

LUXOR

Dator ABC 806

Manual BASIC II



ABC 806[®]

LUXOR
Datörer

Förord

Denna manual beskriver programspråket BASIC II för ABC 806. Manualen är inte avsedd som lärobok i BASIC, utan läsaren förutsätts ha erfarenhet av programmering sedan tidigare.

Kapitel 1 presenterar programspråket BASIC. I kapitel 2 behandlas uppbyggnaden av datorprogram i BASIC II.

Kapitlen 3, 4 och 5 beskriver de data, som kan hanteras i ett program.

I kapitel 6 talas om det praktiska arbetet med BASIC II. Detta kapitel innehåller många råd och tips om hur man skriver och ändrar ett program.

Kapitel 7 beskriver, hur datorns instruktioner och kommandon används direkt, utan att bilda något program. Detta är särskilt användbart vid felsökning.

Kapitlen 8, 9 och 10 innehåller utförliga beskrivningar av de kommandon, funktioner och instruktioner, som ingår i BASIC II. De flesta beskrivningar är kompletterade med exempel, som visar hur varje programdel byggs upp.

Kapitel 11 behandlar grafik med attributhantering.

Kapitel 12 beskriver användningen av högupplösningssgrafik.

Kapitel 13 talar om hur man använder datorns funktionstangenter.

Kapitel 14 beskriver skillnaderna mellan ABC 806 och ABC 800.

Kapitel 15 innehåller en lista med felmeddelanden och kommentarer till dessa.

Kapitel 16, som är markerat med grått i kanten, innehåller kortfattade beskrivningar av instruktionerna, funktionerna och kommandona, samlade i bokstavsordning. Listan i kapitel 16 är avsedd som ett uppslagsregister, som ger syntax och sidhänvisningar till de mera omfattande beskrivningarna tidigare i manualen (kapitel 8, 9 och 10).

Kapitel 17 är en litteraturlista och kapitel 18 innehåller ett antal bilagor. Manualen avslutas med ett sakregister.

Systemprogramvara levererad av Luxor Datorer AB får endast mångfaldigas och användas tillsammans med av Luxor levererade produkter.

Innehåll

1	Om programspråket BASIC	1
2	Program	2
2.1	Radnummer	2
2.2	Kommentarer	3
2.3	BASIC-satser	3
2.4	Uttryck	4
2.5	Logiska enheter	5
2.6	Felhantering	5
3	Data	7
3.1	Talområden	7
3.2	Konstanter	7
3.3	Variabler	8
3.4	Indexerade variabler (vektorer) och DIM-satsen	9
3.5	Filhantering	10
3.5.1	Att öppna en fil	10
3.5.2	Dataöverföring till/från en fil	10
3.5.3	Att stänga en fil	11
4	Heltal och flyttal	12
4.1	Matematiska operationer	12
4.2	Heltalsaritmetik	13
4.3	In- och utmatning av heltal och flyttal	13
4.4	Användardefinierade funktioner	13
4.5	Heltal som logiska variabler	14
4.6	Logiska operationer på heltalsdata	14
5	Teckensträngar	15
5.1	Strängkonstanter	15
5.2	Strängvariabler	15
5.3	Strängvektorer	15
5.4	Strängstorlek	15
5.5	Strängfunktioner	16
5.6	Strängaritmetik	16
5.7	Stränginmatning	16
5.8	Utmatning av strängar	17
5.9	Relationsoperatorer på strängar	17
6	Att arbeta med BASIC II	18
6.1	Hur man skriver programrader	18
6.1.1	Fritt format i varje sats	18
6.1.2	Skrivsätt	18
6.1.3	Att rätta fel	18
6.1.4	Att ta bort en rad	19
6.1.5	Att ändra en programrad	19
6.2	Att ändra i ett program	19
6.3	Att köra ett program	20
6.4	Vilka satser används var?	20
6.5	Deklarationer	22

7	Direktanvändning	23
8	Kommandon	24
9	Instruktioner	33
10	Funktioner	62
10.1	Matematiska funktioner	62
10.2	Strängfunktioner	67
10.3	Övriga funktioner	72
11	Grafik med attributhantering	78
11.1	TELEDATA-grafik	78
11.1.1	ASCII kod för tecken, grafik och argument	79
11.1.2	Grafikkarta för 40 tecken	80
11.2	Attributhantering	81
11.3	Programexempel	81
11.4	Instruktioner	82
12	Högupplösningsgrafik	84
12.1	Högupplösningsgrafik på ABC 815	85
12.2	Instruktioner	85
12.3	Exempel	87
12.4	Färgvalstabell	89
13	Funktionstangenter	91
14	Skillnader ABC 806—ABC 800	92
15	Felmeddelanden	93
16	Kommando- och instruktionssammanfattning	96
17	Litteraturförteckning	108
18	Bilagor	109
	Bilaga 1 I/O-portar	109
	Bilaga 2 Minneskarta	110
	Bilaga 3 Tangentbordslayout, ASCII-koder	112
19	Sakregister	114

1 Om programspråket BASIC

För att tala om för datorn, vad den ska göra, använder vi ett formellt språk eller programmeringsspråk. Det formella språket är uppbyggt av vissa nyckelord på engelska. BASIC är ett av de allra enklaste programspråk, som finns. Alla instruktioner, kommandon och funktioner är lätta att förstå och använda. Trots detta är språket tillräckligt omfattande för att ge en smidig och effektiv problemlösning.

Namnet BASIC kommer av Beginner's All-purpose Symbolic Instruction Code. Programspråket BASIC var från början avsett enbart för elementär undervisning i programmering. Det visade sig dock fungera så bra, att det numera används i många tillämpningar.

Många programspråk bygger på principen att hela programmet först matas in i datorn och därefter översätts till maskinspråk (kompileras). Vid kompileringen undersöks, om programmet innehåller formella fel. Datorn skriver då ut en fellista. Programmeraren rättar programmet och matar in det på nytt. Det kompileras igen och eventuella fel skrivs ut.

När man arbetar med BASIC, däremot, finns ett program, som kallas interpretator (BASIC-tolk), i datorn. För varje programrad, som skrivs, kontrollerar BASIC-tolken om den är formellt korrekt. Formella fel ger en felutskrift direkt på skärmen, medan man matar in programmet. Man kan också när som helst under inmatningen provköra sitt program. Detta brukar kallas interaktiv programmering. I många fall är detta mycket effektivare än traditionell programmering med kompilering.

Den interaktiva programmeringen löser givetvis inte alla problem. När programmet är fritt från formella fel, återstår kanske logiska fel, som man bara kan upptäcka vid provkörning.

BASIC har grammatikregler, precis som alla andra språk. Grammatiken för ett programspråk är dock mycket enklare än motsvarande regler för ett naturligt språk. Nedanstående exempel beräknar medelvärdet av fem tal, som användaren matar in. Här kan Du se, hur språket är uppbyggt.

Exempel

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 END
```

BASIC II innehåller dels de elementära instruktioner, som behövs för att skriva enkla program, dels också instruktioner och funktioner, som gör det möjligt att skriva mera avancerade program med högre effektivitet. Nyckelordet för denna typ av programmering är effektivitet. Efterhand som användaren skaffar sig större programmeringserfarenhet, ökar effektiviteten och man vill använda alltmer avancerad databehandling. BASIC II är så omfattande, att man kan lösa praktiskt taget vilket problem som helst.

BASIC II ger möjlighet till AUTOSTART. AUTO START-funktionen behandlas utförligt i bruksanvisningen för flexskiveenheten.

2 Program

Ett program består av programrader, som innehåller satser. Satserna innehåller instruktioner till BASIC-tolken. Varje programrad börjar med ett unikt radnummer. Efter radnumret skrivs en eller flera BASIC-satser. Mellan två satser på samma rad ska finnas ett kolon (:). Programraderna utförs i nummerordning. Varje sats inleds av ett nyckelord, som anger vilken operation, som ska utföras.

Satsen ger datorn en instruktion (i detta fall **PRINT**):

```
30 PRINT S
```

Vissa instruktioner kräver operander, som specificerar vilken variabel eller vilken programdel, instruktionen ska utföras på. I exemplet ovan är operanden "S".

Sista satsen i ett program är en **END**-sats:

```
10 INPUT A,B,C,D,E  
20 LET S=(A+B+C+D+E)/5  
30 PRINT S  
40 END
```

Satsen **END**, som talar om för datorn, att programmet är slut, är inte obligatorisk, men bör vara den sista sats, som genomlöps i programmet. Den medför, att alla filer stängs, men variablerna behåller sina värden.

2.1 Radnummer

Varje programrad inleds med ett radnummer, som bl.a. fyller följande funktioner:

1. Anger i vilken ordning satserna genomlöps. Det är likgiltigt i vilken ordning de matas in.
2. Gör det möjligt, att ändra den normala genomloppsordningen med hjälp av t.ex. satser som **GOTO** eller **GOSUB**. Radnumret fungerar som lägesangivelse och hoppadress.
3. Gör det möjligt, att ändra vilken rad som helst, utan att resten av programmet påverkas.

Du, som programmerar, väljer radnummer. Det får vara ett godtyckligt heltal från 1 t.o.m. 65 535.

Varje rad ska ha ett, unikt radnummer. Datorn använder radnumren för att identifiera och hålla reda på instruktionerna. Om man skriver in en ny rad med ett radnummer, som redan finns, ersätts den existerande raden med den nyskrivna.

Satserna kan skrivas in i godtycklig ordning. Datorn tar hänsyn till nummerordningen. Om Du t.ex. skriver in raderna i följande ordning: 30; 10, 20, så sorterar datorn dem: 10, 20, 30.

Raderna bör numreras i jämna multipler av 10 eller 5, för att det ska vara lätt att föra in nya rader. Det finns kommandon för automatisk radnumrering (**AUTO**) och för omnumrering av raderna (**RENUMBER**).

2.2 Kommentarer

BASIC II har två sätt att beteckna kommentarer i programmet.

1. **REM**-satser (som i standard BASIC)

```
10 A=7: REM sju
```

2. Utropstecken. Skall inte avgränsas med kolon.

```
10 A=7 ! sju
```

Kommentarer är en del av ett BASIC-program. De skrivs ut, när programmet listas på skärmen eller på skrivare. Kommentarsatserna exekveras inte, utan hoppas över då programmet körs. Vilka tecken som helst (utom RETURN) kan användas i kommentaren. För att göra det lättare att se kommentarerna, brukar man markera dem med något väl synligt tecken:

```
10 REM ***** MÄTDATA IN *****  
200 GOSUB 3100 ! # # # TILL DIFF-BERÄKNING # # #  
3240 RETURN ! & & & & X7=DIFFERENSEN & & & &
```

OBS!

En kommentar kan inte avslutas med kolon. Hela satsen, inklusive kolon, betraktas som en kommentar.

```
150 REM *** Beräkningsmetod *** : LET R1=3.52E2.1+Y5
```

Den avslutande satsen blir inte utförd. Hela raden behandlas som en kommentar.

2.3 BASIC-satser

Efter radnumret följer en BASIC-sats. Satsens nyckelord anger, vilken typ av sats det är. På detta sätt vet BASIC-tolken (det program, som översätter BASIC till datorinstruktioner) vad, som ska utföras och hur de eventuella data, som står efter nyckelordet, ska behandlas.

Användaren får lov att skriva flera BASIC-satser på samma programrad. Mellan två satser ska finnas ett kolon (:). En rad med flera satser exekveras något snabbare än samma satser på var sin rad, vilket kan vara väsentligt i vissa tillämpningar.

```
100 PRINT A,B,C
```

är en enkel programrad

```
200 LET X=X+1 : PRINT X : IF Y=1 THEN 100
```

är en sammansatt programrad med de tre satstyperna:

LET, PRINT och IF-THEN

Praktiskt taget alla instruktioner kan användas i sammansatta programrader. De undantag, som finns, är klart angivna i beskrivningarna av respektive instruktion.

OBS!

Enligt god programmeringssed bör man dock skriva endast en sats per rad.

2.4 Uttryck

Ett uttryck består av symboler, som representerar konstanter, variabler, funktioner eller någon kombination av dessa, åtskilda av aritmetiska, relations- eller logiska operatorer.

Exempel: Aritmetiska uttryck

```
4.123  
3%+A%  
B6*(C**3+1.0)
```

Relationsuttryck

```
X>Y  
Y8>=0  
A=B
```

Logiska uttryck

```
(A<1.) AND (B=5)  
((B<A) OR (D=C)) AND B/A<>D/C
```

Aritmetiska uttrycks värde är antingen heltal eller flyttal.

Relationsuttryck har värdet sant eller falskt, då de båda ingående värdena jämförts.

Logiska uttryck har värdet sant eller falskt, beroende på om villkoren i uttrycket är uppfyllda eller inte.

Beträffande stränguttryck se kapitel 5.

2.5 Logiska enheter

BASIC II gör användaren oberoende av vilka in/ut-enheter, som faktiskt används. Den aktuella enheten tilldelas ett filnummer. Filnumret kan behandlas som en logisk enhet och hanteras med instruktionerna **OPEN**, **PREPARE** och **CLOSE**. Filnumret kan representera t.ex. skrivare eller en fil på flexskiva.

Exempel:

```
10 - -  
20 OPEN "PR:" AS FILE 2 !Öppna skrivaren  
30 - -  
40 - -  
50 CLOSE 2 !Stäng skrivaren  
60 END
```

Observera: **CON**: läggs upp som standardenhet för hantering av tangentbord och bildskärm som logisk enhet.

2.6 Felhantering

Under körning av ett program kan BASIC upptäcka vissa typer av fel. Dessa fel kan t.ex. vara beräkningsfel (t.ex. division med 0) eller I/O-fel (t.ex. när en end-of-file kod läses som indata till en **INPUT**-sats). Huvudregeln är, att då ett fel upptäcks, avbryts körningen av programmet och ett felmeddelande skrivs ut.

I vissa tillämpningar kan det vara nödvändigt, att programmet fortsätter genomlöpas även efter ett fel. För att uppnå detta, kan användaren skriva en **ON ERROR GOTO** <radnr>-instruktion i programmet. Programmet hoppar då till ett underprogram, som börjar på den angivna raden. Underprogrammet kan t.ex. innehålla en felhanterare, som analyserar det fel, som inträffar då programmet körs.

ON ERROR GOTO-satsen placeras i programmet före alla exekverbara satser, som ska behandlas av underprogrammet.

När ett fel inträffar i ett program, undersöks om någon **ON ERROR GOTO**<radnr>-sats har genomlöpts. Om detta inte har skett, skrivs ett felmeddelande ut på skärmen och programmet avbryts. Om en **ON ERROR GOTO** <radnr>-sats har genomlöpts innan felet, sker uthopp till det angivna radnumret, där användarens felhanteringsrutin t.ex. kan undersöka funktionen **ERRCODEs** värde, för att få veta, vilket fel det rör sig om och behandla det.

I vissa delar av programmet ska kanske alla fel hanteras av systemet istället för av användarens felrutin. En sådan del av programmet ska inledas med satsen

radnummer **ON ERROR GOTO**

Datorn tar då hand om alla fel och ger felmeddelande enligt kapitel 15.

Felhanteringen avslutas med **RESUME**. **RESUME**-satsen fungerar ungefär som **RETURN** i slutet av ett vanligt underprogram. Återhopp sker till uthoppspunkten (om det finns någon) i den sats, som orsakade feluthoppet. Om man måste hoppa in i någon annan rad i programmet, för att kunna fortsätta, anges radnumret i **RESUME**-satsen.

Exempel på felhantering:

```
10 ON ERROR GOTO 100 !Vid felaktig inmatning hoppa till rad 100
20 INPUT "Ålder, Vikt "A,W
30 ON ERROR GOTO !Stäng av felhanteraren
40 STOP
100 PRINT !Felhanterare
110 PRINT "Felaktig inmatning!"
120 RESUME !Återhopp till rad 20
```

3 Data

3.1 Talområden

Flyttal

Talområdet för flyttal är det största talområdet i BASIC och omfattar:
 $\pm 1 E-38 - \pm 1 E+38$

I enkel precision (**SINGLE**) finns sju signifikanta siffror och i dubbel precision (**DOUBLE**) 16 signifikanta siffror. Talen avrundas internt, så att de anges med aktuell precision.

Talen kan matas in och skrivas ut på tre olika format:

Exempel: 153, 34.53, 134E-2

Heltal

Talområdet för heltal är $-32\ 768 - +32\ 767$

Strängar

En teckensträng kan innehålla hur många tecken som helst.

Observera!
Strängar, som används vid strängaritmetik, får maximalt innehålla 125 tecken inklusive + eller - tecken och decimalpunkt.

3.2 Konstanter

Numeriska konstanter behåller sitt konstanta värde genom hela programmet. De kan vara positiva eller negativa. Numeriska konstanter kan skrivas enligt följande:

Exempel: +3%
 -4.765
 1234.6
 -.0001

De tre sista konstanterna i exemplet lagras som flyttal, eftersom de inte följs av något %-suffix. Man bör alltid ange antingen %-suffix eller decimalpunkt, för att undvika onödiga omvandlingar och för att göra dokumentationen mera lättläst.

3.3 Variabler

Variabler är data, vars värde kan ändras medan programmet körs. En variabel betecknas med ett specifikt variabelnamn.

Ett variabelnamn består av en bokstav eller en bokstav följt av en siffra. När man använder **EXTEND**, kan man använda långa variabelnamn (alfanumeriska tecken med en bokstav först).

Tillåtna tecken A,B,C,.....,Ö
 0,1,2,.....,9

Ett namn kan också ha prefixet **FN**, som betecknar en användarfunktion, suffixet **%**, som betecknar heltal, suffixet **∩**, som betecknar en sträng eller ett indexsuffix, som består av index inom parentes.

Ett stränguttryck har ett värde, som består av en teckenföljd, där varje tecken upptar en byte. En sträng kan skrivas antingen som en teckenföljd inom citationstecken eller som en strängvariabel betecknad med ett variabelnamn följt av suffixet **∩**.

Data av olika typ bör inte blandas i samma sats. Använd i första hand heltal, där det är möjligt. Heltal tar mindre minnesutrymme och behandlas snabbare i datorn.

Samma namn kan förekomma med olika prefix och suffix i ett och samma program. Det representerar då helt olika variabler. Flyttalsvariabeln **A** är inte samma sak som heltalsvariabeln **A%**. Namnet **A** kan användas på följande, skilda sätt:

A	flyttalsvariabeln A
A%	heltalsvariabeln A%
A∩	strängvariabeln A∩
A(d)	flyttalsvektorn A med dimensionen d
A%(d)	heltalsvektorn A% med dimensionen d
A∩(d)	strängvektorn A∩ med dimensionen d
FNA	flyttalsfunktionen A
FNA%	heltalsfunktionen A%
FNA∩	strängfunktionen A∩

I **EXTEND**-mod kan man skriva enligt följande:

Antal	flyttalsvariabeln Antal
Antal%	heltalsvariabeln Antal%
Antal∩	strängvariabeln Antal∩
Antal(d)	flyttalsvektorn Antal med dimensionen d
Antal%(d)	heltalsvektorn Antal% med dimensionen d
Antal∩(d)	strängvektorn Antal∩ med dimensionen d
FNAntal	flyttalsfunktionen Antal
FNAntal%	heltalsfunktionen Antal%
FNAntal∩	strängfunktionen Antal∩

Variabler kan tilldelas värden med bl.a. instruktionerna **LET**, **INPUT** och **READ**. Innan programmet genomlöps, nollställs variablerna, om de inte är skyddade av en **COMMON**-deklaration. Det är inte nödvändigt att tilldela en variabel ett startvärde, annat än när den måste ha ett startvärde skilt från 0.

3.4 Indexerade variabler (vektorer) och DIM-satsen

Man får använda indexerade variabler (vektorer) förutom de enkla variablerna. Indexerade variabler ger programmeraren utökade möjligheter att hantera listor, tabeller, matriser eller liknande variabelgrupper. Variablerna får indexeras i hur många dimensioner som helst.

En indexerad variabel betecknas med ett namn, som består av ett godtyckligt valt, tillåtet variabelnamn följt av ett antal heltal inom parentes. En lista kan t.ex. skrivas $A(I)$, där I går från 0 till 5:

$A(0), A(1), A(2), A(3), A(4), A(5)$

Programmeraren kan nu använda vart och ett av listans sex element. De kan anses utgöra en endimensionell, algebraisk vektor enligt nedan:

$A(0)$
 $A(1)$
 $A(2)$
 $A(3)$
 $A(4)$
 $A(5)$

En tvådimensionell matris $B(I,J)$ kan definieras på samma sätt. Man kan avbilda den så här:

$B(0,0)$	$B(0,1)$	$B(0,2)$...	$B(0,J)$
$B(1,0)$	$B(1,1)$	$B(1,2)$...	$B(1,J)$
$B(2,0)$	$B(2,1)$	$B(2,2)$...	$B(2,J)$
$B(3,0)$	$B(3,1)$	$B(3,2)$...	$B(3,J)$
.....				
$B(I,0)$	$B(I,1)$	$B(I,2)$...	$B(I,J)$

Varje index i en vektorvariabel måste vara ett heltal. Om index är ett flyttal, avrundas det till ett heltal.

En **DIM**-sats används ofta för att ange maximalt index i en vektor. Om en vektorvariabel används utan att vara dimensionerad, antas varje index i den vara dimensionerat till maximala värdet 10. Begynnelsevärdet för varje index kan ändras med instruktionen **OPTION BASE**. Normalt är detta värde 0, men det kan ändras till 1, så att en standardvektor omfattar 10 element i varje dimension i stället för 11. Alla **DIM**-satser ska placeras i början av programmet.

3.5 Filhantering

BASIC II innehåller instruktioner för datalagring och åtkomst av data på kassett eller flexskiva.

En datafil består av en följd av data, som överförs mellan ett BASIC-program och en yttre I/O-enhet. Den yttre enheten kan t.ex. vara skrivare, kassett eller flexskiva. Med instruktionen **OPEN** anger man, vilka enheter, som är tillgängliga och deras referensnummer. Varje enhet har ett namn, som används för att identifiera den i systemet. Ex: Drivenhet 0 i flexskiveenheten betecknas DR0:.

Varje datafil har ett unikt namn. Ex. ABC123.BAC betecknar en fil på flexskiva. Inom programmet anropas filen med ett filnummer. Filnumret anges i programmet med någon av instruktionerna **PREPARE** eller **OPEN**. Dessa instruktioner öppnar kanalen, d.v.s. upprättar förbindelsen. För att stänga en sådan dataöverföringskanal används instruktionen **CLOSE**. Instruktionerna **INPUT** och **PRINT** eller **GET** och **PUT** används för att utföra dataöverföringen.

Då en fil öppnas, skapas en buffertarea i systemet. All dataöverföring går via buffertutrymmen.

3.5.1 Att öppna en fil

En ny fil skapas med **PREPARE**. Om filen redan finns, öppnas den med **OPEN**.

Exempel

```
10 OPEN "FIL1.FIL" AS FILE 1
```

öppnar den existerande filen med namnet FIL1.FIL för in- och utmatning med filnummer 1.

3.5.2 Dataöverföring till/från en fil

Dataöverföring sker direkt mellan den interna kanalen och angiven strängvariabel eller värdet av angivet uttryck. All dataöverföring avser antingen en byte (bitgrupp) eller en sträng (tecknen före vagnretur).Följande instruktioner kan användas:

INPUT #	läser ett värde till en variabel eller sträng från filpekarens läge till vagnretur.
INPUT LINE #	läser ett värde till en strängvariabel inklusive vagnretur (CR) och radframmatning (LF).
PRINT #	skriver variabelns innehåll på filen
GET # COUNT	läser en byte eller angivet antal bytes från filpekarens läge
PUT #	skriver en post på filen
POSIT #	flyttar filpekaren till önskat läge

Om inget filnummer anges i **GET**-instruktionen, läser den från tangentbordet. Om **COUNT** ej anges, läser **GET** en byte, d.v.s. ett tecken.

Exempel

```
20 GET # 1,D2☐ COUNT 6%
```

läser på filen med filnummer 1 från filpekarens läge sex tecken framåt. Dessa tecken läggs i strängen D2☐.

Instruktionen **POSIT** används för att positionera filpekaren på angiven position i filen. Antalet tecken räknas alltid från filens början (position 0).

Exempel:

fil 1 innehåller ABCDEFGHIJK

```
|  
|  
50 POSIT # 1,5  
60 GET # 1,A☐ COUNT 3  
70 PRINT A☐  
80 END  
RUN  
FGH
```

Funktionen **POSIT**(<filnr>) läser filpekarens läge. I detta fall har **POSIT**(1) värdet 8, då exemplet ovan har genomlöpts. **POSIT** har flyttalsvärde, för att kunna omfatta långa filer.

VARNING

POSIT skall inte användas i samband med sekventiella filer, d.v.s. filer som hanteras med **PRINT** och **INPUT/INPUT LINE**.

3.5.3 Att stänga en fil

Dataöverföring på en fil avslutas inte korrekt, förrän filen stängs. Då överförs innehållet i buffertarean, och filen får ett filslut (EOF).

Filen kan stängas på två sätt:

CLOSE 2 Stänger filen med filnummer 2

CLOSE Stänger samtliga filer

4 Heltal och flyttal

Alla numeriska värden (variabler och konstanter), som används i ett BASIC-program, lagras normalt som flyttal. Om heltal används i ett program, kan man spara mycket minnesutrymme genom att deklarerar dessa data som heltal. Heltal lagras i 2 byte, medan flyttal kräver 4 alternativt 8 byte beroende på precision. Aritmetiken är också snabbare för heltal än för flyttal. Heltalsdeklarationen för konstanter, variabler eller funktioner görs med ett procenttecken, % efter namnet.

Exempel $A\%$, $FNX\%(Y)$, -8% , $Z3\%$

Observera, att %-tecknet måste finnas med varje gång ett heltal ska genereras, annars antas talet vara ett flyttal. Variabeln $H\%$ kan aldrig vara samma sak som variabeln H .

Om ett tal ska upphöjas till en heltalsexponent, skall exponenten alltid tydligt anges som heltal.

Här ovan förutsätts att BASIC II arbetar i sitt normala läge, d.v.s. **FLOAT**. Med instruktionen **INTEGER** kan man få BASIC II att förutsätta, att alla tal är heltal, så länge inget annat anges.

4.1 Matematiska operationer

Om mer än en operation ska utföras i ett och samma uttryck, tillämpas nedanstående prioritetsordning mellan de olika operatorerna, där nr 1 har högst prioritet.

1. Beräkningar inom parentes utförs först. Resultatet av en sådan beräkning används sedan i de följande stegen. För kapslade parenteser: $(A+(B*(C**3)))$ gäller, att den innersta beräknas först.
2. I övrigt gäller följande prioritetsordning:
 - a. Inbyggda eller användardefinierade funktioner
 - b. Exponentiering ($**$)
 - c. Multiplikation och division ($*$, $/$)
 - d. Addition, subtraktion ($+$, $-$) och minustecken som negativt förtecken.
 - e. Relationsoperatorer ($=$, $<>$, $>=$, $<=$, $<$, $>$)
 - f. **NOT**
 - g. **AND**
 - h. **OR** och **XOR**
 - i. **IMP**
 - j. **EQV**Så är t.ex. $-A**B$ ett tillåtet uttryck och betyder det samma som $-(A**B)$. Detta innebär, att $-2**2$ blir -4 . Uttrycket $A**B$ är inte tillåtet. Det ska skrivas som $A**(-B)$.
3. Bortsett från parentesuttryck utförs alla operationer med samma prioritet från vänster till höger, så som uttrycket har skrivits.

4.2 Heltalsaritmetik

Heltalsaritmetiken utförs modulo 2×16 . Ett BASIC-heltal kan ligga mellan -32768 och $+32767$. Heltalsrepresentationen kan betraktas som en obruten cirkel, där -32768 ligger efter $+32767$.

Heltalsdivision innebär, att en eventuell rest trunkeras. Jämför dock funktionen **MOD**, som gör det möjligt att ta fram resten från divisionen.

Exempel $3\%/4\%=0\%$ och $283\%/100\%=2\%$

En och samma beräkning kan innehålla både heltal och flyttal. Resultatet får då den form, som passar resultatvariabelns deklaration.

Exempel `LET B%=Z%+3/X`

Resultatet avrundas, så att B% får ett heltalsvärde.

4.3 In- och utmatning av heltal och flyttal

In- och utmatning av heltal görs på samma sätt som för flyttal.

De tal, som kan skrivas med högst sju siffror (vid **SINGLE**) eller högst sexton siffror (vid **DOUBLE**) skrivs utan exponentialformat.

De flyttal, som har ett heltalsvärde, skrivs som heltal men lagras fortfarande som flyttal.

Om ett tal omfattar mer än sju resp. sexton siffror, skrivs det automatiskt ut på följande sätt:

`[−].nE[−]m`

där n är ett tal med upp till sju siffror och m är en exponent med högst två siffror.

Vid inmatning kan alla de format användas, som används vid utmatning. Om en heltalsvariabel tilldelas ett flyttalsvärde, avrundas värdet till heltal.

4.4 Användardefinierade funktioner

En funktion deklarerar som heltalsfunktion, genom att ett procenttecken (%) sätts direkt efter funktionsnamnet.

Exempel `10 DEF FNV%(X%)=X%*(Z%+X%)`

En flyttalsfunktion kan skrivas på följande sätt:

Exempel `10 DEF FNV(X%)=X%*(Z+X%)`

4.5 Heltal som logiska variabler

Heltalsvariabler eller heltalsuttryck kan användas i **IF**-satser på samma sätt som logiska uttryck. Alla värden skilda från 0 tolkas då som "sant" och värdet 0 som "falskt". De logiska operatorerna (**AND**, **OR**, **NOT**, **XOR**, **IMP** och **EQV**) utför bitorienterade operationer på logiska data (eller heltalsdata).

OBS!

Normalt motsvaras värdet "sant" av heltalet **-1** och "falskt" av heltalet **0**. Detta gäller de logiska värden, som kommer från **BASIC II**.

4.6 Logiska operationer på heltalsdata

BASIC II gör det möjligt för användaren att kombinera heltalsvariabler och heltalsuttryck med logiska operatorer, som arbetar databit för databit.

Sanningstabellen nedan gäller för de logiska operatorerna. A är tillståndet hos en viss databit i ett heltal och B är tillståndet hos motsvarande databit i ett annat heltal.

A	B	A AND B	A OR B	A XOR B	A EQV B	A IMP B	NOT A
1	1	1	1	0	1	1	0
1	0	0	1	1	0	0	0
0	1	0	1	1	0	1	1
0	0	0	0	0	1	1	1

Resultatet av en logisk operation är ett heltalsvärde, som bildas då motsvarande databitar i de båda heltalen utsätts för operationer enligt tabellen ovan.

En heltalsvariabel eller flyttalsvariabel kan tilldelas värdet av detta resultat.

Exempel

```
10 A% = 13% OR 14%
20 PRINT A%
RUN
15
```

AND, **OR**, **XOR**, **EQV**, **IMP** och **NOT** kan användas på variabler och beräknade uttryck. De opererar då på en databit i taget. Flyttalsvariabler, som behandlas med logiska operatorer, omvandlas till heltal innan operationen utförs.

5 Teckensträngar

BASIC II bearbetar inte bara numeriska data utan också data i form av teckensträngar. En teckensträng är en följd av tecken, t.ex. alfanumeriska och skiljetecken.

5.1 Strängkonstanter

På samma sätt som man kan använda numeriska konstanter, är också strängkonstanter tillåtna. Strängkonstanter omges av ett av två avgränsningstecken, citations-tecken (") eller apostrof ('). Vill man använda något av dessa tecken inne i texten, ska det dubbeltecknas, om det samtidigt används som gränstecken. Om det andra tecknet används som gräns, går det bra att skriva strängen, som den ska se ut.

Strängen MATS' BIL skrivs "MATS' BIL" eller 'MATS' BIL'.

5.2 Strängvariabler

Tillåtet variabelnamn för en strängvariabel är ett godtyckligt, tillåtet variabelnamn, följt av tecknet $\$$, valutatecken. Detta tecken motsvaras i engelskspråkig litteratur av dollartecken.

Exempel $D\$$, $G1\$$ är enkla strängvariabler.
 $A\$ (8)$, $G5\$ (M,N)$, $J\$ (I)$ är vektorsträngar.

OBS!

Samma namn, utan $\$$, definierar en helt annan variabel, som kan användas i samma program som strängvariabeln.

Exempel F , $F\$$ och $F\%$ kan användas parallellt.

5.3 Strängvektorer

Instruktionen **DIM** används för att definiera en- eller flerdimensionella strängvektorer. Följande **DIM**-satser finns att välja på:

Exempel **DIM** $W\$ (2,4)=8$ stränglängd 8, högsta index 2 och 4
DIM $R5\$ (9,9)$ strängl. max. 80, högsta index 9 och 9
DIM $NAMN\$ (7,6,3,2)=10$ strängl. 10, fyrdimensionell matris med högsta index 7,6,3 och 2

5.4 Strängstorlek

Längden av en icke dimensionerad strängvariabel sätts automatiskt till den aktuella längden första gången strängen tilldelas ett värde, som är skilt från "tom sträng" (<>").

Om färre än 80 tecken används, får strängen standardlängden 80 tecken.

Varje sträng, även vektorer, har två längder:

1. Den maximala längden, d.v.s. det antal bytes, som reserverats för strängen.
2. Den aktuella längden, d.v.s. det antal bytes, som för tillfället används. Den aktuella längden kan växla mellan noll och maximala längden. Det är den aktuella längden, som kan undersökas med **LEN**.

Om en sträng tilldelas värdet "tom sträng" (""), sätts dess aktuella längd till noll. Inga andra åtgärder vidtas.

När en sträng tilldelas ett värde skilt från "tom sträng" och har en maximal längd skild från noll, kontrolleras att det nya värdet får plats. Om den reserverade längden inte räcker, fås en felutskrift. Om längden räcker till, tilldelas strängvariabeln det nya värdet, och den aktuella längden blir då aktuellt antal bytes.

5.5 Strängfunktioner

Ett helt sortiment av funktioner används på teckensträngar. Med hjälp av dessa kan man i programmet utföra aritmetiska operationer på numeriska strängar, sammanfoga två strängar, ta fram en del av en sträng, ta reda på antalet tecken i strängen, skapa den teckensträng som motsvarar ett visst tal eller tvärtom, söka efter en viss delsträng i en större sträng o.s.v. Se avsnitt 10.2.

5.6 Strängaritmetik

Med strängaritmetiken kan man behandla numeriska strängar med aritmetiska operatörer. På detta sätt kan man få högre precision i beräkningarna. Numeriska strängvariablers namn måste avslutas med ett α -tecken. Numeriska strängkonstanter måste avgränsas av citationstecken (") eller apostrofer(').

De numeriska strängar, som ska behandlas med strängaritmetiken, får ha en största längd av 125 tecken.

5.7 Stränginmatning

Precis som övriga variabler kan strängvariabler ges värden med instruktionerna **READ**, **DATA** och **INPUT**.

Exempel

```
10 INPUT "Adress, Namn?" A4 $\alpha$ , A5 $\alpha$ 
är samma sak som
10 PRINT "Adress, Namn";
20 INPUT A4 $\alpha$ , A5 $\alpha$ 
```

Instruktionen **INPUT LINE** är mycket användbar, då man vill mata in strängar. Den tar emot en rad från tangentbordet.

Exempel

```
45 INPUT LINE D $\alpha$ 
```

Exempel

```
210 READ A,B,C
290 DATA 17,14,61
```

Variablerna tilldelas följande värden:

```
A=17
B=14
C="61"
```

Instruktionen **INPUT** används för att mata in numeriska värden på samma sätt som teckensträngar.

5.8 Utmatning av strängar

När en sträng skrivs med en **PRINT**-instruktion, matas enbart tecknen inom citationstecken eller apostrofer ut. Avgränsningstecknen syns aldrig.

Exempel

```
10 PRINT "Javisst!"
RUN
Javisst!
```

Strängar kan också lagras i filer på någon yttre enhet.

5.9 Relationsoperatorer på strängar

Relationsoperatorer, när de tillämpas på strängar, avser alfabetisk ordningsföljd.

Exempel

```
15 IF A(I%)<A(I%+1%) THEN GOTO 115
```

När rad 15 genomlöps, sker följande: $A(I\%)$ jämförs med $A(I\%+1\%)$. Om $A(I\%)$ kommer före $A(I\%+1\%)$ i alfabetisk ordning sker uthopp till rad 115.

När strängarna har olika längd, jämförs den kortare strängen med de första tecknen i den längre, tecken för tecken. När den kortare strängen är slut, gäller det resultat, som då finns, utom i det fall att de första tecknen i den längre strängen är lika med den kortare. Då anses den kortare strängen vara minst.

Relationsoperatorer, som används på strängvariabler

Operator	Exempel	Betydelse
=	$A\<B\<$	strängarna $A\<$ och $B\<$ är lika
<	$A\<<B\<$	$A\<$ är före $B\<$ i alfabetisk ordning
<=	$A\<<=B\<$	$A\<$ är före $B\<$ eller lika med $B\<$
>	$A\<>B\<$	$A\<$ är efter $B\<$ i alfabetisk ordning
>=	$A\<>=B\<$	$A\<$ är efter $B\<$ eller lika med $B\<$
<>	$A\<<>B\<$	strängarna $A\<$ och $B\<$ är ej lika

6 Att arbeta med BASIC II

6.1 Hur man skriver programrader

6.1.1 Fritt format i varje sats

BASIC tillåter "fritt textformat". Datorn bryr sig inte om extra mellanslag i en sats. Dessa fyra satser är helt likvärdiga:

```
30 PRINTS
30 PRINT S
30PRINTS
30P RINT S
```

Då programmen skrivs ut på skärm eller skrivare, listar datorn dem alltid på sitt vanliga sätt, oavsett hur Du skrivit in satserna.

OBSERVERA! I följande fall är mellanslagen viktiga:

- **EXTEND**-mod
- **DATA**-satser

6.1.2 Skrivsätt

En rad kan antingen utföras omedelbart (direktexekveras) eller lagras i programminnet för att exekveras senare och slutligen lagras på yttre minnesenhet (flexskiva eller kassett).

Då en sats är färdigskriven, ska **RETURN**-tangenter tryckas ned.

Exempel:

10 INPUT A,B,C,D,E	(tryck RETURN)
20 LET S=(A+B+C+D+E)/5	(tryck RETURN)
30 PRINT S	(tryck RETURN)
40 END	(tryck RETURN)

RETURN-tangenten talar om, att programsatsen är avslutad. Om satsen innehåller fel, skrivs ett felmeddelande på skärmen.

6.1.3 Att rätta fel

Tangenten ← fungerar som radertangenten på en skrivmaskin. Då den trycks ned, raderas närmast föregående tecken på skärmen.

Att skriva så här: 20 LR←ET S=10
är likvärdigt med: 20 LET S=10

På samma sätt är: 30 LET←←←PRINT S
likvärdigt med: 30 PRINT S

En avslutad rad, som ger felmeddelande, kan ändras med piltangenterna (→ och ←).

Exempel

```
10 LE S=10
```

Detta ger ett felmeddelande. Stega fram med → så att 10 LE framträder. Tryck därefter T och fortsätt med → så att hela raden framträder.

För att ändra en färdigskrivna rad använder Du kommandot **ED**.

6.1.4 Att ta bort en rad.

För att ta bort den sats, Du håller på att skriva, trycker Du antingen CTRL och X samtidigt, eller tangenten CE. Då raderas hela den programrad, Du just skrivit.

OBS!

Om Du vill ta bort en redan inskriven programrad, skriv radnumret följt av RETURN, varvid raden med det angivna numret raderas.

Exempel

```
5 LET S=0
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
```

För att ta bort rad 5 skriver Du:

```
5
```

(tryck RETURN)

Kontrollera med **LIST**-kommandot.

6.1.5 Att ändra en programrad

Ett sätt att ändra en redan skriven programrad, är att skriva om den i dess nya form. Då Du avslutar raden med RETURN, ersätter den nya raden den gamla med samma nummer.

För att ändra rad 5 ovan kan Du t.ex. skriva:

```
5 LET S=5
```

(tryck RETURN)

Den gamla rad 5 ersätts nu med den nya.

Om Du bara vill göra små ändringar i programraden, kan Du använda kommandot **ED**.

6.2 Att ändra i ett program

Rader kan tas bort, läggas till eller ändras enligt 6.1. Kommandot **MERGE** gör det möjligt att länka program i minnet med en uppsättning programrader, som lagrats på ett yttre minne. Med kommandot **ERASE** raderar Du flera programrader åt gången. Kommandot **ED** gör det lätt att ändra några tecken i en redan inskriven rad.

När Du redigerar (eller editerar) Dina program, kan radnumreringen behöva ändras. Använd då kommandot **RENUMBER**.

6.3 Att köra ett program

Kommandot **RUN** startar körningen (exekveringen) av programmet. När Du skrivit **RUN** och avslutar med **RETURN**, börjar datorns BASIC utföra satserna på programraderna i minnet för användarprogram. Den rad, som har det lägsta numret, genomlöps först. Exekveringen fortsätter tills BASIC finner någon av följande tre orsaker att stanna:

STOP
END
feluthopp

När programmet genomlöper någon av satserna **STOP** eller **END**, stannar det. Alla variabler behåller sina värden. Du kan undersöka variablerna genom att adressera dem med variabelnamnen.

Ex: Du vill veta värdet på A,S och K%. Skriv då så här

```
PRINT A,S,K%                (RETURN)
```

Datorn skriver då ut de värden, variablerna hade, då exekveringen avbröts.

Fel presenteras med ett felmeddelande på skärmen.

Ett program, som körs, kan stoppas genom att Du trycker

CTRL/C (båda tangenterna på en gång)

Datorn kan då utföra en instruktion i taget om Du trycker

CTRL/S (båda tangenterna på en gång)

För fortsatt exekvering kan valfri tangent tryckas ner.

För att avbryta programexekveringen måste Du trycka

CTRL/C ytterligare en gång.

6.4 Vilka satser används var?

Syfte:

Att skriva meddelanden och förklaringar i programtexten

Använd:

REM-satser eller !

Syfte:

Att tilldela en variabel ett värde

Använd:

instruktionen **LET**

Syfte:

Att tilldela variabler värden

Använd:

instruktionerna **READ, DATA, ON – RESTORE** och **RESTORE**

Syfte:

Dataöverföring till och från systemet

Använd:

instruktionerna **INPUT, INPUT LINE, PRINT, GET, PUT, PREPARE, OPEN, CLOSE**

instruktionen **OUT** och funktionen **INP** för att styra in- och utmatning via I/O-portarna

Syfte:

Att styra programexekveringen

1. Ovillkorligt hopp

Använd:

instruktionen **GOTO**

2. Villkorligt hopp

Använd:

instruktionerna **IF – THEN, ON ...–** och **WHILE – WEND**

3. Programloopar (slingor)

Använd:

instruktionerna **FOR – NEXT**

4. Modulär programmering med underprogram

Använd:

instruktionerna **GOSUB – RETURN** och **DEFFN – FNEND**

Syfte:

Felhantering

Använd:

instruktionerna **ON ERROR GOTO – RESUME**

funktionen **ERRCODE**

Syfte:

Kombinera BASIC-program med program i ASSEMBLER-språk

Använd:

funktionen **PEEK**

instruktionen **POKE**

instruktionen **CALL**

funktionen **VARPTR**

Övriga instruktioner:

COMMON och **DIM** för att reservera utrymme för variabler

STOP, TRACE och **NOTRACE** förenklar felsökning av program

WIDTH för att välja antal tecken per rad (40 eller 80).

I kapitel 10 finns en lista över alla tillgängliga matematiska, logiska och andra funktioner, som utökar användarens möjligheter att skriva avancerade program.

6.5 Deklarationer

Följande deklarasatser förekommer:

- **FLOAT/INTEGER**
- **SINGLE/DOUBLE**
- **NO EXTEND/EXTEND**
- **OPTION BASE (0/1)**

Om deklarasatser används, måste de placeras först i programmet. Därefter ska eventuella **COMMON**- och **DIM**-satser följa.

7 Direktanvändning

Med BASIC II är det lätt att använda datorn för att lösa sådana problem, vanligtvis rent matematiska, som inte kräver ett datorprogramms förmåga att upprepa samma förfarande.

Enklare uttryckt: BASIC II gör det möjligt att använda datorn som en avancerad, elektronisk räknare, genom att BASIC-satserna kan exekveras direkt.

När BASIC väntar på kommandon kan en BASIC-sats skrivas utan radnummer. Då en sådan sats avslutas med RETURN, utförs den omedelbart. Detta kallas direktanvändning eller körning i direktmod.

De flesta BASIC-satser kan användas i direktmod.

Exempel

```
A=1.5 : B=3  
PRINT "(A+B*A)=";(A+B*A)
```

En sats, som skrivs efter ett radnummer, antas vara en programsats, som ska utföras vid ett senare tillfälle.

Körning i direktmod är mycket användbart vid programutveckling och felsökning. Man kan lätt ändra eller läsa av variablernas värden, och programmets förlopp kan på detta sätt övervakas och styras.

Direktanvändning i samband med program kan användas

- när man tryckt CTRL/C två gånger
- vid felmeddelande
- efter **STOP** eller **END**

8 Kommandon

Då ett kommando är skrivet och avslutat med RETURN, utför datorn den beordrade aktiviteten omedelbart. Ett BASIC-program med radnummer lagras däremot först i minnet och börjar utföras, då man ger kommandot **RUN**.

På skärmen visas texten ABC 806 då BASIC väntar på nästa kommando. Kommandon ska skrivas utan radnummer.

Olika kommandon styr programmeditering, körning av program och viss filhantering. Varje kommando identifieras av ett nyckelord i början av raden.

I det följande anges:

- reserverade ord – med fet stil, t.ex. **LOAD**, **SAVE** och **RUN**
- uppgifter som kan utelämnas – inom hakparentes, t.ex. [enhet:]
- alternativa uppgifter – med snedstreck, t.ex. "data"/strängvariabel
- ytterligare uppgifter – med punkter, t.ex. ["data"/strängvariabel,.....]

Allmänt gäller att:

- enheten adresseras med DR0:, DR1:, PR:, V24: eller CON:.
- om enhet utelämnas, adresseras alltid drivenhet 0 (DR0:) först och därefter drivenhet 1 (DR1:).
- filnamn får bestå av maximalt åtta bokstäver/siffror, varav det första tecknet måste vara en bokstav. Dessutom kan filtyp (3 tecken) användas valfritt för förtydligande av filnamnet.
- filtyp inte behöver anges. Undantag finns dock. Detta anges i så fall vid respektive syntax. Om filtyp utelämnas, utförs kommandot först på filtyp BAC och därefter på BAS.
- RETURN-tangenten måste tryckas ned efter varje avslutad inmatning.

Nedan följer en lista med kort beskrivning av varje kommando:

AUTO	Automatisk radnumrering
BYE	Övergång till skivoperativsystemet (DOS)
↩BAS	Återgång till BASIC
CLEAR	Nollställer alla variabler och stänger alla filer
CON	Fortsätter exekveringen av ett stoppat program
ED	Möjliggör editering (korrigerig)
ERASE	Raderar flera programrader

LIST	Listar programmet
LOAD	Laddar in ett program
MERGE	Länkar programfiler
NEW	Raderar programminnet
RENUMBER REN	Ändrar radnumreringen
RUN	(Laddar in och) Exekverar ett program
SAVE	Lagrar ett program
SCR	Raderar programminnet
UNSAVE	Raderar en fil

Nedan följer en utförlig beskrivning av samtliga kommandon.

AUTO

Format	AUTO [argument 1 [, argument 2]]
	Där <argument 1> anger första radnummer, som ska skrivas och <argument 2> anger radintervall.
	Argumenten är ej nödvändiga. Om inga argument anges, börjar radnumreringen med första jämna tiotal efter de redan skrivna raderna. Intervall sätts till 10.
Funktion	Skriver automatiskt ett nytt radnummer efter varje vagnretur. Man slipper alltså skriva radnummer vid programmering.
Användning	Kommandot kan användas när som helst under pågående programmering. För att avsluta den automatiska radnumreringen trycker man RETURN som första tecken på en ny rad. Om inmatad rad ger felmeddelande, avbryts radnumreringen och raden kan rättas. Numreringen kan återupptas med AUTO .
Exempel	AUTO 10,5 Första raden får nummer 10 och radnumren stegas upp med 5 för varje rad.

```
AUTO 10,5
10 LET A=1
15 --
20 --
25 --
```

O.S.V.

BYE

Format	BYE
Funktion	Övergång till skivoperativsystemet (DOS).
Verkan	Stänger de filer, som är öppna, och laddar kommandointerpretatorn CMDINT.SYS.

⌘BAS

Format	⌘BAS
Funktion	Övergång till BASIC.
Verkan	Avbryter arbetet under DOS och återgår till BASIC.

CLEAR

Format	CLEAR
Funktion	Nollställer alla variabler och stänger alla öppna filer.
Verkan	Påverkar ej programmet, som finns i minnet.

CON

Format	CON (eller CONTINUE)
Funktion	Fortsätter programexekveringen, där den har stoppats med antingen två CTRL/C eller en STOP -sats.
Användning	Direktkommandon kan användas för att ändra eller skriva ut variabelvärden, innan programmet startas igen med CON .
Observera	CON kan inte användas om programmet ändrats eller om ett feluthopp skett.

ED

Format	ED [radnummer]
	Där <radnummer> är numret på den rad, som ska ändras.
Funktion	Gör det möjligt att editera (ändra) i en programrad.
Verkan	När kommandot avslutats med RETURN, använder man tangenten högerpil → för att stega sig framåt i raden. Varje gång → trycks ned, visas ytterligare ett tecken. För att ändra innehållet, skriver man det nya innehållet på sin plats. Tangenten vänsterpil ← används för att radera bort felaktiga tecken.

Exempel

Rad 100 har följande innehåll:

```
100 A=B+C+E
```

Vi antar nu, att C skall ersättas av D och att man inte vill skriva om hela raden. Gör då på följande sätt:

Skriv **ED** 100. Tryck RETURN. På skärmen visas då rad 100:

```
100 A=B+C+E
```

Stega fram med → tills följande text syns:

```
100 A=B+C+E  
100 A=B+C
```

På raden under den ursprungliga framträder alltså Din nya text. Radera bort C med en tryckning på ←. Skriv D. Den understa raden ser nu ut så här:

```
100 A=B+D
```

Tryck på →, så att resten av raden blir synlig.

```
100 A=B+C+E  
100 A=B+D+E
```

Läs igenom hela raden och kontrollera. Tryck RETURN för att lagra den nya raden i stället för den ursprungliga.

Observera

Då felmeddelande erhålls i samband med programmering, finns den felaktiga raden kvar i datorn och kan editeras med → och ← enligt ovan, utan föregående **ED**-kommando.

Om **ED** skrivs utan något radnummer, hämtas programmets första rad.

ERASE

Format

ERASE radnummer [-radnummer]
ERASE radnummer -
ERASE - radnummer

Funktion

Raderar en del av det program, som finns i minnet.

Verkan

Alla rader fr.o m. radnummer t.o.m radnummer utplånas.

Exempel

```
ERASE 20-200
```

Raderar raderna 20–200

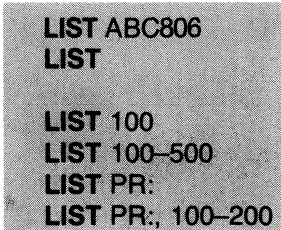
```
ERASE -200
```

Raderar raderna 1–200

```
ERASE 20-
```

Raderar raderna 20–programmets slut

LIST

Format	LIST [enhet:] filnamn[.typ][,radnummer[-radnummer]] LIST [enhet:][,radnummer[-radnummer]] LIST radnummer - LIST - radnummer												
	Där <filnamn.typ> är namnet på en programfil. <Enhet> kan vara t.ex. PR:, DR0: eller DR1:.												
Funktion	Listar hela eller del av program.												
Verkan	<ol style="list-style-type: none">LIST [enhet:] filnamn [typ] Lagrar programmet, som är i arbetsminnet, på yttre minne i textformat. Programmet lagras under namnet <filnamn>. Jämför SAVE. Filtypen blir BAS, om inget annat anges.LIST Hela programmet listas på skärmen.LIST radnummer Endast den angivna raden listas på skärmen.LIST radnummer I - radnummer II Alla rader fr.o.m. radnummer I t.o.m. radnummer II listas.LIST PR: Hela programmet listas på skrivare.LIST - radnummer Programmet listas t.o.m. angivet radnummer.LIST radnummer - Programmet listas fr.o.m. angivet radnummer.												
Observera	Ett långt program listas på skärmen, tills den är fylld. Nästa rad framträder, då man trycker på mellanslagstangenten. Listningen stoppas med CTRL/C, RETURN eller valfritt BASIC-kommando.												
Exempel	 <table><tr><td>LIST ABC806</td><td>Lagrar filen ABC806 på externt minne</td></tr><tr><td>LIST</td><td>Listar hela programmet på bildskärmen</td></tr><tr><td>LIST 100</td><td>Listar rad 100</td></tr><tr><td>LIST 100–500</td><td>Listar raderna 100–500</td></tr><tr><td>LIST PR:</td><td>Listar hela programmet på skrivare</td></tr><tr><td>LIST PR:, 100–200</td><td>Listar raderna 100–200 på skrivare</td></tr></table>	LIST ABC806	Lagrar filen ABC806 på externt minne	LIST	Listar hela programmet på bildskärmen	LIST 100	Listar rad 100	LIST 100–500	Listar raderna 100–500	LIST PR:	Listar hela programmet på skrivare	LIST PR:, 100–200	Listar raderna 100–200 på skrivare
LIST ABC806	Lagrar filen ABC806 på externt minne												
LIST	Listar hela programmet på bildskärmen												
LIST 100	Listar rad 100												
LIST 100–500	Listar raderna 100–500												
LIST PR:	Listar hela programmet på skrivare												
LIST PR:, 100–200	Listar raderna 100–200 på skrivare												

LOAD

Format	LOAD [enhet:] filnamn[.typ] Där <filnamn.typ> är namnet på det program, som ska laddas in. Enheten kan t.ex. vara DR0: eller DR1:.
Funktion	Laddar in ett program från yttre minne.
Exempel	LOAD ABC LOAD DR1: PROG.BAS
Observera	Om filtyp (de tre tecknen efter punkten) inte anges, söker datorn först efter filtyp .BAC och sedan filtyp .BAS. Datorn läser hela filen fram till EOF (end of file) och inte bara till END .

MERGE

Format	MERGE [enhet:] filnamn[.typ] Där <filnamn> är namnet på en fil på en yttre minnesenhet. Filen ska vara lagrad i textform (med LIST).
Funktion	Länkar programfiler så att programraderna kommer i nummerordning. Programmet i arbetsminnet överlagras med den begärda programfilen från en yttre enhet.
Verkan	De numrerade BASIC-raderna från den begärda filen läggs in i programmet i datorns arbetsminne. Varje rad felsöks, då den läggs in. Radnummer, som inte finns i programmet förut, läggs på sin plats i nummerordning. Om en ny rad har samma nummer som en rad i programmet, läggs den nya raden på den gamlas plats. Den yttre filen läses till EOF(end of file).
Exempel	I datorn finns följande program:

```
5 Y=10
10 PRINT
20 FOR L=1 TO 10
30 PRINT L;TAB(Y);"I";
40 READ Y
50 FOR I=1 TO Y
60 PRINT "×";
70 NEXT I
80 PRINT
90 PRINT TAB(Y);"I"
100 NEXT L
```

På yttre minnesenhet (flexskiva) finns följande programfil under namnet TABELL:

```
200 DATA 5,4,2,3,1
300 DATA 10,15,28,15,6
999 END
```

För att länka dessa båda filer ges kommandot:

```
MERGE TABELL
```

Raderna 200, 300 och 999 läses då in till programmet i arbetsminnet.

NEW

Format	NEW
Funktion	Raderar innehållet i arbetsminnet.
Verkan	Raderar arbetsminnet, nollställer alla variabler och återställer pekare. Kommandot rensar på detta sätt minnet från alla spår av programmet och börjar på nytt. Alla öppna filer stängs. Använd detta kommando innan Du börjar skriva in ett nytt program.
Observera	Kommandot SCR (scratch) kan också användas. SCR har samma funktion som NEW .

RENUMBER REN

Format	REN[UMBER] [radn[,intervall[,fr.o.m. rad -t.o.m. rad]]]
	Där <radnr> är det nummer, som första raden ska få. Om inget radnummer anges, får första raden nummer 10. <Intervall> anger intervallet mellan två rader. Om inget intervall anges, ökas radnumret med 10 för varje ny rad. <Fr.o.m. rad> – <t.o.m. rad> anger vilka rader, som ska numreras om. Om inget anges, numreras alla rader i programmet om. För att <Intervall> ska kunna anges, måste <radnr> anges. Både <radnr> och <intervall> måste anges, om man ska kunna ange <fr.o.m rad>–<t.o.m. rad>.
Funktion	Ändrar radnumreringen i det aktuella programmet.

Verkan Alla radnummer i programmet ändras på det sätt, som begärts i kommandot **RENUMBER**.

Alla radnummer, som används i satser med **GOSUB**, **GOTO**, **IF**, **ON** eller **RESUME** ändras, om det behövs för att hoppen ska ske till samma programsats som tidigare.

Om någon rad i programmet ger ett hopp till en icke existerande rad, skrivs ett felmeddelande ut på skärmen. I detta fall behåller raderna sin ursprungliga numrering.

Exempel

```
REN  
REN 10  
RENUMBER 10,5  
REN 10,5,10-50  
REN 10,5,-100  
REN 10,5,100-
```

RUN

Format **RUN** [enhet:] [filnamn[.typ]]

Där <filnamn> är namnet på den programfil, som ska laddas in och köras. Om ingen filtyp anges, söker datorn först efter .BAC och i andra hand efter .BAS.

Funktion Laddar och exekverar ett BASIC-program eller exekverar (kör) programmet i datorns minne.

Verkan

1. RUN

Alla variabler och fält i programminnet nollställs. Datasatser återställs och programmet i minnet genomlöps med början på den rad, som har lägst nummer.

2. RUN filnamn[.typ]

Programmet <filnamn[.typ]> laddas in på samma sätt som med kommandot **LOAD**. Det nyss inlästa programmet genomlöps med början på den rad, som har lägst nummer.

Exempel

Ex.1

```
10 READ A,B  
20 LET A=A+B  
30 PRINT A  
40 DATA 2,3  
50 END  
RUN  
5
```

Ex.2

Om samma program hade varit lagrat på en yttre minnesenhet under namnet APLUSB, ser bildskärmen ut så här:

```
RUN APLUSB  
5
```

SAVE

Format	SAVE [enhet:] filnamn[.typ]
	Där <filnamn> är namnet på den nya filen i form av en teckensträng. Om ingen filtyp anges, lagras filen som typ .BAC.
Funktion	Skapar en programfil på ett yttre minne och lagrar programmet, som finns i datorns arbetsminne, i denna fil.
Verkan	Kommandot medför, att programmet i arbetsminnet lagras under det angivna filnamnet. Programmet lagras i internkodsformat, så att det kan laddas snabbt.
Exempel	<pre>10 - - 900 - - 999 END SAVE TEST</pre>
Observera	Om filen redan finns på flexskiva, kommer den gamla filen att förstöras och ersättas av det nya programmet, om inte filen på skivan är skrivskyddad.

SCR

Format	SCR
Funktion	Se NEW .

UNSAVE

Format	UNSAVE [enhet:] filnamn[.typ]
Funktion	Raderar en fil på flexskiva.
Exempel	När man har slutfört allt arbete med filen XYZ, raderas filen med följande kommando: <pre>UNSAVE XYZ</pre>
Observera	Om man inte anger filtyp, söker datorn först efter .BAC och i andra hand .BAS. Kommandot UNSAVE kan inte användas på en raderskyddad fil.

9 Instruktioner

Detta kapitel behandlar programsatserna i BASIC II. En programsats är en instruktion, som beordrar BASIC att utföra en viss operation. De flesta instruktioner kan även användas som kommandon.

Nedan följer en instruktionslista med kort beskrivning av varje instruktion.

CHAIN	Laddar en programfil och exekverar den
CLOSE	Stänger filer
COMMON	Överför variabler till nästa program vid CHAIN
DATA	Datasats, värdet hämtas med READ
DEF FN	Definierar en användarfunktion
DIGITS	Anger max. antal siffror, som skrivs ut
DIM	Anger storlek för vektor/matris och sträng
DOUBLE	Dubbel precision (16 siffror)
END	Logiskt sista instruktion i BASIC-program
EXTEND	Möjliggör användning av långa variabelnamn
FLOAT	Återställer till flyttalsformat
FNEND	Avslutar flerradiga användarfunktioner
FOR	Inleder programloop (NEXT avslutar)
GET	Läser ett tecken.
GET COUNT	Läser angivet antal tecken
GOSUB	Anropar en subrutin
GOTO	Hoppar till angivet radnummer
IF-THEN-ELSE	Styr villkorlig exekvering
INPUT INPUT LINE	Läser in data
INTEGER	Ändrar till heltalsformat
KILL	Tar bort en fil
LET	Tilldelar en variabel ett värde
NAME	Ger en fil nytt namn

NEXT	Stegar räknevariabeln i programloop
NO EXTEND	Stänger av EXTEND -mod
NO TRACE	Stänger av TRACE -funktion
ON ERROR GOTO	Hantering av feluthopp
ON GOSUB	Villkorssats för hopp till subrutin
ON GOTO	Villkorssats för hopp till olika radnummer
ON RESTORE	Villkorlig återställning av datapekare
ON RESUME	Villkorligt hopp från felhantering
OPEN-AS FILE	Öppnar en fil och ger den ett filnummer
OPTION BASE	Sätter lägsta värde för vektorindex
POSIT	Positionerar filpekare
PREPARE-AS FILE	Skapar en ny fil och ger den ett filnummer
PRINT PRINT USING	Skriver ut data med angivet format
PUT	Skriver en post
RANDOMIZE	Ger slumpalsgeneratorn ett nytt startvärde
READ	Tilldelar en variabel ett värde från DATA -sats
REM (!)	Inleder kommentar
RESTORE	Ställer datapekare
RESUME	Återhopp från felhantering
RETURN	Återhopp från subrutin
SINGLE	Ändrar precisionen till sju siffror
STOP	Stoppar programexekveringen
TRACE	Startar TRACE -funktion (felsökning)
WEND	Avslutar WHILE -funktion
WHILE	Ger uthoppsvillkor för WHILE -funktion
WIDTH	Väljer antal tecken per rad

Nedan följer en utförlig beskrivning av samtliga instruktioner.

CHAIN

Format	CHAIN "filnamn.typ"/strängvariabel
	Där <filnamn.typ> är namnet på det program, som ska laddas in. Om filnamnet avser en fil på flexskiva, och filtypen inte är angiven, söker datorn först efter .BAC och i andra hand .BAS.
Funktion	Det program, som anropas, laddas in och körs.
Användning	Om användarprogrammet är för stort för att laddas in i arbetsminnet, kan man dela upp programmet i flera delprogram. CHAIN -instruktionen används som logisk avslutning på ett program för att hämta in nästa. Varje program anropas med sitt namn. Det förra programmet raderas, och det nya laddas in. Den programrad, som har lägst nummer, utförs först, på samma sätt som vid kommandot RUN . CHAIN -instruktionen blir den sista instruktion, som utförs. Det sista programmet i en sådan kedja behöver alltså ingen CHAIN -sats, men det är vanligt, att man återgår med CHAIN till ett program, där användaren kan välja, vilket delprogram, som ska köras.
Observera	För att överföra variabler från ett program till ett annat, används instruktionen COMMON .
Exempel	<pre>550 CHAIN "PROGRAM4.BAC"</pre>

CLOSE

Format	CLOSE [filnummer, ...]
	<Filnummer> anger den fil, som ska stängas.
Funktion	Avslutar in/utmatning och stänger filen.
Användning	Instruktionen CLOSE används för att stänga en eller flera filer. Om inte filnummer anges, stängs samtliga filer.
Observera	Instruktionen END stänger alla öppna filer. Vid utmatning med PRINT matas innehållet i den sista buffertarean ut, när filen stängs.

COMMON

Format	COMMON variabel[(n,...)] [,variabel,...] COMMON strängvariabel [(n,...)] = längd[,strängvariabel = längd,...]
	<n>=vektorindex och <längd>=strängens maximala längd
Funktion	Överför variablers värden från ett program till ett annat vid CHAIN .

Användning	Deklarationerna måste se likadana ut i alla de berörda programmen. Programmen måste vara lika beträffande precision och hel- eller flyttal.
Observera	COMMON -strängvariablers längd måste deklarerars. COMMON -satserna måste placeras omedelbart efter deklaratonsatserna. Innan programmet sparas skall programmet startas och stoppas för upplägg av common-variabler.
Exempel	<code>10 COMMON A%,B(7),C\$(15)=25</code>

DATA

Format	DATA värde [,värde, ...] Där alla DATA -satser, var de än står i programmet, skapar en datalista. DATA -satsen måste stå ensam på en programrad, dvs raden får ej innehålla tex REM då detta tolkas som data.
Funktion	Utgör tillsammans med READ -instruktionen en metod att tilldela variabler värden.
Användning	DATA används enbart tillsammans med READ och omvänt. Se vidare READ .
Exempel	<pre>100 DATA ...en.text..., ".en.text." 110 DATA ' "...en.text..." ' 120 FOR I=1 TO 3 130 READ A\$ 140 PRINT "!";A\$;"!" 150 NEXT I</pre>

ger följande utskrift:

```
!...en.text...!
!.en.text.!
!"...en.text.."!
```

DEF FN

Format	Enradig funktion: DEF FN identifierare [(argument)]=funktion Flerradig funktion: DEF FN identifierare [%/\$(argument)] [LOCAL variabel [,variabel,...]]
Funktion	Definierar enradiga och flerradiga funktioner.
Användning	En flerradig DEF -funktion skiljer sig från enradiga användarfunktioner, genom att funktionsnamnet på första raden inte följs av något likhetstecken. Vilka typer av argument som helst kan användas. Det är också tillåtet att blanda olika typer av argument. Den flerradiga funktionsdefinitionen ska innehålla satser med följande utseende: RETURN uttryck FNEND När RETURN genomlöps, returneras funktionsvärdet varefter återhopp sker. Det kan finnas mer än en sådan sats, som framgår av exempel nedan.

Observera

ON ERROR GOTO radnummer, **GOSUB** radnummer, **GOTO** radnummer och **RESUME** radnummer får endast anropa rader som finns inom funktionen. Endast **RESTORE** till datasatser får peka på rader utanför funktionen.

Exempel på enradig funktion:

```
10 DEF FNA(X,Y)=X+X * Y
```

I nedanstående exempel väljs det större av två tal. Detta tal returneras som funktionsvärde. Det är vanligt, att man på detta sätt använder **IF – THEN** instruktionen i flerradiga funktioner:

```
10 DEF FNM(X,Y)
20 IF Y <=X THEN RETURN X
30 RETURN Y
40 FNMEND
```

Nästa exempel visar en rekursiv funktion, som beräknar N! (N-fakultet). (Det finns effektivare, icke-rekursiva metoder att beräkna N!)

```
5 EXTEND
10 DEF FNFak(M%)
20 IF M%=1% THEN RETURN 1% ELSE RETURN
M%*FNFak(M%-1%)
30 FNMEND
35 INPUT "Beräkna fakultet för följande tal: ";X
40 PRINT X;"-fakultet är ";FNFak(X)
50 END
```

RUN

Beräkna fakultet för följande tal: 4
4-fakultet är 24

En variabel, som används inne i själva funktionen, och inte är ett argument eller en lokal variabel för just den funktionen, behåller sitt värde i det överordnade programmet. En flerradig användarfunktion kan anropa en annan, men denna andra måste då vara helt innesluten i den första. Här gäller samma regler för programmets uppbyggnad som vid programslingor. De radnummer, som används i definitionen, får inte anropas från resten av programmet. Om man använder **ON ERROR GOTO** i definitionen, stängs denna felhantering av vid återhopp till det program, som anropat funktionen.

Om man behöver tillfälliga variabler i en definierad funktion, bör dessa variabler vara lokala, så att man inte råkar skada de globala variablerna. Man behöver då inte leta fram "oanvända" variabler.

Modifieraren **LOCAL** gör, att man kan använda ett lokalt variabelnamn. Vektorer kan inte deklaras som lokala variabler. Strängvariabler måste anges med sin längd.

```

10 DEF FNA(X) LOCAL A,A=10
20 A=33: A="Lokal"
30 PRINT A
40 PRINT A
50 RETURN 5*X
60 FNEND
100 A=22: A="Global"
110 PRINT A
120 PRINT A
130 PRINT FNA(8)

```

```

RUN
Global
22
Lokal
33
40

```

Nedanstående exempel visar en strängfunktion:

```

100 PRINT FNV1("AaBbCcDdEeFf",5%,10%)
110 END
120 DEF FNV1(A,B%,C%)
130 IF B%=C% THEN RETURN LEFT(A,B%) ELSE
    RETURN RIGHT(A,C%-B%)
140 FNEND
RUN
CcDdEeFf

```

FNEND

Format	FNEND
Funktion	Avslutar en flerradig funktion.
Användning	Se instruktionen DEF FN .
Observera	Denna sats får aldrig genomlöpas. Innan FNEND ska funktionen ha gett ett uthopp med RETURN .

DIGITS

Format	DIGITS antal siffror
Funktion	Ger antal siffror, som skrivs ut med PRINT
Verkan	Det tal, som skrivs ut med PRINT , avrundas till angivet antal siffror. Talvärden, som är för stora för att skrivas med detta antal siffror, skrivs på exponentiell form med angivet antal siffror.
Observera	Instruktionen DIGITS påverkar inte beräkningarnas noggrannhet. Default: 6/16 siffror beroende på precision.

DIM

Format	DIM variabel(n)[,variabel(n,...),...] DIM strängvariabel[(n,...)] [=uttryck]
	Där värdet efter "=" anger stränglängden. n,... är högsta indexvärde. Begynnelseindex är <undre gräns> om inget annat angivits. <undre gräns> är antingen 0 eller 1 beroende på OPTION BASE . Om ingen sådan instruktion har genomlöpts, har <undre gräns> värdet 0.
Funktion	Ger maximala antalet element i indexerade variabler och strängar.
Användning	Man kan ha hur många index som helst. Alla värden, som anges i en DIM -sats, avrundas till heltal. Om man använder en indexerad variabel, utan att dimensionera den, antas dimensionen till 10 för varje index. Alla variabler har värdet 0 tills de tillordnats ett annat värde. Om en strängvariabel inte dimensionerats, sätts dess maxlängd automatiskt till den längd strängen har, då den för första gången tillordnats ett värde skilt från tom sträng. För en sträng, som inte dimensionerats, reserveras dock aldrig kortare längd än 80 tecken.
Exempel	DIM Aα(N) En strängvektor med strängarna A α (<undre gräns>) – A α (N). Varje sträng har längden 80. DIM Aα(N)=14 Som ovan men varje sträng har längden 14 tecken. 10 DIM X(5),Z(4,3),A(10,10) 12 DIM A4(100) 14 DIM A α (20),B α (10,20) 16 DIM C α (40)=4 18 DIM D α (10,10)=8 20 DIM Q α =253

AVANCERAD PROGRAMMERING:

Den undre gränsen 0 eller 1 kan vid behov ändras för varje index. Man ersätter maxvärdet med två värden åtskilda av kolon.

Exempel	DIM A(-2:2) Här får vi en vektor med fem element A(-2), A(-1), A(1), A(2) oberoende av den undre gräns, som gäller för tillfället. Omdimensionering av tidigare dimensionerad variabel är tillåten, om den nya DIM-satsen medför en mindre dimension.
---------	---

DOUBLE

Format	DOUBLE
Funktion	Ändrar alla variabler och uttryck med flyttal till dubbel precision (16 siffror).
Användning	Deklarationen DOUBLE ska stå innan variablerna används i programmet och kan inte ändras när programmet väl har startats med RUN . Ändringen kan göras om en programrad har editerats eller om man har använt kommandot CLEAR . Om precisionen inte deklarerats i programmet, gäller SINGLE .
Observera	Det går inte att blanda SINGLE och DOUBLE i ett och samma program.

END

Format	END
Funktion	Avslutar programmet.
Användning	Instruktionen END ska ha det högsta förekommande radnumret i huvudprogrammet. Efter END får bara förekomma underprogram (subrutiner) och funktioner, vilka ger återhopp till huvudprogrammet. END stänger alla filer.
Observera	Variablerna behåller sina värden efter END . END måste stå först på raden.

EXTEND

Format	EXTEND
Funktion	Tillåter långa variabelnamn.
Observera	Då EXTEND begärts, är mellanslag signifikanta, utom efter radnummer och intill aritmetiska operatorer ($- + * /$). Hopskrivna nyckelord kan misstolkas som långa variabelnamn. Variabelnamnen får vara hur långa som helst och är signifikanta till alla delar.

Exempel

```
10 EXTEND
20 LET MOMS=MOMSGRUNDANDE*MOMSROCENT
```

FLOAT

Format	FLOAT
Funktion	Samtliga tal tolkas som flyttal. Heltal betecknas med %.
Användning	Se instruktionen INTEGER
Observera	Som defaultvärde gäller FLOAT . Det är inte tillåtet att blanda FLOAT och INTEGER i samma program.

Exempel

```
10 A=125.5 !Flyttal
20 A%=12% !Heltal
```

FOR

Format

FOR variabel=uttryck **TO** uttryck [**STEP** intervall]

Där variabeln i **FOR...TO**-satsen får det begynnelsevärde, som ges av det första uttrycket. Därefter utförs instruktionerna mellan **FOR** och **NEXT**. När **NEXT** påträffas, adderas det angivna intervallet till variabeln. Se även under **NEXT**.

Om variabeln har ett värde, som är större än uttrycket efter **TO**, sker uthopp till satsen efter **NEXT**.

Uttrycken i **FOR**-satsen beräknas en enda gång vid inhoppet i loopen. Variabeln kontrolleras mot den övre gränsen varje gång loopen ska genomlöpas. Beträffande uttryck se avsnitt 2.4.

Funktion

Instruktionerna **FOR** och **NEXT** används tillsammans för att skapa programloopar. En loop innebär, att en eller flera instruktioner utförs ett visst antal gånger.

Användning

En programloop består av fyra delar:

1. Initiering av de villkor, som måste gälla, för att loopen ska genomlöpas en första gång.
2. Programloopens innehåll, d.v.s. de instruktioner, som ska upprepas.
3. Modifieringen, som ändrar variabelns värde, så att varje genomlöpning av loopen skiljer sig från de övriga.
4. Uthoppsvillkoret, som kontrolleras vid varje genomlöpning. När det är uppfyllt, sker uthopp till programsatsen närmast efter loopen.

Om **STEP**-uttrycket inte anges i **FOR**-satsen, antas intervallet vara +1. Detta medför, att de flesta **FOR**-satser inte innehåller något **STEP**-uttryck.

Variabeln kan också modifieras inne i själva loopen. När uthopp sker, har variabeln sitt nya värde, d.v.s. senast använda variabelvärdet + intervall.

FOR-loopar kan inte överlappa varann utan måste vara antingen totalt inneslutna i varann (kapsling) eller totalt fristående från varann.

Även om variabeln inte har uppnått sitt gränsvärde, kan man hoppa ur programloopen med ett villkorligt eller ovillkorligt uthopp. Om man vill hoppa tillbaka in i en loop, som inte hade genomlöpts fullständigt, måste man vara noga med att sätta korrekta värden för övre gräns och steglängd.

Exempel

Ex.1 Exempel på en **FOR – NEXT**-loop. Loopen utförs 20 gånger. Före uthoppet ur loopen skrivs $A=20$. **FOR**-satsen innehåller inget **STEP**-uttryck, alltså antas intervallet vara $+1$.

```
10 FOR A%=1% TO 20%
20 PRINT "A=";A%
30 NEXT A%
40 PRINT "A=";A%
```

Loopen består av raderna 10, 20 och 30. Då loopen genomlöps, skrivs $A=1, A=2, \dots, A=20$. När $A\%$ har värdet 20 och rad 30 genomlöps, ökas $A\%$ med 1 och rad 10 genomlöps. Eftersom $A\%$ nu är större än övre gränsvärdet, ger rad 10 uthopp till rad 40. Utskriften från rad 40 blir $A=21$.

Ex.2 Godtagbar kapsling

```
50 FOR A=1 TO 10
60 FOR B=2 TO 11
70 NEXT B
80 FOR C=1 TO 10
90 NEXT C
100 NEXT A
```

Ej godtagbar kapsling

```
150 FOR A=1 TO 10
160 FOR B=2 TO 11
170 NEXT A
180 NEXT B
```

Observera

I **FOR**-satser bör man använda heltalsvariabler, eftersom loop-
en då genomlöps mycket snabbare.

NEXT

Format

NEXT variabel

Där <variabel> är den variabel, som specificeras i tillhörande **FOR**-sats. Satserna **FOR** och **NEXT** avgränsar programloopen. När **NEXT**-satsen genomlöps, adderas intervallet till variabelns värde och programmet undersöker, om variabeln överskridit den övre gräns, som anges i **FOR**-satsen. När variabelns värde är större än det övre gränsvärdet, utförs satsen närmast efter **NEXT**-satsen.

Funktion

NEXT anger slutet på en programloop, som inletts med en **FOR**-sats. Vid **NEXT** inkrementeras variabeln (ökas med intervallet).

Användning

Se **FOR**.

GET

Format

GET strängvariabel [**COUNT** antal tecken]

Funktion

Läser in ett eller angivet tecken från tangentbordet till en strängvariabel.

Observera

Om tangentbordsbufferten är tom, väntar BASIC-tolken tills en tangent trycks ned. Vilket tecken som helst kan hämtas in.

GET

Format	GET # filnummer, strängvariabel [COUNT antal tecken] <Filnummer> är det filnummer, som definierats med instruktionen OPEN . <Strängvariabel> är den sträng, dit tecknet(nen) överförs. COUNT <antal tecken> anger antalet tecken, som ska läsas från filen.
Funktion	Läser in ett eller flera tecken från angiven fil till angiven strängvariabel.
Användning	Se kapitel 3.5

GOSUB

Format	GOSUB radnummer <Radnummer> är det första radnumret i den anropade subrutinen. Programexekveringen fortsätter med detta radnummer.
Funktion	Ger uthopp till en subrutin.
Användning	En subrutin är en följd av programinstruktioner, som utför en uppgift, vilken återkommer vid flera tillfällen i ett program. Subrutiner och funktioner gör det möjligt, att anropa en sådan instruktionsföljd från flera ställen i programmet. En subrutin är en programdel, som kan anropas med en GOSUB -instruktion. När subrutinen har fullgjort sin uppgift, sker ett återhopp via en RETURN -instruktion till programsatsen närmast efter den anropande GOSUB -satsen.
Exempel	<pre>50 GOSUB 1300 1290 REM Detta är en subrutin 1300 LET K=1 1360 RETURN 9999 END</pre>
Observera	Uthopp ur en subrutin får endast ske med GOSUB eller RETURN .

GOTO

Format	GOTO radnummer Där <radnummer> vanligtvis inte är närmast följande rad i programmet.
Funktion	Ovillkorligt hopp till angivet radnummer i programmet.
Användning	Instruktionen GOTO används, när man vill åstadkomma ett ovillkorligt hopp till en annan programrad än den nästföljande. Med instruktionen GOTO kan man hoppa framåt eller bakåt i programmet. När GOTO används i en programrad med flera satser, ska denna instruktion alltid stå sist på raden. Om någon annan programsats står senare på raden, utförs den aldrig
Exempel	<pre>10 X=20 20 PRINT X 30 X=X+1 40 GOTO 20 50 END</pre>
Observera	GOTO kan användas i direktmod i stället för CON (Continue), om man vill att programmet ska exekveras från en viss rad.

IF-THEN-ELSE

Format	IF villkor THEN sats[er]/radnummer[ELSE sats[er]/radnummer] Där test sker med avseende på det angivna villkoret. Om det är uppfyllt (logiskt sant) utförs det, som står efter THEN . Om villkoret inte är uppfyllt (logiskt falskt), utförs den programrad, som följer närmast efter IF -satsen.
Funktion	Används för villkorlig styrning av ordningsföljden mellan programraderna.
Användning	Efter THEN i IF -satsen får finnas antingen ett radnummer eller en eller flera BASIC-satser. Om där finns BASIC-satser, och villkoret är uppfyllt, utförs dessa satser innan programmet fortsätter med raden närmast efter IF -satsen. Villkoret gäller för alla satser, som följer på samma rad som IF -satsen. Efter ELSE anges antingen ett radnummer, dit uthopp sker, eller en eller flera satser, som utförs innan raden närmast efter IF -satsen. Om villkoret är uppfyllt, utförs de instruktioner, som finns mellan THEN och ELSE . Vid beräkningen av villkorsuttryck utförs aritmetiska operationer i sin vanliga prioritetsordning. Relationsoperatorerna har samma inbördes prioritet och utförs efter de aritmetiska operatorerna men före de logiska operatorerna.

Relationsoperatorer

= lika med
<> icke lika med
< mindre än
> större än
<= mindre än eller lika med
>= större än eller lika med

Villkorsuttryck har värdet -1, om de är sanna, och värdet 0 om de är falska.

Ex. $5+6 \times 5 > 15 \times 2$ är sant

Exempel

```
170 IF A<B+3 THEN 160
180 IF A=B+3 THEN PRINT "A har värdet ";A
190 IF A=>B THEN T1=B
200 IF A=B THEN PRINT "Lika ": A=1/B
210 IF A>B THEN PRINT "Större" ELSE PRINT "Mindre"
```

I rad 200 gäller villkoret både för **PRINT**-satsen och tilldelningssatsen.

INPUT

Format

INPUT [#filnummer/"ledtext"] variabel [,variabel,...]

Där <filnummer> är den beteckning filen givits i instruktionen **OPEN**.

<Variabel> innehåller namn på aritmetiska variabler, element i numeriska vektorer, strängvariabler eller element i strängvektorer.

Om # <filnummer> inte anges, antar systemet, att indata kommer från tangentbordet. Data läses från den fil eller den enhet, som tilldelats det angivna filnumret.

<Ledtext> är en teckensträng inom citationstecken. Kan endast användas vid inmatning från tangentbordet.

Funktion

Hämtar in data till det program, som körs.

Användning

Medan programmet exekveras, kan användaren mata in data efterhand som programmet begär in dem. **INPUT** låter datorn vänta på ett svar. Om ingen ledtext angivits, skrivs ett frågetecken på skärmen.

Det är ofta lämpligt att låta programmet skriva en ledtext, för att påminna användaren om vilka data, som ska skrivas in. Se Ex. 1 nedan. Efter en ledtext skrivs inget frågetecken.

Exempel

Ex. 1

```
10 INPUT "Ditt namn : ?" A$
20 INPUT "Din adress : ?" B$
```

är samma sak som:

```
10 PRINT "Ditt namn : ";
20 INPUT A$
30 PRINT "Din adress : ";
40 INPUT B$
```

Ex. 2

```
50 INPUT #3,C$
```

Instruktionen medför, att data läses från fil 3. Data placeras i strängen C\$.

INPUTLINE

Format

INPUT LINE [# filnummer,] strängvariabel

<Filnummer> är det nummer, som ges i instruktionen **OPEN** och betecknar en yttre enhet eller en fil som logisk enhet.

Funktion

Tar emot inmatning av en rad tecken.

Användning

Programmet tar emot en rad tecken från den angivna filen. Alla tecken på raden läses in, även mellanslag, skiljetecken och citationstecken. De avslutande tecknen vagnretur (CR) och radframmatning (LF) läses också in.

Med instruktionen **INPUT LINE** kan man inte få någon ledtext, utan den får skrivas med en **PRINT**-sats.

Exempel

Ex. 1

```
10 PRINT "Din adress : ";
20 INPUT LINE A$
```

Ex. 2

```
10 INPUT LINE A$
20 A$=LEFT$(A$,LEN(A$)-2)
```

Exempel 2 tar bort CR och LF från strängen A\$.

INTEGER

Format

INTEGER

Funktion

Vid inmatning och listning av program antas alla variabler vara heltalsvariabler, om inget annat anges.

Användning När ett program skrivs in, och man givit instruktionen **INTEGER**, behöver programmeraren inte ange suffixet % på heltalsvariablerna. Däremot ska alla flyttalsvariabler markeras med suffixet . (decimalpunkt), och strängvariabler ska som vanligt ha suffixet \$.

Ett program, som är lagrat i textform och innehåller flyttalsvariabler, kan köras som ett rent heltalsprogram, om man ger kommandot **INTEGER** innan det laddas. När ett sådant program lagras, har man omformat det till heltalsprogram.

Observera Om inget annat anges, gäller flyttalsformat. Det är inte tillåtet att blanda **INTEGER** och **FLOAT** i samma program.

Exempel

```
100 REM Visar utskrift av flyttal/heltal
110 INTEGER
120 A.=12.345
130 B.=123
140 C=B.
150 D1=A.
160 PRINT A.,B.,C,D1
170 END
RUN
12.345 123 123 12
```

KILL

Format **KILL** "[enhet:] filnamn[.typ]"

Där filen med namnet <filnamn.typ> inte är raderskyddad. Användaren kan inte radera en fil, som är raderskyddad.

Funktion Den fil, som anges med filnamnet, raderas från det yttre minnet.

Exempel När användaren inte längre behöver filen XYZ.TXT på flexskivan, kan filen raderas från skivan med följande sats:

```
460 KILL "XYZ.TXT"
```

LET

Format [**LET**] variabel=uttryck

Ordet **LET** kan utelämnas.

Användning Instruktionen **LET** används för att tilldela en variabel ett värde.

Exempel

```
10 LET A=5.02
20 LET B9=5*(X/2)
30 D=(3*A)/2-B
```

NAME

Format	NAME "[enhet:]filnamn1.typ" AS "filnamn2.typ"
Funktion	Ändrar namn på en fil på flexskiva.
Användning	Den fil, som har namnet <filnamn1.typ> erhåller det nya namnet <filnamn2.typ>. Ingen filtyp antas, om den inte anges. Filtyp måste anges i båda fallen, om filen är lagrad med filtyp och filtyp önskas i det nya filnamnet.

Exempel Ex.1

```
100 NAME "DR0:GAMMAL.BAC" AS "NY.BAC"
```

Ex.2
Följande sats:

```
200 NAME "DR0:ABC.BAC" AS "XYZ.BAC"
```

ändrar namnet på filen ABC.BAC på skivan i DR0:
Instruktionen **NAME – AS** kan inte överföra en fil från en enhet till en annan.

Ex.3

```
120 NAME "NYTT" AS "NYTT1"
```

NO TRACE

Format	NO TRACE
Funktion	Avslutar den utskrift av radnummer, som begärts med instruktionen TRACE

Exempel

```
10 PRINT "Start"  
20 K=-1  
30 TRACE  
40 IF K> 1 THEN 80  
50 K=K+1  
60 PRINT "Nr ";K  
70 GOTO 40  
80 A=K  
90 NO TRACE  
100 PRINT "Slut"  
RUN  
Start  
40 50 60 Nr 0  
70 40 50 60 Nr 1  
70 40 50 60 Nr 2  
70 40 80 90 Slut
```

TRACE-funktionen är avstängd före rad 40 och efter rad 90.

NO EXTEND

Format	NO EXTEND
Funktion	Avslutar arbete i EXTEND -mod.
Användning	När man arbetar under NO EXTEND , får variabelnamn bestå av en bokstav följt av en valfri siffra. Om inget annat begärts, arbetar BASIC med NO EXTEND .

ON ERROR GOTO

Format	ON ERROR GOTO [radnummer]
Funktion	Ger hopp till angivet radnummer vid fel.
Användning	Se vidare avsnitt 2.6
Observera	Om radnummer utelämnas sker inget hopp vid fel. För återhopp används RESUME .

ON-GOSUB

Format	ON uttryck GOSUB radnummer[,radnummer,...]
Funktion	Uthopp sker till ett av radnumren i listan beroende på uttryckets värde (heltal). Återhopp sker till raden efter ON-GOSUB . Om uttryckets värde adresserar ett radnummer, som inte finns i listan, fås en felutskrift.
Exempel	Används för att villkorligt hoppa till en av flera subrutiner eller en av flera ingångar i en subrutin.

```
10 FOR X=1 TO 5
20 PRINT X
40 ON X GOSUB 1300,200,1300,400,1300
50 PRINT A$
60 NEXT X
70 END
200 LET A$="Sub200"
210 RETURN
400 LET A$="Sub400"
410 RETURN
1300 LET A$="Sub1300"
1310 RETURN
```

Uthopp sker till rad	för X-värdet
1300	1
200	2
1300	3
400	4
1300	5

ON-GOTO

Format	ON uttryck GOTO radnummer[,radnummer,...]
Funktion	Instruktionen ON-GOTO gör det möjligt att hoppa till en av flera rader, beroende på uttryckets värde.
Exempel	<pre>100 ON A/B GOTO 1000,1500,1700</pre>

ger uthopp enligt följande tabell:

1. rad nr 1000 om $0.5 \leq A/B < 1.5$
2. rad nr 1500 om $1.5 \leq A/B < 2.5$
3. rad nr 1700 om $2.5 \leq A/B < 3.5$
4. fel om $A/B < 0.5$
5. fel om $A/B \leq 3.5$

ON-RESTORE

Format	ON uttryck RESTORE radnummer[,radnummer,...]
Funktion	Ställer DATA -pekaren efter samma urvalsförfarande som ON-GOTO -satsen.
Användning	Satsen ON-RESTORE kan på detta sätt användas för att ge villkorlig ändring av DATA -pekaren till en viss plats i data-buffern.

Exempel

```
10 FOR X=1 TO 3
20 READ A,B,C
30 ON X RESTORE 60,70,80
40 PRINT A;B;C
50 NEXT X
60 DATA 1,2,3
70 DATA 4,5,6
80 DATA 7,8,9
90 END
RUN
1 2 3
1 2 3
4 5 6
```

ON-RESUME

Format	ON uttryck RESUME radnummer[,radnummer,...]
Funktion	Instruktionen ON-RESUME gör det möjligt att hoppa till en av flera rader, beroende på uttryckets värde, då satsen utförs. Felhantering återställs.

Användning **ON-RESUME** används för villkorligt återhopp efter felhantering.

Exempel

```
10 ON ERROR GOTO 100
|
|
100 REM Felhantering
|
|
150 ON A RESUME 1000,2000
```

Observera **ON-RESUME** används tillsammans med **ON ERROR GOTO**.

OPEN

Format **OPEN** "[enhet:][filnamn[.typ]]" **AS FILE** filnummer

Där <enhet:> kan vara

DR0: Drivenhet 0

DR1: Drivenhet 1

PR: Skrivare

CON: Bildskärm/tangentbord

Uttrycket efter **AS FILE** ska vara ett heltalsvärde mellan 0 och 255.

Om skrivare öppnas, utelämnas <filnamn.typ>.

Funktion Används för att öppna en fil med ett filnummer, som gäller inom det aktuella BASIC-programmet.

Användning **OPEN** används för filer, som redan existerar.

Datafiler eller enheter har både sitt eget namn, som identifierar dem i systemet, och ett filnummer, som identifierar dem i programmet. **OPEN**-satsen ger sambandet mellan namnet och filnumret.

Skrivning på och läsning från filen sker med bl.a. **INPUT**, **PRINT**, **GET** och **PUT**.

Observera För att kunna läsa data i en redan existerande fil, ska man alltid öppna filen med **OPEN**. Maximalt sju filer får vara öppna samtidigt.

Exempel

Ex. 1

```
50 OPEN "TEST.TXT" AS FILE 1
```

Ex. 2

```
10 OPEN "DATA.TXT" AS FILE 2
20 INPUT #2,A
30 INPUT #2,B
40 INPUT #2,C7X
```

Värdet på variablerna A, B och C7X läses in från den fil, som öppnats som fil 2. Dessa värden läses direkt efter de senast lästa värdena. Om filen ska läsas från början, måste den öppnas igen med en **OPEN**-sats.

OPTION BASE

Format

OPTION BASE n
där n=0 eller 1

Funktion

Anger lägsta värdet för vektorindex.

Observera

Om ingen **OPTION BASE** angivits, gäller 0 som lägsta index. Deklarationen av **OPTION BASE** måste ligga före all dimensionering och användning av vektorer.

POSIT

Format

POSIT # filnummer, position
POSIT (filnummer)

Funktion

Positionerar filpekare, eller läser filpekarens läge.

Användning

POSIT används för flyttning av filpekaren till angiven position räknat från filens början (första positionen). Första positionen = 0. **POSIT** används tillsammans med **GET** och **PUT**. Instruktionen pekar ut ett speciellt tecken eller första tecknet i en teckenföljd, som ska läsas eller skrivas. **POSIT(filnummer)** används för läsning av filpekarens läge (position). Se avsnitt 3.5.

Exempel

Ex. 1

```
10 POSIT # 1,5
```

Filpekaren flyttas till position 5, d.v.s. pekar på det sjätte tecknet i fil 1.

Ex. 2

```
50 A=POSIT(1)
```

A = filpekarens position. I Ex. 1 befinner sig filpekaren i position 5, d.v.s. A=5.

PREPARE

Format	PREPARE "[enhet:][filnamn.typ]" AS FILE filnummer
Funktion	Skapar och öppnar en ny fil med ett filnummer, som gäller inom det aktuella BASIC-programmet.
Användning	PREPARE används som OPEN , men skapar en ny fil. OPEN används då filen redan finns.

Exempel

```
10 PREPARE "DATA.TXT" AS FILE 2
20 PRINT # 2,A
30 PRINT # 2,B
40 PRINT # 2,C
```

Värdet på variablerna A, B och C skrivs på fil 2 (DATA.TXT).

PRINT

Format	PRINT [#filnummer] "data"/variabel [, "data"/variabel] Där # <filnummer> motsvarar filnumret i instruktionerna OPEN och PREPARE . Om filnumret utelämnas, skrivs värdena på bildskärmen. PRINT kan bytas mot ; (semikolon). Om ett element inte är en enkel variabel eller konstant, beräknas uttryckets värde, innan data skrivs ut. Instruktionen kan också innehålla teckensträngar inom citationstecken.
Funktion	Matar ut data i ASCII-form.
Användning	Användning av PRINT i samband med grafik beskrivs i kapitel 11.

Positionerna på en rad är numrerade från 0 till 39 alternativt 79. Raden är indelad i kolumner, fasta tablägen, som börjar i pos. 0, 15, 30, 45, 60 och 75 vid **SINGLE**. Vid **DOUBLE** blir positionerna 0, 30 och 60. Ett kommatecken (,) efter en variabel eller en sträng i **PRINT**-listan innebär, att nästa element i listan börjar skrivas ut i nästa kolumn. Två kommatecken efter varann i listan innebär, att en kolumn hoppas över vid utskriften.

Ett semikolon (;) efter en variabel eller en sträng i listan innebär, att nästa element i listan börjar skrivas ut i nästa position, d.v.s. omedelbart efter det föregående tecknet. Om listan avslutas med ett semikolon, ges ingen radframmatning efter **PRINT**-satsen.

Då en rad är fullskriven, görs en radframmatning, och utskriften fortsätter på nästa rad. För att styra **PRINT**-satsens utskrifter till vissa, bestämda lägen används funktionerna **TAB** och **CUR**. Dessa funktioner ger information om vilken position utskriften skall påbörjas.

En **PRINT**-sats utan argument ger vagnretur och radframmatning, d.v.s. en blankrad.

Exempel

Ex. 1

```
110 PRINT X;Y;"5"
120 PRINT
130 PRINT "Värde= ";X3,"SAM2= ";A+2
140 END
```

Ex. 2

```
10 LET A=5
20 LET B=2
30 PRINT A,B,A+B,A×B,A-B,B-A,A/B
40 END
```

Ex. 3

```
100 PREPARE "DR0:SKRFIL.DAT" AS FILE 3
110 PRINT # 3,A
```

skapar filen SKRFIL.DAT på skivan i DR0: och skriver värdet av variabeln A i filen.

PRINT USING

Format

PRINT [# filnummer] **USING** "formatsträng"; "data"/variabel
[,"data"/variabel,...]

Där <"formatsträng">, med citationstecken, är sammansatt av speciella formateringstecken. Dessa tecken, som beskrivs nedan, bestämmer uppställning och format för de strängar och tal, som ska skrivas.

Funktion

Skriver tal och strängar med angivet format.

STRÄNGFÄLT

När **PRINT USING** används för att formatera strängar, kan man välja mellan tre styrtecken för att ange formatet:

"!"

Anger att bara det första tecknet i strängen ska skrivas ut.

"Ö n blankaÖ"

Anger, att strängens 2+n första tecken ska skrivas ut.
<n blanka> är n stycken mellanslag. Om enbart bokstäverna "ÖÖ" (versaler) skrivs, kommer två tecken att skrivas ut o.s.v. Om strängen är längre än det angivna antalet tecken i fältet, skrivs de första tecknen ut och resten ignoreras. Om det angivna fältet är större än strängen, blir strängen vänsterjusterad i fältet. Resten av fältet fylls med blanka tecken (mellanslag).

Exempel

```
10 Aα="Se" : Bα="upp"
20 PRINT USING "ÖÖ";Bα
30 PRINT USING " ÖÖ";Aα,Bα
40 PRINT USING " Ö Ö ";Aα,Bα,"!!"
RUN
up
Se up
Se upp !!
```

"&" Anger ett strängfält med variabel längd. När ett fält specificeras med "&", skrivs strängen ut precis som den ser ut.

Exempel

```
10 Aα="Se": Bα="upp"  
20 PRINT USING "!";Aα;  
30 PRINT USING "&";Bα  
RUN  
Supp
```

NUMERISKA FÄLT

Följande specialtecken kan användas för att formatera ett numeriskt fält (gäller även numeriska strängar).

#

Nummertecknet # används för att beteckna varje sifferposition. Alla sifferpositioner fylls ut vid utskrift. Om det tal, som ska skrivas, har färre siffror än det reserverade antalet positioner, högerjusteras talet i fältet och föregås av mellanslag.

En decimalpunkt kan sättas i valfri position i fältet. Om formatsträngen anger, att en siffra ska stå före decimalpunkten, skrivs siffran alltid ut (ev. 0). Talen avrundas vid behov.

```
PRINT USING "# #.# #";.78 ger utskriften  
0.78  
  
PRINT USING "# # #.# #";987.654 ger utskriften  
987.65  
  
PRINT USING "# #.# # #";10.2,5.3,66.789,.234  
10.20 5.30 66.79 0.23
```

I det sista exemplet avslutas formatsträngen med två mellanslag, för att de utskrivna värdena inte ska hamna alldeles intill varann på raden.

+

Ett plustecken i början eller slutet av formatsträngen medför, att talets tecken (+ eller -) alltid skrivs ut före eller efter talet.

-

Ett minustecken i slutet av formatsträngen medför, att de negativa talens minustecken placeras efter talet i stället för före, som är det normala.

```
PRINT USING"+# #.# #";-68.95,2.4,55.6,-.9  
-68.95 +2.40 +55.60 -0.90  
  
PRINT USING"# #.# #-";-68.95,22.449,-7.01  
68.95- 22.45 7.01-
```

**

Två asterisker i början av formatsträngen medför, att blanka tecken i början av ett tal fylls med asterisker. Tecknen ** reserverar också plats för två siffror.

```
PRINT USING"*# #.# ";12.39,-0.9,765.1  
*12.4 *-0.9 765.1
```

¤¤

Två ¤¤ gör att ett valutatecken skrivs omedelbart till vänster om talet. ¤¤ reserverar också två platser, varav den ena används för ¤ i utskriften. Negativa tal måste ha minustecken i slutet av talet. Denna funktion används mest med amerikansk teckenuppsättning, där ¤ motsvaras av \$ (dollartecken).

```
PRINT USING "¤¤ #.#.# #";456.78
¤456.78
```

¤¤¤

¤¤¤ i början av en formatsträng kombinerar verkan av ¤¤ och ¤¤. Mellanslag i början av ett tal fylls med ¤ och ¤-tecken skrivs före talet. ¤¤¤ reserverar tre positioner, varav en används för ¤-tecknet.

```
PRINT USING "¤¤¤ #.#.# #";2.34
¤¤¤¤2.34
```

,

Ett kommatecken till vänster om decimalpunkten i en formatsträng medför, att ett mellanslag skrivs som avgränsare före var tredje siffra räknat från decimalpunkten. Ett kommatecken i slutet av formatsträngen skrivs ut som en del av strängen. Detta kommatecken är en avgränsare mellan två tal. Kommatecknet har ingen inverkan, om det används med exponentialformat.

```
PRINT USING "#.#.#.#.# #";1234.5
1 234.50
```

```
PRINT USING "#.#.#.#.# #,";1234.5
1234.50,
```

ÜÜÜÜ

Fyra versala Ü kan placeras i slutet av formatsträngen för att ange exponentialformat. De fyra tecknen reserverar utrymme för E+xx. Decimalpunkten kan sättas i valfritt läge. Exponenten anpassas efter det valda formatet. Om man inte särskilt anger, var + eller - ska placeras, används en sifferposition i början av talet som teckenposition. I denna position skrivs antingen ett mellanslag eller ett minustecken.

```
PRINT USING "#.#.# #ÜÜÜÜ";234.56
2.35E+02
```

```
PRINT USING ".#.#.# #ÜÜÜÜ";-888888
-.8889E+06
```

```
PRINT USING "+.# #ÜÜÜÜ";123
+.12E+03
```

—

Ett understyrkningstecken i formatsträngen gör, att nästa tecken skrivs ut som det står i strängen.

```
PRINT USING "_l#.#.# #_l";12.34
!12.34!
```

Om man vill skriva ut ett `_`-tecken, ska formatsträngen innehålla `"_ "`.

`%` Om talet, som ska 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.

```
PRINT USING"# #.# #";111.22
%111.22

PRINT USING".# #";.999
%.999
```

PUT

Format `PUT #filnummer, strängvariabel`

Där `# <filnummer>` ger det filnummer, som definierats med någon av satserna **OPEN** eller **PREPARE**. `<strängvariabel>` kan vara en strängvariabel eller ett stränguttryck.

Funktion Skriver en strängvariabel i binär form.

Användning Se avsnitt 3.5.

RANDOMIZE

Format `RANDOMIZE`

Funktion Sätter ett slumpmässigt startvärde för funktionen **RND** (slumptalsgeneratoren).

Användning Ska stå före det första anropet av slumptalsgeneratoren **RND** i programmet. **RANDOMIZE** får slumptalsgeneratoren att ge olika värden för varje gång programmet körs, genom att **RND** får ett nytt startvärde då **RANDOMIZE** påträffas.

VARNING Bör bara användas en gång i varje program.

READ

Format `READ variabel [, variabel, ...]`

Funktion Utgör, tillsammans med **DATA**-satser, ett sätt att tilldela variabler värden.

Användning Instruktionen **READ** tilldelar variablerna i listan i tur och ordning värden från **DATA**-satserna. Innan programmet körs, har **BASIC** sammanfört alla data från **DATA**-satserna till ett datablock. Varje gång en **READ**-sats exekveras, hämtas nästa data från datablocket. **READ** används tillsammans med **DATA**.

Om man behöver använda samma data mer än en gång i ett program, kan man åstadkomma detta med instruktionerna **RESTORE** eller **ON RESTORE** Se vidare under dessa instruktioner.

Exempel

Ex. 1

```
100 READ A,B,C,D,X1,X2
|
|
150 DATA 3,6,1.8
200 DATA 6.83E-3,-86.4,3.14
```

När programmet genomlöps, får variablerna följande värden:

```
A=3
B=6
C=1.8
D=6.83E-3
X1=-86.4
X2=3.14
```

Ex. 2

```
10 READ A$,B$,C$
20 PRINT A$,B$,C$
30 DATA OSKAR, MATS, ""STEN""
RUN
OSKAR MATS "STEN"
```

Observera

Om ett kommatecken, citationstecken eller apostrof ska ingå i en sträng, måste det stå inom citationstecken.

REM

Format

REM text
! text

Funktion

Där <text> får innehålla alla tecken på tangentbordet. BASIC-tolken ignorerar allt, som står efter **REM** eller ! på en rad. Inleder kommentar i program.

Användning

Man bör lägga in täta kommentarer i sina program, för att göra dem mera lättlästa och för att underlätta underhåll.

Exempel

```
10 REM Detta program beräknar
20 ! medelvärdet.
2010 !***** Skriver ut en tabellrad *****
```

RESTORE

Format

RESTORE [radnummer]

Funktion

Gör det möjligt att använda innehållet i **DATA**-satser flera gånger.

Exempel

Ex. 1

```
60 RESTORE
```

Ställer datapekaren till början av den första **DATA**-satsen i programmet.

Ex. 2

```
50 RESTORE 100
```

Ställer datapekaren till **DATA**-satsen med radnummer 100.

RESUME

Format **RESUME** [radnummer]

Funktion Återhopp från felhanteringsrutin.

Användning När felet är åtgärdat kan man fortsätta att köra programmet, genom att placera en **RESUME**-sats i slutet av felhanteringsrutinen.

Om man vill hoppa in på någon annan rad i programmet anges radnumret i **RESUME**-satsen.

Exempel

```
2000 RESUME  
2010 RESUME 100
```

Rad 2000 ger återhopp till den rad, där felet genererades. Rad 2010 ger återhopp till rad 100.

RETURN

Format **RETURN** [variabel]

Funktion Ger återhopp från subrutin eller flerradiga funktioner.

RETURN ger återhopp från subrutin till satsen närmast efter anropet.

RETURN <variabel> ger återhopp från funktionen med funktionsvärdet <variabel>.

Användning Se **GOSUB** och **DEF FN**.

SINGLE

Format **SINGLE**

Funktion Ändrar alla variabler och uttryck, som är flyttal, till enkel precision (7 siffror).

Användning	Deklarationen SINGLE måste göras innan variablerna används och kan inte ändras, när programmet väl startats med RUN . Om en rad har ändrats (editerats) eller om kommandot CLEAR har använts, kan man ändra SINGLE till DOUBLE eller tvärtom. Om inget annat deklarerats, gäller SINGLE .
Observera	Det är inte tillåtet att blanda SINGLE och DOUBLE i samma program.

STOP

Format	STOP
Funktion	Stoppar programexekveringen.
Användning	Instruktionen STOP stoppar exekveringen av programmet. Variablernas värden kvarstår och inga filer stängs. STOP -instruktionen används med fördel vid felsökning. STOP -satser får finnas på flera ställen i ett program. Följande utskrift erhålls:
	STOP IN LINE radnummer
Observera	Programexekveringen kan fortsättas med något av kommandona CON eller GOTO .

TRACE

Format	TRACE [# filnummer]
Funktion	Skriver ut radnumren på de programrader, som genomlöps.
Användning	Vid felsökning i program, för att spåra hur programmet utförs.

Exempel

```
100 OPEN "PR:" AS FILE 1%
110 A=12.345
115 TRACE # 1%
120 B=123
125 IF A=0 THEN STOP
130 C%=B
135 X=A*2
140 D1%=A
145 NOTRACE
150 PRINT # 1% A,B,C%,D1%,X
160 CLOSE 1%
170 END
RUN
```

Följande utskrift erhålls på skrivare:
120 125 130 135 140 145
12.345 123 123 12 24.69

WEND

Format	WEND
Funktion	WEND avslutar en loop, som inleds av WHILE .
Användning	Se WHILE .

WHILE

Format	WHILE uttryck
Funktion	Anger villkor för uthopp ur en programloop.
Användning	I programloopar där de värden, som testas i uthoppsvillkoret, ändras då loopen genomlöps. Jämför med FOR -loopar, där uthoppsvillkoret automatiskt uppnås, oavsett innehållet i loopen. Det kan vara önskvärt, att utföra loopen tills ett visst värde har uppnåtts.
Exempel	<pre>10 WHILE X<10 20 X=X*X+1 30 WEND</pre>

Innan loopen genomlöps första gången och vid varje nytt varv undersöks villkoret $X < 10$. Så länge detta är sant, fortsätter iterationen.

WIDTH

Format	WIDTH [#filnummer,] antal
Funktion	Anger antal tecken per rad för aktuell fil. Då # filnummer utelämnas avses bildskärmen (40 alternativt 80 tecken). (Antalet tecken är ett tal mellan 0 och 255).
Användning	Används för omställning av radlängd
Exempel	<pre>10 WIDTH 40</pre>

Ställer om datorn till 40-teckens mod

```
10 WIDTH # 1, 64
```

Ställer fil 1 till 64 tecken per rad

10 Funktioner

10.1 Matematiska funktioner

Varje programmerare träffar ofta på diverse, vanliga, matematiska operationer. Deras resultat brukar finnas i matematiska tabellverk. Detta gäller t.ex. sinus, cosinus, kvadratrötter, logaritmer och många andra. Datorn kan beräkna dessa värden snabbt och noggrant. Man har därför valt att bygga in ett antal sådana funktioner i BASIC II. När ett av dessa funktionsvärden behövs i en beräkning,

anropas de inbyggda funktionerna t.ex.:

```
SIN(23 * PI/180)  
LOG(144)
```

Här följer en tabell över de matematiska funktionerna.

ABS(X)	absolutvärdet av X
ATN(X)	arcustangens för X
COS(X)	cosinus för X (X i radianer)
EXP(X)	e^{**X} , där $e = 2.71828$ (i enkel precision)
FIX(X)	trunkerat värde av X som SGN(X)*INT (ABS(X))
HEX↔(X)	omvandlar decimalt tal till hexadecimal sträng
INT(X)	största heltal $\leq X$
LOG(X)	naturliga logaritmen för X
LOG10(X)	tiologaritmen för X
MOD(X,Y)	resten vid heltalsdivisionen X/Y
OCT↔(X)	omvandlar decimalt tal till oktal sträng
PI	konstant med värdet 3.14159 (enkel precision)
RND	slumptal mellan 0 och 0.999999. RND genererar samma följd av slumptal varje gång ett program körs, om inte en RANDOMIZE -sats finns före RND i programmet.
SGN(X)	0 för $X = 0$, -1 för $X < 0$, $+1$ för $X > 0$
SIN(X)	sinus för X (X i radianer)
SQR(X)	kvadratroten ur X
TAN(X)	tangenten för X (X i radianer)

ABS

Format	ABS(argument)
Funktion	Ger absolutvärdet av argumentet.
Exempel	<code>10 Y=ABS(-3.1)</code> Detta ger Y=3.1

ATN

Format	ATN(argument)
Funktion	Ger arcustangens för argumentet i radianer.
Exempel	<code>10 Y=ATN(PI/2)</code> Detta ger Y=1

COS

Format	COS(argument)
Funktion	Ger cosinus för argumentet. Argumentet ska anges i radianer.
Exempel	<code>10 Y=COS(0)</code> Detta ger Y=1

EXP

Format	EXP(argument)
Funktion	Ger $e \times \times$ argumentet, där $e = 2.71828$ (i enkel precision).
Exempel	<code>10 Y=EXP(1)</code> Detta ger Y=2.71828

FIX

Format	FIX(argument)
Funktion	Ger trunkerat värde av argumentet (X) d.v.s. SGN(X)*INT(ABS(X)) .
Exempel	<code>10 Y%=FIX(-.5)</code> Detta ger Y=0
Observera	Jämför INT .

HEX

Format	HEX (argument)
Funktion	Omvandlar decimalt tal till hexadecimal sträng
Exempel	<code>10 Y=HEX(255)</code> Detta ger Y="FF"

INT

Format	INT (argument)
Funktion	Ger värdet av det största heltal, som är mindre än eller lika med argumentet. Jämför FIX .
Användning	INT kan användas, då man vill ha ett tal avrundat. Man använder då INT (X+.5) Funktionen INT kan också användas vid avrundning till önskat antal decimaler enligt följande: $\text{INT}(X \times 10^{D\%} + .5) / 10^{D\%}$ där D% är antalet decimaler, som önskas. Om talet är negativt, ger INT det största heltal, som är mindre än talet självt.
Exempel	Ex. 1 <code>10 Y=INT(34.67)</code> Detta ger Y=34 Ex. 2 <code>10 Y=INT(34.67+.5)</code> Detta ger Y=35 Ex. 3 <code>10 Y=INT(-23.15)</code> Detta ger Y=-24

LOG

Format	LOG (argument)
Funktion	Ger naturliga logaritmen för argumentet.
Exempel	<code>10 Y=LOG(2)</code> Detta ger Y=.693147

LOG10

Format **LOG10(argument)**
Funktion Ger tiologaritmen för argumentet.

Exempel `10 Y=LOG10(10)`

Detta ger Y=1

MOD

Format **MOD(argument1,argument2)**
Funktion Ger resten vid heltalsdivision av argumenten.

Exempel `10 Y=MOD(22,4)`

Detta ger Y=2

OCT↯

Format **OCT↯(argument)**
Funktion Omvandlar decimalt tal till oktal sträng.

Exempel `10 Y↯=OCT↯(59)`

Detta ger Y↯="73"

PI

Format **PI**
Funktion Konstant med värdet 3.14159 (enkel precision)

Exempel `10 Y=2*PI`

Detta ger Y=6.28319

RND

Format **RND**
Funktion Ger ett slumpstal mellan 0 och 0.999999. **RND** genererar samma följd av slumpstal varje gång ett program körs, om inte en **RANDOMIZE**-sats finns före **RND** i programmet.

Exempel Ex. 1

`10 Y=RND`

Ex. 2

`10 Y=(B-A)*RND+A`

Y tilldelas ett slumpstal i intervallet (A,B).

Observera Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

ASCII

Format **ASCII(A α)**

Funktion Ger ett heltal, som är lika med ASCII-värdet (decimalt) för första tecknet i A α .

Exempel **10 A = ASCII("T")**

Detta ger A = 84

CHR α

Format **CHR α (argument [,argument, ...])**

Funktion Ger en teckensträng, som motsvarar argumentens ASCII-värden.

Exempel **PRINT CHR α (65)**

Detta ger utskriften A.

COMP%

Format **COMP%(A α ,B α)**

Funktion Ger värdet -1, 0 eller 1 som resultat av en numerisk jämförelse av två numeriska strängar. Funktionsvärdet är -1 om A α <B α , 0 om A α =B α och 1 om A α >B α .

Exempel

```
30 A $\alpha$ ="123.456" : B $\alpha$ ="12.8907"  
40 T%=COMP%(A $\alpha$ ,B $\alpha$ )  
50 PRINT T%  
60 PRINT COMP%(B $\alpha$ ,A $\alpha$ )  
100 END  
RUN  
1  
-1
```

DIV α

Format **DIV α (A α ,B α ,P%)**

Funktion Ger kvoten A α /B α avrundad till P% decimaler. Om P% föregås av — ges summan med antalet tillgängliga siffror, upp till P% stycken.

Exempel

```
100 LET C $\alpha$ ="3.5"  
110 V9 $\alpha$ =DIV $\alpha$ (C $\alpha$ ,"1.7777",3%)  
120 PRINT V9 $\alpha$   
200 END  
RUN  
1.969
```

Observera Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

INSTR

Format **INSTR(N%,A\$,B\$)**

Funktion Söker efter strängen B\$ i A\$ med början i position N%. Ger värdet 0, om B\$ ej finns i den undersökta delen av A\$, annars värdet för den position i A\$, där B\$ börjar. Positionen räknas från strängens början. Första tecknet står i position 1.

Exempel

```
10 A$="AaBbCcDdEeFf"  
20 PRINT INSTR(5%,A$,"eF")  
30 END  
RUN  
10
```

LEFT

Format **LEFT[\$](A\$,I%)**

Funktion Ger de I% första tecknen av strängen A\$.

Exempel

```
10 D2$='ABCDEFGHJKLMNOPQRSTUVWXYZÅÖ'  
20 T8$=LEFT$(D2$,6%)  
40 PRINT T8$  
100 END  
RUN  
ABCDEF
```

LEN

Format **LEN(A\$)**

Funktion Ger antalet tecken i strängen A\$, d.v.s. stränglängden (inklusive mellanslag).

Exempel

```
10 S$="MOTALA"  
20 PRINT LEN(S$)  
200 END  
RUN  
6
```

MID

Format **MID[\$](A\$,P%,K%)**

Funktion Tillordnar tecken nr P% t.o.m. P%+K%-1 av A\$ det nya värdet, d.v.s. byter tecken i strängen.

Exempel

```
10 A$="ABCDEFGHI"  
20 MID$(A$,6%,2%)="MM"  
30 PRINT A$  
60 END  
RUN  
ABCDEMMHI
```

MID

Format **MID**[α]($A\alpha$, $P\%$, $K\%$)

Funktion Ger den delsträng av $A\alpha$, som börjar i position $P\%$ och är $K\%$ tecken lång, d.v.s. tecknen nr $P\%$ t.o.m. $P\% + K\% - 1$.

Exempel

```
200 W $\alpha$  = "radframmatning"  
210 A2 $\alpha$  = MID $\alpha$ (W $\alpha$ ,8%,3%)  
220 PRINT A2 $\alpha$   
500 END  
RUN  
mat
```

MUL α

Format **MUL** α ($A\alpha$, $B\alpha$, $P\%$)

Funktion Ger produkten $A\alpha \times B\alpha$ med $P\%$ decimaler. Om $P\%$ föregås av $-$ ges summan med antalet tillgängliga siffror, upp till $P\%$ stycken.

Exempel

```
10 LET A $\alpha$  = "12345.6789"  
20 LET B $\alpha$  = "987.54321"  
30 Y $\alpha$  = MUL $\alpha$ (A $\alpha$ ,B $\alpha$ ,6%)  
40 PRINT Y $\alpha$   
50 END  
RUN  
12191891.370535
```

NUM α

Format **NUM** α (argument)

Funktion Ger den numeriska sträng, som motsvarar argumentet. Strängen skrivs enligt följande: positivt tal – teckenposition markeras inte, negativt tal – minustecken skrivs ut.

Exempel

```
10 PRINT NUM $\alpha$ (345709702134)  
20 END  
RUN  
3.4571E + 11
```

RIGHT

Format **RIGHT**[α]($A\alpha$, $N\%$)

Funktion Ger de sista tecknen i $A\alpha$, fr.o.m position $N\%$.

Exempel

```
10 M $\alpha$  = "ABCDEFGHijkl"  
20 H $\alpha$  = RIGHT $\alpha$ (M $\alpha$ ,7%)  
30 PRINT H $\alpha$   
90 END  
RUN  
GHIJKL
```


SPACE

Format	SPACE(N%)
Funktion	Ger en sträng, som består av N% mellanslag.
Exempel	<pre>10 Y=SPACE(10)</pre>

Detta ger en sträng (Y) med 10 blanka.

STRING

Format	STRING(I%,K%)
Funktion	Ger en sträng med I% st ASCII-tecken, d.v.s. strängen har längden I% och består av likadana tecken, vars ASCII-värde är K%.

Exempel	<pre>10 G5=STRING(4%,33%) 20 PRINT G5 30 END RUN !!!!</pre>
---------	---

SUB

Format	SUB(A,B,P%)
Funktion	Ger den aritmetiska differensen $A-B$ för de numeriska strängarna A och B med P% decimaler. Om P% föregås av — ges summan med antalet tillgängliga siffror, upp till P% stycken.

Exempel	<pre>10 LET H="9876.54321" 20 PRINT SUB(H,'98.76',5%) 30 END RUN 9777.78321</pre>
---------	---

Observera Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

VAL

Format	VAL(A)
Funktion	Beräknar numeriska värdet av den numeriska strängen A. En numerisk sträng får innehålla siffror, +, -, . och E. Resultatet ges i form av ett flyttal.

Exempel	<pre>330 V4=VAL("14.3E-5") 340 PRINT V4 400 END RUN .000143</pre>
---------	---

A α +B α

Format A α +B α

Funktion Sammanfogar (konkatenerar) strängar.

Exempel

```
10 D $\alpha$ ="God"  
20 S $\alpha$ ="Jul"  
* 30 A $\alpha$ =D $\alpha$ +" "+S $\alpha$ 
```

Detta ger A α ="God Jul"

10.3 Övriga funktioner

CALL(A%, D%) anropar maskinspråksrutiner.

CUR(M%,N%) sätter markören på rad M%, position N%

CVT% α (X%)
CVT α %(X α) byter typ på variabel från heltal till sträng och vice versa

CVTF α (X)
CVT α F(X α) byter typ på variabel från flyttal till sträng och vice versa

ERRCODE ger värdet av senast genererade felkod

FN användardefinierad funktion

INP(I%) ger datavärde från inport I%

OUT skickar data till utport

PEEK(I%) ger minnesinnehållet i adress I%

PEEK2(I%) **PEEK(I%)+256*PEEK(I%+1%)**

POKE skriver i angiven minnesadress

SWAP%(N%) första och andra byten av N% skiftar plats

SYS(I%) ger systemstatus

TAB(I%) flyttar skrivhuvud eller markör till position I% på raden

TIME α ger datum och tid

VAROOT(X) ger adressen till variabeln X

VARPTR(X) ger adressen till värdet för X

CALL

Format	CALL (A%[,D%])
Funktion	Anropar maskinspråksrutiner med början i adress A% (decimalt). Ger ett funktionsvärde från Z80-processorns HL-register vid återhoppet till BASIC. D%, om det anges, läggs i Z80-processorns DE-register vid anropet.
WARNING	Detta är en maskinorienterad funktion, som endast bör användas vid avancerad programmering. Eftersom den adresserar processorn direkt, kan en körning förstöras, om CALL används på fel sätt.

CUR

Format	CUR (R%,N%)
	där R% (rad) finns i intervallet 0—24 (fysiskt 1—25) och N% (position) i intervallet 0—39/79. Rad 24 är en statusrad.
Funktion	Sätter markören på rad R%, position N%
Användning	Vid tabellutskriften eller i samband med grafik.
Exempel	<pre>10 PRINT CUR(12,20)"Text"</pre>

CVT

Format	CVT% ⌘(heltalsvariabel) CVT ⌘%(strängvariabel) CVTF ⌘(flyttalsvariabel) CVT ⌘F(strängvariabel)
Funktion	Lagrar tal som strängar respektive återskapar talen.
Användning	CVT-funktionen används för att spara utrymme på skiva. Numeriska värden, som lagras på flexskiva, tar lika stor plats, som när de skrivs ut med PRINT . Heltal kräver upp till sex tecken, flyttal i enkel precision (SINGLE) tolv tecken och flyttal i dubbel precision (DOUBLE) behöver tjugotvå tecken. Varje tecken lagras i en byte. Med hjälp av funktionen CVT (av engelska convert) kan man spara dessa data i respektive 2, 4 och 8 bytes.
Exempel	<pre>10 PREPARE "TAL.DAT" AS FILE 1 20 I% = 15973% 30 PUT #1, CVT%⌘(I%) 40 CLOSE 1</pre>

Heltalet I% har nu lagrats på filen TAL.DAT i två bytes. För att återskapa talet kan följande program användas:

```

10 OPEN "TAL.DAT" AS FILE 1
20 GET #1,A $\alpha$  COUNT 2
30 I%=CVT $\alpha$ %(A $\alpha$ )
40 CLOSE 1

```

Nedan följer ett exempel på hur ett flyttal kan sparas. Flyttalet i exemplet kan vara antingen enkel eller dubbel precision:

```

10 DIM A(100)
20 PREPARE "TAL2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 PUT #1%,CVTF $\alpha$ (A(I%))
50 NEXT I%
60 CLOSE 1%

```

Nästa exempel visar, hur talet i ovanstående exempel återskapas.

```

10 DIM A(100): L%=LEN(CVTF $\alpha$ (0))
20 OPEN "TAL2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 GET #1%,A $\alpha$  COUNT L%: A(I%)=CVT $\alpha$ F(A $\alpha$ )
50 NEXT I%
60 CLOSE 1%

```

Observera

Man använder **LEN(CVTF α (0))** för att undersöka om enkel eller dubbel precision används.

ERRCODE

Format

ERRCODE

Funktion

Ger värdet av senast genererade felkod. Är 0 om inga fel indikerats.

FN

Format

FNidentifierare[%/ α] [(parameter [,parameter,...])]

där <identifierare> utgör funktionens namn.

Funktion

Anrop av användardefinierad funktion. Jämför instruktionen **DEF FN**.

INP

Format

INP(I%)

Funktion

Ger datavärdet från inport I%.

WARNING

Detta är en maskinorienterad funktion, som ska användas enbart vid avancerad programmering.

OUT

Format	OUT port,data [port,data, ...] där portnummer och data anges i decimal form.
Funktion	Används för att adressera utportar vid utmätning av data.
WARNING	Detta är en maskinorienterad instruktion för avancerad programmering. Denna instruktion och instruktionen INP ger användaren tillgång till I/O-funktionerna i ABC 802 och dess I/O-buss.
Observera	För att välja en I/O-kanal måste man alltid först ge instruktionen OUT för denna kanal. I/O-kanalen i fråga förblir sedan tillgänglig, tills ett nytt val görs med OUT .

PEEK

Format	PEEK (I%)
Funktion	Ger minnesinnehållet i adress I%.
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.

PEEK2

Format	PEEK2 (B0%)
Funktion	Läser innehållet i två bytes enligt följande: $J\% = \text{PEEK}(B0\%) + 256 \times \text{PEEK}(B0\% + 1\%)$
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.

POKE

Format	POKE adress,data [,data, ...]
	Där <adress> anges decimalt. Om flera <data> anges, räknas adressen upp för varje nytt värde.
Funktion	Används för att ladda in ett värde i en minnescell.
Användning	POKE används huvudsakligen då BASIC samarbetar med maskinspråksrutiner
WARNING	Denna instruktion är maskinorienterad och ska endast användas vid avancerad programmering. Om den används på fel sätt, kan innehållet i datorns minne förstöras.

SWAP%

Format	SWAP%(N%)
Funktion	Första och andra byten av N% skiftar plats.
VARNING	Denna funktion är avsedd att användas vid avancerad programmering.

SYS

Format	SYS(1%)
Funktion	Ger systemstatus enligt lista nedan.
Användning	Programmeraren får tillgång till följande systemstatus: SYS(2) Totalt minnesutrymme SYS(3) Programmets storlek SYS(4) Kvarvarande minnesutrymme SYS(5) Tangentbordsflagga. Nollställs med GET, INPUT eller INPUT LINE . SYS(6) Läger tillbaka senast inlästa tecken i tangentbordsbufferten. SYS(8) Är -1 om tangent är nedtryckt SYS(11) Programmets startadress SYS(12) Variabelrot

Exempel

```
PRINT SYS(3)
```

Detta ger utskrift av programmets storlek.

TAB

Format	TAB(1%) där 1% kan ha värdet 1 – 40/80
Funktion	Flyttar skrivhuvud eller markör till position 1% på raden.

Exempel

```
10 PRINT TAB(20)"Data"
```

TIME☒

Format	TIME☒
Funktion	Ger tidsangivelse år-mån-dag tim.min.sek Realtidsklockan laddas automatiskt från CMOS-kalendern vid spänningstillslag eller RESET.

Användning

Datorns realtidsklocka kan sättas med följande program:

```
10 PRINT CHR ☒(12)  
20 PRINT "*** ABC806 Sätt klockan! **"  
30 INPUT "Datum : ÅÅ,MM,DD ";Y,M,D !Mata in datum  
40 INPUT "Tid : HH,MM,SS ";H,M1,S !Mata in tid  
50 POKE — 17,Y,M,D,H,M1,S !Skriv tid i minnet  
60 PRINT CUR(12,12);TIME ☒ !Skriv tiden på bildskärm  
70 GOTO 60  
80 END
```

OBS!

Datorns CMOS kalender med klocka sättas med ett program som finns på den flexskiva som medföljer datom.

VAROOT

Format	VAROOT (variabel)
Funktion	Ger adressen till en tabell, som innehåller uppgifter om en variabel.
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.

VARPTR

Format	VARPTR (variabel)
Funktion	Ger adressen till en variabls värde.
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.

11 Grafik med attributhantering

ABC 806 kan använda grafik med 78/158 x 72 punkters upplösning. Om man arbetar med 40 teckens mod (78 punkter) är grafiken kompatibel med TELEDATA.

Datorn har ett separat attributminne vilket kan användas för att lagra text- och grafikstyrtecken. I TELEDATA mod lagras dock styrtecknen i teckenminnet. Genom att använda instruktionen **ATTRIBUTE 1** kan man bestämma vilken variant som ska användas.

Högupplösningsgrafik enligt kapitel 12 kan användas tillsammans med vanlig text och grafik på skärmen eftersom högupplösningsgrafiken lagras i ett separat minne. Genom att ange prioritet bestämmer man själv vad som ska visas främst.

11.1 TELEDATA-grafik

Om man vid programmeringen inte anger något attribut, eller anger **ATTRIBUTE 0**, lagras alla tecken i det ordinarie teckenminnet. Vid presentation på skärmen tar då varje styrtecken en position.

Denna mod i ABC 806 har grafik som motsvarar standarden för TELEDATA. I grafisk mod tolkas då varje utmatat tecken som en figur, bildad av sex grafiska punkter, eller som en versal bokstav.

Vid utskrift av text eller grafik på bildskärmen, styr man val av färg m m, genom speciella argument i den aktuella **PRINT**-satsen som påverkar en rad åt gången. Varje argument lägger ut ett styrtecken på skärmen. Dessa tecken syns inte men tar en position. De kan skrivas över med en bakgrundsfärg, om styrargumenten ges i lämplig ordningsföljd.

Följande färger är tillgängliga:

Svart	(BLK)
Röd	(RED)
Grön	(GRN)
Gul	(YEL)
Blå	(BLU)
Magenta	(MAG)
Cyan	(CYA)
Vit	(WHT)

Under 11.1.1 visas en tabell över tillgängliga tecken i ABC 806. Tabellen ger ASCII koden för varje tecken, dess betydelse i teckenmod och i grafisk mod. Ett sätt att planera en bild är att rita den på en grafikkarta, enligt 11.1.2, och sedan mata in lämpliga data till programmet.

När Du ritat bilden på grafikkartan, skriver Du en rad i taget. Glöm då inte att ta hänsyn till styrtecken om Du arbetar med **ATTRIBUTE 0** eller inte angett något attribut.

Observera att versaler (stora bokstäver) inte förändras i grafisk mod. Dessa kan alltså blandas fritt med grafiska tecken.

Antalet grafiska positioner per rad beror på om man valt 40 eller 80 tecken. Det är dock bara 40 tecken per rad som är standard i TELEDATA.

11.1.1 ASCII kod för tecken, grafik och argument

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

ASCII-koder (A) tolkade i teckenmod (T) och grafikmod (G).

Argument	CHRα()	Argument	CHRα()	Argument	CHRα()
BLK	128	ULN	138	GMAG	149
RED	129	NULN	139	GCYA	150
GRN	130	NRML	140	GWHT	151
YEL	131	DBLE	141	HIDE	152
BLU	132	EL	142	GCON	153
MAG	133	GBLK	144	GSEP	154
CYA	134	GREDE	145	BLBG	156
WHT	135	GGRN	146	NWBG	157
FLSH	136	GYEL	147	GHOL	158
STDY	137	GBLU	148	GREL	159

11.2 Attributhantering

Datorn har vid normal text och grafik en attributhantering som gör att inga styrtecken för grafik och liknande behöver sättas på skärmen. All sådan information lagras istället i attributminnet som kan innehålla information om

- Färg på tecken
- Dubbel teckenbredd
- Bakgrundsfärg
- Understrykning
- Fast tecken
- Dold text
- Blinkande tecken
- TELEDATA-grafik
- Dubbel teckenhöjd
- Soft-scroll, laddas från flexskiva

Anges därför **ATTRIBUTE 1** läggs alla styrtecken i det separata attributminnet vilket innebär att färger m m kan ändras tecken för tecken utan att det påverkar teckenminnet. Styrtecknet och tecknet ligger parallellt i minnet vilket medför att de läggs i samma position på bildskärmen vid visning.

11.3 Programexempel

Skillnaden mellan **ATTRIBUTE 0** och **ATTRIBUTE 1** framgår bäst av nedanstående två exempel, där varje styrtecken tar en position på skärmen vid **ATTRIBUTE 0**.

```
100 !
110 ! Exempel på användning
120 ! av instruktionen ATTRIBUTE 0
130 !
140 ATTRIBUTE 0
150 PRINT CHR ♂ (12)
160 PRINT 'Användning ATTRIBUTE 0'
170 PRINT
180 PRINT CYA NWBG BLK 'Styr' EL MAG 'tecken' FLSH RED ULN 'tar'
      NULN STDY NRML BLK 'teckenpositioner' BLBG WHT 'i bilden.'
```

```
100 !
110 ! Exempel på användning
120 ! av instruktionen ATTRIBUTE 1
130 !
140 ATTRIBUTE 1
150 PRINT CHR ♂ (12)
160 PRINT 'Användning ATTRIBUTE 1'
170 PRINT
180 PRINT CYA NWBG BLK 'Styr' EL MAG 'tecken' FLSH RED ULN 'tar EJ'
      NULN STDY NRML BLK 'teckenpositioner' BLBG WHT 'i bilden.'
```

11.4 Instruktioner

ATTRIBUTE

Format	ATTRIBUTE n
Funktion	Anger om text- och grafikstyrtecknen ska läggas i teckenminnet (kompatibel med ABC 800, gäller om inget attribut anges) eller attributminnet. Styr även funktionen soft-scroll.
	ATTRIBUTE 0 Kompatibel mod med ABC 800. Text- och grafikstyrtecknen läggs i teckenminnet.
	ATTRIBUTE 1 Text- och grafikstyrtecknen läggs i attributminnet.
	ATTRIBUTE 2 Ger soft-scroll.

En kombination av olika attribut är möjlig e x v 3 som ger både soft-scroll samt lagring i attributminnet. Drivrutinen **SOFTOPT. REL** som hanterar attribut 2 laddas från flexskiva.

PRINT

Format	PRINT [CUR(R,K)]argument[:argument;...] "text"
Funktion	Används för att skriva text och grafik. Argumenten styr färgval m.m. Ett G i början av färgvalsargument (t.ex. GREd) sätter raden i grafisk mod, varvid tecken inom citationstecken tolkas som grafik (se ASCII-tabell). Om CUR(R,K) används, börjar utskriften i den angivna positionen R = rad (0—24) och K = position (0—39/79).
	Följande argument finns:
	RED, GRN, YEL, BLU, MAG, CYA, WHT, BLK alfanumeriska tecken i färg
	GREd, GGRN, GYEL, GBLU, GMAG, GCYA, GWHT, GBLK grafik och versaler i färg
	FLSH, STDY blinkande resp. fast tecken
	EL, DBLE, NRML dubbel teckenbredd resp höjd normal bredd och höjd
	GCON, GSEP sammanhängande resp. separerad grafik OBS! kan ej genereras på ABC 806
	NWBG, BLBG ny bakgrund resp. svart bakgrund
	GHOL, GREL grafik över styrtecken resp. återgång OBS! kan ej genereras på ABC 806

HIDE	dold text/grafik
ULN, NULN	understrykning resp. ej understrykning

Argumenten kan även ges med **CHR** , se 11.1.1.

Följande ordningsföljd gäller för styrtecken:

PRINT <tecken för bakgrundsfärg> <tecken för ändring av bakgrundsfärg> <styrtecken för textfärg> "Text" <styrtecken för svart bakgrund>

Exempel

10 PRINT RED NWBG GYEL "1,6 VOV"

Resultatet blir en gul hund på röd bakgrund.

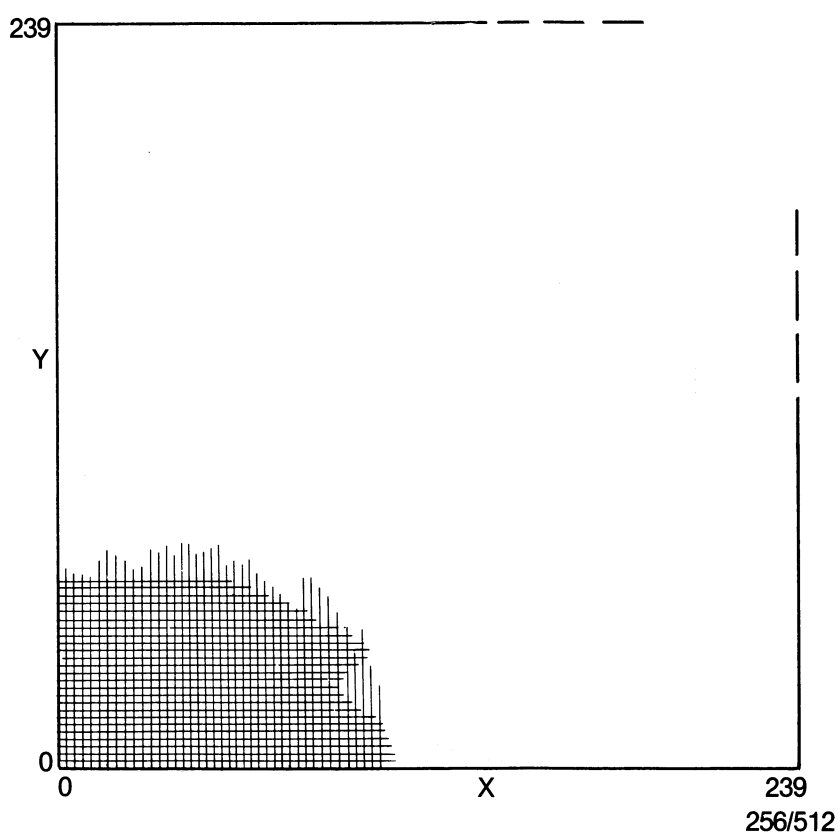
Instruktionerna **TXPOINT**, **SET DOT**, **CLR DOT** och **DOT** i ett ABC 800 program ger ingen åtgärd på bildskärmen i ABC 806. Däremot får man felmeddelande 200, se kapitel 15.

12 Högupplösningsgrafik

Högupplösningsgrafiken kan presenteras på tre olika sätt.

- Grafik enligt ABC 800 där färgerna väljs enligt tabellen under 12.4. Upplösning 240 × 240 punkter (pixels)
- Grafiken väljs fritt mellan fyra valfria färger. Text och grafik kan blandas på skärmen och ges olika prioritet. Upplösning 512 × 240 punkter.
- Grafiken väljs fritt mellan 2 × 8 färger. Text och grafik kan blandas på skärmen och ges olika prioritet. Upplösning 256 × 240 punkter.

Varje bildelement kan adresseras oberoende av de övriga med en X och en Y koordinat. Bildens origo ligger i nedre, vänstra hörnet och positionerna är numrerade från 0 till 239 alternativt 256/512 samt 0 till 239.



Bildens bredd höjdförhållande är:

	Höjd	Bredd	B/H
ABC 812/ABC 815	187 ±2	225 ±2	1.2

Vilken av moderna 256 eller 512 punkters upplösning det blir bestäms av hur många färger som väljs. Anger man fyra färger eller mindre blir det 512 × 240 punkter.

Exempel

FGCTL GRED + RED + BLU + GRN

0 1 2 3

Färgerna tilldelas därefter ordningsnummer 0—3 som används i instruktionerna **FGFILL**, **FGLINE** och **FGPAINT**.

Anges ett G framför färgen betyder det förgrundsfärg. Enligt ovan ger **FGFILL** X, Y, 0 grafiken prioritet 1 d v s den kommer att ligga framför eventuell text. Anges däremot **FGFILL** X, Y, 1 kommer grafiken att ligga bakom eventuell text.

Anges mer än fyra färger blir det 256 x 240 punkter och 0 till 15 färger kan användas.

Exempel

```
FGCTL RED + GRN + GBLU . . . GWHT
```

0 1 2 15

Med instruktionen **FGPICTURE** kan man välja vilken bild som ska visas medan datorn ritat någon annan.

Exempel

```
FGPICTURE 0, 1, 2
```

I detta exempel kommer bild 1 att visas medan bild 0 ritas. Den sista siffran anger antalet bilder.

En bild visas dock inte automatiskt utan man måste ange instruktionen **FGCTL** för att få fram bilden på skärmen. Anges ingen **FGPICTURE** blir det bild 0 som visas.

Högupplösningsgrafiken kan användas tillsammans med text eller grafik enligt kapitel 11 eftersom informationen lagras i olika minnen.

Om man inte använder högupplösningsgrafiken kan hela grafikminnet på 128 kbyte användas som datalagringsminne/RAM-floppy.

12.1 Högupplösningsgrafik på ABC 815

För att få högupplösningsgrafik på ABC 815 krävs att man i färgvalet anger att man vill ha täckande grafik e x v **FGCTL RED + GBLU** . Observera att även färgval **GBLK** ger synlig grafik på monokrom skärm.

12.2 Instruktioner

Allmänt gäller att:

- färgnummer ges med en siffra från 0 till 3 alternativt 0 till 15. I kompatibel mod med ABC 800 anges siffrans betydelse i avsnitt 12.4.
- färgnummer får utelämnas. I så fall gäller föregående färgnummer.

FGCTL

Format

FGCTL färgvalskommando (kompatibel mod)
FGCTL färg + färg

Funktion

Väljer den färgkombination som ska användas. I kompatibel mod styrs färgvalkommandot av en siffra mellan 0—255, enligt 12.4. Vid 256/512 pixels/rad definieras detta med val av färg.

Följande färger finns

Bakgrundsfärg	Förgrundsfärg	
BLK	GBLK	Svart
RED	GRED	Röd
GRN	GGRN	Grön
YEL	GYEL	Gul
BLU	GBLU	Blå
MAG	GMAG	Magenta
CYA	GCYA	Cyan
WHT	GWHT	Vit

Exempel

FGCTL 132

Kompatibel mod med färgkombination 132.

FGCTL BLK + RED + GCYA + GYEL

512 pixel per rad med upp till fyra färgparametrar.

FGCTL BLK + RED + BLU + WHT + GRED . . .

256 pixel per rad med upp till 16 färgparametrar

FGFILL

Format

FGFILL x,y [färgnr]

Funktion

Fyller en rektangel från föregående position till det läge koordinaterna x,y anger.

FGLINE

Format

FGLINE x,y [färgnr]

Funktion

Ritar en linje från föregående position till det läge, som koordinaterna, x,y anger.

FGPAINT

Format

FGPAINT x,y [,färgnummer]

Funktion

Fyller en sluten yta

FGPICTURE

Format

FGPICTURE nr 1, nr 2, antal bilder.

Funktion

Styr vilken bild som ska visas samt vilken bild som ska ritas. Detta ger möjlighet att visa en bild och uppdatera en annan. Den bild som anges som nummer 2 visas medan nummer 1 uppdateras. Bilderna numreras 0 till 3. Antalet bilder anges mellan 1—4.

Exempel

FGPICTURE 2, 3, 4

Bild 3 visas medan bild 2 uppdateras.
Fyra stycken bilder används.

OBS!

Vid skiftning till högupplösningsgrafik med färgvalscommando som är kompatibel med ABC 800 raderas grafikminnet.

FGPOINT

Format	FGPOINT x,y [färgnr].
Funktion	Sätter ett bildelement x,y på skärmen.
Exempel	FGPOINT 100, 100, 2
Observera	FGPOINT (x,y) ger färgnumret för bildelement (x,y).

FGPOINT, funktion

Format	FGPOINT (x,y)
Funktion	Ger färgnumret för bildelementet (x,y)
Exempel	FGPOINT (100, 100)

Enligt exemplet ovan blir det 2.

12.3 Exempel

Exempel 1

```
100 PRINT CHR α (12)
110 PRINT CUR (3,7) 'Detta program visar prioritet och icke prioritet
    på högupplösningsgrafik.'
120 PRINT CUR (10,0);
130 FOR I = 32 TO 127
140   PRINT CHR α (I);
150 NEXT I
160 ! Töm högupplösningsminnet
170 FGCTL BLK + GRED + BLU + GGRN
180 FGPOINT 0,0,0
190 FGFILL 511,239
200 ! Rita två rektanglar
210 FGPOINT 0,125,1
220 FGFILL 50,239
230 FGPOINT 511,239,2
240 FGFILL 450,0
250 ! Rita triangel
260 FGPOINT 200,100,2
270 FGLINE 250,105
280 FGLINE 230,200
290 FGLINE 200,100
300 ! Måla triangel
310 FGPAINT 205,101,3
```

Exempel 2

```
100 EXTEND
110 FLOAT
120 PRINT 'Vänta!!'
130 DEF FNDelay(K) LOCAL J
140   J = 0
150   WHILE J < K
160     J = J + 1
170   WEND
180   RETURN 0
190 FNEND
200 ! Rita cirkel
210 DEF FNCirkel (X,Y) LOCAL Fi,Epsylon,R
220   Fi = 0
230   R = 80
240   Epsylon = (2*PI/36)
250   FGPOINT X + 2*R,Y
260   WHILE Fi < 2*PI
270     Fi = Fi + Epsylon
280     FGLINE X + 2*(R*COS(Fi)),Y + 1.2*(R*SIN(Fi)),1
290   WEND
300   FGPAINT X,Y,2
310   RETURN 0
320 FNEND
330 ! Städa 4 sidor och rita cirkel på varje sida
340 FGCTL BLK + BLK + BLK + BLK
350 I = 0
360 WHILE I <= 3
370   FGPICTURE I,I,I + 1
380   FGPOINT 511,239,0
390   FGFILL 0,0
400   FGFILL 511,10,3
410   Z = FNCirkel(255,SQR(I)*3 + 107)
420   I = I + 1
430 WEND
440 FGCTL GBLK + GRED + GBLU + GGRN
450 ! Studsa till man trycker CTRL-C
460 WHILE -1
470   I = 3
480   ! Bollen faller
490   WHILE I >= 0
500     FGPICTURE 0,I,4
510     I = I - 1
520     Z = FNDelay(I*13)
530   WEND
540   I = 0
550   ! Bollen studsar upp
560   WHILE I <= 3
570     FGPICTURE 0,I,4
580     I = I + 1
590     Z = FNDelay(I*13)
600   WEND
610 WEND
```

12.4 Färgvalstabell

Färgvalskommandot ligger i intervallet 0 till 255.

För färgvalskommando 1—71 gäller att färg 0 ligger bakom text medan färg 1—3 ligger framför text.

Från 72—127 gäller att färg 0 och den färg som är samma som färg 0 ligger bakom text medan de två övriga färgerna ligger framför.

Från 128—255 gäller att all högupplösningsgrafik ligger framför text.

B = Blå, C = cyan, G = gul, GR = grön, M = magenta, R = röd, S = svart, V = vit.

Förvals-kommando		Färg				Förvals-kommando		Färg			
Grafik + text	Enbart grafik	0	1	2	3	Grafik + text	Enbart grafik	0	1	2	3
0	128	S	S	S	S	35	163	S	B	C	V
1	129	S	V	V	V	36	164	S	M	C	V
2	130	S	R	GR	G	37	165	R	GR	G	B
3	131	S	R	GR	B	38	166	R	GR	G	M
4	132	S	R	GR	M	39	167	R	GR	G	C
5	133	S	R	GR	C	40	168	R	GR	G	V
6	134	S	R	GR	V	41	169	R	GR	B	M
7	135	S	R	G	B	42	170	R	GR	B	C
8	136	S	R	G	M	43	171	R	GR	B	V
9	137	S	R	G	C	44	172	R	GR	M	C
10	138	S	R	G	V	45	173	R	GR	M	V
11	139	S	R	B	M	46	174	R	GR	C	V
12	140	S	R	B	C	47	175	R	G	B	M
13	141	S	R	B	V	48	176	R	G	B	C
14	142	S	R	M	C	49	177	R	G	B	V
15	143	S	R	M	V	50	178	R	G	M	C
16	144	S	R	C	V	51	179	R	G	M	V
17	145	S	GR	G	B	52	180	R	G	C	V
18	146	S	GR	G	M	53	181	R	B	M	C
19	147	S	GR	G	C	54	182	R	B	M	V
20	148	S	GR	G	V	55	183	R	B	C	V
21	149	S	GR	B	M	56	184	R	M	C	V
22	150	S	GR	B	C	57	185	GR	G	B	M
23	151	S	GR	B	V	58	186	GR	G	B	C
24	152	S	GR	M	C	59	187	GR	G	B	V
25	153	S	GR	M	V	60	188	GR	G	M	C
26	154	S	GR	S	V	61	189	GR	G	M	V
27	155	S	G	B	M	62	190	GR	G	C	V
28	156	S	G	B	C	63	191	GR	B	M	C
29	157	S	G	B	V	64	192	GR	B	M	V
30	158	S	G	M	C	65	193	GR	B	C	V
31	159	S	G	M	V	66	194	GR	M	C	V
32	160	S	G	C	V	67	195	G	B	M	C
33	161	S	B	M	C	68	196	G	B	M	V
34	162	S	B	M	V	69	197	G	B	C	V

Förvals-kommando		Färg			
Grafik + text	Enbart grafik	0	1	2	3
70	198	G	M	C	V
71	199	B	M	C	V
72	200	S	R	S	R
73	201	S	S	R	R
74	202	S	GR	S	GR
75	203	S	S	GR	GR
76	204	S	G	S	G
77	205	S	S	G	G
78	206	S	B	S	B
79	207	S	S	B	B
80	208	S	M	S	M
81	209	S	S	M	M
82	210	S	C	S	C
83	211	S	S	C	C
84	212	S	V	S	V
85	213	S	S	V	V
86	214	R	GR	R	GR
87	215	R	R	GR	GR
88	216	R	G	R	G
89	217	R	R	G	G
90	218	R	B	R	B
91	219	R	R	B	B
92	220	R	M	R	M
93	221	R	R	M	M
94	222	R	C	R	C
95	223	R	R	C	C
96	224	R	V	R	V
97	225	R	R	V	V
98	226	GR	G	GR	G
99	227	GR	GR	G	G

Förvals-kommando		Färg			
Grafik + text	Enbart grafik	0	1	2	3
100	228	GR	B	GR	B
101	229	GR	GR	B	B
102	230	GR	M	GR	M
103	231	GR	GR	M	M
104	232	GR	C	GR	C
105	233	GR	GR	C	C
106	234	GR	V	GR	V
107	235	GR	GR	V	V
108	236	G	B	G	B
109	237	G	G	B	B
110	238	G	M	G	M
111	239	G	G	M	M
112	240	G	C	G	C
113	241	G	G	C	C
114	242	G	V	G	V
115	243	G	G	V	V
116	244	B	M	B	M
117	245	B	B	M	M
118	246	B	C	B	C
119	247	B	B	C	C
120	248	B	V	B	V
121	249	B	B	V	V
122	250	M	C	M	C
123	251	M	M	C	C
124	252	M	V	M	V
125	253	M	M	V	V
126	254	C	V	C	V
127	255	C	C	V	V

13 Funktionstangenter

Datorn är utrustad med speciella funktionskoder. Dessa genereras på olika sätt beroende på vilket tangentbord som används. Av tabellen nedan framgår vilka tangenttryckningar som ska göras på de olika tangentborden för att generera koderna. Totalt är det 32 olika koder som kan användas för olika fasta funktioner i program. Det kan t.ex. vara markörförflyttning, sidbyte eller hopp till programmodul.

ABC 77 och ABC 22	Värde	ABC 55
PF1	192	CTRL + 1
PF2	193	CTRL + 2
PF3	194	CTRL + 3
PF4	195	CTRL + 4
PF5	196	CTRL + 5
PF6	197	CTRL + 6
PF7	198	CTRL + 7
PF8	199	CTRL + 8
SHIFT + PF1	208	CTRL + SHIFT + 1
SHIFT + PF2	209	CTRL + SHIFT + 2
SHIFT + PF3	210	CTRL + SHIFT + 3
SHIFT + PF4	211	CTRL + SHIFT + 4
SHIFT + PF5	212	CTRL + SHIFT + 5
SHIFT + PF6	213	CTRL + SHIFT + 6
SHIFT + PF7	214	CTRL + SHIFT + 7
SHIFT + PF8	215	CTRL + SHIFT + 8
CTRL + PF1	224	CTRL + SHIFT + Q
CTRL + PF2	225	CTRL + SHIFT + W
CTRL + PF3	226	CTRL + SHIFT + E
CTRL + PF4	227	CTRL + SHIFT + R
CTRL + PF5	228	CTRL + SHIFT + T
CTRL + PF6	229	CTRL + SHIFT + Y
CTRL + PF7	230	CTRL + SHIFT + U
CTRL + PF8	231	CTRL + SHIFT + I
SHIFT + CTRL + PF1	240	CTRL + SHIFT + A
SHIFT + CTRL + PF2	241	CTRL + SHIFT + S
SHIFT + CTRL + PF3	242	CTRL + SHIFT + D
SHIFT + CTRL + PF4	243	CTRL + SHIFT + F
SHIFT + CTRL + PF5	244	CTRL + SHIFT + G
SHIFT + CTRL + PF6	245	CTRL + SHIFT + H
SHIFT + CTRL + PF7	246	CTRL + SHIFT + J
SHIFT + CTRL + PF8	247	CTRL + SHIFT + K

Exempel för ABC 55:

PF1 till PF8 fås med samtidigt tryck på CTRL + 1 8

SHIFT + PF1—PF8 fås med tryck på CTRL + SHIFT + 1 8

CTRL + PF1—PF8 fås medtryck på CTRL + SHIFT + Q I

SHIFT + CTRL + PF1—PF8 fås med tryck på CTRL + SHIFT + A K

En tryckning på en funktionstangent kan leda till ett anrop av ett underprogram.

Exempel:

```
10 ON ERROR GOTO 100
20 INPUT "Tal",P(I)
30 I=I+1
40 GOTO 20
|
|
|
|
100 IF ERRCODE <>53 THEN RESUME
110 A=SYS(6)
120 GET X␣
130 ON ASC(X␣)-191 RESUME 400, 500, 600
```

När en funktionstangent trycks ned vid **INPUT** eller **INPUT LINE**, genereras ett fel med **ERRCODE=53**. Programmet bör alltså innehålla en rutin, som tar hand om fel 53. Genom att använda funktionen **SYS(6)** och därefter läsa tecknet med **GET** kan man ta reda på vilken funktionstangent, som har tryckts ned.

14 Skillnader ABC 806—ABC 800

Ett program som är skrivet på ABC 806 och som använder någon av de nya instruktionerna t.ex. **ATTRIBUTE** kan inte laddas på ABC 800.

Följande instruktioner är nya:

ATTRIBUTE
FGCTL färg + färg . . .
FGPICTURE
WIDTH

Instruktionerna **TXPOINT**, **SET DOT**, **CLR DOT** och **DOT** i ett ABC 800 program ger ingen åtgärd på bildskärmen i ABC 806. Däremot får man felmeddelande 200, se kapitel 15.

15 Felmeddelanden

Fel 19— 68: I/O-fel
 Fel 120—129: ISAM fel
 Fel 130—176: Fel vid programkörning
 Fel 180—191: Logiska fel
 Fel 200—211: Allmänna fel
 Fel 220—234: Formella BASIC-fel

Fel	Meddelande	Kommentar
19	Kan ej öppna fler filer	Sju filer är öppnade
20	För lång rad (>160 tecken)	En rad får innehålla max. 160 tecken
21	Hittar ej filen	Filen finns inte eller har sökts under fel namn
32	Filen ej öppnad	
34	Slut på filen	Försökt läsa efter filslut
35	Checksummafel vid läsning	Skivan är skadad
36	Checksummafel vid skrivning	Skivan är skadad
37	Felaktigt sektorformat	Fel på skiva
38	Sektornummer utanför filen	Försök att läsa längre än filen medger
39	Filen skrivskyddad	
40	Filen raderskyddad	
41	Skivan full	Filen får ej plats på skivan
42	Enheten ej klar	Ingen flexskiva isatt eller luckan öppen
43	Skivan skrivskyddad	
44	Logisk fil ej öppnad	
45	Fel logiskt filnummer	
46	Fel enhetsnummer	
47	Fel trapnummer	
48	Fel i biblioteket	
49	Felaktigt fysiskt filnummer	Fel på skivan
51	Enheten upptagen	
52	Ej till denna enhet	
53	Funktionstangent	Funktionstangent har tryckts ned i INPUT- eller INPUT LINE- sats
54	IEC både sändare och mottagare	IEC-option
55	IEC-mottagare ej aktiv	IEC-option
56	IEC-sändare ej aktiv	IEC-option
57	Tecken från tangentbord ej i tid	
58	Ogiltigt tecken inläst	
64	Felaktigt "NAME"	Nya filnamnet existerar redan
68	Felaktig tidsspecifikation	
120	Nyckeln finns ej	ISAM option
121	Dubblatnyckel	ISAM option
122	Felaktig nyckel	ISAM option
123	Fel vid kontrolläsning	ISAM option
124	Index finns ej	ISAM option
125	Felaktig postlängd	ISAM option
126	Fel ISAM fil version	ISAM option
127	Reserverad kod	ISAM option
128	Slut på minnet i centralen	ISAM option
129	Reserverad kod	ISAM option
130	För stort flyttal	

Fel	Meddelande	Kommentar
131	Index utanför tillåtet område	Försök att använda index större än motsvarande DIM
132	För stort heltal	
133	Fel i ASCII-aritmetiskt uttryck	
134	Index utanför strängen	Index för stort eller negativt
135	Negativ " SPACE ", " STRING " eller " TAB " < 1	
136	För lång sträng	För liten dimension på den mottagande strängen
137	Ej tillåtet öka " DIM "	Ett fält får inte ökas utöver sin ursprungliga längd
138	Fel värde i " ON "-uttryck	
139	" RETURN " utan " GOSUB "	En RETURN -sats påträffad utan att en föregående GOSUB -sats har blivit utförd
140	Felaktigt " RETURN "-variabel	
141	Data slut	Datalistan har blivit tömd och en READ -sats efterfrågade ytterligare data
142	Felaktigt argument i funktion	
143	Felaktig " SYS "-funktion	
144	Ej tillåten rad	
145	" FNEND " utan föregående " RETURN "	
146	" PRINT USING "-fel	Felaktigt format i PRINT USING -sats
147	Felaktiga data	
148	För lite indata	För få data inmatade vid INPUT
149	" RESTORE " ej på en " DATA "-rad	
150	För mycket indata	För många data inmatade vid INPUT
151	" RESUME " utan fel	
176	Grafisk punkt utanför bildskärmen	
180	Hittar ej detta radnummer	Referens till ett radnummer som inte finns i programmet
181	Felaktigt in hopp i funktion	
182	" NEXT " eller " WEND " saknas	
183	" FOR " eller " WHILE " saknas	
184	Fel variabel efter " NEXT "	
185	Blandade " FOR "-loopar med samma variabel	
186	" FOR "-loop med lokal variabel ej tillåtet	Gäller i flerradiga funktioner
187	Funktion ej definierad	Anrop till ej definierad funktion
188	Flera funktioner med samma namn	
189	Felaktig funktion	Ej tillåtet att blanda flera DEF
190	Fel antal index	Antalet index överensstämmer ej med DIM
191	Ej tilldelningsbar i funktion	Funktionens argument är ej tilldelningsbart i funktionen
200	Enheten ej ansluten	
201	Minnet fullt	Datorns primärminne har ej plats för program och data
202	" LIST "-skyddat program	

Fel	Meddelande	Kommentar
203	Fel programformat	Programmet är sparad under en icke kompatibel BASIC- version
204	" MERGE " går ej på "BAC"-fil	
205	" COMMON "-fel	
206	Använd kommandot " RUN "	
207	Kan ej fortsätta	Gäller GOTO radnr och CON
208	Otillåtet som kommando	Instruktionen kan ej användas som kommando
209	Fel data till kommando	Felaktigt argument till kommandot, t.ex. LIST ##
210	Felaktigt tal	Talet innehåller tecken som inte är siffror
211	Precision får ej ändras	Ej tillåtet ändra precision efter tilldelning av variabel
220	Förstår ej	Formellt BASIC-fel
221	Otillåtet tecken efter satsen	Formellt BASIC-fel. Datorn förväntade sig Return, kolon (:) eller utropstecken (!)
222	Måste vara först på en rad	
223	Fel antal eller typ av argument	
224	Otillåten blandning av tal och strängar	
225	Ej enkel variabel	Ej tillåtet ha index på variabel t.ex. i FOR -loop
226	Felaktig sats efter " ON "	Formellt BASIC-fel
227	"," saknas	Formellt BASIC-fel
228	"=" saknas	Formellt BASIC-fel
229	")" saknas	Formellt BASIC-fel
230	" AS FILE " saknas	Förekommer i OPEN - och PRE-PARE -satser
231	" AS " saknas	Fel i NAME AS
232	" TO " saknas	Förekommer i FOR -loopar
233	Radnummer saknas	
234	Felaktig variabel	

16 Kommando- och instruktions-sammanfattning

ABS	(funktion)	Sid. 63
Format	ABS (argument)	
Funktion	Ger absolutvärdet av argumentet	
ADD	(funktion)	Sid. 67
Format	ADD (A, B, P%)	
Funktion	Adderar värdet av strängarna A och B. Svar med P% decimaler.	
ASCII	(funktion)	Sid. 68
Format	ASCII (A)	
Funktion	Ger ASCII-värdet för första tecknet i A.	
ATN	(funktion)	Sid. 63
Format	ATN (argument)	
Funktion	Arcustangens för argumentet i radianer.	
ATTRIBUTE	(instruktion)	Sid. 82
Format	ATTRIBUTE n	
Funktion	Anger var styrtecken ska lagras samt soft-scroll	
Observera	Anges inget ATTRIBUTE sker lagringen i teckenminnet.	
AUTO	(kommando)	Sid. 25
Format	AUTO [argument 1 [,argument 2]]	
	Där < argument 1 > anger första radnummer, som ska skrivas och < argument 2 > anger radintervall.	
Funktion	Automatisk radnumrering.	
↵BAS	(kommando under DOS)	Sid. 26
Format	↵BAS	
Funktion	Övergång till BASIC	
BYE	(kommando)	Sid. 26
Format	BYE	
Funktion	Övergång till DOS	
CALL	(funktion)	Sid. 73
Format	CALL (A[,D%])	
Funktion	Anrop av assemblerprogram.	
VARNING	Man kan förstöra en körning, om CALL används på fel sätt.	
CHAIN	(instruktion)	Sid. 35
Format	CHAIN "filnamn.typ"/strängvariabel	
Funktion	Laddar och startar exekveringen av ett program.	

CHR α	(funktion)	Sid. 68
Format	CHR α (argument[,argument,...])	
Funktion	Ger teckensträng som motsvarar argumentens ASCII-värden.	
CLEAR	(kommando)	Sid. 26
Format	CLEAR	
Funktion	Nollställer alla variabler och stänger alla öppna filer.	
CLOSE	(instruktion)	Sid. 35
Format	CLOSE [filnummer, ...]	
Funktion	Stänger angiven fil.	
COMMON	(instruktion)	Sid. 35
Format	COMMON variabel[(n,...)] [,variabel, ...] COMMON strängvariabel[(n,...)]=längd [,strängvariabel, = längd,...]	
Funktion	Deklaration av de variabler, vars värden ska överföras till ett annat program.	
COMP%	(funktion)	Sid. 68
Format	COMP% (A α ,B α)	
Funktion	Jämför två numeriska strängar.	
CON	(kommando)	Sid. 26
Format	CON (eller CONTINUE)	
Funktion	Fortsätter programexekvering.	
COS	(funktion)	Sid. 63
Format	COS (argument)	
Funktion	Ger cosinus för argumentet (argumentet i radianer).	
CUR	(funktion)	Sid. 73
Format	CUR (R%,N%)	
Funktion	Flyttar markören till rad R%, (0–23) position N% (0–39/79).	
CVT	(funktion)	Sid. 73
Format	CVT α α (heltalsvariabel) CVT α %(strängvariabel) CVTF α (flyttalsvariabel) CVT α F(strängvariabel)	
Funktion	Överför tal till strängar respektive återskapar talen.	
DATA	(instruktion)	Sid. 36
Format	DATA värde [,värde, ...]	
Funktion	Utgör tillsammans med READ , ett sätt att tilldela variabler värden.	
DEF	(instruktion)	Sid. 36
Format	Enradig funktion: DEF FN identifierare[(argument)]=funktion Flerradig funktion: DEF FN identifierare [%/ α] [(argument)] [LOCAL variabel [,variabel, ...]]	
Funktion	Definierar enradiga och flerradiga funktioner.	

DIGITS	(instruktion)	Sid. 38
Format	DIGITS antal siffror	
Funktion	Ger antal siffror för utskrift.	
DIM	(instruktion)	Sid. 39
Format	DIM variabel(n)[, variabel(n,...), ...] DIM strängvariabel[(n,...)] [=uttryck]	
Funktion	Reserverar utrymme för strängar och vektorer.	
DIVα	(funktion)	Sid. 68
Format	DIVα (A α ,B α ,P%)	
Funktion	Ger kvoten A α /B α avrundad till P% decimaler.	
DOUBLE	(instruktion)	Sid. 40
Format	DOUBLE	
Funktion	Flyttal får dubbel precision (16 siffror).	
ED	(kommando)	Sid. 26
Format	ED [radnummer]	
Funktion	Inleder ändringar i program.	
END	(instruktion)	Sid. 40
Format	END	
Funktion	Avslutar programmet.	
ERASE	(kommando)	Sid. 27
Format	ERASE radnummer I [- radnummer II] ERASE radnummer - ERASE - radnummer	
Funktion	Raderar en eller flera programrader.	
ERRCODE	(funktion)	Sid. 74
Format	ERRCODE	
Funktion	Ger värdet av senast genererade felkod.	
EXP	(funktion)	Sid. 63
Format	EXP (argument)	
Funktion	Ger e**argument.	
EXTEND	(instruktion)	Sid. 40
Format	EXTEND	
Funktion	Tillåter långa variabelnamn.	
FGCTL	(instruktion — högupplösningsgrafik)	Sid. 85
Format	FGCTL färgvalskommando alt. färg + färg . . .	
Funktion	Väljer färgkombination.	
FGFILL	(instruktion — högupplösningsgrafik)	Sid. 86
Format	FGFILL x,y [,färgnummer]	
Funktion	Fyller en rektangel från föregående läge till det läge koordinaterna x,y, anger, x = 0—239 alt. 0—256/512, y = 0—239.	
Observera	Origo är i nedre, vänstra hörnet.	

FGLINE	(instruktion — högupplösningsgrafik)	Sid. 86
Format	FGLINE x,y[,färgnummer]	
Funktion	Ritar en linje från föregående position till det läge, som koordinaterna x,y anger, x = 0—239 alt. 0—256/512, y = 0—239.	
Observera	Origo är i nedre, vänstra hörnet.	
FGPAINT	(instruktion — högupplösningsgrafik)	Sid. 86
Format	FGPAINT x,y[,färgnummer]	
Funktion	Fyller en sluten yta. x = 0—239 alt. 0—256/512, y = 0—239.	
Observera	Origo är i nedre, vänstra hörnet.	
FGPICTURE	(instruktion — högupplösningsgrafik)	Sid. 86
Format	FGPICTURE nr 1, nr 2, antal bilder	
Funktion	Anger vilken bild som ska visas när en annan uppdateras	
Observera	Anges inget FGPICTURE visas bild 0	
FGPOINT	(instruktion — högupplösningsgrafik)	Sid. 87
Format	FGPOINT x,y[,färgnummer]	
Funktion	Sätter ett bildelement på den position som koordinaterna x,y anger, x = 0—239 alt. 0—256/512, y = 0—239.	
Observera	Origo är i nedre, vänstra hörnet.	
FGPOINT	(funktion-högupplösningsgrafik)	Sid. 87
Format	FGPOINT (x,y)	
Funktion	Ger färgnumret för bildelementet i den position som koordinaterna x,y anger, x = 0—239 alt. 0—256/512, y = 0—239.	
Observera	Origo är i nedre, vänstra hörnet.	
FIX	(funktion)	Sid. 63
Format	FIX (argument)	
Funktion	Ger trunkerat värde av argumentet.	
Observera	Jämför INT (X).	
FLOAT	(instruktion)	Sid. 40
Format	FLOAT	
Funktion	Alla variabler antas vara flyttade om inget annat anges.	
FN	(funktion)	Sid. 74
Format	FN identifierare [%/α] ((parameter [,parameter,...])	
Funktion	Anrop av användardefinierad funktion.	
Observera	Jämför DEF FN .	
FNEND	(instruktion)	Sid. 38
Format	FNEND	
Funktion	Avslutar flerradig funktion.	
FOR	(instruktion)	Sid. 41
Format	FOR variabel=uttryck TO uttryck [STEP intervall]	
Funktion	Inleder programloop.	
GET	(instruktion)	Sid. 42
Format	GET strängvariabel	
Funktion	Läser ett tecken från tangentbordet.	
GET #	(instruktion)	Sid. 43
Format	GET # filnummer, strängvariabel [COUNT antal tecken]	
Funktion	Läser från fil.	

GOSUB	(instruktion)	Sid. 43
Format	GOSUB radnummer	
Funktion	Ger ovillkorligt hopp till subrutin.	
GOTO	(instruktion)	Sid. 44
Format	GOTO radnummer	
Funktion	Ger ovillkorligt hopp till angivet radnummer.	
HEX	(funktion)	Sid. 64
Format	HEX (argument)	
Funktion	Omvandlar decimalt tal till hexadecimal sträng.	
IF - THEN - ELSE	(instruktion)	Sid. 44
Format	IF villkor THEN sats[er]/radnr [ELSE sats[er]/radnr]	
Funktion	Villkorlig styrning av ordningsföljden mellan programraderna.	
INP	(funktion)	Sid. 74
Format	INP (I%)	
Funktion	Ger datavärdet från inport I%.	
WARNING	Detta är en maskinorienterad funktion, som ska användas enbart vid avancerad programmering.	
INPUT	(instruktion)	Sid. 45
Format	INPUT [#filnr/"ledtext"] variabel [,variabel, ...]	
Funktion	Hämtar in data till det program, som körs.	
INPUT LINE	(instruktion)	Sid. 46
Format	INPUT LINE [#filnummer,]strängvariabel	
Funktion	Tar emot inmatning av en rad tecken.	
INSTR	(funktion)	Sid. 69
Format	INSTR (N%,A,B)	
Funktion	Söker efter strängen B i A med början i position N%.	
INT	(funktion)	Sid. 64
Format	INT (argument)	
Funktion	Ger värdet av det största heltal, som är mindre än eller lika med argumentet. Jämför FIX .	
INTEGER	(instruktion)	Sid. 46
Format	INTEGER	
Funktion	Alla variabler antas vara heltalsvariabler, om inget annat anges.	
KILL	(instruktion)	Sid. 47
Format	KILL "[enhet:]filnamn.typ"	
Funktion	Den angivna filen raderas från det yttre minnet.	
LEFT	(funktion)	Sid. 69
Format	LEFT (A)(I%)	
Funktion	Ger de I% första tecknen av strängen A.	

LEN	(funktion)	Sid. 69
Format	LEN (A α)	
Funktion	Ger antalet tecken i strängen A α , d.v.s. stränglängden (inklusive mellanslag).	
LET	(instruktion)	Sid. 47
Format	[LET] variabel=uttryck	
Funktion	Tilldelar en variabel ett värde.	
LIST	(kommando)	Sid. 28
Format	LIST [enhet:]filnamn[.typ] LIST [radnummer [-radnummer]] LIST radnummer - LIST - radnummer LIST enhet: [radnummer - radnummer]	
Funktion	Listar hela eller del av program.	
LOAD	(kommando)	Sid. 29
Format	LOAD [enhet:]filnamn[.typ]	
Funktion	Laddar in ett BASIC-program.	
LOG	(funktion)	Sid. 64
Format	LOG (argument)	
Funktion	Ger naturliga logaritmen för argumentet.	
LOG10	(funktion)	Sid. 65
Format	LOG10 (argument)	
Funktion	Ger tiologaritmen för argumentet.	
MERGE	(kommando)	Sid. 29
Format	MERGE [enhet:]filnamn[.typ]	
Funktion	Länkar BAS-filer.	
MID	(instruktion)	Sid. 69
Format	MID [α](A α ,P%,K%)	
Funktion	Tillordnar tecken nr P% t.o.m. P%+K%-1 av A α ett nytt värde, d.v.s. byter tecken i strängen.	
MID	(funktion)	Sid. 70
Format	MID [α](A α ,P%,K%)	
Funktion	Ger den delsträng av A α , som börjar i position P% och är K% tecken lång, d.v.s. tecknen nr P% t.o.m. P%+K%-1.	
MOD	(funktion)	Sid. 65
Format	MOD (argument 1, argument 2)	
Funktion	Ger resten vid heltalsdivision av argumenten..	
MULα	(funktion)	Sid. 70
Format	MULα (A α ,B α ,P%)	
Funktion	Ger produkten A α * B α med P% decimaler.	

NAME	(instruktion)	Sid. 48
Format	NAME "[enhet:]filnamn1.typ" AS "filnamn2.typ"	
Funktion	Ändrar namnet på en fil.	
NEW	(kommando)	Sid. 30
Format	NEW	
Funktion	Raderar innehållet i minnet.	
Observera	Kommandot SCR (scratch) kan också användas.	
NEXT	(instruktion)	Sid. 42
Format	NEXT variabel	
Funktion	NEXT anger slutet på en programloop, som inletts med en FOR -sats.	
NO EXTEND	(instruktion)	Sid. 49
Format	NO EXTEND	
Funktion	Avslutar arbete i EXTEND -mod.	
NO TRACE	(instruktion)	Sid. 48
Format	NO TRACE	
Funktion	Avslutar den utskrift av radnummer, som begärts med instruktionen TRACE .	
NUM ⌘	(funktion)	Sid. 70
Format	NUM ⌘(argument)	
Funktion	Ger den numeriska sträng, som motsvarar argumentet.	
OCT ⌘	(funktion)	Sid. 65
Format	OCT ⌘ (argument)	
Funktion	Omvandlar decimalt tal till oktal sträng.	
ON ERROR		
GOTO	(instruktion)	Sid. 49
Format	ON ERROR GOTO [radnummer]	
Funktion	Ger hopp till angivet radnummer vid fel.	
ON - GOSUB	(instruktion)	Sid. 49
Format	ON uttryck GOSUB radnummer[,radnummer,...]	
Funktion	Används för att villkorligt hoppa till en av flera subrutiner eller en av flera ingångar i en subrutin.	
ON - GOTO	(instruktion)	Sid. 50
Format	ON uttryck GOTO radnummer[,radnummer,...]	
Funktion	För att hoppa till en av flera rader, beroende på uttryckets värde.	
ON -		
RESTORE	(instruktion)	Sid. 50
Format	ON uttryck RESTORE radnummer [radnummer,...]	
Funktion	Ställer DATA -pekaren efter samma urvalsförfarande som ON - GOTO .	

ON - RESUME	(instruktion)	Sid. 50
Format	ON uttryck RESUME radnummer[,radnummer,...]	
Funktion	För att hoppa till en av flera rader, beroende på uttryckets värde, då satsen utförs. Felhantering återställs. Används tillsammans med ON ERROR GOTO .	
OPEN	(instruktion)	Sid. 51
Format	OPEN "[enhet:][filnamn[.typ]]" AS FILE filnummer	
Funktion	Används för att öppna en befintlig fil.	
OPTION BASE	(instruktion)	Sid. 52
Format	OPTION BASE n	
Funktion	Anger lägsta värde för vektorindex (n = 0 eller 1).	
OUT	(instruktion)	Sid. 75
Format	OUT port,data [,port,data, ...]	
Funktion	Används för att adressera utportar vid utmatning av data.	
WARNING	Detta är en maskinorienterad instruktion för avancerad programmering. Denna instruktion och instruktionen INP ger användaren tillgång till I/O-funktionerna i ABC 806 och dess I/O-buss.	
PEEK	(funktion)	Sid. 75
Format	PEEK (I%)	
Funktion	Ger innehållet i adress I%.	
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.	
PEEK2	(funktion)	Sid. 75
Format	PEEK2 (B%)	
Funktion	Läser innehållet i två bytes.	
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.	
PI	(funktion)	Sid. 65
Format	PI	
Funktion	Konstant med värdet 3.14159 (enkel precision)	
POKE	(funktion)	Sid. 75
Format	POKE adress,data (,data, ...)	
Funktion	Används för att ladda in ett värde i en minnescell.	
WARNING	Denna instruktion är maskinorienterad och ska endast användas vid avancerad programmering. Om den används på fel sätt, kan innehållet i minnet förstöras.	
POSIT	(instruktion)	Sid. 52
Format	POSIT # filnummer, position	
Funktion	Positionerar filpekare.	
PREPARE	(instruktion)	Sid. 53
Format	PREPARE "[enhet:][filnamn.typ]" AS FILE filnummer	
Funktion	Skapar och öppnar en ny fil.	

PRINT	(instruktion)	Sid. 53
Format	PRINT [# filnummer,] "data"/variabel (,"data"/variabel, ...)	
Funktion	Matar ut data i ASCII-form.	
PRINT	(instruktion - färgval m.m.)	Sid. 82
Format	PRINT [CUR (R,K)]argument [;argument;...]"text"	
Funktion	Används för att skriva text och grafik. Argumenten styr färgval m.m. Bilden ritas med början på rad R (0-23), position K (0-39/79).	
PRINT USING	(instruktion)	Sid. 54
Format	PRINT [# filnummer] USING "formatsträng";"data"/variabel [,"data"/variabel, ...]	
Funktion	Skriver tal och strängar med angivet format.	
PUT	(instruktion)	Sid. 57
Format	PUT #filnummer, strängvariabel	
Funktion	Skriver en strängvariabel.	
RANDOMIZE	(instruktion)	Sid. 57
Format	RANDOMIZE	
Funktion	Sätter ett slumpmässigt startvärde för funktionen RND (slumptalsgeneratorn).	
VARNING	Bör bara användas en gång i varje program.	
READ	(instruktion)	Sid. 57
Format	READ variabel[, variabel, ...]	
Funktion	Utgör, tillsammans med DATA -satser, ett sätt att tilldela variabler värden.	
REM	(instruktion)	Sid. 58
Format	REM text	
Funktion	Inleder kommentar i program.	
Observera	REM kan bytas mot !	
RENUMBER	(kommando)	Sid. 30
REN		
Format	REN [UMBER] [radnummer[,intervall[,fr.o.m. rad -t.o.m. rad]]]	
Funktion	Ändrar radnumreringen i det aktuella programmet.	
RESTORE	(instruktion)	Sid. 58
Format	RESTORE [radnummer]	
Funktion	Gör det möjligt att använda innehållet i DATA -satser flera gånger.	
RESUME	(instruktion)	Sid. 59
Format	RESUME [radnummer]	
Funktion	Återhopp från felhanteringsrutin.	

RETURN	(instruktion)	Sid. 59
Format	RETURN [variabel]	
Funktion	Ger återhopp från subrutin eller flerradiga funktioner.	
RIGHT	(funktion)	Sid. 70
Format	RIGHT [α](A α ,N%)	
Funktion	Ger de sista tecknen i A α , fr.o.m position N%.	
RND	(funktion)	Sid. 65
Format	RND	
Funktion	Ger ett slumpstal mellan 0 och 0.999999.	
RUN	(kommando)	Sid. 31
Format	RUN [[enhet:]filnamn[.typ]]	
Funktion	Laddar och exekverar ett BASIC-program eller exekverar (kör) programmet i datorns minne.	
SAVE	(kommando)	Sid. 32
Format	SAVE [enhet:]filnamn[.typ]	
Funktion	Skapar en programfil på ett yttre minne och lagrar programmet, som finns i datorns arbetsminne, i denna fil.	
SGN	(funktion)	Sid. 66
Format	SGN (argument)	
Funktion	Funktionen SGN (X) har värdet +1 om X är positivt, 0 om X är 0 och -1 om X är negativt.	
SIN	(funktion)	Sid. 66
Format	SIN (argument)	
Funktion	Ger sinus för argumentet (argumentet i radianer).	
SINGLE	(instruktion)	Sid. 59
Format	SINGLE	
Funktion	Ändrar alla variabler och uttryck, som är flyttal, till enkel precision (7 siffror).	
Observera	Det är inte tillåtet att blanda SINGLE och DOUBLE i samma program.	
SPACEα	(funktion)	Sid. 71
Format	SPACE α (N%)	
Funktion	Ger en sträng, som består av N% mellanslag.	
SQR	(funktion)	Sid. 66
Format	SQR (argument)	
Funktion	Ger kvadratroten ur argumentet.	
STOP	(instruktion)	Sid. 60
Format	STOP	
Funktion	Stoppas programexekveringen.	
Observera	Programexekveringen kan fortsättas med något av kommandona CON eller GOTO .	

STRING α	(funktion)	Sid. 71
Format	STRING α (I%,K%)	
Funktion	Ger en sträng med I% st tecken, som har ASCII-värdet K%.	
SUB α	(funktion)	Sid. 71
Format	SUB α (A α ,B α ,P%)	
Funktion	Ger den aritmetiska differensen A α -B α med P% decimaler.	
SWAP %	(funktion)	Sid. 76
Format	SWAP %(N%)	
Funktion	Första och andra byten av N% skiftar plats.	
SYS	(funktion)	Sid. 76
Format	SYS (I%)	
Funktion	Ger systemstatus enligt följande: SYS (2) Totalt minnesutrymme SYS (3) Programmets storlek SYS (4) Kvarvarande minnesutrymme SYS (5) Tangentbordsflaggan SYS (6) Lägger tillbaka sist inlästa tecken i tangentbordsbufferten. SYS (8) Är -1 då någon tangent är nertryckt. SYS (11) Programmets startadress SYS (12) Variabelrot	
TAB	(funktion)	Sid. 76
Format	TAB (I%)	
Funktion	Flyttar skrivhuvud eller markör till position I% på raden.	
TAN	(funktion)	Sid. 66
Format	TAN (argument)	
Funktion	Ger tangens för argumentet (argumentet i radianer).	
TIME α	(funktion)	Sid. 76
Format	TIME α	
Funktion	Ger tidsangivelse år-mån-dag tim.min.sek	
TRACE	(instruktion)	Sid. 60
Format	TRACE [# filnummer]	
Funktion	Skriver ut radnumren på de programrader, som genomlöps.	
UNSAVE	(kommando)	Sid. 32
Format	UNSAVE [enhet:]filnamn[.typ]	
Funktion	Raderar en fil på skiva.	
VAL	(funktion)	Sid. 71
Format	VAL (A α)	
Funktion	Beräknar numeriska värdet av den numeriska strängen A α .	

VAROOT	(funktion)	Sid. 77
Format	VAROOT (variabel)	
Funktion	Ger adressen till en tabell, som innehåller uppgifter om en variabel.	
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.	
VARPTR	(funktion)	Sid. 77
Format	VARPTR (variabel)	
Funktion	Ger adressen till en variabels värde.	
WARNING	Denna funktion är avsedd att användas vid avancerad programmering.	
WEND	(instruktion)	Sid. 60
Format	WEND	
Funktion	WEND avslutar en loop, som inleds av WHILE .	
WHILE	(instruktion)	Sid. 61
Format	WHILE uttryck	
Funktion	Anger villkor för uthopp ur en programloop.	
WIDTH	(instruktion)	Sid. 61
Format	WIDTH [# filnummer] antal	
Funktion	Anger antal tecken per rad.	
Aα+Bα	(funktion)	Sid. 72
Format	A α +B α	
Funktion	Sammankedjar (konkatenerar) två strängar.	

17 Litteraturförteckning

- "BASIC II-boken" av Jan Lundgren och Sören Thornell.
- "Mikrodatorns ABC" av Gunnar Markesjö.
Beskriver hur ABC 80 fungerar.
- "Styr och mät med smådatorer" av Åke Westh.
- "ABC om programmering och dokumentation" av Jan Lundgren och Bengt Lundin.
- "Bygg ut ABC 80 med Databoard 4680". SATTCO AB.
- "Att programmera ABC 80" av Lennart Rodhe.
- "ABC om användardokumentation". Luxor.
- "Lärobok i PASCAL" av Anders Haraldsson.
- "Datoranvändning med IEC-buss" av Sune Windisch.
- "Vår elektroniska framtid" av B-G Wennersten.
- "Privatdatorn — din egen dator" av B-G Wennersten.
- "Z80, Technical Manual". Zilog.
- "Z80, Programming Manual". Zilog.
- "Dataordboken". SIS handbok 142.
- "Bit för bit med ABC 800". Luxor.
Detta kompendium vänder sig till dem som redan kan en del om datorer och vill veta hur man utnyttjar ABC 800 datorer fullt ut.

18 Bilagor

Bilaga 1

I/O-portar

Krets	Adress 7	bit 0	Funktion	Ut Decimalt	In
ABC-buss	000XX000		Inport 0		0
	000XX001		Inport 1		1
	000XX010		Inport 2		2
	000XX111		I/O RESET		7
	000XX000		Utport 0	0	
	000XX001		Utport 1	1	
	000XX010		Utport 2	2	
	000XX011		Utport 3	3	
	000XX100		Utport 4	4	
	000XX101		Utport 5	5	
HR-grafik	000XX110		HRC	6	
	000XX111		HRS	7	
ABC-buss			XOUTSTB, XINSTB	0—31	
DART	0010XX00		Skrivare data CH.A	32	32
	0010XX01		Skrivare kontroll CH.A	33	33
	0010XX10		Tangentbord data	34	34
	0010XX11		Tangentbord kontroll	35	35
CRTC VIDEO	00110XX1		Läsregister		49
	00110100		Minnesblocktabell	52	52
	00110101		Attribut	53	53
	00110111		Synk. fördröjning	55	
CRTC	00110111		FGCTL PROM + CLOCK		55
	00111XX0		Skriv reg. adress	56	
	00111XX1		Skriv register	57	
SIO/2	010XXX00		V 24 data CH.B	64	64
	010XXX01		V 24 kontroll CH.B	65	65
CTC	011XXX00		Kanal 0	96	96
	011XXX01		Kanal 1	97	97
	011XXX10		Kanal 2	98	98
	011XXX11		Kanal 3	99	99
ABC-buss	1XXXXXXX		XOUTSTB, XINSTB	128—255	

Övriga ej redovisade adresser används ej.

X = godtyckligt värde

Bilaga 2

Minnesdisposition

Minneskarta med flexskiveenhet ansluten

Decimal adress		Hexadecimal adress	Oktaal adress
	64 kbyte Grafik/dataminne		
	64 kbyte Grafik/dataminne		
65 535	Enkla variabler	FFFFH	377:377
65280		FF00H	377:000
	Systemvariabler		
64768		FD00H	375:000
64512	DOSBUF 7	FC00H	374:000
64256	DOSBUF 6	FB00H	373:000
64000	DOSBUF 5	FA00H	372:000
63744	DOSBUF 4	F900H	371:000
63488	DOSBUF 3	F800H	370:000
63232	DOSBUF 2	F700H	367:000
62976	DOSBUF 1	F600H	366:000
62720	DOSBUF 0	F500H	365:000
	32 kbyte RAM Arbetsminne		
32768	2 kbyte RAM Teckenminne	8000H	200:000
31744		7C00H	174:000
30720		7800H	170:000
	2 kbyte PROM Printer/Terminal		
28678		7000H	160:000
	4 kbyte PROM DOS		
24576		6000H	140:000
	24 kbyte PROM BASIC II		

Teckenminnet 2 kbyte ligger parallellt med systemprogrammet för högupplösning. De båda minnesareorna inkräktar dock inte på varandra utan ABC 806 går över i en specialmod då högupplösningsminnet adresseras.

Vid laddning av operativsystemet CP/M används delar av grafikminnet.

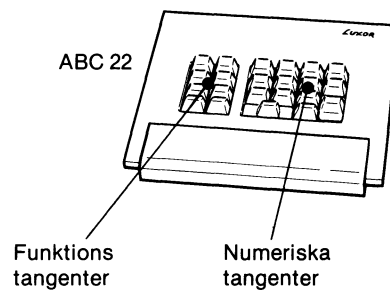
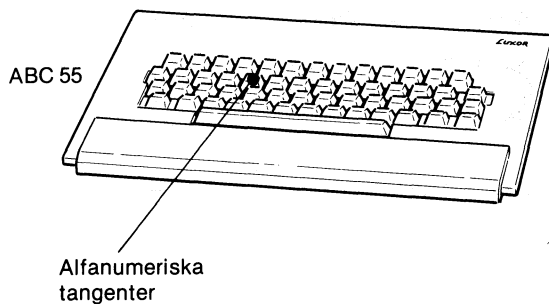
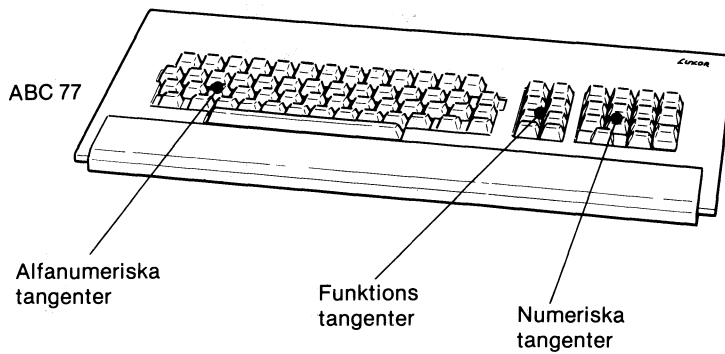
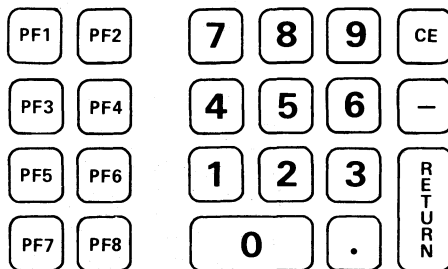
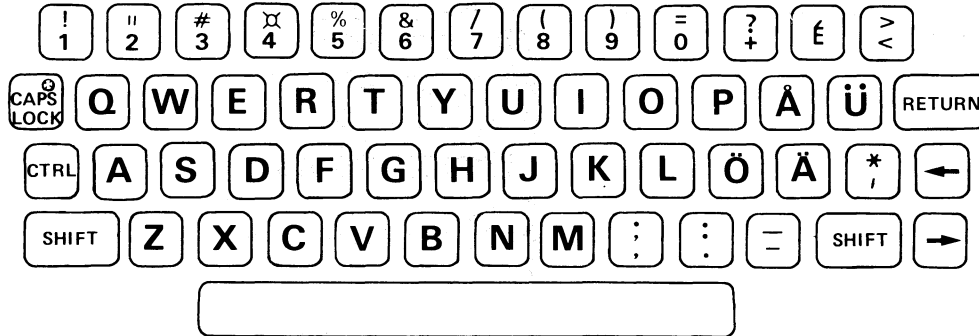
Om minnesutrymme för maskinspråkrutiner ska reserveras, ändras följande adresser.

- Pekare till lägsta minnesadressen för BASIC-program (BOTTOM): 65292.
- Pekare till högsta minnesadressen för BASIC-program (TOP): 65294.

Grafik/dataminnet är inte direkt adresserbart i BASIC.

Bilaga 3

Tangentbordslayout, ASCII-koder



ASCII-kod	Ctrl	Shift	Tangent	ASCII-namn	Funktion
0	X		É	NUL	Tidsutfyllnadstecken
1	X		A	SOH	—
2	X		B	STX	—
3	X		C	ETX	Stoppar exekvering
4	X		D	EOT	—
5	X		E	ENQ	—
6	X		F	ACK	—
7	X		G	BEL	"Pip" i högtalaren
8	X		H	BS	*) "←" tangenten
9	X		I	HT	*) "→" tangenten
10	X		J	LF	Radframmatning
11	X		K	VT	—
12	X		L	FF	*) Raderar skärmen
13	X		M	CR	*) "RETURN" tangenten
14	X		N	SO	—
15	X		O	SI	—
16	X		P	DLE	—
17	X		Q	DC1	—
18	X		R	DC2	—
19	X		S	DC3	Stegar en programinstruktion
20	X		T	DC4	—
21	X		U	NAK	—
22	X		V	SYN	—
23	X		W	ETB	—
24	X		X	CAN	*) Tar bort skriven rad
25	X		Y	EM	—
26	X		Z	SUB	—
27	X		Ä	ESC	—
28	X		Ö	FS	—
29	X		Å	GS	—
30	X		Ü	RS	—
31	X	X	O	US	—
127	X		<	DEL	Ger fylld kvadrat (■).

*) Dessa tecken påverkar skärmen direkt.

Följande kommandon används som styrfunktioner och matas in från tangentbordet:

RETURN	verkställighetskommando
CTRL/C	stoppar exekveringen (OBS! Två CTRL/C avbryter exekveringen.)
CTRL/H eller ←	raderar ett tecken
CTRL/I eller →	används vid editering
CTRL/L	tömmer skärmen
CTRL/S	single-step exekvering
CTRL/X eller CE	raderar inskriven rad

19 Sakregister

A

ABS , matematisk funktion	63, 96
ADD ␣, strängfunktion	67, 96
Addition	12
Användardefinierad funktion	13
AND , operator	14
Aritmetiskt uttryck	4
ASCII , strängfunktion	68, 96
ASCII-tabell	79
ATN , matematisk funktion	63, 96
ATTRIBUTE , instruktion	82, 96
Attributminne	78
AUTO , kommando	25, 96
Avrundning	13

B

␣ BAS	26, 96
BAC , filtyp	32
BAS , filtyp	28
BASIC -tolk	1
Blandning av datatyper	8
BLBG	79, 82, 86
BLK	79, 82, 86
BLU	79, 82, 86
Buffert	10
BYE , kommando	26, 96

C

CALL , funktion	73, 96
CHAIN , instruktion	35, 96
CHR ␣, strängfunktion	68, 97
CLEAR , kommando	26, 97
CLOSE , instruktion	11, 35, 97
COMMON , instruktion	35, 97
COMP% , strängfunktion	68, 97
CON , kommando	26, 97
COS , matematisk funktion	63, 97
COUNT	10, 43, 99
CTRL-C	20, 113
CTRL-H	113
CTRL-I	113
CTRL-L	113
CTRL-S	20, 113
CTRL-X	113
CUR , funktion	73, 97
CVT , funktion	73, 97
CYA	79, 82, 86

D

DATA , instruktion	36, 97
Data	7
DBLE	79, 82

DEF-FN , instruktion	13, 36, 97
Deklaration	13
DIGITS , instruktion	38, 98
DIM , instruktion	9, 15, 39, 98
Dimension	9
Direktanvändning	23
DIV ∅, strängfunktion	78, 98
DOUBLE , instruktion	40, 98
DOS	26

E

ED , kommando	26, 98
Editering	19, 27
EL	79, 82
END , instruktion	40, 98
Enhet	5, 24
EOF, end-of-file	11
EQV , operator	14
ERASE , kommando	27, 98
ERRCODE , funktion	74, 98
Exekvering	20
Exklusiv-OR (XOR), operator	12, 14
EXP , matematisk funktion	63, 98
Exponentiering	62
EXTEND , instruktion	40, 98
EXTEND -mod	8, 40

F

Falskt	4
Felhantering	5
Felmeddelanden	93
Felsökning	60
FGCTL , instruktion	85, 98
FGFILL , instruktion	86, 98
FGLINE , instruktion	86, 99
FGPAINT , instruktion	86, 99
FGPICTURE , instruktion	86, 99
FGPOINT , instruktion	87, 99
FGPOINT , funktion	87, 99
Fil	10
Filhantering	10
Filnamn	24
Filnummer	5, 10
Filpekare	10, 52
Filslut	11
Filtyp	24
FIX , matematisk funktion	63, 99
FLOAT , instruktion	12, 40, 99
FLSH	79, 81
Flyttal	12
Flyttals in/utmatning	13
FN , funktion	74, 99
FNEND , instruktion	38, 99
FOR-TO-STEP , instruktion	41, 99
Funktioner, matematiska	62
Funktioner, sträng-	67
Funktioner, övriga	72

Funktionstangenter	91, 112
Färg	78
Färgvalskommando	85

G

GBLK	79, 82, 86
GBLU	79, 82, 86
GCON	79, 82
GCYA	79, 82, 86
GET , instruktion	42, 99
GET #-COUNT , instruktion	10, 43, 99
GGRN	79, 82, 86
GHOL	79, 82
GMAG	79, 82, 86
GOSUB , instruktion	43, 100
GOTO , instruktion	44, 100
Grafik	78
GREC	79, 82, 86
GREL	79, 82
GRN	79, 82, 86
GSEP	79, 82
GWHT	79, 82, 86
GYEL	79, 82, 86

H

Hakparentes	24
Heltal	12
Heltalsområde	13
Heltalsaritmetik	13
Heltalssuffix	8
HEX \square , matematisk funktion	64, 100
HIDE	83
Hopp, ovillkorligt	44
Hopp, villkorligt	44
Högupplösningsgrafik	84

I

IF-THEN-ELSE , instruktion	44, 100
IMP , operator	14
Indexerade variabler	9
INP , funktion	74, 100
Inport	109
INPUT , instruktion	10, 45, 100
INPUT LINE , instruktion	10, 46, 100
INSTR , strängfunktion	69, 100
Instruktioner	33
INT , matematisk funktion	64, 100
INTEGER , instruktion	46, 100
Interpretator	1
In/ut-enhet	10
I/O-portar	109

K

Kapslad programloop	41
KILL , instruktion	47, 100
Kolon	2, 4
Kommandon	24

Kommentar	3
Kompatibel mod	78
Konkatenera	72, 107
Konstanter	7
Konventioner	24

L

LEFT ☐, strängfunktion	69, 100
LEN , strängfunktion	69, 101
LET , instruktion	47, 101
LIST , kommando	28, 101
LOAD , kommando	29, 101
LOG , matematisk funktion	64, 101
LOG10 , matematisk funktion	65, 101
Logisk enhet	5
Logiska heltalsvariabler	14
Logiska uttryck	4
Logiska operatörer	14
Logiska variabler	14
Loop	41

M

MAG	79, 82, 86
Matematiska funktioner	62
Matematiska operationer	12
Matris	9
MERGE , kommando	29, 93
MID ☐, strängfunktion	70, 101
Minneskarta	110
MOD , matematisk funktion	65, 101
MUL ☐, strängfunktion	70, 101

N

NAME , instruktion	48, 102
NEW , kommando	30, 102
NEXT , instruktion	42, 102
NO EXTEND , instruktion	49, 102
NOT , operator	14
NO TRACE , instruktion	48, 102
NRML	79, 82
NULN	79, 83
NUM ☐, strängfunktion	70, 102
Numeriska värden	12
Numeriska variabler	8
NWBG	79, 82
Nyckelord	3

O

OCT ☐, matematisk funktion	65, 102
ON ERROR GOTO , instruktion	49, 102
ON-GOSUB , instruktion	49, 102
ON-GOTO , instruktion	50, 102
ON-RESTORE , instruktion	50, 102
ON-RESUME , instruktion	50, 103
OPEN , instruktion	10, 51, 103
Operand	2

Operatorer, logiska	14
Operatorer, relations-	12, 45
Operatorernas prioritet	12
OPTION BASE , instruktion	9, 52, 103
OR , operator	14
OUT , funktion	75, 103
Ovillkorligt hopp	44

P

Parenteser	12
PEEK , funktion	75, 103
PEEK2 , funktion	75, 103
PI , matematisk funktion	65, 103
POKE , funktion	75, 103
POSIT , instruktion	10, 52, 103
Precision	13, 40, 59
Prefixet FN	8
PREPARE , instruktion	10, 53, 103
Prioritet	12
Programrader	18
PRINT , instruktion	10, 53, 104
PRINT , instruktion – grafik	81, 104
PRINT USING , instruktion	54, 104
PUT , instruktion	10, 57, 104

R

Radering	19
Radnummer	2
RANDOMIZE , instruktion	57, 104
READ , instruktion	57, 104
RED	79, 82, 86
Relationsoperatorer	12, 45
Relationsuttryck	4
REM , instruktion	58, 104
REN , kommando	30, 104
RENUMBER	30, 104
Reserverade ord	24
RESTORE , instruktion	58, 104
RESUME , instruktion	59, 104
RETURN -tangenter	18, 24, 113
RETURN , instruktion	59, 105
RIGHT χ , strängfunktion	70, 105
RND , matematisk funktion	65, 105
RUN , kommando	31, 105
Rättelser i program	19

S

Sant	4
Sammanfatt programrad	4
Satser	2
SAVE , kommando	32, 105
SCR , kommando	25
SGN , matematisk funktion	66, 105
SIN , matematisk funktion	66, 105
SINGLE , instruktion	59, 105
Skivoperativsystem	26
Skrivsätt	18

Soft-scroll	81
SPACE ⌘, strängfunktion	71, 105
SQR , funktion	66, 105
STDY	79, 82
STOP , instruktion	60, 105
STRING ⌘, funktion	71, 106
Strängaritmetik	16
Strängkonstanter	15
Stränguttryck	15
Strängfunktioner	16, 67
Stränginmatning	16
Stränglängd	15
Strängstorlek	15
Strängutmatning	17
Strängvariabler	15
Strängvektorer	15
Styrtecken, grafik	78
Stängning av fil	11
SUB ⌘, strängfunktion	71, 106
Subrutiner (underprogram)	43
Suffixet %	8, 12
Suffixet⌘	8
SWAP %, funktion	76, 106
Syntaxbeskrivning	24
SYS , funktion	76, 106

T

TAB , funktion	76, 106
Talområde	7
TAN , matematisk funktion	66, 106
Teckensträngar	15
Teledata	78
TIME ⌘, funktion	76, 106
TRACE , instruktion	60, 106

U

ULN	79, 83
UNSAVE , kommando	32, 106
Utport	109
Utropstecken	3
Uttryck	4

V

VAL , strängfunktion	71, 106
Variabler	8
VAROOT , funktion	77, 107
VARPTR , funktion	77, 107
Vektorer	9
Vektorvariabler	9
Villkorligt hopp	44, 60

W

WEND , instruktion	60, 107
WHILE , instruktion	61, 107
WHT	79, 82, 86
WIDTH	61, 107

X	
XOR, operator	14
Y	
YEL	79, 82, 86
Å	
Återhopp	59
Ä	
Ändring av programrad	19
Ö	
Öppning av fil	10

LUXOR
Datorer

Art. nr. 66 78089-11

Teknikformatörerna i Linköping AB ROLAND FÖRETAGEN AB, LINKÖPING