Commodore BASIC 7.0

för Commodore 128

av Anders Hesselbom

Commodore BASIC 7.0

för Commodore 128

av Anders Hesselbom

Publicerad 2025-01-03, revision 3, 2025-12-11. Rättningar och kommentarer skickas till anders@winsoft.se
Revisionshistorik finns här: https://ahesselbom.se/pages/commodorebasic70.html
Samtliga böcker om Commodore finns listade här: https://ahesselbom.se/pages/commodorebasic.html

Innehåll

KAPITEL 1: INTRODUKTION

Introduktion

Commodore 128 är en av de mest mångsidiga och kapabla datorer som någonsin skapats. Maskinen har en avancerad BASIC (som är denna boks primära fokus), är kompatibel med Commodore 64, har två huvud-processorer som används antingen inom eller utanför diskoperativsystemet CP/M, och har en generös uppsättning av inbyggda kommandon och verktyg.

Min bok om **Commodore BASIC 2.0 second release** handlar främst om det nämnda *språket*. Om du köpte en VIC-20 eller en Commodore 64 var det den BASIC-versionen du fick inbyggd i din dator. Commodore BASIC 2.0 second release saknar kommandon för multimedia, så ska du skriva program som utnyttjar datorns kapacitet för grafik och ljud är du hänvisad till att sätta minnesadresser eller rent av välja maskinkod istället för BASIC. Den boken är neutral till ditt val av dator.

Commodore BASIC 7.0 är ett språk framtaget för just Commodore 128, och den datorn har ungefär samma multimediakapacitet som Commodore 64. Den här boken är därför knuten till en specifik dator, nämligen just Commodore 128, och vänder sig till dig som vill bemästra den datorn, främst genom att lära sig dess BASIC – Commodore BASIC 7.0 – ett språk som erbjuder avancerade kommandon för multimedia.

Den här boken ger inte någon komplett bild av Commodore BASIC 7.0, utan fokuserar på datorn Commodore 128 och nyheterna i Commodore BASIC 7.0, som introducerats sedan Commodore BASIC 2.0 second release, vilket innebär att det kan vara bra att ha det min tidigare nämnda bok som förkunskap.

För att få grunderna i Commodore BASIC, få en mer komplett terminologi och kunskap om principerna som gäller för din Commodore-dator, läs min bok om **Commodore BASIC 2.0 second release** först! Har du redan grundkunskaper BASIC och vill fördjupa din kunskap om Commodore 128, är det rätt bok du håller i din hand.

Konventioner i boken

Indata som programrader eller kommandon skrivs med följande teckensnitt:

```
PRINT "HEJ"
```

Samma teckensnitt används för svaren från datorn.

Hänvisningar till tangenter på Commodore 128 skrivs med fetstil. Bilden visar till exempel **Return** till höger, **Run Stop** till vänster, och så vidare.

Figur 1: Tangentbordslayout på Commodore 128. Foto: Evan Amos

Den exakta tangentbordslayouten varierar beroende på vilken marknad du den dator du köpt är avsedd för. Bilden ovan visar en engelsk Commodore 128. För information om de olika tangenternas funktion, se din dators användarmanual.

Ibland ska du trycka ner två tangenter. Om det står till exempel **Shift+A** ska **Shift** hållas nedtryckt medan **A** trycks ner.

Bildförklaringar och kodförklaringar skrivs i *kursiv stil*, som också används för att emfasera termer eller viktiga poänger. Även namn på felmeddelanden skrivs med kursiv stil.

Commodore BASIC 7.0

Commodore BASIC 7.0 är en vidareutveckling av Commodore BASIC 2.0 second release, och innehåller ungefär samma uppsättning av kommandon som Commodore BASIC 3.6. Förutom samtliga kommandon från 2.0 och några kommandon för flödeskontroll och felsökning handlar de flesta antingen om I/O eller multimedia.

Version 3.6 togs fram till en bärbar dator, *Commodore LCD*, som aldrig nådde marknaden. Men mycket arbete som Commodore gjorde, togs med till Commodore 128, däribland BASIC, som färdigutvecklat fick versionsnumret 7.0.

Kommandot `SYS`, har utökats för att vara mer mångsidigt.

För flödeskontroll har vi fått både ett `ELSE`-kommando och några nya sätt att skapa iterationer med `DO` och `LOOP`.

För den som vill analysera eller manipulera textsträngar finns t.ex. det nya kommandot `INSTR`.

För I/O finns en rik uppsättning kommandon för att skriva och läsa data, som t.ex. `BLOAD` och `BSAVE`.

För att hantera användarinteraktioner finns kommandon för att läsa av joystick (styrspak) och ljuspenna.

För högupplöst grafik finns inbyggda kommandon för att rita figurer som linjer, cirklar och rektanglar.

För rörlig grafik finns en uppsättning av kommandon för att hantera sprites.

För att skapa musik finns det mycket avancerade kommandot `PLAY`, och för den som vill utveckla egna ljudeffekter finns kommandot `SOUND`.

I princip finns det kommandon för att komma åt Commodore 128:s samtliga funktioner.

Versioner

Commodore BASIC finns i de versioner som presenteras nedan.

Version 1.0 för Commodore PET 2001 som baseras på Microsoft BASIC.

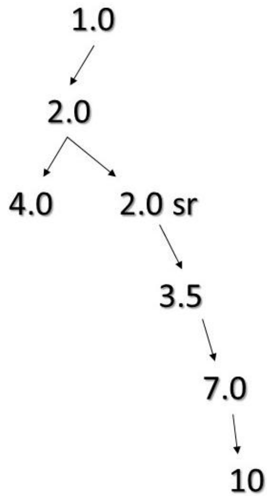
Version 2.0 för Commodore PET 2001 som är en vidareutveckling av version 1.0.

Version 4.0 för Commodore PET 4000 och CBM 8000 är den sista vidareutvecklingen av första version 2.0.

Version 2.0 second release som avhandlas i min tidigare bok (se sida 3) är en buggrättad version av version 2.0 för PET 2001.

Version 4+ för Commodore CBM-II är en vidareutveckling på version 2.0 för PET 2001.

Version 3.5 för Commodore 16, Commodore 116 och Plus/4 är en vidareutveckling av Commodore BASIC 2.0 second release.



Figur 2: Språkets utveckling.

Version 7.0 för Commodore 128 är en vidareutveckling av version 3.5. Denna fanns även i prototypen Commodore LCD med versionsnumret 3.6, och är den version som är ämnet för denna bok.

Version 10 utvecklades för prototypen Commodore 65. Varken Commodore 65 eller Commodore LCD nådde någonsin konsumentmarknaden. Tyska *MEGA Museum of Electronic Games and Art* arbetar med att få ut en färdigställd Commodore 65-klon på marknaden.

Bokens innehåll

Den här boken innehåller, inklusive introduktionen, 13 kapitel och fem bilagor. Här följer en överblick över bokens kapitel, utöver detta första kapitel:

- Det andra kapitlet beskriver **allmänna förbättringar** i Commodore BASIC 7.0 jämfört med
- Det tredje kapitlet ger en övergripande beskrivning om datorn boken handlar om, **Commodore 128**.
- Kapitlet **Text** presenterar nya möjligheter att analysera och manipulera text.
- I kapitlet om **grafik** beskrivs hur högupplöst grafik kan skapas med Commodore BASIC 7.0.
- **Sprites** handlar om rörlig grafik och enklare animationer.
- Kapitlet om **ljud** visar hur man kan få Commodore 128 att spela upp enklare toner och effekter.
- I kapitlet om **musik** beskrivs hur melodier kan komponeras och framföras av flera röster.
- I kapitlet om **avancerade ljudeffekter** beskrivs tidigare odokumenterade funktioner för att skapa nya ljud med Commodore BASIC 7.0.
- Kapitlet om **användarinteraktion** beskriver hur man läser av tangentbordet, joysticks och ljuspennan.
- I kapitlet om **80-kolumnsläge** förklaras hur man kan dra nytta av datorns förmåga att dubblera antalet tecken som visas på skärmen.
- Därefter beskrivs de utökade möjligheterna att bevara data på disk i kapitlet om **DOS**.
- **Commodore 64-läget** beskrivs i det tolfte kapitlet.
- Det trettionde och sista kapitlet ger en introduktion till **CP/M**.

Här följer en beskrivning av bokens fyra bilagor, kallade *appendix A, B, C, D* och *E*:

- **Appendix A** handlar om felsökning (debugging)
- **Appendix B** förklarar de tekniska termer som används i boken
- **Appendix C** jämför Commodore 128 med föregångarna Commodore 64 och VIC-20
- **Appendix D** ger en introduktion till maskinkod
- **Appendix E** beskriver övriga funktioner i Commodore 128

KAPITEL 2: OM COMMODORE 128

Om Commodore 128

Commodore 128 introducerades på marknaden år 1985, och såldes fram till och med år 1989, när 16-bitarssystemen som Atari ST och Amiga började vinna mark. Som namnet indikerar har datorn 128 kilobyte (KB) RAM (som kunde utökas till 640 KB) vilket räcker ganska långt för många olika typer av program, men det är kanske lite i underkant för mer avancerade animationer och avancerad multimedia.

Detta kapitel ger en snabb överflygning över datorns chips, hur BASIC-program matas in, vilka datatyper BASIC hanterar och vilken minneshanteringskapacitet som BASIC erbjuder.

Datorn har två huvudprocessorer. MOS 8502 klarar samma instruktioner som de processorer som satt i bl.a. Commodore 64 (MOS 6510 eller MOS 8500) och VIC-20 (MOS 6502). Det är denna som normalt driver runt din Commodore 128 med en arbetshastighet på 1-2 megahertz (MHz). Den andra processorn är en Zilog Z80 på 4 MHz. Det är denna som driver runt din dator när du arbetar i CP/M-läge.

Det finns ett antal olika operativsystem för Commodore 128. Datorn kan köras med eller utan operativsystemet CP/M. CP/M (*Control Program for Microcomputers*) behöver läsas in från diskett (att ”boota” operativsystemet). Datorn levererades med version CP/M Plus version 3.0, och ger tillgång till avancerad mjukvara som t.ex. Turbo Pascal eller Microsoft Basic.

För den som inte vill köra operativsystem som likt CP/M styrs med textkommandon, finns möjligheten att köpa till det grafiska operativsystemet GEOS som kontrolleras med mus. GEOS (Graphic Environment Operating System) använder sig av rullgardinsmenyer, fönster och ikoner för att låta användaren kontrollera datorn.

Commodore 128 har samma ljudkapacitet som Commodore 64, som drivs av MOS-chippet 6581 (8580 i senare modeller). Dessa kallas kort och för SID (Sound Interface Device).

För grafik har Commodore 128 en VIC-II E med ungefär samma kapacitet som VIC-II som satt i Commodore 64, vilket innebär en skärmupplösning på 320×200 punkter (pixlar), 16 färger, 8 sprites och raster. Förutom detta har version E även stöd för blitter och ett extra grafikläge med en skärmupplösning på 640×200 punkter.

Om inget operativsystem startas, används själva Commodore BASIC 7.0 för att kontrollera datorn genom BASIC-kommandon.

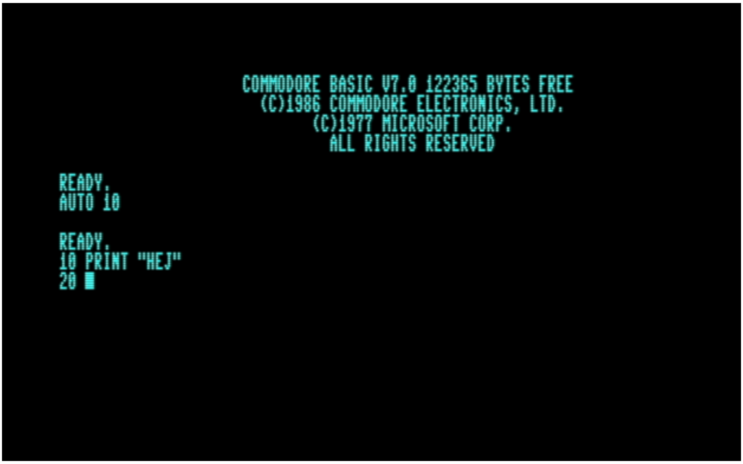
För att programmera Commodore 128 används företrädelsevis just Commodore BASIC, som i utförande 7.0 är väldigt kraftfullt. Den stora nackdelen med BASIC är dess undermåliga prestanda. BASIC kan göra allt du önskar men om prestanda är en faktor måste du titta på något annat, t.ex. maskinkod. Maskinkod presenteras överskådligt i appendix C.

Detta kapitel berör följande ämnen:

- Inmatning av BASIC-program
- Funktionstangenterna
- Datatyper
- Minneshantering
- Tre datorer i en

Inmatning av BASIC-program

Språket har två lägen. Det ena kallas *direkt*, och innebär att man skriver en instruktion utan radnummer, som exekveras direkt när man trycker på **Return**. Det andra kallas *runtime*. Instruktioner som får ett radnummer, exekveras i runtime, alltså när programmet körs med (normalt) RUN. Om inget annat anges, kan alla kommandon användas både i direktläge och i runtime-läge.För den intresserade bjuds en hel del trevliga tricks, som t.ex. `AUTO`, som visas på bilden.



Figur 3: AUTO erbjuder automatisk inskrivning av radnummer.

Kommandot `AUTO` tar ett argument, och det är avståndet från nuvarande rad till nästa. Genom att skriva `AUTO 10` så säger man till datorn att nästa radnummer ska vara nuvarande plus 10, vilket gör att datorn föreslår 20 efter att ett kommando matats in på rad 10, och så vidare.

För att stänga av automatiskt förslag på radnummer, skriv `AUTO` utan några parametrar och tryck **Return**.

Om du vill infoga ett kommando mellan, säg, rad `10` och rad `20`, kan du kalla den nya raden för `15`. Genom att skriva in dessa tre rader…

```
10 PRINT "A"
20 PRINT "B"
15 PRINT "C"
```

…så får du följande program, som kan visas med `LIST`:

```
10 PRINT "A"
15 PRINT "C"
20 PRINT "B"
```

Om du infogar tillräckligt många rader mellan rad `10` och `20`, kommer utrymmet att ta slut. Kommandot `RENUMBER` justerar avståndet mellan existerande rader så att den första raden börjar på `10` och att avståndet mellan alla rader. Exempel:

```
RENUMBER

READY.
LIST

10 PRINT "A"
20 PRINT "C"
30 PRINT "B"
```

Det finns även en möjlighet att själv ange startnummer och steg. Genom att anropa `RENUMBER 3,2` kommer programmet att starta på rad `3` och efterföljande rader kommer med ett avstånd på `2`.

```
RENUMBER 3,2

READY.
LIST

3 PRINT "A"
5 PRINT "C"
7 PRINT "B"
```

Du kan även ange `RENUMBER` endast ska agera på rader som är lika med eller större än ett visst befintligt radnummer genom att komplettera med ett startnummer. `RENUMBER 1000,10,5` säger att kodrader i programmet ska starta på rad `1000` och efterföljande rader kommer på ett avstånd på `10`, men det gäller bara raderna som nu är större än eller lika med `5` (alltså inte rad `3`).

```
RENUMBER 1000,10,5

READY.
LIST

3 PRINT "A"
```

```
1000 PRINT "C"
1010 PRINT "B"
```

Om du använder `RENUMBER` i ett program så uppstår felet *direct mode only*, och `RENUMBER` får inte användas när programmet har en `GOTO` eller `GOSUB` som inte pekar på en existerande rad, då uppstår felet *unresolved reference*. Om programmet slår över taket (högsta tillåtna radnummer är `63999`) uppstår felet *line number too large*.

En liten varning! I Commodore BASIC definieras subrutiner av ett radnummer, vilket innebär att du behöver kontrollera vilka nya radnummer dina subrutiner har fått efter att du använt `RENUMBER`. Annars kan det bli svårt att anropa dessa på nytt.

Funktionstangenterna

Commodore 128 har endast fyra fysiska funktionstangenter. Dessa är **F1**, **F3**, **F5** och **F7**. **F2**, **F4**, **F6** och **F8** är virtuella funktionstangenter. När du förväntas trycka **F2**, ska du egentligen trycka **Shift+F1** och när du förväntas tycka **F4**, ska du egentligen trycka **Shift+F3**, och så vidare. De åtta (inklusive de virtuella) funktionstangenterna erbjuder en blandning av skrivhjälp och snabbtangenter. Som skrivhjälp hittar vi **F1** som ger `GRAPHIC`, **F2** ger `DLOAD"` och **F5** ger `DSAVE"`). Och som snabbtangenter hittar vi **F3** som visar innehållet på din diskett, **F4** rensar textskärmen, **F6** startar aktuellt BASIC-program, **F7** listar aktuellt BASIC-program och **F8** tar dig till maskinkodsmonitorn.

Innehållet i funktionstangenterna ligger lagrat på rad på adress `4106`, rygg i rygg. Har man inte ändrat något, så ser man hur `GRAPHIC` (**F1**, `4106-4112`) ligger rygg-i-rygg med `DLOAD"` (**F2**, `4113-4118`). Efter `DIRECTORY` (**F3**, `4119-4128`) ser vi även värdet `13` (som motsvarar tangenten **Return**) efter `DIRECTORY`, eftersom kommandot `DIRECTORY` exekverar direkt när man trycker på **F3**. Det är därför `DIRECTORY` konsumerar 10 bytes, trots att ordet endast är 9 tecken långt. Och så vidare.

Anledningen till att Commodore 128 vet vilket kommando eller makro som finns lagrat var, har med adress `4096` att göra (innebörden av **F1** återges alltid på adress `4106`). Där har vi tio bytes (`4096-4105`) som beskriver de olika funktionernas längd. Därför vet datorn att `GRAPHIC` (**F1**) tar upp sju bytes, för siffran `7` står på adress `4096`. Det är också därför siffran `6` står på adress `4097` – `DLOAD"` tar upp sju tecken. Adress `4096` (t.o.m. `4105`) är helt enkelt ett register på 10 bytes, men funktionstangenterna är endast åtta i antal.

Efter de åtta funktionstangenterna finns två längdangivelser till. Den nionde längdangivelsen ger oss definitionen av makrot `BOOT`, som används för att starta det första programmet på den förnärvarande laddade disketten. Detta makro körs varje gång datorn startas. Adress `4104` talar om hur långt det makrot är. Den tionde adressen (`4105`) ger oss länden av innehållet bakom tangenten **Help**, vilket är fem tecken: Ordet `HELP` följt av ett enterslag (`13`).

Om du definierar om funktionstangenterna, återställs dem när du startar om datorn. Detta BASIC-program kan användas för att hitta på egna betydelser för både funktionstangenterna, `BOOT`[1] och `HELP`.

```
10 DIM A$(10)
20 A$(1)="GRAPHIC"
30 A$(2)="DLOAD"+CHR$(34)
40 A$(3)="DIRECTORY"+CHR$(13)
50 A$(4)="SCNCLR"+CHR$(13)
60 A$(5)="DSAVE"+CHR$(34)
70 A$(6)="RUN"+CHR$(13)
80 A$(7)="LIST"+CHR$(13)
90 A$(8)="MONITOR"+CHR$(13)
100 A$(9)="D[Shift+L]"+CHR$(34)+"*"+
    CHR$(13)+"RUN"+CHR$(13)
110 A$(10)="HELP"+CHR$(34)
120 FOR I=0 TO 9:POKE 4096+I,LEN(A$(I+1)):NEXT
130 PADR=4096:CADR=4106
140 FOR I=0 TO 9
150 POKE PADR+I,LEN(A$(I+1))
160 FOR J=1 TO LEN(A$(I+1))
170 POKE CADR,ASC(MID$(A$(I+1),J,1)):CADR=CADR+1
180 NEXT
190 NEXT
```

Notera instruktionen om tangenttryck på rad 100.

Detta program låter dig konfigurerar alltså betydelsen av de åtta funktionstangenterna, vad kommandot `BOOT` gör, samt vad som ska ligga

[1] Notera att F1-F8 samt HELP anger vad som gömmer sig bakom knapparna, medan för BOOT ändrar man vad kommandot BOOT gör.

under **Help**-knappen. Låt säga att jag vill att kommandot BOOT ska spela en trudelutt, kan jag ändra rad `100` till detta:

```
100 A$(9)="PLAY"+CHR$(34)+"CDE"+CHR$(34)+CHR$(13)
```

Vill du anpassa funktionstangenterna, `BOOT` och **Help**, behöver du köra ett program vid uppstart (t.ex. detta), för funktionaliteten återställs vid omstart.

Datatyper

Commodore BASIC 7.0 har stöd för tre datatyper. Dessa är *realtal*, *heltal* och *strängar*. Realtal använder punkt som decimalavgränsare och strängar omges av citattecken. Variablernas typ deklareras med ett postfix på variabelnamnet, där `$` (dollartecken) avser sträng och `%` (procenttecken) avser heltal. Realtal är variabler som saknar ett avslutande tecken. Apropå variabelnamn så identifieras variabler endast av de två första bokstäverna i sitt namn. En mer ingående beskrivning av datatyper och variabelnamn finns i boken *Commodore BASIC 2.0 second release*[2].

Minneshantering

Commodore 128 arbetar med s.k. minnesbanker. En minnesbank är en fördefinierad minneskonfiguration, och du bestämmer vilken minnesbank som är tillgänglig för processorn genom att använda kommandot `BANK`. Som argument tar `BANK` ett tal mellan `0` och `15`. Exempel:

```
BANK 4
```

Det finns totalt 16 stycken minnesbanker 64 kilobyte vardera, men inte alla har någon funktion. På en Commodore 128 utan någon minnesexpansion finns det egentligen bara två unika banker (därav namnet 128). Om ingen bank har valts, arbetar du med bank `15`, vars RAM-del speglar bank `0`, `2`, `4`, `6`, `8`, `10`, `12`, `13` och `14`. RAM-delen i bank nummer `12` speglar bank `1`, `3`, `5`, `7`, `9` och `11`.

Val av minnesbank påverkar kommandon som använder minnet direkt. Dessa kommandon är `SYS`, `PEEK`, `POKE` och `WAIT`. Minnesbank `15` är den som är förvald, och vad den speglar beror på hur mycket extraminne som

[2] Finns att ladda hem i PDF-format eller EPUB-format här: https://ahesselbom.se/pages/commodorebasic20.html

sitter i datorn. Vill du se en konsekvens av att växla mellan minnesbanker, kan du köra ett enkelt testprogram[3].

```
10 POKE 4192,75
20 PRINT PEEK(4192)
30 BANK 1
40 POKE 4192,90
50 PRINT PEEK(4192)
60 BANK 15
70 PRINT PEEK(4192)
```

Rad 10 skriver värdet 75 i aktuell minnesbank (vilket är 15 om inget annat har sagts).
Rad 20 konstaterar det skrivna värdet.
Rad 30 växlar minnesbank till 1.
Rad 40 skriver 90 till samma adress, fast i bank 1 istället för bank 15.
Rad 50 konstaterar det skrivna värdet.
Rad 60-70 växlar tillbaka till bank 15 och konstaterar att adressen fortfarande innehåller 75.

Programmet ger `75`, `90`, `75` (givet att du befann dig i bank `15` när du startade programmet). Kapitel 17 i systemguiden beskriver banker, och detta program (nästa sida) visar hur många unika RAM-banker (innehållet i ROM kan variera – se manualen) som är tillgängliga på din maskin:

```
10 FOR A=0 TO 15:BANK A:POKE 4192,A+1:NEXT
20 DIM B%(15)
30 FOR A=0 TO 15:BANK A
40 IF PEEK(4192)=A+1 THEN B%(A)=B%(A)+1
50 NEXT
60 FOR A=0 TO 15
70 IF B%(A)>0 THEN PRINT "BANK";A
80 NEXT
```

På en oexpanderad maskin borde resultatet bli:

```
BANK 11
BANK 15
```

På en expanderad maskin kommer programmet att stoppas av interpretatorn[4].

Utöver detta, kommandot `STASH` och kommandot `FETCH` används för att kopiera data mellan minnesbanker och `SWAP` används för att låta två minnesbanker byta data mellan varandra. Normalt behöver du inte tänka på minnesbanker när du programmerar i BASIC, men om du vill jobba direkt mot minnet är det bra att veta hur man kommer åt det som finns tillgängligt.

---

[3] Kom ihåg att använda kommandot NEW för att radera eventuellt befintligt BASIC-program ur minnet, innan du skriver in ett nytt program.
[4] Interpretatorn är den mjukvara som är inbyggd i (bl.a.) Commodore 128 som läser kodraderna i ett BASIC-program och utför dem.

Avslutningsvis, om du vill läsa av minnesadressen till en specifik variabel används funktionen `POINTER`. Exempel:

```
PRINT POINTER(A)
```

Resultatet av ovanstående är att minnesadressen till `A` skrivs på skärmen.

Tre datorer i en

Din dator kan köras i tre lägen:

- Commodore 128-läge
- CP/M-läge
- Commodore 64-läge

Commodore 128 är egentligen tre datorsystem i en enda låda. Vid normal uppstart hamnar man i det som kallas **Commodore 128-läge**, och det är där vi har Commodore BASIC 7.0 som primärt kontrollprogram.

Om datorn startas med boot-disketten för CP/M 3.0, hamnar man i **CP/M-läge**. I detta läge är det inte längre MOS-processorn (MOS 8502) som är huvudprocessor, utan Z80-processorn. I CP/M-läge kan du utföra avancerade I/O-instruktioner och köra program som Turbo Pascal eller Microsoft BASIC. Du kan växla till CP/M-läget från Commodore 128-läget genom att skriva `BOOT` (givet att du inte ändrat definitionen av makrot i fråga och att rätt diskett är laddad i diskettstationen[5]). CP/M-läget beskrivs i ett eget kapitel (sida 72).

Om du håller ner Commodore-tangenten medan du startar datorn, hamnar du i **Commodore 64-läge**. Detta läge är mycket attraktivt för sitt gigantiska utbud av mjukvara, främst spelprogram. Du kan växla till Commodore 64-läget från Commodore 128-läget genom att skriva:

```
GO 64
```

(Skrivs ofta `GO64`.) Commodore 64-läget beskrivs i ett eget kapitel (sida 70).

[5] Jag använder ordet diskdrive synonymt med ordet diskettstation i den här boken.

KAPITEL 3: ALLMÄNNA FÖRBÄTTRINGAR I BASIC

Allmänna förbättringar i BASIC

Den stora skillnaden mellan Commodore BASIC 2.0 second release (programmeringsspråket som Commodore 64 och VIC-20 är utrustad med) och Commodore BASIC 7.0 (som är inbyggt i din Commodore 128) handlar om DOS och om multimedia. Men det finns några allmänna förbättringar i version 7.0 som verkligen underlättar arbetet för den som vill bygga mjukvara. Förbättringarna rör följande områden:

- Tester
- Iterationer
- Prestanda
- Övrigt

Kapitlet tar även upp några övriga förbättringar som inte passar in i något av de övriga kapitlen.

Tester

En viktig förändring i hur tester fungerar är att ett eller flera kommandon kan exekveras om ett uttryck är sant. Nyckelordet `ELSE`, som används tillsammans med `IF`, åstadkommer detta.

Låt säga att du vill att två saker ska hända om `A` är lika med `5`, men två andra saker om `A` inte är lika med `5`. På en Commodore 64 skulle detta kunna se ut så här:

```
630 IF X=5 THEN PRINT "A":PRINT "B"
640 IF X<>5 THEN PRINT "C":PRINT "D"
```

Exemplet ovan är hypotetiskt och utgör inte ett komplett program. De två ting som utförs om `X` har värdet `5` är förmodligen något annat än `PRINT "A"` och `PRINT "B"` i ett realistiskt exempel.

Det finns två problem med ovanstående kod, båda beror på att det falska uttrycket på rad `640` (i förhållande till det på rad `630`) måste formuleras uttryckligen. Dels ökar det risken för buggar, eftersom vi har två uttryck som egentligen beskriver ett uttryck och dess motsats. Och dels måste uttrycket `X=5` utvärderas två gånger – först som det är skrivet och därefter i sin motsats.

Genom att istället använda `ELSE` så löser man båda dessa problem. `ELSE` antar att om det första uttrycket är falskt, är det koden som står efter `ELSE` som ska köras. Det innebär att `X=5` endast behöver utvärderas en gång, och att det inverterade uttrycket inte behöver tillhandahållas av programmeraren. Denna programsats gör samma sak som exemplet ovan (radbytet ska inte vara med):

```
630 IF X=5 THEN PRINT "A":PRINT "B":
    ELSE PRINT "C":PRINT "D"
```

> Notera att `ELSE` kräver ett kolon (`:`) före, men inte efter.

Om du vill att flera programrader ska köras om ett villkor är sant, kan `IF` kompletteras med `BEGIN` och `BEND`. `BEGIN` och `BEND` anger att vi har att göra med ett *kodblock*, d.v.s. programsatser som hör ihop. Exemplet på föregående sida, som skriver ut `"A"` och `"B"` om `X` är lika med `5` skrevs så här:

```
630 IF X=5 THEN PRINT "A":PRINT "B"
```

Att skriva mer än en programsats på en rad kan göra att programmet blir svårare att underhålla. Därför är det att föredra att skriva `BEGIN` efter `THEN` enligt följande:

```
630 IF X=5 THEN BEGIN
640 PRINT "A"
650 PRINT "B"
660 BEND
```

Resultatet blir detsamma, men vi slipper klämma in flera programsatser på en rad. Hela exemplet från föregående sida, fast med `BEGIN` och `BEND` skulle se ut så här:

```
630 IF X=5 THEN BEGIN
640 PRINT "A"
650 PRINT "B"
660 BEND: ELSE BEGIN
670 PRINT "C"
680 PRINT "D"
690 BEND
```

> Om du skriver programsatser på samma rad som `BEGIN`, behövs ett kolon (`:`) efter, vilket skiljer `BEGIN` från `ELSE`.

Apropå tester så har vi fått en ny logisk operator som inte fanns tillgänglig på Commodore 64 och VIC-20. Dessa maskiner hade `AND`, `NOT` och `OR` att tillgå. Sanningstabellerna för dessa beskrivs på sida 72 och framåt i min bok om Commodore BASIC 2.0 second release. Det nya tillskottet heter `XOR`[6] och är en variant på `OR`. Skillnaden mellan `OR` och `XOR` är att `OR` anser att ett test är lyckat om någon operand utvärderas som sann, medan `XOR` anser att ett test är lyckat om den ena eller den andra operanden utvärderas som sann[7], dock inte om båda operander utvärderas som sanna.

Så här ser sanningstabellen för `OR` och `XOR` ut:

				OR	**XOR**
Sant	`OR/XOR`	Sant	=	Sant	Falskt
Sant	`OR/XOR`	Falskt	=	Sant	Sant
Falskt	`OR/XOR`	Sant	=	Sant	Sant
Falskt	`OR/XOR`	Falskt	=	Falskt	Falskt

Operatorn `XOR` efterliknar hur man använder ordet *eller* i dagligt tal. Om du har en påse med äpplen och päron, och någon gör påståendet att påsen innehåller äpplen *eller* päron, blir svaret nej, för påsen innehåller äpplen *och* päron.

[6] XOR betyder "exclusive or".
[7] Sant innebär ett numeriskt värde skilt från 0, typiskt -1, men vilket som helst.

Iterationer

Förutom direkta hopp med `GOTO` och iterationer med `FOR` (som båda beskrivs i min bok *Commodore BASIC 2.0 second release*), kan Commodore 128:ans BASIC iterera[8] så länge ett villkor är sant eller tills ett villkor blir sant med `DO` och `LOOP`. Följande kod itererar 10 gånger med hjälp av `GOTO`. Talen `1` till `10` skrivs ut på skärmen, följt av ordet `KLAR` när programmet körts klart. Metoden var tillgänglig redan i tidigare versioner av Commodore BASIC.

```
10 A=0
20 A=A+1
30 PRINT A
40 IF A<10 GOTO 20
50 PRINT "KLAR"
```

En förenkling av denna kod som också har varit tillgänglig sedan start (sidan 6) är `FOR`. Följande program ger samma resultat som föregående:

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

(Notera att på rad `30` går det bra att skriva `NEXT A` i stället för bara `NEXT`, vilket gör att interpretatorn kan identifiera syftningsfel.)

På Commodore 128, där `DO` och `LOOP` är tillgängligt, öppnas flera möjligheter. En oändlig iteration som räknar upp och skriver ut värdet av `A`, kan se ut så här:

```
10 A=0
20 DO
30 A=A+1
40 PRINT A
50 LOOP
```

Om man vill villkora att koden mellan `DO` och `LOOP` (rad `30` och `40` i exemplet) kan `WHILE` eller `UNTIL` användas. En summering:

[8] Med iterera menas att upprepa utförandet av ett antal programsatser.

Syftet med att sätta `WHILE` efter `DO` är att säga: Kör endast koden mellan `DO` och `LOOP` om uttrycket som följer efter `WHILE` är sant.

Syftet med att sätta `UNTIL` efter `DO` är att säga: Kör endast koden mellan `DO` och `LOOP` om uttrycket som följer efter `UNTIL` är falskt.

Syftet med att sätta `WHILE` efter `LOOP` är att säga: Upprepa koden mellan `DO` och `LOOP` om uttrycket som följer efter `WHILE` sant.

Syftet med att sätta `UNTIL` efter `LOOP` är att säga: Upprepa koden mellan `DO` och `LOOP` om uttrycket som följer efter `UNTIL` är falskt.

Detta ger oss flera ytterligare sätt att skriva programmet som räknar till 10, och här är ett:

```
10 A=0
20 DO
30 A=A+1
40 PRINT A
50 LOOP UNTIL A>=10
60 PRINT "KLAR"
```

Eftersom uttrycket efter `UNTIL` på rad `50` inte blir falskt förrän när värdet av `A` når upp till 10, upprepas koden mellan `DO` och `LOOP` så länge `A` har ett värde lägre än 10.

Vill man ha fler villkor för att avbryta en iteration, kan `EXIT` användas. I detta fall bryts iterationen redan när `A` har nått värdet `5` (se rad 45):

```
10 A=0
20 DO
30 A=A+1
40 PRINT A
45 IF A=5 THEN EXIT
50 LOOP UNTIL A>=10
60 PRINT "KLAR"
```

Programmet ovan skriver ut `1`, `2`, `3`, `4`, `5` och `KLAR` på skärmen.

Prestanda

Commodore 128 är tillräckligt bra för att driva runt någorlunda avancerade spel och andra typer av datorprogram. En väldigt stor del av datorns funktionalitet görs tillgänglig för användaren genom BASIC, men BASIC är ytterst långsamt. Detta märks främst när man programmerar datorspel.

Men om en tidsödslande beräkning ska göras, så erbjuder BASIC en lösning: Man kan stänga av bilduppdateringen och därmed använda mer av processorns uppmärksamhet till att lösa en uppgift. Nackdelen är att bildskärmen är blank under tiden beräkningen görs. Kommandot `FAST` läcker skärmen och ökar prestandan i BASIC, och kommandot `SLOW` återgår till normalläge.

Följande program exekverar på 482 jiffies:

```
10 T=TI
20 SCNCLR
30 A=150000
40 A=A+1:A=A*2
50 A=A/3:A=A+A/4
60 PRINT A
70 IF A>5.01 THEN 40
80 PRINT TI-T
```

Om vi kompletterar programmet så att första raden släcker skärmen…

```
5 FAST
```

…och sista raden tänder skärmen…

```
90 SLOW
```

…så blir exekveringstiden istället 236 jiffies, vilket innebär en förbättring på ungefär 50%.

(Kommandot `SCNCLR,` som används i programmet på föregående sida, rensar textskärmen och placerar markören på sin hemposition, som är 0×0.

Behöver du hålla reda på om programmet är i `FAST`-läge eller ej, så kan man kontrollera adress `53296`. Denna är normalt satt till `252`, men i `FAST`-läge innehåller `53296` istället värdet `253`. Skillnaden är alltså den minst signifikativa biten, bit `0`. Detta program ger således resultatet `252`, `253` och `252`:

`10 PRINT PEEK(53296)`	*Rad 10 ger 252 eftersom FAST inte är aktiverat.*
`20 FAST`	*Rad 20 sätter programmet i FAST-läge.*
`30 PRINT PEEK(53296)`	*Rad 30 ger 253 eftersom programmet nu befinner sig i FAST-läge.*
`40 SLOW`	*Rad 40 tar programmet ur FAST-läge.*
`50 PRINT PEEK(53296)`	*Rad 50 ger 252 eftersom FAST inte längre är aktiverat.*

Övrigt

För att pausa ett program används `SLEEP` följt av ett önskat antal sekunder (angivet som heltal). Det innebär att kommandot `SLEEP 60` pausar datorn under en hel minut. `SLEEP 0` pausar till nästa interrupt, vilken aldrig är längre än 16,7 millisekunder bort. Tidigare nämnda `FAST` och `SLOW` påverkar inte beteendet för `SLEEP`. Detta program konsumerar 302 jiffies[9]:

```
10 T=TI
20 SLEEP 5
30 PRINT TI-T
```

Kompletterar vi med SLOW och FAST blir resultatet fortfarande 302 jiffies.

```
10 T=TI
15 FAST
20 SLEEP 5
25 SLOW
30 PRINT TI-T
```

[9] En jiffy är som bekant en sextiondelssekund, och 300/60 är fem, och programmet pausar i fem sekunder.

KAPITEL 4: TEXT

Text

Commodore 128 erbjuder en rik uppsättning av funktioner för att tolka och manipulera text. Detta kapitel tar upp det viktigaste från Commodore BASIC 2.0 second release samt alla nyheter. För att få full förståelse för datorns kapacitet beträffande texthantering behöver du läsa kapitlet om text i min bok om Commodore BASIC 2.0 second release. Detta kapitel berör följande aspekter:

- Skiftläge
- Omvandla mellan text och tal
- Formatera ett tal
- Avancerade funktioner för att manipulera text
- Fönsterhantering

Grunderna gällande textbehandling berörs inte av denna bok.

Skiftläge

Likt Commodore 64 har Commodore 128 två teckenuppsättningar.

- Versaler och pseudografik
- Gemener och versaler

I ursprungsläget (versaler och pseudografik) ger tangenterna bokstäver, och när **Shift** hålls nere ger tangenterna grafiska symboler. Användaren kan växla till nästa teckenuppsättning genom att trycka **Commodore**+**Shift**, vilket gör att tangenterna ger gemener, och när **Shift** hålls nere ger tangenterna versaler. Mer information finns i kapitlet om text i boken *Commodore BASIC 2.0 second release*.

Om du som programmerare vill växla till versaler och pseudografik genom att skriva tecknet 14.

```
PRINT CHR$(14)
```

Och du kan växla till gemener och versaler genom att skriva tecknet 142.

```
PRINT CHR$(142)
```

Omvandla mellan text och tal

En textsträng innehållande ett decimaltal (alltså ett tal som beskrivs av tecknen `0` till och med `9` och möjligtvis en punkt som decimalavgränsare) konverteras till ett tal med funktionen `VAL`. Detta enkla program skriver värdet `40` på skärmen.

```
10 X=VAL("20")
20 PRINT X*2
```

Rad 10 konverterar en sträng innehållande en tvåa och en nolla till talet 20.
Rad 20 multiplicerar resultatet med 2 och skriver ut det (40) på skärmen.

Exemplet visar hur en sträng innehållande ett tal kan konverteras till ett riktigt tal och användas i numeriska beräkningar. Detta är ganska standard inom BASIC, och har varit med länge i Commodore BASIC.

Utöver `VAL` kan Commodore 128 även jobba med strängrepresentationer av det hexadecimala talsystemet. De lite förlåtande formuleringarna som `VAL` tillåter[10], hittar vi även i `DEC`, men `DEC` tillåter inte någon decimalavgränsare (som är en punkt i Commodore BASIC).

Funktionen `DEC` tar en textsträng som representerar ett hexadecimalt tal, och ger ett heltal som svar. Följande kommando…

```
PRINT DEC("FF")
```

…skriver ut talet `255`, eftersom decimaltalet `255` representeras av `FF` i det hexadecimala talsystemet.

För att konvertera åt andra hållet, alltså från det decimala talsystemet till det hexadecimala, används funktionen `HEX$`, som tar ett heltal och ger en fyra tecken lång textsträng som representerar ett 16-bitars hexadecimalt heltal som svar. Om ett flyttal skickas till funktionen, ignoreras allt som står till höger om decimalavgränsaren. Följande kommando…

```
PRINT HEX$(255)
```

…skriver ut `00FF`, eftersom det hexadecimala talet `FF` representerar `255` i det decimala talsystemet. De inledande nollorna beror på att `HEX$` alltid ger ett fyra tecken långt svar. `FFFF` motsvarar `65535` i det decimala talsystemet, och är också det största heltal som Commodore BASIC kan hantera. Om ett tal mindre än `0` eller större än `65535` skickas in, uppstår felet *illegal quantity*.

Formatera ett tal

Detta stycke kommer att förklara `PRINT USING` och `PUDEF`.

Avancerade funktioner för att manipulera text

Detta stycke kommer att förklara de nya funktionerna för att manipulera och analysera textsträngar.

Fönsterhantering

Detta stycke kommer att avhandla kommandot `WINDOW` (samt tillhörande kommandon).

[10] Se min bok om *Commodore BASIC 2.0 second release*.

KAPITEL 5: GRAFIK

Grafik

Commodore 128 erbjuder flera möjligheter att visa högupplöst grafik och flerfärgsgrafik. Den som bygger ett datorspel som behöver ha rörlig grafik kommer förmodligen att använda *teckenbaserad grafik* för bakgrundsbilder till spel, eftersom den erbjuder skärmrullning (scrollande grafik) och *sprites* (små objekt som kan flyttas omkring fritt). Till skillnad från Commodore 64 erbjuder Commodore 128 en uppsättning BASIC-kommandon för att skapa vektorgrafik och sprites. Detta kapitel avhandlar möjligheten att skapa vektorgrafik med BASIC.

Vektorgrafik som skapas med BASIC skickas alltid till din 40-kolumnsskärm, så använd gärna din Commodore i 40-kolumnsläge. Du kan alltså skapa vektorgrafik i 80-kolumnsläge, men den kommer att skickas till den skärm som visar 40-kolumnsläget, om någon.

Detta kapitel avhandlar möjligheten att välja grafikläge, hur man ritar punkter och polygoner, rektanglar, cirklar och bågar. Dessutom berörs några avancerade koncept som grafikmarkören och skalning, med mera.

- **Välj grafikläge** – en genomgång av de grafiklägen Commodore 128 presenterar genom BASIC-kommandot `GRAPHIC`
- **Punkter och polygoner** – grundläggande vektorgrafik
- **Rektanglar**
- **Cirklar och bågar**
- **Grafikmarkören** – hjälp att skapa sekvenser av grafiska element
- **Övriga kommandon** – skalning och linjebredd
- **Stenciler** – klipp och klistra grafik
- **Flerfärgsgrafik**

Välj grafikläge

Commodore 128 har olika lägen för att visa text och grafik. Normalt syns inte textmarkören när man jobbar med högupplöst grafik, men du kan (nästan) alltid (om du inte stängt av den möjligheten) hålla nere **Run Stop** (till vänster på tangentbordet) och trycka **Restore** (till höger) för att bryta en programkörning och/eller återgå till textläget.

Kommandot `GRAPHIC` låter dig växla mellan de grafiklägen som din dator kan använda, och här vill jag utfärda en varning till dig som använder Commodore 128 med endast en monitor (vilket verkligen är det normala): **Run Stop**+**Restore** tar dig till aktuellt textläge, men kommandot `GRAPHIC` kan användas mellan 40- och 80-kolumnsläge. Jobbar du i 40-kolumnsläge och använder `GRAPHIC` för att växla till 80-kolumnsläge, kan du inte återgå genom **Run Stop**+**Restore** – då måste du (ev. i blindo) växla tillbaka till 40-kolumnsläge genom att skriva ett textkommando.

Kommandot `GRAPHIC` tar två parametrar. Det låter dig välja ett av sex grafiklägen, vilket anges i den första parametern.

`0` är **standardtextläget**, 40 kolumner.

`1` är **högupplöst bitmapsgrafik**.

`2` är en s.k. **split screen mellan högupplöst bitmapsgrafik och 40 kolumner text**.

`3` är **flerfärgsgrafik** (lågupplöst).

`4` är en **split screen mellan flerfärgsgrafik och 40 kolumner text**.

`5` är **80-kolumnstextläget**.

Den andra parametern kan innehålla `0` eller `1` och anger om grafikminnet ska rensas. Normalt vill man rensa grafikminnet åtminstone första gången man går in i läget, eftersom det kan finnas skräpdata på den minnesadress som utgör grafikminnet (normalt `8192`). Om den andra parametern utelämnas, rensas inte grafikminnet (motsvarar `0`). Detta exempel rensar grafikminnet och ritar en glad gubbe. När användaren trycker **Return**, återgår programmet till standardtextläget (nästa sida).

```
10 GRAPHIC 1,1
20 POKE 8192,60
30 POKE 8193,126
40 POKE 8194,219
50 POKE 8195,255
60 POKE 8196,219
70 POKE 8197,231
80 POKE 8198,126
90 POKE 8199,60
100 INPUT A$
110 GRAPHIC 0
```

Men Commodore 128 har flera avancerade kommandon för att peta in grafik i ett koordinatsystem, t.ex. `DRAW`, som används för både punkter, linjer och polygoner, vilket gör att du inte behöver använda `POKE`.

Punkter, linjer och polygoner

Alla kommandon som ritar linjer påverkar grafikmarkörens position, vilket kan användas för att minska mängden kod man behöver skriva. Grafikmarkören avhandlas senare i detta kapitel.

Kommandot `DRAW` har många uppgifter. Det används för att rita en punkt, för att rita en linje eller för att rita en polygon.

`DRAW` tar två argument: En färg och en eller flera koordinater. I högupplöst läge (grafikläge `1` eller `2` som beskrivs på föregående sida) kan färgen vara antingen `0` (aktuell bakgrundsfärg) eller `1` (aktuell förgrundsfärg). I flerfärgsläge kan även `2` eller `3` anges. Normalt anger man `1` som färg i högupplöst läge, eftersom det anger förgrundsfärgen. Följande program visar en avbildning av stjärnbilden Karlavagnen (eller *Stora björnen*).

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 DRAW 1,22,76
50 DRAW 1,84,49
60 DRAW 1,135,72
70 DRAW 1,197,89
80 DRAW 1,224,136
90 DRAW 1,295,60
100 DRAW 1,304,130
110 INPUT A$
120 GRAPHIC 0
```

Rad 10: Välj svart som bakgrundsfärg.

Rad 20: Välj vit som förgrundsfärg.

Rad 30: Välj högupplöst grafik och ränsa skärmen.

Rad 40-110 placerar ut individuella pixlar med vald förgrundsfärg.

Rad 110 väntar på att användaren ska trycka Return.

Rad 120 återställer till standardtextläget (40 kolumner).

När du kör programmet kommer du att se Karlavagnen på svart bakgrund, och när du trycker på **Return** återgår du till standardtextläget.

Kommandot `COLOR` sätter någon av de tillgängliga färginställningarna (det första argumentet). Färgerna som kan sättas är bakgrundsfärg (`0`) och förgrundsfärg (`1`). I flerfärgsläge kan även två extra färger sättas (`2` och `3`). I standardtextläget är bakgrundsfärgen heltäckande och förgrundsfärgen gäller för nästa tecken som skrivs. I grafikläge gäller bakgrundsfärgen och förgrundsfärgen för nästa område av 8×8 pixlar. Om du har flerfärgsläge är området egentligen 4×8 pixlar breda pixlar då en logisk pixel är två fysiska pixlar bred, och en bild kan ha två extra färger som är gemensamma för hela bilden. I 80-kolumnsläge är bakgrundsfärgen alltid svart men förgrundsfärgen gäller för nästa tecken som skrivs. Det andra argumentet är val av kulör, och tillgängliga kulörer presenteras på sidan 56.

En linje skapas genom att två koordinater anges. Detta program ritar en diagonal linje över skärmen:

```
10 COLOR 0,16
20 COLOR 1,1
30 GRAPHIC 1,1
40 DRAW 1,0,0 TO 319,199
50 INPUT A$
60 GRAPHIC 0
```

Och för att skapa en polygon, fortsätt att lägga till fler koordinater. Detta program ritar i stället två överlappande trianglar.

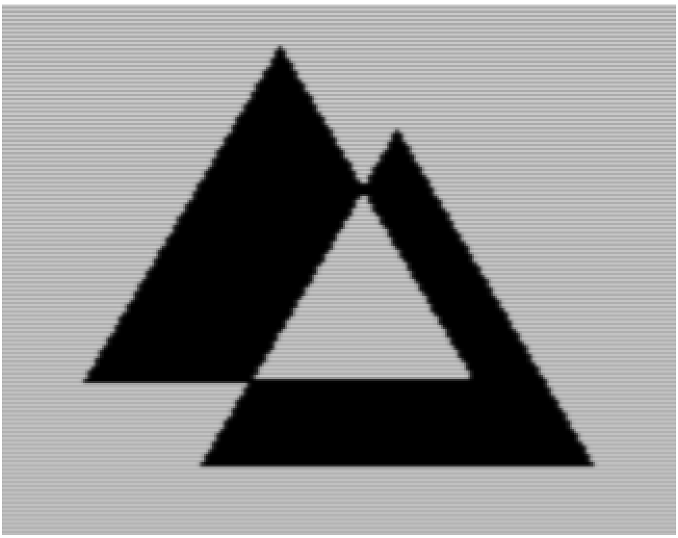
```
10 COLOR 0,16
20 COLOR 1,1
30 GRAPHIC 1,1
40 DRAW 1,100,110 TO 150,30 TO 200,110 TO 100,110
50 DRAW 1,130,130 TO 180,50 TO 230,130 TO 130,130
60 INPUT A$
70 GRAPHIC 0
```

Polygoner (och andra figurer) kan vara fyllda eller lämnas som konturer. Kommandot `PAINT` fyller en figur genom att man anger en punkt inom figuren. Det första argumentet som `PAINT` tar är färgen som ska fyllas (`0`-`3`, precis som `DRAW`). Därefter ska en koordinat som pekar ut ytan som ska fyllas anges. Det sista argumentet anger om ytan som ska fyllas definieras av färgen som angavs i det första argumentet (`0`) eller av en färg som avviker från färgen på den avvikande punkten (1). Här är de två ytorna som inte delas mellan de två trianglarna fyllda.

Om PAINT anropas med två argument, antas dessa vara koordinaten som fyllningen ska utgå ifrån. Om PAINT anropas med tre argument antas dessa vara val av färg att fylla med samt koordinaten. Om PAINT anropas med fyra argument antas dessa vara samma, och dessutom val om den yttre gränsen.

```
10 COLOR 0,16
20 COLOR 1,1
30 GRAPHIC 1,1
40 DRAW 1,100,110 TO 150,30 TO 200,110 TO 100,110
50 DRAW 1,130,130 TO 180,50 TO 230,130 TO 130,130
60 PAINT 1,102,108
70 PAINT 1,228,128
80 INPUT A$
90 GRAPHIC 0
```

Resultatet borde se ut enligt bilden.



Figur 4: Två trianglar där den överlappande delen lämnats utan ifyllnad.

Det fjärde argumentet som kommandot `PAINT` tar, som styr fyllningens yttre gräns, är endast relevant i flerfärgsläge. Detta demonstreras sist i avsnittet om cirklar och bågar på sida 37.

> Tänk på att bildelementens (pixlarnas) form ändras rejält i flerfärgsläge. De blir dubbelt så breda, vilket påverkar vilka koordinater du behöver ange för att få önskat resultat. Kommandot `SCALE` kan användas för att kompensera för detta.

Rektanglar

Denna del kommer att avhandla kommandot `BOX`.

Cirklar och bågar

Denna del kommer att avhandla kommandot `CIRCLE`.

Angående kommandot `PAINT` som introducerades på sida 35, och som syftar till att fylla en yta med färg, så har den två lägen. Detta program illustrerar skillnaden mellan dessa lägen. Skillnaden är endast relevant i flerfärgsläge, vilket programmet använder.

```
10 GRAPHIC 3,1
20 REM"RITA TRE CIRKLAR
30 REM"MED OLIKA FARGER
40 CIRCLE 1,30,50,14,28
50 CIRCLE 2,30,50,10,22
60 CIRCLE 3,30,50,06,17
70 REM"RITA TRE TILL
80 REM"MED OLIKA FARGER
90 CIRCLE 1,70,50,14,28
100 CIRCLE 2,70,50,10,22
110 CIRCLE 3,70,50,06,17
120 REM"FYLL DEN FORSTA
130 REM"FIGUREN MED
140 REM"FARG 1, LAGE 0
150 PAINT 1,30,50,0
160 REM"FYLL DEN ANDRA
170 REM"MED FARG 2, LAGE 1
180 PAINT 2,70,50,1
190 INPUT A$
200 GRAPHIC 0
```

Rad 10 visar flerfärgsläget på din 40-kolumnsmonitor.

Rad 40-60 ritar tre cirklar i olika färger på position 30,50. Den yttersta har färgindex 1, den innersta har färgindex 3.

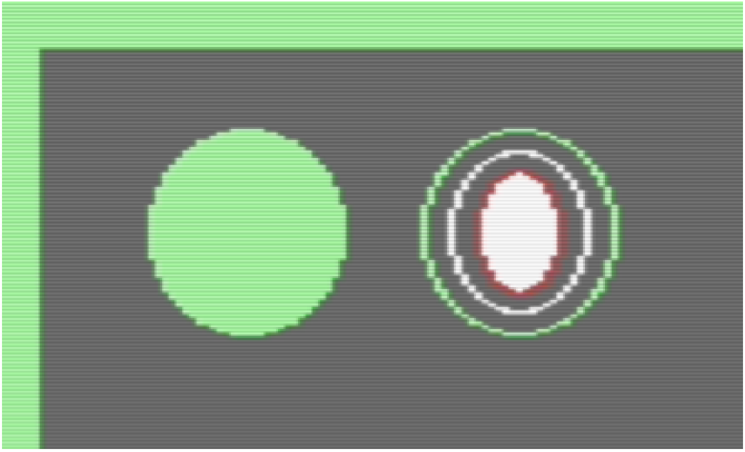
Rad 90-110 ritar tre cirklar i olika färger på position 70,50. Den yttersta har färgindex 1, den innersta har färgindex 3.

Rad 150 fyller den första figuren med färg 1 i läge 0, vilket innebär att fyllningen stannar när den påträffar färg 1.

Rad 180 fyller den andra figuren med färg 2 i läge 1, vilket innebär att fyllningen stannar när den påträffar färg som avviker från bakgrundsfärgen (eftersom bakgrundsfärgen var representerad på koordinat 70,50).

Därefter väntar programmet på ett enterslag.

Resultatet ser ut så här:



Figur 5: Olika lägen för PAINT

Grafikmarkören

Denna del kommer att avhandla grafikmarkörens funktion, samt kommandot `LOCATE`.

Övriga kommandon

Denna del kommer att avhandla `PAINT`, `SCALE` och `WIDTH`.

Stenciler

Denna del kommer att avhandla kommandot `SSHAPE` och kommandot `GSHAPE`.

Flerfärgsgrafik

Denna del kommer att avhandla möjligheten att skapa flerfärgsgrafik med lite lägre upplösning.

KAPITEL 6: SPRITES

Sprites

Detta kapitel kommer att beskriva de utökade möjligheterna att hantera sprites på Commodore 128.

KAPITEL 7: LJUD

Ljud

Commodore 128 har ungefär samma kapacitet till ljud som Commodore 64. I datorn finns en avancerad synthesizer innehållande tre röster och fyra vågformer. Vågformerna är triangel, sågtand, fyrkantsvåg och brus. Fyrkantsvågen har ställbar pulsbredd, vilket ger möjlighet till stora variationer.

Till skillnad från Commodore 64, tillåter Commodore 128 att man använder BASIC-kommandon för att åt datorns ljudkapacitet. Kommandot `SOUND` ger dig möjlighet att spela ljud i någon av de tre kanalerna, i valfri frekvens under valfri tid. Kommandot har även stöd för att böja frekvensen[11] på tonen ljudet som spelar.

Tre argument är obligatoriska. Dessa är val av röst (`1`, `2` eller `3`), val av frekvens (`0` till `65535`) och längd (angivet i sextiondels sekunder). Använd endast dessa tre argument, spelas en rak ton med i en fyrkantsvåg med en neutral inställning på pulsbredden. Följande spelar tonen A under en sekund:

```
SOUND 1,3800,60
```

Du som kan musikteori kanske noterar att 3800 inte är jämt delbart med 220, och det beror på att frekvensangivelsen (det andra argumentet) ligger på en egen skala som inte stämmer med faktisk svängningshastighet.

Argumenten identifieras av sin ordning. Samtliga argument är:

1. Kanal (`1`, `2` eller `3`)
2. Val av frekvens (`0` till `65535`)
3. Längd (antal sekunder genom 60)
4. Frekvensändring (`0` = upp, `1` = ned, `2` = upp och ned)
5. Låg frekvens (vid frekvensändring, också `0` till `65535`)
6. Hastighet vid frekvensändring (stegstorlek, också `0` till `65535`)
7. Vågform (`0` till `3`, se sida 44 i detta kapitel)
8. Pulsbredd (`0` till `4096`, se sida 45 i detta kapitel)

Bara genom att läsa argumentlistan så förstår vi att kommandot `SOUND` är ganska kompetent. Men vi ser också att det saknas en hel del kapacitet innan

[11] På engelska: *Pitch bend*.

vi kan säga att vi har en komplett synthesizer, inte minst ADSR[12]-inställningar.

Argumenten identifieras av sin ordning

Låt säga att jag vill spela samma ljud som tidigare, men jag vill ange vågformens pulsbredd. Igen, pulsbredd kommer att förklaras på sida 45, men för nu behöver du bara förhålla dig till att jag vill ange den. Att anropa `SOUND` med argument 1, 2, 3 (som tidigare) och 8 (pulsbredd), utan att ange något där emellan, innebär att jag anger fem kommatecken för att tala om att det är just det åttonde argumentet jag vill ange.

```
SOUND 1,3800,60,,,,2,50
```

Du bör nu höra ett A, fast med en lite förändrad karaktär på ljudet.

Frekvensändring

Oavsett om du vill böja en ton uppåt, nedåt eller oscillera (upp och ned), så måste du ange den högsta frekvensen som argument 2 och den lägsta frekvensen som argument 5.

Om du vill gå upp en oktav från låga A till höga A, kan du skriva följande kommando:

```
SOUND 1,3800,60,0,1900,32
```

Kanal `1`, hög tonhöjd är `3800`, längd är en sekund, riktning är uppåt, låg tonhöjd är `1900`, frekvensändringens hastighet är `32`.

Ett problem med att böja toner uppåt är att det andra argumentet faktiskt inte bara är den höga frekvensen, utan även startfrekvensen. Ljudet du hör från ovanstående kommando klättrar en oktav (från `1900`), men avslöjar målfrekvensen (`3800`) under en mycket kort stund.

Om vi ändrar riktning från `0` (upp) till `1` (ned) så spelas frekvensändringen helt felfritt, eftersom vi lämnar startfrekvensen till målfrekvensen som är argument 5.

```
SOUND 1,3800,60,1,1900,32
```

---

[12] Attack, Decay, Sustain, Release – se kapitlet om ordförklaringar (appendix B).

Men när jobbet är slutfört, så återstartas det, vilket kan höras i slutet på ovanstående ljud. Det kan man vända till sin fördel, om man vill sätta ljud på t.ex. ett rymdskepp som upprepat skjuter med laser på en fiende:

```
SOUND 1,8000,60,1,3000,700
```

Man kan riktigt se laserstrålarna färdas genom rymden!

Nu har vi testat att böja en ton uppåt (riktning `0`) och nedåt (riktning `1`). Genom att välja att oscillera kan du skapa ljud som låter som larmtoner, som t.ex. detta:

```
SOUND 1,12000,360,2,9000,64
```

Kanal `1`, tonjöjd är `12000`, längden är sex sekunder, riktning är upp och ned, låg tonhöjd är `9000`, frekvensändringens hastighet är `64`.

Vågformer

Vågformen beskriver ljudets mest basala karaktär. Commodore 128 kan återge fyra olika vågformer. Dessa är:

- `0`: Triangel (mjuka ljud som xylofon eller flöjt)
- `1`: Sågtand (en klang liknande dragspel eller gitarr)
- `2`: Fyrkant med variabel pulsbredd (lite vassare ljud som trumpet eller piano)
- `3`: Brus (ofta slagverk)

Dessa fyra vågformer utgör grunden för de olika musikinstrument som din dator kan återge. Detta program spelar upp två sekunder från varje ljud, där fyrkantsvågen återges i sitt grundutförande med en pulsbredd på `2048`:

```
10 FOR A=0 TO 3
20 SOUND 1,3800,120,,,,A
30 SLEEP 3
40 NEXT
```

Rad 10 räknar från 0 till 3, vilket innebär att rad 20 och 30 körs fyra gånger.
Rad 20 spelar en ton i två sekunder som byggs upp av de fyra vågformer som din Commodore kan återge.
Rad 30 pausar och rad 40 upprepar.

När du kör programmet så kommer du att höra fyra vågformer, som spelas upp i turordning. Förutom dessa fyra vågformer, kan din Commodore 128 ändra ljudets karaktär rejält när vågform `2` (fyrkant) används, för fyrkantsvågen har variabel pulsbredd.

Pulsbredd

Pulsbredden anger förhållandet mellan impulsens höga och låga värde. Fyrkantens värde består i sitt ursprungsförhållande av lika lång hög som låg impuls, vilket skapar denna figur:

```
+-------------------------------------------------+
|    ----    ----    ----    ----    ----    ----|
|                                                 |
|----    ----    ----    ----    ----    ----    |
+-------------------------------------------------+
```

Detta motsvara alltså den ursprungliga pulsbredden `2048`. `0` är lägst och `4096` är högst. Du kan skriva in följande kommando för att höra den ursprungliga pulsbredden:

```
SOUND 1,3800,60,,,,2,2048
```

Halverar du talet `2048` (alltså om du anger `1024`) får du följande kurva:

```
+-------------------------------------------------+
|  ------  ------  ------  ------  ------  ------|
|                                                 |
|--      --      --      --      --      --      |
+-------------------------------------------------+
```

Testa genom att skriva:

```
SOUND 1,3800,60,,,,2,1024
```

Ljudet är nästan identiskt med det ljud du hade fått av att istället addera hälften (`1024`) ursprungspulsbredden (`2048`):

```
SOUND 1,3800,60,,,,2,3072
```

Det motsvarar följande vågform:

```
+-------------------------------------------------+
|      --      --      --      --      --      --|
|                                                 |
|------  ------  ------  ------  ------  ------  |
+-------------------------------------------------+
```

Testa gärna att skriva program som varierar pulsbredden!

Ljudstyrka

Kommandot `VOL` påverkar ljudstyrkan både för pågående ljud och efterföljande ljud. `VOL` accepterar ett argument som är ett tal mellan 0 och `15`, där `0` betyder absolut tystnad och `15` betyder högsta möjliga ljud. Ett för högt eller för lågt värde ger felet *illegal quantity*. Följande program använder `VOL` för att oscillera ljudstyrkan:

```
10 SOUND 1,12000,360,2,9000,64
20 FOR T=0 TO 8
30 FOR A=15 TO 0 STEP -1
40 VOL A
50 FOR P=0 TO 10:NEXT
60 NEXT A
70 FOR A=0 TO 15
80 VOL A
90 FOR P=0 TO 10:NEXT
100 NEXT A
110 NEXT T
```

Rad 10 startar ett larmljud.

Rad 20 säger att volymoscillering ska ske 9 gånger (0 till 8).

Rad 30-60 sänker volymen med en kort paus (rad 50).

Rad 70-100 höjer volymen.

Rad 110 hoppar tillbaka till rad 30 så länge T är mindre än eller lika med 8.

Inför de kommande två kapitlen om musik och om avancerade ljudeffekter, så kan det vara bra att veta när datorn spelar ett ljud direkt, och när datorn väntar. Grundregeln är att datorn väntar med att spela upp ett ljud, tills det ljud som spelas ljust nu är färdigt, givet att det nya ljudet begärs i en kanal som redan spelar ett ljud. Det innebär följande:

Kommandot `SOUND` sätter processorn i arbete utan att blockera BASIC-tolken, vilket vi tidigare sett i kapitlet om sprites. Skillnaden här är att om din dator satts i arbete att spela ett ljud i en sekund på en viss kanal, och stöter på ett kommando som ger datorn i uppdrag att spela en ny ton i samma kanal, så väntar BASIC-tolken på att pågående uppdrag ska vara utfört innan nästa uppdrag påbörjas. Vi kan studera detta genom att titta på följande program som spelar ett C dur-ackord i en kanal. Det kommer inte att fungera. Istället för att höra ett ackord, så hör vi en ton i taget spelas upp (C, E och G). Vi väljer full ljudstyrka.

```
10 VOL 15
20 SOUND 1,8900,120,,,,1
30 SOUND 1,11100,120,,,,1
40 SOUND 1,13500,120,,,,1
```

Orsaken är att kanal 1 inte kan spela något annat än C innan den är färdig. Och när den är färdig, och börjar spela ett E, så kan den inte spela nästa ton innan den har spelat färdigt E. Lösningen är att spela varje ton i C dur-ackordet i var sin kanal, så här:

```
10 VOL 15
20 SOUND 1,8900,120,,,,1
30 SOUND 2,11100,120,,,,1
40 SOUND 3,13500,120,,,,1
```

Om nya instruktioner att spela ljud inkommer, väntar din dator med att utföra dem tills ackordet är färdigt. Men det finns undantag, vilket vi tittar på i kapitlet om avancerade ljudeffekter.

KAPITEL 8: MUSIK

Musik

Detta kapitel kommer att beskriva de utökade möjligheterna att skapa syntetisk musik på Commodore 128.

KAPITEL 9: AVANCERADE LJUDEFFEKTER

Avancerade ljudeffekter

Detta kapitel kommer att beskriva möjligheterna att skapa avancerade ljudeffekter på Commodore 128.

KAPITEL 10: ANVÄNDARINTERAKTION

Användarinteraktion

Detta kapitel kommer att beskriva de utökade möjligheterna att till användarinteraktion på Commodore 128.

KAPITEL 11: 80-KOLUMNSLÄGE

80-kolumnsläge

Commodore 128 kan startas i 40-kolumnsläge (vilket motsvarar det läge som Commodore 64 alltid körs i) och det mer textorienterade 80-kolumnsläget. Vid uppstart tittar datorn på om knappen **40/80 display** är nedtryckt eller ej, och väljer läge därefter.

Om knappen inte är nedtryckt, startar Commodore 128 i 40-kolumnsläge och bilden skickas ut genom kompositporten[13] (5-polig DIN) eller RF-porten. Om knappen är nedtryckt, startar datorn i 80-kolumnsläge och bilden skickas ut genom RGBI-anslutningen. (En monokrom bild skickas dessutom ut genom den 5-poliga DIN-kontakten.)

80-kolumnsläget är främst till för att köra mer avancerade textbaserade program för ordbehandling och kalkyl, så alla BASIC-kommandon som har med grafik att göra, har ingen synlig effekt[14] i det läget (såvida du inte har två bildskärmar inkopplade). När jag kör mer avancerade program, som t.ex. SwiftCalc, eller CP/M-program som t.ex. Microsoft BASIC eller Turbo Pascal, så föredrar jag ändå 80-kolumnsläget eftersom det då är värt väldigt mycket att ha gott om utrymme på skärmen.

Färgpaletten ser lite annorlunda ut i 80-kolumnsläge. Tabellen på nästa sida visar vilken färg varje kod representerar i respektive läge. Dessutom är det bra att veta att bakgrundsfärgen och borderfärgen i 80-kolumnsläge alltid är svart. Kommandot `COLOR` påverkar endast den grafik som skickas till kompositporten eller RF-porten.

[13] Se manualen till Commodore 128 för information om vilka portar som finns tillgängliga och vilken som gör vad.
[14] Om du har två monitorer, en som är inkopplad i RGBI-porten och en som är inkopplad i DIN-porten eller RF-porten, så kommer du faktiskt se effekten av de grafikkommandon som körs i den sistnämnda monitorn. Detta är inte dokumenterat av Commodore, förmodligen för att deras monitorer är så dyra att de flesta ändå bara äger högst en enda.

Tabellen visar färg per färgkod i respektive läge. Skillnaderna (hälften av färgerna) är markerade med en stjärna (*).

Färgkod	**Färg i 40-kolumnsläge**	**Färg i 80-kolumnsläge**
1	Svart	Svart
2	Vit	Vit
3	Röd	Mörkröd*
4	Turkos	Ljusturkos*
5	Lila	Ljuslila*
6	Grön	Mörkgrön*
7	Blå	Mörkblå*
8	Gul	Ljusgul*
9	Orange	Mörklila*
10	Brun	Brun
11	Ljusröd	Ljusröd
12	Mörkgrå	Mörkturkos*
13	Grå	Grå
14	Ljusgrön	Ljusgrön
15	Ljusblå	Ljusblå
16	Ljusgrå	Ljusgrå

Om du arbetar med dubbla skärmar så växlar du mellan 40- och 80-kolumnskärmen genom att skriva:

```
SYS 49194
```

Om du skulle råka göra detta anrop utan att ha en andra skärm inkopplad, skriv samma kommando igen (i blindo) och tryck **Enter**, så återgår din enda skärm till att vara aktiv.

Innehållet på den skärm du lämnar, bevaras tills du återkommer. Text som skrivs ut (med PRINT eller med tangentbordet) hamnar på den skärm som är aktiv just nu. Grafik som skapas med datorns inbyggda BASIC-kommandon, skickas alltid till 40-kolumnsskärmen.

För att läsa av vilket läge du befinner dig, använd funktionen `RGR(0)`. *(Argumentet `0` har ingen betydelse, men funktioner måste ha ett argument enligt syntax-reglerna.)* Funktionen ger ett heltal som svar enligt:

0 till 4 betyder att du befinner dig på 40-kolumnskärmen, 5 till 9 betyder att du befinner dig på 80-kolumnsskärmen. Vilket tal som visas, beror på vilket grafikläge som är aktivt på 80-kolumnsskärmen:

`0` (eller `5` om du befinner dig i 80-kolumnsläge) betyder att 40-kolumnsskärmen är i textläge.

`1` (eller `6` om du befinner dig i 80-kolumnsläge) betyder att 40-kolumnsskärmen visar högupplöst grafik.

`2` (eller `7` om du befinner dig i 80-kolumnsläge) betyder att 40-kolumnsskärmen befinner sig i split screen-läge mellan text och högupplöst grafik.

`3` (eller `8` om du befinner dig i 80-kolumnsläge) betyder att 40-kolumnsskärmen visar lågupplöst färggrafik.

`4` (eller `9` om du befinner dig i 80-kolumnsläge) betyder att 40-kolumnsskärmen befinner sig i split screen-läge mellan text och lågupplöst färggrafik.

Om det enda du vill veta är huruvida 80-kolumnsskärmen är aktiv eller ej, kontrollera om `PEEK(215)` ger `128` som svar – då är den aktiv. Följande program skriver ut vilken skärm som är aktiv, och låter användaren växla. Om du endast har en skärm, kommer bilden att frysa om du väljer att växla. Återgå genom att trycka **J** följt av **Return**. Har du två skärmar kommer du hoppa mellan skärmarna.

```
10 IF RGR(0)<5 THEN PRINT "DU ÄR I 40-KOLUMNSLÄGE." :
ELSE PRINT "DU ÄR I 80-KOLUMNSLÄGE."
20 INPUT "VILL DU ANDRA (J/N)"; A$
30 IF A$="J" THEN SYS 49194:PRINT "NU ÄR VI PÅ DENNA
SKÄRM!":GOTO 10
```

Radbryten på `10` och `30` är av platsskäl.

KAPITEL 12: COMMODORE BASIC 7.0 DOS

Commodore BASIC 7.0 DOS

Diskoperativsystemet (DOS) är implementerat i din diskdrive, men Commodore 128 har en rik uppsättning kommandon som låter dig utföra avancerad filhantering direkt från BASIC. För att följa med i detta kapitel behöver du ha tillgång till en diskdrive av modell 1540, 1541, 1570, 1571, 1580 eller 1581[15]. Dessutom behöver du ha en floppydisk som inte innehåller något som du inte kan avvara, för det som ligger lagrat på den diskett du använder när du följer med i övningarna kommer att försvinna. Alla praktiska frågor kring hur en diskdrive fungerar och hur disketter används finns i din manual.

Du förbereder din diskett med kommandot `HEADER`. Processen kallas normalt för formatering, och innebär att information om vad som ligger på disketten skrivs ned, tillsammans med ett namn och ett ID-nummer. Informationen som skrivs ned om diskettens innehåll säger att disketten är tom, vilket är orsaken till att du behöver ha en ny diskett eller en gammal diskett vars innehåll du kan avvara för att testa.

Det argument som behöver anges för att `HEADER` ska göra jobbet, är diskettens namn. 16 tecken är reserverade för namnet, och skickar man in fler tecken uppstår felet *string too long*.

```
HEADER "NY DISKETT"
```

Efter att detta kommando har körts, är din disk tömd på innehåll och redo att spara ditt data. Om något fel uppstår under någon process, visar din diskdrive det genom att pulsera med grönt ljus. Alltså, om du ser pulserande grönt ljus, har den senaste uppgiften du gav din diskdrive misslyckats.

[15] Jag använder en Commodore 1571, som är den bästa enheten för att hantera 5,25"-disketter. Commodore 1580 och framåt jobbar med de mer moderna 3,5"-disketterna.

Detta kapitel tar upp:

- Hur formaterar man en diskett (`HEADER`)
- Vad finns på disketten (`DIRECTORY` eller `CATALOG`)
- Hur man sparar och återkallar ett BASIC-program
- Hur man sparar och återkallar annan binär data, som t.ex. sprites
- Om ett fel uppstår

HEADER

Kommandot `HEADER` formaterar alltså en diskett (se ovan). Förutom diskettens namn (16 tecken) kan kommandot ta emot ett ID (två tecken), ett drivarnummer och ett enhetsnummer. Kommandot använder sig av prefix för att hålla reda på vilken inmatning som är vad. För ID används `I`, för drivarnummer används prefixet `D` och för enhetsnummer används prefixet `U`.

För snabbformatering, utelämna ID. Befintligt ID återanvänds och diskettens innehållstabell rensas utan att diskdriven gör någon översyn över ytan i övrigt.

Enhetsnummer beror på hur din diskdrive är konfigurerad, men normalt har den enhetsnumret `8`. Drivarnumret är `0` för den första diskdriven och `1` för en eventuell annan diskdrive. Om det är disketten i den första diskdriven som ska formateras så kan du välja mellan att specificera `D0` eller `U8` (om den är konfigurerad som enhet `8`), och det går att ange båda. Följande exempel ger en full formatering på disketten i enhet `8` med ID `YT`:

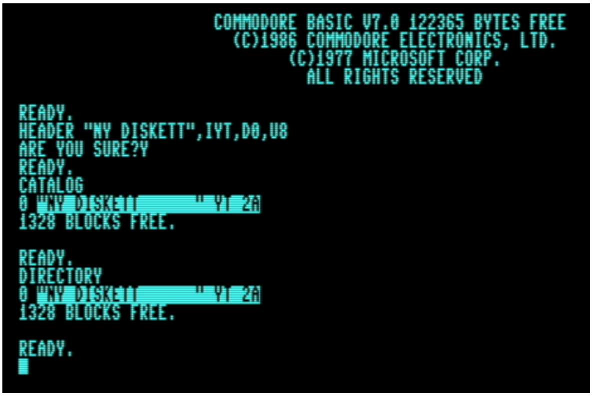
```
HEADER "NY DISKETT",IYT,D0,U8
```

Eftersom ID angavs, tar formateringen en ganska lång stund att genomföra, men nu har du en diskett som är redo att laborera med!

Vad finns på disketten?

Nu är disketten tom, men helt oavsett så används kommandot `DIRECTORY` (som är synonymt med `CATALOG`) för att presentera innehållet på en diskett.

Detta är en radikal förbättring från förutsättningarna som Commodore gav till t.ex. sina Commodore 64-användare. Där behövde man skriva avancerade DOS-kommandon, eller helt enkelt offra allt man hade i BASIC-minnet, bara för att få veta vad som finns på den diskett som sitter i diskettstationen!



Figur 6: Tom diskett.

Detta är en inte så liten förbättring mot Commodore BASIC 2.0 second release, där man lydigt skrev `LOAD "$",8` på bekostnad av eventuella BASIC-program!

Har du flera diskettstationer kan du specificera vilken, antingen genom att ange drivarnummer (`0` eller `1`) eller enhetsnummer (`8`, `9`, `10` eller `11`) eller båda. Givet att din andra diskettstation har enhetsnummer `9`, kommer samtliga dessa alternativ visa innehållet på disketten i den andra diskettstationen:

```
DIRECTORY D1

DIRECTORY U9

DIRECTORY ON U9

DIRECTORY D1,U9

DIRECTORY D1,ON U9
```

Du har även möjlighet att filtrera resultatet på filnamn, där asterisk (`*`) används som joker. Filtret anges efter drivare och enhet (om drivare och enhet anges). Detta listar endast filer vars filnamn börjar på bokstaven `A`:

```
DIRECTORY "A*"
```

Möjligheten att filtrera är praktisk om du jobbar med en diskett som innehåller fler filer än vad som får plats att visa på skärmen. Andra kommandon, som t.ex. `DLOAD` har också stöd för jokertecken, så det kan vara bra att hålla i huvudet vad man kan göra med jokertecken. Förutom asterisk (som används för att söka efter en filnamnets inledning), kan frågetecken användas (`?`) för okänt tecken. Följande exempel listar alla filer vars filnamn är exakt fem tecken långt:

```
DIRECTORY "?????"
```

Och följande exempel listar alla filer vars filnamn är fem tecken långt, om det sista tecknet är bokstaven `N`:

```
DIRECTORY "????N"
```

Medan detta listar alla filer som börjar på `B`, oavsett vilken bokstav som är den sista, för när `*` är påträffad, ignoreras allt annat.

```
DIRECTORY "B*N"
```

Om filnamnet lagras i en variabel, måste variabeln anges inom parentes. Detta fungerar:

```
A$="B*"

READY.
DIRECTORY(A$)
```

Om disketten innehåller några filer som börjar på bokstaven `B`, kommer dessa nu att visas på skärmen.

> Hur man anger vilken disk man vill titta på och hur man använder jokertecken är återkommande för flera kommandon i Commodore BASIC 7.0.

Ladda och spara BASIC-program

Kommandot för att läsa in ett BASIC-program från disk heter `DLOAD`. Att ladda in ett BASIC-program på disk, sker på bekostnad av det BASIC-program som finns i minnet, och eventuellt tillstånd från tidigare körningar. Likt `NEW` rensar `DLOAD` även variabelminnet.

Syntaxen är precis som `DIRECTORY`, vilket innebär att:

- Filnamnet på filen som ska laddas, kan anges med jokertecken, vilket laddar första filen som passar in
- Drivarnummer och/eller enhetsnummer kan anges om man vill ladda in ett BASIC-program från en annan enhet än den första
- Om filnamnet ligger i en variabel, måste det anges inom parentes

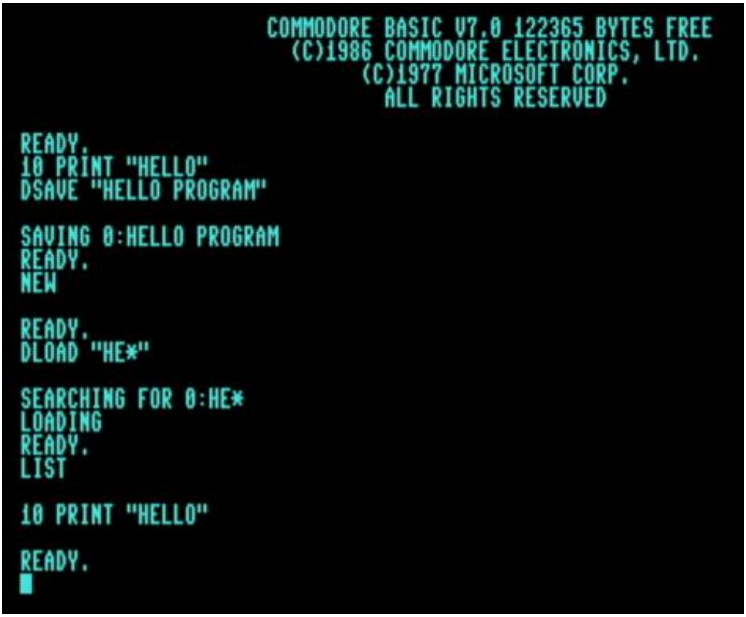
Att ladda ett BASIC-program från en diskett som finns i enhet `8` (vilket är normalt om du bara har en diskettstation) och att denna har drivarnummer `0` (vilket också är normalt för den som bara har en diskettstation) kan göras på följande vis:

```
DLOAD "PROGRAMNAMN",D0

DLOAD "PROGRAMNAMN",U8

DLOAD "PROGRAMNAMN",ON U8

DLOAD "PROGRAMNAMN",D0,U8

DLOAD "PROGRAMNAMN",D0,ON U8
```

Och om programmets namn ligger i en variabel, ska variabelns namn omges av parentes:

```
A$="PROGRAMNAMN"

READY.
DLOAD(A$),ON U8
```

Till skillnad från `DIRECTORY` rensar `DLOAD` BASIC-minnet, alla pekare och all status. Men om inläsningen från disk gick bra, har du framgångsrikt återkallat ett BASIC-program från disk! Om du läst avsnittet om `HEADER`, så har du säkert förberett en diskett som kan hålla BASIC-program. Om disketten är tom, kan du enkelt producera en BASIC-fil för att ha något att laborera med. I detta exempel har jag en förberedd diskett i drivare `1` med enhet `8`:



Figur 7: DSAVE och DLOAD.

Här har ett mycket enkelt program skapats, och sparats med `DSAVE`, för att sedan återkallas med `DLOAD`. Syntaxen för `DLOAD` är alltså densamma som för `DIRECTORY`. För `DSAVE` är skillnaden att jokertecken inte kan användas.

> `DSAVE` sparar alltså det BASIC-program som för tillfället finns i minnet till en fil, så att det senare kan återkallas med `DLOAD`.

> `DLOAD "FILNAMN",ON U8` motsvarar `LOAD "FILNAMN",8,0` eller `LOAD "FILNAMN",8` i BASIC 2.0 second release. `DSAVE "FILNAMN",ON U8` motsvarar `SAVE "FILNAMN",8,0` eller `SAVE "FILNAMN",8` i BASIC 2.0 second release.

Ladda och spara data i datorns arbetsminne

Kommandot `BLOAD` används för att hämta data, antingen till den adress den sparades ifrån från början, eller till en önskad adress. Kommandot `BSAVE` skriver önskad del av datorns minne till diskett.

För att förstå syntaxen behöver du läsa de tidigare inläggen om `DIRECTORY` och `DLOAD`/`DSAVE`. Det klargör hur du anger drivarnummer och/eller enhetsnummer, samt vilka regler som gäller för att ange filnamn (där `BLOAD` har stöd för jokertecken, till skillnad från `BSAVE`).

Utöver filnamn och drivarnummer och/eller enhetsnummer, behöver `BSAVE` (som sparar data som finns i minnet) veta vilket data som ska sparas. Detta anges med prefixet `P` enligt följande:

```
BSAVE "FILNAMN",D0,P[STARTADRESS] TO P[SLUTADRESS]
```

Ett fungerande exempel på BSAVE kommer nedan.

Vi vet att datorns inbyggda sprite-editor (`SPRDEF`) redigerar sprites på adress `3584` och framåt, och vi vet att en sprite består av 63 bytes (byte 64 från startadressen används inte). Det säger oss att den första spriten som redigeras med `SPRDEF` kan sparas i en fil enligt följande:

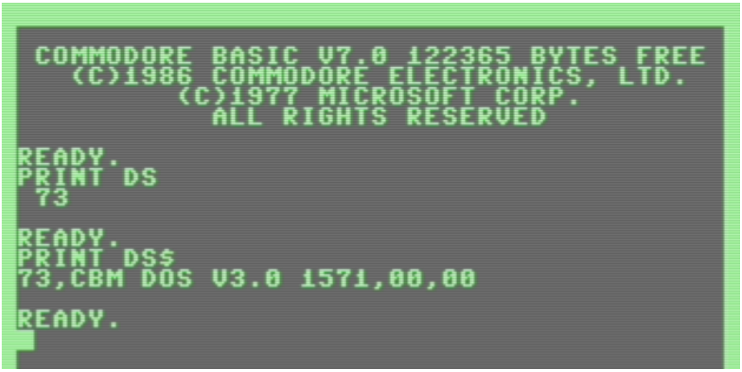
```
BSAVE "SPRITE1", D0,P3584 TO P3647
```

> Hur du skapar en sprite med `SPRDEF` avhandlas i kapitel 6.

Om ett fel uppstår

Du ser att ett fel har uppstått genom att din diskettstations aktivitetsindikator blinkar konstant. Felkoden lagras i systemvariabeln `DS` och lite mer information lagras som text i `DS$`.

Ursprungsvärdet i `DS` är `73`, vilket är en platshållare för versionsnumret på DOS samt diskettstationens modell. Läser du av `DS$` innan ett fel har uppstått, så får du versionsnummer och modellnamn i klartext. I mitt fall ger `DS$` värdet `73,CBM DOS V3.0 1571,00,00`.



Figur 8: Ursprungsvärdet i DS$.

Om en lyckad DOS-operation har utförts, sätts `DS` till `0` och `DS$` till `00, OK,00,00`.

Låt säga att du försökt att läsa en diskett som är trasig på spår 1, sektor 16, skulle DS sättas till 66 och DS$ till `66,ILLEGAL TRACK OR SECTOR,01,16`[16].

Du ser att ett DOS-relaterat fel har uppstått genom att diskettstationens aktivitetslampa (typiskt grön) blinkar snabbt. När felet lästs av, antingen från DS eller DS$, släcks lampan.

Om du arbetar med filer och kan avvara vad som finns lagrat i BASIC-minnet, kan detta program köras när helst diskettstationen börjar blinka (nästa sida):

[16] Om du inte äger någon trasig diskett men vill återskapa detta fel, testa att läsa innehållet på systemdisketten för CP/M från Commodore BASIC.

```
10 IF X=0 THEN PRINT "STATUS: OK":END
20 MP=INSTR(X$,",",1)
30 TP=INSTR(X$,",",MP+1)
40 SP=INSTR(X$,",",TP+1)
50 M$=MID$(X$,MP+1,TP-MP-1)
60 IF X=73 THEN PRINT "STATUS: OK":PRINT"SYSTEM: ";:PRINT
M$:END
70 T$=MID$(X$,TP+1,2)
80 S$=MID$(X$,SP+1,2)
90 PRINT "STATUS: "; M$
100 PRINT "TRACK:  "; T$
110 PRINT "SECTOR: "; S$
```

Tänk på att radbrytningen på rad `60` är av platsskäl, och ska inte vara en radbrytning i programmet. I 40-kolumnskäge kommer det ske en radbrytning på rad `60` även vid inmatning, men tryck inte **Return** förrän du skrivit färdigt raden.

> Programmet kommer inte att visa korrekt status om det inte är inladdat i minnet innan DOS-operationerna utförs. Skriv in (eller läs in) programmet innan du börjar jobba med DOS, så att det finns när ett fel uppstår.

KAPITEL 13: COMMODORE 64-LÄGE

Commodore 64-läge

Din manual beskriver hur du startar din Commodore 128 i olika lägen, men du kan växla från standardläget till Commodore 64-läget med kommandot `GO64`. I direktläge frågar datorn om du verkligen vill växla till Commodore 64-läget (ditt BASIC-program går förlorat).

Du får frågan `ARE YOU SURE?`, och ett svar som börjar på bokstaven `Y` kommer att uppfattas som en bekräftelse.

Om `GO64` används i ett program, växlar datorn läge direkt, utan varning. Det går att infoga ett blanksteg mellan `GO` och `64`, och som kuriosa kan det vara bra att veta att `64` kan ersättas med ett uttryck som utvärderas till just `64`. Om du t.ex. tilldelar variabeln `AT` värdet `64`, går det bra att växla till 64-läge genom att skriva `GOAT`, eller om du tilldelar `HO` värdet `64`, kan du växla till 64-läge genom att skriva `GO HOME`.

Om du från direktläget vill växla till 64-läget, utan att få frågan om du är säker, kan du skriva:

```
SYS 65357
```

Eller:

```
SYS 57931
```

Men som sagt, i runtime-läge så gör `GO64` jobbet precis lika bra, eftersom kontrollfrågan inte ställs i runtime-läge. Det är endast i direktläget som det finns en skillnad.

Att växla till C64-läge innebär att ditt BASIC-program (som inte är sparat) går förlorat, oavsett hur du gör det.

Om du vill programmera BASIC i 64-läge, läs gärna boken *Commodore BASIC 2.0 second release* från 2021.

KAPITEL 14: CP/M

CP/M

Som jag nämnde på sida 18 så är Commodore 128 på sätt och vis tre datorer i en, tack vare tre olika lägen. Ett av dessa lägen är CP/M-läget. CP/M är ett avancerat diskoperativsystem som kräver en del inkopplad hårdvara för att fungera bra. Z80-processorn, tillsammans med tillräckligt mycket RAM-minne, finns inbyggt i datorn. Du behöver ha minst en diskdrive, men helst två stycken för att inte behöva byta diskett så ofta.

CP/M kräver inte att du har en monitor, för det kan användas i 40-kolumnsläge på din tv, men det är inte särskilt praktiskt. Så helst behöver du ha en monitor inkopplad och ha tangenten **40/80 display** är nedtryckt vid uppstart. Kapitlet innehåller dessa ämnen:

- Starta CP/M
- Grundläggande användning

Avsnittet om grundläggande användning visar hur man kan arbeta med filer i CP/M under lite olika förutsättningar.

Starta CP/M

Om du äger en monitor, se till att knappen **40/80 display** är nedtryckt. Knappen stannar i nedtryckt läge när man trycker på den en gång, och man släpper upp den genom att trycka på den en gång till. Om du inte äger någon monitor, lämnar du **40/80 display**-knappen uppe.

När du köpte din dator, fick du med två stycken CP/M-disketter. Den som är märkt *CP/M System Disk* ska vara inmatad i din primära diskdrive (normalt enhet 8) när du slår på datorn. Är datorn redan påslagen i C128-läge så kan du köra kommandot `BOOT` eller trycka på **reset-knappen** på datorns högra sida.

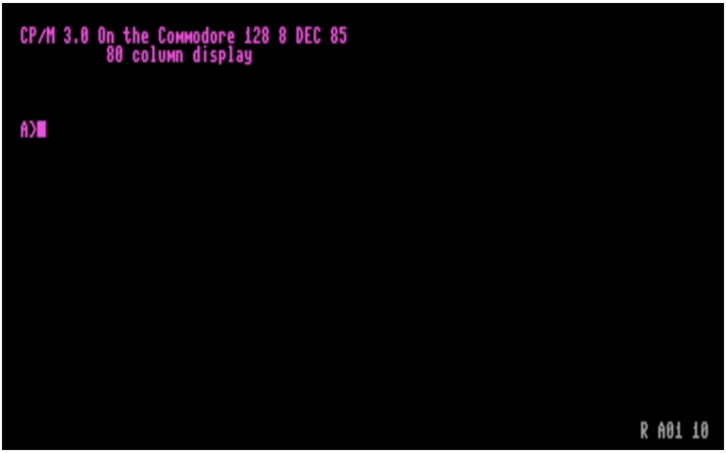
Om du slår på datorn med disketten i, möts du av den vanliga startbilden, men i stället för budskapet `READY` så får du information om att CP/M håller på att starta. När allt är färdigt, möts du av en prompt som säger `A>`. Detta är typiskt för diskoperativsystem, för de vill berätta vilken disk som ditt nästa kommando kommer att utföras mot. I CP/M heter diskettstationerna inte längre enhet `8`, enhet `9`, och så vidare. Inte heller heter de drivare `0`, drivare `1` och så vidare. I CP/M heter de i stället `A`, `B`, `C`, och så vidare.

Det du möts av om starten av CP/M har varit framgångsrik visas på bilden på nästa sida. Förutom `A>` uppe till vänster, så kan du se att bilden visar `R A01 10` längst ner till höger. `A>` (uppe till vänster) betyder att det kommando du tänker exekvera agerar mot disketten i din primära diskdrive. `R` (nere till höger) berättar att den senaste operationen är en läsoperation. Om du istället ser `W` är den senaste operationen en skrivning till disk.

Det du ska tänka när du tittar på koden nere till höger är:

1. Läs eller skriv? Alltså `R` eller `W`.
2. Vilken enhet (`A`, `B`, `C`, och så vidare) ihopskrivet med det senast använda spåret (track `01` om inget annat sägs).
3. Senast använda sektor (`10`) i aktuellt spår (`01`).

Innan du gjort något, är det just `R A01 10` som troligtvis står skrivet i hörnet, vilket visas på nästa sida.



Figur 9: Startskärmen för CP/M.

Den bästa CP/M-dokumentationen som finns att använda till din Commodore 128 finns att läsa på nätet[17]. Men om du saknar erfarenhet och vill komma igång, så bjuder jag på en överflygning här.

I korthet: Program som måste läsas in från disk, används effektivt på system som har två diskettstationer anslutna. Program betraktas som en del av operativsystemet fungerar utmärkt även när bara en diskettstation finns tillgänglig.

Figur 10: Boot-disketten för CP/M.

[17] http://www.cpm.z80.de/manuals/cpm22-m.pdf

Grundläggande användning

Varje gång du skriver något i CP/M[18], utan att ha ett program igång, antar datorn att du försöker exekvera ett kommando. Ett kommando är i princip ett datorprogram som utför det du vill ha gjort, och som kan hantera eventuella parametrar (argument) du bifogar. Kommandot, eller programmet, kan finnas i minnet, eller behöva läsas in från diskett. För att arbeta effektivt i CP/M behöver man ha minst två diskettstationer, men detta exempel fungerar även om du bara har en diskdrive.

På CP/M-språk pratar man om program som är eller inte är *transient*. De kommandon som *inte* benämns som transient (`DIR`, `TYPE`, m.fl.), är de kommandon som är en del av operativsystemet, och som hålls i minnet, redo att exekveras när de anropas. De kommandon som *är* transient (`ED`, `ASM`, m.fl.), behöver finnas redo att hämtas från disk.

Som exempel på grundläggande användning visar jag

- steg för att skapa en textfil när endast en diskdrive finns tillgänglig
- steg för att skapa en textfil på en separat disk när två diskdrives finns tillgängliga

Första exemplet syftar till att visa hur du kan skapa en textfil på en diskett, och kräver inte att du har några andra disketter tillgängliga än **systemdisketten för CP/M** (märkt *CP/M System Disk*) och **verktygsdisketten** (märkt *CP/M System User Utilities Disk*).

Om du har tillgång till två diskettstationer, kan du ha systemdisketten (eller verktygsdisketten) i den ena, och en tom diskett (eller en diskett vars data du kan avvara) i den andra, och därmed skapa en textfil utan att påverka systemdisketten eller verktygsdisketten, vilket avhandlas i det andra exemplet. Men stegen i det första exemplet visar hur man skapar en textfil utan att ha tillgång till två diskettstationer, vilket innebär att textfilen som skapas hamnar på själva verktygsdisketten.

[18] CP/M är en förkortning av Control Program for Microcomputers.

Steg för att skapa en textfil när endast en diskdrive finns tillgänglig

Starta CP/M genom att mata in systemdisketten och slå på datorn. Vänta några ögonblick på att systemet ska laddas in. Det är färdigt när din skärm ser ut som på bilden på sida 74.

Kontrollera att verktygsdisketten inte är skrivskyddad och mata därefter in den i diskdrive A (8). Den innehåller texteditorn `ED`. `ED` kan startas genom att man bara skriver `ED`, men då kommer programmet att fråga efter en textfil att läsa in. Genom att starta `ED` och tillhandahålla ett filnamn, förstår programmet att den ska skapa en ny fil. Skriv:

```
ED test.txt
```

Vänta tills `ED` har lästs in klart. Du ska nu se `NEW FILE`, och att textmarkören är placerad efter en asterisk (`*`).

Kommandot `I` betyder att du vill infoga textrader. Skriv `I` och tryck **Return**. Nu kan du mata in text.

> I CP/M har gemener och versaler olika betydelse. Att använda versal när man skriver kommandot för att infoga text innebär att man konverterar den infogade texten till versaler.

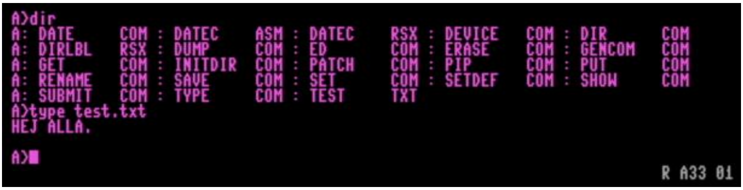
Du ser en siffra till vänster som anger vilken rad du befinner dig på. Kontrollera att du har angivit texten korrekt, innan du trycker **Return** för att gå till nästa rad.

När inmatningen är färdig, tryck **Control+Z**. Du ska nu ha textmarkören vid en asterisk igen.

Figur 11: En textrad infogad i en ny textfil.

För att spara och avsluta, skriv `E` och tryck **Return**. (Om du i stället hade velat avsluta utan att spara används kommandot `Q`, och då måste du godkänna att du vill avsluta genom att trycka **Y**).

Du kan bekräfta att filen `test.txt` är sparad genom att skriva `DIR`. Och vill du se innehållet, använd kommandot `TYPE test.txt` för att läsa innehållet i filen.



Figur 12: Innehållet i test.txt har skrivits ut på skärmen.

Notera att jag skrev "`Hej alla.`" när jag skapade filen, men att filen innehåller "`HEJ ALLA.`", eftersom jag använde versalt `I` för *insert*.

Vill du radera filen från verktygsdisketten, skriv:

```
ERA test.txt
```

Om du har redigerat filen `test.txt`, kan du eventuellt även behöva radera en fil som heter `test.bak`.

Skapa en textfil på en separat disk när två diskdrives finns tillgängliga

För att följa med i detta exempel behöver du ha

- två diskdrives
- systemdisketten med CP/M
- verktygsdisketten
- en tom diskett (eller en diskett vars innehåll du kan avvara)

Starta CP/M genom att mata in systemdisketten och starta om datorn (eller skriv `BOOT`). Vänta några ögonblick på att systemet ska laddas in. Det är färdigt när din skärm ser ut som på bilden på sida 58.

Medan systemdisketten fortfarande är inmatad, skriv:

```
FORMAT
```

Programmet `FORMAT`, som ligger på systemdisketten, formaterar en diskett i CP/M-format, så att den kan användas för att lagra filer från CP/M.

Invänta information om att `C128 FORMAT PROGRAM` är inladdat.

Plocka ut systemdisketten och mata in en disk som du vill använda för att laborera. **Allt eventuellt befintligt innehåll på disketten kommer att förloras.**

Därefter, använd piltangenterna nere till höger för att välja önskat format. Använder du en Commodore 1571, vill du troligen att alternativet `C128 double sided` ska vara markerat. När du valt ett alternativ, tryck **Return**.

Programmet uppmanar dig att stoppa in disketten som ska formateras (vilket du redan har gjort), och att du ska skriva `$` (alltså **Shift+4**) när du är redo. Om du vill avbryta, tryck på någon annan tangent, t.ex. **Escape**.

Din diskdrive kommer att arbeta en stund. Du får information om att disketten formateras och att lite information skrivs till den. När allt är klart frågar programmet om du vill formatera en annan diskett. Neka genom att trycka **N**.

Programmet avslutas. Du kan direkt konstatera att disketten fungerar och är tom genom att skriva `dir`. Datorn ger `No File` som svar.

Nu har du en tom och nyformaterad disk i enhet A. Flytta den till enhet B och infoga verktygsdisketten i enhet A.

Starta `ED` genom att skriva:

```
ED B:test.txt
```

Nu kan du redigera en textfil som lagras på din avsedda diskett som du nyss formaterade, enligt föregående stycke. När du har arbetat färdigt, kommer verktygsdisketten fortfarande vara oförändrad.

APPENDIX A: FELSÖKNING

Appendix A: Felsökning

Detta kapitel handlar inte om generell felsökning, utan om felsökning av ditt BASIC-program. Commodore 128 innehåller en del avancerade funktioner som underlättar felsökningen av ett program (debugging). Bland dessa hittar vi `EL`. Denna bilaga beskriver funktionerna och hur de kan vara till nytta.

EL

Så snart ett programfel uppstår laddas systemvariabeln `EL` med det senaste radnumret där felet uppstod. Följande program kommer att orsaka ett fel på rad `20`, eftersom Commodore 128 inte tillåter division med 0.

```
10 PRINT "EN DIVISION"
20 PRINT 10/0
```

När detta program startats med RUN ger datorn följande svar:

```
EN DIVISION

?DIVISION BY ZERO ERROR IN 20
READY
```

Om du nu läser av EL får du svaret 20, eftersom felet inträffade på svar 20.

```
PRINT EL
 20

READY.
```

Om du läser av `EL` innan något fel har inträffat, får du svaret `65535`. Du vet att inget fel har uppstått, eftersom ett giltigt radnummer på Commodore 128 är ett tal mellan `0` och `63999`. Fel som inträffar i direktläge ändrar inte värdet i `EL`. Om rättar ett fel i ett program så att inget fel uppstår, och därefter kör programmet så återställs värdet av EL till `65535`.

ER

Systemvariabeln `ER` innehåller normalt värdet `-1`, men om ett fel skulle uppstå laddas `ER` med felkoden för det senaste felet, och håller det värdet tills antingen ett nytt fel uppstår eller tills variabelminnet rensas. De kommandon som rensar variabelminnet är `RUN`, `NEW` och `CLR`. Trots att `ER` innehåller `-1` när inget fel har uppstått, så finns det ingen felkod 0 i datorn. De fel som kan uppstå har kod `1`, `2`, `3`, och så vidare upp till `41`. Funktionen `ERR$` ger namnet på dessa fel. Alla fel som kan uppstå listas på nästa sida.

Lista över BASIC-felkoder

Dessa felkoder kan rapporteras i BASIC på Commodore 128:

Kod	Namn	Förklaring
1	TOO MANY FILES	
2	FILE OPEN	
3	FILE NOT OPEN	
4	FILE NOT FOUND	
5	DEVICE NOT PRESENT	
6	NOT INPUT FILE	
7	NOT OUTPUT FILE	
8	MISSING FILE NAME	
9	ILLEGAL DEVICE NUMBER	
10	NEXT WITHOUT FOR	
11	SYNTAX	
12	RETURN WITHOUT GOSUB	
13	OUT OF DATA	
14	ILLEGAL QUANTITY	
15	OVERFLOW	
16	OUT OF MEMORY	
17	UNDEF'D STATEMENT	
18	BAD SUBSCRIPT	
19	REDIM'D ARRAY	
20	DIVISION BY ZERO	
21	ILLEGAL DIRECT	*Du har skrivit ett kommando i direktläge som endast tillåts i program.*
22	TYPE MISMATCH	
23	STRING TOO LONG	
24	FILE DATA	
25	FORMULA TOO COMPLEX	
26	CAN'T CONTINUE	
27	UNDEFINED FUNCTION	
28	VERIFY	
29	LOAD	
30	BREAK	
31	CAN'T RESUME	
32	LOOP NOT FOUND	
33	LOOP WITHOUT DO	
34	DIRECT MODE ONLY	*Du har skrivit in en ett kommando i ett program som endast tillåts i direktläge.*
35	NO GRAPHICS AREA	*Vissa kommandon behöver ha en grafisk yta att agera på. Om du t.ex. försöker anropa kommandot CIRCLE innan du skapat en grafisk yta med kommandot GRAPHIC uppstår detta fel.*
36	BAD DISK	*Antingen har snabbformatering försökts att utföra på en diskett som inte är formaterad*

		sedan tidigare, eller så är disketten i fråga helt enkelt trasig.
37	`BEND NOT FOUND`	*Nyckelordet BEGIN kan användas i en IF-sats för att deklarera att ett kodblock ska exekvera under ett visst villkor. Detta block måste avslutas med BEND, annars uppstår detta fel.*
38	`LINE NUMBER TOO LARGE`	*Felet uppstår när kommandot RENUMBER körs, om körningen resulterar i att någon rad skulle få ett radnummer högre än 63999.*
39	`UNRESOLVED REFERENCE`	*Felet uppstår när kommandot RENUMBER körs, om det finns programrader som refererar till icke-existerande programrader (t.ex. en GOTO-sats som pekar på en icke-existerande rad).*
40	`UNIMPLEMENTED COMMAND`	*Det finns reserverade nyckelord som inte är implementerade kommandon (t.ex. QUIT eller OFF). Använder man dessa uppstår detta fel.*
41	`FILE READ`	*Något fel har uppstått under en läsning från en fil. Det kan t.ex. vara att diskettluckan har öppnats under pågående operation.*

Eftersom funktionen `ERR$` tar en felkod och ger felets namn som svar, så kommer `PRINT ERR$(41)` ge svaret `FILE READ`, medan `PRINT ERR$(ER)` ger det senast inträffade felet.

Om inget fel har uppstått ger `ER` värdet `-1`, vilket orsakar felet *illegal quantity* om det skickas som input till `ERR$`.

DS och DS$

Om ett DOS-relaterat fel uppstår lagras felinformation i `DS` och `DS$`, som förklaras på sidan 66 (kapitel 12).

APPENDIX B: ORDFÖRKLARINGAR

Appendix B: Ordförklaringar

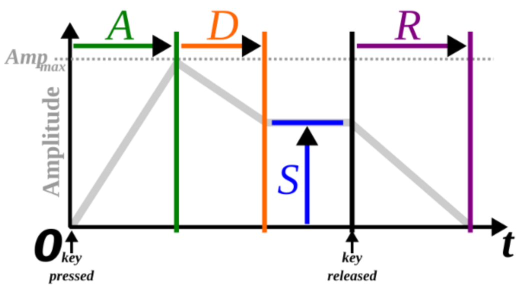
Adress

Den minsta allokeringsbara enheten i en 8-bitarsdator är en byte, varje byte har en *adress* (eller *minnesadress*). Vissa är läs- och skrivbara (RAM), andra är endast läsbara (ROM).

ADSR

ADSR är en engelsk förkortning för *attack, decay, sustain, release*, vilket på engelska kallas för *envelope*. Termen används för att beskriva hur ett musikinstrument beter sig i förhållande till ljudstyrka.

- *Attack* är tiden det tar från det att en ton spelas till det att den når sin fulla ljudstyrka. Ett högre värde innebär längre tid, ett lägre värde innebär kortare tid.
- *Decay* är tiden det tar för tonen att förlora sin ljudstyrka och landa på nivån som kallas *sustain* (upprätthållning).
- *Sustain* är ljudstyrkan som upprätthålls så länge tonen spelar.
- *Release* är den tid det tar för en ton som inte längre spelas att tysta.



Figur 13: ADSR - Attack och Decay handlar om tid, Sustain om nivå, och Release om tid.

Användardefinierad variabel

En användardefinierad variabel är en variabel skapad av datoranvändaren. Här placeras värdet `24` i variabeln `AX`:

```
AX=24
```

En användardefinierad variabel är inte en systemvariabel.

Argument
Data som skickas till (till exempel) en funktion för att påverka dess funktionalitet. Ett argument är samma sak som en *parameter*.

Array
En array är en samling av variabler som identifieras av ett index. I BASIC har arrayen själv ett namn som följer reglerna för variabelnamn, där varje element identifieras av ett 0-baserat index som anges inom parentes, till exempel `A(31)`. Ordet *vektor* kan användas synonymt med array.

Binär logik
Binär logik är logik som är tvåställig i betydelsen att ett uttryck antingen är sant eller falskt, precis som en bit.

Binära talsystemet
Det binära talsystemet använder endast två tecken för att beskriva ett tal, till skillnad från det decimala talsystemet som använder tio eller det hexadecimala som använder sexton. Det innebär att de fem första talen (noll till fyra) skrivs 0, 1, 10, 11 och 100.

Bit
En bit är ett tal mellan 0 och 1. Åtta bitar utgör en byte. Det binära talsystemet utgörs av bitar, från engelskans "binary digits".

Bitvis
En bitvis operator agerar på värdets bitmönster, bestående av `0` eller `1`. och kan därför användas i binär aritmetik. Som exempel är `2 OR 4` lika med `6`, eftersom `00000010 OR 00000100` är lika med `00000110`. De binära bitvisa operatorerna är `AND` och `OR`, den unära är `NOT`.

Bitmaskning
Med bitmaskning avses metoder att läsa av individuella bitar i en byte, eller skriva till en eller flera bitar utan att påverka andra bitar. Bitmaskning måste behärskas av den som vill kunna sätta eller läsa av flaggor, som t.ex. vilken sprite som ska vara synlig.

Blitter
Ett chip som hanterar blitter ansvarar för snabb manipulering av data i RAM, vilket möjliggör rörlig datorgrafik.

Booleskt värde

Ett booleskt värde är ett värde som antingen är sant eller falskt. I Commodore BASIC 2.0 second release representeras 0 som falskt och icke-0 som sant vid test. Som testresultat är 0 falskt och -1 sant. Vid bit-operationer är 0 falskt och 1 sant.

Border

Ramen runt den yta som VIC-20 och Commodore 64 kan visa text och högupplöst grafik på benämns som *border*. Borderfärgen är ramens färg, medan bakgrundsfärgen avser färgen på ytan med text och grafik.

Byte

En byte är den minsta enheten med en egen adress. En byte består av åtta bitar (eller två nibbles), vilket innebär att en byte kan befinna sig i ett av 256 olika tillstånd, till exempel ett tal mellan 0 och 255.

Call stack

Varje hopp som görs med `GOSUB` lagras i datorns *call stack*, som är ett slags lista över vilka rader som anropats med `GOSUB`. När kommandot `RETURN` påträffas, plockas ett hopp från datorns call stack, och exekveringen fortsätter på raden efter. Både VIC-20 och Commodore 64 har avsatt 256 bytes för sin call stack, och klarar att hålla programhopp i 23 led i minnet.

Datatyp

Din Commodore-dator kan endast lagra ettor och nollor, vilket representerar talet `0` eller talet `1`. Dessa kallas bitar och är alltid grupperade om 8. Åtta bitar utgör en byte, och antalet tal som kan beskrivas med åtta bitar (en byte) är 256 (0-255). De individuella bitarna saknar minnesadress, men varje byte (åtta bitar) har var sin minnesadress. Så snart man vill läsa något annat än individuella tal mellan 0 och 255 (åtta bitar) måste man veta hur olika bytes ska kombineras. En datatyp är en konfiguration av kombinationer av bytes. till exempel består talet `16961` av en kombination av två bytes, nämligen `65` och `66` (66 * 256 + 65 = 16961), medan textsträngen `AB` också består av samma kombination av bytes. Om man vet vilken datatyp man hanterar, kan man avkoda informationen korrekt.

DOS

DOS står för *Disk Operating System* och med DOS avser operativsystem kapabla att serva användaren i användandet av skivminnen som sekundärminne. Commodore-maskinerna erbjuder BASIC-tolken som användarens gränssnitt för att använda skivminnen (floppydisk), och den diskdrive som är inkopplad till enheten, erbjuder ytterligare kommandon som kan skickas till enheten för utförande. Exakt vilka kommandon som står till förfogande beror på vilken modell av diskdrive som är inkopplad till datorn. Några modeller som kan förekomma är 1540, 1541 som normalt såldes till VIC-20 och Commodore 64, 1541 II, den externa 1571 och den interna 1571 som normalt såldes till Commodore 128 respektive 128D. Denna bok tar avstamp i modell 1541.

Drivarnummer

Ordet drivarnummer kommer från *diskdrive-nummer*. Din primära diskettstation (diskdrive) har drivarnummer `0`. Om du har två diskettstationer inkopplade, har den andra drivarnummer `1`. Drivarnummer anges med prefixet `D`.

Enhetsnummer

Alla seriekopplade enheter (diskettstationer, printers, eller liknande) har ett enhetsnummer. De tillgängliga enhetsnumren är `8`, `9`, `10` och `11`. Normalt brukar den första diskdriven ligga på enhet `8`. Om du kopplar in en diskdrive till (eller en printer) behöver du konfigurera enhetsnumret, och hur det görs beskrivs i manualen till den utrustning du köpt. På diskettstationen Commodore 1571 finns en DIP-omkopplare på enhetens baksida.

Envelope

Engelsk term för ADSR (sida 85).

Floppydisk

En floppydisk är medium för ett skivminne. Ordet används synonymt med diskett. Enheten som läser disketter heter diskettstation, vilket används synonymt med (det engelska) ordet diskdrive.

Fysisk fil

En *fysisk fil* är, till skillnad från en *logisk fil*, en faktisk fil som finns lagrad på floppydisk eller på kassettband. En fysisk fil har ett namn och en startadress eller en (förhoppningsvis) specifik struktur.

Grafikläge

En Commodore-maskin har olika grafiklägen som anger vad som kan visas på skärmen. I textläget (som används vid uppstart) kan VIC-20 visa text i 22 rader med 23 kolumner eller 25 rader och 40 kolumner för Commodore 64. I bitmapsläge hanterar man i stället individuella pixlar i två olika upplösningar, där flerfärgsläget har hälften så många individuella pixlar på skärmen som är möjligt i enfärgsläget (eller det högupplösta läget).

- Commodore 64, enfärgsläget: 320×200 pixlar
- Commodore 64, flerfärgsläget: 160×200 pixlar
- VIC-20, enfärgsläget: 176×184 pixlar
- VIC-20, flerfärgsläget: 88×184 pixlar

Hard reset

En *hard reset* innebär att datorn återställs och att hela datorns minne (BASIC-program och övrig data) raderas. Jämför med *soft reset*.

Hexadecimala talsystemet

Det hexadecimala talsystemet använder hela sexton olika tecken för att beskriva ett tal, till skillnad från det decimala talsystemet som använder tio eller det binära talsystemet som använder två. Det innebär att de tjugo första talen (noll till nitton) skrivs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12 och 13.

Interrupten

Interrupten är en funktion för multitasking som finns inbyggd i VIC-20 och Commodore 64. Huvudprocessorn kan, med jämna mellanrum, släppa in andra uppgifter, vilket liknar det som idag kallas *time slicing*.

I/O

I/O är en förkortning av input/output och avser operationer som läser eller skriver från/till externa enheter som till exempel tangentbord, printer, skärm eller floppydisk.

Jiffy

Vad en *jiffy* (flera *jiffies*) är varierar från plattform till plattform, men i Commodore BASIC 2.0 second release är en jiffy en sextionsdels sekund. En jiffy är alltså 1/60 sekunder och 60 jiffies är en sekund. En mer samtida godtycklig tidsenhet är en *tick*, som ofta är kortare, till exempel en tusendels sekund.

Joystick

En joystick (styrspak) är en populär enhet för att styra spel. Commodore 128 (och Commodore 64) kan hantera två joysticks samtidigt, vilket innebär att spel där två spelare samarbetar eller tävlar är möjliga. Både i standardutförande och i D-modell återfinns styrportarna till höger.

Figur 14: Commodores Atari-kompatibla joystick (styrspak) för VIC-20, Commodore 64/128 och Amiga.

Kilobyte

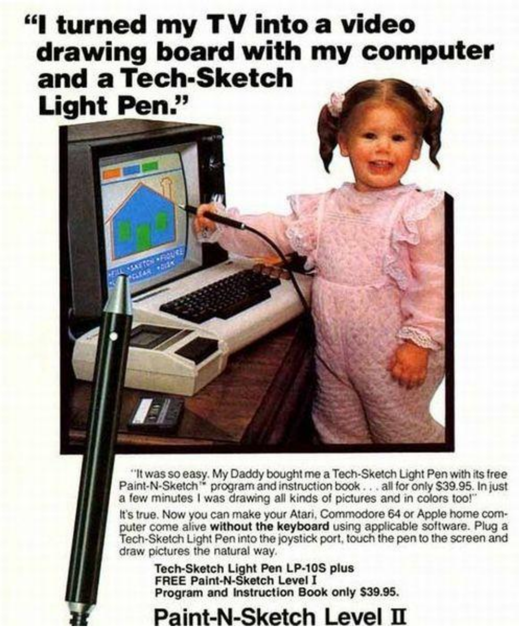
En *kilobyte* (kort *K*) är 1024 bytes. 16 K betyder således 16 kilobytes eller 16384 bytes.

Konsol

Konsolen (textkonsolen) är gränssnittet mellan användaren och datorn som användaren använder genom datorns tangentbord. Man känner igen konsolen på den blinkande markören.

Ljuspenna

En ljuspenna är en enhet man kopplar till datorn och håller mot skärmen för att ange en position. Med korrekt mjukvara kan man använda en ljuspenna för att rita, ungefär som man även med korrekt mjukvara kan använda musen till detsamma. Ljuspennan rapporterar till datorn när skärmens elektronstråle träffar den, och därmed kan datorn räkna ut var på skärmen ljuspennan pekar.



Figur 15: Reklam för ljuspennan Paint-N-Sketch Level II.

Loader

En *loader* är ett program som laddar in ett program i minnet. I Commodore-världen är detta typiska exempel:

- Ett dataspel som ska laddas från kassett. Första programmet har till uppgift att visa en pausbild på skärmen, medan det hämtar in huvudprogrammet från kassettbandet.
- Ett BASIC-program som skriver ett maskinkodsprogram till minnet, som visas i avsnittet om `DATA`.

Logisk fil

Normalt är en fil ett stycke namngivet data på floppydisk, men en *logisk fil* kan vara precis vad som helst som kan läsas från eller skrivas till *som om* det vore en fil på en floppydisk. Förutom olika typer av filer, kan en logisk fil vara en kanal till en skrivare.

Markör

Textmarkören anger var nästa tecken från tangentbordet kommer att hamna. Markören illustreras som en blinkande rektangel.

Multitasking

Multitasking innebär att två program körs samtidigt. Det kan till exempel vara ett maskinkodsprogram som körs samtidigt som ett BASIC-program. När man programmerar en Commodore-maskin pratar man om *interrupten*, som lite slarvigt är en svensk bestämd form av engelskans *interrupt*, vilket i sammanhanget närmast kan liknas med *att bryta in*. I enkelhet kan man tänka att datorn är utrustad timers som med jämna mellanrum kan exekvera din maskinkod. Det innebär att flera maskinkodsrutiner, och även någon BASIC-rutin, kan exekvera simultant. Detta sker på bekostnad av prestandan.

Operand

En *operand* är ett värde i en ekvation. I följande exempel är `1` och `2` operander, medan `+` är *operator*:

```
X = 1 + 2
```

Operator

En *operator* är symbol eller en funktion i en matematisk operation. I följande exempel är `+` operator medan `1` och `2` är *operander*:

```
X = 1 + 2
```

Paddel

xxxxxxxxx

PAL

PAL, Phase Alternate Line, är en standard för analog färg-tv som primärt används i Europa. I Amerika används NTSC och i Asien används SECAM.

Parameter

Data som skickas till (till exempel) en funktion för att påverka dess funktionalitet. En parameter är samma sak som ett *argument*.

Pekare
Pekare är inget som BASIC har stöd för, men man kommer ofta i kontakt med pekare när man använder datorns inbyggda funktioner. En minnesadress som syftar till att hålla reda på en minnesadress är en pekare. Olika pekare används på olika sätt. Ett par exempel som specifikt gäller Commodore 64: Adress `785-786` ett 16-bitarstal som är den exakta adressen för funktionen `USR`, medan på adress `2048` talar om var i minnet bilddata för den åttonde spriten på ligger, om man multiplicerar värdet med 64.

PETSCII
PETSCII (PET Standard Code of Information Interchange) är namnet på teckentabellen som används av bland andra VIC-20 och Commodore 64. PETSCII är Commodores version av ASCII, och teckentabellen är uppkallad efter den första datorn som använde den, Commodore PET (år 1977).

Pixel
Bilden som visas på skärmen består av punkter av olika färger, i en matris. Varje punkt (egentligen *bildelement*) som utgör bilden, kallas för *pixel*.

Primärminne
Primärminnet är datorns arbetsminne (datorns RAM-minne).

Pseudografik
Pseudografik är textbaserad grafik som syftar till att se ut som högupplöst rastergrafik. På din Commodore-dator finns massor av tecken framtagna för att kunna passa till pseudografik. Se kapitlet om text för mer information.

Radnummer
BASIC-program på Commodore 128 använder tal för att hålla reda på vilket kommando som ska utföras i vilken ordning, och framför allt, vilka kommandon som ingår i ett program, genom att numrera programsatserna.

RAM
RAM står för *random access memory* och ordet avser minne som man kan skriva till eller läsa av i valfri ordning – man kan ange värdets minnesadress. I boken menas *datorns primärminne* eller *arbetsminne*.

Ren funktion
En *ren funktion* agerar endast på inkommande data och ger ett svar utifrån det, utan att läsa från annat minne eller skriva till ett annat minne. Ett sådant exempel är `SQR` som ger roten av ett tal (beskrivs i boken *Commodore BASIC 2.0 second release*) eller funktionen `ERR$` som beskrivs appendix A om felsökning.

Reset
En *reset* återställer datorn, vilket innebär att BASIC-program går förlorat. Se *hard reset* och *soft reset*.

ROM
ROM står för *read only memory* (minne som endast kan läsas, inte skrivas) och avser datorminne som beskriver datorns inbyggda funktioner.

Sekundärminne
Sekundärminnet används, till skillnad från primärminnet, till långvarig lagring. Commodore använder antingen floppydisk (diskett) eller datasette (kassettband) som sekundärminne.

Sekventiell fil
Definitionsmässigt utgörs en sekventiell datafil innehåll av data i den ordningen den skrevs till filen. En konsekvens av detta är att filen måste läsas samma ordning som den skrevs. Detta är utmärkande för programfiler, textfiler och till exempel bildfiler.

Soft reset
En *soft reset* innebär att datorn återställs och att datorns BASIC-minne raderas. Jämför med *hard reset*.

Sprite
En sprite är en liten grafisk symbol som kan röra sig fritt på skärmen, utan att störa annan grafik. Commodore 64 kan visa åtta sprites men VIC-20 saknar sprites – vill man ha tillgång till sprites på VIC-20 så får man bygga en egen sprite-rutin. Se appendix E, som jämför VIC-20, Commodore 64 och Commodore 128 med varandra för mer information.

Systemvariabel
En systemvariabel är en variabel som får sitt värde från systemet snarare än från datoranvändaren. En systemvariabel är inte en *användardefinierad* variabel. Vissa systemvariabler kan initieras eller uppdateras av användaren, andra systemvariabler är helt skrivskyddade, men alla uppdateras av systemet.

Terminal
Terminalen är en del av den inbyggda programvaran i datorer som Commodore 128, Commodore 64, Sinclair ZX81, Sinclair Spectrum och VIC-20. Den samlar in användarens kommandon som ges via tangentbordet, skickar dem till programmet som ska utföra, och tar emot svaret för att presentera det för användaren.

Vektor
I BASIC används ordet *vektor* synonymt med ordet *array*. En array (eller en vektor) har ett antal element och varje element har ett värde.

Vektorgrafik
Med *vektorgrafik* avses bilder som beskrivs av en serie av primitiva figurer, typiskt linjer, ellipser och rektanglar. Commodore 128 erbjuder en rik uppsättning av kommandon för att skapa vektorgrafik.

APPENDIX C: EN JÄMFÖRELSE MELLAN COMMODORE 128, COMMODORE 64 OCH VIC-20

Appendix C: En jämförelse mellan Commodore 128, Commodore 64 och VIC-20

Tabellen nedan visar specifikationerna för VIC-20, Commodore 64 (C64) och Commodore 128 (C128).

	VIC-20	C64	C128
Lanseringsår	1980	1982	1985
Programmeringsspråk	Commodore BASIC 2.0 second release	Commodore BASIC 2.0 second release	Commodore BASIC 7.0
Längsta programsats	88 tecken (4 rader)	80 tecken (2 rader)	160 tecken (4 rader i 40-kolumnsläge, 2 rader i 80-kolumnsläge)
Operativsystem[19]	-	GEOS (tillval)	GEOS (tillval), CP/M 3.0
Huvudprocessor	MOS 6502	MOS 6510	MOS 8502, Z80B
Klockfrekvens[20]	1,1 MHz	0,99 MHz	1-2 MHz, 4 MHz
ROM	20 KB	20 KB	72 KB
RAM	5 KB	64 KB	128 KB
Expansionsmöjlighet RAM	32 KB[21]	320 KB	512 KB
Text	22×23	40×25	40×25, 80×25
Skärmupplösning	176×184 plus border	320×200 plus border	320×200 plus border, 640×200
Monokrom grafik	176×184	320×200	320×200
Flerfärgsgrafik	88×184	160×200	160×200
Videoutgång	Analog (A/V)	Analog (RF, A/V)	Analog (RF, A/V)/Digital (RGBI)
Ljud	Tre fyrkantsvågor och ett brusljud, tre kanaler	Konfigurerbar fyrkantsvåg, triangelvåg, sinusvåg, brus, filter, med mera. Tre kanaler	Konfigurerbar fyrkantsvåg, triangelvåg, sinusvåg, brus, filter, med mera. Tre kanaler
Maskinkodsmonitor	-	-	Ja
Sprites	-	8	8
Sprite-editor	-	-	Ja

[19] Utöver BASIC.

[20] Processorns hastighet är aningen olika för PAL-anpassade datorer och NTSC-anpassade datorer. För Commodore 128 gäller 4 MHz när Z80B-processorn används.

[21] Bryter kompatibiliteten.

APPENDIX D: MASKINKOD

Appendix D: Maskinkod

Denna bilaga kommer att ge en introduktion till assembler-programmering på Commodore 128.

APPENDIX E: ÖVRIGA FUNKTIONER I COMMODORE 128

Appendix E: Övriga funktioner i Commodore 128

Denna bilaga nämner övriga funktioner som din dator har, som kan vara bra att känna till.

Om du vill att datorn ska spela en ton, skriv `PRINT CHR$(7)`. Om du bygger ett program, kan en ton användas för att påkalla användarens uppmärksamhet. Du kan höra hur tonen låter genom att trycka **Control+G**.

Behöver du starta om din dator, så har du **Reset**-knappen på datorns högra sida. Vill du starta om datorn från ett program, använd `SYS 65341`. Oavsett hur du väljer att starta om datorn, kommer ditt BASIC-program att försvinna ur minnet.

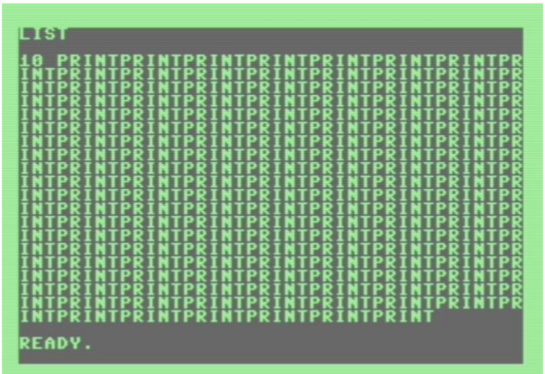
APPENDIX F: FÖRKORTNINGAR

Appendix F: Förkortningar

Genom att memorera förkortningar på de kommandon du använder ofta, kan du spara en del tid när du knackar in dina BASIC-program. T.ex. kan du skriva **RE** följt av **Shift+A** för att ange kommandot **READ**, eller bara ett frågetecken för att ange kommandot `PRINT`. Men det finns också ett annat användningsområde, som kan vara aktuellt. En programsats kan vara ungefär 160 tecken lång baserat på storleken på inmatningsbufferten[22]. Men genom att använda förkortningar, kan en programsats vara mycket längre när den väl är inmatad, eftersom tecknet **?** endast belastar inmatningsbufferten med ett tecken av 160 tillgängliga, medan det väl inmatat representerar ett kommando som konsumerar 5 tecken, nämligen `PRINT`. Detta är en giltig inmatning i Commodore 128 (radbryten för att illustrera skärmstorleken i 40-kolumnsläge):

```
10??????????????????????????????????????
????????????????????????????????????????
????????????????????????????????????????
????????????????????????????????????????
```

Om du tittar på resultatet genom att skriva `LIST`, får du en väldigt lång programsats.



Figur 16: En programrad som är mycket längre än 160 tecken.

[22] Inmatningsbufferten (eller input-bufferten) används av terminalen från det att man börjar skriva ett kommando, till dess att man trycker på Return. Då behandlas det inmatade och bufferten töms.

Här följer alla tillgängliga förkortningar i Commodore BASIC 7.0:

`ABS`: **A Shift+B**
`AND`: **A Shift+N**
`APPEND`: **A Shift+P**
`ASC`: **A Shift+S**
`ATN`: **A Shift+T**
`AUTO`: **A Shift+U**
`BACKUP`: **BA Shift+C**
`BANK`: **B Shift+A**
`BEGIN`: **B Shift+E**
`BEND`: **BE Shift+N**
`BLOAD`: **B Shift+L**
`BOOT`: **B Shift+O**
`BSAVE`: **B Shift+S**
`BUMP`: **B Shift+U**
`CATALOG`: **C Shift+A**
`CHAR`: **CH Shift+A**
`CHR$`: **C Shift+H**
`CIRCLE`: **C Shift+I**
`CLOSE`: **CL Shift+O**
`CLR`: **C Shift+L**
`CMD`: **C Shift+M**
`COLLECT`: **COLL Shift+E**
`COLLISION`: **COL Shift+L**
`COLOR`: **COL Shift+O**
`CONCAT`: **C Shift+O**
`COPY`: **CO Shift+P**
`DATA`: **D Shift+A**
`DCLEAR`: **DCL Shift+E**
`DCLOSE`: **D Shift+C**
`DELETE`: **DE Shift+L**
`DIM`: **D Shift+I**
`DIRECTORY`: **DI Shift+R**
`DLOAD`: **D Shift+L**
`DOPEN`: **D Shift+O**
`DRAW`: **D Shift+R**
`DSAVE`: **D Shift+S**
`DVERIFY`: **D Shift+V**
`ELSE`: **E Shift+L**
`ENVELOPE`: **E Shift+N**
`ERR$`: **E Shift+R**
`MONITOR`: **MO Shift+N**
`NEXT`: **N Shift+E**
`NOT`: **N Shift+O**
`OPEN`: **O Shift+P**
`PAINT`: **P Shift+A**
`PEEK`: **PE Shift+E**
`PEN`: **P Shift+E**
`PLAY`: **P Shift+L**
`POINTER`: **PO Shift+I**
`POKE`: **PO Shift+K**
`POT`: **P Shift+O**
`PRINT`: **?**
`PRINT#`: **P Shift+R**
`PRINT USING`: **?US Shift+I**
`PUDEF`: **P Shift+U**
`RCLR`: **R Shift+C**
`RDOT`: **R Shift+D**
`READ`: **RE Shift+A**
`RECORD`: **R Shift+E**
`RENAME`: **RE Shift+N**
`RENUMBER`: **REN Shift+U**
`RESTORE`: **RE Shift+S**
`RESUME`: **RES Shift+U**
`RETURN`: **RE Shift+T**
`RGR`: **R Shift+G**
`RIGHT$`: **R Shift+I**
`RND`: **R Shift+N**
`RREG`: **R Shift+R**
`RSPCOLOR`: **RSP Shift+C**
`RSPPOS`: **R Shift+S**
`RSPRITE`: **RSP Shift+R**
`RUN`: **R Shift+U**
`RWINDOW`: **R Shift+W**
`SAVE`: **S Shift+A**
`SCALE`: **SC Shift+A**
`SCNCLR`: **S Shift+C**
`SCRATCH`: **SC Shift+R**
`SGN`: **S Shift+G**
`SIN`: **S Shift+I**
`SLEEP`: **S Shift+L**

`EXIT`: **EX Shift+I**
`EXP`: **E Shift+X**
`FETCH`: **F Shift+E**
`FILTER`: **F Shift+I**
`FOR`: **F Shift+O**
`FRE`: **F Shift+R**
`GET`: **G Shift+E**
`GETKEY`: **GETK Shift+E**
`GOSUB`: **GO Shift+S**
`GOTO`: **G Shift+O**
`GRAPHIC`: **G Shift+R**
`GSHAPE`: **G Shift+S**
`HEADER`: **HE Shift+A**
`HELP`: **HE Shift+L**
`HEX$`: **H Shift+E**
`INPUT#`: **I Shift+N**
`INSTR`: **IN Shift+S**
`JOY`: **J Shift+O**
`KEY`: **K Shift+E**
`LEFT$`: **LE Shift+F**
`LET`: **L Shift+E**
`LIST`: **L Shift+I**
`LOAD`: **L Shift+O**
`LOCATE`: **LO Shift+C**
`LOOP`: **LO Shift+O**
`MID$`: **M Shift+I**
`SOUND`: **S Shift+O**
`SPRCOLOR`: **SPR Shift+C**
`SPRDEF`: **SPR Shift+D**
`SPRITE`: **S Shift+P**
`SPRSAV`: **SPR Shift+S**
`SQR`: **S Shift+Q**
`SSHAPE`: **S Shift+S**
`STASH`: **S Shift+T**
`STEP`: **ST Shift+E**
`STOP`: **ST Shift+O**
`STR$`: **ST Shift+R**
`SWAP`: **S Shift+W**
`TAB(`: **T Shift+A**
`TEMPO`: **T Shift+E**
`THEN`: **T Shift+H**
`TRAP`: **T Shift+R**
`TROFF`: **TRO Shift+F**
`TRON`: **TR Shift+O**
`USR`: **U Shift+S**
`VERIFY`: **V Shift+E**
`VOL`: **V Shift+O**
`WAIT`: **W Shift+A**
`WHILE`: **WA Shift+I**
`WINDOW`: **W Shift+I**
`XOR`: **X Shift+O**

INDEX

Index

BILDER

Bilder

Erkännanden

Omslagsbilden föreställande en Commodore 128 är fotograferad av Evan Amos (CC BY-SA 3.0). Bilden används även för att visa datorns tangentbordslayout på sida 4. Martin Sjöblom har korrekturläst kapitlet om Commodore 128. Pekka Takala och Dan Henderson har bidragit till kapitlet om CP/M. Bilden föreställande boot-disketten på sida 74 är tagen av Alex Lozupone.

Buggar

Om du startar din Commodore 128 i dess normala BASIC-läge och skriver `PRINT""+-1` kan din dator reagera märkligt. Hamnar du i maskinkodsmonitorn, skriv `X` för att avsluta. Upprepar du detta kommando totalt fyra gånger så har datorn låst sig helt.

Figur 17: Om detta uppstår befinner du dig i maskinkodsmonitorn. Skriv X för att återgå till BASIC så att du kan skriva kommandot igen.

Böcker i denna serie

Commodore BASIC 2.0 second release (2021)
https://ahesselbom.se/pages/commodorebasic20.html

Commodore BASIC 7.0 för Commodore 128 (2025)
https://ahesselbom.se/pages/commodorebasic70.html