna, utförs först när alla avbrottsrutiner genomlöpts. Om det under behandlingen av avbrottsrutinen uppträder nya avbrottshändelser, lagras dessa och genomförs efter RETURN i respektive prioritetsordning.

Före den första avbrottsinstruktionen med ON...GOSUB (t.ex ON STOP GOSUB) måste motsvarande ...ON ha genomlöpts (i detta fall STOP ON).

Instruktionerna har följande prioritet (utförandeordning):

ERROR, KEY, SPRITE, INTERVAL, STOP

där ERROR har högsta och STOP lägsta prioritet.

2.5.1 INTERVAL

Format: INTERVAL ON / OFF / STOP

Funktion: In- eller urkoppling eller förhindrande av avbrott som genereras av tids funktionen

Användning: Efter det att INTERVAL ON genomlöpts, verkställs avbrott vid ON INTERVAL...

Efter INTERVAL OFF verkställs inga avbrott vid ON INTERVAL...

Efter INTERVAL STOP verkställs inga avbrott vid ON INTERVAL... men resultatet mellanlagras och när instruktionen INTERVAL ON påträffas nästa gång utförs motsvarande subrutin.

Exempel:
```
INTERVAL ON
INTERVAL STOP
```

2.5.2 KEY

Format: KEY [(<tangentnummer>)] ON / OFF / STOP

Funktion: In- eller urkoppling eller förhindrande av avbrott som genereras av tryck på funktionstangent

Användning: <tangentnummer> anger vilken tangent som genererar avbrottet. Tillåtna värden är 1 till 10. Om inget tangentnummer anges, avkännes alla funktionstangenter.

För betydelsen av ON, OFF och STOP se INTERVAL.

Exempel:

KEY ON	alla funktionstangenter ger avbrott
KEY (5) STOP	förhindrar avbrott för funktionstangent 5

2.5.3 STOP

Format: STOP ON / OFF / STOP

Funktion: In- eller urkoppling eller avstängning av avbrott som genereras av tryck på tangenterna <CTRL-STOP>

Användning: När instruktionen STOP ON utförts går det inte längre att stanna programmet med <CTRL-STOP>. För att detta skall kunna ske måste STOP OFF utföras. Om STOP OFF inte utförts kan datorn återställas i kommandoläge endast genom att man stänger av och därefter slår på den igen.
För betydelsen av ON, OFF och STOP, se INTERVAL

Exempel:
```
100 STOP STOP
```

2.5.4 SPRITE

Format: SPRITE ON / OFF / STOP

Funktion: In- eller urkoppling eller förhindrande av avbrott som genereras av att två eller flera spritar kolliderar på bildskärmen.

Användning: När bildskärmen är inställd i grafikläge och två eller flera spritar kolliderar på bildskärmen genereras ett programavbrott.
För betydelsen av ON, OFF och STOP se INTERVAL

Exempel: 100 SPRITE OFF

2.5.5 ON ERROR

Format: ON ERROR GOTO <radnummer>

Funktion: Programhopp vid uppträdande av fel.

Användning: Om ett <radnummer> större än noll anges så kommer ett uppträdande fel (även fel vid direktkommando) att medföra att programmet går till denna rad (eller startas om). Har <radnummer> inte angivits fås felmeddelandet "Undefined line" ("radnummer saknas").
För att avbryta hopp vid fel, ges instruktionen med radnummer noll. Varje fel som uppstår efter ON ERROR GOTO 0 skrivs ut och programmet stoppas.
Om ett fel uppträder i felhanteringsrutinen avbryts programmet.
Se även instruktionerna ERR och ERL.

Exempel: ON ERROR GOTO 200

2.5.6 ON INTERVAL

Format: ON INTERVAL = <uttryck> GOSUB <radnummer>

Funktion: Programhopp vid tidsvillkor

Användning: När den interna klockan gått den tid som anges av <uttryck>, anropas subrutinen som anges med <radnummer> (förutsatt att instruktionen INTERVAL ON givits). <uttryck> anges i 1/50 sekunder. Se även funktionen TIME.

Exempel:

```
20 INTERVAL ON                    startar TIME-avbrott
30 ON INTERVAL=3*20 GOSUB 100     ger vid alla 60/50
                                  sekunder ett GOSUB
                                  100
40 GOTO 40                        ändlös slinga
100 PRINT TIME : RETURN           utskrift av TIME och
                                  återhopp
```

2.5.7 ON KEY

Format: ON KEY GOSUB [<radnummer>] / [,[<radnummer>] / [,[<radnummer>] / .....]]

Funktion: Definierar subrutiner som anropas vid tryck på funktionstangenter

Användning: Före denna instruktion måste en KEY...ON-instruktion ha genomlöpts. Genom ON KEY tilldelas varje funktionstangent ett radnummer som anropas när denna tangent trycks ner. För att koppla ur denna funktion hos en tangent kan <radnummer> 0 anges. Om det för en tangent inte angivits något <radnummer> ger ett tryck på denna tangent inget hopp eller så sker hopp till en rad som tidigare angivits.

Funktionerna INPUT$ och INKEY$ (se dessa) kan inte användas för att få numret på den tangent som medförde hoppet.

Exempel:

```
.
.
100 ON KEY GOSUB 200,300,400,0,500
.
.
```

Utförligt exempel i bilaga.

2.5.8 ON SPRITE

Format: ON SPRITE GOSUB <radnummer>

Funktion: Programhopp vid kollision mellan två eller flera spritar.

Användning: <radnummer> är första raden i en subrutin. Subrutinen anropas när en sprite som är ritad på skärmen berör en annan sprite.

Exempel:

```
.
.
100 ON SPRITE GOSUB 400
.
```

2.5.9 ON STOP

Format: ON STOP GOSUB <radnummer>

Funktion: Definierar en subrutin som anropas när <CTRL>-<STOP> trycka ner

Användning: se STOP ON

Exempel:

```
.
100 ON STOP GOSUB 300
.
```

2.5.10 RESUME

Format: RESUME [<radnummer> / NEXT]

Funktion: Fortsätter programkörningen efter ett fel

Användning: Om inget <radnummer> eller radnummer 0 anges fortsätter programmet med instruktionen som åstadkom felet. Om NEXT anges fortsätter programmet med den instruktion som följer omedelbart efter den som åstadkom felet. Om ett radnummer skilt från noll anges, fortsätter programmet med denna rad. Instruktionen RESUME kan endast användas för felhantering eller som direktkommando om programmet genom ett fel gjort att datorn står i kommandoläge. Se även ON ERROR GOTO, ERROR och funktionerna ERR och ERL.

Exempel:

```
RESUME NEXT

100 RESUME 0

RESUME
```

3. FUNKTIONER

3.1 Funktioner med numeriskt resultat

3.1.1 ABS

Format: ABS (<numeriskt uttryck>)

Resultat: Ger absolutvärdet av <numeriskt uttryck>

Exempel:
```
PRINT ABS (3*(-5)),ABS(3*5)
  15           15
Ok
```

3.1.2 ASC

Format: ASC (<stränguttryck>)

Resultat: Ger ett talvärde som är ASCII-koden för det första tecknet i <stränguttryck> (se bilagan med ASCIIkoder). Om <stränguttryck> är en tom sträng fås felmeddelandet "Illegal function call".

Exempel:
```
 10  B$ = "TEST"
 20  PRINT ASC("A"),ASC("A"+B$),ASC(B$)
 RUN
   65            65            84
 Ok
```

För omvandling i andra riktningen, se funktionen CHR$

3.1.3 ATN

Format: ATN (<numeriskt uttryck>)

Resultat: Ger arcustangens för <numeriskt uttryck> i radianer. Resultaten ligger inom intervallet -PI/2 till PI/2. Uttrycket <numeriskt uttryck> får vara av valfri typ men resultatet anges alltid med enkel precision.

Exempel:
```
PRINT ATN(3)
  1.24905
Ok
```

3.1.4 CDBL

Format: COBL (<numeriskt uttryck>)

Resultat: Omvandlar <numeriskt uttryck> till ett tal med dubbel precision.

3.1.5 CINT

Format: CINT(<numeriskt uttryck>)

Resultat: Omvandlar <numeriskt uttryck> till ett heltal genom avrundning av decimaldelen. Om <numeriskt uttryck> inte ligger inom intervallet -32768 till 32767 fås felmeddelandet "Overflow". Se även funktionerna FIX och INT.

Exempel:
```
PRINT CINT(45.67),CINT(-45.67)
  46          -46
Ok
```

3.1.6 COS

Format: COS(<numeriskt uttryck>)

Resultat: Ger cosinus för <numeriskt uttryck> i radianer. Beräkningen av COS(<numeriskt uttryck>) görs med enkel precision.

Exempel:
```
PRINT 2*COS(.4)
 1.84212
Ok
```

3.1.7 CSNG

Format: CSNG(<numeriskt uttryck>)

Resultat: Omvandlar <numeriskt uttryck> till ett tal med enkel precision.

Exempel:
```
 10 A#=975.34218967
 20  PRINT A#;CSNG(A#)
 RUN
  975.34218967  975.342
 Ok
```

3.1.8 ERL

Format: ERL

Resultat: Ger radnumret för den rad där fel senast uppstod.
Om funktionen står på högra sidan av ett likhetstecken vid en jämförelseoperation så kan ett RENUM-kommando inte ändra det på den vänstra sidan eventuellt stående radnumret.

3.1.9 ERR

Format: ERR

Resultat: Ger det senast påträffade felets nummer

3.1.10 EXP

Format: EXP (<numeriskt uttryck >)

Resultat: Ger e upphöjt till <numeriskt uttryck> som måste vara <= 145.06286085

Exempel:
```
PRINT EXP(4)
 54.5982
Ok
```

3.1.11 FIX

Format: FIX (<numeriskt uttryck>)

Resultat: Ger heltalsdelen av <numeriskt uttryck> (med bortskalade decimaler). Se även CINT, INT

Exempel:
```
PRINT FIX(45.67),FIX(-45.67)
 45           -45
Ok
```

3.1.12 FRE

Format: FRE (<numeriskt uttryck> / <stränguttryck>)

Resultat: Ger antalet lediga bytes i arbetsminnet som inte används av BASIC-tolken. Om ett <stränguttryck> anges i stället för ett <numeriskt uttryck> genomförs en "städning" av minnet innan antalet lediga bytes skrivs ut. En sådan städning kan ta upp till 1.5 minuter. Denna städning av minnet görs automatiskt när allt minnesutrymme har använts.

Exempel:
```
PRINT FRE(0)
 xxxxx
Ok
PRINT FRE ("")
 200
Ok
```

3.1.13 INSTR

Format: INSTR ([<position> ,]<stränguttryck>, <sök-
uttryck>)

Resultat: Ger den första förekomsten av stränyen <sökutt-
ryck> i strängen <stränguttryck> och anger den
position i vilken förekomsten fanns. <position>
anger startpositionen för sökningen och måste
ligga inom intervallet 0 till 255. Om <position>
är större än längden av <stränguttryck>,<sträng-
uttryck> är en tom sträng eller om <sökuttryck>
inte påträffas ger INSTR värdet 0. Om <sökut-
tryck> är en tom sträng fås värdet 1.

Exempel:
```
10 X$="F0cdEF098"
20 PRINT INSTR (3,X$,"F"+"0"),INSTR ("gggg"+X$,X$)
 RUN
  6  5
 Ok
```

3.1.14 INT

Format: INT (<numeriskt uttryck>

Resultat: Ger det största heltalet som är mindre än eller
lika med <numeriskt uttryck>. Se även CINT och
FIX

Exempel:
```
PRINT INT (45.67);INT(-45.67)
 45         -46
Ok
```

3.1.15 LEN

Format: LEN (<stränguttryck>)

Resultat: Ger antalet tecken i strängen <stränguttryck>.
Även ej utskrivbara tecken och mellanslag räk-
nas.

Exempel:
```
PRINT LEN(12345"+CHR$(0)+"678")
  9
 Ok
```

3.1.16 LOG

Format: LOG <numeriskt uttryck>

Resultat: Ger naturliga logaritmen (e-logaritmen) för <nu-
meriskt uttryck> som måste vara större än noll.

Exempel:
```
PRINT LOG (45/7)
 1.86075
Ok
```

3.1.17 RND

Format: RND (<numeriskt uttryck>)

Resultat: Ger ett slumptal ur en likformig fördelning. Efter RUN fås alltid samma sekvens av slumptal. Med <numeriskt uttryck> kan följande alternativ väljas:

<uttryck>	alternativ
mindre än 0	startar om samma slumptalsföljd
lika med 0	upprepar senaste slumptal
större än 0	ger nästa slumptal

Exempel:
```
PRINT RND;RND(1);RND(0);RND(-1)
 .245121  .305003  .305003  .305003
Ok
```

3.1.18 SGN

Format: SGN (<numeriskt uttryck>)

Resultat: Ger följande resultat:

1	om <numeriskt uttryck>	är	större än 0
0	"	"	" lika med 0
-1	"	"	" mindre än 0

Exempel:
```
PRINT SGN(4.5);SGN(-4.5);SGN(0)
 1  -1   0
Ok
```

3.1.19 SIN

Format: SIN (<numeriskt uttryck>)

Resultat: Ger sinusvärdet av <numeriskt uttryck> som skall uttryckas i bågmått (radianer).

Exempel:
```
PRINT SIN(1.5)
 .997495
Ok
```

3.1.20 SQR

Format: SQR (<numeriskt uttryck>)

Resultat: Ger kvadratroten av <numeriskt uttryck>. Argumentet måste vara större än eller lika med 0.

Exempel:
```
10 FOR X=10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
  10          3.16228
  15          3.87298
  20          4.47214
  25          5
Ok
```

3.1.21 TAN

Format: TAN (<numeriskt uttryck>)

Resultat: Ger tangensvärdet av <numeriskt uttryck> som skall uttryckas i bågmått (radianer).

Exempel:
```
PRINT TAN(3.1415927/4)
 1
Ok
```

3.1.22 TIME

Format: TIME

Resultat: Ger ställningen hos datorns interna klocka. Klockan går från 0 till 65535. Klockan räknas var 50-dels sekund upp en enhet och när 65535 uppnås ger nästa 50-del värdet 0.

Exempel:
```
 10 INTERVAL ON
 20 ON INTERVAL=50*60 GOSUB 40
 30 GOTO 30
 40 PRINT TIME
 50 RETURN
 60 REM GER UTSKRIFT AV TIME MED EN MINUTS MEL-
    LANRUM
```

3.1.23 VAL

Format: VAL (<stränguttryck>

Resultat: Ger det numeriska värdet av det inledande talet i <stränguttryck>. Mellanslag och tabulatorsteg ignoreras.

Exempel:
```
PRINT VAL("20011 MALMÖ")
 20011
Ok

PRINT VAL ("MALMÖ 20011")
 0
Ok
```

3.2 Funktioner med strängresultat

3.2.1 BIN$

Format: BIN$ (<numeriskt uttryck>)

Resultat: Ger en sträng med det binära (talsystem med basen 2) värdet på <numeriskt uttryck> som avrundas till ett heltal innan beräkningen genomförs. Se även HEX$ och OCT$.

Exempel:
```
PRINT BIN$(16)
 10000
Ok
```

3.2.2 CHR$

Format: CHR$ (<numeriskt uttryck>)

Resultat: Ger en sträng med ett element som har ASCII-koden <numeriskt uttryck>. Värdet på <numeriskt uttryck> måste ligga mellan 0 och 255. Se även funktionen ASC.

Exempel:
```
PRINT CHR$(66)
B
Ok
```

3.2.3 HEX$

Format: HEX$ (<numeriskt uttryck>)

Resultat: Ger en sträng med det hexadecimala (talsystem med basen 16) värdet på <numeriskt uttryck> som avrundas till ett heltal innan beräkningen genomförs. Se även BIN$ och OCT$.

Exempel:
```
PRINT HEX$(74)
 4A
Ok
```

3.2.4 LEFT$

Format: LEFT$ (<stränguttryck> , <antal>)

Resultat: Ger en sträng som består av de <antal> första tecknen i <stränguttryck>. <antal> måste ligga inom intervallet 0 till 255. Om <antal> är större än längden av <stränguttryck> fås hela strängen <stränguttryck>. Om <antal> är lika med 0 fås en tom sträng (noll tecken).

Exempel:
```
PRINT LEFT$("1234567",4)
 1234
Ok
```

3.2.5 MID$

Format: MID$ (<stränguttryck> , <position> [, <antal>])

Resultat: Ger en sträng med längden <antal> tecken från och med <position> i strängen <stränguttryck>. <position> och <antal> måste ha värde mellan 0 och 256. Om <antal> utelämnas eller om det finns färre än <antal> tecken till höger om tecken <position>, ges alla tecken till höger från och med <position>. Om <position> större än längden av <stränguttryck>) ger MID$ en tom sträng.

Exempel:
```
 10 A$="GOD"
 20 B$="MORGON KVÄLL NATT"
 30 PRINT A$;MID$(B$,8,5)
 Ok
 RUN
 GOD KVÄLL
 Ok
```

3.2.6 OCT$

Format: OCT$ (<numeriskt uttryck>)

Resultat: Ger en sträng med det oktala (talsystem med basen 8) värdet på <numeriskt uttryck> som avrundas till ett heltal innan beräkningen genomförs. Se även HEX$ och OCT$.

Exempel:
```
PRINT OCT$(24)
 30
Ok
```

3.2.7 RIGHT$

Format: RIGHT$ (<stränguttryck> , <antal>)

Resultat: Ger en sträng som innehåller <antal> tecken, räknat från höger i <stränguttryck>. Om <antal> är större än eller lika med antalet tecken i <stränguttryck> får hela <stränguttryck>. Om <antal> är lika med noll fås en tom sträng.

Exempel:
```
PRINT RIGHT$("#"+"1234567",3+2)
#34567
Ok
```

3.2.8 SPACE$

Format: SPACE$ (<antal>)

Resultat: Ger en sträng som innehåller <antal> mellanslag. Värdet på heltalsuttrycket <antal> måste ligga mellan 0 och 255.

Exempel:
```
 10 FOR I=1 TO 5 : PRINT SPACE$(I);I : NEXT I
RUN
  1
   2
    3
     4
      5
Ok
```

3.2.9 STR$

Format: STR$ (<numeriskt uttryck>)

Resultat: Ger en sträng med samma innehåll som <numeriskt uttryck>. Se även VAL.

Exempel:
```
A$=STR$(584.39+27)
Ok
PRINT A$
 611.39
Ok
```

3.2.10 STRING$

Format: STRING$ (<antal> , <ASCII-värde> / <strängutt-ryck>)

Resultat: Ger en sträng med <antal> tecken. Samtliga tecken har antingen angivna ASCII-värdet eller också motsvarar de det första tecknet i <strängutt-ryck>. Om <stränguttryck> anges, får det inte vara en tom sträng.

Exempel:
```
X$=STRING$(10,45)
Ok
PRINT X$;"Rapport";STRING$(10,"-")
----------Rapport----------
Ok
```

3.3 IN- OCH UTFUNKTIONER

3.3.1 In- och utfunktioner som ger numeriskt resultat

3.3.1.1 CSRLIN

Format: CSRLIN

Resultat: Ger radnumret för den rad som markören befinner sig på. Värdet ligger mellan 1 (översta raden) och 25 (understa raden). Se även funktionen POS och instruktionen LOCATE.

3.3.1.2 CVI, CVS, CVD

Format: CVI (<sträng med 2 tecken>)
CVS (<sträng med 4 tecken>)
CVD (<sträng med 8 tecken>)

Resultat: Omvandlar strängar till numeriska värden.Tal som lagras som strängar med MKI$, MKS$ och MKD$ måste återomvandlas till tal med dessa funktioner. CVI ger ett heltal, CVS ett tal med enkel precision och CVD ett tal med dubbel precision.

3.3.1.3 DSKF

Format: DSKF (<skivenhet>)

Resultat: Ger antalet fria block på flexskivan i <skivenhet>.

3.3.1.4 EOF

Format: EOF (<filnummer>)

Resultat: Ger resultatet -1 (sant) när vid läsning av en sekventiell fil slutet på filen nås. Med denna funktion kan felet "Input past end" förhindras. Vid direktfiler kan EOF inte användas.

Exempel: Se bilaga

3.3.1.5 FPOS

Format: FPOS (<filnummer>)

Resultat: Ger numret på den senast bearbetade fysiska sektorn i angiven fil.

Exempel: Se bilaga

3.3.1.6 LOC

Format: LOC (<filnummer>

Resultat: Ger vid en direktfil numret på den post som bearbetades vid den sista GET eller PUT-satsen. Om inga GET eller PUT använts på filen fås 0. Vid sekventiella filer är resultatet det antal sektorer (128 bytes per sektor) som lästs eller skrivits sedan OPEN.

Exempel: Se bilaga

3.3.1.7 LOF

Format: LOF (<filnummer>

Resultat: Ger det antal poster i filen som lästes eller skrevs i det senaste blocket. Om filen inte är längre än ett block så motsvarar det filens verkliga längd. Med denna funktion kan fillängden hos direkt filer fastställes.

Exempel: Se bilaga

3.3.1.8 LPOS

Format: LPOS <uttryck>

Resultat: Ger aktuellt läge (position) för skrivhuvudet som det är angivet i skrivarbufferten. Detta behöver inte vara det verkliga (fysiska) läget. <uttryck> är ett argument utan betydelse.

Exempel: Se bilaga

3.3.1.9 POS

Format: POS <uttryck>

Resultat: Ger positionen för markören på bildskärmen. Första positionen (till vänster) har värdet 1. <uttryck> är ett argument utan betydelse.

3.3.1.10 SWITCH

Format: SWITCH

Resultat: Ger 0 när minnesblock 0 används och -1 när block 2 används.

3.3.2 In- och utfunktioner som ger strängresultat

3.3.2.1 ATTR$

Format: ATTR$ (<skivenhet> / #<filnummer> / <fil>)

Resultat: Ger en sträng som innehåller det aktuella attributet för flexskivan eller en (öppnad) fil. Attributen beskrivs vid instruktionen SET (avsnitt 2.3).

Exempel:
```
SET 1, "R"
Ok
PRINT ATTR$(1)
R
Ok
```

3.3.2.2 DSKI$

Format: DSKI$ (<skivenhet> , <spår> , <sektor>)

Version: Endast med flexskivor

Funktion: Läser en sektor från flexskivan

Resultat: Ger maximalt de första 255 bytes från sektorn. För att alla 256 bytes skall erhållas måste innan DSKI$ används en FIELD-instruktion med <filnummer> 0 utföras. Denna ger en buffert med 256 bytes (minst 2 strängvariabler). I denna buffert lagras sektorinformationen.

Exempel:
```
10  FIELD #0, 128 AS A$, 128 AS B$
20 DUMMY$=DSKI$(2,20,3)
```

3.3.2.3 INKEY$

Format: INKEY$

Resultat: Ger ett tecken när en tangent på tangentbordet trycks ner resp en tom sträng när ingen tangent trycks ner. Tecknet som läses av INKEY$ visas inte på bildskärmen. Alla tecken utom <CTRL-STOP> tas emot och vidarebefordras.

3.3.2.4 INPUT$

Format: INPUT$ (<antal> [, [#] <filnummer>])

Resultat: Ger en sträng med <antal> tecken som läses in från tangentbordet resp från en fil. Tecknen som läses av INPUT$ visas inte på bildskärmen. Alla tecken utom <CTRL-STOP> accepteras.

3.3.2.5 MKI$, MKS$, MKD$

Format: MKI$ (<heltalsuttryck>)
MKS$ (<uttryck med enkel precision>)
MKD$ (<uttryck med dubbel precision>)

Resultat: Konverterar numeriska uttryck till sträng-uttryck. Varje numeriskt uttryck som läggs in i en direktfilsbuffert måste konverteras till en sträng. MKI$ konverterar ett heltal till en 2--bytes sträng, MKS$ konverterar ett tal med enkel precision till en 4-bytes sträng och MKD$ konverterar ett tal med dubbel precision till en 8-bytes sträng. Omvandlingen tillbaka till talvärden görs med CVI, CVS och CVD.

Exempel: Se bilaga.

3.3.2.6 SPC

Format: SPC (<numeriskt uttryck>)

Resultat: Ger det antal blanktecken som motsvarar antalet i <numeriskt uttryck>. SPC kan bara användas tillsammans med [L]PRINT. Värdet på <uttryck> måste ligga mellan 1 och 255. Uttrycket efter SPC i PRINT-instruktionen skrivs direkt efter blanktecknen (motsvarar skiljetecknet ;). Se även SPACE$ och TAB.

Exempel:
```
PRINT "HIT"SPC(15) "DIT"
HIT               DIT
Ok
```

3.3.2.7 TAB

Format: TAB (<numeriskt uttryck>)

Resultat: Placerar markören på den position som anges av <uttryck>. Om detta är mindre än aktuell position sätts markören på nästa rad. TAB kan bara användas tillsammans med [L]PRINT. Värdet på <uttryck> måste ligga mellan 1 och 255. 1 anger den första positionen.

Exempel:
```
PRINT 5;TAB(8);7
 5         7
Ok
```

3.4 FUNKTIONER FÖR GRAFIK

3.4.1 PAD

Format: PAD (<padport>)

Resultat: PAD ger ett utvärde när vridkontroller används för styrning. <padport> är ett numeriskt uttryck vars värde måste ligga mellan 0 och 3.
<padport> 0 ger resultatet -1 när ytan berörs, annars 0
<padport> 1 ger värdet på x-koordinaten
<padport> 2 ger värdet på y-koordinaten
<padport> 3 ger resultatet -1 när tangenten trycka ner, annars 0

Exempel:
```
10  IF PAD(3) THEN PRINT "X=";PAD(1);" Y=";
    PAD(2) ELSE 10
```

3.4.2 PDL

Format: PDL (<vridkontrollnummer>)

Resultat: Ger värdet på vridkontrollen med <vridkontrollnummer> (ett numeriskt värde mellan 1 och 4). Om <vridkontrollnummer> är 1 eller 2 tas värdet från vridkontrollen i port 1, annars från vridkontrollen i port 2.

Exempel: PRINT PDL(2)

3.4.3 POINT

Format: POINT (<x> , <y>)

Resultat: Ger färgnumret på bildskärmspunkten med koordinaterna <x>,<y>.

Exempel: A=POINT(X,3*I)

3.4.4 STICK

Format: STICK (<styrspaknummer>)

Resultat: Ger den riktning i vilken styrspaken rörs. <styrspaknummer> kan ha följande värden:
0 ger riktningen hos markörförflyttningsplattan
1 " " " styrspak 1
2 " " " 2
Riktningen - STICK -kan ha följande heltalavärden:

```
        1
      8   2
    7   0   3
      6   4
        5
```

3.4.5 STRIG

Format: STRIG (<styrspaknummer>)

Resultat: Ger tillståndet hos tryckknappen på styrspaken resp mellanslagstangenten.

<styrspaknummer>	funktionsvärde
0	ger -1 om mellanslagstangenten tryckts ner
1	ger -1 när knappen på styrspak 1 tryckts ner
2	ger -1 när knappen p& styrspak 2 tryckts ner.

I alla andra fall är värdet 0

Exempel:

```
        .
        .
 100 TE=STRIG(1)
        .
        .
```

3.4.6 SPRITE$

Format: SPRITE$ (<spritenummer>)

Resultat: Ger den sträng som tilldelats spriten <sprite-nummer> eller en tom sträng om spriten ännu inte tilldelats någon sträng.

Exempel: A$=SPRITE$(12)

4. MINNESÅTKOMST OCH IN/UT-PORTAR

4.1 INSTRUKTIONER

4.1.1 DEF USR

Format: DEF USR [<siffra>] = <adress>

Resultat: Definierar startadressen för en assemblersubrutin

Användning: <siffra> kan vara valfri siffra mellan 0 och 9. Det är således möjligt att anropa 10 olika subrutiner i maskinspråk. Om man behöver fler subrutiner måste en ny startadress definieras med DEF USR. DEF USR= motsvarar DEF USR0=. Värdet på heltalsuttrycket <adress> är startadreasen för assemblersubrutinen. Se även CLEAR, funktionen USR och bilaga.

Exempel: DEF USR6=&HCF00 definierar den hexadecimala adressen CF00 som startadress för en assemblersubrutin

4.1.2 OUT

Format: OUT <ioportadress> , <databyte>

Resultat: Sänder <databyte> till porten med adressen <ioportadress>.

Användning: Denna instruktion kan, om den används felaktigt, försätta räknaren i processorn i ett odefinierat tillstånd. För att använda instruktionen krävs omfattande kunskaper om hur datorn fungerar internt.

Exempel: OUT 23,I%

4.1.3 POKE

Format: POKE <adress> , <databyte>

Resultat: <databyte> lagras i minnescellen med adressen <adress>

Exempel: POKE &H5500,128

4.1.4 WAIT

Format: WAIT <ioportadress>, <mask> [, <jämförelse>]

Resultat: En byte läses från in/utporten med adressen <ioportadress>, kombineras XOR med <jämförelse> (alla lika bitar blir då noll) och kombineras därefter AND med <mask>. Denna procedur upprepas tills dess att ett resultat fås som är skilt från noll. Om <jämförelse> inte anges antas det vara noll.

Exempel: WAIT 23,12,I%

4.1.5 VPOKE

Format: VPOKE <bildminnesadress> , <databyte>

Resultat: Skriver <databyte> i minnescellen med adressen <bildminnesadress> i bildminnet. <bildminnesadress> måste ligga inom intervallet 0 till 16383 (&H0 till &H3FFF).

Exempel: VPOKE &H400,68

4.2 FUNKTIONER

4.2.1 INP

Format: INP (<ioportadress>)

Resultat: Ger den från in/utport med adressen <ioportadresa> inlästa databyten.

Exempel: PRINT INP(23)

4.2.2 PEEK

Format: PEEK (<minnesadress>)

Resultat: Ger den byte som är lagrad i minnescellen med adressen <minnesadress>.

Exempel: PRINT PEEK(&H1000)

4.2.3 USR

Format: USR [<siffra>] (<uttryck>)

Resultat: Ger resultatet av den assemblersubrutin som definierades med DEF USR. Angående tilldelningen av <uttryck> till subrutinen, se bilaga.

Exempel: A%=USR6(23*8)

4.2.4 VARPTR

Format: VARPTR (<variabelnamn> / # <filnummer>)

Resultat: Ger den första (den med lägsta adressen) databyte som är tilldelad variabeln med <variabelnamn>. Innan VARPTR anropas måste variabeln ha tilldelats ett värde. Värdet på funktionen ligger inom intervallet -32768 till 32767. Adressen till den första variabel i listan A kan fås genom VARPTR(A(0)).

Om <filnummer> anges fås vid en sekventiell fil adressen till första minnescellen i skivbufferten och vid en direkt fil den till <filnummar> hörande FIELD-bufferten.

Varning: Anrop av VARPTR måste göras omedelbart före användningen av variabeladressen eftersom adresserna till listvariabler och strängar kan ändras.

Exempel: A%=USR(VARPTR(C$))

4.2.5 VPEEK

Format: VPEEK (<bildminnesadress>)

Resultat: Ger den databyte som är lagrad i bildminnescellen med adressen <bildminnesadress>.

Exempel: PRINT VPEEK(&H1000)

BILAGA A

ASCII-KODER

ASCII		ASCII		ASCII	
1	SOH	45	-	89	Y
2	STX	46	.	90	Z
3	ETX	47	/	91	Ä ([)
4	EOT	48	0	92	Ö (\)
5	ENQ	49	1	93	Å (])
6	ACK	50	2	94	Ü (^)
7	BEL	51	3	95	_
8	BS	52	4	96	é (`)
9	HT	53	5	97	a
10	LF	54	6	98	b
11	VT	55	7	99	c
12	FF	56	8	100	d
13	CR	57	9	101	e
14	SO	58	:	102	f
15	SI	59	;	103	g
16	DLE	60	<	104	h
17	DC1	61	=	105	i
18	DC2	62	>	106	j
19	DC3	63	?	107	k
20	DC4	64	É (@)	108	l
21	NAK	65	A	109	m
22	SYN	66	B	110	n
23	ETB	67	C	111	o
24	CAN	68	D	112	p
25	EM	69	E	113	q
26	SUB	70	F	114	r
27	ESC	71	G	115	s
28	FS	72	H	116	t
29	GS	73	I	117	u
30	RS	74	J	118	v
31	US	75	K	119	w
32	MELLANSLAG	76	L	120	x
33	!	77	M	121	y
34	"	78	N	122	z
35	#	79	O	123	ä (<)
36	$	80	P	124	ö (:)
37	%	81	Q	125	å (>)
38	&	82	R	126	ü (~)
39	'	83	S	127	DEL
40	(	84	T		
41	)	85	U		
42	*	86	V		
43	+	87	W		
44	,	88	X		

( ) = Tecken då svensk teckenmodul inte används

BILAGA B

FILER

1. Programfiler

Dessa framställs och bearbetas med hjälp av kommandon som SAVE, MERGE m.fl.

2. Datafiler

Datafiler bearbetas och behandlas genom tillämpningsprogram. Det finns två typer av datafiler:

1. Sekventiella filer
2. Direktfiler (kallas även randomfiler)

Skillnaden består i hur data i filerna hämtas och skrivs på filerna.

2.1 Sekventiella filer

Det är i allmänhet lättare att framställa filer av denna typ än direktfiler. Hämtningen av enskilda poster och datamängder är dock svårare och tar längre tid. Data lagras i den ordningsföljd de skrivs på filen och kan bara läsas i samma ordningsföljd. Det betyder att om man vill hämta data som finns mitt i filen så måste alla data från början av filen läsas igenom fram till dess att önskade data påträffas.

Sekventiella filer kan hanteras med följande anvisningar och funktioner:

OPEN, PRINT [USING] #, INPUT#, LINE INPUT#,
CLOSE, LOC, EOF

Exempel:

```
10 OPEN "sekdat" FOR OUTPUT AS #1   öppnar filen sek-
                                    dat för skrivning
20 PRINT #1,"sats1 PI=";3.1415      skriver strängen
                                    och talet på
                                    filen
30 PRINT #1,"sats2 PI=,";3.1415     skriver dessutom
                                    ett kommatecken
40 CLOSE #1                         stänger filen
```

Med ovanstående programavsnitt framställs en fil som innehåller två satser. Med ett annat programavsnitt kan dessa satser återläsas:

```
10 OPEN "sekdat" FOR INPUT AS #1   öppnar filen för
                                   läsning
20 IF EOF(1) THEN END              när slutet på
                                   filen nåtts skall
                                   den stängas
30 INPUT #1,A$                     läser en sträng ur
                                   filen
```

```
40 PRINT A$
50 GOTO 20
RUN
sats1 PI= 3.1415
sats2 PI=
3.1415
```

Skillnaden i utskriften beror på att ett kommatecken är inlagt i sats 2. Innehållet i denna sats består alltså av två uttryck. Om satsen

```
sats2 PI=,3.1415
```

skall skrivas ut (vilket i och för sig inte är särskilt meningsfullt) måste strängen inneslutas av citationstecken (mer om detta finns under INPUT):

```
A$=CHR$(34)      (34 är ASCII-värdet för ")
PRINT #1,A$;"sats2 PI=,";A$;3.1415
```

2.2 Direktfiler

Framställandet och användandet av direktfiler kräver åtskilligt mer programsteg än sekventiella filer. Läsningen av en post kan dock utföras direkt från skivan, utan att all information måste läsas in först.

För hanteringen av direktfiler används följande instruktioner:

OPEN, FIELD, GET, PUT, LSET, RSET, CLOSE

och dessa funktioner:

LOC, LOF, CVD, CVI, CVS, MKD$, MKI$, MKS$

Ett exempel på hur man framställer av en direktfil:

```
10 OPEN "2:test" AS #1     öpnnar filen 'test' på skiv-
                           enhet 2
20 FIELD #1,30 AS A$       30 bytes per post reserveras
30 FOR I%=1 TO 20
40 LSET A$="Detta är satsen" + STR(I%)
                           den angivna strängen läggs
                           in från vänster i A$ som
                           fylls ut med blanktecken
                           till 30 tecken
50 PUT #1                  utmatning av direktfilsbuf-
                           ferten. Eftersom inget post-
                           nummer finns, ökas det
                           senast använda med 1
60 NEXT I%
70 CLOSE #1
```

Hämtningen och läsningen av de olika poster som finns i den fil som framställdes ovan, kan göras på följande sätt:

```
10 OPEN "2:test" AS #1
20 FIELD #1,30 AS S$
40 INPUT "Post som skall läsas:";NR%
50 IF NR% > LOF(1) THEN PRINT "Största postnummer i
   filen är";LOF(1) : GOTO 40
60 IF NR% < 0 THEN PRINT "Postnummer mindre än 1 är
   inte tillåtna" : GOTO 40
70 IF NR%=0 THEN END
80 GET #1,NR%
90 PRINT S$
100 GOTO 40
```

På rad 50 kontrolleras att det angivna postnumret inte är större än det högsta numret i filen. Avslutande av programmet görs på rad 70 genom att 0 skrivs in som postnummer. Rad 80 hämtar direkt post med angivet nummar. Posten läggs sedan in i variabeln S$. Buffertminnets innehåll skrivs sedan ut med rad 90.

Eftersom endast strängar kan lagras i filbufferten måste tal omvandlas till strängar så att lagring blir möjlig. Detta görs med MK.$-funktionerna:

```
 5 MAXFILES=2
 10 OPEN "2:test" AS #1
 20 FIELD #1, 4 AS Z1$,8 AS Z2$,2 AS Z3$
 30 A!=100000
 40 B#=3.123456789
 50 C%=&H0F0F
 60 LSET Z1$=MKS$(A!)
 61 LSET Z2$=MKS$(B#)
 62 LSET Z3$=MKS$(C%)
 70 PUT #1
 80 OPEN "2:test" AS #2
 81 FIELD #2,4 AS Q1$,8 AS Q2$, 2 AS Q3$
 90 GET #2
 100 PRINT CVS(Q1$),CVD(Q2$),HEX$(CVI(Q3$))
 110 END
 RUN
  100000                3.123456789
 F0F
 Ok
```

I ovanstående program reserverar rad 20 utrymme för talen i buffertminnet (antal bytes). Rad 60 omvandlar talen till strängar som med raderna 60-62 placeras in i respektive buffertvariabler. Rad 70 skriver sedan buffertminnets innehåll på filen. På rad 80 öppnas filen för läsning, rad 90 läser in posten, rad 100 omvandlar strängvärdena till talvärden och skriver ut dem på skärmen.

BILAGA C

RESERVERADE ORD

Följande ord, uttryck och instruktioner i BASIC är reserverade. Det betyder att de inte får användas i eller som variabelnamn eller i andra sammanhang. Om du försöker använda något av de uppräknade orden som ett variabelnamn eller som en del av ett sådant, kommer du att få ett felmeddelande.

ABS	ELSE	LPRINT	RUN
ANB	END	LSET	SAVE
ASC	EOF	MAX	SCREEN
ATN	EQV	MDM	SET
ATTR$	ERASE	MERGE	SGN
AUTO	ERL	MID$	SIN
BEEP	ERR	MKD$	SOUND
BIN$	ERROR	MKI$	SPACE$
BLOAD	EXP	MKS$	SPC
BSAVE	FIELD	MOD	SPRITE$
CDBL	FILES	MON	SQR
CH$	FIX	MOTOR	STEP
CINT	FN	NAME	STICK
CIRCLE	FOR	NEW	STOP
CLEAR	FPOS	NEXT	STR$
CLICK	FRE	NOT	STRIG
CLOSE	GET	OCT$	STRING
CLOAD	GOTO	OFF	SWAP
CLS	GOSUB	ON	SMITCH
CMD	HEX$	OPEN	TAB
COLOR	IF	OR	TAN
CONT	IMP	OUT	THEN
COPY	INKEY$	PAINT	TIME
COS	INP	PAD	TO
CSAVE	INPUT	PDL	TROFF
CSNG	INSTR	PEEK	TRON
CSRLIN	INT	PLAY	USING
CVD	IPL	POINT	VAL
CVI	KEY	POKE	VARPTR
CVS	KILL	POS	VPEEK
DATA	LEFT$	PRESET	VPOKE
DEF	LEN	PRINT	WAIT
DEFDBL	LET	PSET	WIDTH
DEFINT	LFILES	PUT	XOR
DEFSNG	LINE	READ	
DEFSTR	LIST	REM	
DELETE	LLIST	RENUM	
DIAL	LOAD	RESTORE	
DIM	LOC	RESUME	
DRAW	LOCATE	RETURN	
DSKF	LOF	RIGHT$	
DSKI$	LOG	RND	
DSKO$	LPOS	RSET	

BILAGA D

Programexempel med PUT SPRITE, ON KEY och STICK

I nedanstående program kan färgerna på föremålen (spritarna) på skärmen ändras genom att funktionstangenterna används. En tryckning på funktionstangent 10 följd av en tryckning på någon av funktionstangenterna 1 till 6 ger färgerna 10 till 15. Spriten på skärmen kan sedan förflyttas med hjälp av markörkontrollplattan.

```
10 S$=SPACE$(32)
20 FOR I%=1 TO 16
30 READ A$
40 MID$(S$,I%,1)=CHR$(VAL("&B"+LEFT$(A$,8)))
50 MID$(S$,I%+16)=CHR$(VAL("B&"+RIGHT$(A$,8)))
60 NEXT I%
70 SCREEN 2,3
80 SPRITE$(1)=S$
85 X=0 : Y=0 : F=2 : B=0
90 PUT SPRITE 1,(128-8,96-8),2
100 KEY ON
110 ON KEY GOSUB 150,151,152,153,154,155,156,157,158,159
120 A%=STICK(0)
130 IF A% THEN FOR I=1 TO 300 : NEXT : ON A% GOSUB
    141,142,143,144,145,146,147,148, : GOTO 120 ELSE 120
141 X=0 : Y=10 : GOTO 149
142 X=10 : Y=10 : GOTO 149
143 X=10 : Y=0 : GOTO 149
144 X=10 : Y=-10 : GOTO 149
145 X=0 : Y=-10 : GOTO 149
146 X=-10 : Y=-10 : GOTO 149
147 X=-10 : Y=0 : GOTO 149
148 X=-10 : Y=10 : GOTO 149
149 PUT SPRITE 1, STEP (X,-Y),F : X=0 : Y=0 : B=0
    RETURN
150 F=1+B MOD 16 : GOTO 149
151 F=2+B MOD 16 : GOTO 149
152 F=3+B MOD 16 : GOTO 149
153 F=4+B MOD 16 : GOTO 149
154 F=5+B MOD 16 : GOTO 149
155 F=6+B MOD 16 : GOTO 149
156 F=7+B MOD 16 : GOTO 149
157 F=8+B MOD 16 : GOTO 149
158 F=9+B MOD 16 : GOTO 149
159 B=9 : RETURN
```

```
200 DATA 0000000000000000
210 DATA 0000000110000000
220 DATA 0000001001000000
230 DATA 0000010000100000
240 DATA 0000100000010000
250 DATA 0001000000001000
260 DATA 0010000000000100
270 DATA 0111111111111110
280 DATA 0100000000000010
290 DATA 0100000000000010
300 DATA 0100000000000010
310 DATA 0100000000000010
320 DATA 0100000000000010
330 DATA 0100000000000010
340 DATA 0111111111111110
350 DATA 0100000000000010
```

Beskrivning av vissa programrader:

Rad 10:	Reserverar ett utrymme på 32 bytes för en sprite med 16 * 16 tecken
Rad 40:	Framställer spritarna ur det från DATA-satserna inlästa bitmönstret
Rad 70:	Ställer in lågupplösningsgrafiken för 16 * 16 spritar
Rad 80:	Definiering av spriten
Rad 85:	Initiering av alla variabler (nollställning)
Rad 90:	Placerar ut spriten mitt på skärmen
Rad 100:	Alla funktionstangenter avläses
Rad 110:	Hopp till respektive delrutin
Rad 120:	Avläsning av markörkontrollplatta
Rad 141-148:	Sätter koordinaterna för spriten beroende på hur markörkontroliplattan trycks ner
Rad 149:	Placerar ut spriten på skärmen , därefter nollställs koordinaterna
Rad 150-158:	Sätter om färgen på spriten när en funktionstangent trycks ner
Rad 200-350:	Spritemönster, 0=bakgrundsfärg, 1=förgrundsfärg

BILAGA E

Escape-sekvenser för skärmredigering

Följande sammanställning avser teckenkombinationer som ger resultat på skärmen. "Escape" framställs med ASCII-koden 27 som skrivs med CHR$. Observera att <ESC>-tangenten på tangentbordet inte kan användas för att generera dessa teckenkombinationer. Escape-sekvenserna kan även användas i program.

<ESC> E	Tömmer bildskärmen
<ESC> J	Tömmer bildskärmen från markören fram till sidans slut
<ESC> K	Fyller en rad från markören till radslutet med blanktecken
<ESC> l	Fyller en rad från radbörjan till markören med blanktecken
<ESC> L	Infogning av en rad
<ESC> M	Raderar en rad
<ESC> A	Flyttar markören en rad uppåt
<ESC> B	Flyttar markören en rad neråt
<ESC> C	Flyttar markören en rad åt höger
<ESC> D	Flyttar markören en rad åt vänster
<ESC> H	Flyttar markören till "hemmaläget" (övre högra hörnet)
<ESC> p	Kopplar in omvänd video
<ESC> q	Kopplar bort omvänd video
<ESC> x 5	Markören visas ej
<ESC> y 4	Markören visas med halv höjd
<ESC> y 5	Markören visas normalstor
<ESC> Y <r><k>	Placering av markören CHR$(32+<radnummer>) CHR$(32+<kolumnnummer>)

Exempel:

```
PRINT CHR$(27) +"p" ; "Detta är omvänd video" ; CHR$(27)+"q"
```

Med kommandot <ESC> p kopplas omvänd video in. Det innebär att alla tecken därefter skrivs omvänt (mörk skrift mot ljus bakgrund, dock beroende på vilka färger som ställt in på för- och bakgrund). Med <ESC> q kopplas den omvända skriften bort igen.

BILAGA F

Parameteröverföringar vid USR-funktionen

När funktionen USR anropas, innehåller A-registret i processorn följande värden:

Argument för USR	Värde på A
heltal	2
sträng	3
tal med enkel noggrannhet	4
tal med dubbel noggrannhet	8

I HL-registret står adressen till BASIC-ackumulatorn, som lagrar informationen på följande sätt:

Om argumentet är ett heltal finns den lägre delen av byten (LSB) på adressen HL+2, den högre delen av byten (MSB) på adressen HL+3 i minnet.

Om argumentet är en sträng innehåller BASIC-ackumulatorn också ett helt tal (se föregående stycke) som skall läsas som adressen till ett minnesområde på 3 bytes som i sin tur beskriver strängen. Den första byten av detta område innehåller stränglängden. De bägge följande bytes (LSB, MSB) är adressen till strängen. Vid strängkonstanter pekar denna adress mot arbetsminnet (en viss försiktighet bör hör iakttagas eftersom strängen kan fyllas ut med tomma strängar).

Exempel: USR ("ABC"+"DEF")

Ack	Ack+1	Ack+2	Ack+3	
??	??	adrL	adrH	desc=adrH*256+adrL

desc	desc+1	desc+2	
6	strL	strH	strad=strH*256+strL

	strad	strad+1	strad+2	....	strad+5	
HEX	41	42	43	44	45	46
CHR	A	B	C	D	E	F

Om argumentet är ett tal med enkel noggrannhet kan BASIC-ackumulatorn tolkas på följande Sätt

Ack	Ack+1	Ack+2	Ack+3
Exponent	<--------mantissa ------------>		
	MS8	....	LSB

Sjunde biten i exponenten innehåller tecknet för talet (1= negativt, 0=positivt). Bitarna 0 till 6 innehåller exponenten + 64. De åtta positionerna ligger i packat BCD-format i de följande tre minnesadresserna.

Om argumentet är ett tal med enkel noggrannhet gäller det som sagts ovan. Skillnaden gentemot tal med dubbel noggrannhet består i att mantissan förlängs med fyra bitar.

```
     Ack        Ack+1      Ack+2        Ack+7
  Exponent      <------- mantissa ------------>
                MSB        ....          LSB
```

Exempel: USR(2.345678)

```
        Ack         Ack+1     .....    Ack+7
HEX      41          23  45  67  80  00  00
DEC      65         0.2345678
```

Argumentet är alltså 0.2345678*10Ü(65-64) = 2.345678

BILAGA G

Videoramedresser

Adresser till textbildskärm:

```
     0   1   2   3    ....   37   38   39

    40                ....             79
     .
     .
     .
   920                ....            959
```

Teckenmatriser i BASICs teckengenerator:

```
   2048 - 4095   alltid två bytes per tecken.
                 Därvid används bara de 6 högsta bitarna.
                 Bit 7 (MSB) finns på tecknets vänstra sida.

   2048 - 2055   Teckenkoden 0 i videoRAM motsvarar blank
   2056 - 2063         "     1 "    "        "        !
   2064 - 2071         "     2 "    "        "        "
   2072 - 2079         "     3 "    "        "        #
        .
        .        osv i enlighet med ASCII-koden
        .
        - 4095
```

Teckenkoden 95 ger på nytt ett blanktecken.

På teckenkod 96 börjar hela ASCII-sekvensen om igen men med omvänd video.

Teckenkod 191 i videoRAM är en kopia av det tecken som markören ligger på.

På teckenkod 192 börjar grafiktecknen.

Teckenkoderna 248 till 255 kan användas fritt. De kan nås med hjälp PRINT-instruktionen och från och med CHR$(216) till och med CHR$(223).

Exempel på matrisen för utropstecknet.

VideoRAMkod 1:

VRAM-adress	Binärt innehåll	Decimalt
2056	00100000	32
2057	00100000	32
2058	00100000	32
2059	00100000	32
2060	00100000	32
2061	00000000	0
2062	00100000	32
2063	00000000	0

Exempel på hur tecken kan förändras (här blanktecknet):

```
 10 FOR I%=0 TO 7 : READ A$
 20 VPOKE 2048+I%,VAL("&B"+A$)
 30 NEXT I%
 40 DATA 00000000
 50 DATA 10000100
 60 DATA 11001100
 70 DATA 10110100
 80 DATA 10110100
 90 DATA 11001100
100 DATA 10000100
110 DATA 00000000
```

De lägsta två bitarna används inte eftersom ett tecken bara består av en matris med 6 gånger 8 tecken.

Efter det att detta program körts kommer alla blanktecken (mellanslag) att se ut på det sätt som framgår av matrisen i datasatserna.

Man kan således lägga in egna teckenuppsättningar i video-RAM förutom den normala BASIC-teckenuppsättningen. De måste då börja på en 2K-bytegräns (jämna multiplar av 2048. Om man därefter använder PRINT måste man beakta den kodifiering som inte överensstämmer med ASCII-uppsättningen. Omkopplingen görs med följande kommando:

```
OUT 129,(generatoradress/2048) : OUT 129,132
```

På "generatoradress" börjar teckengeneratormatriserna i videoRAM. Återgång till BASIC-tecknen fås med:

```
OUT 129,1 : OUT 129,132
```

Man kan även återställa BASIC-tecknen och teckengeneratoradressen 2048 med kommandot SCREEN 0.

Ovanstående är en enkel metod att generera tecken som normalt inte finns på tangentbordet, t.ex Å,Ä och Ö som saknas på tidigare versioner av SV-318/SV-328.

BILAGA H

Felmeddelanden och felkoder

Nr	Felmeddelande
1	NEXT without FOR En variabel i en NEXT-instruktion motsvaras inte av någon tidigare utförd FOR-instruktion.
2	Syntax error En rad har påträffats som innehåller någon felaktig teckenföljd (som till exempel ett ensamt parentestecken, felstavat kommando eller instruktion, felaktig interpunktion e.d). BASIC går automatiskt över till redigeringsläge på raden som orsakade felet.
3	RETURN without GOSUB En RETURN-instruktion påträffas som inte svarar mot någon tidigare GOSUB-instruktion.
4	Out of DATA En READ-instruktion har utförts utan att det finns några DATA-satser med olästa data kvar i programmet.
5	Illegal function call En parameter utanför tillåtet området har angivits till en matematisk funktion eller strängfunktion. Felet kan också uppstå som resultatet av: 1. Ett negativt eller onormalt stort index 2. Argumentet noll eller negativt till LOG 3. Argumentet negativt till SQR 4. En negativ mantissa till heltal skilt från noll 5. Anrop till en USR-funktion utan att startadressen angivits 6. Felaktigt argument till MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR eller ON....GOTO
6	Overflow Resultatet av en beräkning är för stort för att kunna användas av BASIC. Om underflow (för litet eller negativt tal) uppstår blir resultatet negativt och programmet fortsätter utan felmeddelande.
7	Out of memory Minnet fullt. Ett program är för stort eller har för många FOR-NEXT-slingor eller GOSUB, för många variabler eller uttryck som är alltför komplicerade.
8	Undefined line Någon av instruktionerna GOTO, GOSUB, IF...THEN...ELSE eller DELETE hänvisar till en rad som inte finns.

9 Subscript out of range
Felaktigt index. Ett element i en lista anges med ett index som antingen är utanför listans dimensioner eller med fel antal index.

10 Redimensioned array
Två DIM-utryck har angivits för samma lista eller listan har fått nytt DIM efter det att standardvärdet 10 har accepterats för listan.

11 Division by zero
Division med noll har uppkommit i ett uttryck eller en operation har medfört att noll har upphöjts till ett negativt tal.

12 Illegal direct
Ett otillåtet uttryck har använts som direktkommando.

13 Type mismatch
Ett stränguttryck har tilldelats ett numeriskt uttryck eller tvärtom. En funktion som förväntar sig ett numeriskt argument har tilldelats ett strängargument eller tvärtom.

14 Out of string space
På grund av strängvariabler har kvarvarande minnesutrymme överskridits. BASIC tilldelar strängar minnesutrymme efterhand, tills dess att minnet är slut.

15 String too long
Försök har gjorts att tilldela en sträng mer än 255 tecken vilket är maximum.

16 String formula too complex
Ett stränguttryck är för långt eller för komplicerat. Uttrycket måste brytas ner i kortare uttryck.

17 Can't continue
Försök har gjorts att fortsätta ett program som:
1. Avbrutits på grund av fel
2. Har förändrats under ett uppehåll i utförandet
3. Inte finns

18 Undefined user function
En USR-funktion anropas innan den definierats (DEF-uttryck saknas).

19 Device I/O error
Fel vid in- eller utmatning från ansluten enhet.

20 Verify error
Fel vid kontrolläsning av lagrat program eller fil.

21 No RESUME
En felhanteringsrutin har påträffats som inte innehåller något RESUME.

22 RESUME without ERROR
Ett RESUME har påträffats innan någon felhanterings-rutin upptäckts.

23 Unprintable error
Felmeddelande för inträffat fel finns inte.

24 Missing operand
Ett uttryck innehåller en operator utan efterföljande operand.

25 Line buffer overflow
Försök att skriva en rad med för många tecken.

26 / 49 Unprintable error
Felmeddelande för inträffat fel finns inte.

50 Field overflow
Ett FIELD-uttryck försöker att tilldela fler bytes än vad som var tillåtet för postlängden för en viss fil.

51 Internal error
En intern fel funktion har inträffat i BASIC. Meddela återförsäljaren och tala om under vilka förhållanden felet uppstod.

52 Bad file number
Ett uttryck eller kommando hänvisar till en fil som inte är öppnad med OPEN eller vars nummer går utanför gränserna som angavs med MAXFILE.

53 File not found
Ett LOAD, KILL eller OPEN-kommando hänvisar till en fil som inte finns på aktuell skiva.

54 File already open
Kommandot eller instruktionen OPEN ges för en fil som redan är öppnad eller ett KILL-kommando ges för en öppnad fil.

55 Input past end
En INPUT-instruktion utförs efter det att alla data i filen har lästs eller utförs för en blank (tom) fil. För att undvika detta fel, använd EOF-funktionen (End Of File) för att upptäcka filslut.

56 Bad file name
En otillåten form på filnamn används tillsammans med ett LOAD, SAVE, KILL eller OPEN-uttryck (t ex ett filnamn med för många tecken).

57 Direct statement in file
Ett direktkommando har påträffats vid laddning av en fil i ASCII-format. Laddningen avbryts.

58 Sequential after PUT
Försök har gjorts att läsa/skriva på en sekventiell fil efter PUT. PUT kan bara användas vid direktfiler (random acceasfiler).

59 Sequential I/O only
Läsning och skrivning kan bara göras på sekventiell fil.

60 File not open
Filen har inte öppnats (med OPEN).

Övriga felkoder ger "Unprintable error".

BILAGA I

Matematiska funktioner

Härledda funktioner

Funktioner som inte finns inbyggda i SV BASIC kan beräknas med hjälp sv nedanstående formler.

Funktion	Ekvivalent i SV BASIC
Sekant	Sec(X) = 1/COS(X)
Cosekant	Csc(X) = 1/SIN(X)
Cotangent	Cot(X) = 1/TAN(X)
Arcsin	Arcsin(X) = ATN(X/SQR(-X*X + 1))
Arccosin	Arccos(X) = -ATN(X/SQR(-X*X + 1)) + 1.5708
Arcsec	Arcsec(X) = ATN (X/SQR(X*X-1)) + SGN(SGN(X)-1)*1.5708
Arccosec	Arccsc(X) = ATN(X/SQR(X*X-1))+(SGN(X)-1)* 1.5708
Arccot	Arccot(X) = ATN(X) + 1.5708
Sinh	Sinh(X) = (EXP(X) - EXP(-X))/2
Cosh	Cosh(X) = (EXP(X) + EXP(-X))/2
Tanh	Tanh(X) = (EXP(-X)/EXP(X) + EXP(-X))*2 + 1
Sech	Sech(X) = 2/(EXP(X) + EXP(-X))
Cosech	Cosech(X) = 2/(EXP(X) - EXP(-X)
Coth	Coth(X) = EXP(-X)/(EXP(X) - EXP(-X))*2 + 1
Ar sinh	Arcsinh(X) = LOG(X + SQR(X*X + 1))
Ar cosh	Arccosh(X) = LOG(X + SQR(X*X - 1))
Ar tanh	Arctanh(X) = LOG((1 + X)/(1 - X))/2
Ar sech	Arcsech(X) = LOG((SQR(-X*X + 1) + 1)/X)
Ar cosech	Arccsch(X) = LOG((SGN(X)*SQR(X*X + 1) + 1)/X)
Ar coth	Arccoth(X) = LOG((X + 1)/(X - 1))/2

SAKREGISTER

	Sid
ABS, numerisk funktion	73
ASC, numerisk funktion	73
ATN, numerisk funktion	73
ATTR$, in/utfunktion	85
AUTO, kommando	16
BEEP, instruktion	34
BIN$, strängfunktion	79
BLOAD, instruktion	34
BSAVE, instruktion	35
CDBL, numerisk funktion	73
CHR$, strängfunktion	79
CINT, numerisk funktion	74
CIRCLE, grafikinstruktion	60
CLEAR, instruktion	21
CLICK, instruktion	35
CLOAD, instruktion	36
CLOAD?, instruktion	36
CLOSE, instruktion	37
CLS, instruktion	37
COLOR, grafikinstruktion	61
CONT, kommando	16
COPY, instruktion	38
COS, numerisk funktion	74
CSAVE, instruktion	38
CSNG, numerisk funktion	74
CSRLIN, in/utfunktion	82
CVD, in/utfunktion	82
CVI, in/utfunktion	82
CVS, in/utfunktion	82
DATA, instruktion	21
DEF FN, instruktion	22
DEF USR, minneshanteringsinstruktion	89
DEFDBL, instruktion	23
DEFINT, instruktion	23
OEFSNG, instruktion	23
DEFSTR, instruktion	23
DELETE, kommando	17
DIM, instruktion	23
DRAW, grafikinstruktion	62
DSKF, in/utfunktion	82
DSKI$, in/utfunktion	85
DSKO$, instruktion	39
Dubbel precision	5
END, kommando	17
Enkel precision	5
EOF, in/utfunktion	82
ERASE, instruktion	24
ERL, numerisk funktion	74
ERR, numerisk funktion	74
ERROR, instruktion	24

	Sid
EXP, numerisk funktion	75
FIELD, instruktion	39
FILES, instruktion	40
FIX, numerisk funktion	75
FOR...NEXT, instruktion	25
FPOS, in/utfunktion	83
FRE, numerisk funktion	75
Funktionsoperatorer	12
GET, instruktion.	41
GET, grafikinstruktion	64
GOSUB...RETURN, instruktion	26
GOTO, instruktion	27
HEX$, strängfunktion	79
IF, instruktion	28
INKEY$, in/utfunktion	85
INP, minneshanteringsfunktion	91
INPUT, instruktion	41
INPUT#, instruktion	42
INPUT$, in/ut funktion	85
INSTR, numerisk funktion	76
INT, numerisk funktion	76
INTERVAL, avbrottsinstruktion	69
Inmatning	1
IPL, instruktion	43
KEY, instruktion	43
KEY, avbrottsinstruktion	69
KEY LIST, instruktion	43
KILL, instruktion	44
Konstanter	3
Kontrolltangenter	2
Konvertering	7
LEFT$, strängfunktion	29, 79
LEN, numerisk funktion	76
LET, instruktion	29
LFILES, instruktion	40
LINE, grafikinstruktion	64
LINE INPUT, instruktion	44
LINE INPUTff, instruktion	45
LIST, kommando -	18
Listvariabler	6
LLIST, kommando	18
LOAD, instruktion	45
LOC, in/utfunktion	83
LOCATE, instruktion	46
LOF, in/utfunktion	83
LOG, numerisk funktion	76
Logiska operatorer	11
LPOS, in/ut funktion	83
LPRINT, instruktion	45
LPRINT USING, instruktion	45

	Sid
LSET, instruktion	46
MAXFILES, instruktion	47
MERGE, kommando	18
MID$, strängfunktion	80
MID$, instruktion	29
MKD$, in/utfunktion	86
MKI$, in/utfunktion	86
MKS$, in/utfunktion	86
MOTOR ON/OFF, instruktion	47
NAME, instruktion	47
NEW, kommando	19
OCT$, strängfunktion	80
ON ERROR, avbrottsinstruktion	70
ON INTERVAL, avbrottsinstruktion	70
ON KEY, avbrottsinstruktion	71
ON SPRITE, avbrottsinstruktion	71
ON STOP, avbrottsinstruktion	71
ON...GOSUB, instruktion	30
ON...GOTO, instruktion	30
OPEN, instruktion	48
Operatorer	8
OUT, minneshanteringsinstruktion	89
PAD, grafikfunktion	87
PAINT, grafikinstruktion	65
PDL, grafikfunktion	87
PEEK, minneshanteringsfunktion	91
PLAY, instruktion	49
POINT, grafikfunktion	87
POKE, minneshanteringsinstruktion	89
POS, in/utfunktion	83
PRESET, grafikinstruktion	65
PRINT, instruktion	51
PRINT USING, instruktion	52
PRINT#, instruktion	55
PRINT# USING, instruktion	55
PSET, grafikinstruktion	65
PUT, grafikinstruktion	65
PUT SPRITE, grafikinstruktion	67
Radformat	3
READ, instruktion	31
REM, instruktion	31
RENUM, kommando	19
RESTORE, instruktion	32
RESUME, avbrottsinstruktion	72
Relationsoperatorer	10
RIGHT$, strängfunktion	33, 80
RND, numerisk funktion	77
RSET, instruktion	46
RUN, kommando	20
SAVE, instruktion	55
SCREEN, grafikinstruktion	66

Sid

SET, instruktion 56
SGN, numerisk funktion 77
SIN, numerisk funktion 77
SOUND, instruktion 56
SOUND ON/OFF, instruktion 58
SPACE$, strängfunktion 81
SPC, in/utfunktion 86
SPRITE, avbrottsinstruktion 70
SPRITE$, grafikinstruktion 67
SPRITE$, grafikfunktion 88
Spill 10
SQR, numerisk funktion 77
STICK, grafikfunktion 87
STOP, kommando 20
STOP, avbrottsinstruktion 69
STR$, strängfunktion 81
STRIG, grafikfunktion 88
STRING$, strängfunktion 81
SWAP, instruktion 32
SWITCH, instruktion 58
SWITCH, in/utfunktion 84

TAB, in/utfunktion 86
TAN, numerisk funktion 78
TIME, numerisk funktion 78
TROFF, instruktion 33
TRON, instruktion 33
USR, minneshanteringsfunktion 91
Utrymmesbehov 7
Uttryck 8
VAL, numerisk funktion 78
VARPTR, minneshanteringsfunktion 91
Variabelnamn 5
Variabler 5
VPEEK, minneshanteringsfunktion 92
VPOKE, minneshanteringsinstruktion 90
WAIT, minneshanteringsinstruktion 90
WIDTH, instruktion 58
WRITE#, instruktion 58

LEFT$ 33
RIGHT$ 33